CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Formal Analysis of Real-World Security Protocols

Lecture 6: Using Tamarin in Practice

Aleksi Peltonen | 09.12.2024

Midterm

- Midterm inspection
  - **Date**: Friday, December 13, 2024
  - **Time**: 9:00 - 11:00
  - **Location**: E9.1, Room 0.07
  - Please make sure to **bring your student ID**
- Re-exam
  - **Date:** Friday, January 10, 2025
  - **Time:** 16:00 - 18:00
  - **Location:** E9.1, Lecture Hall
- Grading
  - Passing the midterm is a requirement for passing the course
  - The midterm accounts for 30% of your final grade

Exercises

- Six exercise sheets in total, each worth 100 points
  - Passing the course requires at least **300 points in total**
  - Beyond that, the exercise points will **not** affect your final grade
- In the remaining two exercise sheets, you will model and analyze protocols with Tamarin
  - The topics will be relevant for working on the project
  - Even if you do not submit solutions, it is highly recommended to attempt solving the exercises
- **Solutions will only be graded if they are syntactically correct**, i.e., Tamarin does not produce any errors or warnings

Tentative course plan

Lecture	Content	Other
6	Using Tamarin in Practice	Exercise 5
7	Advanced Security Properties and Threat Models	Exercise 6
	End-of-year Break (21.12.2024-5.1.2025)	
8	Advanced Features (Part 1)	Project introduction
	Re-exam (Friday 10.1.2025)	
9	Advanced Features (Part 2)	-
10	Case Studies	-
11	Recent Research and Future Work	-
12	TBD	-
	Project DL (28.2.2025)	

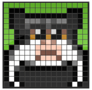
This lecture

The Tamarin Prover

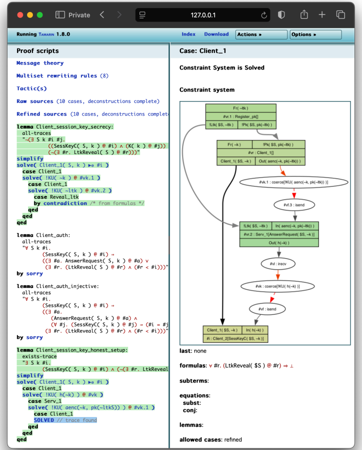
Using Tamarin in Practice

The Tamarin Prover

The Tamarin prover



- Originally developed at ETH Zürich by David Basin, Cas Cremers, Simon Meier, and Benedict Schmidt
- First version developed over 2-3 years, released in 2012
- Built on the experiences of previous tools (e.g., Scyther)
- Currently: four active maintainers and a large number of contributors

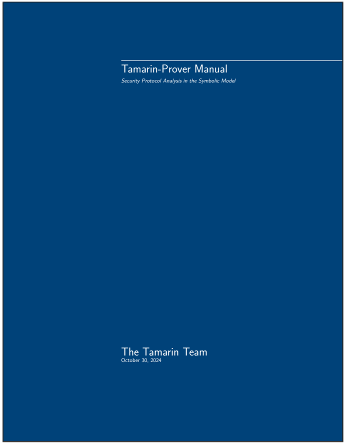


Tamarin use in the industry

Resources

Many resources available online[*]:

- Extensive **user manual** with a focus on “*explaining Tamarin’s usage so that a new user can download, install, and use the system.*”
- Teaching material from summer schools, workshops, and tutorials
- Research papers on Tamarin, its theory, extensions, and case studies


Tamarin-Prover Manual

Security Protocol Analysis in the Symbolic Model

The Tamarin Team

October 30, 2024

[*] `https://tamarin-prover.com/`

Using Tamarin in Practice

Installing Tamarin

- See instructions in the manual or in Exercise Sheet 5
- For this course, we will be using **version 1.10.0**
  - If you already installed Tamarin, check the version (`tamarin-prover --version`)
- The easiest way to install Tamarin on macOS or Linux is to use Homebrew (`brew install tamarin-prover/tap/tamarin-prover`)
- Alternatively, you can also compile Tamarin from source
- For Windows, use Windows Subsystem for Linux (WSL)
- **Office hours** to help with installation:
  - **Time:** Friday, December 13, 13:00-15:00
  - **Location:** E9.1, Room 0.07

Getting started

- Input: *.spthy (**s**ecurity **p**rotocol **th**eor**y**)
- Read Chapter 11 in the book for hints on getting started

```
/*
   Protocol:    Protocol
   Modeler:     Name(s)
   Date:        Month Year

   Status:      Working

   Description of the protocol.
*/

theory PROTOCOL
begin

/* Built-in equational theories,
   e.g., symmetric-encryption */
builtins:
/* User-defined functions and
   equational theories, e.g., */
functions: // enc/2, dec/2
equations: // dec(enc(m,k),k) = m

/** Rules **/
/** Lemmas **/

end
```

Writing models

- Tamarin does not include an editor - use your favorite text editor or IDE to write models
  - Syntax highlighting (and other features) exists for e.g., VS Code, Vim, Sublime Text 3, GNU Emacs, and Notepad++
- **Write readable models** by considering things like
  - Indentation
  - Consistent and self-explanatory naming convention
  - **Comments** to explain the model

```
rule r1:
[Fr(~a),Fr(~b)]
--[A(~a,~b)]->
[!B(~a,~b)]
```

vs.

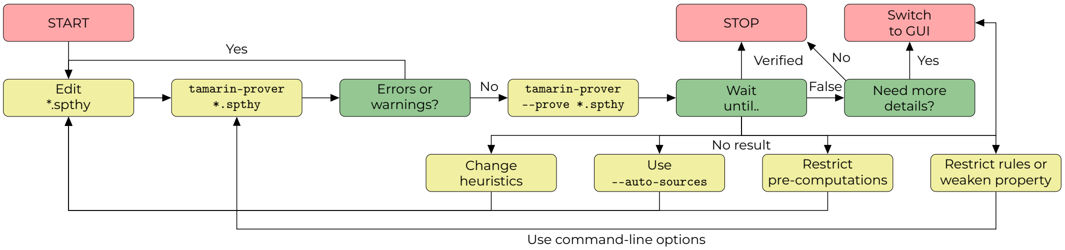
```
// Create new secret
rule derive_secret:
  [ Fr(~id), Fr(~k) ]
--[ Secret(~id, ~k) ]->
  [ !Secret(~id,~k) ]
```

Command-line interface (CLI)

- **Syntax:** `tamarin-prover *.spthy`
- Good for running large batches, bad for debugging
  - Provides less feedback about potential problems
  - Use when you think proof is working
- Prove lemmas with the argument `--prove[=LEMMA*]`
- Proofs can be exported with the argument `--output=FILE`
- Supports Haskell's runtime system (RTS) parameters `(+RTS -RTS)`

  e.g., `tamarin-prover model.spthy --prove=LEMMA +RTS -N4 -M4G -RTS`
- For a complete list of options, run `tamarin-prover --help`

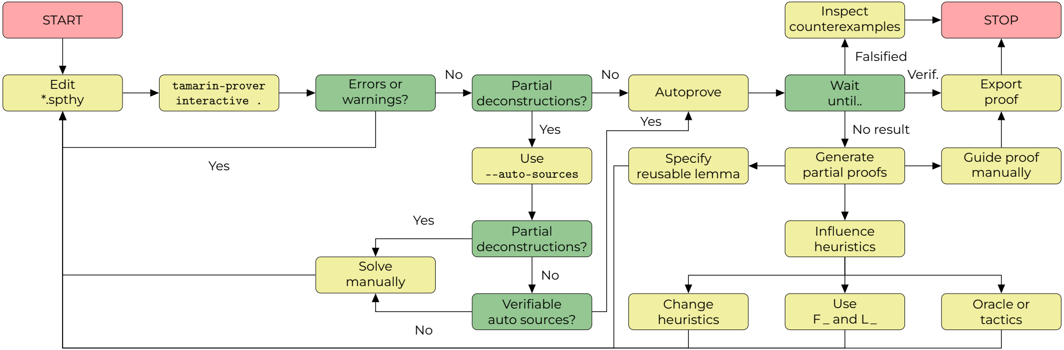
CLI workflow



Graphical user interface (GUI)

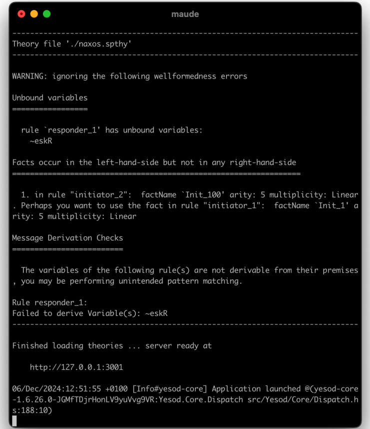
- **Syntax:** `tamarin-prover interactive [dir]`
- Loads all models in the given directory (e.g., `.` for cwd) and starts a server on port 3001
  - Access server at `http://127.0.0.1:3001`
  - Possible to change port with the argument `--port`
- Possible to run remotely: `ssh -L 3001:localhost:3001 SERVERNAME`
- No reloading; if you edit the model, you need to restart the server

GUI workflow



Syntactic correctness

- When loading a file, Tamarin will report **errors** and **wellformedness warnings**
- Most errors will not stop you from running the model, but may cause issues if not fixed
- Use the argument `--quit-on-warning` to prevent Tamarin from proceeding if there are errors
- If you do not understand an error, see Chapter 12.4 in the book



Wellformedness checks

- No `Out` or `K` facts should appear in the premises of protocol rules and no `Fr`, `In`, or `K` facts should appear in the conclusions
- All action facts used in lemmas or restrictions should appear somewhere in the rules
- Facts must have the same arity everywhere, i.e., in all rules, lemmas, and restrictions
- `Fr`, `In`, `Out`, and `K` facts must be of arity one
- `Fr` facts must be used with a variable of type message or type fresh

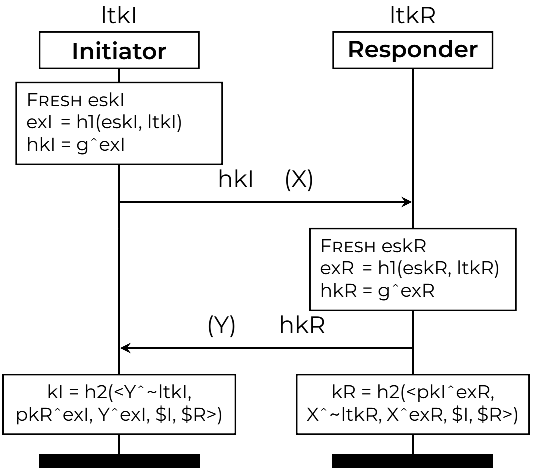
Wellformedness checks

- All lemmas must be guarded formulas
- All variables in the conclusions of a rule must appear in the premises, or be public variables
- The premises of a rule must not contain reducible function symbols such as decryption, XOR, etc
- The conclusions of a rule must not contain multiplication *

Check the book for a complete lists of errors and how to fix them!

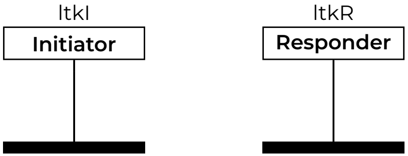
Example: NAXOS

NAXOS



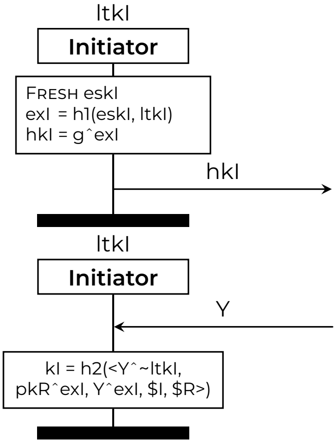
ltkI
ltkR
Initiator
Responder
FRESH eskI
exI = h1(eskI, ltkI)
hkI = g^exI
hkI (X)
FRESH eskR
exR = h1(eskR, ltkR)
hkR = g^exR
(Y) hkR
kI = h2(<Y^~ltkI, pkR^exI, Y^exI, $I, $R>)
kR = h2(<pkI^exR, X^~ltkR, X^exR, $I, $R>)

Initialization



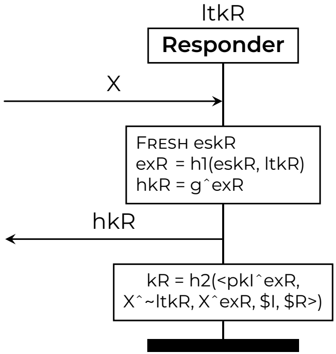

```
/* Generate long-term key-pair */
rule generate_ltk:
    let
      pkA = 'g'^~lkA
    in
    [ Fr(~lkA) ]
  --[ Register($A) ]->
    [ !Ltk($A,~lkA)
    , !Pk($A,pkA)
    , Out(pkA) ]
```

Initiator model




```
/* Initiator: send hkI */
rule initiator_1:
    let
      exI = h1(<~eskI,~ltkI>)
      hkI = 'g'^exI
    in
    [ Fr(~eskI), !Ltk($I, ~ltkI) ]
  --[ ]->
    [ Init_1(~eskI,$I,$R,~ltkI,hkI)
    , Out(hkI) ]
/* Initiator: receive hkR */
rule initiator_2:
    let
      exI = h1(<~eskI,~ltkI>)
      kI  = h2(<Y^~ltkI,pkR^exI,
                Y^exI,$I,$R>)
    in
    [ In(Y), !Pk($R,pkR)
    , Init_1(~eskI,$I,$R,~ltkI,hkI) ]
  --[ Accept($I,$R,kI) ]->
    [ ]
```

Responder model




```
/* Responder: receive hkI+send hkR */
rule responder_1:
    let
      exR = h1(<~eskR,~ltkR>)
      hkr = 'g'^exR
      kR  = h2(<pkI^exR,X^~ltkR,
                X^exR,$I,$R>)
    in
    [ In(X)
    , Fr(~eskR)
    , !Ltk($R,~ltkR)
    , !Pk($I,pkI) ]
  --[ Accept($R,$I,kR) ]->
    [ Out(hkr) ]
```

Summary

Next lecture

- Security properties and real-world protocols
  - Modeling a **hierarchy of authentication properties**
- Fixing non-terminating loops with **induction**
- **Lemma annotations**
- Additional features for modeling properties:
  - Predicates
  - **Restrictions**

Reading material

Recommended reading: [Bas+24, Ch. 7-7.1, 11–11.4, 12–12.6],

[Bas+24] D. Basin, C. Cremers, J. Dreier, and R. Sasse. **Modeling and Analyzing Security Protocols with Tamarin: A Comprehensive Guide.** Draft v0.5. Sept. 2024.

Reading material

Further reading: [LLM06], [Cre08]

[Cre08] C. Cremers. **Session-state Reveal is stronger than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange protocol.** Cryptology ePrint Archive, Paper 2008/376. 2008.

[LLM06] B. LaMacchia, K. Lauter, and A. Mityagin. **Stronger Security of Authenticated Key Exchange.** Cryptology ePrint Archive, Paper 2006/073. 2006.