Special topics in Logic and Security I

Master Year II, Sem. I, 2025-2026

Ioana Leuștean
FMI, UB

References

1. Tamarin Manual
   `https://tamarin-prover.com/manual/`
2. Tamarin Book
   `https://tamarin-prover.com/book/index.html`

Tamarin-overview

- A protocol is specified using (order-sorted) multiset rewriting. The rewrite rules define a labelled transition system with agent's actions as labels.
- Protocol security properties are defined in a first order language, as properties on traces.
- The tool provides a proof that all traces satisfy the required property, or provides a counterexample (an attack).

Simple Example

$A \longrightarrow B : \quad \{\!|\, n_a \,|\!\}_{pk(B)}$

- The Tamarin model and the security properties are defined in a file named `Name.spthy`, where `spthy` means *security protocol theory*.
- The adversary is a Dolev-Yao adversary by default.

```
theory SimpleExample
begin
builtins
rule R1:[F1,.., Fn]--[]->[Q1,...,Qm]
rule R2: [F1,.., Fn]--[A1,...,Ak]->[Q1,...,Qn]
lemma L1: "..."
lemma L2: "..."
end
```

Modelling Public Key Infrastructure

```
rule Register_pk:
    [ Fr(~ltk) ] --> [ !Ltk($I, ~ltk), !Pk($I, pk(~ltk)) ]

rule Get_pk:
    [ !Pk(I, pubkey) ] --> [ Out(pubkey) ]

rule Reveal_ltk:
    [ !Ltk(I, ltk) ] --[ LtkReveal(I) ]->  [ Out(ltk) ]
```

- Terms: `ltk`, `I`, `pk(ltk)`, `pubkey`
- Functions: `pk/1`
- Sort declarations: `~ltk` (fresh), `$ltk` (public)
- Facts: `Fr(~ltk)`,`Out(pubkey)`, `Out(ltk)`
- Persistent facts: `!Ltk($I, ~ltk)`, `!Pk($I, pk( ~ltk))`, `!Pk(I, pubkey)`, `!Ltk(I, ltk)`
- Action facts: `LtkReveal(I)`

Terms [Tamarin book]

- Tamarin terms are elements if an **order-sorted** term algebra. The top-sort is `msg`, which have three incomparable subsorts: `pub`, `fresh`, `Nat` (the `natural-numbers built-in`).
- Public constants are written as text strings enclosed in single quotes, such as 'Hello World' or 'label1'.
- We also assume the existence of a countably infinite set of Variables for each sort. Variables have names that are unquoted text strings, like myVar3, and are optionally preceded by a type declaration. Note that variables do not have a global scope: the variables in a multiset rewriting rule are local to that rule. Hence when a variable is instantiated during a rule's application, this instantiation has no effect on variables in other rules that happen to have the same name or on other instantiations of the same rule.
  Variables of sort `pub` are declared by `$t`, variables of sort `fresh` are declared by `~t` and variables of sort `nat` are declared by `%t`. Variables without an explicit type declaration are of the generic top sort msg,

Rules, Facts and Actions [Tamarin book]

- The protocol and its environment are modeled using multiset **rewriting rules**. The rules operate on the system's state, which is expressed as a multiset of facts.
- **Facts** and **actions** are predicate symbols applied to zero or more terms.
  A fact represents part of the (global) system state. For example, the fact `Out(token)` models that the message token is output that has been sent to the network by some agent, whereas the fact `In(token)` models that token is available on the network as input to be read by some agent.
  Actions play a role analogous to labels in labeled transition systems and they model actions taken by agents or actions taken during the protocol's set up.
- System transitions consume facts from the state and produce (other) facts.
  We differentiate between: **persistent facts** (defined with `!`) can be used multiple times, and **linear facts** (defined without `!`) are removed from system's state by performing transitions.

Built-in fact symbols [Tamarin book]

- Tamarin has several built-in fact symbols:
  - K/1: K(t) is used to check whether the adversary can derive the term t,
  - In/1: In(t) represents that t was received from the (adversary controlled) network,
  - Out/1: Out(t) represents that t was sent to the network,
  - Fr/1: Fr(t) represents that t was freshly generated.

- Built-in rules for adversary knowledge

```
[ Out(x) ]-->[ !K(x) ]
[ !K(x) ]--[ K(x) ]->[ In(x) ]
```

- Built-in rule for fresh values

```
Fresh:  []--[]->[ Fr(~x)]
```

  where `x` must be of type fresh. In practice, we use this to model the generation of random values from a sufficiently large space, private keys, etc.

Modelling the Public Key Infrastructure

```
rule Register_pk:
    [ Fr(~ltk) ] --> [ !Ltk($I, ~ltk), !Pk($I, pk(~ltk)) ]
```

First, generate a fresh name ltk (of sort fresh), which is the new private key, and non-deterministically choose a public name A, for the agent for whom we are generating the key-pair. Afterward, generate the fact `!Ltk($A, ~ltk)` (the exclamation mark ! denotes that the fact is persistent, i.e., it can be consumed arbitrarily often), which denotes the association between agent A and its private key `~ltk`, and generate the fact `!Pk($A, pk(~ltk))`, which associates agent A and its public key `pk(~ltk)`.
[Tamarin Manual]

Modelling the Public Key Infrastructure

```
rule Get_pk:
    [ !Pk(I, pubkey) ] --> [ Out(pubkey) ]
```

This rule allows the adversary to retrive any public key.

When using the default Tamarin setup, there is only one public channel modeling the network controlled by the adversary, i.e., the adversary receives all messages from the `Out( )` facts, and generates the protocol's inputs in the `In( )` facts. [Tamarin Manual]

Modelling the Public Key Infrastructure

```
rule Reveal_ltk:
    [ !Ltk(I, ltk) ] --[ LtkReveal(I) ]->  [ Out(ltk) ]
```

This rule models the dynamic compromise of long-term private keys.Intuitively, it reads a private-key database entry and sends it to the adversary. This rule has an observable LtkReveal action stating that the long-term key of agent A was compromised. Action facts are just like facts, but unlike the other facts do not appear in state, but only on the trace.
The security properties are specified on the traces, and the action `LtkReveal` **is used to determine which agents are compromised.**
[Tamarin Manual]

Modelling the protocol

```
rule Initiator:
  [Fr(~k), !Pk($R, pkR) ] --[SentMessage(~k)]->[Out( aenc(~k, pkR))]
```

In details:

```
rule Initiator:
    [ Fr(~k)      // the message is generated as a fresh name

   , !Pk($R, pkR) ] //  lookup the (user, key) association

  --[SentMessage(~k)]-> //     performes the action

    [ Out( aenc(~k, pkR) )  //  sends the encrypted message
    ]
```

- The action (`SendMessage(~k)`) does not appear in the state, but it will be used on the trace.

Modelling the protocol

```
rule Responder:
[!Ltk($R,~ltkR),In( request )]--[AnswerRequest($R,adec(request, ~ltkR))]->[]
```

In details:

```
rule Responder:

    [ !Ltk($R, ~ltkR) // retrives its secret key

    , In( request )]   // receives the request

  --[ AnswerRequest($R, adec(request, ~ltkR)) ]->  []

// the action is needed in order to define
   the security properties
```

Modelling the security property

- Security properties are defined over traces of the action facts of a protocol execution.
- `#i` is a temporal variable and its sort is `time`. The sort `time` of timepoints is not a subsort of `msg`.

```
lemma Responder_auth:
"/* For all messages 'k' and communication party R  */
    ( All R k #i.
/* whenever a party has access to 'k' */
    AnswerRequest(R,k) @ #i
       ==>
         /* there is a responser that answered the request */
       ( (Ex  #a. SentMessage(k) @ a)
      /* or the adversary performed a long-term key reveal
            before the key was setup. */
       | (Ex #r. LtkReveal(R) @ r & r < i)
       )
    )
  "
```

Modelling the security property

```
lemma Responder_auth:
"/* For all session keys 'k'  */
    ( All R k #i.
/* whenever a party has access to 'k' */
    AnswerRequest(R,k) @ #i
       ==>
         /* there is a responser that answered the request */
       ( (Ex  #a. SentMessage(k) @ a)
      /* or the adversary performed a long-term key reveal
            before the key was setup. */
       | (Ex #r. LtkReveal(R) @ r & r < i)
       )
    )
  "
```

Lemmas and traces

All lemmas are implicitly considered to be “all-traces” lemmas unless stated otherwise. This means that they formalize that the given formula holds for all traces of the protocol model and their proof entails checking that there is no trace violating the formula.

The alternative is “exists-trace” lemmas, for which the keyword exists-trace must be explicitly added after the rule NAME: before the actual formula to be proven. An exists-trace lemma is used to check that at least one trace exists. In essence, the proof of such a lemma attempts to find (and show) a trace.

Thus, if a trace is found, its interpretation is different: for an all-traces lemma, the trace represents a violation, while for an exists-trace lemma, it represents a success.

[Tamarin book]

Modelling the security property

To ensure that our lemmas do not just hold vacuously because the model is not executable, we also include an executability lemma that shows that the model can run to completion. This is given as a regular lemma, but with the `exists-trace` keyword, as seen in the following lemma. This keyword says that the lemma is true if there exists a trace on which the formula holds; this is in contrast to the previous lemmas where we required the formula to hold on all traces. When modeling protocols, such existence proofs are useful sanity checks.

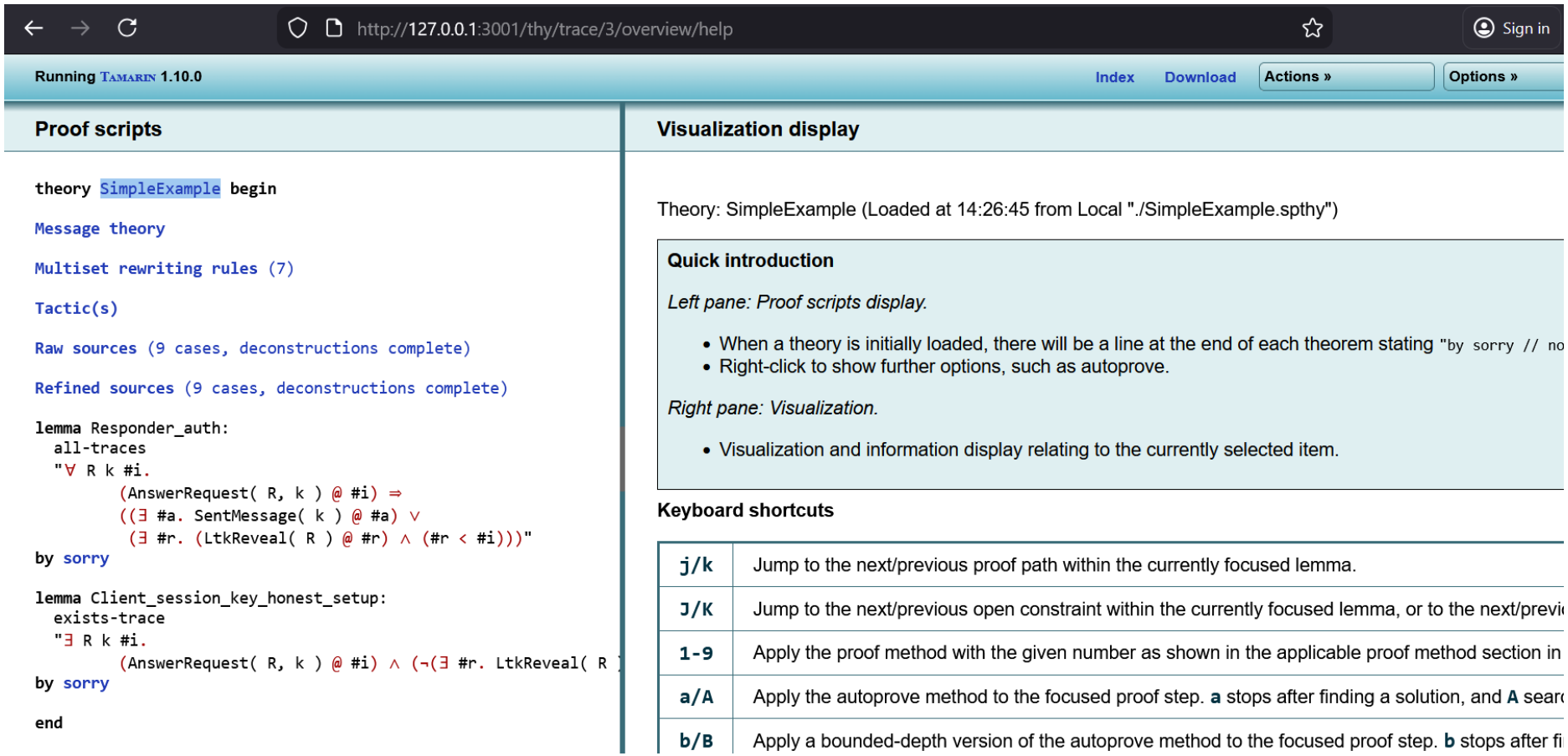
```
lemma Client_session_key_honest_setup:
  exists-trace  /* the lemma is true if there is a trace */
  " Ex R k #i.
        AnswerRequest(R,k) @ #i
      & not(Ex #r. LtkReveal(R) @ r)
  "
```

Modelling the security property

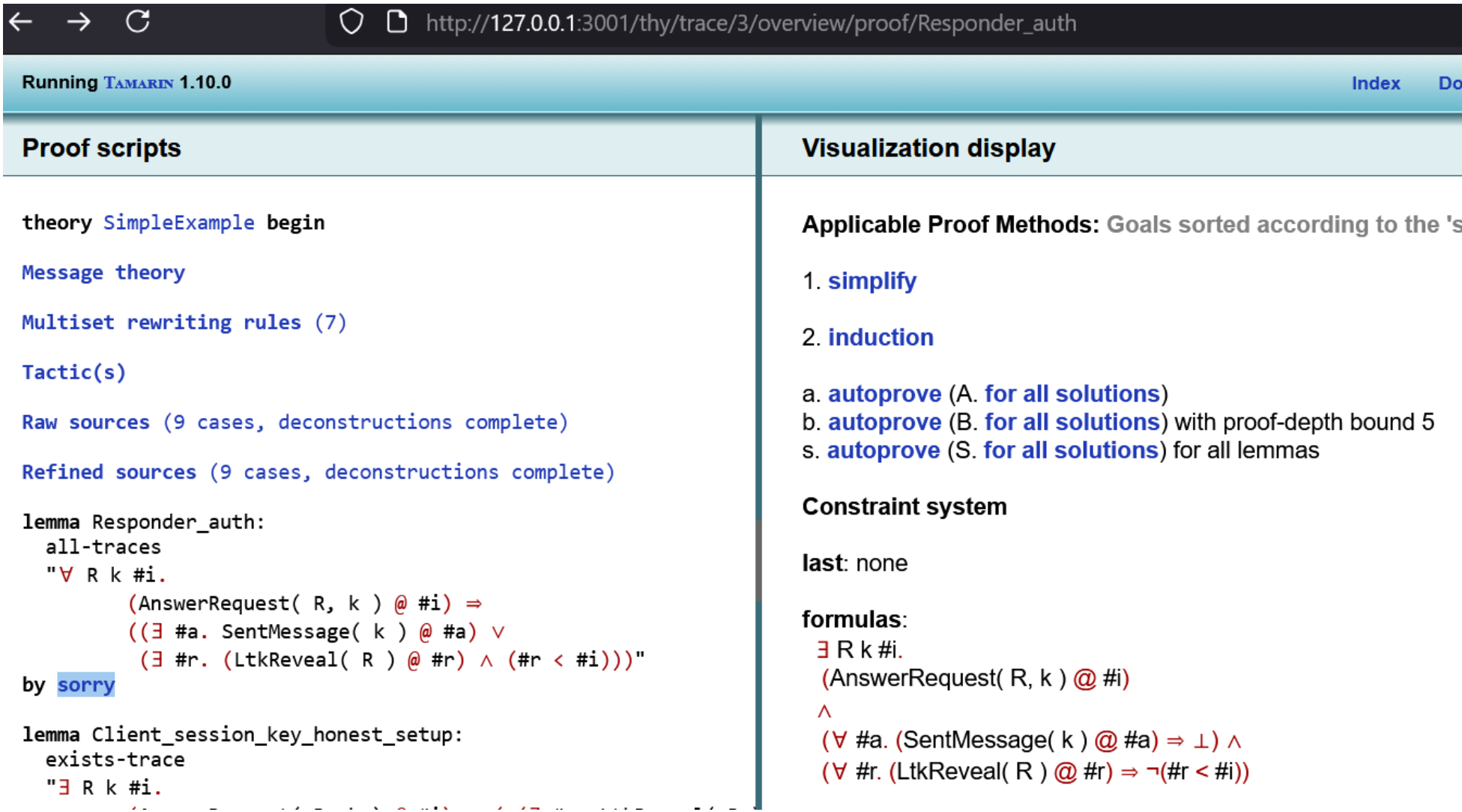
```
lemma Responder_auth:
"
    ( All R k #i.  AnswerRequest(R,k) @ #i
       ==>
          ( (Ex  #a. SentMessage(k) @ a)
          | (Ex #r. LtkReveal(R) @ r & r < i)
       ))
"

lemma Client_session_key_honest_setup:
   exists-trace
  " Ex R k #i.
        AnswerRequest(R,k) @ #i
      & not(Ex #r. LtkReveal(R) @ r)
  "
```

Tamarin

```
> tamarin-prover interactive name.spthy .... Finished loading theori
... server ready at
http://127.0.0.1:3001
```



Tamarin


http://127.0.0.1:3001/thy/trace/3/overview/proof/Responder_auth
Running Tamarin 1.10.0
Index
Do
Proof scripts
theory SimpleExample begin
Message theory
Multiset rewriting rules (7)
Tactic(s)
Raw sources (9 cases, deconstructions complete)
Refined sources (9 cases, deconstructions complete)
lemma Responder_auth:
all-traces
"∀ R k #i.
(AnswerRequest(R, k) @ #i) ⇒
((∃ #a. SentMessage(k) @ #a) ∨
(∃ #r. (LtkReveal(R) @ #r) ∧ (#r < #i)))"
by sorry
lemma Client_session_key_honest_setup:
exists-trace
"∃ R k #i.
Visualization display
Applicable Proof Methods: Goals sorted according to the 's
1. simplify
2. induction
a. autoprove (A. for all solutions)
b. autoprove (B. for all solutions) with proof-depth bound 5
s. autoprove (S. for all solutions) for all lemmas
Constraint system
last: none
formulas:
∃ R k #i.
(AnswerRequest(R, k) @ #i)
∧
(∀ #a. (SentMessage(k) @ #a) ⇒ ⊥) ∧
(∀ #r. (LtkReveal(R) @ #r) ⇒ ¬(#r < #i))

Tamarin

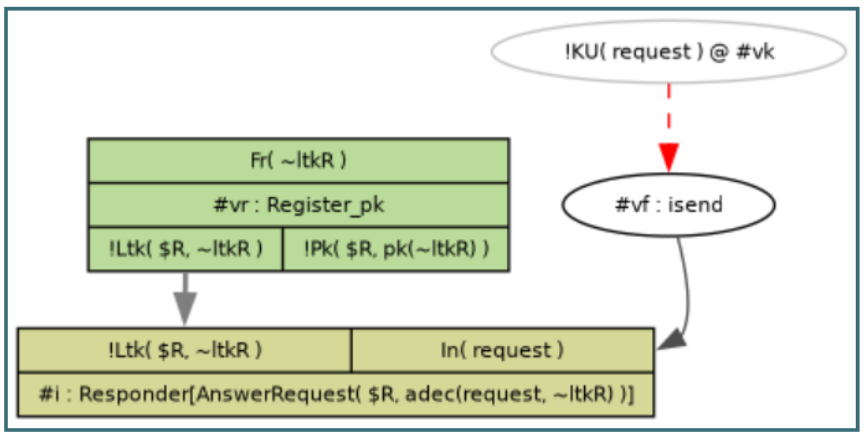
Proof scripts

```
Message theory

Multiset rewriting rules (7)

Tactic(s)

Raw sources (9 cases, deconstructions complete)

Refined sources (9 cases, deconstructions complete)

lemma Responder_auth:
  all-traces
  "∀ R k #i.
         (AnswerRequest( R, k ) @ #i) ⇒
         ((∃ #a. SentMessage( k ) @ #a) ∨
          (∃ #r. (LtkReveal( R ) @ #r) ∧ (#r < #i)))"
simplify
solve( !Ltk( $R, ~ltkR ) ▶₀ #i )
  case Register_pk
  solve( splitEqs(0) )
    case split_case_1
    SOLVED // trace found
  qed
qed
```

Visualization display

Constraint System is Solved

Constraint system



last: none

formulas:

- If the lemma is colored in red, then Tamarin has found a counterexample.

Tamarin

Running Tamarin 1.10.0 — Index — Download — Actions »

Proof scripts

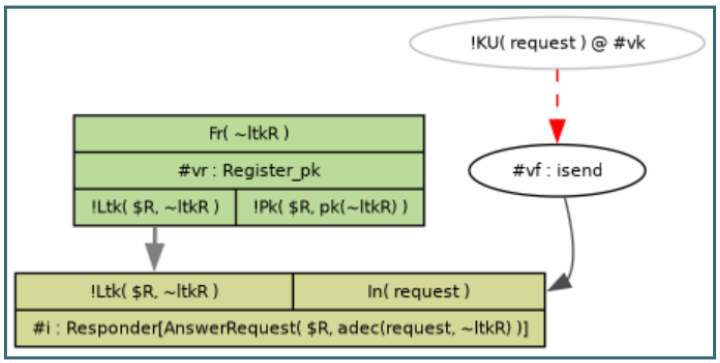
Refined sources (9 cases, deconstructions complete)

```
lemma Responder_auth:
  all-traces
  "∀ R k #i.
        (AnswerRequest( R, k ) @ #i) ⇒
        ((∃ #a. SentMessage( k ) @ #a) ∨
         (∃ #r. (LtkReveal( R ) @ #r) ∧ (#r < #i)))"
simplify
solve( !Ltk( $R, ~ltkR ) ▶₀ #i )
  case Register_pk
  solve( splitEqs(0) )
    case split_case_1
    SOLVED // trace found
  qed
qed

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ R k #i.
        (AnswerRequest( R, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( R ) @ #r))"
simplify
solve( !Ltk( $R, ~ltkR ) ▶₀ #i )
  case Register_pk
  solve( splitEqs(0) )
    case split_case_1
    SOLVED // trace found
  qed
qed
```

Visualization display

Constraint System is Solved

Constraint system



last: none

formulas: ∀ #r. (LtkReveal($R) @ #r) ⇒ ⊥

subterms:

equations:
subst:

References

1 Tamarin Manual
`https://tamarin-prover.com/manual/`

2 Tamarin Book
`https://tamarin-prover.com/book/index.html`