Formal verification of unlinkability for stateful protocols
Bridging the gap between symbolic and computational models

Solène MOREAU — PhD defense — November 18, 2021

PhD supervisors: Stéphanie DELAUNE and David BAEELDE
Computer systems and networks are everywhere.

Very often, communications involve sensitive data and occur on unreliable networks.

► Security is needed! Authentication, secrecy, unlinkability, ...
Computer systems and networks are everywhere.

Very often, communications involve **sensitive data** and occur on **unreliable networks**.

▶ **Security** is needed! Authentication, secrecy, unlinkability, . . .
Security protocols

**Definition**

A *security protocol* is a set of rules specifying:
- how agents exchange information through a communication channel,
- while ensuring security properties.

- **Roles** of each agent.
- **Messages**, using cryptographic primitives (hash, encryption, ...).
- **Evolution of states**.
### Definition

A **security protocol** is a set of rules specifying:
- how agents exchange information through a communication channel,
- while ensuring security properties.

- **Roles** of each agent.
- **Messages**, using **cryptographic primitives** (hash, encryption, ...).
- Evolution of **states**.
How to build secure protocols?

Several levels of attacks, which may exploit:

- weaknesses of cryptographic primitives;
- flaws in the design of the protocol;
- bugs in implementations.

Exploring all possible executions is not possible!

▶ A successful approach so far is the use of formal proofs.

Example

How to prove that $\forall n \in \mathbb{N}, P(n)$?

- Prove $P(0), P(1), P(2), \ldots, P(1811), \ldots$
- Prove $P(0)$ and $\forall k \geq 0, P(k) \Rightarrow P(k + 1)$ (proof by induction).
How to build secure protocols?

Several levels of attacks, which may exploit:

- weaknesses of cryptographic primitives;
- **flaws in the design of the protocol**;
- bugs in implementations.

Exploring all possible executions is not possible!

▶ A successful approach so far is the use of **formal proofs**.

**Example**

How to prove that \( \forall n \in \mathbb{N}, \ P(n) \)?

- Prove \( P(0), P(1), P(2), \ldots, P(1811), \ldots \)
- Prove \( P(0) \) and \( \forall k \geq 0, \ P(k) \Rightarrow P(k + 1) \) (proof by induction).
How to build secure protocols?

Several levels of attacks, which may exploit:

- weaknesses of cryptographic primitives;
- flaws in the design of the protocol;
- bugs in implementations.

Exploring all possible executions is not possible!

▶ A successful approach so far is the use of formal proofs.

Example

How to prove that $\forall n \in \mathbb{N}, P(n)$?

- Prove $P(0), P(1), P(2), \ldots, P(1811), \ldots$
- Prove $P(0)$ and $\forall k \geq 0, P(k) \Rightarrow P(k + 1)$ (proof by induction).
How to build secure protocols?

Several levels of attacks, which may exploit:

- weaknesses of cryptographic primitives;
- flaws in the design of the protocol;
- bugs in implementations.

Exploring all possible executions is not possible!

▶ A successful approach so far is the use of **formal proofs**.

**Example**

How to prove that \( \forall n \in \mathbb{N}, \ P(n) \)?

- Prove \( P(0), \ P(1), \ P(2), \ldots, \ P(1811), \ldots \)
- Prove \( P(0) \) and \( \forall k \geq 0, \ P(k) \Rightarrow P(k + 1) \) (proof by induction).
Formal verification

Protocol specification
- Roles
- Messages
- States

\[ \mathcal{P} \overset{\text{def}}{=} \text{in}(x). \]
\[ \quad \text{if } x = \ldots \]
\[ \quad \text{then out(ok)} \]
\[ \quad \text{else out(error)} \]

Security goals
- Reachability properties (weak secrecy, authentication...)
- Equivalence properties (unlinkability, anonymity...)

\[ \mathcal{P} \models \phi \]
Formal verification

Attacker model

Protocol specification
- Roles
- Messages
- States

\[ \mathcal{P} \overset{\text{def}}{=} \text{in}(x).
\]
\[ \text{if } x = \ldots \]
\[ \text{then out(ok)} \]
\[ \text{else out(error)} \]

Security goals
- Reachability properties
  (weak secrecy, authentication...)
- Equivalence properties
  (unlinkability, anonymity...)

\[ \mathcal{P} \models \phi \]

\[ \phi \]
Formal verification

Protocol specification
- Roles
- Messages
- States

\[ P \overset{\text{def}}{=} \text{in}(x). \]
\[ \text{if } x = \ldots \]
\[ \text{then out(ok)} \]
\[ \text{else out(error)} \]

Security goals
- Reachability properties
  (weak secrecy, authentication...)
- Equivalence properties
  (unlinkability, anonymity...)}
Formal verification

Attacker model

Protocol specification
- Roles
- Messages
- States

\[ \mathcal{P} \overset{\text{def}}{=} \text{in}(x). \]
if \( x = \ldots \)
then \( \text{out(ok)} \)
else \( \text{out(error)} \)

Security goals
- Reachability properties
  (weak secrecy, authentication...)
- Equivalence properties
  (unlinkability, anonymity...)

\( \mathcal{P} \models \phi \)
Formal verification

Attacker model

Protocol specification
- Roles
- Messages
- States

\[ \mathcal{P} \overset{\text{def}}{=} \text{in}(x). \]
- if \( x = \ldots \)
  - then out(ok)
  - else out(error)

Security goals
- Reachability properties
  (weak secrecy, authentication...)
- Equivalence properties
  (unlinkability, anonymity...)

\[ \mathcal{P} \models \phi \]
Formal verification

Attacker model

Protocol specification
- Roles
- Messages
- States

\[ \mathcal{P} \overset{\text{def}}{=} \text{in}(x). \]
if \( x = \ldots \)
then \text{out}(ok)
else \text{out}(error)\]

Security goals
- Reachability properties
  (weak secrecy, authentication...)
- Equivalence properties
  (unlinkability, anonymity...)

\[ \mathcal{P} \models \phi \]
## How to model the attacker?

<table>
<thead>
<tr>
<th>Symbolic models [DY83]</th>
<th>Computational models [GM84]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages are <strong>terms</strong>.</td>
<td>Messages are <strong>bitstrings</strong>.</td>
</tr>
<tr>
<td>What the attacker <strong>can do</strong>.</td>
<td>What the attacker <strong>cannot do</strong>. Everything else is allowed!¹</td>
</tr>
<tr>
<td>$n \approx m$ with $n, m$ atomic</td>
<td>$n \approx m$ with $n, m \in {0, 1}^\eta$</td>
</tr>
<tr>
<td>▶ <strong>Weaker guarantees.</strong></td>
<td>▶ <strong>Stronger guarantees.</strong></td>
</tr>
<tr>
<td>▶ <strong>Amenable to automation.</strong></td>
<td>▶ <strong>Harder to automate.</strong></td>
</tr>
</tbody>
</table>

The **CCSA model** [BC12; BC14] tries to keep the best of both worlds.

---

¹But the attacker is actually a probabilistic polynomial-time Turing machine.
## How to model the attacker?

<table>
<thead>
<tr>
<th>Symbolic models [DY83]</th>
<th>Computational models [GM84]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages are <strong>terms</strong>.</td>
<td>Messages are <strong>bitstrings</strong>.</td>
</tr>
<tr>
<td>What the attacker <strong>can do</strong>.</td>
<td>What the attacker <strong>cannot do</strong>. Everything else is allowed!(^1)</td>
</tr>
<tr>
<td>( n \approx m ) with ( n, m ) atomic</td>
<td>( n \approx m ) with ( n, m \in {0, 1}^\eta )</td>
</tr>
<tr>
<td>➤ Weaker guarantees.</td>
<td>➤ Stronger guarantees.</td>
</tr>
<tr>
<td>➤ Amenable to automation.</td>
<td>➤ Harder to automate.</td>
</tr>
</tbody>
</table>

The **CCSA model** [BC12; BC14] tries to keep the **best** of both worlds.

\(^1\)But the attacker is actually a probabilistic polynomial-time Turing machine.
How to model the attacker?

Symbolic models [DY83]  
Messages are terms.
What the attacker can do.

Computational models [GM84]  
Messages are bitstrings.
What the attacker cannot do.
Everything else is allowed!\(^1\)

\[ n \approx m \quad \text{with } n, m \text{ atomic} \]

- Weaker guarantees.
- Amenable to automation.

\[ n \approx m \quad \text{with } n, m \in \{0, 1\}^\eta \]

- Stronger guarantees.
- Harder to automate.

The **CCSA model** [BC12; BC14] tries to keep the best of both worlds.

---

\(^1\)But the attacker is actually a probabilistic polynomial-time Turing machine.
# How to model the attacker?

<table>
<thead>
<tr>
<th>Symbolic models [DY83]</th>
<th>Computational models [GM84]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages are <strong>terms</strong>.</td>
<td>Messages are <strong>bitstrings</strong>.</td>
</tr>
<tr>
<td>What the attacker <strong>can do</strong>.</td>
<td>What the attacker <strong>cannot do</strong>. Everything else is allowed!</td>
</tr>
<tr>
<td>( n \approx m ) with ( n, m ) atomic</td>
<td>( n \approx m ) with ( n, m \in {0, 1}^\eta )</td>
</tr>
</tbody>
</table>

- ► **Weaker guarantees.**
- ► **Amenable to automation.**
- ► **Stronger guarantees.**
- ► **Harder to automate.**

The **CCSA model** [BC12; BC14] tries to keep the **best** of both worlds.

---

1But the attacker is actually a probabilistic polynomial-time Turing machine.
## How to model the attacker?

<table>
<thead>
<tr>
<th>Symbolic models [DY83]</th>
<th>Computational models [GM84]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages are <strong>terms</strong>.</td>
<td>Messages are <strong>bitstrings</strong>.</td>
</tr>
<tr>
<td>What the attacker <strong>can do</strong>.</td>
<td>What the attacker <strong>cannot do</strong>. Everything else is allowed!(^1)</td>
</tr>
</tbody>
</table>

\[ n \approx m \quad \text{with } n, m \text{ atomic} \]

- [ ] Weaker guarantees.
- [ ] Amenable to automation.
- [ ] Stronger guarantees.
- [ ] Harder to automate.

The **CCSA model** [BC12; BC14] tries to keep the **best** of both worlds.

---

\(^1\)But the attacker is actually a probabilistic polynomial-time Turing machine.
## How to model the attacker?

<table>
<thead>
<tr>
<th>Symbolic models [DY83]</th>
<th>Computational models [GM84]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages are <strong>terms</strong>.</td>
<td>Messages are <strong>bitstrings</strong>.</td>
</tr>
<tr>
<td>What the attacker <strong>can do</strong>.</td>
<td></td>
</tr>
<tr>
<td>$n \approx m$ with $n, m$ atomic</td>
<td></td>
</tr>
<tr>
<td>$n \approx m$ with $n, m \in {0, 1}^\eta$</td>
<td></td>
</tr>
<tr>
<td>▶ <strong>Weaker guarantees.</strong></td>
<td>▶ <strong>Stronger guarantees.</strong></td>
</tr>
<tr>
<td>▶ <strong>Amenable to automation.</strong></td>
<td>▶ <strong>Harder to automate.</strong></td>
</tr>
</tbody>
</table>

The CCSA model [BC12; BC14] tries to keep the **best** of both worlds.

---

1But the attacker is actually a probabilistic polynomial-time Turing machine.
## How to model the attacker?

<table>
<thead>
<tr>
<th>Symbolic models [DY83]</th>
<th>Computational models [GM84]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages are <strong>terms</strong>.</td>
<td>Messages are <strong>bitstrings</strong>.</td>
</tr>
<tr>
<td>What the attacker <strong>can do</strong>.</td>
<td>What the attacker <strong>cannot do</strong>.</td>
</tr>
<tr>
<td>$n \approx m$ with $n, m$ atomic</td>
<td>$n \approx m$ with $n, m \in {0, 1}^\eta$</td>
</tr>
<tr>
<td>► <strong>Weaker guarantees.</strong></td>
<td>► <strong>Stronger guarantees.</strong></td>
</tr>
<tr>
<td>► <strong>Amenable to automation.</strong></td>
<td>► <strong>Harder to automate.</strong></td>
</tr>
</tbody>
</table>

The **CCSA model** [BC12; BC14] tries to keep the **best** of both worlds.

---

$^1$But the attacker is actually a probabilistic polynomial-time Turing machine.
How to model the attacker?

Symbolic models [DY83]

Messages are terms.

What the attacker can do.

$n \approx m$ with $n, m$ atomic

- Weaker guarantees.
- Amenable to automation.

Computational models [GM84]

Messages are bitstrings.

What the attacker cannot do.

Everything else is allowed!\(^1\)

$n \approx m$ with $n, m \in \{0, 1\}^\eta$

- Stronger guarantees.
- Harder to automate.

The CCSA model [BC12; BC14] tries to keep the best of both worlds.

\(^{\text{1But the attacker is actually a probabilistic polynomial-time Turing machine.}}\)
How to precisely model unlinkability for stateful two-party protocols and verify it using mechanized methods?
How to precisely model **unlinkability** for stateful two-party protocols and verify it using mechanized methods?
How to precisely model unlinkability for stateful two-party protocols and verify it using mechanized methods?
Research problem

How to precisely model unlinkability for stateful two-party protocols and verify it using mechanized methods?
Research problem

How to precisely model unlinkability for stateful two-party protocols and verify it using mechanized methods?
Contributions

Precise **model** for unlinkability of stateful two-party protocols.

Theoretical **verification** method in the **symbolic** model, based on sufficient conditions.

Validated by **case studies** on existing RFID protocols, using the **tool** Tamarin.

Theoretical **verification** framework providing **computational** guarantees, by extending the CCSA approach.

**Implemented** in a new interactive prover, Squirrel. Validated by **case studies**.
Contributions

Precise **model** for unlinkability of stateful two-party protocols.

Theoretical **verification** method in the **symbolic** model, based on sufficient conditions. Validated by **case studies** on existing RFID protocols, using the **tool** **Tamarin**.

Theoretical **verification** framework providing **computational** guarantees, by extending the CCSA approach. **Implemented** in a new interactive prover, **SQUIRREL**. Validated by **case studies**.
Contributions

Precise model for unlinkability of stateful two-party protocols.

Theoretical verification method in the symbolic model, based on sufficient conditions.

Validated by case studies on existing RFID protocols, using the tool Tamarin.

Theoretical verification framework providing computational guarantees, by extending the CCSA approach.

Implemented in a new interactive prover, SQUIRREL. Validated by case studies.
Contributions

Precise **model** for unlinkability of stateful two-party protocols.

Theoretical **verification** method in the **symbolic** model, based on sufficient conditions.

Validated by **case studies** on existing RFID protocols, using the **tool Tamarin**.

Theoretical **verification** framework providing **computational** guarantees, by extending the **CCSA** approach.

**Implemented** in a new interactive prover, **Squirrel**. Validated by **case studies**.
How to model unlinkability for stateful protocols?
Unlinkability

- [ISO 15408-209] "Ensuring that a user may make multiple uses of a service or resource without others being able to link these uses together."
- The real system is indistinguishable from an ideal system.
Unlinkability

- [ISO 15408-209] "Ensuring that a user may make multiple uses of a service or resource without others being able to link these uses together."
- The real system is indistinguishable from an ideal system.
Stateful two-party protocols

- Two types of roles: tags and readers.
- A state is some data shared from one session to the next:
  - a memory cell associated to each tag;
  - a global database shared by all readers.

- This global state may remain unchanged, or may be updated e.g. to avoid replay attacks, to achieve forward privacy...

- Updating a state can introduce observables!
Stateful two-party protocols

- Two types of roles: tags and readers.
- A state is some data shared from one session to the next:
  - a memory cell associated to each tag;
  - a global database shared by all readers.

- This global state may remain unchanged, or may be updated e.g. to avoid replay attacks, to achieve forward privacy...

- Updating a state can introduce observables!
Stateful two-party protocols

- Two types of roles: **tags** and **readers**.

- A **state** is some data shared from one session to the next:
  - a **memory cell** associated to each tag;
  - a global **database** shared by all readers.

- This global state may remain **unchanged**, or may be **updated**
  e.g. to avoid replay attacks, to achieve forward privacy...

- Updating a state can introduce **observables**!
Some related work

[BCH10]

- Unlinkability for simple protocols (single-step).
  Only tags are modelled.

[Ara+10]

- Weak unlinkability, for $n$-party protocols.
  Weaker than the [ISO 15408-209] definition.
- Strong unlinkability, for $n$-party protocols.
  Too strong notion of equivalence.

[HBD16]

- Unlinkability for stateless two-party protocols.
  False attacks if generic readers.
Some related work

[BCH10]
- Unlinkability for simple protocols (single-step).
  Only tags are modelled.

[Ara+10]
- Weak unlinkability, for $n$-party protocols.
  Weaker than the [ISO 15408-209] definition.
- Strong unlinkability, for $n$-party protocols.
  Too strong notion of equivalence.

[HBD16]
- Unlinkability for stateless two-party protocols.
  False attacks if generic readers.
Some related work

[BCH10]

- Unlinkability for simple protocols (single-step).
  Only tags are modelled.

[Ara+10]

- Weak unlinkability, for $n$-party protocols.
  Weaker than the [ISO 15408-209] definition.

- Strong unlinkability, for $n$-party protocols.
  Too strong notion of equivalence.

[HBD16]

- Unlinkability for stateless two-party protocols.
  False attacks if generic readers.
Some related work

[BCH10]
- Unlinkability for simple protocols (single-step).
  Only tags are modelled.

[Ara+10]
- Weak unlinkability, for $n$-party protocols.
  Weaker than the [ISO 15408-209] definition.
- Strong unlinkability, for $n$-party protocols.
  Too strong notion of equivalence.

[HBD16]
- Unlinkability for stateless two-party protocols.
  False attacks if generic readers.
Some related work

[BCH10]
- Unlinkability for simple protocols (single-step).
  Only tags are modelled.

[Ara+10]
- Weak unlinkability, for $n$-party protocols.
  Weaker than the [ISO 15408-209] definition.
- Strong unlinkability, for $n$-party protocols.
  Too strong notion of equivalence.

[HBD16]
- Unlinkability for stateless two-party protocols.
  False attacks if generic readers.
Basic Hash protocol

Figure 1: Description of the Basic Hash protocol [BCH10]
Basic Hash protocol

Figure 2: With specific readers as in [HBD16], unlinkability attack
Basic Hash protocol

Figure 3: With generic readers, no unlinkability attack
A definition of unlinkability

Definition

A protocol Π ensures **unlinkability** if \( M_\Pi \approx S_\Pi \).

\[
M_\Pi \overset{\text{def}}{=} (\neg \text{new } \overline{k}.\text{init}(\text{DB, cell}). \text{i new } \overline{n}_T.T) \mid (\neg \text{new } \overline{n}_R.R) \\
S_\Pi \overset{\text{def}}{=} (\neg \text{new } \overline{k}.\text{init}(\text{DB, cell}). \text{new } \overline{n}_T.T) \mid (\neg \text{new } \overline{n}_R.R)
\]
A definition of unlinkability

Definition

A protocol $\Pi$ ensures **unlinkability** if $M_\Pi \approx S_\Pi$.

\[
M_\Pi \overset{\text{def}}{=} (! \text{new } k. \text{init}(DB, \text{cell}). \text{i new } n_T.\mathcal{T}) \mid (! \text{new } n_R.\mathcal{R})
\]

\[
S_\Pi \overset{\text{def}}{=} (! \text{new } k. \text{init}(DB, \text{cell}). \text{new } n_T.\mathcal{T}) \mid (! \text{new } n_R.\mathcal{R})
\]
A definition of unlinkability

A protocol $\Pi$ ensures **unlinkability** if $M_\Pi \approx S_\Pi$.

\[
M_\Pi \overset{\text{def}}{=} (\! \text{new } \overline{k}.\text{init(DB, cell)}. \text{i new } \overline{n}_T.\mathcal{T}) \parallel (\! \text{new } \overline{n}_R.\mathcal{R})
\]

\[
S_\Pi \overset{\text{def}}{=} (\! \text{new } \overline{k}.\text{init(DB, cell)}. \text{new } \overline{n}_T.\mathcal{T}) \parallel (\! \text{new } \overline{n}_R.\mathcal{R})
\]
A definition of unlinkability

A protocol $\Pi$ ensures **unlinkability** if $\mathcal{M}_\Pi \approx S_\Pi$.

$$
\mathcal{M}_\Pi \stackrel{\text{def}}{=} (\!\! \text{new } \bar{k}.\text{init(DB, cell)}. \ i \ \text{new } \bar{n}_T.\mathcal{T}) \ | \ (\!\! \text{new } \bar{n}_R.\mathcal{R})
$$

$$
S_\Pi \stackrel{\text{def}}{=} (\!\! \text{new } \bar{k}.\text{init(DB, cell)}. \ \text{new } \bar{n}_T.\mathcal{T}) \ | \ (\!\! \text{new } \bar{n}_R.\mathcal{R})
$$
A definition of unlinkability

A protocol $\Pi$ ensures **unlinkability** if $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$.

\[
\mathcal{M}_\Pi \overset{\text{def}}{=} (\text{! new } \overline{k}.\text{init}(DB, \text{cell}). \text{! new } \overline{n}_T.\mathcal{T}) \mid (\text{! new } \overline{n}_R.\mathcal{R})
\]

\[
\mathcal{S}_\Pi \overset{\text{def}}{=} (\text{! new } \overline{k}.\text{init}(DB, \text{cell}). \text{ new } \overline{n}_T.\mathcal{T}) \mid (\text{! new } \overline{n}_R.\mathcal{R})
\]
A definition of unlinkability

A protocol $\Pi$ ensures **unlinkability** if $M_\Pi \approx S_\Pi$.

\[
M_\Pi \overset{\text{def}}{=} (\text{new } \overline{k}.\text{init}(\text{DB, cell}). \text{i new } \overline{n}_T.\mathcal{T}) \mid (\text{new } \overline{n}_R.\mathcal{R})
\]

\[
S_\Pi \overset{\text{def}}{=} (\text{new } \overline{k}.\text{init}(\text{DB, cell}). \text{new} \overline{n}_T.\mathcal{T}) \mid (\text{new } \overline{n}_R.\mathcal{R})
\]
A definition of unlinkability

Definition

A protocol Π ensures **unlinkability** if $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$.

\[
\mathcal{M}_\Pi \overset{\text{def}}{=} (! \text{ new } \bar{k}.\text{init}(\text{DB, cell}). \text{ i new } \bar{n}_T . T) \mid (! \text{ new } \bar{n}_R . R)
\]

\[
\mathcal{S}_\Pi \overset{\text{def}}{=} (! \text{ new } \bar{k}.\text{init}(\text{DB, cell}). \text{ new } \bar{n}_T . T) \mid (! \text{ new } \bar{n}_R . R)
\]
A definition of unlinkability

A protocol \( \Pi \) ensures **unlinkability** if \( M_\Pi \approx S_\Pi \).

\[
M_\Pi \overset{\text{def}}{=} (\!\! \text{new } \overline{k}.\text{init}(\text{DB, cell}). \text{ i new } \overline{n}_T.T) \mid (\!\! \text{new } \overline{n}_R.R)
\]

\[
S_\Pi \overset{\text{def}}{=} (\!\! \text{new } \overline{k}.\text{init}(\text{DB, cell}). \text{ new } \overline{n}_T.T) \mid (\!\! \text{new } \overline{n}_R.R)
\]
A definition of unlinkability

Definition

A protocol $\Pi$ ensures **unlinkability** if $M_\Pi \approx S_\Pi$.

\[
M_\Pi \overset{\text{def}}{=} (\! \text{new } \overline{k}.\text{init}(\text{DB, cell}) \cdot \! \text{new } \overline{n}_T.T) \mid (\! \text{new } \overline{n}_R.R)
\]

\[
S_\Pi \overset{\text{def}}{=} (\! \text{new } \overline{k}.\text{init}(\text{DB, cell}) \cdot \! \text{new } \overline{n}_T.T) \mid (\! \text{new } \overline{n}_R.R)
\]

sequential replication
A definition of unlinkability

A protocol $\Pi$ ensures **unlinkability** if $M_\Pi \approx S_\Pi$.

\[
M_\Pi \overset{\text{def}}{=} (\text{! new } k.\text{init(DB, cell)}). \text{i new } \overline{n}_T.T) \mid (\text{! new } \overline{n}_R.R)
\]

\[
S_\Pi \overset{\text{def}}{=} (\text{! new } k.\text{init(DB, cell)}). \text{new } \overline{n}_T.T) \mid (\text{! new } \overline{n}_R.R)
\]
How to verify unlinkability for stateful protocols?
A verification method in the symbolic model

Extending [HBD16] to stateful protocols

Theorem

If a protocol \( \Pi \) ensures well-authentication, frame opacity and no desynchronization then \( \Pi \) ensures unlinkability.

<table>
<thead>
<tr>
<th>Protocols</th>
<th>unlink.</th>
<th>WA</th>
<th>FO</th>
<th>ND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Hash</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hash-Lock</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feldhofer</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OSK (v1)</td>
<td>attack</td>
<td>✓</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>OSK (v2)</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LAK (pairs)</td>
<td>attack</td>
<td>✓</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>LAK (pairs, fixed)</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LAK (pairs, no update)</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5G-AKA (simplified)</td>
<td>ok</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

 ✓  = property holds
 ×  = property does not hold
A verification method in the symbolic model

Extending [HBD16] to stateful protocols

Theorem

If a protocol $\Pi$ ensures well-authentication, frame opacity and no desynchronization then $\Pi$ ensures unlinkability.

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

✓ = property holds
× = property does not hold
Can we do better?

Theoretical and practical limitations of the previous method

- Only guarantees against a (weak) symbolic attacker.
- Finely tuned models and intermediate lemmas.
- Limitations of the tools (e.g. XOR, inductive reasoning).

State of the art in the computational model

- Some tools (CryptoVerif, EasyCrypt, CryptHOL, F*).
- CryptoVerif is designed to naturally model protocols, but does not support stateful protocols.

What about the CCSA approach?
Can we do better?

Theoretical and practical limitations of the previous method

- Only guarantees against a (weak) symbolic attacker.
- Finely tuned models and intermediate lemmas.
- Limitations of the tools (e.g. XOR, inductive reasoning).

State of the art in the computational model

- Some tools (CryptoVerif, EasyCrypt, CryptHOL, F*).
- CryptoVerif is designed to naturally model protocols, but does not support stateful protocols.

What about the CCSA approach?
Can we do better?

Theoretical and practical limitations of the previous method

- Only guarantees against a (weak) symbolic attacker.
- Finely tuned models and intermediate lemmas.
- Limitations of the tools (e.g. XOR, inductive reasoning).

State of the art in the computational model

- Some tools (CryptoVerif, EasyCrypt, CryptHOL, F*).
- CryptoVerif is designed to naturally model protocols, but does not support stateful protocols.

▶ What about the CCSA approach?
An approach between symbolic and computational models

CCSA approach [BC12; BC14]

- Relies on a symbolic setting while providing computational guarantees.
- Has been demonstrated on various protocols [BCE18; CK17; Kou19; SS16].
- But proofs are manual and limited to a bounded number of sessions.

▶ A theoretical framework, called meta-logic, providing computational guarantees for an arbitrary number of sessions.
▶ A new interactive prover, SQUIRREL.
An approach between symbolic and computational models

CCSA approach [BC12; BC14]

- Relies on a symbolic setting while providing computational guarantees.
- Has been demonstrated on various protocols [BCE18; CK17; Kou19; SS16].
- But proofs are manual and limited to a bounded number of sessions.

▶ A theoretical framework, called meta-logic, providing computational guarantees for an arbitrary number of sessions.
▶ A new interactive prover, Squirrel.
CCSA model [BC14], or base logic

A first-order logic built over terms and a single predicate \( \sim \), where:

- **terms** are interpreted as PPT Turing machines;
- **\( \sim \)** is interpreted as **computational indistinguishability**.

More precisely:

- **names** are independent random samplings;
- a special **function symbol** represents the attacker’s computations and corresponds to a probabilistic machine;
- other **function symbols** correspond to deterministic machines.

A computational model \( \mathbb{M} \) is such a possible interpretation.

**Validity**

A base logic formula \( \phi \) is **valid** if \( \forall \mathbb{M}, \mathbb{M} \models \phi \).
CCSA model [BC14], or base logic

A first-order logic built over terms and a single predicate $\sim$, where:

- **terms** are interpreted as PPT Turing machines;
- $\sim$ is interpreted as **computational indistinguishability**.

More precisely:

- **names** are independent random samplings;
- a special **function symbol** represents the **attacker**’s computations and corresponds to a probabilistic machine;
- other **function symbols** correspond to deterministic machines.

A computational model $\mathbb{M}$ is such a possible interpretation.

**Validity**

A base logic formula $\phi$ is **valid** if $\forall \mathbb{M}, \mathbb{M} \models \phi$. 
CCSA model [BC14], or base logic

A first-order logic built over terms and a single predicate \( \sim \), where:

- **terms** are interpreted as PPT Turing machines;
- \( \sim \) is interpreted as **computational indistinguishability**.

More precisely:

- **names** are independent random samplings;
- a special **function symbol** represents the **attacker**’s computations and corresponds to a probabilistic machine;
- other **function symbols** correspond to deterministic machines.

A **computational model** \( \mathcal{M} \) is such a possible interpretation.

**Validity**

A base logic formula \( \phi \) is **valid** if \( \forall \mathcal{M}, \mathcal{M} \models \phi \).
CCSA model [BC14], or base logic

A first-order logic built over terms and a single predicate $\sim$, where:

- **terms** are interpreted as PPT Turing machines;
- $\sim$ is interpreted as **computational indistinguishability**.

More precisely:

- **names** are independent random samplings;
- a special **function symbol** represents the attacker’s computations and corresponds to a probabilistic machine;
- other **function symbols** correspond to deterministic machines.

A **computational model** $\mathcal{M}$ is such a possible interpretation.

Validity

A base logic formula $\phi$ is valid if $\forall \mathcal{M}, \mathcal{M} \models \phi$. 
Axiomatic approach

- **Security properties** are represented by formulas.
- A **proof** is a derivation tree using **inference rules** that correspond to **logical/structural** or **cryptographic** axioms.

**Proof scheme**

\[
\phi_11 \quad \phi_12 \quad \ldots \\
\phi_1 \quad \phi_2 \\
\phi
\]
Axioms as inference rules: two examples

**Structural axiom**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUP</td>
<td>$\Delta \vdash \vec{u}, s \sim \vec{v}, t$</td>
</tr>
<tr>
<td></td>
<td>$\Delta \vdash \vec{u}, s, s \sim \vec{v}, t, t$</td>
</tr>
</tbody>
</table>

**Cryptographic axiom**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRF</td>
<td>if $\text{HFresh}^k(t; \vec{u}, t)$</td>
</tr>
<tr>
<td></td>
<td>$\Delta \vdash \vec{u}$, then $n \sim \vec{v}$</td>
</tr>
<tr>
<td></td>
<td>else $H(t, k)$</td>
</tr>
<tr>
<td></td>
<td>$\Delta \vdash \vec{u}, H(t, k) \sim \vec{v}$</td>
</tr>
</tbody>
</table>

when $\text{SC}^k(t, \vec{u})$
Axioms as inference rules: two examples

**Structural axiom**

\[
\frac{\Delta \vdash \vec{u}, s \sim \vec{v}, t}{\Delta \vdash \vec{u}, s, s \sim \vec{v}, t, t}
\]

**Cryptographic axiom**

\[
\text{PRF} \quad \begin{align*}
&\text{if } \text{HFresh}^k(t; \vec{u}, t) \\
&\text{then } \text{H}(t, k) \sim \vec{v} \\
&\text{else } H(t, k) \sim \vec{v}
\end{align*}
\]

when \( \text{SC}^k(t, \vec{u}) \)
Limitations of the CCSA model [BC14]

Let's say we want to prove (a light notion of) unlinkability for the Basic Hash protocol for tags $T_A$, $T_B$ that can each play 2 sessions.

We would have to manually prove all these equivalences!

\[
\begin{align*}
& m_{T_A} \sim m_{T_1} \\
& m_{T_B} \sim m_{T_1} \\
& m_{T_A}, m'_{T_A} \sim m_{T_1}, m_{T_2} \\
& m_{T_A}, m_{T_B} \sim m_{T_1}, m_{T_2} \\
& m_{T_B}, m'_{T_B} \sim m_{T_1}, m_{T_2} \\
& m_{T_A}, m'_{T_A}, m_{T_B} \sim m_{T_1}, m_{T_2}, m_{T_3} \\
& m_{T_A}, m_{T_B}, m'_{T_B} \sim m_{T_1}, m_{T_2}, m_{T_3} \\
& m_{T_A}, m'_{T_A}, m_{T_B}, m'_{T_B} \sim m_{T_1}, m_{T_2}, m_{T_3}, m_{T_4}
\end{align*}
\]
Building a meta-logic on the base logic

\[
\begin{align*}
\{ \phi_1, \phi_2 \} + \{ \text{base logic inference rules} \} & \rightarrow \{ \text{proof}_1, \text{proof}_2, \text{proof}_3, \text{proof}_4 \} \\
\{ \phi_3, \phi_4 \} & \ldots
\end{align*}
\]
Building a meta-logic on the base logic

\[
\{ \phi_1, \phi_2, \phi_3, \phi_4, \ldots \} + \text{base logic inference rules} \rightarrow \{ \text{proof}_1, \text{proof}_2, \text{proof}_3, \text{proof}_4, \ldots \}\]
Building a meta-logic on the base logic

$\psi$

$\{\phi_1, \phi_2, \phi_3, \phi_4, \ldots\}$

$\{\text{base logic inference rules}\}$

$\rightarrow$

$\{\text{proof}_1, \text{proof}_2, \text{proof}_3, \text{proof}_4, \ldots\}$
Building a meta-logic on the base logic

\[ \psi \]

\[ \{ \phi_1, \phi_2 \} \]
\[ \{ \phi_3, \phi_4 \} \]
\[ ... \]

\[ \{ \text{meta-logic inference rules} \} \]

\[ \rightarrow \]

\[ \text{proof} \]

\[ \{ \text{base logic inference rules} \} \]

\[ \rightarrow \]

\[ \{ \text{proof}_1, \text{proof}_2 \} \]
\[ \{ \text{proof}_3, \text{proof}_4 \} \]
\[ ... \]
Protocols as a set of actions

A protocol is defined by:
- a set of actions,
- equipped with a **dependency relation** to constrain the execution order of actions.

An action is defined by:
- a **condition**,
- an **update term** for each mutable cell,
- and an **output** message.

A trace is a sequence of actions.
A meta-logic built on the CCSA model

Extension of the base logic with:

- **index variables** $i, j$ to parameterize unbounded collections of objects (e.g. names $n[i, j]$);
- **timestamps variables** $\tau$ to quantify over all possible instants of a trace;
- **macros** cond@$\tau$, input@$\tau$, output@$\tau$, ... referring to the action at instant $\tau$;
- **quantifications** over timestamps and indices.
**Basic Hash protocol**

\[
\begin{align*}
\text{cond@T[i,j]} & \overset{\text{def}}{=} \text{true} \\
\text{output@T[i,j]} & \overset{\text{def}}{=} \langle n[i,j], H(n[i,j], k[i]) \rangle \\
\text{cond@R[j']} & \overset{\text{def}}{=} \exists i', \ \text{snd}(\text{input@R[j']}) = H(\text{fst}(\text{input@R[j']}), k[i']) \\
\text{output@R[j']} & \overset{\text{def}}{=} \text{ok} \\
\text{cond@R1[j']} & \overset{\text{def}}{=} \neg(\exists i', \ \text{snd}(\text{input@R1[j']}) = H(\text{fst}(\text{input@R1[j']}), k[i'])) \\
\text{output@R1[j']} & \overset{\text{def}}{=} \text{error}
\end{align*}
\]

*Examples of meta-logic formulas*

\[\forall j', \ \text{cond@R[j']} \Rightarrow (\exists i, j, \ T[i,j] < R[j'] \land \text{output@T}[i,j] = \text{input@R}[j'])\]

\[\forall \tau, \ \text{frame}_{\text{real}}@\tau \sim \text{frame}_{\text{ideal}}@\tau\]
Basic Hash protocol

\[
\begin{align*}
\text{cond@T}[i,j] & \triangleq \text{true} \\
\text{output@T}[i,j] & \triangleq \langle n[i,j], H(n[i,j], k[i]) \rangle
\end{align*}
\]

\[
\begin{align*}
\text{Tag} & \rightarrow \text{Reader} \quad \langle n, H(n, k) \rangle \\
\text{Reader} & \rightarrow \text{Tag} \quad \text{ok}
\end{align*}
\]

\[
\begin{align*}
\text{cond@R}[j'] & \triangleq \exists i', \text{snd(input@R}[j']) = H(\text{fst(input@R}[j']), k[i']) \\
\text{output@R}[j'] & \triangleq \text{ok}
\end{align*}
\]

\[
\begin{align*}
\text{cond@R1}[j'] & \triangleq \neg(\exists i', \text{snd(input@R1}[j']) = H(\text{fst(input@R1}[j']), k[i']) \\
\text{output@R1}[j'] & \triangleq \text{error}
\end{align*}
\]

Examples of meta-logic formulas

\[
\forall j', \text{cond@R}[j'] \Rightarrow (\exists i, j, T[i,j] < R[j'] \land \text{output@T}[i,j] = \text{input@R}[j'])
\]

\[
\forall \tau, \text{frame}_{\text{real}}@\tau \sim \text{frame}_{\text{ideal}}@\tau
\]

25
Basic Hash protocol

\[
\begin{align*}
\text{cond}@T[i,j] & \overset{\text{def}}{=} \text{true} \\
\text{output}@T[i,j] & \overset{\text{def}}{=} \langle n[i,j], H(n[i,j], k[i]) \rangle
\end{align*}
\]

\[
\begin{align*}
\text{Tag} & \rightarrow \text{Reader} \quad \langle n, H(n, k) \rangle \\
\text{Reader} & \rightarrow \text{Tag} \quad \text{ok}
\end{align*}
\]

\[
\begin{align*}
\text{cond}@R[j'] & \overset{\text{def}}{=} \exists i', \text{snd}(\text{input}@R[j']) = H(\text{fst}(\text{input}@R[j']), k[i']) \\
\text{output}@R[j'] & \overset{\text{def}}{=} \text{ok}
\end{align*}
\]

\[
\begin{align*}
\text{cond}@R1[j'] & \overset{\text{def}}{=} \neg (\exists i', \text{snd}(\text{input}@R1[j']) = H(\text{fst}(\text{input}@R1[j']), k[i'])) \\
\text{output}@R1[j'] & \overset{\text{def}}{=} \text{error}
\end{align*}
\]

Examples of meta-logic formulas

\[
\forall j', \text{cond}@R[j'] \Rightarrow (\exists i, j, \ T[i,j] < R[j'] \land \text{output}@T[i,j] = \text{input}@R[j'])
\]

\[
\forall \tau, \text{frame}_{\text{real}}@\tau \sim \text{frame}_{\text{ideal}}@\tau
\]

Basic Hash protocol

\[
\begin{align*}
\text{cond} @ T[i,j] & \overset{\text{def}}{=} \text{true} \\
\text{output} @ T[i,j] & \overset{\text{def}}{=} \langle n[i,j], H(n[i,j], k[i]) \rangle
\end{align*}
\]

\[
\begin{align*}
\text{cond} @ R[j'] & \overset{\text{def}}{=} \exists i', \ \text{snd}(\text{input} @ R[j']) = H(\text{fst}(\text{input} @ R[j']), k[i']) \\
\text{output} @ R[j'] & \overset{\text{def}}{=} \text{ok}
\end{align*}
\]

\[
\begin{align*}
\text{cond} @ R1[j'] & \overset{\text{def}}{=} \neg(\exists i', \ \text{snd}(\text{input} @ R1[j']) = H(\text{fst}(\text{input} @ R1[j']), k[i'])) \\
\text{output} @ R1[j'] & \overset{\text{def}}{=} \text{error}
\end{align*}
\]

Examples of meta-logic formulas

\[
\forall j', \ \text{cond} @ R[j'] \Rightarrow (\exists i, j, \ T[i,j] < R[j'] \land \text{output} @ T[i,j] = \text{input} @ R[j'])
\]

\[
\forall \tau, \ \text{frame}_{\text{real}} @ \tau \sim \text{frame}_{\text{ideal}} @ \tau
\]
Translation from the meta-logic to the base logic

Trace model

For each possible trace of a protocol $\mathcal{P}$, a trace model $\mathcal{T}$ explains how to translate:

- **terms and formulas from the meta-logic**
- **to terms and formulas from the base logic**

by giving a meaning to **index and timestamp variables**.
Translation from the meta-logic to the base logic: example

Let’s consider a trace of the Basic Hash protocol: $T[3, 1].R[2]$. Let’s consider a trace model with:

$$D_I \overset{\text{def}}{=} \{1, 2, 3\} \text{ and } \sigma_I \overset{\text{def}}{=} \{i \mapsto 3, j \mapsto 1, k \mapsto 2\}.$$ 

$$(n[i, j])_P^T \overset{\text{def}}{=} n_{3, 1}$$

$$(\text{output}@T[i, j])_P^T \overset{\text{def}}{=} (\langle n[i, j], H(n[i, j], k[j]) \rangle)_P^T \overset{\text{def}}{=} \langle n_{3, 1}, H(n_{3, 1}, k_3) \rangle$$

$$(\text{input}@R[j'])_P^T \overset{\text{def}}{=} \text{att}(...)$$

$$(\text{cond}@R[j'])_P^T \overset{\text{def}}{=} (\exists i', \text{snd}(\text{input}@R[j'])) = H(\text{fst}(\text{input}@R[j']), k[i'])$$

$$= \text{snd}(\text{att}(...)) = H(\text{fst}(\text{att}(...)), k_1)$$

$$\lor \text{snd}(\text{att}(...)) = H(\text{fst}(\text{att}(...)), k_2)$$

$$\lor \text{snd}(\text{att}(...)) = H(\text{fst}(\text{att}(...)), k_3)$$
Translation from the meta-logic to the base logic: example

Let’s consider a **trace** of the Basic Hash protocol: \( T[3, 1].R[2] \).

Let’s consider a **trace model** with:

\[ D_I \overset{\text{def}}{=} \{1, 2, 3\} \text{ and } \sigma_I \overset{\text{def}}{=} \{i \mapsto 3, j \mapsto 1, k \mapsto 2\} \]

\[
(n[i, j])^T_P \overset{\text{def}}{=} n_{3,1} \\
(output@T[i, j])^T_P \overset{\text{def}}{=} (\langle n[i, j], H(n[i, j], k[i]) \rangle)^T_P \overset{\text{def}}{=} \langle n_{3,1}, H(n_{3,1}, k_3) \rangle \\
(input@R[j'])^T_P \overset{\text{def}}{=} \text{att(...)} \\
(cond@R[j'])^T_P \overset{\text{def}}{=} (\exists i', \text{snd}(input@R[j'])) = H(fst(input@R[j']), k[i']) \overset{\text{def}}{=} \text{snd(att(...))} = H(fst(\text{att(...))}, k_1) \\\n\quad \lor \text{snd(\text{att(...))} = H(fst(\text{att(...))}, k_2) \\\n\quad \lor \text{snd(\text{att(...))} = H(fst(\text{att(...))}, k_3)
Translation from the meta-logic to the base logic: example


Let’s consider a trace model with:

\[
D_{I} \overset{\text{def}}{=} \{1, 2, 3\} \quad \text{and} \quad \sigma_{I} \overset{\text{def}}{=} \{i \mapsto 3, j \mapsto 1, k \mapsto 2\}.
\]

\[
(n[i,j])_{T}^{\mathcal{P}} \overset{\text{def}}{=} n_{3,1}
\]

\[
(\text{output}@T[i,j])_{T}^{\mathcal{P}} \overset{\text{def}}{=} \langle n[i,j], H(n[i,j], k[j]) \rangle_{T}^{\mathcal{P}} \overset{\text{def}}{=} \langle n_{3,1}, H(n_{3,1}, k_{3}) \rangle
\]

\[
(\text{input}@R[j'])_{T}^{\mathcal{P}} \overset{\text{def}}{=} \text{att}(...)
\]

\[
(\text{cond}@R[j'])_{T}^{\mathcal{P}} \overset{\text{def}}{=} (\exists i', \ \text{snd}(\text{input}@R[j'])) = H(\text{fst}(\text{input}@R[j']), k[i'])
\]

\[
\overset{\text{def}}{=} \text{snd}(\text{att}(...)) = H(\text{fst}(\text{att}(...)), k_{1})
\]

\[
\lor \text{snd}(\text{att}(...)) = H(\text{fst}(\text{att}(...)), k_{2})
\]

\[
\lor \text{snd}(\text{att}(...)) = H(\text{fst}(\text{att}(...)), k_{3})
\]
Translation from the meta-logic to the base logic: example

Let’s consider a trace of the Basic Hash protocol: $T[3, 1].R[2]$. Let’s consider a trace model with:

$$D_I \overset{\text{def}}{=} \{1, 2, 3\} \text{ and } \sigma_I \overset{\text{def}}{=} \{i \mapsto 3, j \mapsto 1, k \mapsto 2\}.$$ 

$$(n[i, j])_{T \mathcal{P}} \overset{\text{def}}{=} n_{3, 1}$$

$$(\text{output}@T[i, j])_{T \mathcal{P}} \overset{\text{def}}{=} \langle n[i, j], H(n[i, j], k[i]) \rangle_{T \mathcal{P}} \overset{\text{def}}{=} \langle n_{3, 1}, H(n_{3, 1}, k_3) \rangle$$

$$(\text{input}@R[j'])_{T \mathcal{P}} \overset{\text{def}}{=} \text{att}(\ldots)$$

$$(\text{cond}@R[j'])_{T \mathcal{P}} \overset{\text{def}}{=} (\exists i', \text{snd}(\text{input}@R[j']) = H(\text{fst}(\text{input}@R[j']), k[i']))_{T \mathcal{P}}$$

$$= \text{snd}(\text{att}(\ldots)) = H(\text{fst}(\text{att}(\ldots)), k_1)$$

$$\lor \text{snd}(\text{att}(\ldots)) = H(\text{fst}(\text{att}(\ldots)), k_2)$$

$$\lor \text{snd}(\text{att}(\ldots)) = H(\text{fst}(\text{att}(\ldots)), k_3)$$
Translation from the meta-logic to the base logic: example


Let’s consider a trace model with:

$$D \defeq \{1, 2, 3\} \text{ and } \sigma \defeq \{i \mapsto 3, j \mapsto 1, k \mapsto 2\}.$$

$$(n[i, j])_P \defeq n_{3, 1}$$

$$(\text{output}@T[i, j])_P \defeq (\langle n[i, j], H(n[i, j], k[j]) \rangle)_P \defeq \langle n_{3, 1}, H(n_{3, 1}, k_3) \rangle$$

$$(\text{input}@R[j'])_P \defeq \text{att}(\ldots)$$

$$(\text{cond}@R[j'])_P \defeq (\exists i', \text{snd}(\text{input}@R[j'])) = H(\text{fst}(\text{input}@R[j']), k[i'])}_P \defeq \text{snd}(\text{att}(\ldots)) = H(\text{fst}(\text{att}(\ldots)), k_1)$$

$$\lor \text{snd}(\text{att}(\ldots)) = H(\text{fst}(\text{att}(\ldots)), k_2)$$

$$\lor \text{snd}(\text{att}(\ldots)) = H(\text{fst}(\text{att}(\ldots)), k_3)$$
Quick reminder: in the base logic

A base logic formula $\phi$ is valid if

$$\forall M, M \models \phi.$$ 

In the meta-logic

Given a protocol $P$, a meta-logic formula $\psi$ is valid if

$$\forall T, \forall M, M \models (\psi)^T_P.$$
Reasoning with the meta-logic on protocols

Quick reminder: in the base logic

A base logic formula $\phi$ is valid if

$$\forall M, M \models \phi.$$

In the meta-logic

Given a protocol $\mathcal{P}$, a meta-logic formula $\psi$ is valid if

$$\forall T, \forall M, M \models (\psi)_T^\mathcal{P}.$$
### Lifting axioms from the base logic to the meta-logic (1)

<table>
<thead>
<tr>
<th>Base logic rule</th>
<th>Meta-logic rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DUP</strong></td>
<td><strong>DUP</strong></td>
</tr>
<tr>
<td>[ \Delta \vdash \vec{u}, s \sim \vec{v}, t ]</td>
<td>[ \Delta \vdash \vec{u}, s \sim \vec{v}, t ]</td>
</tr>
<tr>
<td>[ \Delta \vdash \vec{u}, s, s \sim \vec{v}, t, t ]</td>
<td>[ \Delta \vdash \vec{u}, s, s \sim \vec{v}, t, t ]</td>
</tr>
</tbody>
</table>
Lifting axioms from the base logic to the meta-logic (2)

Base logic rule

\[ \text{PRF} \]
\[ \Delta \vdash u, \quad \text{if } \text{HFresh}^k(t; \bar{u}, t) \]
\[ \text{then } n \quad \sim \bar{v} \]
\[ \text{else } H(t, k) \]
\[ \Delta \vdash u, H(t, k) \sim \bar{v} \]

when \( SC^k(t, \bar{u}) \)

Meta-logic rule

\[ \text{PRF} \]
\[ \Delta \vdash \bar{u}, \quad \text{if } \text{HFresh}_P^{k[i]}(t; \bar{u}, t) \]
\[ \text{then } n \quad \sim \bar{v} \]
\[ \text{else } H(t, k[i]) \]
\[ \Delta \vdash \bar{u}, H(t, k[i]) \sim \bar{v} \]

when \( SC_P^{k[i]}(t, \bar{u}) \)

HFresh\(^k(t; \bar{u}, t)\) and \( SC^k(t, \bar{u}) \) can be checked syntactically.

HFresh\(_P^{k[i]}(t; \bar{u}, t)\) and \( SC_P^{k[i]}(t, \bar{u}) \) need to be checked for:
- **direct** occurrences (syntactically),
- and **indirect** occurrences (any action of the protocol).
Lifting axioms from the base logic to the meta-logic (2)

**Base logic rule**

\[
\text{PRF} \quad \frac{\text{if } \text{HFresh}^k(t; \bar{u}, t)}{
\Delta \vdash \bar{u}, \text{ then } n \sim \bar{v} \quad \text{else } H(t, k)}
\]

\[
\Delta \vdash \bar{u}, H(t, k) \sim \bar{v}
\]

when \( \text{SC}^k(t, \bar{u}) \)

**Meta-logic rule**

\[
\text{PRF} \quad \frac{\text{if } \text{HFresh}^{P\bar{i}}_P(t; \bar{u}, t)}{
\Delta \vdash \bar{u}, \text{ then } n \sim \bar{v} \quad \text{else } H(t, k[\bar{i}])}
\]

\[
\Delta \vdash \bar{u}, H(t, k[\bar{i}]) \sim \bar{v}
\]

when \( \text{SC}^{P\bar{i}}_P(t, \bar{u}) \)

\( \text{HFresh}^k(t; \bar{u}, t) \) and \( \text{SC}^k(t, \bar{u}) \) can be checked syntactically.

\( \text{HFresh}^{P\bar{i}}_P(t; \bar{u}, t) \) and \( \text{SC}^{P\bar{i}}_P(t, \bar{u}) \) need to be checked for:
- **direct** occurrences (syntactically),
- and **indirect** occurrences (any action of the protocol).
Lifting axioms from the base logic to the meta-logic (2)

**Base logic rule**

\[ \text{PRF} \]

If \( \text{HFresh}^k(t; \vec{u}, t) \)

\[ \Delta \vdash \vec{u}, \text{then } n \sim \vec{v} \]

else \( H(t, k) \)

\[ \frac{\Delta \vdash \vec{u}, H(t, k) \sim \vec{v}}{\Delta \vdash \vec{u}, H(t, k) \sim \vec{v}} \]

when \( \text{SC}^k(t, \vec{u}) \)

\( \text{HFresh}^k(t; \vec{u}, t) \) and \( \text{SC}^k(t, \vec{u}) \) can be checked syntactically.

---

**Meta-logic rule**

\[ \text{PRF} \]

If \( \text{HFresh}_{\mathcal{P}}^{k[\vec{i}]}(t; \vec{u}, t) \)

\[ \Delta \vdash \vec{u}, \text{then } n \sim \vec{v} \]

else \( H(t, k[\vec{i}]) \)

\[ \frac{\Delta \vdash \vec{u}, H(t, k[\vec{i}]) \sim \vec{v}}{\Delta \vdash \vec{u}, H(t, k[\vec{i}]) \sim \vec{v}} \]

when \( \text{SC}_{\mathcal{P}}^{k[\vec{i}]}(t, \vec{u}) \)

\( \text{HFresh}_{\mathcal{P}}^{k[\vec{i}]}(t; \vec{u}, t) \) and \( \text{SC}_{\mathcal{P}}^{k[\vec{i}]}(t, \vec{u}) \) need to be checked for:

- direct occurrences (syntactically),
- and indirect occurrences (any action of the protocol).
Lifting axioms from the base logic to the meta-logic (2)

Base logic rule

\[ \text{PRF} \]

\[ \Delta \vdash \bar{u}, \quad \text{if } \text{HFresh}^k(t; \bar{u}, t) \]

\[ \Delta \vdash \bar{u}, \quad \text{then } n \sim \bar{v} \]

\[ \quad \text{else } H(t, k) \]

\[ \Delta \vdash \bar{u}, H(t, k) \sim \bar{v} \]

when \( SC^k(t, \bar{u}) \)

Meta-logic rule

\[ \text{PRF} \]

\[ \Delta \vdash \bar{u}, \quad \text{if } \text{HFresh}^k[\bar{i}](t; \bar{u}, t) \]

\[ \Delta \vdash \bar{u}, \quad \text{then } n \sim \bar{v} \]

\[ \quad \text{else } H(t, k[\bar{i}]) \]

\[ \Delta \vdash \bar{u}, H(t, k[\bar{i}]) \sim \bar{v} \]

when \( SC^k[\bar{i}](t, \bar{u}) \)

HFresh\( ^k(t; \bar{u}, t) \) and \( SC^k(t, \bar{u}) \)
can be checked syntactically.

HFresh\( ^k[\bar{i}](t; \bar{u}, t) \) and \( SC^k[\bar{i}](t, \bar{u}) \)
need to be checked for:
- direct occurrences (syntactically),
- and indirect occurrences (any action of the protocol).
Building a meta-logic on the base logic

\[ \psi \] + \{ meta-logic inference rules \} \rightarrow \{ proof_1, proof_2, proof_3, proof_4 \} + \{ base logic inference rules \} \rightarrow \{ \ldots \}
Building a meta-logic on the base logic

\[
\psi + \{ \text{base logic inference rules} \} \rightarrow \{ \text{proof_1, proof_2, proof_3, proof_4} \}
\]
The **Squirrel** tool

The **input language** is a variant of the applied-pi calculus.

The tool implements ($\approx$ 10,000 lines of OCaml code):

- the **translation** of the specification of the protocol from the input language to actions,
- **proof tactics**, corresponding to inference rules,
- **automated reasoning** to ease the proof effort.

The **user** interacts with the prover by **calling proof tactics** to derive formulas step by step.
The **Squirrel** tool

The **input language** is a variant of the applied-pi calculus.

The tool implements (≈ 10,000 lines of OCaml code):

- the **translation** of the specification of the protocol from the input language to actions,
- **proof tactics**, corresponding to inference rules,
- **automated reasoning** to ease the proof effort.

The user interacts with the prover by calling **proof tactics** to derive formulas step by step.
The Squirrel tool

The input language is a variant of the applied-pi calculus.

The tool implements (≈ 10,000 lines of OCaml code):

- the translation of the specification of the protocol from the input language to actions,
- proof tactics, corresponding to inference rules,
- automated reasoning to ease the proof effort.

The user interacts with the prover by calling proof tactics to derive formulas step by step.
Basic Hash with SQUIRREL (1)

BASIC HASH


T --> R : <n, h(n,k)>
R --> T : ok

abstract ok : message
abstract error : message

name key : index->message
name key' : index->index->message

process tag(i:index, k:index) =
    new nT;
    out(cT, <nT, h(nT,diff(key(i),key'(i,k))>)

process reader(j:index) =
    in(cT,x);
    if exists (i,k:index),
        snd(x) = h(fst(x),diff(key(i),key'(i,k))
    then out(cR,ok)
    else out(cR,error)

system
    (!(j R: reader(j)) | !(i !_k T: tag(i,k)))
Basic Hash with SQUIRREL (2)

-------------------------------
BASIC HASH


T --> R : <n, h(n,k)>
R --> T : ok
-------------------------------

hash h

abstract ok : message
abstract error : message

name key : index->message
name key' : index->index->message

channel cT
channel cR.

process tag(i:index,k:index) =
    new nT;
    out(cT, <nT, h(nT,diff(key(i),key'(i,k))>>)

process reader(j:index) =
    in(cT,x);
    if exists (i,k:index),
        snd(x) = h(fst(x),diff(key(i),key'(i,k)));
    then out(cR,ok)
    else out(cR,error)

system
((!_j R: reader(j)) | (!_i !_k T: tag(i,k))).
Basic Hash with Squirrel (3)

(* Authentication goal for the action R
 (then branch of the reader) *)

goal wa_R :
  forall (j: index),
  happens(R(j)) =>
  (cond(R(j) <=
   (exists (i,k: index), T(i,k) < R(j) &&
   fst(output@T(i,k)) = fst(input@R(j)) &&
   snd(output@T(i,k)) = snd(input@R(j))).

Proof.
  intro *.
  expand cond.
  split.
  project.
  (* LEFT *) by euf Meq; exists i, k0.
  (* RIGHT *) by euf Meq; exists i,k.
  by exists i,k.
Qed.

[goal> Focused goal (1/3):
System: default/left
Variables: i,j,k: index
D: input@R(j) = att(frame@pred(R(j)))
Hap: happens(R(j))
Meq: snd(input@R(j)) = h(fst(input@R(j));key(i))
----------------------------------------
exists (i,k: index),
  (T(i,k) < R(j) && fst(output@T(i,k)) = fst(input@R(j)) &&
snd(output@T(i,k)) = snd(input@R(j)))

U:88c-- *goals* All L1 (squirrel goals +1)
## Case studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Security Properties</th>
<th>Lemmas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Hash</td>
<td>Authentication, Unlinkability</td>
<td>-</td>
</tr>
<tr>
<td>Hash-Lock</td>
<td>Authentication, Unlinkability</td>
<td>-</td>
</tr>
<tr>
<td>Feldhofer</td>
<td>Authentication, Unlinkability</td>
<td>-</td>
</tr>
<tr>
<td>LAK (pairs, fix v1)</td>
<td>Authentication, Unlinkability</td>
<td>-</td>
</tr>
<tr>
<td>MW</td>
<td>Authentication, Unlinkability</td>
<td>-</td>
</tr>
<tr>
<td>Private Authentication</td>
<td>Anonymity</td>
<td>-</td>
</tr>
<tr>
<td>Toy Counter</td>
<td>Secrecy</td>
<td>(i2)</td>
</tr>
<tr>
<td>SLK06</td>
<td>Authentication</td>
<td>-</td>
</tr>
<tr>
<td>YPLRK05</td>
<td>Authentication</td>
<td>(i1)</td>
</tr>
<tr>
<td>YubiKey</td>
<td>Injective Correspondence, Monotonicity</td>
<td>(i2)</td>
</tr>
<tr>
<td>Toy Hash</td>
<td>No Replay, Authentication</td>
<td>-</td>
</tr>
<tr>
<td>Toy Hash (only tags)</td>
<td>Strong Secrecy</td>
<td>(i1), (i2)</td>
</tr>
</tbody>
</table>

(i1) reasoning on the last update
(i2) reasoning on monotonicity of state values
Conclusion
Contributions

Precise **model** for unlinkability of stateful two-party protocols.

Theoretical **verification** method in the **symbolic** model, based on sufficient conditions. Validated by **case studies** on existing RFID protocols, using the **tool** **Tamarin**.

Theoretical **verification** framework providing **computational** guarantees, by extending the CCSA approach. **Implemented** in a new interactive prover, **SQUIRREL**. Validated by **case studies**.

► Parts of these results have been published at the conferences **CSF 2020** ("distinguished paper") and **S&P 2021**.
Contributions

Precise **model** for unlinkability of stateful two-party protocols.

Theoretical **verification** method in the **symbolic** model, based on sufficient conditions. Validated by **case studies** on existing RFID protocols, using the tool **Tamarin**.

Theoretical **verification** framework providing **computational** guarantees, by extending the CCSA approach. **Implemented** in a new interactive prover, **Squirrel**. Validated by **case studies**.

- Parts of these results have been published at the conferences **CSF 2020** ("distinguished paper") and **S&P 2021**.
Perspectives

Bringing SQUIRREL to the level of mature tools

- Support more cryptographic primitives.
- **Generalize** the proof system:
  - *e.g.* to enable proofs in the Random Oracle Model.
- Add more **automated reasoning**:
  - *e.g.* bi-deduction, reasoning on states, SMT solvers...

Bridging symbolic and computational approaches

- Combine approaches and tools into a progressive analysis.
- Similarities in the proof methodology.
Perspectives

Bringing **SQUIRREL** to the level of mature tools

- Support more cryptographic primitives.
- **Generalize** the proof system:
  - *e.g.* to enable proofs in the Random Oracle Model.
- Add more **automated reasoning**:
  - *e.g.* bi-deduction, reasoning on states, SMT solvers...

Bridging symbolic and computational approaches

- Combine approaches and tools into a **progressive analysis**.
- **Similarities** in the proof methodology.