Automatic Security Analyses of Network Protocols with Tamarin-Prover

Introductory Talk

Eike Stadtländer

May 17, 2018
Outline

Motivation

Tamarin-Prover
  Overview
  Language and Environment State
  Demo

Goals for the Lab
The Thing with Proofs

Consider the following "proof":

\[ 1 = 1 \]

\[ \sqrt{1} = \sqrt{1} \]

\[ 1^p = 1^p \]

\[ i^1 = 1 \]

\[ i^2 = 1 \]

Thus, clearly \( 1 = 1 \).

Lesson: It is easy to make subtle mistakes in proofs which makes them difficult to verify.
Consider the following “proof”:

\[
1 = 1 \\
\sqrt{1} = \sqrt{1} \\
1 = 1 \\
\therefore 1 = 1.
\]

Lesson: It is easy to make subtle mistakes in proofs which makes them difficult to verify.
Consider the following “proof”:

\[
\begin{align*}
-1 & = 1 \\
-1 & = -1
\end{align*}
\]
The Thing with Proofs

Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}}
\]

Thus, clearly \(1 = 1\).

Lesson: It is easy to make subtle mistakes in proofs which makes them difficult to verify.
The Thing with Proofs

Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}}
\]
The Thing with Proofs

Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{i}{1} = \frac{1}{i}
\]
The Thing with Proofs

Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{i}{1} = \frac{1}{i}
\]

\[
\Rightarrow -1 = i^2 = \frac{i}{i} = 1
\]
The Thing with Proofs

Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{i}{1} = \frac{1}{i}
\]

\[
\Rightarrow -1 = i^2 = \frac{i}{i} = 1
\]

Thus, clearly \(-1 = 1\).
Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{i}{1} = \frac{1}{i}
\]

\[
\Rightarrow -1 = i^2 = \frac{i}{i} = 1
\]

Thus, clearly $-1 = 1$. 😞
The Thing with Proofs

Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{i}{1} = \frac{1}{i}
\]

\[
\Rightarrow -1 = i^2 = \frac{i}{i} = 1
\]

Thus, clearly \(-1 = 1\). 😞

**Lesson:**

It is easy to make subtle mistakes in proofs which makes them difficult to verify.
Consider the following “proof”:

\[
\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{i}{1} = \frac{1}{i}
\]

\[\Rightarrow -1 = i^2 = \frac{i}{i} = 1\]

Thus, clearly \(-1 = 1\). 😞

**Lesson:**
It is easy to make subtle mistakes in proofs which makes them difficult to verify for **humans**, at least.
Experts on Security Proofs\textsuperscript{1}

\textsuperscript{1}Slide inspired by Barthe (2014)
Experts on Security Proofs

“...In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor. [...] game-playing may play a role in the answer.”

Bellare and Rogaway 2004

1 Slide inspired by Barthe (2014)
Experts on Security Proofs$^1$

- “In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor. [...] game-playing may play a role in the answer.” Bellare and Rogaway 2004

- “We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect).” Halevi 2005

---

$^1$Slide inspired by Barthe (2014)
The Cryptographer’s Wish List

Wouldn’t it be great if we had a **machine** that
The Cryptographer’s Wish List

Wouldn’t it be great if we had a **machine** that

- can verify a proof

of statements or security properties for a given protocol.
The Cryptographer’s Wish List

Wouldn’t it be great if we had a machine that
- can verify a proof
- can complete a partial proof

of statements or security properties for a given protocol.
The Cryptographer’s Wish List

Wouldn’t it be great if we had a **machine** that

- can verify a proof
- can complete a partial proof
- can find a proof

of statements or security properties for a given protocol.
The Cryptographer’s Wish List

Wouldn’t it be great if we had a **machine** that

- can verify a proof
- can complete a partial proof
- can find a proof
- can find counter examples for disproof

of statements or security properties for a given protocol.
The Cryptographer’s Wish List

Wouldn’t it be great if we had a machine that

- can verify a proof
- can complete a partial proof
- can find a proof
- can find counter examples for disproof of statements or security properties for a given protocol.

**Goal**: Extensible framework for plug-and-play security.
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky
- ProVerif, CryptoVerif, ...
- EasyCrypt
  - e.g. "Proving the TLS Handshake Secure (as it is)" (Bhargavan et al. 2014)
- Tamarin-Prover
  - based on constraint logic
  - symbolic analysis
  - e.g. "A Comprehensive Symbolic Analysis of TLS 1.3" (Cremers et al. 2017)

Our Goal: Analyse IPSec protocol using automatic provers
Automatic Provers - A Status Quo

- Mathematics: Coq
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory

Our Goal: Analyse IPSec protocol using automatic provers
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky

- ProVerif, CryptoVerif, ...
- EasyCrypt
  - e.g. "Proving the TLS Handshake Secure (as it is)" (Bhargavan et al. 2014)

- Tamarin-Prover
  - based on constraint logic
  - symbolic analysis
  - e.g. "A Comprehensive Symbolic Analysis of TLS 1.3" (Cremers et al. 2017)

Our Goal: Analyse IPSec protocol using automatic provers
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky
- ProVerif, CryptoVerif, ...

Our Goal: Analyse IPSec protocol using automatic provers
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky
- ProVerif, CryptoVerif, ...
- EasyCrypt

Our Goal: Analyse IPSec protocol using automatic provers
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky

- ProVerif, CryptoVerif, ...

- EasyCrypt
  - e.g. “Proving the TLS Handshake Secure (as it is)”
    (Bhargavan et al. 2014)
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky
- ProVerif, CryptoVerif, ...
- EasyCrypt
  - e.g. “Proving the TLS Handshake Secure (as it is)”
    (Bhargavan et al. 2014)
- Tamarin-Prover
Automatic Provers - A Status Quo

- **Mathematics: Coq**
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky

- **ProVerif, CryptoVerif, ...**

- **EasyCrypt**
  - e.g. “Proving the TLS Handshake Secure (as it is)” (Bhargavan et al. 2014)

- **Tamarin-Prover**
  - based on constraint logic
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky

- ProVerif, CryptoVerif, ...

- EasyCrypt
  - e.g. “Proving the TLS Handshake Secure (as it is)”
    (Bhargavan et al. 2014)

- Tamarin-Prover
  - based on constraint logic
  - symbolic analysis

Our Goal: Analyse IPSec protocol using automatic provers
Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky
- ProVerif, CryptoVerif, ...
- EasyCrypt
  - e.g. “Proving the TLS Handshake Secure (as it is)” (Bhargavan et al. 2014)
- **Tamarin-Prover**
  - based on constraint logic
  - symbolic analysis
  - e.g. “A Comprehensive Symbolic Analysis of TLS 1.3” (Cremers et al. 2017)
Automatic Provers - A Status Quo

- **Mathematics: Coq**
  - based on homotopy type theory
  - Univalent Foundations of Mathematics, Vladimir Voevodsky

- **ProVerif, CryptoVerif, …**

- **EasyCrypt**
  - e.g. “Proving the TLS Handshake Secure (as it is)” (Bhargavan et al. 2014)

- **Tamarin-Prover**
  - based on constraint logic
  - symbolic analysis
  - e.g. “A Comprehensive Symbolic Analysis of TLS 1.3” (Cremers et al. 2017)

**Our Goal:** Analyse IPSec protocol using automatic provers
Tamarin

Brocken Inaglory, edited by Fir0002, edited by Brocken Inaglory
(https://commons.wikimedia.org/wiki/File:Tamarin_portrait_2_edit3.jpg)
https://creativecommons.org/licenses/by-sa/4.0/legalcode
The Cryptographer’s Wish List

Tamarin-Prover can

- verify a proof
- complete a partial proof
- find a valid proof
- find a counter example for disproving statements or security properties for a given protocol.

(Tamarin-Prover Manual, Basin et al. 2018)

However, Tamarin-Prover is not guaranteed to terminate.
The Cryptographer’s Wish List

Tamarin-Prover can

\[\times\] verify a proof

of statements or security properties for a given protocol.

(Tamarin-Prover Manual, Basin et al. 2018)
The Cryptographer’s Wish List

Tamarin-Prover can

- ✗ verify a proof
- ❓ complete a partial proof

of statements or security properties for a given protocol. (Tamarin-Prover Manual, Basin et al. 2018)
The Cryptographer’s Wish List

Tamarin-Prover can

- ✗ verify a proof
- ❓ complete a partial proof
- ✓ find a valid proof

of statements or security properties for a given protocol. (Tamarin-Prover Manual, Basin et al. 2018)
The Cryptographer’s Wish List

Tamarin-Prover can

- X verify a proof
- ? complete a partial proof
- ✓ find a valid proof
- ✓ find a counter example for disproving

of statements or security properties for a given protocol.

The Cryptographer’s Wish List

Tamarin-Prover can

- x verify a proof
- ? complete a partial proof
- ✓ find a valid proof
- ✓ find a counter example for disproving

of statements or security properties for a given protocol. (Tamarin-Prover Manual, Basin et al. 2018)

However, Tamarin-Prover is not guaranteed to terminate.
The Language of Tamarin-Prover

Anatomy of Tamarin Scripts

A script for Tamarin-Prover is a text file with the extension .spthy (stands for security protocol theory).

```
theory TheoryName
begin
# stuff goes here
end
```

Constructs
- Variables, Constants
- Function symbols
- Equations
- Rules
- Axioms
- Lemmata
  - etc.

During execution, the state of Tamarin is a multiset of facts.
A script for Tamarin-Prover is a text file with the extension .spthy.
The Language of Tamarin-Prover

Anatomy of Tamarin Scripts

A script for Tamarin-Prover is a text file with the extension .spthy (stands for security protocol theory).
The Language of Tamarin-Prover
Anatomy of Tamarin Scripts

A script for Tamarin-Prover is a text file with the extension `.spthy` (stands for security protocol theory).

```plaintext
theory TheoryName
begin
  # stuff goes here
end
```
The Language of Tamarin-Prover  
Anatomy of Tamarin Scripts

A script for Tamarin-Prover is a text file with the extension `.spthy` (stands for security protocol theory).

```
theory TheoryName
begin
  # stuff goes here
end
```

Constructs

- Variables, Constants
- Function symbols
- Equations
- Rules
- Axioms
- Lemmata
The Language of Tamarin-Prover

Anatomy of Tamarin Scripts

A script for Tamarin-Prover is a text file with the extension .spthy (stands for security protocol theory).

theory TheoryName
begin
# stuff goes here
end

Constructs

- Variables, Constants
- Function symbols
- Equations
- Rules
- Axioms
- Lemmata
- etc.
A script for Tamarin-Prover is a text file with the extension `.spthy` (stands for *security protocol theory*).

```
theory TheoryName
begin
# stuff goes here
end
```

Constructs

- Variables, Constants
- Function symbols
- Equations
- Rules
- Axioms
- Lemmata
- etc.

During execution, the state of Tamarin is a **multiset of facts**.
The Language of Tamarin-Prover

Variables and Constants
The Language of Tamarin-Prover

Variables and Constants

'g' constants, e.g. DH group element
The Language of Tamarin-Prover
Variables and Constants

'g' constants, e.g. DH group element
m messages, e.g. encrypted data, plaintexts
The Language of Tamarin-Prover
Variables and Constants

'\texttt{g}' constants, e.g. DH group element
\texttt{m} messages, e.g. encrypted data, plaintexts
\texttt{~x} random variables, e.g. nonces, private keys
The Language of Tamarin-Prover
Variables and Constants

'g' constants, e.g. DH group element
m messages, e.g. encrypted data, plaintexts
~x random variables, e.g. nonces, private keys
$S$ publicly known variables, e.g. server identity
The Language of Tamarin-Prover
Variables and Constants

'g' constants, e.g. DH group element

m messages, e.g. encrypted data, plaintexts

~x random variables, e.g. nonces, private keys

$S$ publicly known variables, e.g. server identity

#i temporal variable, e.g. to determine the order in which events happened
The Language of Tamarin-Prover

Rules

```prolog
rule RuleIdentifier:
    [ Premise Facts ]
    -- [ Action Facts ] ->
    [ Conclusion Facts ]
```

The facts In(...) and Out(...) represent messages received or sent over an unprotected channel, respectively. The fact Fr(...) generates fresh variables.
rule RuleIdentifier:
  [ Premise Facts ]
  --[ Action Facts ]-->
  [ Conclusion Facts ]

# can be abbreviated by -->
rule RuleIdentifier:
  let
    key = value
    # ...
  in
  [ Premise Facts ]
  --[ Action Facts ]-->
  # can be abbreviated by -->
  [ Conclusion Facts ]
The Language of Tamarin-Prover

Rules

rule RuleIdentifier:
  let
    key = value
    # ...
  in
  [ Premise Facts ]
  --[ Action Facts ]-->
  [ Conclusion Facts ]

The facts In(...) and Out(...) represent messages received or sent over an unprotected channel, respectively.
The Language of Tamarin-Prover

Rules

rule RuleIdentifier:
  let
    key = value
    # ...
  in
  [ Premise Facts ]
  --[ Action Facts ]-->  # can be abbreviated by -->
  [ Conclusion Facts ]

The facts In(...) and Out(...) represent messages received or sent over an unprotected channel, respectively. The fact Fr(...) generates fresh variables.
Trace:

rule RuleConstant:
[ ] --> [ Fact('a') ]

rule RuleConsumer:
[ Fact('a') ] --> [ NewFact('b') ]

State (multiset of facts):
rule RuleConstant:
    [ ] --> [ Fact('a') ]
State of the Environment I
Create Something from Nothing

rule RuleConstant:
    [ ] --> [ Fact('a') ]

State
(multiset of facts):
State of the Environment I
Create Something from Nothing

Trace:

rule RuleConstant:
  [ ] --> [ Fact('a') ]

State (multiset of facts):
State of the Environment I
Create Something from Nothing

Trace: RuleConstant

rule RuleConstant:
    [ ] --> [ Fact('a') ]

State
(multiset of facts):
- Fact('a')
Trace: RuleConstant, RuleConstant

rule RuleConstant:
    [ ] --> [ Fact('a') ]

State
(multiset of facts):
- Fact('a')
- Fact('a')
State of the Environment I
Create Something from Nothing

Trace: RuleConstant, RuleConstant

rule RuleConstant:
    [ ] --> [ Fact('a') ]

rule RuleConsumer:
    [ Fact('a') ] --> [ NewFact('b') ]

State
(multiset of facts):
- Fact('a')
- Fact('a')
State of the Environment I
Create Something from Nothing

Trace: RuleConstant, RuleConstant, RuleConsumer

rule RuleConstant:
    [ ] --> [ Fact('a') ]

rule RuleConsumer:
    [ Fact('a') ] --> [ NewFact('b') ]

State
(multiset of facts):
- Fact('a')
- NewFact('b')
Tamarin-Prover’s Attack Model

There are predefined rules for the attacker, e.g.

\[ \neg K(x) \rightarrow K(x) \rightarrow \text{In}(x) \]

Tamarin implements the Dolev-Yao attack model (Dolev and Yao 1983).

- Cryptographic primitives are handled symbolically or as a black-box.
- Complete control over the network: sending, receiving messages is done by the attacker.
- Usually, access to a reveal oracle.
Tamarin-Prover’s Attack Model

There are predefined rules for the attacker.
Tamarin-Prover’s Attack Model

There are predefined rules for the attacker, e.g.

rule isend:
    [ !KU(x) ] --[ K(x) ]--> [ In(x) ]
There are predefined rules for the attacker, e.g.

\[
\text{rule isend:}
\begin{align*}
[ \neg KU(x) ] & \rightarrow [ K(x) ] \\
& \rightarrow [ In(x) ]
\end{align*}
\]

Tamarin implements the \textbf{Dolev-Yao attack model} (Dolev and Yao 1983).
Tamarin-Prover’s Attack Model

There are predefined rules for the attacker, e.g.

\[
\text{rule isend:} \\
\begin{array}{c}
[ \neg KU(x) ] \rightarrow [ K(x) ] \rightarrow [ \text{In}(x) ]
\end{array}
\]

Tamarin implements the Dolev-Yao attack model (Dolev and Yao 1983).

- Cryptographic primitives are handled symbolically or as a black-box.
Tamarin-Prover’s Attack Model

There are predefined rules for the attacker, e.g.

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

Tamarin implements the Dolev-Yao attack model (Dolev and Yao 1983).

- Cryptographic primitives are handled symbolically or as a black-box.
- Complete control over the network: sending, receiving messages is done by the attacker.
Tamarin-Prover’s Attack Model

There are predefined rules for the attacker, e.g.

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

Tamarin implements the Dolev-Yao attack model (Dolev and Yao 1983).

- Cryptographic primitives are handled symbolically or as a black-box.
- Complete control over the network: sending, receiving messages is done by the attacker.
- Usually, access to a reveal oracle
State of the Environment II

Public Channel vs. State

```
createdIdentity, getPK, irec, coerce, isend

builtins: diffie-hellman

rule CreateIdentity:
[Fr(~sk)] --> ![Id(A,~sk,‘g’^~sk)]

rule GetPk:
![Id(A,sk,pk)] --> ![Out(<A,pk>)]

State:
• ![Id($A,~sk,’g’^~sk)]
• ![KU(<A,pk>)]
• ![In(<A,pk>)]
• ![K(<A,pk>)]
```

Public Channel:
• ![<A,pk>]
rule CreateIdentity:
  [ Fr(~sk) ]
  -->
  [ !Id(A,~sk, ) ]
rule CreateIdentity:
  [ Fr(~sk) ]
  -->
  [ !Id($A,~sk,'g'~sk) ]
State of the Environment II
Public Channel vs. State

builtins: diffie-hellman

rule CreateIdentity:

\[
[ \text{Fr}(\sim sk) ]
\]

\[ \rightarrow \]

\[ [ \text{!Id}(A, \sim sk, 'g' \sim sk) ] \]
builtins: diffie-hellman

rule CreateIdentity:
  [ Fr(~sk) ]
  -->
  [ !Id($A,~sk,'g'^~sk) ]

rule GetPk:
  [ !Id(A,sk,pk) ]
  -->
  [ Out(<A, pk>) ]
State of the Environment II
Public Channel vs. State

Trace:

State:

builtins: diffie-hellman

rule CreateIdentity:

\[
\text{Fr}(\sim sk) \rightarrow \text{!Id}(\$A, \sim sk, 'g' \sim sk) \]

rule GetPk:

\[
\text{!Id}(A, sk, pk) \rightarrow \text{Out}(\langle A, pk \rangle) \]
State of the Environment II
Public Channel vs. State

Trace: CreateIdentity

builtins: diffie-hellman

rule CreateIdentity:
  [ Fr(~sk) ]
  -->
  [ !Id($A,~sk,'g'~sk) ]

rule GetPk:
  [ !Id(A,sk,pk) ]
  -->
  [ Out(<A, pk>) ]
State of the Environment II
Public Channel vs. State

Trace: CreatelIdentity, GetPk

builtins: diffie-hellman

rule CreateIdentity:
  \[ Fr(\sim sk) \]
  \[ !Id(A, sk, 'g' ^ \sim sk) \]

rule GetPk:
  \[ !Id(A, sk, pk) \]
  \[ Out(A, pk) \]

State:
- \!Id($A, \sim sk, 'g' ^ \sim sk)
- Out(<A, pk>)

Public Channel:
State of the Environment II
Public Channel vs. State

Trace: CreateIdentity, GetPk, irecv

builtins: diffie-hellman

rule CreateIdentity:

\[ Fr(\sim sk) \]  
-->  
\[ !Id(\sim sk, 'g' \sim sk) \]

rule GetPk:

\[ !Id(A, sk, pk) \]  
-->  
\[ Out(A, pk) \]

State:
- !Id($A, \sim sk, 'g' \sim sk)
- !KD(<A, pk>)

Public Channel:
- <A, pk>
State of the Environment II
Public Channel vs. State

Trace: CreatIdentity, GetPk, irecv, coerce

builtins: diffie-hellman

rule CreateIdentity:
  \[ Fr(\sim sk) \]
  \[ !Id(A,sk,'g'\sim sk) \]

rule GetPk:
  \[ !Id(A,sk,pk) \]
  \[ Out(<A,pk>) \]

State:
- !Id($A,~sk,'g'~sk)
- !KD(<A,pk>)
- !KU(<A,pk>)

Public Channel:
- <A,pk>
State of the Environment II
Public Channel vs. State

Trace: CreateIdentity, GetPk, irecv, coerce, isend

builtins: diffie-hellman

rule CreateIdentity:
  \[ \text{Fr}(\sim sk) \]
  -->
  \[ !\text{Id}(A, \sim sk, 'g' \sim sk) \]

rule GetPk:
  \[ !\text{Id}(A, sk, pk) \]
  -->
  \[ \text{Out}(A, pk) \]

State:
- \!\text{Id}(A, \sim sk, 'g' \sim sk)
- \!\text{KD}(A, pk)
- \!\text{KU}(A, pk)
- \text{In}(A, pk)
- \text{K}(A, pk) \text{ (action fact)}

Public Channel:
- \text{Out}(A, pk)
lemma LemmaIdentifier:
    exists-trace / all-traces
    "
    formula to prove
    "
lemma LemmaIdentifier:

  exists-trace / all-traces
  "

  formula to prove
  "

The formula is given in first-order logic and uses symbols such as Ex, All, =>, etc.
lemma LemmaIdentifier: 
  \textit{exists-trace} / \textit{all-traces} \\
  " \\
  \textit{formula to prove} \\
  " \\

The formula is given in first-order logic and uses symbols such as \texttt{Ex}, \texttt{All}, \texttt{=>}, etc.

\textbf{Important}: In the formula we can only access \texttt{action facts}!
Demo 😊
Goals for the Lab

• Theory of Tamarin-Prover
  • mathematical foundation, in particular
    • order-sorted term algebras
    • equational theories
    • operations: substitution, replacements, unification, matching, rewriting modulo equational theories

• How is the language of Tamarin-Prover reflecting those notions?

• What are the limitations of Tamarin-Prover?

• Practical Application
  • Implementing small toy examples to learn the language
  • Working on (parts of) the IPSec protocol
Goals for the Lab

- Theory of Tamarin-Prover
- Practical Application
Goals for the Lab

- Theory of Tamarin-Prover
  - mathematical foundation

- Practical Application
Goals for the Lab

- Theory of Tamarin-Prover
  - mathematical foundation, in particular
    - order-sorted term algebras
    - equational theories

- Practical Application

Goals for the Lab

- Theory of Tamarin-Prover
  - mathematical foundation, in particular
    - order-sorted term algebras
    - equational theories
    - operations: substitution, replacements, unification, matching, rewriting modulo equational theories

- Practical Application

  - Implementing small toy examples to learn the language
  - Working on (parts of) the IPSec protocol
Goals for the Lab

- Theory of Tamarin-Prover
  - mathematical foundation, in particular
    - order-sorted term algebras
    - equational theories
    - operations: substitution, replacements, unification, matching, rewriting modulo equational theories
  - How is the language of Tamarin-Prover reflecting those notions?

- Practical Application

  - Implementing small toy examples to learn the language
  - Working on (parts of) the IPSec protocol
Goals for the Lab

- Theory of Tamarin-Prover
  - mathematical foundation, in particular
    - order-sorted term algebras
    - equational theories
    - operations: substitution, replacements, unification, matching, rewriting modulo equational theories
  - How is the language of Tamarin-Prover reflecting those notions?
  - What are the limitations of Tamarin-Prover?

- Practical Application
Goals for the Lab

- Theory of Tamarin-Prover
  - mathematical foundation, in particular
    - order-sorted term algebras
    - equational theories
    - operations: substitution, replacements, unification, matching, rewriting modulo equational theories
  - How is the language of Tamarin-Prover reflecting those notions?
  - What are the limitations of Tamarin-Prover?

- Practical Application
  - Implementing small toy examples to learn the language
Goals for the Lab

- **Theory of Tamarin-Prover**
  - mathematical foundation, in particular
    - order-sorted term algebras
    - equational theories
    - operations: substitution, replacements, unification, matching, rewriting modulo equational theories
  - How is the language of Tamarin-Prover reflecting those notions?
  - What are the limitations of Tamarin-Prover?

- **Practical Application**
  - Implementing small toy examples to learn the language
  - Working on (parts of) the IPSec protocol
References


Thank you for your attention!