Automated verification of privacy-type properties for security protocols

Ivan Gazeau

LORIA, INRIA

March 16th, 2018
Security protocols

Cryptographic protocols are intensively used on many contexts:

- https
- electronic passport
- contact less credit card
- electronic voting
Threats

Every year, newspapers report attacks on these protocols. We can distinguish between four kinds of attacks:

- **Attack on cryptographic primitives**
  - SHA-1

- **Logical attacks on the specification of protocols**
  - BAC on French passports, Crack on WPA2 Wifi

- **Attacks on the implementation of the protocols**
  - Heart blood

- **Side channel attacks**
  - Spectre
Protocol specification example

\[
\langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle
\]

\[
(id \oplus r_2) \oplus (h(\langle r_1, k \rangle) \oplus r_2) \oplus id
\]

\[
\？= h(\langle r_1, k \rangle)
\]
Security properties

We distinguish between two kinds of security properties:

- Trace properties (predicates on system behavior)
  - (weak) secrecy of a key
  - authentication (correspondence properties)
- Equivalence properties (involving two systems)
  - Unlinkability
  - Offline Guessing-attack
  - Non interference

We focus on equivalence properties.

Unlinkability
Equivalence property

**Testing equivalence** \((P \approx Q)\)
for all processes \(A\), we have that:

\[ A \parallel P \downarrow c \text{ if, and only if, } A \parallel Q \downarrow c \]

\(P \downarrow c\) when \(P\) can send a message on the channel \(c\).
Authentication protocol of a RFID tag (KCL)

\[
\text{READER} \\
\begin{array}{l}
k, id \\
\text{new } r_1
\end{array} \\
\begin{array}{l}
r_1
\end{array} \\
\text{TAG} \\
\begin{array}{l}
k, id \\
\text{new } r_2
\end{array} \\
\langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle
\]

\[
(id \oplus r_2) \oplus (h(\langle r_1, k \rangle) \oplus r_2) \oplus id \\
\overset{?}{=} h(\langle r_1, k \rangle)
\]

Is unlinkability satisfied?

\[
\text{tag}(id, k) \mid \text{tag}(id, k) \overset{?}{=} \text{tag}(id, k) \mid \text{tag}(id', k')
\]
Modelling the protocol

The interactions between devices

Protocols written in a process calculus, the applied pi calculus

\[
P ::= 0 \\
| \text{in}(c, x).P \quad \text{input} \\
| \text{out}(c, t).P \quad \text{output} \\
| \text{if } t_1 = t_2 \text{ then } P \text{ else } Q \quad \text{conditional} \\
| P \parallel Q \quad \text{parallel} \\
| !P \quad \text{replication} \\
| \text{new } n.P \quad \text{restriction}
\]

The network is the attacker

No communication between the parallel processes:

- messages are sent to the attacker
- received messages are the ones chosen by the attacker
- concurrency is determined by the attacker
Modelling the protocol

Cryptographic primitives

Data structure and functions used are represented by terms.

- messages = terms

- perfect cryptography (equational theories)
  \[ \text{dec} (\text{enc}(x, y), y) = x \quad \text{fst}(\text{pair}(x, y)) = x \quad \text{snd}(\text{pair}(x, y)) = y \]

Specificities:
- messages are terms
- equality in conditionals interpreted modulo an equational theory
Reasoning about attacker knowledge

The execution of the process is formalized by an operational semantics.

\[
\text{Send } (\text{out}(c, t).P, \varphi) \xrightarrow{\text{out}(c)} (P, \varphi \cup \{w_{|\varphi|+1} \mapsto t\downarrow\})
\]

There is no attacker process, the semantics only remember the messages received by the attacker in a frame.

\[
P := \text{out}(c, t_1).\text{out}(c, t_2).\text{out}(c, t_3)
\]

\[
(P, \emptyset) \xrightarrow{\text{out}(c)}^3 (0, \text{new } \vec{n}. \{t_1/w_1, t_2/w_2, t_3/w_3\})
\]
Reasoning about attacker knowledge

Inputs

**Recipe:** A term which contains elements of the frame.

It represents the computations performed by the attacker.

The attacker uses these recipes to:

- provide inputs to the process
- test equalities

Semantics for input:

\[
\text{RECV } (\text{in}(c, x). P, \varphi) \xrightarrow{\text{in}(c, R)} (P\{x \mapsto t\}, \varphi)
\]

if \( t \) can be obtained from \( \varphi \) with recipe \( R \).

The label \( \text{in}(c, R) \) keeps track of which recipe has been chosen by the attacker. It is used to compare with traces of the other process.
Reasoning about attacker knowledge

Deducibility

A recipe provides a term to the process.
Formally, $\phi \vdash^R t$ if $R$ is a public term and $R\phi =_E t$

$$
\varphi = \text{new } n_1, n_2, k_1, k_2. \{ \text{enc}(n_1,k_1) / w_1, \text{enc}(n_2,k_2) / w_2, k_1 / w_3 \}
$$

$$
\varphi \vdash^\text{dec}(w_1,w_3) n_1 \quad \varphi \not\vdash n_2 \quad \varphi \vdash^0 0
$$
Reasoning about attacker knowledge

Static equivalence:

The received messages look like random nonce for the attacker except if some equalities hold.

$$\text{new } n, k. \{^{\text{enc}(0,k)}/w_1, k/w_2\}$$

$$(\text{dec}(w_1, w_2)) = 0$$

If some equality hold in one scenario but not in another one, it can distinguish the two scenarios.

$$\text{new } n_1, n_2. \{^{n_1}/w_1, ^{n_2}/w_2\} \not\sim \text{ new } n_1. \{^{n_1}/w_1, ^{n_1}/w_2\}$$

Check ($$w_1 \equiv w_2$$)
Trace equivalence (for processes)

Static equivalence: two frames where the same equalities hold.

\[ \phi_1 \sim_s \phi_2 \text{ if } \forall \text{ public terms } R, R'. \]

\[ R\phi_1 = R'\phi_1 \iff R\phi_2 = R'\phi_2 \]

Formal definition of the equivalence property:

Definition (Trace equivalence: \( P \approx Q \))

if \((P, \emptyset) \xrightarrow{\text{tr}} (P', \varphi)\)
then \(\exists Q', \varphi'. (Q, \emptyset) \xrightarrow{\text{tr}} (Q', \varphi') \land \varphi \sim_s \varphi' \) and reciprocally.
Linkability attack

1 Tag
\( k, id \)

\[ \langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle \]

2 Tags
\( k, id \)

\[ \langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle \]

\[ \langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle \]

\[ \langle id \oplus r_2, h(\langle r_1, k \rangle) \oplus r_2 \rangle \]

\[ \langle id' \oplus r_2', h(\langle r_1, k' \rangle) \oplus r_2' \rangle \]

\[ \langle id' \oplus r_2', h(\langle r_1, k' \rangle) \oplus r_2' \rangle \]

\[ \langle id' \oplus r_2', h(\langle r_1, k' \rangle) \oplus r_2' \rangle \]

\[ (id \oplus r_2) \oplus (h(\langle r_1, k \rangle) \oplus r_2) \overset{?}= ([id/\bar{id}'] \oplus r_2') \oplus (h(\langle r_1, [k/k'] \rangle) \oplus r_2') \]

\[ (id \oplus r_2) \oplus (h(\langle r_1, k \rangle) \oplus r_2) \overset{?}= ([id/\bar{id}'] \oplus r_2') \oplus (h(\langle r_1, [k/k'] \rangle) \oplus r_2') \]
Verification tools

- Good support to verify traces properties (which rely on one process)
- Equivalence properties (which rely on two processes): Maude-NPA, APTE, SPEC, ProVerif, Tamarin, Deepsec, Akiss

But

- Except for APTE and DeepSec, no support for else branches or support for simple cases (same control flow in both scenarios).
- No tool for equivalence provides Xor primitive support and else branches.

My personal contribution to Akiss

- Add xor support [CSF’17]
- Add else branches support [ESORICS’17]
AKISS: overview

Rewrite rules

Protocol specification
process calculus
no replication

Query
\( P \not\equiv Q \)

Translation into first order Horn clauses

Linearization

Saturation of Horn clauses
(Resolution based procedure)

Pass on the other process? No

Disequalities treatment

Yes

Equivalence is proven (if \( P, Q \) are determinate)

Prise en charge du xor [CSF’17]

Witness of non equivalence [ESORICS’17]
Three kinds of predicate:

- $r_w$ there is a trace $L_1, \ldots, L_n$ whose abstraction is $w$.
- $k_w(R, t)$ after a trace $w$, the attacker can build $t$ with recipe $R$.
- $i_w(R, R')$ after a trace $w$, recipes $R$ and $R'$ provide the same term.

where $w$ represents the trace where the predicate is valid.
Modelling protocols in Horn clauses

Direct translation from the protocol and the theory to Horn Clauses

The seed of a process

\[ R = \{ \text{dec}(\text{enc}(x, y), y) \to x \} \]

\[ T = \text{in}(c, x).\text{if}\ (\text{dec}(x, k) = a)\ \text{then}\ \text{out}(c, s) \]

\[ k_w(\text{dec}(X, Y), z) \iff k_w(X, \text{enc}(z, y)), k_w(Y, y) \]

\[ k_{\text{receive}}(c, \text{enc}(a, k)), \text{test}, \text{send}(c)(w_1, s) \iff k(X, \text{enc}(a, k)) \]
Resolution procedure from the seed statements

\[
H \leftarrow B_1, B_2 \quad B_1 \leftarrow B_3, B_4 \\
H \leftarrow B_2, B_3, B_4
\]

Saturation until we get a saturated set where premises do not carry information

Theorem

For any identity that the attacker can get on \( P \), there is a more general identity in the saturated set.
Interest of Xor?

- Used in many low device protocols
  - RFID tags
  - Mobile telephony (AKA)
- Used as a ciphering method but weaker than encryption
  - Some attacks exploit the algebraic properties of xor

There was no tool to effectively analyze equivalence properties for such protocols
Xor: a complex operator

Lot of algebraic properties:

- **Associativity:** \( x \oplus (y \oplus z) = (x \oplus y) \oplus z \)
- **Commutativity:** \( x \oplus y = y \oplus x \)
- **Nilpotent:** \( x \oplus x = 0 \)
- **Unit:** \( x \oplus 0 = x \)

Requires reasoning modulo AC

Unifiers of \( h(x) \oplus y = h(x') \oplus y' \)?

Several solutions (no most general unifier)

- \( x \mapsto x' \), \( y \mapsto y' \)
- \( y \mapsto h(x') \), \( y' \mapsto h(x) \)
- \( y \mapsto h(x') \oplus z \), \( y' \mapsto h(x) \oplus z \)
Dealing with Xor

Partial correctness: no issue
Termination: saturation never terminates

- Restrict the use of some saturation rules
- Show it is equivalent to enforce some parenthesizing
Protocols with Else branches: a large class of protocols

Else branches are used for:
- sending error messages
- avoid replay attacks
- restart processes when failure

The AKA protocol used in 3G telephony.

Ivan Gazeau (LORIA)  Automated verification of protocols  March 16th, 2018  24 / 29
Why are Else branches difficult to analyze?

Easy when checking trace property:
Only need to check that disequalities can be satisfied.

**Issue when considering equivalence of \( P \) and \( Q \):**

- A single action on \( P \) can be simulated in \( Q \) by different actions

Example

\[
P = \text{in}(x)\text{out}(x)
\]

\[
Q = \text{in}(x) \text{ if } x = a \text{ then } \text{out}(x) \text{ else } \text{out}(x)
\]

- Not stable by substitution

Example

\( x = w_1 \) does not imply \( a = w_1 \)
Disequalities: our approach

If an identity exists on $P$ but not on $Q$ due to a disequality test then a test pass on $Q$ where the disequality has been replaced by an equality and on $P$.

Symbolic run: $\text{receive}(c, h(x)).\text{send}(c)$
On process $Q$: $\text{in}(c, x).[x \neq h(0)].\text{out}(c, 0)$
Resulting trace: $\text{in}(c_{\text{art}}, y).\text{in}(c, x).[x = h(y)].[x = h(0)].\text{out}(c, 0)$

This approach uses the procedure for positive processes as a black box.
Case studies

Akiss is implemented in Ocaml and is available on Github.  
https://github.com/akiss/akiss

We test several protocols:

- AKA (attack in 3m)
- BAC (attack in 1m30)
- PAP (safe in 4s)
- RFID protocols: KCL, LAK, LD, MD, NSL xor, OTYT, SLK.
- Guessing attack: Nonce, Direct authentication, Gong
Current work: abstract structure to deal with interleaving

Issue
Currently all interleaving are verified independently: exponential blow up

\[ P = \text{in}(c, x).\text{out}(c, f(x)) \mid \text{in}(c, y).\text{out}(c, g(y)) \mid \text{in}(c, z).\text{out}(c, h(z)) \]

240 interleavings

Idea to solve the problem
- \( k_w(R, t) \): make \( w \) abstract to represent a set of interleavings
Conclusion

A procedure to analyze equivalence of processes.
- bounded number of sessions
- support for xor
- support for disequalities

Properties:
- Sound and complete
- Termination granted without xor
- An efficient tool