Automation of the Computationally Complete Symbolic Attacker (CCSA)

Simon Jeanteur

March 28, 2023

TU Wien, Vienna, Austria

Overview

S&P

- Proofs in the CCSA make use of the **first-order** BC-Logic
- There exists some fairly good automated first-order theorem prover

- Proofs in the CCSA make use of the **first-order** BC-Logic
- There exists some fairly good automated first-order theorem prover

Can we leverage those provers to **automatically** prove computationally sound properties on cryptographic protocols?

Speedrunning an example

We consider a simple protocol inspired by the RFID protocols.

- A tag outputs a fresh nonce n in plain text and hashed with a key k shared with the reader.
- The reader then tries to authenticate the tag by checking that the message it received was properly constructed using a key within its database.

$$\boxed{T_{i\,j} \to R_i : \quad \nu \mathsf{n}.\, \langle \mathsf{n}, \mathcal{H}(\mathsf{n}, \mathsf{k}_j) \rangle}$$

We model a single reader and multiple tags (parametrized by j) over multiple sessions (parametrized by i).

We are concerned with the *authentication* of a tag by the reader. It intuitively means that when a reader accepts an authentication of the j^{th} tag, then the latter really did try to authenticate.

 S&P

```
let condition(j:tag_idx, in:Message)
  { verify(sel2of2(in), sel1of2(in), key(j)) }
step reader(i:session, j:tag_idx)
  { condition(j, input(reader(i, j))) } { ok }
step reader_fail(i:session)
  { not(exists (j:tag_idx) {condition(j, input(reader_fail(i)))}) }{ ko }

step tag(i:session, j:tag_idx)  /* the tag */
  { true } { tpl(nt(i,j), hash(nt(i,j), key(j))) }

/* ordering */
order forall (i:session, j:tag_idx) { reader_fail(i) <> reader(i, j) }
order forall (i:session, j:tag_idx, j2:tag_idx)
  { reader(i, j2) <> reader(i, j) }
```

An axiom example: euf-cma

When **hash** and **verify** form a message authentication code[1] (MAC) scheme that is existentially unforgeable under chosen message attacks (Euf-Cma), the protocol $\mathcal{P}$ verifies:

$$\forall m, \sigma, \mathsf{k}.\ |\ \textbf{verify}\ (\sigma, m, \mathsf{k})| \\ \Rightarrow \left(\mathsf{k} \sqsubseteq_{\textbf{hash}(\_,\bullet)} m, \sigma, \mathcal{P} \vee \exists u.\ (\textbf{hash}(u, \mathsf{k}) \sqsubseteq m, \sigma \wedge |u| = |m|)\right)$$

where k is a nonce and all $\sqsubseteq$ relations are variations of subterm relations.

[1] i.e. A symmetric signature scheme

Encoding the protocol

S&P

$$T_{ij} \to R_i : \quad \nu \mathsf{n}.\, \langle \mathsf{n}, \mathcal{H}(\mathsf{n}, \mathsf{k}_j) \rangle$$

The protocol gets encoded in the logic via its interaction with our axioms:

$$u \sqsubset \underline{\mathtt{input}}(\tau) \Rightarrow \bigvee \begin{array}{r} \exists i, j. T_{ij} < \tau \wedge (u \sqsubseteq \mathit{msg}(T_{ij}) \vee u \sqsubseteq \mathit{cond}(T_{ij})) \\ \exists i, j. R_{ij}^{\mathsf{succ}} < \tau \wedge \left(u \sqsubseteq \mathit{msg}\left(R_{ij}^{\mathsf{succ}}\right) \vee u \sqsubseteq \mathit{cond}\left(R_{ij}^{\mathsf{succ}}\right)\right) \\ \exists i. R_i^{\mathsf{fail}} < \tau \wedge \left(u \sqsubseteq \mathit{msg}\left(R_i^{\mathsf{fail}}\right) \vee u \sqsubseteq \mathit{cond}\left(R_i^{\mathsf{fail}}\right)\right) \\ \mathsf{init} < \tau \wedge (u \sqsubseteq \mathit{msg}(\mathsf{init}) \vee u \sqsubseteq \mathit{cond}(\mathsf{init})) \end{array}$$

$$T_{i\,j} \to R_i : \quad \nu \mathsf{n}.\, \langle \mathsf{n}, \mathcal{H}(\mathsf{n}, \mathsf{k}_j) \rangle$$

We can preprocess the euf-cma axiom to some extent by pre-instantiating it. For instance:

$$\forall i, j. \left| \textbf{verify}\,(\texttt{sel2of2}(\underline{\texttt{input}}(R^{\mathsf{succ}}_{i\,j}))\texttt{, sel1of2}(\underline{\texttt{input}}(R^{\mathsf{succ}}_{i\,j}))\texttt{,}\mathsf{k}[j]) \right|$$
$$\Rightarrow \exists k, j'. \left(T_{k\,j'} < R^{\mathsf{succ}}_{i\,j} \wedge j = j' \wedge \left|\mathsf{n}_T\left[k, j'\right]\right| = \left|\texttt{sel1of2}(\underline{\texttt{input}}(R^{\mathsf{succ}}_{i\,j}))\right|\right)$$

Premiliraly Results

Protocol	cryptoverif (ms)	us (ms)
basic-hash	51.8 ± 6.7	22.8 ± 1.9
feldhofer	59.4 ± 5.1	82.4 ± 19.4
hash-lock	60.6 ± 6.2	37.5 ± 3.3
lak-tag	60.4 ± 4.6	60.7 ± 10.9
mw	71.0 ± 7.9	59.3 ± 9.3