Cryptographic protocols everywhere - Are we well protected?

Stéphanie DELAUNE

Colloquium Polaris - March 16, 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, . . .)

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.

BELENIOS

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:

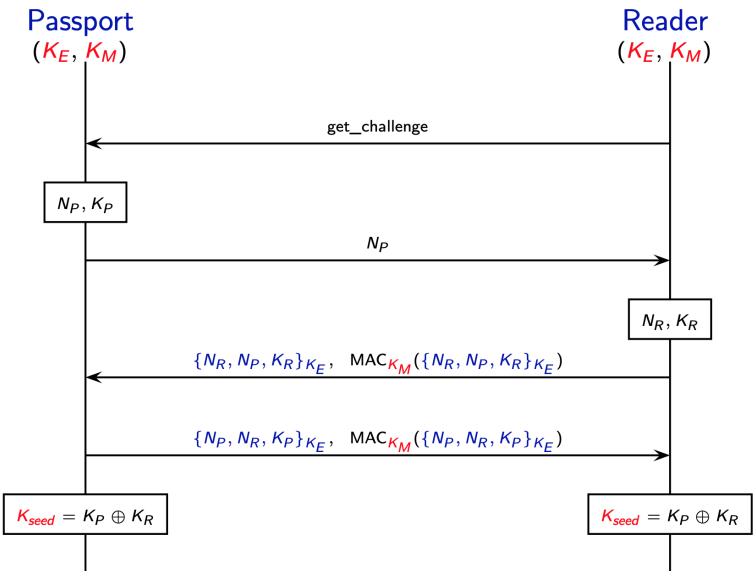
- the information printed on your passport,
- a JPEG copy of your picture.

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.*

Basic Acccess Control protocol



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.

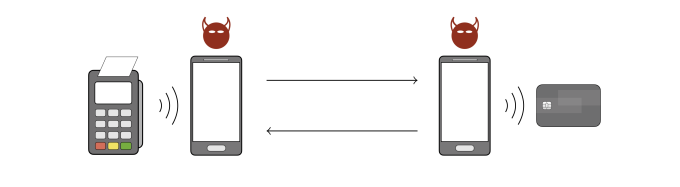
Contactless payment is vulnerable to relay attack

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?



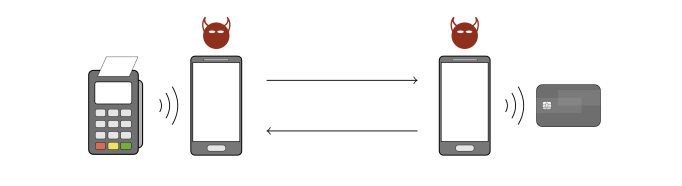
Contactless payment is vulnerable to relay attack

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?



$\longrightarrow$ 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.

Logical attacks

- can be mounted even assuming perfect cryptography,
  $\hookrightarrow$ replay attack, man-in-the middle attack, ...
- subtle and hard to detect by "eyeballing" the protocol

Denning Sacco protocol (1981)

$\longrightarrow$ simplified version

$$\xrightarrow{\quad \mathsf{aenc}(\mathsf{sign}(k_{AB}, \mathsf{priv}(A)), \mathsf{pub}(B)) \quad}$$

Is this protocol a good key establishment protocol?

Denning Sacco protocol (1981)

$\longrightarrow$ simplified version

$$\xrightarrow{\quad \mathsf{aenc}(\mathsf{sign}(k_{AB}, \mathsf{priv}(A)), \mathsf{pub}(B)) \quad}$$

Is this protocol a good key establishment protocol? No!

Denning Sacco protocol (1981)

$\longrightarrow$ simplified version

$$\xrightarrow{\mathsf{aenc}(\mathsf{sign}(k_{AB}, \mathsf{priv}(A)), \mathsf{pub}(B))}$$

Is this protocol a good key establishment protocol? No!

Description of the attack:

$$\xrightarrow{\mathsf{aenc}(\mathsf{sign}(k_{AC}, \mathsf{priv}(A)), \mathsf{pub}(C))}$$

Denning Sacco protocol (1981)

$\longrightarrow$ simplified version

$$\xrightarrow{\mathsf{aenc}(\mathsf{sign}(k_{AB}, \mathsf{priv}(A)), \mathsf{pub}(B))}$$

Is this protocol a good key establishment protocol? No!

Description of the attack:

$$\xrightarrow{\mathsf{aenc}(\mathsf{sign}(k_{AC}, \mathsf{priv}(A)), \mathsf{pub}(C))}$$

$\mathsf{sign}(k_{AC}, \mathsf{priv}(A))$

k_{AC}

$$\xrightarrow{\mathsf{aenc}(\mathsf{sign}(k_{AC}, \mathsf{priv}(A)), \mathsf{pub}(B))}$$

And now, play on . . .

We propose the following variants of the Denning-Sacco protocol.

Version A

$$A \rightarrow B \quad : \quad \mathsf{aenc}(\langle A, B, \mathsf{sign}(k, \mathsf{priv}(A))\rangle, \mathsf{pub}(B))$$

Version B

$$A \rightarrow B \quad : \quad \mathsf{aenc}(\mathsf{sign}(\langle A, B, k\rangle, \mathsf{priv}(A))\rangle, \mathsf{pub}(B))$$

Which version do you prefer to use?

And now, play on . . .

We propose the following variants of the Denning-Sacco protocol.

Version A

$$A \rightarrow B \quad : \quad \mathsf{aenc}(\langle A, B, \mathsf{sign}(k, \mathsf{priv}(A))\rangle, \mathsf{pub}(B))$$

Version B

$$A \rightarrow B \quad : \quad \mathsf{aenc}(\mathsf{sign}(\langle A, B, k\rangle, \mathsf{priv}(A))\rangle, \mathsf{pub}(B))$$

Which version do you prefer to use? Version B

$\longrightarrow$ Version A is still vulnerable to the aforementioned attack.

Some additional examples

An authentication flaw on the Needham Schroeder protocol

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

NS protocol (1978)

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

NS-Lowe protocol (1995)

Some additional examples

An authentication flaw on the Needham Schroeder protocol

$$A \to B : \{A, N_A\}_{\mathrm{pub}(B)}$$
$$B \to A : \{N_A, N_B\}_{\mathrm{pub}(A)}$$
$$A \to B : \{N_B\}_{\mathrm{pub}(B)}$$

NS protocol (1978)

$$A \to B : \{A, N_A\}_{\mathrm{pub}(B)}$$
$$B \to A : \{N_A, N_B, B\}_{\mathrm{pub}(A)}$$
$$A \to B : \{N_B\}_{\mathrm{pub}(B)}$$

NS-Lowe protocol (1995)

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.

$\longrightarrow$ websites affected by the vulnerability included those from the US federal government

Research in the Spicy team - IRISA/Rennes

Security and PrIvaCY

- Cryptographic protocols
- Privacy
- Formal methods for security

7 permanent researchers + 2 post-docs/engineers + 10 PhD students





G. Avoine M. Sabt T. Allard B. Fila J. Lallemand D. Baelde S. Delaune

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?

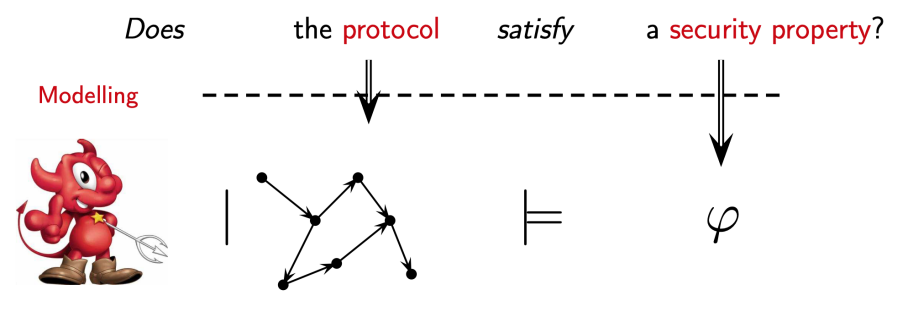
- dissect the protocol and test their resilience against well-known attacks;
  $\longrightarrow$ **this is not sufficient !**

- 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



- Modelling the attacker, the protocols, and their security properties
- State-of-the art on symbolic protocol verification
- A novel approach: the Squirrel prover

Part I

Modelling the attacker, the protocols, and their security properties

Two main families of models

Symbolic models

[Dolev & Yao, 81]

Computational models

[Goldwasser & Micali, 84]

Two main families of models

Symbolic models	Computational models
[Dolev & Yao, 81]	[Goldwasser & Micali, 84]
Messages are terms.	Messages are bitstrings.
What the attacker can do.	What the attacker can **not** do. Everything else is allowed![1]
Weaker guarantees.	Stronger guarantees.
Amenable to automation.	Harder to automate.

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

Two main families of models

Symbolic models	Computational models
[Dolev & Yao, 81]	[Goldwasser & Micali, 84]
Messages are terms.	Messages are bitstrings.
What the attacker can do.	What the attacker can **not** do. Everything else is allowed![1]
Weaker guarantees.	Stronger guarantees.
Amenable to automation.	Harder to automate.

$\longrightarrow$ In this talk, we will mostly focus on **symbolic models**.

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

Messages/Computations as terms

Terms are built over a set of names $\mathcal{N}$, and function symbols Σ equipped with an equational theory E.

Messages/Computations as terms

Terms are built over a set of names $\mathcal{N}$, and function symbols Σ equipped with an equational theory E.

Example:

$$\Sigma = \{\mathsf{senc}/2,\ \mathsf{sdec}/2,\ \mathsf{aenc}/2,\ \mathsf{adec}/2,\ \mathsf{pk}/1,\ \mathsf{sign}/2,\ \mathsf{getmsg}/1,\ \mathsf{checksign}/2,\ \mathsf{ok}/0\}$$

$$\mathsf{E} = \left\{ \begin{array}{rcl} \mathsf{sdec}(\mathsf{senc}(x,y),y) & = & x \\ \mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) & = & x \end{array} \quad \begin{array}{rcl} \mathsf{getmsg}(\mathsf{sign}(x,y)) & = & x \\ \mathsf{checksign}(\mathsf{sign}(x,y),\mathsf{vk}(y)) & = & \mathsf{ok} \end{array} \right.$$

Messages/Computations as terms

Terms are built over a set of names $\mathcal{N}$, and function symbols Σ equipped with an equational theory E.

Example:

$$\Sigma = \{\mathsf{senc}/2,\ \mathsf{sdec}/2,\ \mathsf{aenc}/2,\ \mathsf{adec}/2,\ \mathsf{pk}/1,\ \mathsf{sign}/2,\ \mathsf{getmsg}/1,\ \mathsf{checksign}/2,\ \mathsf{ok}/0\}$$

$$\mathsf{E} = \left\{ \begin{array}{rcl} \mathsf{sdec}(\mathsf{senc}(x,y),y) & = & x \\ \mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) & = & x \end{array} \quad \begin{array}{rcl} \mathsf{getmsg}(\mathsf{sign}(x,y)) & = & x \\ \mathsf{checksign}(\mathsf{sign}(x,y),\mathsf{vk}(y)) & = & \mathsf{ok} \end{array} \right.$$

Let $\Phi = \{\mathsf{w} \mapsto \mathsf{aenc}(\mathsf{sign}(k, ska), \mathsf{pk}(skc))\}$, and $R = \mathsf{getmsg}(\mathsf{adec}(\mathsf{w}, skc))$. We have that

$$\begin{array}{rcl} R\Phi & = & \mathsf{getmsg}(\mathsf{adec}(\mathsf{aenc}(\mathsf{sign}(k, ska), \mathsf{pk}(skc)), skc)) \\ & =_{\mathsf{E}} & \mathsf{getmsg}(\mathsf{sign}(k, ska)) \\ & =_{\mathsf{E}} & k \end{array}$$

Protocols as processes

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

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

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

Protocols as processes

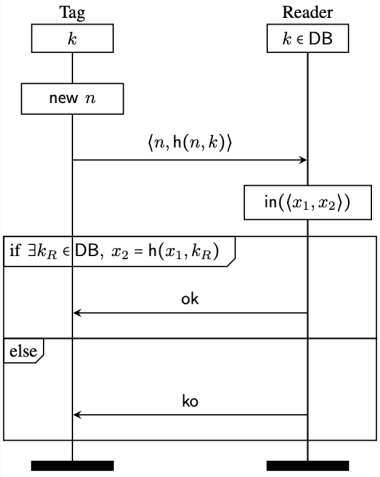
$\longrightarrow$ a programming language with constructs for concurrency and communication

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

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

Example: Basic Hash protocol

$\longrightarrow$ [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.

$\longrightarrow$ authentication, unlinkability

Modelling the Basic Hash protocol

Modelling terms

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

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

Modelling the Basic Hash protocol

Modelling terms

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

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

Modelling the whole system

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

where:

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

Semantics (some selected rules)

Labelled transition system over configurations: $(\mathcal{P}; \Phi; \mathcal{S})$

multiset of processes

frame
knowledge of the attacker

store
content of the database

Semantics (some selected rules)

Labelled transition system over configurations: $(\mathcal{P}; \Phi; \mathcal{S})$

multiset of processes

frame
knowledge of the attacker

store
content of the database

Out $\quad (\{\mathsf{out}(c, M); P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathsf{out}(c, \mathsf{w}_i)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{\mathsf{w}_i \mapsto M\}; \mathcal{S})$ with $i = |\Phi|$

In $\quad (\{\mathsf{in}(c, x); P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathsf{in}(c, R)} (\{P\{x \mapsto R\Phi\}\} \uplus \mathcal{P}; \Phi; \mathcal{S})$

Then $\quad (\{\texttt{if}\ \ M = N\ \texttt{then}\ \ P\ \texttt{else}\ \ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\tau} (\{P\} \uplus \mathcal{P}; \Phi; \mathcal{S})$ with $M =_{\mathsf{E}} N$

... $\quad$ ...

Semantics (some selected rules)

Labelled transition system over configurations: $(\mathcal{P}; \Phi; \mathcal{S})$

multiset of processes

frame
knowledge of the attacker

store
content of the database

Out $\quad (\{\mathsf{out}(c, M); P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathsf{out}(c,\mathsf{w}_i)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{\mathsf{w}_i \mapsto M\}; \mathcal{S})$ with $i = |\Phi|$

In $\quad (\{\mathsf{in}(c, x); P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathsf{in}(c,R)} (\{P\{x \mapsto R\Phi\}\} \uplus \mathcal{P}; \Phi; \mathcal{S})$

Then $\quad (\{\texttt{if}\ \ M = N\ \texttt{then}\ \ P\ \texttt{else}\ \ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\tau} (\{P\} \uplus \mathcal{P}; \Phi; \mathcal{S})$ with $M =_{\mathsf{E}} N$

... $\quad$ ...

$\longrightarrow$ traces(K) = the set of execution traces starting from the configuration K.

What does secure mean?

Verifiability

Confidentiality

Receipt-freeness

Unlinkability

Authentication

Fairness

Anonymity

What does secure mean?

Verifiability

Confidentiality

Receipt-freeness

Unlinkability

Authentication

Fairness

Anonymity

What does this mean exactly?

$\longrightarrow$ formal definitions do not always exist!

Two main families of security properties

- Reachability properties, *e.g.* weak secrecy, authentication, ...

  Is it possible to reach a bad state, e.g. a state in which the secret would be deducible?

- Equivalence properties, *e.g.* anonymity, unlinkability, strong form of secrecy, ...

  Is the attacker able to distinguish between two situations?

$\overset{?}{\approx}$

$\longrightarrow$ different notions of equivalence actually exist.

Trace equivalence

Trace equivalence between configurations: $K \approx_t K'$.

For any execution trace $K \xrightarrow{\text{tr}} (\mathcal{P}; \Phi; \mathcal{S})$ there exists an execution $K' \xrightarrow{\text{tr}} (\mathcal{P}'; \Phi'; \mathcal{S}')$ such that $\Phi \sim_s \Phi'$ (and conversely)

Trace equivalence

Trace equivalence between configurations: $K \approx_t K'$.

For any execution trace $K \xrightarrow{\text{tr}} (\mathcal{P}; \Phi; \mathcal{S})$ there exists an execution $K' \xrightarrow{\text{tr}} (\mathcal{P}'; \Phi'; \mathcal{S}')$ such that $\Phi \sim_s \Phi'$ (and conversely)

Static equivalence between frames: $\Phi \sim_s \Phi'$.

Any test that holds in Φ also holds in Φ' (and conversely).

Example:

$\{ w_1 \mapsto k;\ w_2 \mapsto \langle n, \mathrm{h}(n, k)\rangle\} \not\sim_s \{w_1 \mapsto k;\ w_2 \mapsto \langle n', \mathrm{h}(n', k')\rangle\}$

$\longrightarrow$ with the test $\mathrm{h}(\mathrm{proj}_1(w_2), w_1) \stackrel{?}{=} \mathrm{proj}_2(w_2)$.

Trace equivalence

Trace equivalence between configurations: $K \approx_t K'$.

For any execution trace $K \xrightarrow{\text{tr}} (\mathcal{P}; \Phi; \mathcal{S})$ there exists an execution $K' \xrightarrow{\text{tr}} (\mathcal{P}'; \Phi'; \mathcal{S}')$ such that $\Phi \sim_s \Phi'$ (and conversely)

Static equivalence between frames: $\Phi \sim_s \Phi'$.

Any test that holds in Φ also holds in Φ' (and conversely).

Example:

$\{ w_1 \mapsto k;\ w_2 \mapsto \langle n, \mathsf{h}(n, k)\rangle\} \not\sim_s \{w_1 \mapsto k;\ w_2 \mapsto \langle n', \mathsf{h}(n', k')\rangle\}$

$\longrightarrow$ with the test $\mathsf{h}(\mathsf{proj}_1(w_2), w_1) \stackrel{?}{=} \mathsf{proj}_2(w_2)$.

$\{ w_1 \mapsto k;\ w_2 \mapsto \langle n, \mathsf{h}(n, k)\rangle\} \sim_s \{w_1 \mapsto k;\ w_2 \mapsto \langle n', \mathsf{h}(n', k)\rangle\}$

$\longrightarrow$ they are in static equivalence.

Going back to Basic Hash

Authentication

- add events in the process to keep track of some actions, e.g. $\mathtt{ReaderAccept}(\langle n, \mathrm{h}(n, k)\rangle)$, and $\mathtt{TagStart}(\langle n, \mathrm{h}(n, k)\rangle)$;
- events do not interfere in the semantics: they are simply append in the trace tr during execution;
- $\varphi_A = \forall x.\, \mathtt{ReaderAccept}(x) \Rightarrow \mathtt{TagStart}(x)$.

Authentication holds on configuration K if for all $\mathrm{tr} \in \mathrm{traces}(K)$, we have that $\mathrm{tr} \models \varphi_A$

Going back to Basic Hash

Authentication

- add events in the process to keep track of some actions, e.g. $\mathtt{ReaderAccept}(\langle n, \mathsf{h}(n,k)\rangle)$, and $\mathtt{TagStart}(\langle n, \mathsf{h}(n,k)\rangle)$;
- events do not interfere in the semantics: they are simply append in the trace tr during execution;
- $\varphi_A = \forall x.\, \mathtt{ReaderAccept}(x) \Rightarrow \mathtt{TagStart}(x)$.

Authentication holds on configuration K if for all $\mathsf{tr} \in \mathsf{traces}(K)$, we have that $\mathsf{tr} \models \varphi_A$

Unlinkability

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

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

Part II

State of the art on symbolic protocol verification

Reachability properties: some theoretical results

Unbounded number of sessions

- undecidable in general [Even & Goldreich, 83; Durgin *et al*, 99]
- decidable for restricted classes [Lowe, 99; Rammanujam & Suresh, 03]

Reachability properties: some theoretical results

Unbounded number of sessions

- undecidable in general [Even & Goldreich, 83; Durgin *et al*, 99]
- decidable for restricted classes [Lowe, 99; Rammanujam & Suresh, 03]

Bounded number of sessions

- a decidability result (NP-complete) [Rusinowitch & Turuani, 01; Millen & Shmatikov, 01]
- result extended to deal with various cryptographic primitives.

$\longrightarrow$ several automatic tools have been developed available through e.g. the AVISPA platform [Armando *et al.*, 05 & 12]

Reachability properties: more pragmatic approaches

ProVerif tool: [Blanchet *et al.*, 01]

$\longrightarrow$ fully automatic, quite efficient, but it may not terminate or return cannot be proved

Tamarin prover: [Meier *et al.*, 13]

$\longrightarrow$ suitable to model protocols with mutable states, various primitives (e.g. modular exponentiation, exclusive-or, ...), it may fail to establish automatically a given property (an interactive mode is available)

Sapic$^+$ platform: [Cheval et al., 22]

$\longrightarrow$ a protocol verification platform (with a well-defined semantics) which allows one to use ProVerif, Tamarin, (and also DeepSec) on the same input file.

Reachability properties: some success stories

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

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

A comprehensive, formal and automated analysis of the EDHOC protocol [Jacomme *et al*, 23]

Equivalence for a bounded number of sessions (a long story)

2005-2011: theoretical algorithms with **no implementation**

2010-2015: "practical" algorithms and tools (e.g. SPEC, APTE, Akiss, ...) but they scale badly

2018: The DeepSec tool [Cheval, Kremer & Rakotonirina, 2018]

- large class of processes: else branches, standard cryptographic primitives and beyond (e.g. blind signatures)
- quite efficient: exploit multicore architectures, integrate POR optimisations

Protocol		APTE	Deepsec
BAC (4 roles)	attack	38 min	1s
BAC (6 roles)		time out	time out

Intel Xeon 3.10GHz cores, with 50Go of memory – **35 cores** – time out = 12h

What about ProVerif and Tamarin?

$\longrightarrow$ They have been extended to deal with equivalence-based properties.

ProVerif tool: [Blanchet et al., LICS'05]

fully automatic, quite efficient, but it may not terminate or return cannot be proved.

Tamarin prover: [Basin et al., CCS'15]

suitable to model protocols with mutable states, various primitives, it may fail to establish a given property (an interactive mode is available)

What about ProVerif and Tamarin?

$\longrightarrow$ They have been extended to deal with equivalence-based properties.

ProVerif tool: [Blanchet et al., LICS'05]

fully automatic, quite efficient, but it may not terminate or return cannot be proved.

Tamarin prover: [Basin et al., CCS'15]

suitable to model protocols with mutable states, various primitives, it may fail to establish a given property (an interactive mode is available)

Main limitation:

They both consider a strong form of equivalence, namely diff-equivalence which is well-suited for analysing e.g. strong secrecy but too limitating for analysing e.g. vote-privacy, and unlinkability.

Going back to Basic Hash

Authentication: Both ProVerif and Tamarin are able to establish this property.

Unlinkability: ProVerif returns cannot be proved, and Tamarin returns a false attack trace (diff-equivalence does not hold on this example).

A very recent result **[Baelde *et al.*, CSF'23]**

A simple transformation $\mathcal{T}$ that can be applied on some stateful 2-party protocols such that:

- unlinkability holds on P if, and only if, unlinkability holds on $\mathcal{T}(P)$;
- ProVerif succeeds on $\mathcal{T}(P)$.

$\longrightarrow$ allows us to conclude that unlinkability holds for Basic Hash, and also on some other examples.

Part III

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.

Current team

David Baelde, Stéphanie Delaune, Caroline Fontaine, Clément Hérouard, Charlie Jacomme, Adrien Koutsos, Joseph Lallemand, Thomas Rubiano, Justine Sauvage, ...

$\longrightarrow$ all these people are at IRISA, LMF, and Inria Paris.

Computational assumptions

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

Example: Some usual cryptographic assumptions for a hash function are Collision Resistance (CR), PseudoRandom Function (PRF), Existential UnForgeability (EUF), ...

Computational assumptions

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

Example: Some usual cryptographic assumptions for a hash function are Collision Resistance (CR), PseudoRandom Function (PRF), Existential UnForgeability (EUF), ...

Existential UnForgeability (EUF)

There is a negligible probability of success for the following security 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 $x \to \mathrm{h}(x, k)$.
- Succeed if $u = \mathrm{h}(v, k)$ and $\mathcal{O}$ has *not* been called on v.

Two different approaches to obtain computational guarantees

Computational soundness

Some results show that symbolic attackers account for all computational attacks. [Abadi & Rogaway, 00]

$\longrightarrow$ those results remain limited due too strong assumptions, e.g. no sound abstraction of exclusive-or, we need a way to reflect the different assumptions regarding hash functions, ...

Direct verification in the computational model

CryptoVerif [Blanchet, S&P'06], EasyCrypt [Barthe, CRYPTO'11], and Squirrel [Baelde *et al.*, S&P 21].

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);
- reachability and equivalence properties/goals.

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.

Squirrel Demo - authentication on Basic Hash

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
  euf Meq.
  exists i, k0.
Qed.
```

-:--- basic-hash.sp All L33 (squirrel script Scripting)

```
Un-processed system:

  (!_j R: reader j) | (!_i !_k T: tag i k)

Processed system:

  (!_j
     in(cT,x);
     if exists (i,k:index), snd(x) = h(fst(x),key(i)) then
       R: out(cR,ok); null else R1: out(cR,ko); null) |
  (!_i !_k T: out(cT,pair(nT(i,k),h(nT(i,k),key(i)))); null)

System default registered with actions (init,R,R1,T).
```

U:%*- *response* All L13 (squirrel response)

Use C-c C-. to jump to end of processed region

Squirrel Demo - authentication on Basic Hash

basic-hash.sp

Retract (undo) whole buffer

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
  euf Meq.
  exists i, k0.
Qed.
```

```
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))))))
```

-:--- basic-hash.sp All L33 (squirrel script Scripting) U:%*- *response* All L7 (squirrel response)

tool-bar next

Squirrel Demo - authentication on Basic Hash

basic-hash.sp

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
 euf Meq.
 exists i, k0.
Qed.
```

```
[goal> Focused goal (1/1):
System: default/both
----------------------------------------
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)))))
```

-:--- basic-hash.sp All L34 (squirrel script Scripting) | U:%*- *goals* All L1 (squirrel goals)

tool-bar next

Squirrel Demo - authentication on Basic Hash

basic-hash.sp

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
  euf Meq.
  exists i, k0.
Qed.
```

-:--- basic-hash.sp All L35 (squirrel script Scripting)

```
[goal> Focused goal (1/1):
System: default/both
Variables: j:index
H: cond@R(j)
Hap: happens(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)))
```

U:%*- *goals* All L1 (squirrel goals)

tool-bar next

Squirrel Demo - authentication on Basic Hash

basic-hash.sp

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
▶ euf Meq.
  exists i, k0.
Qed.
```

-:--- basic-hash.sp All L36 (squirrel script Scripting)

```
[goal> Focused goal (1/1):
System: default/both
Variables: i,j,k:index
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:%*- *goals* All L1 (squirrel goals)

tool-bar next

Squirrel Demo - authentication on Basic Hash

basic-hash.sp

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
  euf Meq.
  exists i, k0.
Qed.
```

```
[goal> Focused goal (1/1):
System: default/both
Variables: i,j,k,k0:index
Clt: T(i,k0) < R(j)
Hap: happens(R(j))
Meq: snd(input@R(j)) = h(fst(input@R(j)),key(i))
Meq0: nT(i,k0) = fst(input@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)))
```

-:--- basic-hash.sp All L37 (squirrel script Scripting) | U:%*- *goals* All L1 (squirrel goals)

tool-bar next

Squirrel Demo - authentication on Basic Hash

basic-hash.sp

```
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,k:index), snd(x) = h(fst(x),key(i)) then
    out(cR,ok)
  else
    out(cR,ko)

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

(* 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@R(j).
  euf Meq.
  exists i, k0.
Qed.
```

```
[goal> Goal wa_R is proved
```

-:--- basic-hash.sp All L38 (squirrel script Scripting) U:%%- *goals* All L1 (squirrel goals)

tool-bar next

Squirrel Demo - unlinkability on Basic Hash

mon-basic-hash.sp

```
equiv [BasicHash] unlinkability.

Proof.
  induction t; 1: auto.
  + expand frame, exec, output. fa !<_,_>.
    rewrite /cond (wa_R (R j)) //.
    by fadup 1.

  + expand frame, exec, output. fa !<_,_>.
    rewrite /cond (wa_R (R1 j)) //.
    by fadup 1.

  + expand frame, exec, cond, output.
    fa !<_,_>, if _ then _, <_,_>.
    prf 2. rewrite if_true. {
      split; 1: true.
      project; repeat split; intro *; by fresh Meq.
    }.
    fresh 2; 1:auto.
    by fresh 1.
Qed.
```

```
[goal> Focused goal (1/1):
Systems: left:BasicHash/left, right:BasicHash/right (same for equivalences)
Variables: t:timestamp[const, glob]
H: [happens(t)]
------------------------------------------
0: frame@t
```

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;
  $\longrightarrow$ available in Tamarin from the beginning, and now available also in ProVerif [Blanchet *et al.*, S&P'22]
- some equivalence properties are still challenging to analyse;
- 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]

Future development on Squirrel

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

Some work in progress:

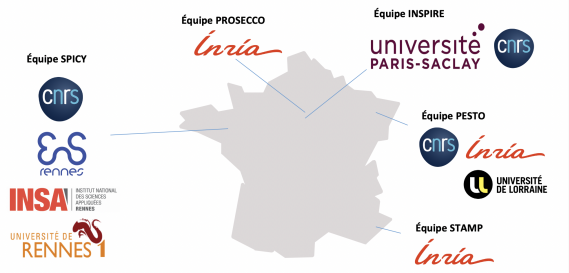
- more complex protocols;
- more powerful automation using e.g. SMT solvers;
- study of translation from pi-calculus processes to systems of actions;
- formally deriving crypto axioms / tactics from games;
- analysing post-quantum or hybrid protocols
- . . .

Advertisement

PEPR Cybersecurity (2022-2028)

Partners: 5 teams in France (Nancy, Paris, Rennes, Sophia)

https://pepr-cyber-svp.cnrs.fr



Job offers:

- PhDs
- Post-docs
- Engineers

⟶ contact me: stephanie.delaune@irisa.fr