CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Formal Analysis of Real-World Security Protocols

Prof. Dr. Cas Cremers, Dr. Aleksi Peltonen
Alexander Dax, Niklas Medinger

https://cms.cispa.saarland/farwsp24/

Exercise Sheet 5

due 16.12.2024 at 14:00

Instructions:

You should submit exercises in groups of **two**. Submissions **must** include the name and matriculation number of **both** students. Please submit only once per group. Solutions must be submitted in PDF format and .SPTHY format to CMS. You should provide solutions in English. If there are additional files, they can be downloaded from the material section in CMS. **Sharing answers with other groups is strictly prohibited!**

Exercise 1: Install TAMARIN (20 points)

In this task you need to install the **TAMARIN-prover version 1.10.**
Depending on the OS you are using, there are different ways to do so.

(a) **Linux**

Depending on your distribution your packet manager may contain a TAMARIN snapshot you can install, e.g., `Arch Linux` packet manager allows to install TAMARIN using

```
pacman -S tamarin-prover
```

For all other distributions where this is not an option, try to install TAMARIN from scratch:

(1) Install dependencies

First you need to install necessary software to run TAMARIN. Namely this would be *Maude*, *Graphviz* and *Stack* (Haskell). Usually you can use your packet manager to install those. In the case of, e.g., Ubuntu this would usually be:

```
apt install maude
apt install graphviz
apt install haskell-stack
apt install haskell-platform - Some versions of Ubuntu might require this
```

Here you need to check that Maude is of version 2.7.1 or higher. If your packet manager does not support this version, you need to install it from scratch (see `http://maude.cs.illinois.edu/w/index.php/The_Maude_System`.) Make sure that the installed binary is then in your `PATH` so you can execute it from anywhere.
Stack is usually also out-of-date when installed so you need to run `stack upgrade`.

(2) Download Tamarin

To download the latest development version of TAMARIN, run:

```
git clone https://github.com/tamarin-prover/tamarin-prover.git
```

(3) Installation

Go to the main directory of the downloaded repository in the command line and execute `make default`. This process may take a while.

(b) **MacOS**

Your first, and the easiest, option is to use *Homebrew* and install Tamarin using:

```
brew install tamarin-prover/tap/tamarin-prover
```

In case you want to install other versions of Tamarin or you do not like to use *Homebrew*, you can install Tamarin as described in the *Linux* section. Using *Homebrew* you also can install the dependencies:

```
brew install tamarin-prover/tap/maude graphviz haskell-stack
```

(c) **Windows**

To install Tamarin on Windows you need to use the Linux subsystem WSL (`https://docs.microsoft.com/en-us/windows/wsl/install`). After installing WSL you can simply follow the instruction in the Linux section to install Tamarin.

For information you can refer to the installation section of the Tamarin-prover documentation `https://tamarin-prover.com/manual/master/book/002_installation.html`.

Additionally, we also provide a *podman* image with a pre-installed version of Tamarin. You can download it at the Materials page of the CMS.

As *proof* that you installed Tamarin, please provide a screenshot of executing `tamarin-prover test`.
(It should be two screenshots as everyone in the group needs to have Tamarin installed).

Good Luck :)

Exercise 2: Understanding a Tamarin model (30 points)

In the following exercise, you will take a look at the `Unknown.spthy` model file from the Materials page. Your main task will be to understand the underlying protocol and the modelled properties, only using the Tamarin file.

(a) (10 points) Translate the protocol into Alice & Bob notation (message sequence chart.) Indicate the computations of each party either next to the chart or below. **Do not add any additional descriptions**.

(b) (10 points) For each value used in the protocol model, briefly explain its purpose in **one sentence**. If a value's purpose seems ambiguous or unclear, indicate that in your answer.

(c) (4 points) Briefly describe and explain the properties that the lemmas `Ex_4_c1` and `Ex_4_c2` prove. Each explanation should be a **single sentence**.

(d) (6 points) Briefly, describe and explain the properties that the lemmas `Ex_4_d1` to `Ex_4_d4` prove. Each explanation should be a **single sentence**.

(e) (5 points (bonus)) In detail, describe and explain the property that the lemma `Ex_4_f` proves. Your explanation should still be **within max. 5 sentences**.

Exercise 3: Modeling Protocols in Tamarin (50 points)

For parts (a) and (b) of the exercise submit the final .spthy file. Make sure to make use of comments to document your design decisions within the file. Note that we will reject .spthy submissions that do not pass the syntax check of Tamarin.

(a) (15 points) In this exercise, you will have to modify a Tamarin model (`dh.spthy`)

(1) In the `dh.spthy`, complete the lemma to prove secrecy of the Diffie-Hellman private keys

(2) Complete the lemma to prove secrecy of the shared final key k.

(3) Complete the lemma stating that both parties actually agree on the same key.

(b) (35 points) Consider the following protocol in Alice & Bob notation.

$$A \to B : aenc(\langle A, N_A \rangle, pk_B)$$
$$B \to A : aenc(N_B, pk_A), MAC(\langle B, A \rangle, K_0)$$
$$A \to B : MAC(\langle A, B \rangle, K_0)$$

Again, A and B are the respective identities of A and B. For the MAC you may use a function symbol with out any equational theory. To verify the MAC use pattern matching.

First, A sends their identity and a nonce N_A encrypted under $B's$ public key to B. Then, B generates the nonce N_B and sends it encrypted under $A's$ public key. Additionally, B computes a key $K_0 = H(N_A, N_B)$ and uses it to create a MAC of the identies of A and B. Now, A decrypts the nonce, uses it to derive K_0, and verifies the MAC. Finally, A sends out another MAC (but with switched parameters) and B verifies this MAC. To finish the protocol, both A and B compute

$$K = MAC(MAC(\langle A, B \rangle, K_0), K_0).$$

(1) Model the above protocol as MSR in the `proto.spthy`.

(2) Prove secrecy for the shared key K.

(3) Prove that A and B agree on the shared key K after executing the protocol.

(4) In the `proto.spthy` we commented out a rule that leaks the secret key of a participant. For this exercise, uncomment this rule. Can you still prove the above statements? If not, can you prove a weaker version of the statements.

(5) Given a security protocol which involves communication of two parties, we often want to guarantee that both parties actually talk to each other. For instance, if the initiating party A finishes a protocol run talking to B, then A wants to be sure that B actually started talking to A at some point as well. This property is called *responder authentication* and it should guarantee that the attacker cannot impersonate B. *Initiator authentication* states the corresponding property with reversed roles.
Can you prove both notions in your model? Can you still prove them assuming one or both keys were leaked? Formulate the appropriate lemmas.