# Analysing cryptographic protocols using Tamarin

Stéphanie Delaune

14 june 2021

Univ Rennes, CNRS, IRISA, Spicy team

# Formal verification of cryptographic protocols


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, ...

# Formal verification of cryptographic protocols


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 developped at ETH Zurich

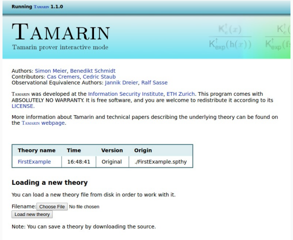
`https://tamarin-prover.github.io`

- 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

# Interaction and automation

Tamarin's **interactive mode** allows the user to inspect and direct proof search

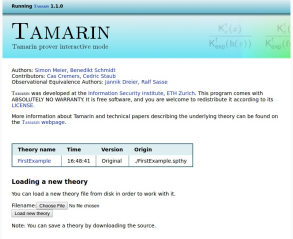
- Gives the **flexibility** required for complex case-studies
- Enables **fine-tuning** of models and proof strategies



# Interaction and automation

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

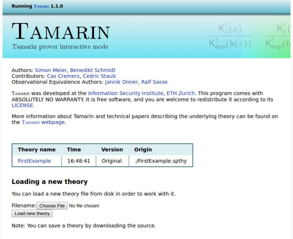



On the downside, Tamarin's **automatic mode** often fails (compared to, e.g., ProVerif), even on relatively **simple examples**. $\longrightarrow$ partial deconstructions.

# Interaction and automation

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**. $\longrightarrow$ 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
  $\longrightarrow$ a transition system which induces a set of traces
- security properties: a fragment of first-order logic
  $\longrightarrow$ 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 funtion symbols, e.g. aenc/2, adec/2, pk/1 ...
- interpreted modulo an equational theory.

Example:

$$\mathrm{aenc}(\langle req, I, n\rangle, \mathrm{pk}(ltkR)) \qquad \mathrm{adec}(\mathrm{aenc}(x, \mathrm{pk}(y), y) = x$$

# Basic ingredients

Terms – messages:

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

Example:

$$\mathrm{aenc}(\langle req, I, n\rangle, \mathrm{pk}(ltkR)) \qquad \mathrm{adec}(\mathrm{aenc}(x, \mathrm{pk}(y), y) = x$$

Facts – think “sticky notes on the fridge”:

- user defined facts of two kinds: linear or persistent (prefixed with !)
- some special facts: $\mathrm{Fr}(n)$, $\mathrm{In}(t)$, $\mathrm{Out}(t)$, $!\mathrm{K}(t)$

A state of a system is a multiset of facts, and rules specify the possible moves.

# Multiset rewriting rules

Each rule has the following form: $[l]\!-\!\![\ a\ ]\!\!\rightarrow\![r]$ 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: $[l]-[\; a \;]\rightarrow[r]$ where:

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

Some examples:

1. $[!K(x_1), !K(x_2)]-[\; K(aenc(x_1, x_2)) \;]\rightarrow[!K(aenc(x_1, x_2))]$
2. $[!K(x_1), !K(x_2)]-[\; K(adec(x_1, x_2)) \;]\rightarrow[!K(adec(x_1, x_2))]$
3. $[Out(x)]-[]\rightarrow[!K(x)]$
4. $[!K(x)]-[\; K(x) \;]\rightarrow[In(x)]$
5. $[]-[]\rightarrow[Fr(n)]$

# Toy example

Consider the following toy protocol between the **initiator** and the **responder**:

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

# Toy example

Consider the following toy protocol between the **initiator** and the **responder**:

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

```
rule Register_pk:
  [ Fr(~ltkA) ]
  --> [ !Ltk($A, ~ltkA), !Pk($A, pk(~ltkA)), Out(pk(~ltkA)) ]
```

# Toy example

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:
  [ 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)]
```

# Semantics

A set of protocol rules $P$ induces a transition relation between states.

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

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

# Semantics

A set of protocol rules $P$ induces a transition relation between states.

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

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

- Executions

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

- Traces

$$Traces(P) = \{[a_1, \ldots, a_n] \mid \{\} \rightsquigarrow_{P}^{a_1} \ldots \rightsquigarrow_{P}^{a_n} S_n \in Exec(P)\}$$

# 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: $\#i = \#j$

# 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)
    )"
```

# Algorithm intuition (1/3)

A backward search algorithm starting form the conclusion.

Running TAMARIN 1.7.0

## 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
  "¬(∃ A B s #i #j.
          (SecretI( A, B, s ) @ #i) ∧ (K( s ) @
#j))"
simplify
by sorry

end
```

## Visualization display

**Applicable Proof Methods:** Goals sorted according to the 'sma

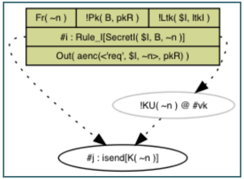
1. solve( !Pk( B, pkR ) ▶$_1$ #i ) // nr. 3 (from rule Rule_I)

2. solve( !Ltk( $I, ltkI ) ▶$_2$ #i ) // nr. 4 (from rule Rule_I)

3. solve( !KU( ~n ) @ #vk ) // nr. 6

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

**Constraint system**



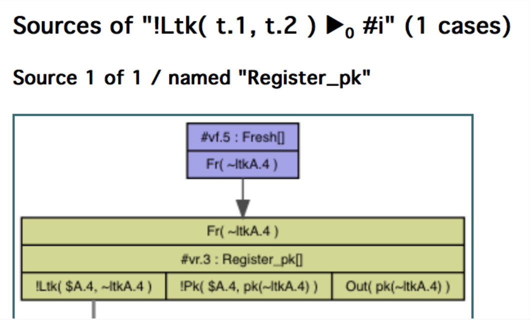

**last:** none

# Algorithm intuition (2/3)

A backward search algorithm that relies on some precomputations: the sources.

Sources are a combination of rules yiedling 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)

Running TAMARIN 1.7.0 Index Download Actions »

## 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
  "¬(∃ A B s #i #j.
           (SecretI( A, B, s ) @ #i) ∧ (K( s ) @
#j))"
simplify
solve( !Pk( B, pkR ) ▶₁ #i )
  case Register_pk
  solve( !Ltk( $I, ltkI ) ▶₂ #i )
    case Register_pk
    by sorry /* removed */
  qed
qed

end
```

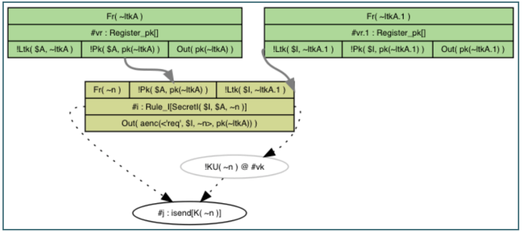
## Visualization display

**Applicable Proof Methods:** Goals sorted according to the 'smart' heuristic (loop breakers delaye

1. solve( !KU( ~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:**

# Algorithm intuition (3/3)

Running Tamarin 1.7.0 Index Download Actions »

**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
  "¬(∃ A B s #i #j.
           (SecretI( A, B, s ) @ #i) ∧ (K( s ) @
#j))"
simplify
solve( !Pk( B, pkR ) ▶1 #i )
  case Register_pk
  solve( !Ltk( $I, ltkI ) ▶2 #i )
    case Register_pk
    by sorry /* removed */
  qed
qed

end
```

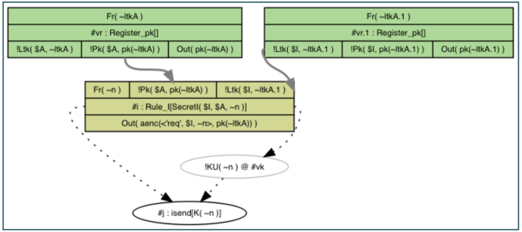
**Visualization display**

**Applicable Proof Methods:** Goals sorted according to the 'smart' heuristic (loop breakers delaye

1. solve( !KU( ~n ) @ #vk ) // nr. 6

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

**Constraint system**



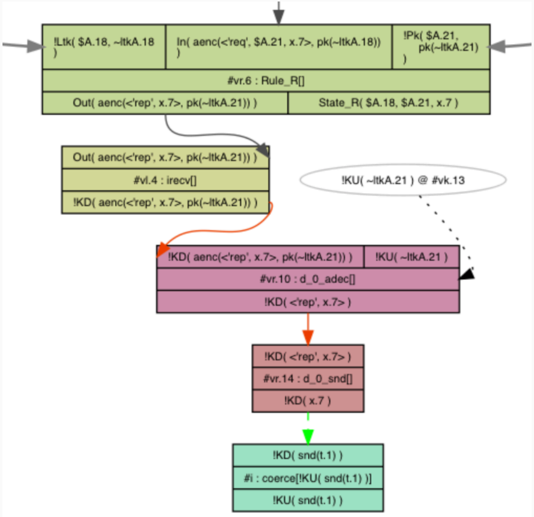

**last:** none

**formulas:**

⟶ the **proof** of this lemma **does not terminate** due to partial deconstructions.

# Partial deconstructions

# Example: Partial deconstruction



# Example: Partial deconstruction

# From raw sources to refined sources

To **resolve** these partial deconstructions, one has to write **sources lemma** detailing the possible origins of the problematic fact.

# From raw sources to refined sources

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.*

$\longrightarrow$ 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)**.

# From raw sources to refined sources

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.*

$\longrightarrow$ 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**.

$\longrightarrow$ this can be done using Tamarin.

# Source lemma on our example

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) ]
```

# Source lemma on our example

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))"
```

# 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

# 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).

# Implementation

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.

# Case studies: SPORE

We tried numerous examples from the **SPORE library**:

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

# Case studies: Tamarin repository

We also tested all examples from the **Tamarin repository**:

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

# 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** (currenly under submission at JCS)

# Questions?