Analysing cryptographic protocols using Tamarin

Stéphanie Delaune
14 June 2021

Univ Rennes, CNRS, IRISA, Spicy team
Security protocol design is **critical** and **error-prone** as illustrated by many **attacks**:

- SSL/TLS: FREAK, Logjam, ...

Use **formal methods** to improve confidence:

- **prove** the **absence of attacks** under certain **assumptions**; or
- **identify** **weaknesses**

Many **tools** already exist:

- ProVerif, Tamarin, AKISS, DeepSec, AVISPA, Squirrel, ...
Security protocol design is **critical** and **error-prone** as illustrated by many **attacks**:

- SSL/TLS: FREAK, Logjam, ...

Use **formal methods** to improve confidence:

- **prove** the **absence of attacks** under certain **assumptions**; or
- **identify** **weaknesses**

Many **tools** already exist:

- ProVerif, **Tamarin**, AKISS, DeepSec, AVISPA, Squirrel, ...

**Problem**: trade-off between **automation** and completeness
Tamarin prover

→ mainly developed at ETH Zurich
https://tamarin-prover.github.io

- A verification tool for the **symbolic model** with induction, loops, mutable state
- Successfully used for many **large-scale case studies**: 5G AKA, TLS 1.3, EMV ...
- Security protocol model based on **multiset rewriting**
- Constraint-solving algorithm for analysis of **unbounded number of sessions**
- **Interactive** and **automatic** modes
Tamarin’s **interactive mode** allows the user to inspect and direct proof search

- Gives the **flexibility** required for complex case-studies
- Enables **fine-tuning** of models and proof strategies
Tamarin’s **interactive mode** allows the user to inspect and direct proof search

- Gives the **flexibility** required for complex case-studies
- Enables **fine-tuning** of models and proof strategies

On the downside, Tamarin’s **automatic mode** often fails (compared to, e.g., ProVerif), even on relatively **simple examples**.  

→ partial deconstructions.
Tamarin’s **interactive mode** allows the user to inspect and direct proof search

- Gives the **flexibility** required for complex case-studies
- Enables **fine-tuning** of models and proof strategies

On the downside, Tamarin’s **automatic mode** often fails (compared to, e.g., ProVerif), even on relatively **simple examples**. → **partial deconstructions**.

**Our contribution:**
automatic handling of partial deconstructions in most cases.
High-level view of Tamarin
In a nutshell

Modelling part:

- protocol and adversary: multiset rewriting
  → a transition system which induces a set of traces
- security properties: a fragment of first-order logic
  → this specifies “good” traces

Verification part – Tamarin tries to

- construct a counterexample trace, i.e. an attack; or
- provide a proof that all the traces produce by the system are good.
Basic ingredients

Terms – messages:

- built using function symbols, e.g. aenc/2, adec/2, pk/1 ...
- interpreted modulo an equational theory.

Example:

\[
\text{aenc}(\langle \text{req, I, n}\rangle, \text{pk}(\text{ltkR})) \quad \text{adec}(\text{aenc}(x, \text{pk}(y), y) = x}
\]
Basic ingredients

Terms – messages:

- built using function symbols, e.g. aenc/2, adec/2, pk/1 ...
- interpreted modulo an equational theory.

Example:

\[
\text{aenc}(\langle \text{req}, l, n \rangle, \text{pk}(ltkR)) \quad \text{adec}(\text{aenc}(x, \text{pk}(y), y)) = x
\]

Facts – think “sticky notes on the fridge”:

- user defined facts of two kinds: linear or persistent (prefixed with !)
- some special facts: Fr(n), ln(t), Out(t), !K(t)

A state of a system is a multiset of facts, and rules specify the possible moves.
Each rule has the following form: \([l] \rightarrow [a] \rightarrow [r]\) where:

- \(l, r\) are multisets of facts, and
- \(a\) is a multiset of annotations used for specifying properties
Multiset rewriting rules

Each rule has the following form: \([l] \rightarrow [a] \rightarrow [r]\) where:

- \(l, r\) are multisets of facts, and
- \(a\) is a multiset of annotations used for specifying properties

Some examples:

1. \([!K(x_1), !K(x_2)] \rightarrow [K(aenc(x_1, x_2))] \rightarrow ![K(aenc(x_1, x_2))]\)
2. \([!K(x_1), !K(x_2)] \rightarrow [K(adec(x_1, x_2))] \rightarrow ![K(adec(x_1, x_2))]\)
3. \([\text{Out}(x)] \rightarrow [] \rightarrow ![K(x)]\)
4. \([!K(x)] \rightarrow [K(x)] \rightarrow [\text{In}(x)]\)
5. \([[]] \rightarrow [!\text{Fr}(n)]\)
Consider the following toy protocol between the **initiator** and the **responder**:

1. $\text{init} \rightarrow \text{res} : \{\text{req}, I, n\}_{\text{pk}(R)}$

2. $\text{res} \rightarrow \text{init} : \{\text{rep}, n\}_{\text{pk}(I)}$
Consider the following toy protocol between the **initiator** 🧑 and the **responder** 🧑:

1. 🧑 → 🧑: \{\text{req}, I, n\}_{\text{pk}(R)}
2. 🧑 → 🧑: \{\text{rep}, n\}_{\text{pk}(I)}

**rule Register\_pk:**

\[
[\text{Fr}(\neg \text{ltkA})] \\
\rightarrow [\neg \text{ltk}(A, \neg \text{ltkA}), \neg \text{Pk}(A, \text{pk}(\neg \text{ltkA})), \text{Out}(\text{pk}(\neg \text{ltkA}))]
\]
Consider the following toy protocol between the initiator and the responder:

1. $\rightarrow$: \{req, I, n\}_{pk(R)}
2. $\rightarrow$: \{rep, n\}_{pk(I)}

rule Register_{pk}:
[ Fr(~ltkA) ]
--> [ !Ltk($A, ~ltkA), !Pk($A, pk(~ltkA)), Out(pk(~ltkA)) ]

rule Rule_I:
[Fr(n), !Pk(R, pkR), !Ltk(I, ltkI) ]
--> [SecretI(I, R, n)]--> [Out(aenc{'req', I, n}pkR)]
A set of protocol rules $P$ induces a transition relation between states.

$$S \xrightarrow{a}^{P} (S \setminus I) \cup r$$

where $[l] \rightarrow [a] \rightarrow [r]$ a ground instance of a rule, and $I \subseteq S$
A set of protocol rules $P$ induces a transition relation between states.

$$S \xrightarrow{a} P (S \setminus l) \cup r$$

where $[l] \rightarrow [a] \rightarrow [r]$ a ground instance of a rule, and $l \subseteq S$

- **Executions**

  $$\text{Exec}(P) = \{ {} \xrightarrow{a_1} P \ldots \xrightarrow{a_n} S_n \mid \forall n. \text{Fr}(n) \text{ apprears only once on rhs of rules} \}$$

- **Traces**

  $$\text{Traces}(P) = \{ [a_1, \ldots, a_n] \mid {} \xrightarrow{a_1} P \ldots \xrightarrow{a_n} S_n \in \text{Exec}(P) \}$$
Property specification

First-order logic interpreted over traces $a_1, \ldots, a_n$:

- message equality: $t_1 = t_2$
- action at a particular timepoint: $A@\#i$
- timepoint ordering: $\#i < \#j$
- timepoint equality: $\#i = \#j$

Example: Secrecy for the nonce $n$.

**lemma nonce_secrecy:**

"not(Ex A B s \#i. SecretI(A, B, s) @ \#i & (Ex \#j. K(s) @ \#j))"
Property specification

First-order logic interpreted over traces $a_1, \ldots, a_n$:

- message equality: $t_1 = t_2$
- action at a particular timepoint: $A@#i$
- timepoint ordering: $#i < #j$
- timepoint equality: $#i = #j$

Example: Secrecy for the nonce $n$.

```
lemma nonce_secrecy:
  "not(
    Ex A B s #i. SecretI(A, B, s) @ #i
    & (Ex #j. K(s) @ #j)
  )"
```
A **backward search** algorithm starting from the conclusion.
A **backward search** algorithm that relies on some **precomputations**: the sources.

Sources are a combination of rules yielding a particular fact as part of the result.

Example:

Computation of raw sources can stop in an incomplete stage (**partial deconstruction**) if **TAMARIN** lacks sufficient information about the origins of some fact.
Algorithm intuition (3/3)

Proof scripts

theory runningV1 begin
Message theory
Multiset rewriting rules (5)
Raw sources (8 cases, 6 partial deconstructions left)
Refined sources (8 cases, 6 partial deconstructions left)

lemma nonce_secrecy:
  all-traces
  \[ \neg \exists A B s \# i \# j. \]
  \[ \text{SecretI}( A, B, s ) \# i \land \text{K}( s ) \# j \]" 
  simplify
  solve( IPk( B, pkR ) \triangleright i )
  case Register_pk
  solve( !lk( $I, lkI ) \triangleright i )
  case Register_pk
  by sorry /* removed */
qed
qed
end

Visualization display

Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed)

1. solve( IKU( \neg n ) \# vk ) // nr. 6

a. autoprove (A. for all solutions)
b. autoprove (B. for all solutions) with proof-depth bound 5

Constraint system

last: none

formulas:
the proof of this lemma does not terminate due to partial deconstructions.
Partial deconstructions
Example: Partial deconstruction
Example: Partial deconstruction
To resolve these partial deconstructions, one has to write sources lemma detailing the possible origins of the problematic fact.
To resolve these partial deconstructions, one has to write sources lemma detailing the possible origins of the problematic fact.

Considering our running example:

the input is either the message sent by the initiator, or a message constructed by the intruder.

→ the previous raw source will lead to two refined sources:

1. either the variable is actually a nonce generated by the initiator;
2. or it a term already known by the attacker (such a detour is not useful).
To *resolve* these partial deconstructions, one has to write *sources lemma* detailing the possible origins of the problematic fact.

Considering our running example:

*the input is either the message sent by the initiator, or a message constructed by the intruder.*

→ the previous raw source will lead to **two refined sources**:

1. either the variable is actually a *nonce* generated by the initiator;
2. or it a term already known by the attacker (**such a detour is not useful**).

Sources lemmas are used to *refine* the sources, but they also need to be *proven correct.*

→ this can be done using Tamarin.
First, we **annotate** the protocol rules:

**rule Rule_I:**

\[ [ Fr(n), !Pk(R, pkR), !Ltk(I, ltkI)] \]

\[ --[ I(aenc{"req", I, n}pkR), SecretI(I, R, n)] \rightarrow \]

\[ [ Out(aenc{"req", I, n}pkR) ] \]

**rule Rule_R:**

\[ [ In(aenc{"req", I, x}pk(ltkR)), 
   !Ltk(R, ltkR), !Pk(I, pkI) ] \]

\[ --[ R(aenc{"req", I, x}pk(ltkR), x)] \rightarrow \]

\[ [ Out(aenc{"rep", x}pkI) ] \]

---

**Lemma typing \([sources]\):**

\[ \text{All } x \, m \#i. R(m,x)@#i ==>(Ex \#j. I(m)@#j & \#j < \#i) \lor (Ex \#j. KU(x)@#j & \#j < \#i) \]
First, we **annotate** the protocol rules:

rule Rule_I:
   [ Fr(n), !Pk(R, pkR), !Ltk(I, ltkI) ]
   -- [ I(aenc{'req', I, n}pkR), SecretI(I, R, n) ]->
   [ Out(aenc{'req', I, n}pkR) ]

rule Rule_R:
   [ In(aenc{'req', I, x}pk(ltkR)),
     !Ltk(R, ltkR), !Pk(I, pkI) ]
   -- [ R(aenc{'req', I, x}pk(ltkR), x) ]->
   [ Out(aenc{'rep', x}pkI) ]

lemma typing [sources]:
"All x m #i. R(m,x)@#i =>> ((Ex #j. I(m)@#j & #j < #i) |
   (Ex #j. KU(x)@#j & #j < #i))"
**Generalize idea & automate** the approach:

1. Inspect the **raw sources** computed by **TAMARIN**
2. For each partial deconstruction:
   - 2.1 Identify the **variables** and **facts** causing the partial deconstruction
   - 2.2 Identify rules producing **matching conclusions**
   - 2.3 Add necessary **annotations** to the concerned rules
3. Generate a **sources lemma** using all annotations and add it to the input file
**Algorithm Idea**

Generalize idea & automate the approach:

1. Inspect the raw sources computed by Tamarin
2. For each partial deconstruction:
   2.1 Identify the variables and facts causing the partial deconstruction
   2.2 Identify rules producing matching conclusions
   2.3 Add necessary annotations to the concerned rules
3. Generate a sources lemma using all annotations and add it to the input file

Note that Tamarin will verify the correctness of the generated lemma.

But we actually proved that the lemmas we generate are correct under some assumptions (well-formed rules, subterm-convergent equational theory).
We **implemented** the algorithm in **TAMARIN** (available in version 1.6.0).

To **enable** automatic source lemma generation, run **TAMARIN** with **--auto-sources**:  

- If **partial deconstructions** are present and there is **no sources lemma**, the algorithm generates a lemma and adds it to the theory.
- If there is already a lemma, or there are no partial deconstructions, **TAMARIN** runs as usual.
We tried numerous examples from the **SPORE library**:

<table>
<thead>
<tr>
<th>Protocol Name</th>
<th>Partial Dec.</th>
<th>Resolved</th>
<th>Automatic</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew Secure RPC</td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>42.8s</td>
</tr>
<tr>
<td>Modified Andrew Secure RPC</td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>134.3s</td>
</tr>
<tr>
<td>BAN Concrete Andrew Secure RPC</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>10.6s</td>
</tr>
<tr>
<td>Lowe modified BAN Andrew Secure RPC</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>29.8s</td>
</tr>
<tr>
<td>CCITT 1</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>0.8s</td>
</tr>
<tr>
<td>CCITT 1c</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>1.2s</td>
</tr>
<tr>
<td>CCITT 3</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>186.1s</td>
</tr>
<tr>
<td>CCITT 3 BAN</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>3.7s</td>
</tr>
<tr>
<td>Denning Sacco Secret Key</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.8s</td>
</tr>
<tr>
<td>Denning Sacco Secret Key - Lowe</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>2.7s</td>
</tr>
<tr>
<td>Needham Schroeder Secret Key</td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>3.6s</td>
</tr>
<tr>
<td>Amended Needham Schroeder Secret Key</td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>7.1s</td>
</tr>
<tr>
<td>Otway Rees</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>7.7s</td>
</tr>
<tr>
<td>SpliceAS</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>5.9s</td>
</tr>
<tr>
<td>SpliceAS 2</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>7.3s</td>
</tr>
<tr>
<td>SpliceAS 3</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>8.7s</td>
</tr>
<tr>
<td>Wide Mouthed Frog</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.6s</td>
</tr>
<tr>
<td>Wide Mouthed Frog Lowe</td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>3.5s</td>
</tr>
<tr>
<td>WooLam Pi f</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.6s</td>
</tr>
<tr>
<td>Yahalom</td>
<td>15</td>
<td>✓</td>
<td>✓</td>
<td>3.1s</td>
</tr>
<tr>
<td>Yahalom - BAN</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.9s</td>
</tr>
<tr>
<td>Yahalom - Lowe</td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>2.2s</td>
</tr>
</tbody>
</table>
Case studies: Tamarin repository

We also tested all examples from the Tamarin repository:

<table>
<thead>
<tr>
<th>Name</th>
<th>Partial Dec.</th>
<th>Resolved</th>
<th>Automatic</th>
<th>Time (new)</th>
<th>Time (previous)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feldhofer (Equivalence)</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>3.8s</td>
<td>3.5s</td>
</tr>
<tr>
<td>NSLPK3</td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>1.8s</td>
<td>1.8s</td>
</tr>
<tr>
<td>NSLPK3 untagged</td>
<td>12</td>
<td>✓</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NSPK3</td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>2.4s</td>
<td>2.2s</td>
</tr>
<tr>
<td>JCS12 Typing Example</td>
<td>7</td>
<td>✓</td>
<td>x</td>
<td>0.3s</td>
<td>0.2s</td>
</tr>
<tr>
<td>Minimal Typing Example</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>0.1s</td>
<td>0.1s</td>
</tr>
<tr>
<td>Simple RFID Protocol</td>
<td>24</td>
<td>✓</td>
<td>x</td>
<td>0.7s</td>
<td>0.5s</td>
</tr>
<tr>
<td>StatVerif Security Device</td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>0.3s</td>
<td>0.4s</td>
</tr>
<tr>
<td>Envelope Protocol</td>
<td>9</td>
<td>✓</td>
<td>x</td>
<td>25.7s</td>
<td>25.3s</td>
</tr>
<tr>
<td>TPM Exclusive Secrets</td>
<td>9</td>
<td>✓</td>
<td>x</td>
<td>1.8s</td>
<td>1.8s</td>
</tr>
<tr>
<td>NSL untagged (SAPIC)</td>
<td>18</td>
<td>✓</td>
<td>✓</td>
<td>4.3s</td>
<td>19.9s</td>
</tr>
<tr>
<td>StatVerif Left-Right (SAPIC)</td>
<td>18</td>
<td>✓</td>
<td>✓</td>
<td>28.8s</td>
<td>29.6s</td>
</tr>
<tr>
<td>TPM Envelope (Equivalence)</td>
<td>9</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5G AKA</td>
<td>240</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Alethea</td>
<td>30</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PKCS11-templates</td>
<td>68</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NSLPK3XOR</td>
<td>24</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Chaum Offline Anonymity</td>
<td>128</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FOO Eligibility</td>
<td>70</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Okamoto Eligibility</td>
<td>66</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Conclusion & Future Work

- Automation in Tamarin often fails because of **partial deconstructions**
- Developed & implemented a new algorithm to **automatically generate** sources lemmas
- Proved **correctness** of the generated lemmas
- Algorithm **works well in practice**, many examples become fully or at least partly automatic
- Available in Tamarin 1.6.0

**Future work:**
- Handle more general **equational theories**
- Handle partial deconstructions stemming from **state facts** (currently under submission at JCS)
Questions?