Proving and analysing security protocols with **Tamarin Prover**

LINCS **Network Theory** Working Group, IMT Palaiseau, France, Wednesday 20th December 2023

Guillaume Nibert
guillaume.nibert@snowpack.eu

Adapted Material – Creative Commons License

Tamarin Prover

Open-source model checker for **formal verification** and **analysis** of **security protocols** in the **symbolic model**. It was initially developed at the Information Security Institute, ETH Zürich.

Core team: David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, Benedikt Schmidt

- Cross-platform (Linux, macOS, Windows with WSL)
- **Falsification** and **unbounded verification** support
- **Diffie-Hellmann exponentiation** and **XOR** messages support
- Security protocols specification → **Multiset rewriting systems**
- Analysis of the protocols “w.r.t. **(temporal) first-order properties**”
- **ProVerif** and **Deepsec** export [9]

TLS 1.3 [1, 2, 3] 5G authentication [4, 5, 6] IEEE 802.11 WPA2 [7] + patched version against KRACK [8]

Symbolic modeling / analysis of security protocols

Input

Output

Security protocol **model**

Specification of the **adversary**

Specification of the protocol's **properties**

Proof that **satisfies** the properties.

Initiator
Responder
Trusted key server

- Can be **automatically** constructed…
- including an **arbitrary** number of protocol **instances** running in **parallel**…
- while taking into account the adversary's actions.

Symbolic modeling / analysis of security protocols

Input → Output

Security protocol **model**

Specification of the **adversary**

Specification of the protocol's **properties**

Proof that **satisfies** the properties.

Initiator
Responder
Trusted key server

Multiset rewriting rules → labeled transition system

Symbolic representation of:

- the adversary's knowledge
- the messages on the network
- information about freshly generated values
- the protocol's state

- Can be **automatically** constructed…
- including an **arbitrary** number of protocol **instances** running in **parallel**…
- while taking into account the adversary's actions.

Symbolic modeling / analysis of security protocols

Input → Output

Security protocol **model**

Specification of the **adversary**

Specification of the protocol's **properties**

Proof that **satisfies** the properties.

Initiator
Responder
Trusted key server

Interactions between ● & ● :

- **Updating** network messages
- **Generating** new messages

- Can be **automatically** constructed…
- including an **arbitrary** number of protocol **instances** running in **parallel**…
- while taking into account the adversary's actions.

Symbolic modeling / analysis of security protocols

Input → Output

Security protocol **model**

Specification of the **adversary**

Specification of the protocol's **properties**

Proof that **satisfies** the properties.

Initiator
Responder
Trusted key server

→ **Trace** properties

↕ reasoning

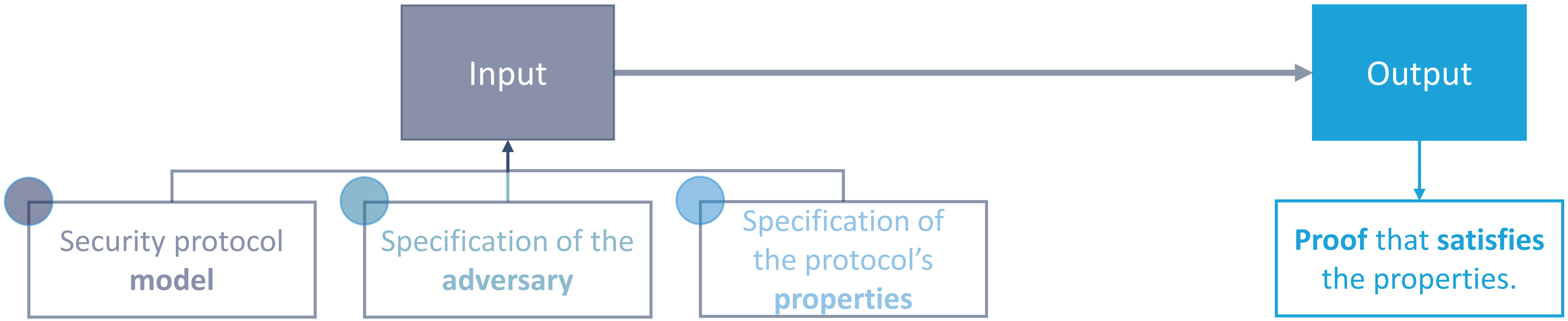
Traces of the transition system

or

Observational equivalence of 2 transition systems [15]

- Can be **automatically** constructed…
- including an **arbitrary** number of protocol **instances** running in **parallel**…
- while taking into account the adversary's actions.

Symbolic modeling / analysis of security protocols



Initiator
Responder
Trusted key server

Automated mode: deduction and equational reasoning with heuristics.
 - Termination: proof or **correctness** or counterexample.
 - May not terminate as correctness of security protocol is an **undecidable problem** [13].

Interactive mode: explore proof states, attack graphs
→ combine manual proof guidance & automated mode.

- Can be **automatically** constructed…
- including an **arbitrary** number of protocol **instances** running in **parallel**…
- while taking into account the adversary's actions.

Symbolic modeling / analysis of security protocols

Input → Output

Security protocol **model**

Specification of the **adversary**

Specification of the protocol's **properties**

Proof that **satisfies** the properties.

Initiator
Responder
Trusted key server

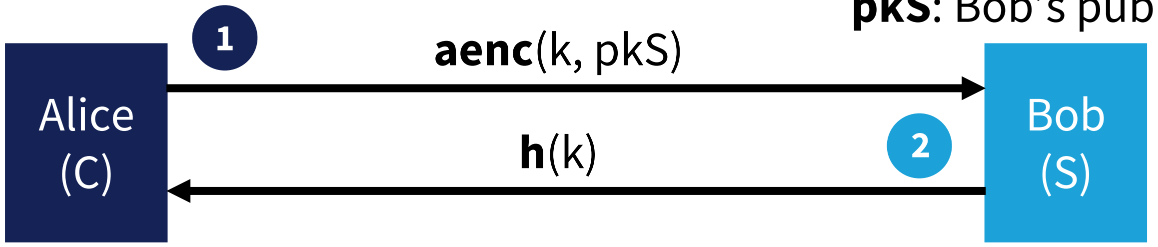
- Can be **automatically** constructed…
- including an **arbitrary** number of protocol **instances** running in **parallel**…
- while taking into account the adversary's actions.

Simple Encrypted Communication

Protocol to model

k: Alice's symmetric key

pkS: Bob's public key



Adversary to consider

A **Dolev-Yao** adversary

- Controls the network
- Can delete, inject, modify and intercept messages
- *+ can dynamically compromise private keys*

Security property to prove

From Alice point of view, **k** sent to Bob is not compromised

aenc: asymmetric encryption function
h: hash function

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Starting with Tamarin

Authors: **Simon Meier**, **Benedikt Schmidt**
Updated by: **Jannik Dreier**, **Ralf Sasse**
Date: **June 2016**

```
theory FirstExample  // theory's name
begin
```

```
builtins: hashing, asymmetric-encryption
```

`h (1 param)`: a cryptophic hash function

`aenc (2 params)`: asymmetric encryption algorithm
`adec (2 params)`: asymmetric decryption algorithm
`pk (1 param)`: public key corresponding to a private key

`adec(aenc(m, pk(sk)), sk)` is reduced to **`m`**

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Multiset rewriting rules

Rules operates on the system's state → **Multiset** of **facts**.
Facts: *predicates* storing **state information**. They appear on the trace.

Rule: "**Premise**", "**-->**", "**Conclusion**".
Execution of a rule:

- *Premise*: all facts in the premise are **present in the current state**.
- --> **execution** of the rule.
- *Conclusion*: facts in the conclusion are added to the **state**, those from the premise are **removed**.

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Creating a PKI

Rules operates on the system's state → **Multiset** of **facts**.
Facts: *predicates* storing **state information**.

Facts' Tamarin representation

`F(t1,...,tn)` with terms `ti` and a fixed arity n.

Rule: "**Premise**", "**-->**", "**Conclusion**".
Execution of a rule:

- *Premise*: all facts in the premise are **present in the current state**.
- --> **execution** of the rule.
- *Conclusion*: facts in the conclusion are added to the **state**, those from the premise are **removed**.

Registering a public key

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

Special built-in Fr fact

`Fr`: built-in **fact**, denotes a freshly generated name. For modelling **nonces**/**keys**.

Variable prefixes

`~x` denotes `x:fresh`
`$x` denotes `x:pub`
`#i` denotes `i:temporal`
`m` denotes `m:msg`

$$C \to S: \{k\}_{pkS}$$
$$S \to C: h(k)$$

Creating a PKI

Registering a public key

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

Variable prefixes

`~x` denotes `x:fresh`
`$x` denotes `x:pub`
`#i` denotes `i:temporal`
`m` denotes `m:msg`

String constant

`'c'` denotes a public name in pub, global constant.

Generation of a fresh name **`~ltk`** (private key) and choice of a public name **A** (non-deterministically) which corresponds to the agent associated with the newly created key-pair.

Persistence

`!` denotes the persistence of a fact.

`!Ltk($A, ~ltk)` : association of agent **A** and its private key `~ltk`
`!Pk($A, pk(~ltk))` : association of agent **A** and its public key `pk(~ltk)`

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Modelling the adversary

Allowing an adversary to get any public key

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

Out/In special built-in facts

Out/In denotes a party sending (resp. receiving) a message to (from) the **untrusted** network (**Dolev-Yao**). Only right-hand (left-hand) of a multiset rewrite rule.

The public key is read from the public-key database and sent to the network using the built-in fact **Out**.

Reminder

Variables	`~x` denotes `x:fresh`
	`$x` denotes `x:pub`
	`#i` denotes `i:temporal`
	`m` denotes `m:msg`
	`'c'` denotes a **public name** in `pub`, global constant.

Facts	`F(t1,...,tn)` with terms `ti` and a fixed arity `n`.
	`!` denotes the persistence of a fact.
	`Fr`: built-in **fact**, denotes a freshly generated name. For modelling **nonces**/**keys**.

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Modelling the adversary

Dynamically compromising long-term private keys

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

Action facts

`-- [ACTIONFACT] ->`: facts that **do not appear in state**, but only on the **trace**.

Located within the arrow.

`!Ltk(A, ltk)`: **A**'s long-term private-key **`ltk`** database entry was read.

`LtkReveal(A)`: states that **A**'s long-term private-key **`ltk`** was compromised.

`Out(ltk)`: **A**'s long-term private-key **`ltk`** was sent to the adversary.

Reminder

Variables	`~x` denotes `x:fresh`
	`$x` denotes `x:pub`
	`#i` denotes `i:temporal`
	`m` denotes `m:msg`
	`'c'` denotes a **public name** in `pub`, global constant.

Facts	`F(t1,...,tn)` with terms `ti` and a fixed arity `n`.
	`!` denotes the persistence of a fact.
	`Fr`: built-in **fact**, denotes a freshly generated name. For modelling **nonces**/**keys**.

`Out`/`In` denotes a party sending (resp. receiving) a message to (from) the **untrusted** network (**Dolev-Yao**). Only right-hand (left-hand) of a multiset rewrite rule.

Simple Encrypted Communication

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Modelling the protocol – client side

```
// Start a new thread executing the client role, choosing the server
// non-deterministically.
rule Client_1:
    [ Fr(~k)          // choose fresh key
    , !Pk($S, pkS)    // lookup public-key of server
    ]
  -->
    [ Client_1( $S, ~k )     // Store server and key for next step of thread
    , Out( aenc(~k, pkS) )  // Send the encrypted session key to the server
    ]

rule Client_2:
    [ Client_1(S, k)  // Retrieve server & session key from previous step
    , In( h(k) )      // Receive hashed session key from network
    ]
  --[ SessKeyC( S, k ) ]-> // State that the session key 'k'
    []                     // was setup with server 'S'
```

Reminder

Variables

- `~x` denotes `x:fresh`
- `$x` denotes `x:pub`
- `#i` denotes `i:temporal`
- `m` denotes `m:msg`
- `'c'` denotes a **public name** in `pub`, global constant.

Facts

- `F(t1,...,tn)` with terms `ti` and a fixed arity `n`.
- `!` denotes the persistence of a fact.
- `Fr`: built-in **fact**, denotes a freshly generated name. For modelling **nonces**/**keys**.

`Out/In` denotes a party sending (resp. receiving) a message to (from) the **untrusted** network (**Dolev-Yao**). Only right-hand (left-hand) of a multiset rewrite rule.

`--[ACTIONFACT]->`: facts that **do not appear in state**, but only on the **trace**. Located within the arrow.

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Modelling the protocol – server side

```
// A server thread answering in one-step to a session-key setup request from
// some client.
rule Serv_1:
  [ !Ltk($S, ~ltkS)                          // lookup the private-key
  , In( request ) ]                          // receive a request
   -->
  [ Out( h(adec(request, ~ltkS)) ) ]         // Return the hash of the
                                             // decrypted request.
```

Reminder

Variables

`~x` denotes `x:fresh`
`$x` denotes `x:pub`
`#i` denotes `i:temporal`
`m` denotes `m:msg`
`'c'` denotes a **public name** in `pub`, global constant.

Facts

`F(t1,...,tn)` with terms `ti` and a fixed arity `n`.

`!` denotes the persistence of a fact.

`Fr`: built-in **fact**, denotes a freshly generated name. For modelling **nonces**/**keys**.

`Out/In` denotes a party sending (resp. receiving) a message to (from) the **untrusted** network (**Dolev-Yao**). Only right-hand (left-hand) of a multiset rewrite rule.

`-- [ACTIONFACT] ->`: facts that **do not appear in state**, but only on the **trace**. Located within the arrow.

Appendix A2: Authentication

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Writing a security property

Security properties are defined over **traces** of the **action facts** of a protocol execution.

```
lemma
```

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Writing a security property

Security properties are defined over **traces** of the **action facts** of a protocol execution.

lemma

```
lemma Client_session_key_secrecy:
  " /* It cannot be that a  */
    not(
      Ex S k #i #j.
        /* client has set up a session key 'k' with a server'S' */
        SessKeyC(S, k) @ #i
        /* and the adversary knows 'k' */
      & K(k) @ #j
        /* without having performed a long-term key reveal on 'S'. */
      & not(Ex #r. LtkReveal(S) @ r)
    )
  "
```

Client point of view – Session key secrecy property

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Writing a security property

Security properties are defined over **traces** of the **action facts** of a protocol execution.

lemma

```
lemma Client_session_key_secrecy:
  " /* It cannot be that a  */
    not(
      Ex S k #i #j.
        /* client has set up a session key 'k' with a server'S' */
        SessKeyC(S, k) @ #i
        /* and the adversary knows 'k' */
      & K(k) @ #j
        /* without having performed a long-term key reveal on 'S'. */
      & not(Ex #r. LtkReveal(S) @ r)
    )
  "
```

Client point of view – Session key secrecy property

`f@i`: predicate symbol representing a fact occurring at timepoint **`i`** (position **`i`** in the trace).

`Pred(t1,…,tn)`: syntactic sugar, instantiation of a predicate for the terms **`t1`** to **`tn`**.

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Writing a security property

Security properties are defined over **traces** of the **action facts** of a protocol execution.

lemma

```
lemma Client_session_key_secrecy:
  " /* It cannot be that a  */
    not(
      Ex S k #i #j.
        /* client has set up a session key 'k' with a server'S' */
        SessKeyC(S, k) @ #i
        /* and the adversary knows 'k' */
      & K(k) @ #j
        /* without having performed a long-term key reveal on 'S'. */
      & not(Ex #r. LtkReveal(S) @ r)
    )
  "
```

Client point of view – Session key secrecy property

Guarded fragment of a **many-sorted first-order logic** with a sort for timepoints.

f@i: predicate symbol representing a fact occurring at timepoint **i** (position **i** in the trace).

Pred(t1,…,tn): syntactic sugar, instantiation of a predicate for the terms **t1** to **tn**.

$$\neg(\exists S,k,i,j.\ \mathrm{SessKeyC}(S,k) @ i \wedge K(k) @ j \wedge \neg(\exists r.\ \mathrm{LtkReveal}(S) @ r))$$

True if it **holds** on **all traces**

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Writing an executability property

Security properties are defined over **traces** of the **action facts** of a protocol execution.

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

Client point of view - Model **executability** property

$$\exists S,k,i.\ (\\ SessKeyC(S,k)\ @\ i \\ \wedge \neg(\exists r.\ LtkReveal(S)\ @\ r) \\ )$$

True if **there exists a trace** on which **it** holds

`exists-trace` keyword

$$C \rightarrow S: \{k\}_{pkS}$$
$$S \rightarrow C: h(k)$$

Ending the theory

```
end
```

? **Guarded fragment** of a **many-sorted first-order logic** with a sort for timepoints. **?**

Propositional logic *(Propositional calculus)*: studies **propositions** and their logical relations (**logical connectives**).

Proposition: statement that is either **true** or **false**, such as "it is raining" or “5+5=10".

Components of a **propositional logic** *language*:

- a set of *primitive* symbols (known as **variables**, atomic formula or proposition letters…)
- a set of *operator* symbols (**logical connectives**; {∧, ∨, →, ↔, ¬, ⊥,...})

Symbols are the **syntactic structures** of a *formal language* used to illustrate ideas, concepts or abstractions.

A **formula** (or *well-formed formula*) is syntactic structure composed of a **finite sequence** of **symbols**.

A **formal language** is a syntactic structure (entity) composed of a **set** of finite **strings of symbols** *(*words that are *well-formed formulas)*.

Syntax is the study of the formal rules that define how logical expressions are constructed from ***symbols*** and ***logical connectors***.

? Guarded fragment of a many-sorted first-order logic with a sort for timepoints. ?

Propositional logic *(Propositional calculus)*: studies **propositions** and their logical relations (**logical connectives**).

Proposition: statement that is either **true** or **false**, such as "it is raining" or "5+5=10".

Components of a **propositional logic** *language*:

- a set of *primitive* symbols (known as **variables**, atomic formula or proposition letters…)
- a set of *operator* symbols (**logical connectives**; {∧, ∨, →, ↔, ¬, ⊥,...})

What is it for?

Creating **proof systems** *(i.e. a formal system, which* ***models*** *a* ***language****)*

Natural deduction system

Simple axiom system

Guarded fragment of a **many-sorted first-order logic** with a sort for timepoints.

First-order logic *(First-order predicate calculus)*: extends ***propositional logic*** by adding **predicates** and two **quantifiers**.

Predicate: symbol representing a relation or a property. E.g. Equal is the symbol of the $\mathrm{Equal(a,b)}$ formula where a and b are elements from the same interpretation domain. Here the arity of the predicate is 2. $=$ could be another symbol to be used…

Quantifiers: ∀ and ∃.

Components of a **first-order logic** *language*:

- a set of *primitive* symbols (known as **variables**, atomic formula or proposition letters…)
- a set of *operator* symbols (**logical connectives**; {∧, ∨, →, ↔, ¬, ⊥,...})
- a set of *predicate* symbols
- a set of *quantifier* symbols ({∀, ∃})

? **Guarded fragment** of a **many-sorted first-order logic** with a sort for **timepoints**. ?

Many-sorted first-order logic *(typed first-order logic)*: extends ***first-order logic*** by allowing variables to have **different sorts** (in different domains).

E.g. $\mathrm{SessKeyC}(\mathrm{S}, \mathrm{k})$ is the predicate symbol of the $\mathrm{SessKeyC}(\mathrm{S}, \mathrm{k})$ formula where S and k are elements from different interpretation domains.

"with a sort for **timepoints**" refer to **temporal logic** a branch of **modal logic**.

Modal logic deals with the concept of *necessity and possibility:*

- **Temporal logic**: type of **modal logic** that deals with the concepts of **time** and **temporal relations** (*necessity/possibility* of a predicate being **true** at **time t**).

Components of a **many-sorted first-order logic with sort for timepoints** *language*:

- a set of *primitive* symbols (known as **variables**, atomic formula or proposition letters where primitive variables belong to different interpretation domains…)
- a set of *operator* symbols (**logical connectives**; {∧, ∨, →, ↔, ¬, ⊥,...})
- a set of *predicate* symbols
- a set of *quantifier* symbols ({∀, ∃})

? **Guarded fragment** of a **many-sorted first-order logic** with a sort for timepoints. ?

Fragment *(from a language)*: **subset** of the original language by applying it **syntax restrictions**.

Guarded logic is a family of first-order logics that have the property that all **quantified variables** are guarded by **atoms** *(specific properties, facts)*.

? **Guarded fragment** of a **many-sorted first-order logic** with a sort for timepoints. ?

Fragment *(from a language)*: **subset** of the original language by applying it **syntax restrictions**.

Guarded logic is a family of first-order logics that have the property that all **quantified variables** are guarded by **atoms** *(specific properties, facts)*.

$\forall x\, P(x) \rightarrow Q(x)$

$\forall x\, Q(x)$

Guarded fragment of a **many-sorted first-order logic** with a sort for timepoints.

Fragment *(from a language)*: **subset** of the original language by applying it **syntax restrictions**.

Guarded logic is a family of first-order logics that have the property that all **quantified variables** are guarded by **atoms** *(specific properties, facts)*.

$\forall x\ P(x) \rightarrow Q(x)$ ✔

$\forall x\ Q(x)$ ✘

Guarded fragment of a **many-sorted first-order logic** with a sort for timepoints.

Fragment *(from a language)*: **subset** of the original language by applying it **syntax restrictions**.

Guarded logic is a family of first-order logics that have the property that all **quantified variables** are guarded by **atoms** *(specific properties, facts)*.

$\forall x\ P(x) \rightarrow Q(x)$ ✔

$\forall x\ Q(x)$ ✖

Decidability of the **logic** → Determine the **truth** or **falsity** of any formula in the logic.

? **Guarded fragment** of a **many-sorted first-order logic** with a sort for timepoints. ?

Fragment *(from a language)*: **subset** of the original language by applying it **syntax restrictions**.

Guarded logic is a family of first-order logics that have the property that all **quantified variables** are guarded by **atoms** *(specific properties, facts)*.

$\forall x\, P(x) \rightarrow Q(x)$ ✔

$\forall x\, Q(x)$ ✖

Decidable logic → Determine the **truth** or **falsity** of any formula in the logic.

Components of a **guarded fragment of a many-sorted first-order logic with sort for timepoints** *language*:

- a set of *primitive* symbols (known as **variables**, atomic formula or proposition letters where primitive variables belong to different interpretation domains…)
- all **quantified variables** are guarded by **atoms**.
- a set of *operator* symbols (**logical connectives**; {∧, ∨, →, ↔, ¬, ⊥,...})
- a set of *predicate* symbols
- a set of *quantifier* symbols ({∀, ∃})

Appendix A6
Appendix A7

Ubuntu installation

```
# Installing the Homebrew package manager
sudo apt install build-essential procps curl file git

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Installation of Tamarin
brew install tamarin-prover/tap/tamarin-prover
```

Other OSes: https://tamarin-prover.github.io/manual/master/book/002_installation.html

Opening the First example

First example available at: https://tamarin-prover.github.io/manual/master/code/FirstExample.spthy

```
tamarin-prover interactive FirstExample.spthy
```

Open your favorite web browser and go to http://127.0.0.1:3001

Running TAMARIN **1.6.1**

TAMARIN

Tamarin prover interactive mode

Core team: David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, Benedikt Schmidt
Tamarin is a collaborative effort: see the manual for a more extensive overview of its development and additional contributors.

TAMARIN was developed at the Information Security Institute, ETH Zurich. This program comes with ABSOLUTELY NO WARRANTY. It is free software, and you are welcome to redistribute it according to its LICENSE.

More information about Tamarin and technical papers describing the underlying theory can be found on the TAMARIN webpage.

Theory name	Time	Version	Origin
FirstExample	11:40:11	Original	./FirstExample.spthy

Loading a new theory

You can load a new theory file from disk in order to work with it.

Filename: Browse... No file selected.
Load new theory

Note: You can save a theory by downloading the source.

Using Tamarin

snowpack | LIP6 | cnrs | SCIENCES SORBONNE UNIVERSITÉ | LINCS

Return to welcome page

Running TAMARIN 1.6.1 | Index | Download | Actions » | Options »

Download the theory + partial proofs if exists

Source code

Graph visualization details level

Proof scripts

```
theory FirstExample begin

Message theory
Multiset rewriting rules (8)
Raw sources (10 cases, deconstructions complete)
Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
         ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
         (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Adversary: Message theory

Protocol: Multiset rewriting rules

Sources: Raw sources, Refined sources

Properties to prove: the lemmas

Visualization display

Theory: FirstExample (Loaded at 11:40:11 from Local "./FirstExample.spthy")

Quick introduction

Left pane: Proof scripts display.

- When a theory is initially loaded, there will be a line at the end of each theorem stating "by sorry // not yet proven". Click on sorry to inspect the proof state.
- Right-click to show further options, such as autoprove.

Right pane: Visualization.

- Visualization and information display relating to the currently selected item.

Keyboard shortcuts

j/k	Jump to the next/previous proof path within the currently focused lemma.
J/K	Jump to the next/previous open goal within the currently focused lemma, or to the next/previous lemma if there are no more sorry steps in the proof of the current lemma.
1-9	Apply the proof method with the given number as shown in the applicable proof method section in the main view.
a/A	Apply the autoprove method to the focused proof step. **a** stops after finding a solution, and **A** searches for all solutions. Needs to have a sorry selected to work.
b/B	Apply a bounded-depth version of the autoprove method to the focused proof step. **b** stops after finding a solution, and **B** searches for all solutions. Needs to have a sorry selected to work.
?	Display this help message.

Proof scripts

```
theory FirstExample begin

Message theory
Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
         ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
         (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Adversary

Message theory

Signature → **List** of symbols of *functions, relations, constants* and *equations*.

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

Construction Rules → Describe the adversary's **applicable functions**.

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]

rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

Describe the adversary's **extractable terms** from larger terms by using functions.

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```


Details: Appendix A3

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Protocol

Multiset rewriting rules and restrictions

Multiset Rewriting Rules

```
rule (modulo AC) isend:
   [ !KU( x ) ] --[ K( x ) ]-> [ In( x ) ]

rule (modulo AC) irecv:
   [ Out( x ) ] --> [ !KD( x ) ]

rule (modulo AC) Register_pk:
   [ Fr( ~ltk ) ] --> [ !Ltk( $A, ~ltk ), !Pk( $A, pk(~ltk) ) ]

rule (modulo AC) Get_pk:
   [ !Pk( A, pubkey ) ] --> [ Out( pubkey ) ]

rule (modulo AC) Reveal_ltk:
   [ !Ltk( A, ltk ) ] --[ LtkReveal( A ) ]-> [ Out( ltk ) ]

rule (modulo AC) Client_1:
   [ Fr( ~k ), !Pk( $S, pkS ) ]
  -->
   [ Client_1( $S, ~k ), Out( aenc(~k, pkS) ) ]

rule (modulo AC) Client_2:
   [ Client_1( S, k ), In( h(k) ) ] --[ SessKeyC( S, k ) ]-> [ ]

rule (modulo AC) Serv_1:
   [ !Ltk( $S, ~ltkS ), In( request ) ]
  --[ AnswerRequest( $S, z ) ]->
   [ Out( h(z) ) ]
  variants (modulo AC)
  1. ~ltkS = ~ltkS.5
     request
           = request.5
     z = adec(request.5, ~ltkS.5)

  2. ~ltkS = ~x.5
     request
           = aenc(x.6, pk(~x.5))
     z = x.6
```

Offer an **interface** that bridges protocol **Output/Input** and **adversary deduction**.

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
           ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
           (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Automated proof generation ☺

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
           ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
           (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin's precomputation phase

Premises inspection of **all rules**

↑ **Facts**

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
           ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
           (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin's precomputation phase

Premises inspection of **all rules**

Facts

Fact $\xrightarrow{\text{precomp.}}$ **set** of *possible* ***sources***

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin’s precomputation phase

Premises inspection of **all rules**

↑ **Facts**

Fact $\xrightarrow{\text{precomp.}}$ **set** of *possible* ***sources***

Source: combination of **rules** ⟶ **Fact** obtainment

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin's precomputation phase

Premises inspection of **all rules**

↳ **Facts**

Fact $\xrightarrow{\text{precomp.}}$ **set** of ***possible sources***

Source: combination of **rules** ⟶ **Fact** obtainment

Raw *sources* | **Refined** *sources*

↳ Automated proof generation ☺

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin's precomputation phase

Premises inspection of **all rules**

Facts

Fact $\xrightarrow{\text{precomp.}}$ **set** of ***possible sources***

Source: combination of **rules** ⟶ **Fact** obtainment

Raw *sources* | **Refined** *sources*

Automated proof generation ☺

For some rules: Tamarin is **unable** to **get the origin** of a **fact** → ***partial deconstruction*** left in the raw sources.

Automated proof generation ⚠

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin's precomputation phase

Premises inspection of **all rules**

Facts

Fact $\xrightarrow{\text{precomp.}}$ **set** of *possible* ***sources***

Source: combination of **rules** ⟶ **Fact** obtainment

Raw *sources*

Automated proof generation ☺

For some rules: Tamarin is **unable** to **get the origin** of a **fact** → ***partial deconstruction*** left in the raw sources.

Automated proof generation ⚠

$\xrightarrow{\text{mitigation}}$

Refined *sources*

sources lemmas

modelling tricks

auto-sources

Automated proof generation ☺

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
           ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
           (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Sources

Tamarin's precomputation phase

Premises inspection of **all rules**

↑ **Facts**

Fact $\xrightarrow{\text{precomp.}}$ **set** of ***possible sources***

Source: combination of **rules** ⟶ **Fact** obtainment

Raw *sources* | **Refined** *sources*

Raw sources: ↳ Automated proof generation ☺

For some rules: Tamarin is **unable** to **get the origin** of a **fact** → ***partial deconstruction*** left in the raw sources.

mitigation ⟶ sources lemmas, modelling tricks, auto-sources

Automated proof generation ⚠

Refined sources: ↳ Automated proof generation ☺

https://tamarin-prover.github.io/manual/master/book/009_precomputation.html

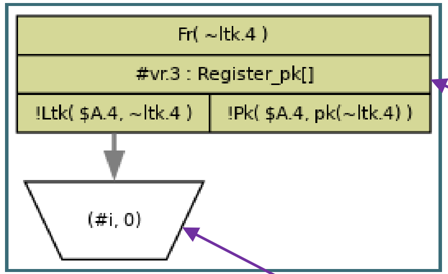
Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
```

Refined sources

Sources of "!Ltk(t.1, t.2) ▶₀ #i" (1 cases)

Source 1 of 1 / named "Register_pk"



"!Ltk(t.1, t.2) ▶₀ #i"

```
last: none

formulas:

equations:
  subst:
    $A.4 <~ {t.1}
    ~ltk.4 <~ {t.2}
  conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:

solved goals:
  !Ltk( $A.4, ~ltk.4 ) ▶₀ #i // nr: 0" (useful2)"
```

Case distinctions

All possible sources for a **fact**

Backward search

Avoid re-computations

Instance of the **`Register_pk`** rule (green box)

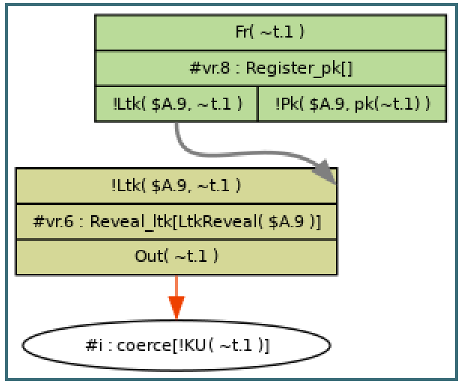
Called the **"sink"** of the `!Ltk( t.1, t.2 )` fact.

→ ⇒ Represents the origins of protocol facts (linear/persistent facts)

Using Tamarin

Sources of "!KU(~t.1) @ #i" (3 cases)

Source 1 of 3 / named "Reveal_ltk"

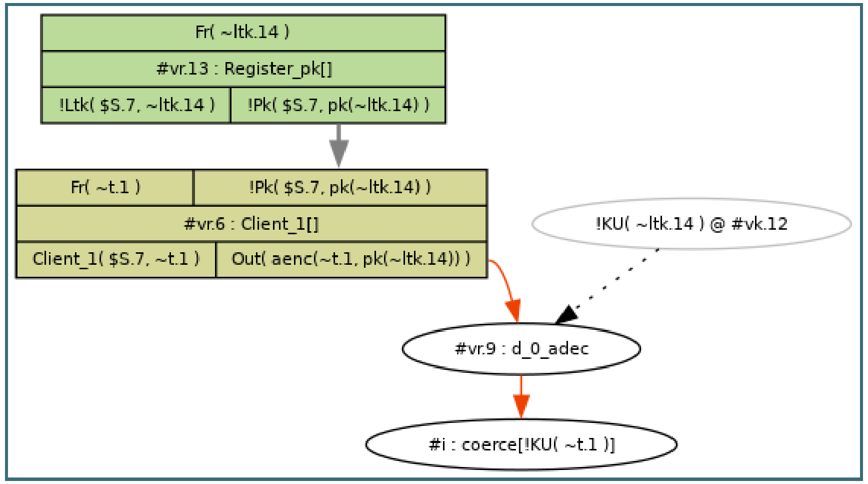


"!KU(~t.1) @ #i"

Requires a **`Register_pk`** instance

```
last: none

formulas:

equations:
  subst:
  conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:

solved goals:
  !KU( ~t.1
  ) @ #i // nr: 0 (from rule Coerce)" (currently deducible)"

  !Ltk( $A.9, ~t.1
  ) ▶₀ #vr.6 // nr: 4 (from rule Reveal_ltk)" (useful2)"
```

Source 2 of 3 / named "Client_1"



"!KU(~t.1) @ #i"

Requires a **`Register_pk`** instance

```
last: none

formulas:

equations:
  subst:
  conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:
  !KU( ~ltk.14 ) @ #vk.12 // nr: 9" (useful2)"

solved goals:
  !KU( ~t.1
  ) @ #i // nr: 0 (from rule Coerce)" (currently deducible)"

  !Pk( $S.7, pk(~ltk.14)
  ) ▶₁ #vr.6 // nr: 4 (from rule Client_1)" (useful2)"
```

Source 3 of 3 / named "fresh"

"!KU(~t.1) @ #i"

```
last: none

formulas:

equations:
  subst:
  conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:

solved goals:
  !KU( ~t.1
  ) @ #i // nr: 0 (from rule FreshConstr)" (useful2)"
```

- Represents the origins of protocol facts (linear/persistent facts)
- Represents steps where the adversary extracts value from a message he received.
- Represents an ordering constraint stemming from formulas, for example from the current lemma or a restriction.

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
```

Lemma: Client_session_key_secrecy

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

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:
∃ S k #i #j.
 (SessKeyC(S, k) @ #i) ∧ (K(k) @ #j)
∧
 ∀ #r. (LtkReveal(S) @ #r) ⇒ ⊥

equations:
 subst:
 conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:

solved goals:

0 sub-case(s)

Constraint solving

||

Refining knowledge about **property & protocol**

Property **holds in all** possible **cases**

Counterexample

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
```

Lemma: Client_session_key_secrecy

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

1. **simplify**
2. **induction**

or

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 state: empty

last: none

formulas:
∃ S k #i #j.
(SessKeyC(S, k) @ #i) ∧ (K(k) @ #j)
∧
∀ #r. (LtkReveal(S) @ #r) ⇒ ⊥

equations:
subst:
conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:

solved goals:

0 sub-case(s)

Constraint solving

||

Refining knowledge about **property & protocol**

Property **holds in all** possible **cases**

Counterexample

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
```

Lemma: Client_session_key_secrecy

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

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:
∃ S k #i #j.
(SessKeyC(S, k) @ #i) ∧ (K(k) @ #j)
∧
∀ #r. (LtkReveal(S) @ #r) ⇒ ⊥

equations:
subst:
conj:

lemmas:

allowed cases: refined

solved formulas:

unsolved goals:

solved goals:

0 sub-case(s)

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
simplify
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry
```

Visualization display

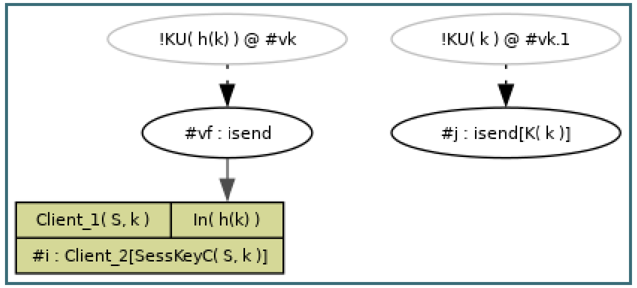
Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed)

1. **solve(** Client_1(S, k) ▶₀ #i **)** // nr. 3 (from rule Client_2)

2. **solve(** !KU(h(k)) @ #vk **)** // nr. 4 (probably constructible)

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. (LtkReveal(S) @ #r) ⇒ ⊥

equations:
subst:
conj:

lemmas:

allowed cases: refined

solved formulas:
∃ S k #i #j.
(SessKeyC(S, k) @ #i) ∧ (K(k) @ #j)
∧
∀ #r. (LtkReveal(S) @ #r) ⇒ ⊥

Searching for an execution that contains

a `SessKeyC( S, k )`
and
a `K( k )` action

The sole method for acquiring `SessKeyC(S, k)` is by using an instance of the `Client_2` rule.

`K( k )`: round box (adversary reasoning)

Proof scripts	Visualization display

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
        ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
        (¬(∃ #r. LtkReveal( S ) @ #r)))"
simplify
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry
```

Visualization display

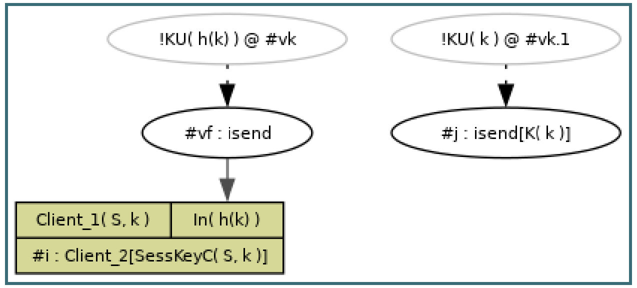
Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed)

1. **solve(** Client_1(S, k) ▶$_0$ #i **)** // nr. 3 (from rule Client_2)

2. **solve(** !KU(h(k)) @ #vk **)** // nr. 4 (probably constructible)

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



Represents the origins of protocol facts (linear/persistent facts)

Represents an ordering constraint stemming from formulas, for example from the current lemma or a restriction.

last: none

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

Contradiction

equations:
subst:
conj:

lemmas:

allowed cases: refined

solved formulas:
∃ S k #i #j.
(SessKeyC(S, k) @ #i) ∧ (K(k) @ #j)
∧
∀ #r. (LtkReveal(S) @ #r) ⇒ ⊥

Proof scripts

```
Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
simplify
solve( Client_1( S, k ) ▶₀ #i )
  case Client_1
  solve( !KU( ~k ) @ #vk.1 )
    case Client_1
    solve( !KU( ~ltk ) @ #vk.2 )
      case Reveal_ltk
      by contradiction /* from formulas */
    qed
  qed
qed

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
```

Green: success

Red: counterexample

Case: Reveal_ltk

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

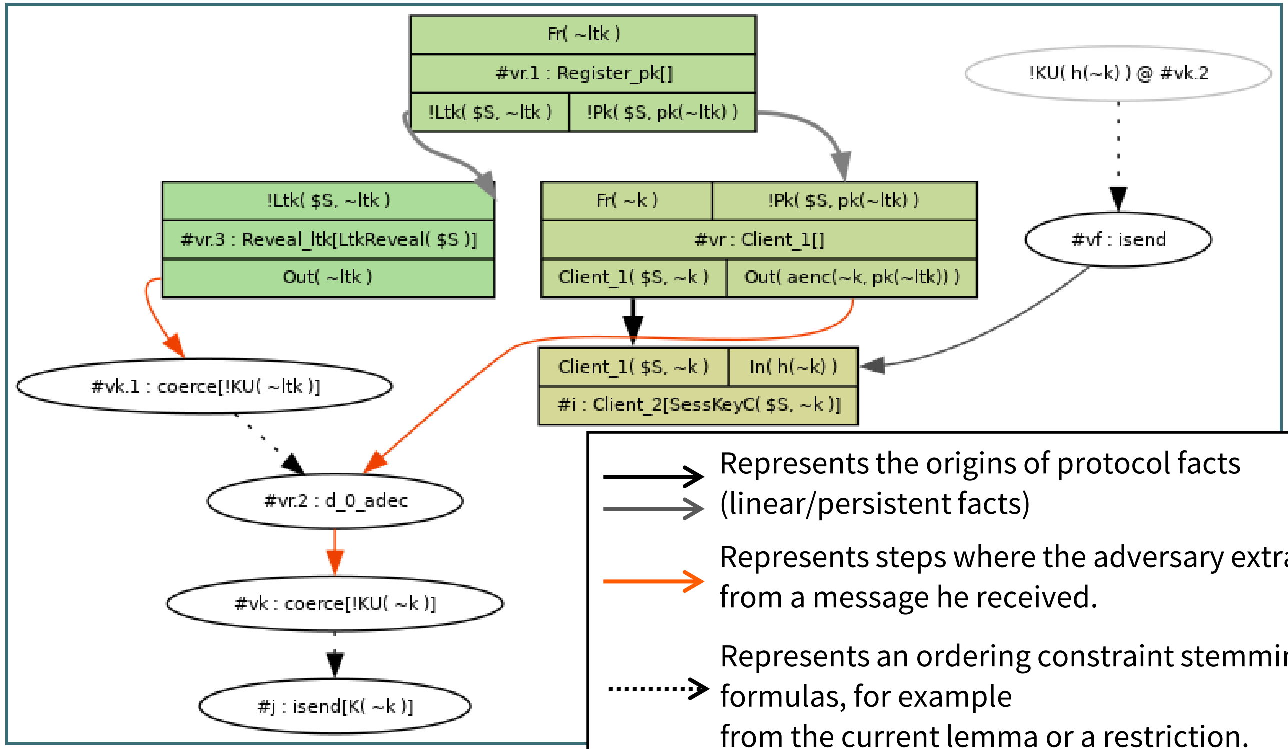
1. **contradiction** /* from formulas */

2. **solve(!KU(h(~k)) @ #vk.2)** // nr. 4

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

Autoprove or **1.** multiple times.

Constraint system



Represents the origins of protocol facts (linear/persistent facts)

Represents steps where the adversary extracts value from a message he received.

Represents an ordering constraint stemming from formulas, for example from the current lemma or a restriction.

Protocol to model (Needham-Schroeder-Lowe Public Key Protocol)

pk(I): Initiator’s public key
ni: nonce generated by I

Initiator (I)

1 `{'1',ni,I}pk(R)` →

2 ← `{'2',ni,nr,R}pk(I)`

3 `{'3',nr}pk(R)` →

Responder (R)

pk(R): Responder’s public key
nr: nonce generated by R

Adversary to consider

A **Dolev-Yao** adversary

- Controls the network
- Can delete, inject, modify and intercept messages
- *+ can dynamically compromise private keys*

Security property to prove

ni and **nr** have been sent secretly so that the adversary does not know them.

I: Initiator’s identity
R: Responder’s identity

$$I \rightarrow R: \{'1', ni, I\}_{pk(R)}$$
$$R \rightarrow I: \{'2', ni, nr, R\}_{pk(I)}$$
$$I \rightarrow R: \{'3', nr\}_{pk(R)}$$

Needham-Schroeder-Lowe Public Key Protocol

Author: **Simon Meier** | Date: **June 2012**
Source: Modeled after the description by **Paulson** in **Isabelle/HOL**/Auth/NS_Public.thy.

```
theory NSLPK3  // theory's name
begin
```

```
builtins: asymmetric-encryption
```

`aenc (2 params)`: asymmetric encryption algorithm
`adec (2 params)`: asymmetric decryption algorithm
`pk (1 param)`: public key corresponding to a private key

`adec(aenc(m, pk(sk)), sk)` is reduced to **`m`**

$$I \rightarrow R: \{'1', ni, I\}_{pk(R)}$$
$$R \rightarrow I: \{'2', ni, nr, R\}_{pk(I)}$$
$$I \rightarrow R: \{'3', nr\}_{pk(R)}$$

Creating a PKI

Registering a public key

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

$$
\begin{aligned}
I &\rightarrow R: \{'1', ni, I\}_{pk(R)} \\
R &\rightarrow I: \{'2', ni, nr, R\}_{pk(I)} \\
I &\rightarrow R: \{'3', nr\}_{pk(R)}
\end{aligned}
$$

- **Creating a PKI**

Registering a public key

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

$$I \rightarrow R: \{'1', ni, I\}_{pk(R)}$$
$$R \rightarrow I: \{'2', ni, nr, R\}_{pk(I)}$$
$$I \rightarrow R: \{'3', nr\}_{pk(R)}$$

Modelling the adversary

Dynamically compromising long-term private keys

```
rule Reveal_ltk:
  [ !Ltk(A, ltkA) ] --[ RevLtk(A)    ]-> [ Out(ltkA) ]
```

`!Ltk(A, ltkA)`: **A**’s long-term private-key **`ltkA`** database entry was read.

`RevLtk(A)`: states that **A**’s long-term private-key **`ltkA`** was compromised.

`Out(ltk)`: **A**’s long-term private-key **`ltkA`** was sent to the adversary.

Partial Deconstructions

$$I \rightarrow R : \{'1', ni, I\}_{pk(R)}$$
$$R \rightarrow I : \{'2', ni, nr, R\}_{pk(I)}$$
$$I \rightarrow R : \{'3', nr\}_{pk(R)}$$

Modelling the protocol

```
rule I_1:
  let m1 = aenc{'1', ~ni, $I}pkR
  in
    [ Fr(~ni), !Pk($R, pkR) ]
  -->
    [ Out( m1 ), St_I_1($I, $R, ~ni)]

rule R_1:
  let m1 = aenc{'1', ni, I}pk(ltkR)
      m2 = aenc{'2', ni, ~nr, $R}pkI
  in
    [ !Ltk($R, ltkR), In( m1 ),
      !Pk(I, pkI), Fr(~nr)]
  --[ Running(I, $R, <'init',ni,~nr>)]->
    [ Out( m2 ), St_R_1($R, I, ni, ~nr) ]
```

```
rule I_2:
  let m2 = aenc{'2', ni, nr, R}pk(ltkI)
      m3 = aenc{'3', nr}pkR
  in
    [ St_I_1(I, R, ni), !Ltk(I, ltkI),
      In( m2 ), !Pk(R, pkR) ]
  --[ Commit(I, R, <'init',ni,nr>),
      Running(R, I, <'resp',ni,nr>) ]->
    [ Out( m3 ), Secret(I,R,nr), Secret(I,R,ni) ]

rule R_2:
    [ St_R_1(R, I, ni, nr), !Ltk(R, ltkR),
      In( aenc{'3', nr}pk(ltkR) ) ]
  --[ Commit(R, I, <'resp',ni,nr>)]->
    [ Secret(R,I,nr), Secret(R,I,ni)    ]
```

$$
\begin{aligned}
I &\rightarrow R\colon \{'1', ni, I\}_{pk(R)} \\
R &\rightarrow I\colon \{'2', ni, nr, R\}_{pk(I)} \\
I &\rightarrow R\colon \{'3', nr\}_{pk(R)}
\end{aligned}
$$

Writing security properties

```
rule Secrecy_claim:
  [ Secret(A, B, m) ] --[ Secret(A, B, m) ]-> []
```

```
lemma nonce_secrecy:
  " /* It cannot be that */
    not(
        Ex A B s #i.
          /* somebody claims to have setup a shared secret, */
          Secret(A, B, s) @ i
          /* but the adversary knows it */
        & (Ex #j. K(s) @ j)
          /* without having performed a long-term key reveal. */
        & not (Ex #r. RevLtk(A) @ r)
        & not (Ex #r. RevLtk(B) @ r)
       )"
```

Opening the theory

Theory available at: https://github.com/tamarin-prover/manual/blob/master/code/NSLPK3.spthy

```
tamarin-prover interactive NSLPK3.spthy
```

Open your favorite web browser and go to http://127.0.0.1:3001

Partial Deconstructions

Running TAMARIN **1.8.0**

Proof scripts

```
theory NSLPK3 begin

Message theory

Multiset rewriting rules (9)

Tactic(s)

Raw sources (11 cases, 12 partial deconstructions left)

Refined sources (11 cases, 12 partial deconstructions left)

lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
          (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
           (¬(∃ #r. RevLtk( A ) @ #r))) ∧
          (¬(∃ #r. RevLtk( B ) @ #r)))"
by sorry

end
```

Tamarin's precomputation phase

Premises inspection of **all rules**

↳ **Facts**

Fact $\xrightarrow{\text{precomp.}}$ **set** of *possible* ***sources***

Source: combination of **rules** ⟶ **Fact** obtainment

Raw *sources* | **Refined** *sources*

↳ Automated proof generation ☺

For some rules: Tamarin is **unable** to **get the origin** of a **fact** → ***partial deconstruction*** left in the raw sources.

Automated proof generation ⚠

$\xrightarrow{\text{mitigation}}$

sources lemmas

modelling tricks

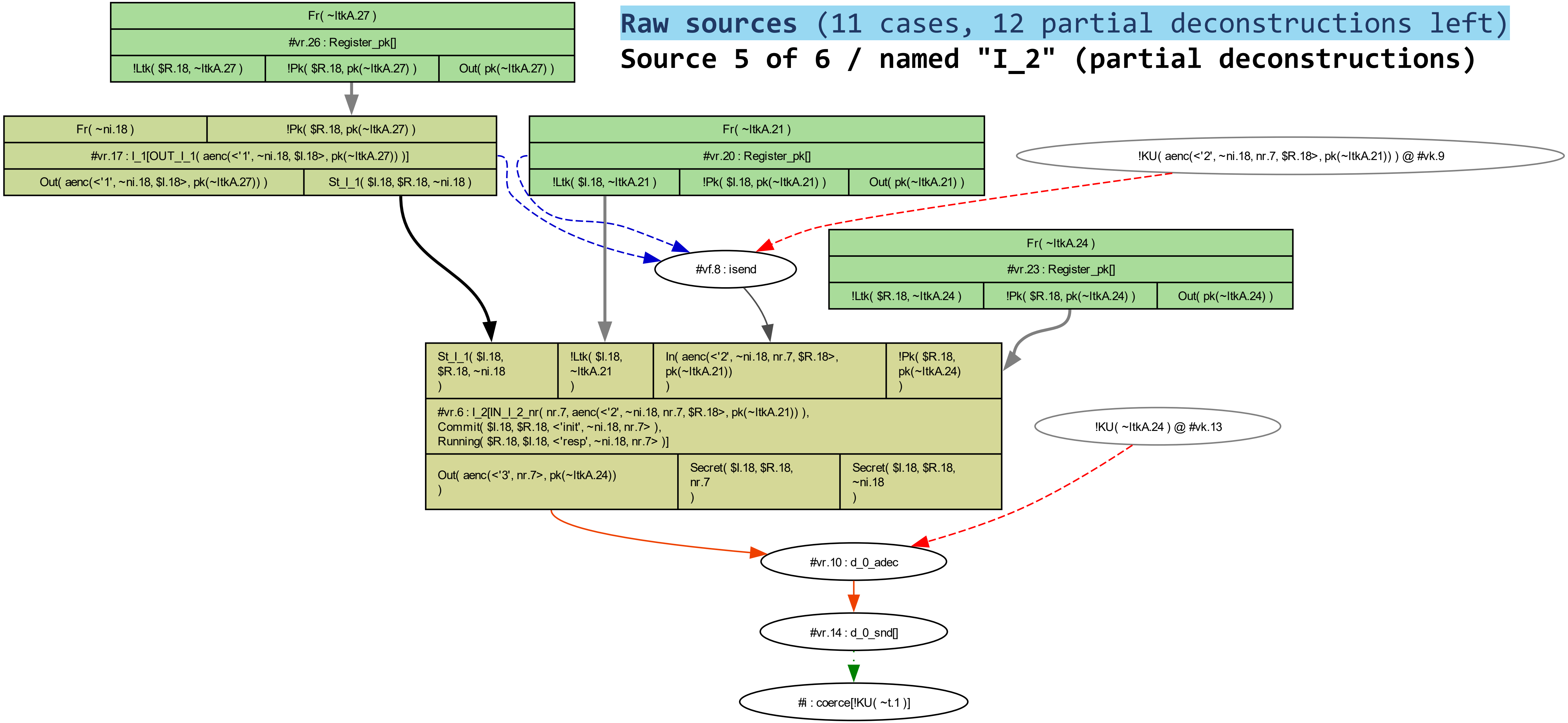
auto-sources

↳ Automated proof generation ☺

https://tamarin-prover.github.io/manual/master/book/009_precomputation.html

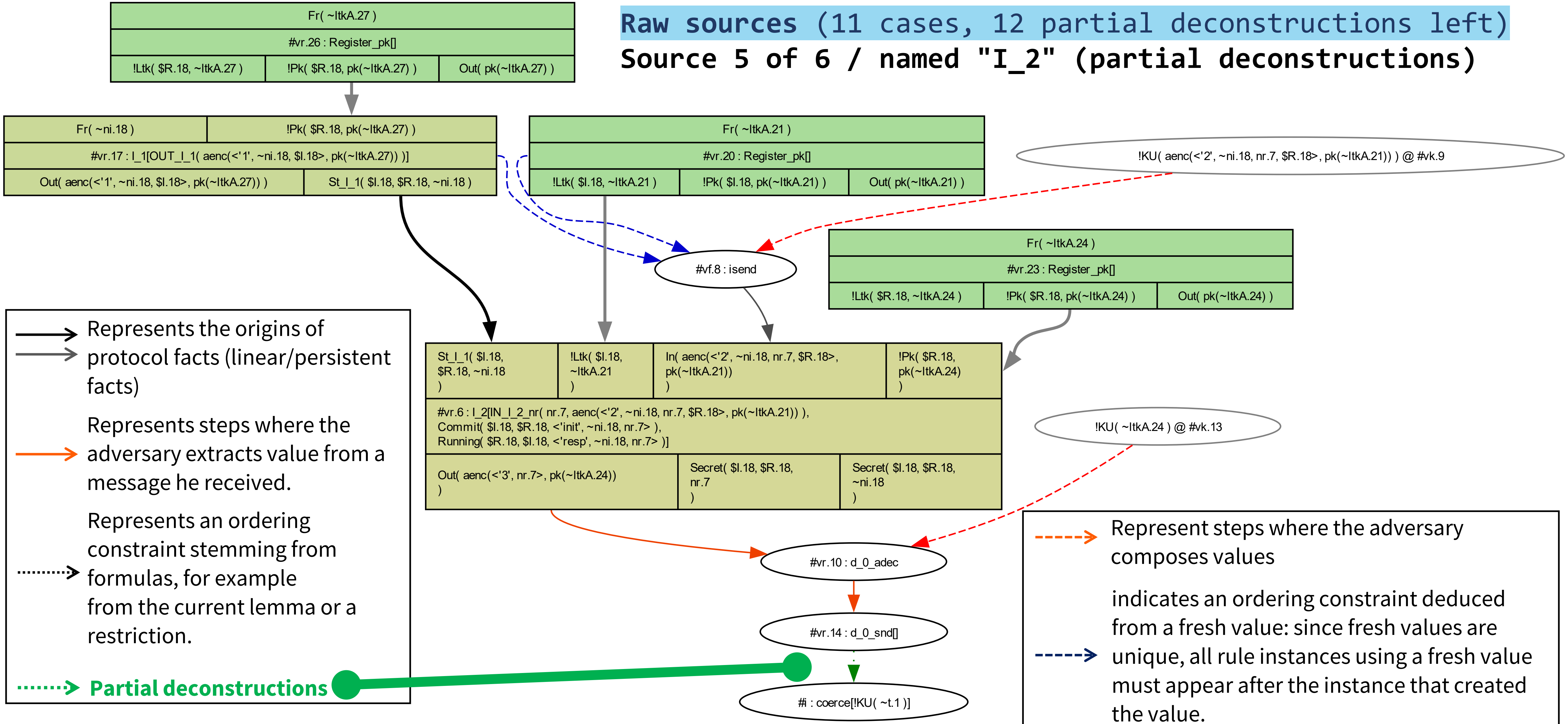
Raw sources (11 cases, 12 partial deconstructions left)

Source 5 of 6 / named "I_2" (partial deconstructions)



Raw sources (11 cases, 12 partial deconstructions left)

Source 5 of 6 / named "I_2" (partial deconstructions)



Represents the origins of protocol facts (linear/persistent facts)

Represents steps where the adversary extracts value from a message he received.

Represents an ordering constraint stemming from formulas, for example from the current lemma or a restriction.

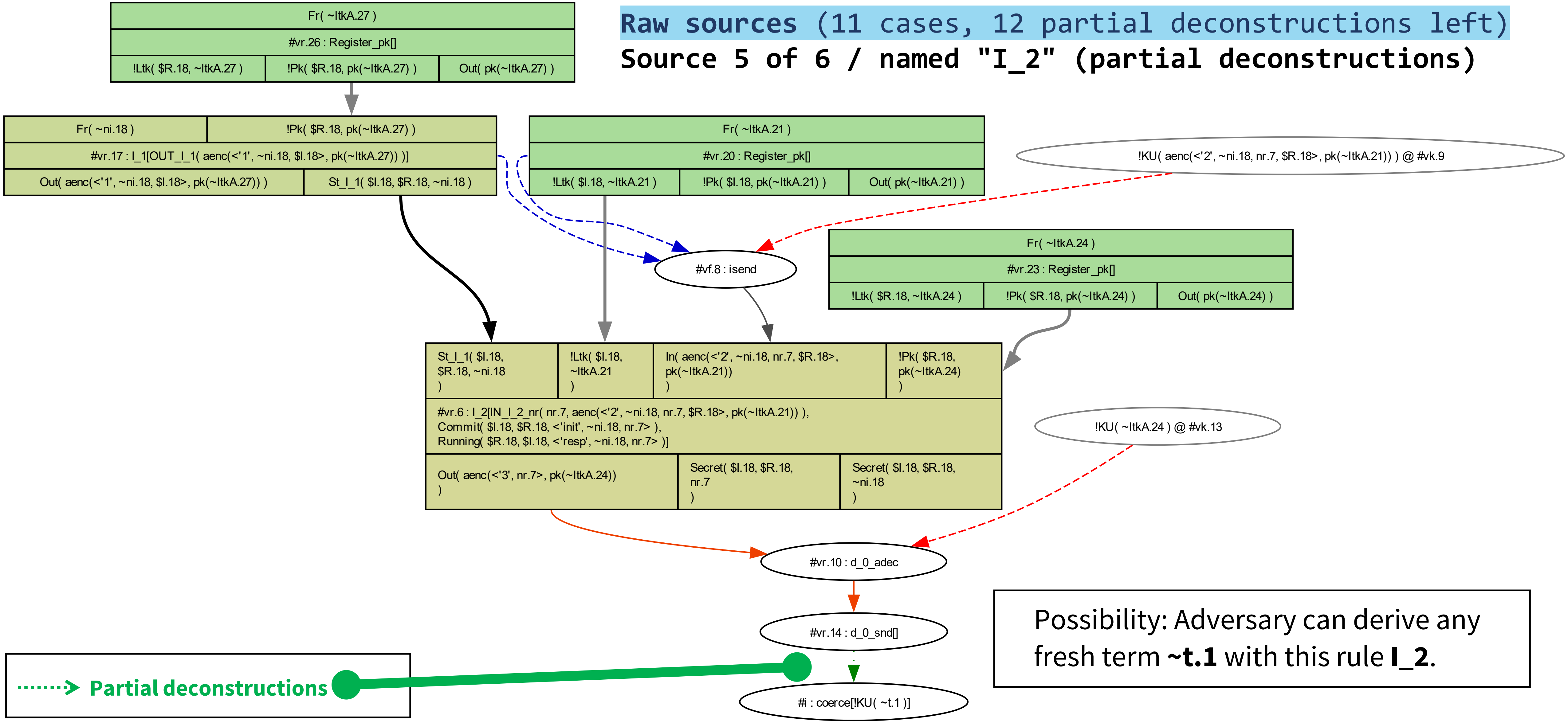
Partial deconstructions

Represent steps where the adversary composes values

indicates an ordering constraint deduced from a fresh value: since fresh values are unique, all rule instances using a fresh value must appear after the instance that created the value.

Raw sources (11 cases, 12 partial deconstructions left)

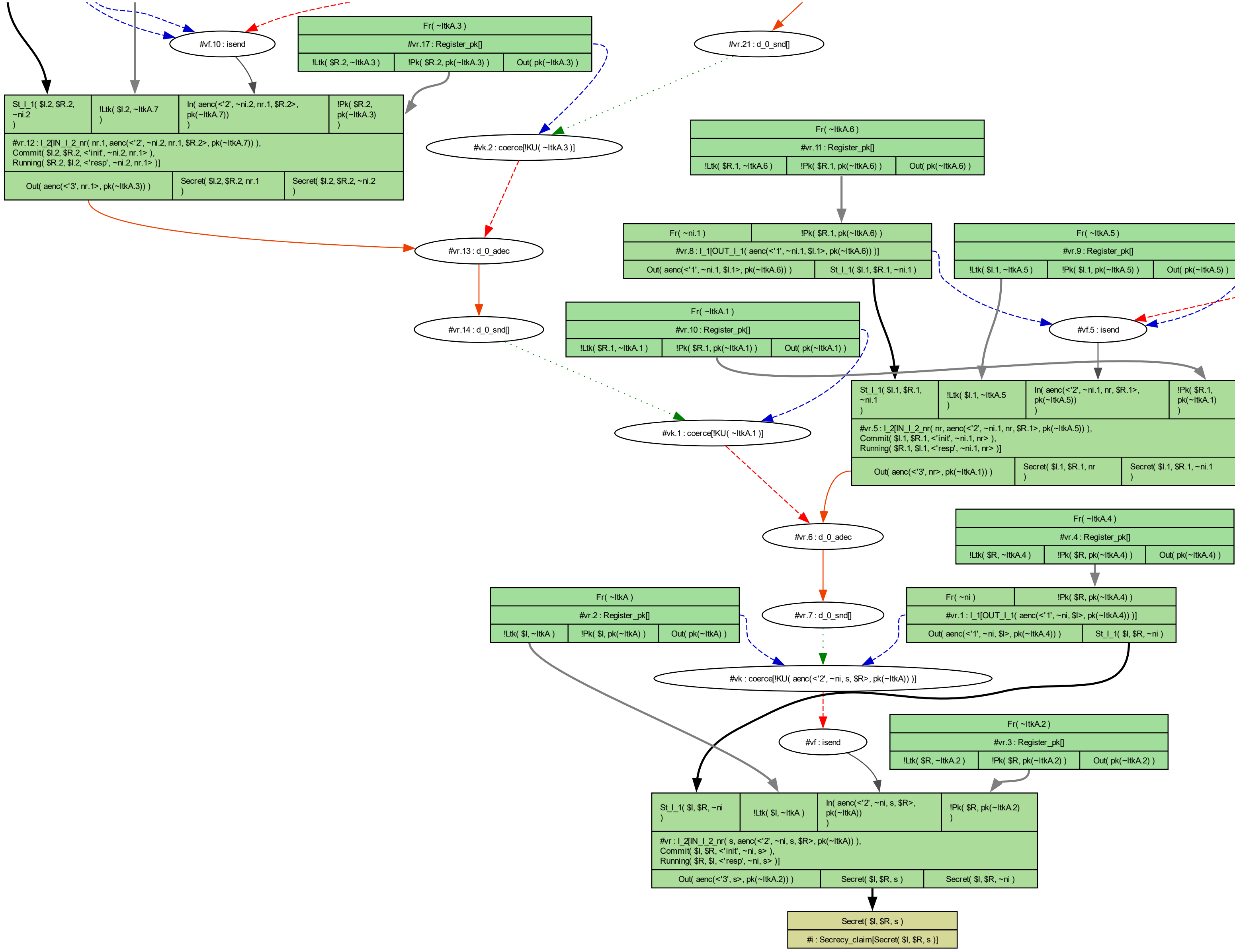
Source 5 of 6 / named "I_2" (partial deconstructions)



Possibility: Adversary can derive any fresh term **~t.1** with this rule **I_2**.

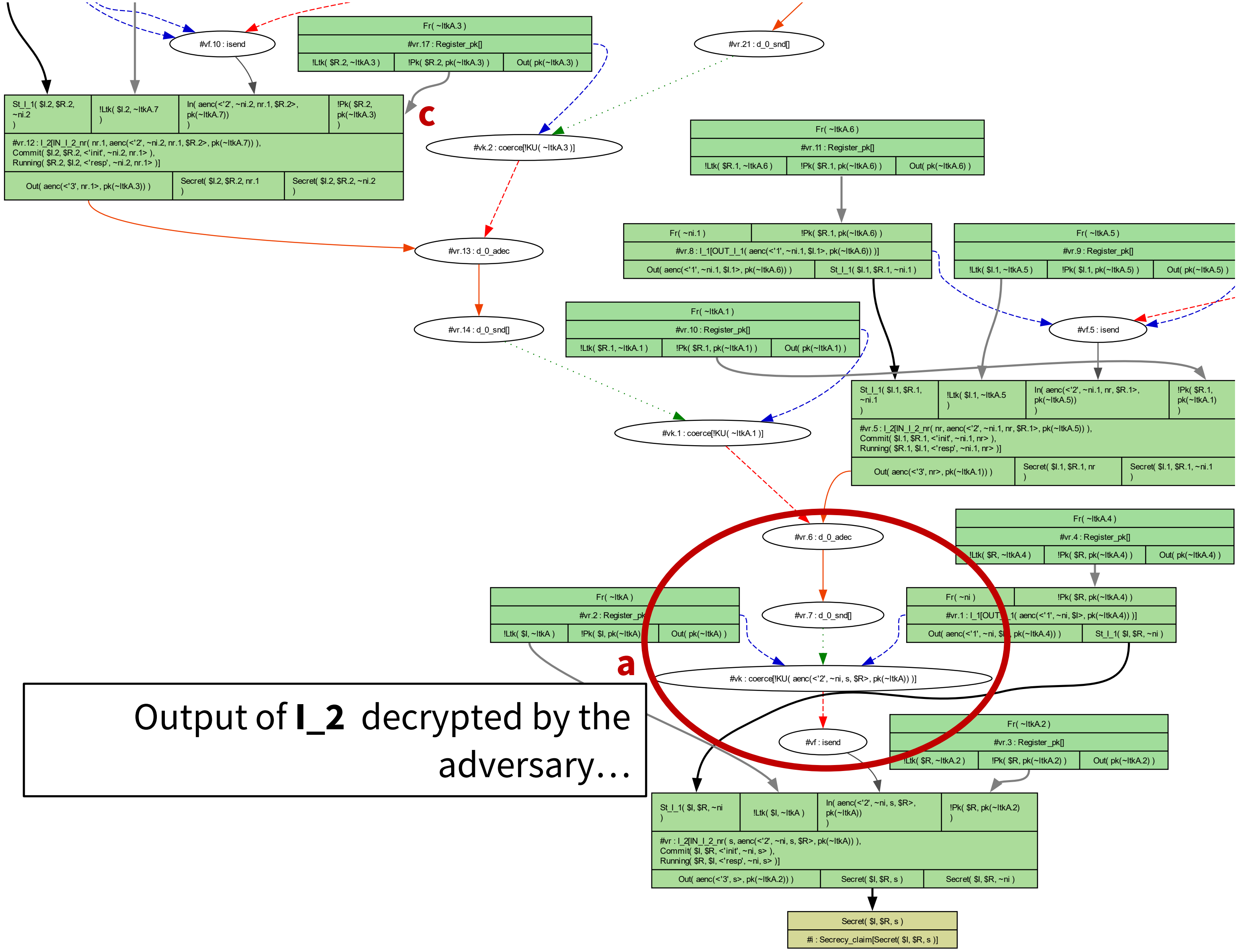
Partial Deconstructions

```
lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
          (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
           (¬(∃ #r. RevLtk( A ) @ #r))) ∧
          (¬(∃ #r. RevLtk( B ) @ #r)))"
simplify
solve( Secret( A, B, s ) ▶₀ #i )
  case I_2_case_1
  solve( !KU( aenc(<'2', ~ni, s, $R>, pk(~ltkA))
         ) @ #vk.1 )
    case I_2
    solve( !KU( ~ltkA.3 ) @ #vk.3 )
      case I_2
      solve( !KU( ~ltkA.6 ) @ #vk.5 )
        case I_2
        by sorry
      next    •••
```




```
lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
        (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
         (¬(∃ #r. RevLtk( A ) @ #r))) ∧
        (¬(∃ #r. RevLtk( B ) @ #r)))"
simplify
solve( Secret( A, B, s ) ▶₀ #i )
  case I_2_case_1
  solve( !KU( aenc(<'2', ~ni, s, $R>, pk(~ltkA))
         ) @ #vk.1 )
    case I_2
    solve( !KU( ~ltkA.3 ) @ #vk.3 )
      case I_2
      solve( !KU( ~ltkA.6 ) @ #vk.5 )
        case I_2
        by sorry
      next   •••
```

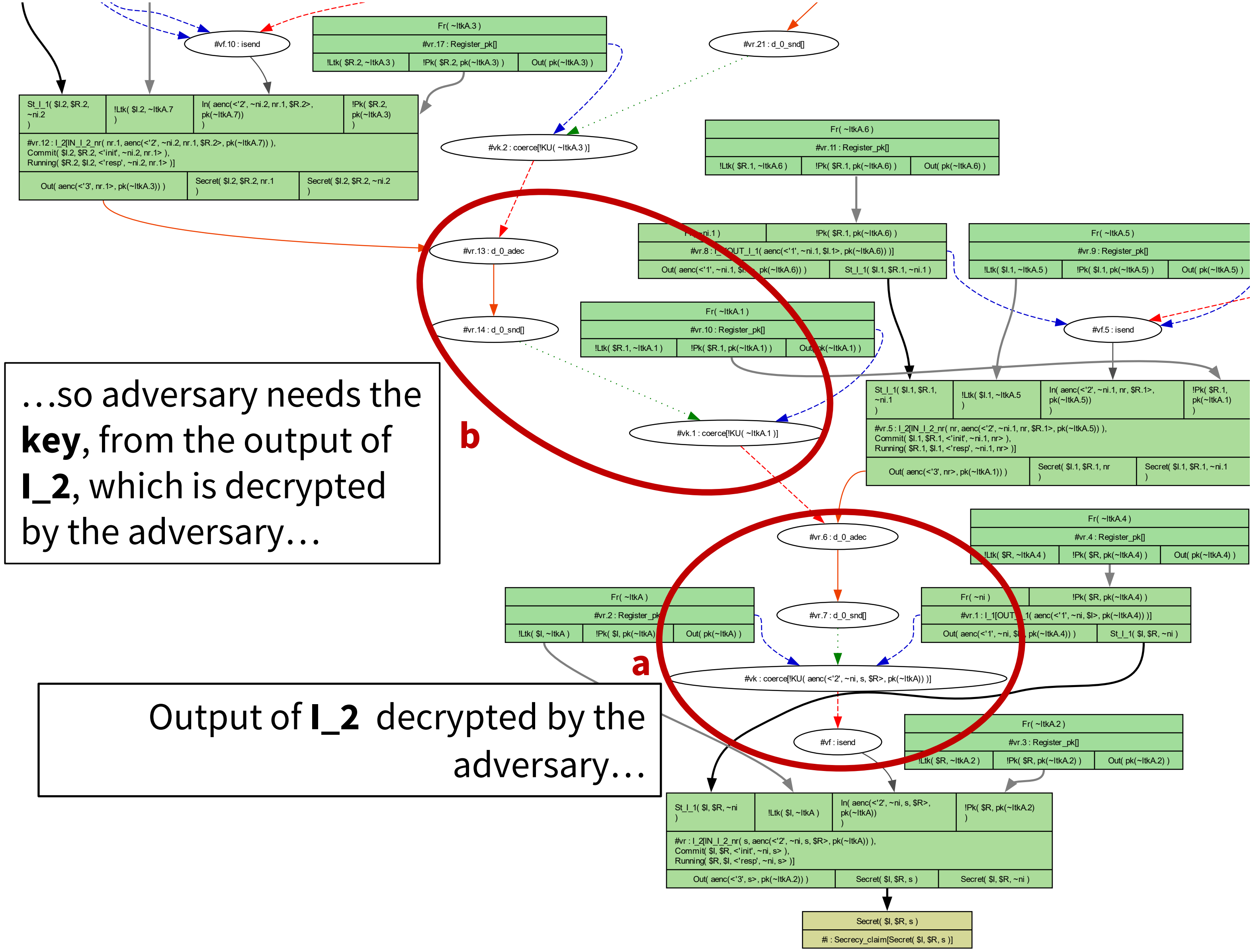
Partial Deconstructions



Output of **I_2** decrypted by the adversary…

```
lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
       (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
        (¬(∃ #r. RevLtk( A ) @ #r))) ∧
       (¬(∃ #r. RevLtk( B ) @ #r)))"
simplify
solve( Secret( A, B, s ) ▶₀ #i )
  case I_2_case_1
  solve( !KU( aenc(<'2', ~ni, s, $R>, pk(~ltkA))
         ) @ #vk.1 )
    case I_2
    solve( !KU( ~ltkA.3 ) @ #vk.3 )
      case I_2
      solve( !KU( ~ltkA.6 ) @ #vk.5 )
        case I_2
        by sorry
      next  •••
```

Partial Deconstructions



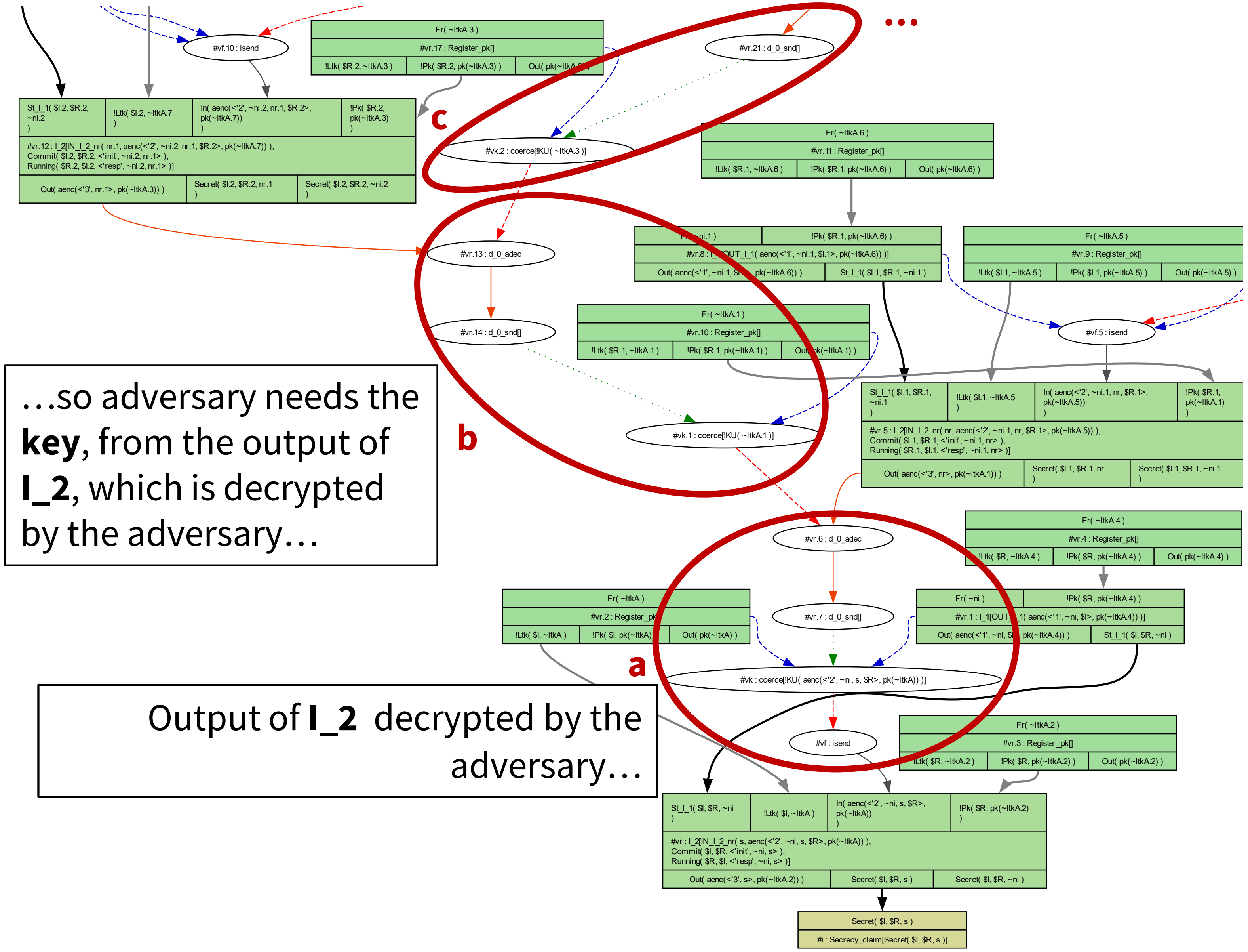
…so adversary needs the **key**, from the output of **I_2**, which is decrypted by the adversary…

Output of **I_2** decrypted by the adversary…

```
lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
       (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
        (¬(∃ #r. RevLtk( A ) @ #r))) ∧
       (¬(∃ #r. RevLtk( B ) @ #r)))"
simplify
solve( Secret( A, B, s ) ▶₀ #i )
  case I_2_case_1
  solve( !KU( aenc(<'2', ~ni, s, $R>, pk(~ltkA))
         ) @ #vk.1 )
    case I_2
    solve( !KU( ~ltkA.3 ) @ #vk.3 )
      case I_2
      solve( !KU( ~ltkA.6 ) @ #vk.5 )
        case I_2
        by sorry
      next   •••
```

Partial Deconstructions

…from the **key** that comes from **c**…

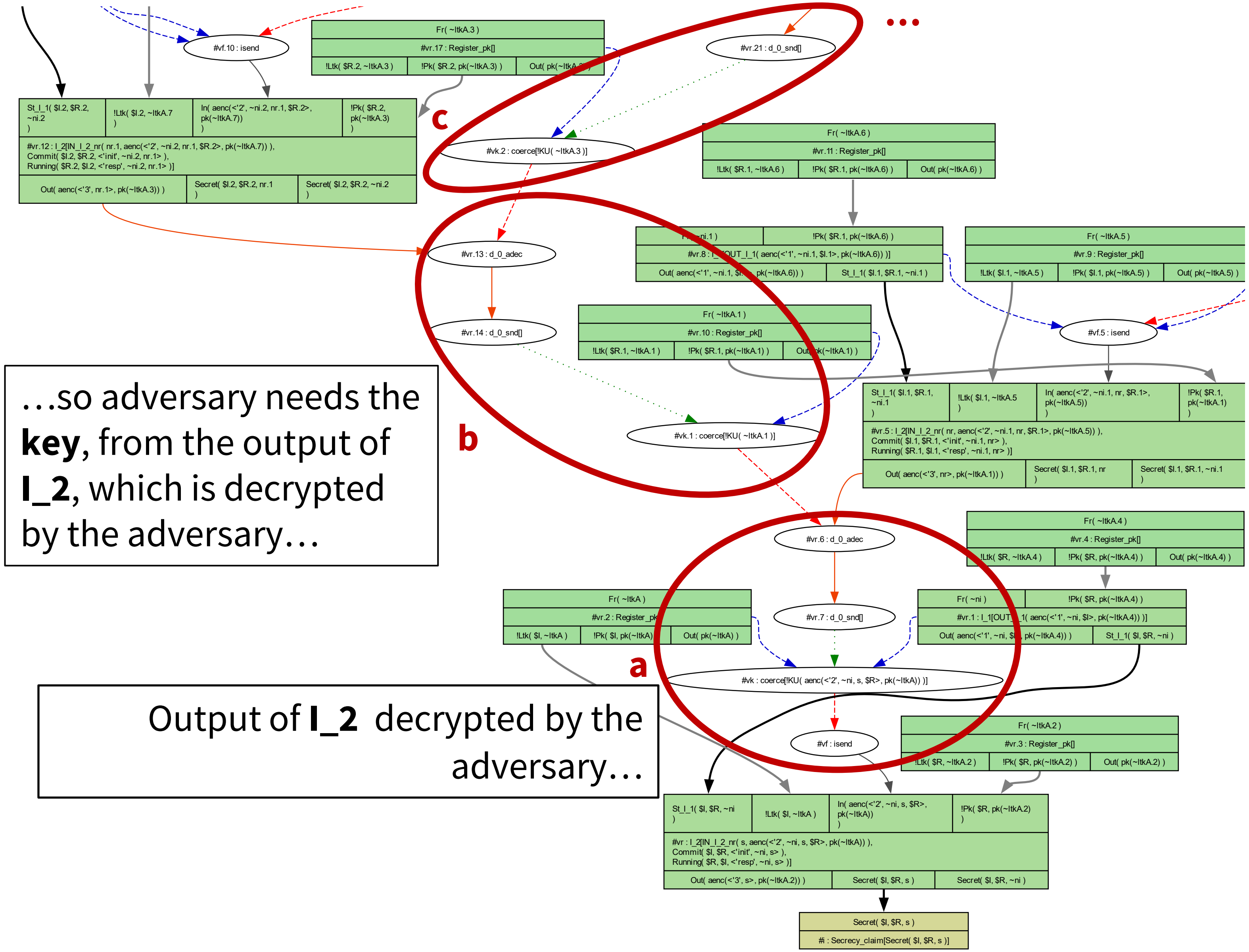


…so adversary needs the **key**, from the output of **I_2**, which is decrypted by the adversary…

Output of **I_2** decrypted by the adversary…

```
lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
       (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
        (¬(∃ #r. RevLtk( A ) @ #r))) ∧
       (¬(∃ #r. RevLtk( B ) @ #r)))"
simplify
solve( Secret( A, B, s ) ▶₀ #i )
  case I_2_case_1
  solve( !KU( aenc(<'2', ~ni, s, $R>, pk(~ltkA))
         ) @ #vk.1 )
    case I_2
    solve( !KU( ~ltkA.3 ) @ #vk.3 )
      case I_2
      solve( !KU( ~ltkA.6 ) @ #vk.5 )
        case I_2
        by sorry
      next •••
```

…from the **key** that comes from **c**…



…so adversary needs the **key**, from the output of **I_2**, which is decrypted by the adversary…

Output of **I_2** decrypted by the adversary…

```
lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
       (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
        (¬(∃ #r. RevLtk( A ) @ #r))) ∧
       (¬(∃ #r. RevLtk( B ) @ #r)))"
simplify
solve( Secret( A, B, s ) ▶₀ #i )
  case I_2_case_1
  solve( !KU( aenc(<'2', ~ni, s, $R>, pk(~ltkA))
         ) @ #vk.1 )
    case I_2
    solve( !KU( ~ltkA.3 ) @ #vk.3 )
      case I_2
      solve( !KU( ~ltkA.6 ) @ #vk.5 )
        case I_2
        by sorry
      next   •••
```

The **key does not come** from **I_2**, but Tamarin is unable to get this information. The proof **will not terminate**.

Get rid of **partial deconstructions**

The **sources lemmas** approach

$$
\begin{aligned}
I &\rightarrow R\colon \{'1', ni, I\}_{pk(R)}\\
R &\rightarrow I\colon \{'2', ni, nr, R\}_{pk(I)}\\
I &\rightarrow R\colon \{'3', nr\}_{pk(R)}
\end{aligned}
$$

- **Adding some action facts**

```
rule I_1:
  let m1 = aenc{'1', ~ni, $I}pkR
  in
    [ Fr(~ni), !Pk($R, pkR) ]
  --[ OUT_I_1(m1) ]->
    [ Out( m1 ), St_I_1($I, $R, ~ni)]

rule R_1:
  let m1 = aenc{'1', ni, I}pk(ltkR)
      m2 = aenc{'2', ni, ~nr, $R}pkI
  in
    [ !Ltk($R, ltkR), In( m1 ),
      !Pk(I, pkI), Fr(~nr)]
  --[ IN_R_1_ni( ni, m1 ), OUT_R_1( m2 ),
      Running(I, $R, <'init',ni,~nr>)]->
    [ Out( m2 ), St_R_1($R, I, ni, ~nr) ]
```

```
rule I_2:
  let m2 = aenc{'2', ni, nr, R}pk(ltkI)
      m3 = aenc{'3', nr}pkR
  in
    [ St_I_1(I, R, ni), !Ltk(I, ltkI),
      In( m2 ), !Pk(R, pkR) ]
  --[ IN_I_2_nr( nr, m2),
      Commit(I, R, <'init',ni,nr>),
      Running(R, I, <'resp',ni,nr>) ]->
    [ Out( m3 ), Secret(I,R,nr), Secret(I,R,ni) ]

rule R_2:
    [ St_R_1(R, I, ni, nr), !Ltk(R, ltkR),
      In( aenc{'3', nr}pk(ltkR) ) ]
  --[ Commit(R, I, <'resp',ni,nr>)]->
    [ Secret(R,I,nr), Secret(R,I,ni)    ]
```

Sources lemmas

$$
\begin{aligned}
I &\rightarrow R: \{'1', ni, I\}_{pk(R)} \\
R &\rightarrow I: \{'2', ni, nr, R\}_{pk(I)} \\
I &\rightarrow R: \{'3', nr\}_{pk(R)}
\end{aligned}
$$

```
lemma types [sources]:
  " (All ni m1 #i.
      IN_R_1_ni( ni, m1) @ i
      ==>
      ( (Ex #j. KU(ni) @ j & j < i)
      | (Ex #j. OUT_I_1( m1 ) @ j)
      )
    )
  & (All nr m2 #i.
      IN_I_2_nr( nr, m2) @ i
      ==>
      ( (Ex #j. KU(nr) @ j & j < i)
      | (Ex #j. OUT_R_1( m2 ) @ j)
      )
    )
  "
```

- **Adding the sources lemma**

Precomputation phase

Proof scripts

```
theory NSLPK3 begin

Message theory

Multiset rewriting rules (9)

Tactic(s)

Raw sources (11 cases, 12 partial deconstructions left)

Refined sources (11 cases, deconstructions complete)

lemma types [sources]:
  all-traces
  "(∀ ni m1 #i.
        (IN_R_1_ni( ni, m1 ) @ #i) ⇒
        ((∃ #j. (!KU( ni ) @ #j) ∧ (#j < #i)) ∨
         (∃ #j. OUT_I_1( m1 ) @ #j))) ∧
      (∀ nr m2 #i.
        (IN_I_2_nr( nr, m2 ) @ #i) ⇒
        ((∃ #j. (!KU( nr ) @ #j) ∧ (#j < #i)) ∨
         (∃ #j. OUT_R_1( m2 ) @ #j)))"
by sorry

lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
         (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
          (¬(∃ #r. RevLtk( A ) @ #r))) ∧
         (¬(∃ #r. RevLtk( B ) @ #r)))"
by sorry

end
```

$$I \rightarrow R: \{'1', ni, I\}_{pk(R)}$$
$$R \rightarrow I: \{'2', ni, nr, R\}_{pk(I)}$$
$$I \rightarrow R: \{'3', nr\}_{pk(R)}$$

- Proving the **sources lemma** and the **secrecy property**

DEMO

Get rid of **partial deconstructions**

The **auto-sources** approach

Opening the theory with the **--auto-sources** option

Theory available at: https://github.com/tamarin-prover/manual/blob/master/code/NSLPK3.spthy

```
tamarin-prover interactive --auto-sources NSLPK3.spthy
```

Open your favorite web browser and go to http://127.0.0.1:3001

Simon Meier, *Advancing automated security protocol verification*, PhD Thesis, ETH Zürich, Switzerland, 2013. doi: 10.3929/ethz-a-009790675.

Véronique Cortier, Stéphanie Delaune, Jannik Dreier, Elise Klein. Automatic generation of sources lemmas in TAMARIN: towards automatic proofs of security protocols. *Journal of Computer Security*, 2022, 30 (4), pp.573-598. ⟨10.3233/JCS-210053⟩. ⟨hal-03767104⟩

```
theory NSLPK3 begin

Message theory

Multiset rewriting rules (9)

Tactic(s)

Raw sources (11 cases, 12 partial deconstructions left)

Refined sources (11 cases, deconstructions complete)

lemma nonce_secrecy:
  all-traces
  "¬(∃ A B s #i.
          (((Secret( A, B, s ) @ #i) ∧ (∃ #j. K( s ) @ #j)) ∧
           (¬(∃ #r. RevLtk( A ) @ #r))) ∧
          (¬(∃ #r. RevLtk( B ) @ #r)))"
by sorry

lemma AUTO_typing [sources]:
  all-traces
  "((⊤) ∧
        (∀ x m #i.
          (AUTO_IN_TERM_1_0_0_1_0__R_1( m, x ) @ #i) ⇒
          ((∃ #j. (!KU( x ) @ #j) ∧ (#j < #i)) ∨
           (∃ #j.
             (AUTO_OUT_TERM_1_0_0_1_0__R_1( m ) @ #j) ∧ (#j < #i))))) ∧
       (∀ x m #i.
         (AUTO_IN_TERM_2_0_0_1_1_0__I_2( m, x ) @ #i) ⇒
         ((∃ #j. (!KU( x ) @ #j) ∧ (#j < #i)) ∨
          (∃ #j.
            (AUTO_OUT_TERM_2_0_0_1_1_0__I_2( m ) @ #j) ∧ (#j < #i))))"
by sorry

end
```

[Modelling tricks](https://tamarin-prover.com/manual/master/book/009_precomputation.html)

⚠ Variants ⚠

More info at: https://tamarin-prover.com/manual/master/book/009_precomputation.html

Website: https://tamarin-prover.com/

User manual: https://tamarin-prover.com/manual/

Teaching materials: https://github.com/tamarin-prover/teaching

Google Groups: https://groups.google.com/g/tamarin-prover

Source code: https://github.com/tamarin-prover/tamarin-prover

Research materials: https://tamarin-prover.com#research_papers_and_theses

Thank you for your attention

Questions?

Appendix A1 – Foundations of Tamarin

Foundations of Tamarin

E: an equational theory that defines cryptographic operators
R: a protocol
φ: a formula that define a trace property

Validity or satisfiability of φ for the traces of R modulo E.

Validity checking reduced to checking the satisfiability of the negated formula.

S. Meier, *Advancing automated security protocol verification*, PhD Thesis, ETH Zürich, Switzerland, 2013. doi: 10.3929/ethz-a-009790675.

B. Schmidt, *Formal analysis of key exchange protocols and physical protocol* , PhD Thesis, ETH Zürich, 2012. doi: 10.3929/ethz-a-009898924.

H. Comon-Lundh and S. Delaune, *The Finite Variant Property: How to Get Rid of Some Algebraic Properties*, Term Rewriting and Applications, J. Giesl, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 294–307. doi: 10.1007/978-3-540-32033-3_22. https://www-verimag.imag.fr/~lakhnech/CRYPTO/05-06-PAPIERS-POUR-ETUDIANTS/Crypto-et-Protocoles/rta05-CD.pdf

V. Cortier, S. Delaune, J. Dreier, and E. Klein, *Automatic generation of sources lemmas in Tamarin: Towards automatic proofs of security protocols*, JCS, vol. 30, no. 4, pp. 573–598, 2022, doi: 10.3233/JCS-210053. https://hal.science/hal-03767104/

Appendix A2 – Simple Encrypted Communication Client authentication lemmas

$C \to S: \{k\}_{pkS}$
$S \to C: h(k)$

Modelling the protocol – server side

```
// A server thread answering in one-step to a session-key setup request from
// some client.
rule Serv_1:
    [ !Ltk($S, ~ltkS)                            // lookup the private-key
    , In( request )                              // receive a request
    ]
  --[ AnswerRequest($S, adec(request, ~ltkS)) ]->
    [ Out( h(adec(request, ~ltkS)) ) ]           // Return the hash of the
                                                 // decrypted request.
```

`AnswerRequest($S, adec(request, ~ltkS))`: **Logging** of the session-key setup requests.

→ Client's **authentication** prop.

Reminder

Variables

- `~x` denotes `x:fresh`
- `$x` denotes `x:pub`
- `#i` denotes `i:temporal`
- `m` denotes `m:msg`
- `'c'` denotes a **public name** in `pub`, global constant.

Facts

- `F(t1,...,tn)` with terms `ti` and a fixed arity `n`.
- `!` denotes the persistence of a fact.
- `Fr`: built-in **fact**, denotes a freshly generated name. For modelling **nonces**/**keys**.

`Out/In` denotes a party sending (resp. receiving) a message to (from) the **untrusted** network (**Dolev-Yao**). Only right-hand (left-hand) of a multiset rewrite rule.

`-- [ACTIONFACT] ->`: facts that **do not appear in state**, but only on the **trace**. Located within the arrow.

$C \rightarrow S: \{k\}_{pkS}$
$S \rightarrow C: h(k)$

Writing security properties

Security properties are defined over **traces** of the **action facts** of a protocol execution.

```
lemma Client_auth:
  " /* For all session keys 'k' setup by clients with a server 'S' */
    ( All S k #i.  SessKeyC(S, k) @ #i
       ==>
         /* there is a server that answered the request */
       ( (Ex #a. AnswerRequest(S, k) @ a)
         /* or the adversary performed a long-term key reveal on 'S'
            before the key was setup. */
       | (Ex #r. LtkReveal(S) @ r & r < i)
       )
    )
  "
```

Client point of view – Authentication property

i < j: temporal ordering/timepoint ordering

$\forall S,k,i.\ (SessKeyC(S,k)\ @\ i) \Rightarrow ($
$\quad (\exists a.\ AnswerRequest(S,k)\ @\ a)$
$\quad \vee$
$\quad (\exists r.\ LtkReveal(S)\ @\ r \wedge r < i)$
$)$

True if it **holds** on **all traces**

$C \rightarrow S: \{k\}_{pkS}$
$S \rightarrow C: h(k)$

Writing security properties

Security properties are defined over **traces** of the **action facts** of a protocol execution.

```
lemma Client_auth_injective:
  " /* For all session keys 'k' setup by clients with a server 'S' */
    ( All S k #i.  SessKeyC(S, k) @ #i
       ==>
         /* there is a server that answered the request */
       ( (Ex #a. AnswerRequest(S, k) @ a
           /* and there is no other client that had the same request */
           & (All #j. SessKeyC(S, k) @ #j ==> #i = #j)
       )
         /* or the adversary performed a long-term key reveal on 'S'
            before the key was setup. */
       | (Ex #r. LtkReveal(S) @ r & r < i)
       )
    )
  "
```

$$
\begin{aligned}
&\forall S,k,i.\ (\mathrm{SessKeyC}(S,k)\ @\ i) \Rightarrow (\\
&\quad (\exists a,j.\ \mathrm{AnswerRequest}(S,k)\ @\ a\\
&\qquad \wedge\ \mathrm{SessKeyC}(S,k)\ @\ j\\
&\qquad \wedge\ i < j\\
&\quad)\\
&\quad \vee\\
&\quad (\exists r.\ \mathrm{LtkReveal}(S)\ @\ r\\
&\qquad \wedge\ r < i\\
&\quad)\\
&)
\end{aligned}
$$

True if it **holds** on **all traces**

Client point of view – Injective authentication property based on uniqueness

Appendix A3 – Message theory (detailed)

Proof scripts

```
theory FirstExample begin

Message theory

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
        ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
        (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Adversary

Message theory

Signature

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

User-defined or from the used built-in functions.

To access 1st part of pairs

To create pairs

2nd part of pairs

+ pair, fst, snd /arity

automatically imported

Shorthand using < >

e.g. snd(<x.1, x.2>)

Construction Rules

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]

rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```

Appendix A3 – Message theory (detailed)

Proof scripts

```
theory FirstExample begin

Message theory    Adversary

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Message theory

Signature

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

Construction Rules

Describe the adversary's **applicable functions**.

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]
```

If adv. **knows** x…
…adv. can **compute** h(x).

```
rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```

Appendix A3 – Message theory (detailed)

Proof scripts

```
theory FirstExample begin

Message theory
Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
           ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
           (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
         (SessKeyC( S, k ) @ #i) ⇒
         ((∃ #a.
            (AnswerRequest( S, k ) @ #a) ∧
            (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
          (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
         (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Adversary

Message theory

Signature

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

Construction Rules

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]

rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

Describe the adversary's **applicable functions**.

`--[(!KU( h( x ) )]->`: *security properties are defined over traces of the action facts (need to be recorded).*

If adv. **knows x** `(!KU( x ))`…

…adv. can **compute h(x)** `(!KU( h( x ) )`.

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```

Appendix A3 – Message theory (detailed)

Proof scripts

```
theory FirstExample begin

Message theory  Adversary

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
         ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
         (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Message theory

Signature

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

Construction Rules

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]

rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

Describe the adversary's **extractable terms** from larger terms by using functions.

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```

Appendix A3 – Message theory (detailed)

Proof scripts

```
theory FirstExample begin

Message theory   Adversary

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
          ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
          (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Message theory

Signature

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

Construction Rules

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]

rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

If adv. **knows** `<x.1, x.2>`…

…adv. can **extract** `x.2` **by using the** `snd` **function and the equation** `snd(<x.1, x.2>) = x.2`.

Describe the adversary's **extractable terms** from larger terms by using functions.

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```

Appendix A3 – Message theory (detailed)

Proof scripts

```
theory FirstExample begin

Message theory  Adversary

Multiset rewriting rules (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

lemma Client_session_key_secrecy:
  all-traces
  "¬(∃ S k #i #j.
         ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
         (¬(∃ #r. LtkReveal( S ) @ #r)))"
by sorry

lemma Client_auth:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_auth_injective:
  all-traces
  "∀ S k #i.
        (SessKeyC( S, k ) @ #i) ⇒
        ((∃ #a.
           (AnswerRequest( S, k ) @ #a) ∧
           (∀ #j. (SessKeyC( S, k ) @ #j) ⇒ (#i = #j))) ∨
         (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
by sorry

lemma Client_session_key_honest_setup:
  exists-trace
  "∃ S k #i.
        (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @
#r))"
by sorry

end
```

Message theory

Signature

```
functions: adec/2, aenc/2, fst/1, h/1, pair/2, pk/1, snd/1
equations:
    adec(aenc(x.1, pk(x.2)), x.2) = x.1,
    fst(<x.1, x.2>) = x.1,
    snd(<x.1, x.2>) = x.2
```

Construction Rules

```
rule (modulo AC) c_adec:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( adec(x, x.1) ) ]->
   [ !KU( adec(x, x.1) ) ]

rule (modulo AC) c_aenc:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( aenc(x, x.1) ) ]->
   [ !KU( aenc(x, x.1) ) ]

rule (modulo AC) c_fst:
   [ !KU( x ) ] --[ !KU( fst(x) ) ]-> [ !KU( fst(x) ) ]

rule (modulo AC) c_h:
   [ !KU( x ) ] --[ !KU( h(x) ) ]-> [ !KU( h(x) ) ]

rule (modulo AC) c_pair:
   [ !KU( x ), !KU( x.1 ) ]
  --[ !KU( <x, x.1> ) ]->
   [ !KU( <x, x.1> ) ]

rule (modulo AC) c_pk:
   [ !KU( x ) ] --[ !KU( pk(x) ) ]-> [ !KU( pk(x) ) ]

rule (modulo AC) c_snd:
   [ !KU( x ) ] --[ !KU( snd(x) ) ]-> [ !KU( snd(x) ) ]

rule (modulo AC) coerce:
   [ !KD( x ) ] --[ !KU( x ) ]-> [ !KU( x ) ]

rule (modulo AC) pub:
   [ ] --[ !KU( $x ) ]-> [ !KU( $x ) ]
```

If adv. **knows `<x.1, x.2>`** (`!KD( <x.1, x.2> )`)…

…adv. can **extract `x.2` by using the snd function and the equation `snd(<x.1, x.2>) = x.2`** (`!KD( x.2 )`).

Describe the adversary's **extractable terms** from larger terms by using functions.

Deconstruction Rules

```
rule (modulo AC) d_0_adec:
   [ !KD( aenc(x.1, pk(x.2)) ), !KU( x.2 ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_fst:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.1 ) ]

rule (modulo AC) d_0_snd:
   [ !KD( <x.1, x.2> ) ] --> [ !KD( x.2 ) ]
```

Appendix A4 – Built-in features

- Hashing
- Asymmetric encryption
- Signing
- Revealing signing
- Symmetric Encryption
- Diffie-Hellmann
- Bilinear Pairing
- XOR
- Multiset
- Reliable channel

- mun
- one
- exp
- mult
- inv
- pmult
- em

More information at: https://tamarin-prover.github.io/manual/master/book/004_cryptographic-messages.html

Appendix A5 – Tamarin Logic Syntax

Mathematical Name	Logic symbol	Tamarin symbol
Universal quantification	∀, ()	`All`
Existential quantification	∃	`Ex`
Implication	⇒, →, ⊃	`==>`
Conjunction	∧, ·, &	`&`
Disjunction	∨, +, ‖	`|`
Negation	¬, ~, !	`not`
Action constraint		`f @ i, f @ #i`
Temporal ordering		`i < j, #i < #j`
Equality between two temporal variables		`#i = #j`
Equality between two message variables		`x = y`
Syntactic sugar for instantiating a predicate Pred for the terms **t1** to **tn**		`Pred(t1,..,tn)`

More information at: https://tamarin-prover.github.io/manual/master/book/007_property-specification.html

Appendix A6 – Semantics

Semantics?

“Colorless green ideas sleep furiously” – Noam Chomsky, 1957 [18]

Syntactically well-formed

Appendix A7 – Decidability

Decidable logic → Determine the **truth** or **falsity** of any formula in the logic.

? ? But proof of correctness of a security protocol is an **undecidable problem**… ? ?

Decidability of a logic ≠ undecidability problem of the correctness of security protocol.

We need at least a decidable logic to prove properties…

“May not terminate as correctness of security protocol is an **undecidable problem** [13].”

References

[1] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. *A Comprehensive Symbolic Analysis of TLS 1.3*. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, New York, NY, USA, 1773–1788. doi: 10.1145/3133956.3134063.
[2] V. Stettler, *Formally Analyzing the TLS 1.3 proposal*, Bachelor thesis, ETH Zürich, Switzerland, 2016. https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/TLS-1.3_thesis_vincent_stettler.pdf
[3] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, *A Comprehensive Symbolic Analysis of TLS 1.3*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17. New York, NY, USA, 2017, pp. 1773–1788. doi: 10.1145/3133956.3134063.
[4] D. Lanzenberger, *Formal Analysis of 5G Protocols*, Bachelor thesis, ETH Zürich, Switzerland, 2017. https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/5G_lanzenberger.pdf
[5] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, *A Formal Analysis of 5G Authentication*, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18. New York, NY, USA, 2018, pp. 1383–1396. doi: 10.1145/3243734.3243846. https://arxiv.org/pdf/1806.10360
[6] C. Cremers and M. Dehnel-Wild, *Component-based formal analysis of 5G-AKA: channel assumptions and session confusion*, Network and Distributed System Security Symposium (NDSS), 2019, doi: 10.14722/ndss.2019.23394. https://ora.ox.ac.uk/objects/uuid:650ef867-79e1-476d-a602-e7e28dc64970
[7] C. Cremers, B. Kiesl, and N. Medinger, *A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters*, 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 1–17. https://www.usenix.org/conference/usenixsecurity20/presentation/cremers
[8] M. Vanhoef and F. Piessens, Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17. New York, NY, USA, Oct. 2017, pp. 1313–1328. doi: 10.1145/3133956.3134027. https://papers.mathyvanhoef.com/ccs2017.pdf
[9] V. Cheval, C. Jacomme, S. Kremer, and R. Künnemann, *SAPIC+: protocol verifiers of the world, unite!*, 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, USA, 2022, pp. 3935–3952. https://www.usenix.org/conference/usenixsecurity22/presentation/cheval
[10] S. Meier, *Advancing automated security protocol verification*, PhD Thesis, ETH Zürich, Switzerland, 2013. doi: 10.3929/ethz-a-009790675.
[11] V. Cortier, S. Delaune, J. Dreier, and E. Klein, *Automatic generation of sources lemmas in Tamarin: Towards automatic proofs of security protocols*, JCS, vol. 30, no. 4, pp. 573–598, 2022, doi: 10.3233/JCS-210053. https://hal.science/hal-03767104/
[12] B. Schmidt, *Formal analysis of key exchange protocols and physical protocol*, PhD Thesis, ETH Zürich, 2012. doi: 10.3929/ethz-a-009898924.

References

[13] F. Ţiplea, C. Enea, and C. V. Bîrjoveanu, *Decidability and Complexity Results for Security Protocols*, Verification of Infinite-State Systems with Applications to Security, 2005. https://www.semanticscholar.org/paper/Decidability-and-Complexity-Results-for-Security-%C5%A2iplea-Enea/c41604e4eacbb922593848de31576160407b6d4e

[14] D. Jackson, *Improving automated protocol verification: real world cryptography*, PhD Thesis, University of Oxford, 2020. https://ora.ox.ac.uk/objects/uuid:28fb885e-5113-438e-bfa2-439babaee563

[15] D. Basin, J. Dreier, and R. Sasse, Automated Symbolic Proofs of Observational Equivalence, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15. New York, NY, 2015, pp. 1144–1155. doi: 10.1145/2810103.2813662. https://hal.science/hal-01337409v2

[16] H. Comon-Lundh and S. Delaune, *The Finite Variant Property: How to Get Rid of Some Algebraic Properties*, Term Rewriting and Applications, J. Giesl, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 294–307. doi: 10.1007/978-3-540-32033-3_22. https://www-verimag.imag.fr/~lakhnech/CRYPTO/05-06-PAPIERS-POUR-ETUDIANTS/Crypto-et-Protocoles/rta05-CD.pdf

[17] S. Escobar, R. Sasse, and J. Meseguer, *Folding variant narrowing and optimal variant termination*, The Journal of Logic and Algebraic Programming, vol. 81, no. 7, pp. 898–928, 2012, doi: 10.1016/j.jlap.2012.01.002.

[18] N. Chomsky, *Syntactic structures*. Oxford, Mouton, 1957.