Automatic Security Analyses of Network Protocols with Tamarin-Prover

Introductory Talk

Eike Stadtländer

May 17, 2018

Outline

The Thing with Proofs

The Thing with Proofs

Consider the following “proof”:

The Thing with Proofs

Consider the following “proof”:

$$\frac{-1}{1} = \frac{1}{-1}$$

The Thing with Proofs

Consider the following “proof”:

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}}$$

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”:

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{\mathsf{i}}{1} = \frac{1}{\mathsf{i}}$$

The Thing with Proofs

Consider the following “proof”:

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{\mathsf{i}}{1} = \frac{1}{\mathsf{i}}$$
$$\Rightarrow -1 = \mathsf{i}^2 = \frac{\mathsf{i}}{\mathsf{i}} = 1$$

The Thing with Proofs

Consider the following “proof”:

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{\mathsf{i}}{1} = \frac{1}{\mathsf{i}}$$
$$\Rightarrow -1 = \mathsf{i}^2 = \frac{\mathsf{i}}{\mathsf{i}} = 1$$

Thus, clearly $-1 = 1$.

The Thing with Proofs

Consider the following "proof":

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{\mathsf{i}}{1} = \frac{1}{\mathsf{i}}$$
$$\Rightarrow -1 = \mathsf{i}^2 = \frac{\mathsf{i}}{\mathsf{i}} = 1$$

Thus, clearly $-1 = 1$. ☹

The Thing with Proofs

Consider the following "proof":

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{\mathsf{i}}{1} = \frac{1}{\mathsf{i}}$$

$$\Rightarrow -1 = \mathsf{i}^2 = \frac{\mathsf{i}}{\mathsf{i}} = 1$$

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":

$$\frac{-1}{1} = \frac{1}{-1} \Rightarrow \sqrt{\frac{-1}{1}} = \sqrt{\frac{1}{-1}} \Rightarrow \frac{\sqrt{-1}}{\sqrt{1}} = \frac{\sqrt{1}}{\sqrt{-1}} \Rightarrow \frac{\mathsf{i}}{1} = \frac{1}{\mathsf{i}}$$
$$\Rightarrow -1 = \mathsf{i}^2 = \frac{\mathsf{i}}{\mathsf{i}} = 1$$

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[1]

[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

[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

Automatic Provers - A Status Quo

- Mathematics: Coq

Automatic Provers - A Status Quo

- Mathematics: Coq
  - based on homotopy type theory

Automatic Provers - A Status Quo

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

Automatic Provers - A Status Quo

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

Automatic Provers - A Status Quo

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

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

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

The Cryptographer's Wish List

Tamarin-Prover can

- ✗ 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

- ✗ 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)

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

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

The Language of Tamarin-Prover

Anatomy of Tamarin Scripts

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*).

```
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.

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**.

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

`'g'`	constants, e.g. DH group element
`m`	messages, e.g. encrypted data, plaintexts
`~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

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

The Language of Tamarin-Prover

Rules

```
rule RuleIdentifier:
    [ 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 ]->    # can be abbreviated by -->
    [ Conclusion Facts ]
```

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 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.

State of the Environment I

Create Something from Nothing

State of the Environment I

Create Something from Nothing

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

State of the Environment I

Create Something from Nothing

State
(multiset of facts):

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

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')`

State of the Environment I

Create Something from Nothing

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

State
(multiset of facts):

- `Fact('a')`
- `NewFact('b')`

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

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

Tamarin-Prover's Attack Model

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) ]
```

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).

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.

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

State of the Environment II

Public Channel vs. State

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

State of the Environment II

Public Channel vs. State

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

State of the Environment II

Public Channel vs. State

```
builtins: diffie-hellman

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

State of the Environment II

Public Channel vs. State

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

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

Public Channel:

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:

- `!Id($A,~sk,'g'^~sk)`

Public Channel:

State of the Environment II

Public Channel vs. State

Trace: CreateIdentity, GetPk

```
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)`
- `Out(<A,pk>)`

Public Channel:

State of the Environment II

Public Channel vs. State

Trace: CreateIdentity, GetPk, irecv

```
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)`
- `!KD(<A,pk>)`

Public Channel:

- `<A,pk>`

State of the Environment II

Public Channel vs. State

Trace: CreateIdentity, GetPk, irecv, coerce

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

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

State:

- `!Id($A,~sk,'g'^~sk)`
- `!KD(<A,pk>)`
- `!KU(<A,pk>)`
- `In(<A,pk>)`
- `K(<A,pk>)` (action fact)

Public Channel:

- `<A,pk>`

The Language of Tamarin-Prover

Lemmata

```
lemma LemmaIdentifier:
    exists-trace / all-traces
    "
        formula to prove
    "
```

The Language of Tamarin-Prover

Lemmata

```
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:
    exists-trace / all-traces
    "
        formula to prove
    "
```

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

Important: In the formula we can only access action facts!

Demo ☺

Goals for the Lab

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

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

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

Gilles Barthe. *EasyCrypt - Lecture 1 - Introduction*. EasyCrypt-F*-CryptoVerif School 2014. Nov. 24, 2014. URL: https://www.easycrypt.info/trac/raw-attachment/wiki/SchoolParis14/lecture1.pdf (visited on 05/11/2018).

David Basin et al. *Tamarin-Prover Manual. Security Protocol Analysis in the Symbolic Model*. Mar. 13, 2018. URL: https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf (visited on 05/13/2018).

Karthikeyan Bhargavan et al. "Proving the TLS Handshake Secure (as it is)". In: *Advances in Cryptology – CRYPTO 2014*. Ed. by Juan A. Garay and Rosario Gennaro. Springer Berlin Heidelberg, 2014, pp. 235–255. DOI: 10.1007/978-3-662-44381-1_14. URL: https://eprint.iacr.org/2014/182 (visited on 05/13/2018).

Mihir Bellare and Phillip Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. Cryptology ePrint Archive, Report 2004/331. 2004. URL: https://eprint.iacr.org/2004/331 (visited on 05/11/2018).

Cas Cremers et al. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. ACM, 2017, pp. 1773–1788. DOI: 10.1145/3133956.3134063. URL: http://doi.acm.org/10.1145/3133956.3134063.

Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208. DOI: 10.1109/tit.1983.1056650.

Shai Halevi. *A plausible approach to computer-aided cryptographic proofs*. Cryptology ePrint Archive, Report 2005/181. 2005. URL: https://eprint.iacr.org/2005/181 (visited on 05/11/2018).

Thank you for your attention!