Formal verification of security protocols -
the Squirrel prover

Stéphanie DELAUNE
FPS - December 11, 2023
Cryptographic protocols everywhere!

Cryptographic protocols

- distributed programs designed to secure communication (e.g. secrecy, authentication, anonymity, ...)
- use cryptographic primitives (e.g. encryption, signature, hash function, ...)
The network is unsecure!

Communications take place over a public network like the Internet.
Cryptographic protocols everywhere!

Cryptographic protocols

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

They aim to secure our communications and protect our privacy.
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.
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 also ensure **unlinkability**.

**ISO/IEC standard 15408**

**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.*
An attack on the BAC protocol

An attack against **unlinkability** on the BAC protocol [Chothia et al., 2010]

- This issue was due to overly specific error messages;
- French passports were vulnerable.
Contactless payment

• In the first quarter of 2020, there was a 40% growth in contactless transactions.
• In France, 4.6 billion of transactions were paid contactless in 2020 (40%).

Authentication with physical proximity

We want to ensure that the transaction is performed by a legitimate credit card, but actually the one close to the reader during the transaction.
Contactless payment is vulnerable to relay attack

Do you know what you’re paying for? How contactless cards are still vulnerable to relay attack

Published: 2 août 2016, 18:19 CEST

The Conversation - Aug. 2016

How does it work?
Contactless payment is vulnerable to relay attack

Do you know what you’re paying for? How contactless cards are still vulnerable to relay attack

The Conversation - Aug. 2016

How does it work?

→ specific protocols, distance bounding protocols, have been designed to mitigate relay attack (included in the EMV specification since 2016)
How cryptographic protocols can be attacked?

Several levels of attacks, which may exploit:

- weaknesses of cryptographic primitives;
- flaws in the design of the protocol;
- bugs in implementations.
How cryptographic protocols can be attacked?

Several levels of attacks, which may exploit:

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

**Flaws in the design of the protocol**

- can be mounted even assuming **perfect** cryptography, → **replay attack**, **man-in-the middle attack**, ...
- **subtle and hard to detect** by “eyeballing” the protocol
Two additional examples of logical attacks

An authentication flaw on the Needham Schroeder protocol

\[
\begin{align*}
A \to B &: \{A, N_A\}_{\text{pub}(B)} \\
B \to A &: \{N_A, N_B\}_{\text{pub}(A)} \\
A \to B &: \{N_B\}_{\text{pub}(B)}
\end{align*}
\]


---

FREAK attack by Barghavan et al. (2015)

A logical flaw that allows a man-in-the-middle attacker to down-grade connections from 'strong' RSA to 'export grade' RSA.

websites affected by the vulnerability included those from the US federal government
Two additional examples of logical attacks

An authentication flaw on the Needham Schroeder protocol

\[ A \to B : \{ A, N_A \}_{\text{pub}(B)} \quad A \to B : \{ A, N_A \}_{\text{pub}(B)} \]
\[ B \to A : \{ N_A, N_B \}_{\text{pub}(A)} \quad B \to A : \{ N_A, N_B, B \}_{\text{pub}(A)} \]
\[ A \to B : \{ N_B \}_{\text{pub}(B)} \quad A \to B : \{ N_B \}_{\text{pub}(B)} \]


FREAK attack by Barghavan et al. (2015)

A logical flaw that allows a man-in-the-middle attacker to downgrade connections from 'strong' RSA to 'export grade' RSA.

→ websites affected by the vulnerability included those from the US federal government
How to verify the absence of logical flaws?

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

Our approach: formal verification using tools

We aim at providing a rigorous framework and verification tools (e.g., Squirrel) to analyse security protocols and find their logical flaws.
How to verify the absence of logical flaws?

- dissect the protocol and test their resilience against well-known attacks;  
  \[\rightarrow \text{this is not sufficient!}\]

- perform a manual security analysis  
  \[\rightarrow \text{this is error-prone!}\]
How to verify the absence of logical flaws?

- dissect the protocol and test their resilience against well-known attacks;
  \[\rightarrow \text{this is not sufficient!}\]

- perform a manual security analysis
  \[\rightarrow \text{this is error-prone!}\]

**Our approach: formal verification using tools**

We aim at providing a rigorous framework and verification tools (e.g. Squirrel) to analyse security protocols and find their logical flaws.
Outline

I. Symbolic versus Computational model

II. A novel approach: the Squirrel prover
Part I

Two main families of models: symbolic versus computational
## Two main families of models

<table>
<thead>
<tr>
<th>Symbolic models</th>
<th>Computational models</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Dolev &amp; Yao, 81]</td>
<td>[Goldwasser &amp; Micali, 84]</td>
</tr>
</tbody>
</table>

- Messages are terms.
- Messages are bitstrings.
- What the attacker can do.
- What the attacker cannot do.
  - Everything else is allowed!

### Unclear guarantees.
- Stronger guarantees.

### Amenable to automation.
- E.g. Proverif, Tamarin.
- E.g. CryptoVerif.

1. The attacker is a probabilistic polynomial-time Turing machine.
Two main families of models

<table>
<thead>
<tr>
<th>Symbolic models</th>
<th>Computational models</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Dolev &amp; Yao, 81]</td>
<td>[Goldwasser &amp; Micali, 84]</td>
</tr>
<tr>
<td>Messages are terms.</td>
<td>Messages are bitstrings.</td>
</tr>
<tr>
<td>What the attacker can do.</td>
<td>What the attacker can not do.</td>
</tr>
<tr>
<td>Unclear guarantees.</td>
<td>Stronger guarantees.</td>
</tr>
<tr>
<td>Amenable to automation.</td>
<td>Harder to automate.</td>
</tr>
<tr>
<td>e.g. Proverif, Tamarin</td>
<td>e.g. CryptoVerif</td>
</tr>
</tbody>
</table>

1 The attacker is a probabilistic polynomial-time Turing machine.
Example: Basic Hash protocol

- Each tag stores a secret key $k$ that is never updated.
- Readers have access to a database DB containing all the keys.

Security properties

- **authentication**: when the reader accepts a message, it has indeed been sent by a legitimate tag;
- **unlinkability**: it is not possible to track tags.

[Weis et al., 03]
Protocols as processes

→ a programming language with constructs for concurrency and communication

(applied-pi calculus [Abadi & Fournet, 01])

\[ P, Q := 0 \text{ null process} \]
\[ \text{in}(c, x); P \text{ input} \]
\[ \text{out}(c, M); P \text{ output} \]
\[ \text{new } n; P \text{ name generation} \]
\[ \text{if } M = N \text{ then } P \text{ else } Q \text{ conditional} \]
\[ !P \text{ replication} \]
\[ (P | Q) \text{ parallel composition} \]
Protocols as processes

→ a programming language with constructs for concurrency and communication

(applied-pi calculus [Abadi & Fournet, 01])

\[ P, Q := \begin{align*}
0 & \quad \text{null process} \\
\text{in}(c, x); P & \quad \text{input} \\
\text{out}(c, M); P & \quad \text{output} \\
\text{new } n; P & \quad \text{name generation} \\
\text{if } M = N \text{ then } P \text{ else } Q & \quad \text{conditional} \\
!P & \quad \text{replication} \\
(P | Q) & \quad \text{parallel composition} \\
\text{insert } tbl(M); P & \quad \text{insertion} \\
\text{get } tbl(x) \text{ st. } M = N \text{ in } P \text{ else } Q & \quad \text{lookup} \\
\ldots &
\end{align*} \]
Basic Hash protocol in the symbolic setting

An abstract model, also known as Dolev-Yao model [Dolev & Yao, 81]

Modelling messages/computations

\[ \Sigma = \{ \langle \rangle, \text{proj}_1, \text{proj}_2, h \} \]
\[ E = \{ \text{proj}_1(\langle x_1, x_2 \rangle) = x_1, \ \text{proj}_2(\langle x_1, x_2 \rangle) = x_2 \} \]

- all the function symbols are public (available to the attacker);
- no equation regarding the hash function.

Modelling protocols as processes, roughly a labelled transition system

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

where:

- \( T(k) = \text{new } n; \text{out}(c, \langle n, h(n, k) \rangle) \).
- \( R = \text{in}(c, y); \text{get } db(k) \text{ st. } h(\text{proj}_1(y), k) = \text{proj}_2(y) \text{ in out}(c, \text{ok}) \text{ else out}(c, \text{ko}) \).
Basic Hash in the computational setting

→ The cryptographer’s mathematical model for provable security

[Goldwasser & Micali, 84]

In computational model, properties only hold with overwhelming probability, under some assumptions on cryptographic primitives

Some usual cryptographic assumptions for a hash function:

- **Collision Resistance (CR)**: « $h(n, k) = h(n', k)$ implies $n = n'$ »
- **PseudoRandom Function (PRF)**: « $h(n, k) \sim r$ »
- **Existential UnForgeability (EUF)**: ...
Basic Hash in the computational setting: authentication

**Existential UnForgeability (EUF)**

There is a negligible probability of success for the following game, for any attacker $A$ (i.e. any PPTM):

- Draw $k$ uniformly at random.
- $(u, v) := A^O$ where $O$ is the oracle $x \rightarrow h(x, k)$.
- Succeed if $u = h(v, k)$ and $O$ has not been called on $v$. 

Security proof: « Reader accepts $m$ implies $m$ emitted by a legitimate tag. »

- Assume reader accepts some $m$ such that $\text{proj}_2(m) = h(\text{proj}_1(m), k_i)$ for some $i$.
- By unforgeability, $\text{proj}_1(m) = n_T$ for some session of tag $T_i$.
- The two projections of $m$ are the two projections of the output of $T_i$. □
Existential UnForgeability (EUF)

There is a negligible probability of success for the following game, for any attacker \( A \) (i.e. any PPTM):

- Draw \( k \) uniformly at random.
- \( \langle u, v \rangle := A^O \) where \( O \) is the oracle \( x \rightarrow h(x, k) \).
- Succeed if \( u = h(v, k) \) and \( O \) has not been called on \( v \).

Security proof: « Reader accepts \( m \) implies \( m \) emitted by a legitimate tag. »

- Assume reader accepts some \( m \) such that \( \text{proj}_2(m) = h(\text{proj}_1(m), k_i) \) for some \( i \).
- By unforgeability, \( \text{proj}_1(m) = n_T \) for some session of tag \( T_i \).
- The two projections of \( m \) are the two projections of the output of \( T_i \).
Limitations of symbolic model

- Security assumptions can be imprecise (cf. EUF and PRF).
- Obtaining computational guarantees from the symbolic model is hard!

**A fundamental problem**

One should not specify what the attacker can do but what is safe.
Limitations of symbolic model

- Security assumptions can be imprecise (cf. EUF and PRF).
- Obtaining computational guarantees from the symbolic model is hard!

A fundamental problem

One should not specify what the attacker can do but what is safe.

The CCSA (Computational Complete Symbolic Attacker) approach, now implemented in the Squirrel prover, does just this, while keeping the modelling of messages as (abstract) terms with a computational semantics, to allow verification via automated reasoning.
### Brief comparison of some existing verification tools

<table>
<thead>
<tr>
<th></th>
<th>DeepSec/Akiss</th>
<th>ProVerif/GSverif</th>
<th>Tamarin</th>
<th>Squirrel</th>
<th>CryptoVerif</th>
<th>EasyCrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>unbounded traces</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>computational attacker</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>concrete security bounds</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>native concurrency</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>global mutable states</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>automation</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↓</td>
<td>↑</td>
<td>↓</td>
</tr>
</tbody>
</table>

**Disclaimer:**

Squirrel is less mature than any of the other tools.
Part II

A Novel approach:
the Squirrel prover
What is Squirrel?

A proof assistant for verifying cryptographic protocols in the computational model.

https://squirrel-prover.github.io/

It is based on the CCSA approach:


A Computationally Complete Symbolic Attacker for Equivalence Properties.
History of Squirrel

Some papers related to Squirrel:

- 2012: Towards Unconditional Soundess: CCSA (Bana & Comon)
- 2014: CCSA for equivalence properties (Bana & Comon)
- 2017: Some manual proofs of RFID protocols (Comon & Koutsos)
- 2021: Introduction of the meta-logic and the Squirrel prover (Baelde et al.)
- 2022: Mutable states and tactics to reason about them (Baelde et al.)

What's new in 2023

A user manual and you can now play with Squirrel without installing it!

https://squirrel-prover.github.io/jsquirrel/


→ most of these people are at IRISA, LMF, and Inria Paris.
Squirrel prover

A tool for verifying security protocols in the computational model which takes in input:

- protocols written in a process algebra (as in symbolic models), and internally translated into a system of actions;
- reachability and equivalence properties.
Squirrel prover

A tool for verifying security protocols in the computational model which takes in input:

- protocols written in a process algebra (as in symbolic models), and internally translated into a system of actions;
- reachability and equivalence properties.

Squirrel is a proof assistant, i.e. users prove goals using sequence of tactics:

- logical tactics: apply, intro, rewrite, ...
- cryptographic tactics: fresh, prf, euf, collision-resistant, ...

→ All the reasoning about probabilities are hidden to the user, and each tactic is proved to be sound (manually once and for all).
Going back to the Basic Hash protocol

The process is immediately translated into a system of actions, i.e. a set of triples: (input; test; output).
Basic Hash as a system of actions

Tag is modelled with **one action**, namely \( T[i, k] \):

- input\( @T[i, k] \);
- true; and
- output\( @T[i, k] = \langle n_T[i, k], h(n_T[i, k], key[i]) \rangle \).
Basic Hash as a system of actions

Tag is modelled with **one action**, namely $T[i, k] :$

- input@ $T[i, k]$;
- true; and
- output@ $T[i, k] = \langle n_T[i, k], h(n_T[i, k], key[i]) \rangle$.

Reader is modelled with **two actions**, namely $R_1[j]$ and $R_2[j] :$

- input@ $R_1[j]$;
- $\exists i. \text{snd}(\text{input}@ R_1[j]) = h(\text{fst}(\text{input}@ R_1[j]), key[i])$;
- output@ $R_1[j] = \text{ok}$;

- input@ $R_2[j]$;
- $\forall i. \text{snd}(\text{input}@ R_2[j]) \neq h(\text{fst}(\text{input}@ R_2[j]), key[i])$;
- output@ $R_2[j] = \text{ko}$. 
Lemma [BasicHash] authentication:

forall (j:index), happens(R1(j)) =>
cond@R1(j) =>
(exists (i,k:index), T(i,k) < R1(j)
&& fst(output@T(i,k)) = fst(input@R1(j))
&& snd(output@T(i,k)) = snd(input@R1(j))).

Proof.
intro j Hap Hcond.
expand cond@R1(j).
destruct Hcond as [i0 HEq].
euf HEq.
intro [k0 [HOrd Eq]].
by exists i0, k0.
Qed.

→ The proof script contains logical tactics (here intro, exists) and also a crypto tactic (here euf).
Logical reasoning

→ All tactics have been proved to be sound manually once and for all.

For crypto axioms, they have been designed first at the base logic level (CCSA), and then lift at the meta-logic level, and their soundness have been established in two steps.

Example:

**Base logic rule:**

\[ \Gamma, t = n \vdash \phi \quad \text{where } n \notin \text{st}(t) \]

**Meta-logic rule:**

\[ \Gamma, \bigvee_{(n[j], k, c) \in \text{st}_P(t)} \exists \vec{k}.c \land \vec{i} = \vec{j} \vdash \phi \]

\[ \Gamma, t = n[i] \vdash \phi \]
Squirrel offline Demo - authentication on Basic Hash

```
else R2: out(cR,ko).

system [BasicHash] (((:_ j R: reader(j)) | ((!_ i !_ k T: tag(i,k))))).

lemma [BasicHash] authentication :
  forall {j: index}, happens(R1(j)) =>
  cond@R1(j) =>
  (exists (i,k: index), T(i,k) < R1(j)
  & & fst(output@T(i,k)) = fst(input@R1(j))
  & & snd(output@T(i,k)) = snd(input@R1(j))).

Proof.
  intro j Hap Hcond.
  expand cond@R1(j).
  destruct Hcond as [i0 HEq].
  euf HEq.
  intro [k0 [H0rd Eq]].
  by exists i0, k0.
  Qed.
```

---

[goal> Focused goal (1/1):]  
System: BasicHash

forall (j: index),
  happens(R1(j)) =>
  cond@R1(j) =>
  exists (i,k: index),
  T(i,k) < R1(j) & &
  fst(output@T(i,k)) = fst(input@R1(j)) & &
  snd(output@T(i,k)) = snd(input@R1(j))

U:%*~ *goals~* All L1 (Squirrel goals)
Squirrel offline Demo - authentication on Basic Hash

```
else R2: out(cR,k0).

system [BasicHash] (((!_j R: reader(j)) | (!_i !_k T: tag(i,k))).

lemma [BasicHash] authentication :
   forall (j:index), happens(R1(j)) =>
      cond@R1(j) =>
         (exists (i,k:index), T(i,k) < R1(j)
           && fst(output@T(i,k)) = fst(input@R1(j))
           && snd(output@T(i,k)) = snd(input@R1(j))).

Proof.
   intro j Hap Hcond.
   expand cond@R1(j).
   destruct Hcond as [i0 HEq].
   euf HEq.
   intro [k0 [HOrd Eq]].
   by exists i0, k0.
   Qed.
```

[goal> Focused goal (1/1):
 System: BasicHash
 Variables: j:index[const]
 Hap: happens(R1(j))
 Hcond: cond@R1(j)

exists (i,k:index),
   T(i, k) < R1(j) &&
   fst(output@T(i, k)) = fst(input@R1(j)) &&
   snd(output@T(i, k)) = snd(input@R1(j))

U:%%- *goals* All L1 (Squirrel goals)
Squirrel offline Demo - authentication on Basic Hash

```plaintext
else R2: out(cR,ko).

system [BasicHash] ((!_ R: reader(j)) | (!_ !_ k: tag(i,k))).

lemma [BasicHash] authentication :
  forall (j: index), happens(R1(j)) =>
  conds(R1(j)) =>
    (exists (i,k: index), T(i,k) < R1(j)
      &\& fst(output@T(i,k)) = fst(input@R1(j))
      &\& snd(output@T(i,k)) = snd(input@R1(j))).

Proof.
intro j Hap Hcond.
expand conds(R1(j)).
destruct Hcond as [H E].
euf E.
intro [k0 [HOrd E]].
by exists 10, k0.
Qed.
```

```
[goal> Focused goal (1/1):
  System: BasicHash
  Variables: j:index[const]
  Hap: happens(R1(j))
  Hcond: exists (i:index), snd (input@R1(j)) = h (fst (input@R1(j)), key i)

exists (i,k:index),
  T(i, k) < R1(j) &\&
  fst(output@T(i,k)) = fst(input@R1(j)) &\&
  snd(output@T(i,k)) = snd(input@R1(j))

U:≡⇒ *goals-* All L1 (Squirrel goals)
```
Squirrel offline Demo - authentication on Basic Hash

```coq
else R2: out(cR, ko).

system [BasicHash] ((\_1_j R: reader(j)) | (\_i !_k T: tag(i,k))).

lemma [BasicHash] authentication :
 forall (j: index), happens(R1(j)) =>
  cond@R1(j) =>
    (exists (i,k: index), T(i,k) < R1(j)
     && fst(output@T(i,k)) = fst(input@R1(j))
     && snd(output@T(i,k)) = snd(input@R1(i))).

Proof.
intro j Hcond.
expand cond@R1(j).
destruct Hcond as [i0 HEq].
rew HEq.
intro [k0 [HOrd Eq]].
by exists i0, k0.
Qed.
```

[goal> Focused goal (1/1):
  System: BasicHash
  Variables: i0, j: index[]
  HEq: snd (input@R1(j)) = h (fst (input@R1(j)), key i0)
  Hap: happens(R1[j])

  exists (i,k: index),
      T(i, k) < R1(j) &&
      fst (output@T(i, k)) = fst (input@R1(j)) &&
      snd (output@T(i, k)) = snd (input@R1(j))

U: %<goals* All L1 (Squirrel goals)
Squirrel offline Demo - authentication on Basic Hash

```plaintext
else R2: out(cR,ko).

system [BasicHash] (((j R: reader(j)) | ((i !(i k T: tag(i,k)))))

lemma [BasicHash] authentication :
for all (j:index), happens(R1(j)) ⇒
cond(R1(j))

{exists (i,k:index), T(i,k) < R1(j)
  \&\& \text{fst(output@T(i,k))} = \text{fst(input@R1(j))}
  \&\& \text{snd(output@T(i,k))} = \text{snd(input@R1(j))}.}

Proof.
intro j Hap Hcond.
expand cond(R1(j)).
destruct Hcond as [i0 HEq].
euf HEq.
intro [k0 [HOrd Eq]].
by exists i0, k0.
Qed.
```

```plaintext
[goal] Focus goal (1/1):
System: BasicHash
Variables: i0,j:index[const]
HEq: snd (input@R1(j)) = h (fst (input@R1(j)), key i0)
Hap: happens(R1(j))

(exists (k:index), T(i0, k) < R1(j) \&\& \text{fst (input@R1(j))} = nT (i0, k)) ⇒
exists (i,k:index),

T(i,k) < R1(j) \&\&
\text{fst (output@T(i,k))} = \text{fst (input@R1(j))} \&\&
\text{snd (output@T(i,k))} = \text{snd (input@R1(j))}

U:%%- #goals# All L1 (Squirrel goals)
in other actions:

nT (i, k) auth. by key(i)
(collision with \text{fst (input@R1(j))} auth. by key(i0))
in action T(i, k)
in term <nT (i, k), h (nT (i, k), key i>)

Total: 1 occurrence
0 of them are subsumed by another
1 occurrence remaining
```
Squirrel offline Demo - authentication on Basic Hash

```lean
else R2: out(cR,k0).

system [BasicHash] (((!_j R: reader(j)) | (!_i !_k T: tag(i,k)))).

lemma [BasicHash] authentication :
  forall (j: index), happens(R1(j)) =>
    condeR1(j) =>
      (exists (i,k: index), T(i,k) < R1(j)
       >> fst(output@T(i,k)) = fst(input@R1(j))
       >> snd(output@T(i,k)) = snd(input@R1(j))).

Proof.
  intro j Hap Hcond.
  expand condeR1(j).
  destruct Hcond as [i0 HEq].
  euf HEq.
  intro [k0 [HOrd Eq]].
  by exists i0, k0.
  Qed.
```

[goal> Focused goal (1/1):
  System: BasicHash
  Variables: i0,j,k0:index[const]
  Eq: fst (input@R1(j)) = h (fst (input@R1(j)), key i0)
  HOrd: T(i0, k0) < R1(j)
  Hap: happens(R1(j))

  exists (i,k: index),
    T(i, k) < R1(j) &
    fst (output@T(i, k)) = fst (input@R1(j)) &
    snd (output@T(i, k)) = snd (input@R1(j))
]

U: %# *goals* All L1 (Squirrel goals)
Squirrel offline Demo - authentication on Basic Hash

```
else R2: out(cR,k0).

system [BasicHash] (((j R: reader(j)) | (!i !k T: tag(i,k))).

lemma [BasicHash] authentication :
for all (j:index), happens(R1(j)) =>
cond@R1(j) =>
(exists (i,k:index), T(i,k) < R1(j)
  && fst(output@T(i,k)) = fst(input@R1(j))
  && snd(output@T(i,k)) = snd(input@R1(j))).

Proof.
intro j Hap Hcond.
expand cond@R1(j).
destruct Hcond as [i0 HEq].
euf HEq.
intro [k0 [HOrd Eq]].
by exists i0, k0.
Qed.
```
## Benchmark

<table>
<thead>
<tr>
<th>Protocol name</th>
<th>LoC</th>
<th>Assumptions</th>
<th>Security Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Hash</td>
<td>100</td>
<td>Prf, Euf</td>
<td>authentication &amp; unlinkability</td>
</tr>
<tr>
<td>Hash Lock</td>
<td>130</td>
<td>Prf, Euf</td>
<td>authentication &amp; unlinkability</td>
</tr>
<tr>
<td>LAK (with pairs)</td>
<td>250</td>
<td>Prf, Euf</td>
<td>authentication &amp; unlinkability</td>
</tr>
<tr>
<td>MW</td>
<td>300</td>
<td>Prf, Euf, Xor</td>
<td>authentication &amp; unlinkability</td>
</tr>
<tr>
<td>Feldhofer</td>
<td>270</td>
<td>Enc-Kp, Int-Ctx</td>
<td>authentication &amp; unlinkability</td>
</tr>
<tr>
<td>Private authentication</td>
<td>100</td>
<td>Cca₁, Enc-Kp</td>
<td>anonymity</td>
</tr>
<tr>
<td>Signed DDH [ISO 9798-3]</td>
<td>240</td>
<td>Euf, Ddh</td>
<td>authentication &amp; strong secrecy</td>
</tr>
<tr>
<td>CANAuth</td>
<td>450</td>
<td>Euf</td>
<td>authentication</td>
</tr>
<tr>
<td>SLK06</td>
<td>80</td>
<td>Euf</td>
<td>authentication</td>
</tr>
<tr>
<td>YPLRK05</td>
<td>160</td>
<td>Euf</td>
<td>authentication</td>
</tr>
</tbody>
</table>

→ between 80 and 450 LoC (for the model and the proof script).
Conclusion
Formal symbolic verification

Take away:

- the two main tools today are ProVerif and Tamarin;
- many success stories regarding reachability properties: they are able to analyse quite complex protocols and scenarios (mostly automatically)

Work in progress:

- allow some user interactions to help the prover;
  → available in Tamarin from the beginning, and now available also in ProVerif [Blanchet et al., S&P’22]
- some equivalence properties (e.g., unlinkability) are still challenging to analyse;
- each tool has its own specificities (syntax, semantics, own features, ...): a need for a platform to ease interactions
  → Sapić+ platform [Cheval et al., USENIX’22]
Future developments on Squirrel

Squirrel is a new tool and it remains a lot to do . . .

Some work in progress:

- more complex protocols;
- more powerful automation using e.g. SMT solvers; (PhD of S. Riou)
- study of the translation from processes to actions; (PhD of C. Hérouard)
- formally deriving crypto axioms / tactics from games; (PhD of J. Sauvage)
- analysing post-quantum or hybrid protocols
- . . .
PEPR Cybersecurity (2022-2028)
Partners: 5 teams in France (Nancy, Paris, Rennes, Sophia)


Job offers:
- PhDs
- Post-docs
- Engineers

→ contact me: stephanie.delaune@irisa.fr