Getting Started

Prof. David Basin
Dr. Dennis Jackson
Dr. Joseph Lallemand
Dr. Jorge Toro

Information Security Lab
November 2020

Installing Tamarin on your machine

- Instructions to install and use Tamarin can be found in the manual: `https://tamarin-prover.github.io/manual/`
- If you use Windows as your main operating system, you will need to either use the VM (instructions below) or install Tamarin via the Linux for Windows Subsystem: `https://docs.microsoft.com/en-us/windows/wsl/install-win10`.
- If you compile Tamarin directly, be sure to check the resulting binary is on your system `$PATH`.

Using Tamarin via the provided Virtual Machine

- We have provided a VM based on Ubuntu 20.04. The VM has the Tamarin Prover and a code editor (Sublime Text) pre-installed.
- The VM can be downloaded from PolyBox. It is 6.1 GB in size. `https://polybox.ethz.ch/index.php/s/8uzrVgb2zQ8zAvc`
- You can run the VM using VirtualBox, which can be downloaded from: `https://www.virtualbox.org`. VirtualBox is available for Windows, Mac and Linux.
- Instructions for importing the VM into VirtualBox can be found here: `https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html`.
- If you are using Windows, previous versions of VirtualBox have been reported as incompatible with the Linux for Windows Subsystem. Be sure to update your version of VirtualBox prior to running the VM.
- The username and password for the VM are both `tamarin`.

Checking Tamarin is working correctly

Once you have installed the tool, create a file `sig.spthy` with the code below. You can also download the code from: `https://polybox.ethz.ch/index.php/s/LA08dJq2Fuhwc8h`

```
theory Sig

begin

builtins: signing

rule LtkGen://PKI
  [ Fr(~ltk) ]
  -->
  [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

//protocol rules
rule Send_Signature:
  [ Fr(~n), !Ltk($A, ltkA) ]
  --[ Send($A, ~n) ]->
  [ Out(<~n, sign{~n}ltkA>) ]
```

```
rule Recv_Signature:
  [ !Pk($A, pkA), In(<n, sig>) ]
  --[ Recv($A, n), Eq( verify(sig, n, pkA), true ) ]->
  [ ]

restriction equal://needed for signature verification
  "All a b #i. Eq(a, b)@i ==> a = b"

lemma executable://sanity check
  exists-trace "Ex A n #i #j. Send(A, n)@i & Recv(A,n)@j"

lemma signature_sent_by_agent://property to be verified
  "All A n #i. Recv(A, n)@i ==> Ex #j. Send(A, n)@j"

end
```

Run `tamarin-prover --prove sig.spthy` and you should get the outcome:

```
  [...]
==============================================================================
summary of summaries:

analyzed: sig.spthy

  executable (exists-trace): verified (6 steps)
  signature_sent_by_agent (all-traces): verified (5 steps)

==============================================================================
```

You can also inspect the file and results in interactive mode by running:
`tamarin-prover interactive .`
(Note the full stop!)

Tips when using Tamarin

- The lab exercises have been carefully selected so that Tamarin terminates quickly. If Tamarin is taking more than a couple of minutes to finish, there is likely a problem with your model.
- Similarly, the lab exercises do not require an excessive amount of RAM and if you see that Tamarin is consuming your system resources, it is likely there is a problem with your model.
- You can use Tamarin to check that a `spthy` file is well formed by passing it to Tamarin without arguments, e.g. `tamarin-prover example.spthy`. Tamarin should report that all wellformedness checks were successful.
- It is vital that the sanity lemmas are proved successfully. These lemmas are designed to ensure that your model of the protocol faithfully describes the intended protocol. If they fail, you may be proving theorems about the empty set of traces, where all security properties are vacuously true.
- When investigating an attack or debugging a model, it can be helpful to use Tamarin's interactive mode to step through the model.

- The Tamarin Manual contains a lot of helpful advice, so consulting it is highly recommended.