Formal verification of security protocols - the Squirrel prover

Stéphanie DELAUNE

GdR SI - National Days

June 23, 2025

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.

BELENIOS

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.

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.

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 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]

Security

Defects in e-passports allow real-time tracking

This threat brought to you by RFID

The register - Jan. 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.

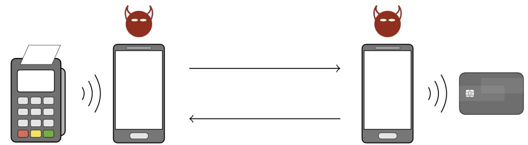
Contactless payment is vulnerable to relay attack

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

Publié: 2 août 2016, 18:19 CEST

The Conversation - Aug. 2016

How does it work ?



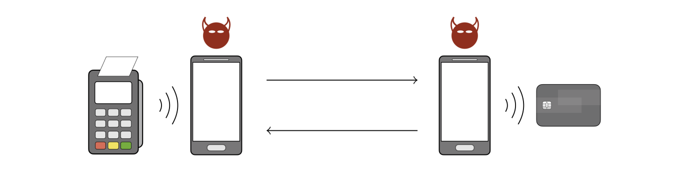
Contactless payment is vulnerable to relay attack

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

Publié: 2 août 2016, 18:19 CEST

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,
  $\hookrightarrow$ 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

$$A \rightarrow B : \{A, N_A\}_{\mathsf{pub}(B)}$$
$$B \rightarrow A : \{N_A, N_B\}_{\mathsf{pub}(A)}$$
$$A \rightarrow B : \{N_B\}_{\mathsf{pub}(B)}$$

NS protocol (1978)

$$A \rightarrow B : \{A, N_A\}_{\mathsf{pub}(B)}$$
$$B \rightarrow A : \{N_A, N_B, B\}_{\mathsf{pub}(A)}$$
$$A \rightarrow B : \{N_B\}_{\mathsf{pub}(B)}$$

NS-Lowe protocol (1995)

Two additional examples of logical attacks

An authentication flaw on the Needham Schroeder protocol

$$\begin{aligned} A \rightarrow B &: \{A, N_A\}_{\mathsf{pub}(B)} \\ B \rightarrow A &: \{N_A, N_B\}_{\mathsf{pub}(A)} \\ A \rightarrow B &: \{N_B\}_{\mathsf{pub}(B)} \end{aligned}$$

NS protocol (1978)

$$\begin{aligned} A \rightarrow B &: \{A, N_A\}_{\mathsf{pub}(B)} \\ B \rightarrow A &: \{N_A, N_B, B\}_{\mathsf{pub}(A)} \\ A \rightarrow B &: \{N_B\}_{\mathsf{pub}(B)} \end{aligned}$$

NS-Lowe protocol (1995)

Pairing confusion attacks : Tschirschnitz et al. (2021) / Claverie et al. (2023)

A logical flaw that allows a *man-in-the-middle* attacker to make two different versions of the protocol interact without the user noticing.

$\longrightarrow$ 5.4 billion Bluetooth devices shipped in 2023.

How to verify the absence of logical flaws ?

- dissect the protocol and test their resilience against well-known attacks ;
  $\longrightarrow$ **this is not sufficient !**

How to verify the absence of logical flaws ?

- dissect the protocol and test their resilience against well-known attacks ;
  $\longrightarrow$ **this is not sufficient !**

- perform a manual security analysis
  $\longrightarrow$ **this is error-prone !**

How to verify the absence of logical flaws ?

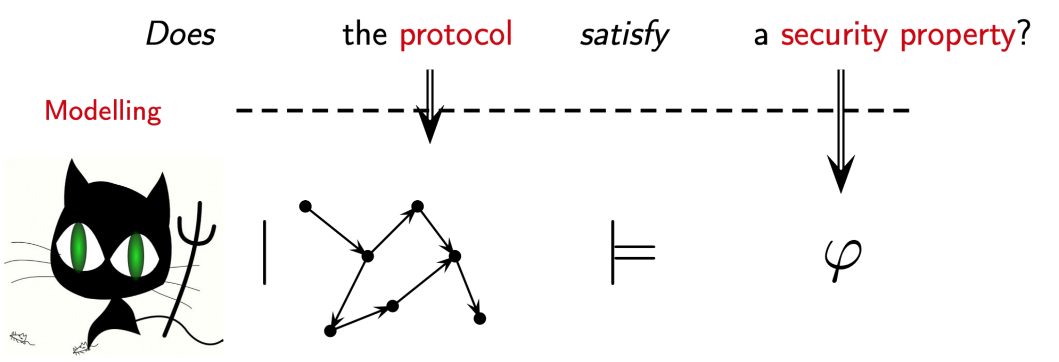
- dissect the protocol and test their resilience against well-known attacks ;
  $\longrightarrow$ **this is not sufficient !**

- perform a manual security analysis
  $\longrightarrow$ **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

Symbolic models
[Dolev & Yao, 81]

Computational models
[Goldwasser & Micali, 84]

Two main families of models

Symbolic models

[Dolev & Yao, 81]

Messages are terms.

What the attacker can do.

Unclear guarantees.

Amenable to automation.

e.g. Proverif, Tamarin

Computational models

[Goldwasser & Micali, 84]

Messages are bitstrings.

What the attacker can **not** do.
Everything else is allowed ![1]

Stronger guarantees.

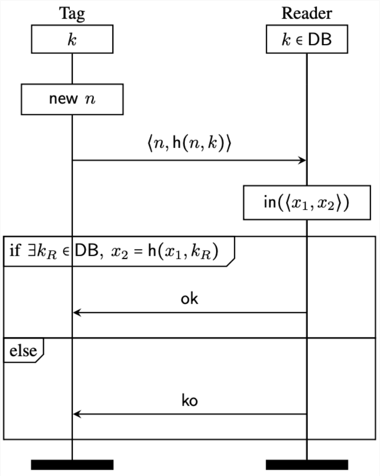
Harder to automate.

e.g. CryptoVerif

[1] The attacker is a probabilistic polynomial-time Turing machine.

Example : Basic Hash protocol

⟶ [Weis *et al.*, 03]



- 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.

Protocols as processes

$\longrightarrow$ a programming language with constructs for concurrency and communication

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

P, Q	$:=$	0	null process
	$\mid$	$\mathsf{in}(c, x); P$	input
	$\mid$	$\mathsf{out}(c, M); P$	output
	$\mid$	$\mathtt{new}\ n; P$	name generation
	$\mid$	$\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q$	conditional
	$\mid$	$!P$	replication
	$\mid$	$(P \mid Q)$	parallel composition

Protocols as processes

$\longrightarrow$ a programming language with constructs for concurrency and communication

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

P, Q	$:=$	0	null process
	$\mid$	$\mathsf{in}(c, x); P$	input
	$\mid$	$\mathsf{out}(c, M); P$	output
	$\mid$	$\mathtt{new}\ n; P$	name generation
	$\mid$	$\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q$	conditional
	$\mid$	$!P$	replication
	$\mid$	$(P \mid Q)$	parallel composition
	$\mid$	$\mathtt{insert}\ tbl(M); P$	insertion
	$\mid$	$\mathtt{get}\ tbl(x)\ \mathtt{st.}\ M = N\ \mathtt{in}\ P\ \mathsf{else}\ Q$	lookup
	$\mid$	$\ldots$	

Basic Hash protocol in the symbolic setting

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

Modelling messages/computations

$$\begin{array}{rcl} \Sigma & = & \{\langle\ \rangle,\ \mathsf{proj}_1,\ \mathsf{proj}_2,\ \mathsf{h}\} \\ \mathsf{E} & = & \{\mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1,\ \ \mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2\} \end{array}$$

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

Modelling protocols as processes

$$!\,\mathsf{R} \mid (!\,\mathtt{new}\ k; \mathtt{insert}\ DB(k);\ !\,\mathsf{T}(k))$$

where :

- $\mathsf{T}(k) = \mathtt{new}\ n; \mathsf{out}(c, \langle n, \mathsf{h}(n, k)\rangle)$.
- $\mathsf{R} =$
  $\mathsf{in}(c, y); \mathsf{get}\ DB(k)\ \mathtt{st.}\ \mathsf{h}(\mathsf{proj}_1(y), k) = \mathsf{proj}_2(y)\ \mathtt{in}\ \mathsf{out}(c, \mathsf{ok})\ \mathsf{else}\ \mathsf{out}(c, \mathsf{ko})$.

Basic Hash in the computational setting

$\longrightarrow$ 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) : « $\mathsf{h}(n, k) = \mathsf{h}(n', k)$ implies $n = n'$ »
- PseudoRandom Function (PRF) : « $\mathsf{h}(n, k) \sim r$ »
- 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 $\mathcal{A}$ (i.e. any PPTM) :

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

Basic Hash in the computational setting : authentication

Existential UnForgeability (EUF)

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

- Draw k uniformly at random.
- $\langle u, v \rangle := \mathcal{A}^{\mathcal{O}}$ where $\mathcal{O}$ is the oracle $x \to \mathsf{h}(x, k)$.
- Succeed if $u = \mathsf{h}(v, k)$ and $\mathcal{O}$ 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 $\mathsf{proj}_2(m) = \mathsf{h}(\mathsf{proj}_1(m), k_i)$ for some i .
- By unforgeability, $\mathsf{proj}_1(m) = n_T$ for some session of tag T_i .
- The two projections of m are the two projections of the output of T_i. $\square$

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 exising verification tools

	DeepSec/Akiss	ProVerif/GSverif	Tamarin	DY⋆	**Squirrel**	CryptoVerif	EasyCrypt
unbounded traces	✗	✓	✓	✓	✓	✓	✓
computational attacker	✗	✗	✗	✗	✓	✓	✓
concrete security bounds	✗	✗	✗	✗	✗	✓	✓
native concurrency	✓	✓	✓	✓	✓	✓	✗
global mutable states	✓	✓	✓	✓	✓	✗	✓
automation	↑	↗	↗	↘	↘	↗	↓

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 veryfing cryptographic protocols in the computational model.

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

It is based on the CCSA approach :

G. Bana & H. Comon. CCS 2014.
A Computationally Complete Symbolic Attacker for Equivalence Properties.

History of 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.
- 2023 : A careful re-design of the logic behind Squirrel — Baelde et al.

On the practical side

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

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

⟶ Recommended browsers : Firefox or Chrome.

Current team : members of Inspire (LMF), Pesto (Inria Nancy), Prosecco (Inria Paris), and Spicy (IRISA).

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, . . .

$\longrightarrow$ 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

```
include Basic.

hash h
abstract ok : message
abstract ko : message.

name key  : index -> message

channel cT
channel cR.

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

process reader(j:index) =
  in(cT,x);
  if exists (i:index), snd(x) = h(fst(x),key(i)) then R1: out(cR,ok)
  else R2: out(cR,ko).

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

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@$T[i,k] = \langle n_T[i,k], \mathrm{h}(n_T[i,k], key[i]) \rangle$.

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

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

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

- input@$R_1[j]$;
- $\exists i. snd(\text{input@}R_1[j]) = h(fst(\text{input@}R_1[j]), key[i])$;
- output@$R_1[j] = ok$;
- input@$R_2[j]$;
- $\forall i. snd(\text{input@}R_2[j]) \neq h(fst(\text{input@}R_2[j]), key[i])$;
- output@$R_2[j] = ko$.

Basic Hash - Authentication

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)

```
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 and also a crypto tactic (here euf).

$\longrightarrow$ 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:

$$\frac{}{\Gamma, t = \mathsf{n} \vdash \phi} \quad \text{where } \mathsf{n} \notin \mathsf{st}(t)$$

Meta-logic rule:

$$\frac{\Gamma, \bigvee_{(\mathsf{n}[\vec{j}],\vec{k},c)\in \bar{\mathsf{st}}_{\mathcal{P}}(t)} \exists \vec{k}.c \wedge \vec{i} = \vec{j} \vdash \phi}{\Gamma, t = \mathsf{n}[\vec{i}] \vdash \phi}$$

freshness of a name *n*

Squirrel offline Demo - authentication on Basic Hash

`https://squirrel-prover.github.io/jsquirrel/` (file basic-hash-auth.sp)

basic-hash-auth.sp

```
  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 [HOrd 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

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)

basic-hash-auth.sp

```
  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 [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

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)

basic-hash-auth.sp

```
  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 [HOrd Eq]].
by exists i0, 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

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)

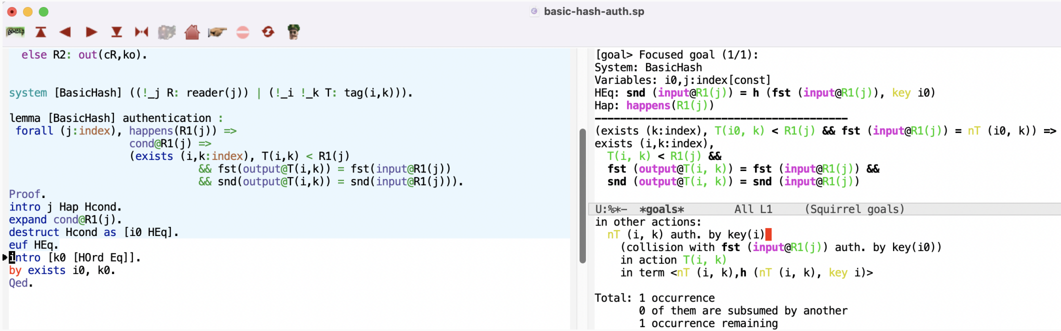
basic-hash-auth.sp

```
  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 [HOrd Eq]].
by exists i0, k0.
Qed.
```

```
[goal> Focused 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 (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

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)



Squirrel offline Demo - authentication on Basic Hash

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)

basic-hash-auth.sp

```
  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 [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)) = nT (i0, k0)
HEq: snd (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

https://squirrel-prover.github.io/jsquirrel/ (file basic-hash-auth.sp)

basic-hash-auth.sp

```
  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 [HOrd Eq]].
by exists i0, k0.
Qed.
```

```
[goal> lemma authentication is proved
```

```
U:%*-  *goals*      All L1      (Squir
```

Benchmark

Protocol name	LoC	Assumptions	Security Properties
Basic Hash	60	Prf, Euf	authentication & unlinkability
Hash Lock	130	Prf, Euf	authentication & unlinkability
LAK (with pairs)	250	Prf, Euf	authentication & unlinkability
MW	300	Prf, Euf, Xor	authentication & unlinkability
Feldhofer	270	Enc-Kp, Int-Ctxt	authentication & unlinkability
Private authentication	100	Cca_1, Enc-Kp	anonymity
Signed DDH [ISO 9798-3]	240	Euf, Ddh	authentication & strong secrecy
CANAuth	450	Euf	authentication
SLK06	80	Euf	authentication
YPLRK05	160	Euf	authentication

$\longrightarrow$ between 60 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 :

- some equivalence properties (e.g. unlinkability) are still challenging to analyse ;
- some equational theories (e.g. AC operators) are still challenging to deal with ;
- each tool has its own specificities (syntax, semantics, own features, . . .) : a need for a platform to ease interactions
  $\longrightarrow$ Sapic$^+$ platform [Cheval *et al.*, USENIX'22]

Ongoing developments on Squirrel

It remains a lot to do to handle more complex protocols

- more automation : SMT solvers (PhD of S. Riou - CSF'25), typing (PhD of C. Hérouard - CSF'25) ;
- formally deriving tactics from crypto games (PhD of J. Sauvage - CCS'24) ;
- soudness of the translation from processes to actions (PhD of C. Hérouard) ;
- concrete security (PhD of T. Vignon - CSF'24) ;
- analysing post-quantum or hybrid protocols ;
- ...