Module 1: Formal Verification of Security Protocols

Lab 1: Tamarin warm-up

David Basin, Xenia Hofmeier and Sofia Giampietro

basin@inf.ethz.ch, xenia.hofmeier@inf.ethz.ch, sofia.giampietro@inf.ethz.ch

September 2023

Welcome to the first lab session of the Information Security Lab! This session is part of the first module on *Formal Verification of Security Protocols*.

Overview

This lab is focused on getting familiar with the Tamarin prover and considers a very simple protocol: Alice and Bob want to exchange two random numbers (m_A and m_B respectively) *secretly*. We will consider different versions of this protocol: different abstraction layers, different security properties and different adversarial models.

For each version, you will write a Tamarin model, specify some expected properties and use Tamarin to verify whether these properties hold or not. We provide the files containing skeletons of the models for each version of the protocol we will study. You can download them on Moodle at:

`https://moodle-app2.let.ethz.ch/mod/folder/view.php?id=954774`.

For this and the following lab assignment, brown boxes will contain concrete tasks to complete:

Task

This box indicates tasks that need to be submitted,

Green ones indicate the expected results (which we will test):

Expected results

This box indicates the results you should obtain.

Finally, we also mark some questions by **"Exercise sheet."** These questions are part of the weekly exercise sheets and are repeated there.

Executability checks.

When modelling protocols in Tamarin, it is good practice to include some sanity checks, e.g. to ensure that the protocol satisfies very basic expected properties.

The most basic property is that it is executable: the parties executing it can complete all their steps. This helps you to detect typos and modelling errors, which may make some of the rewriting rules that model the protocol impossible to execute. In this lab, all files will include an executability check, in the form of a lemma stating that Alice has finished her run of the protocol, sending message m_A and receiving message m_B from Bob, and Bob has finished his run, sending the message m_B and receiving m_A from Alice:

```
lemma executable:
  exists-trace "Ex #i #j A B ma mb.
    FinishedA(A, B, ma, mb)@i & FinishedB(B, A, ma, mb)@j & not (A = B)"
```

More precisely, the last rule of each role must emit an action `FinishedA` (for Alice) or `FinishedB` (for Bob), and Tamarin will attempt to prove that there exists an execution where this action is effectively emitted.

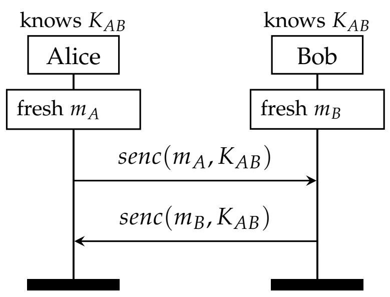
Task

Please make sure to include these actions in all your Tamarin models, so that the executability lemmas can be properly checked.

Task 1.1. Abstracting details away

In some cases (in this task for example!), we are not interested in the particular cryptographic mechanisms Alice and Bob use to send their messages secretly. In this first task, we just assume that Alice and Bob already have a shared fresh symmetric secret key. Hence they can easily send each other encrypted messages. The protocol we will model is the following.

msc Protocol 1



Use Tamarin's rewrite rules to model the setup of the shared symmetric key between Alice and Bob. We do **not** allow either party to get compromised and reveal the key.

Use Tamarin's built-in `symmetric-encryption` theory for the encryption function. It defines the function symbols `senc/2` and `sdec/2`, which are related by the equation `sdec(senc(m,k),k) = m`. This theory can be used in a Tamarin file by adding the line

```
builtins:  symmetric-encryption
```

Task

The skeleton file `Protocol1.spthy` already includes this built-in theory. Complete the other rules in the file, finishing the model of Protocol 1. Follow the instructions in the skeleton file. Particularly, annotate the rules with the action facts as indicated by the comments.

Expected results

Run the executability lemma already provided in the skeleton. Tamarin should verify it.

Exercise sheet.

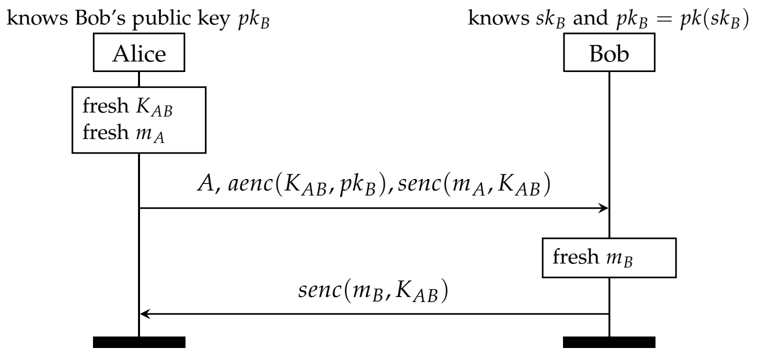
- Use Tamarin in interactive mode to look at the executability trace that Tamarin finds and check that it corresponds to the execution you expect.
- Aside from being executable, do you think this is a good protocol? Would you use it in practice? In the following sections we will use Tamarin to analyze its security, but it's good practice to think about this beforehand. If the Tamarin results don't match your suppositions, it could also be that you might have made modelling errors.

Task 1.2. Concrete key share mechanism

In the previous task, we abstracted away (via a rule) how Alice and Bob share a symmetric key. In real life, this is not an easy problem. In this task, we assume that a public key infrastructure (PKI) is available. We consider the straightforward idea where Alice chooses a symmetric key K_{AB} and sends it to Bob, encrypting it with his public key. This is known as *hybrid encryption*.

In particular the protocol is as follows:

msc Protocol 2


knows Bob's public key pk_B
knows sk_B and $pk_B = pk(sk_B)$
Alice
Bob
fresh K_{AB}
fresh m_A
$A, aenc(K_{AB}, pk_B), senc(m_A, K_{AB})$
fresh m_B
$senc(m_B, K_{AB})$

Tamarin already has a built-in theory `asymmetric-encryption` that models a public key encryption scheme. It defines the function symbols `aenc/2`, `adec/2`, and `pk/1`, which are related by the equation

```
adec(aenc(m, pk(sk)), sk) = m.
```

To be able to use such encryption mechanism, protocol agents need to generate private/public key pairs. Please refer to the lecture slides on infrastructure rules or the section "Public-key infrastructure" of the Tamarin manual (`https://tamarin-prover.github.io/manual/master/book/007_property-specification.html`) for examples of how to define such a rule. For now, we do **not** allow agents to get compromised and reveal their secret keys.

Task

Use Tamarin's rewrite rules to setup a public key infrastructure and model Protocol 2 in the Tamarin file `Protocol2.spthy`. Again, follow the instructions in the skeleton file and add the indicated action facts.

Expected results

Run the executability lemma provided in the file and check that it is successfully proven.

Security properties

Now that we have modelled several protocols, we need to define what properties they should guarantee beyond mere executability.

Note in particular that so far we have only checked (via the lemma `executable`) that the protocol actually works: i.e. Alice and Bob can execute it and share numbers. Such an executability lemma is a sanity check and is crucial to exclude major modeling errors. However that this is *not* a security property.

Task 1.3. Secrecy In this task we will concentrate on secrecy, i.e. the protocol should guarantee that no party other than Alice and Bob can obtain m_A or m_B, see the lectures or the Tamarin manual (`https://tamarin-prover.github.io/manual/master/book/007_property-specification.html#sec:elsewhere`) for a formalization in Tamarin.

Add the following two action facts to the appropriate rules:

- `SecretA(ma)`: Indicates that once Alice finishes her role, exchanging the message `ma` with Bob, she believes `ma` to be secret.
- `SecretB(mb)`: Indicates that once Bob finishes his role, exchanging the message `mb` with Alice, he believes `mb` to be secret.

Task

In both Tamarin models `Protocol1.spthy` and `Protocol2.spthy`, write a lemma `secrecyA` stating that when Alice finishes her role, exchanging a message m_A (apparently) with Bob, then the message is a secret, in that it is only known by Alice and Bob, but no other agents, in particular the intruder. Write an analogous lemma `secrecyB` referring to the message m_B exchanged by Bob at the end of his role. You must only use the action facts `SecretA`, `SecretB`, and `K` to formulate your lemmas. Add the action facts `SecretA and SecretB` in the appropriate rules according to the above description.

Expected results

The lemmas should have the following results:

– Tamarin should **verify** `secrecyA` from `Protocol1.spthy`.

– Tamarin should **verify** `secrecyB` from `Protocol1.spthy`.

– Tamarin should **verify** `secrecyA` from `Protocol2.spthy`.

– Tamarin should **falsify** `secrecyB` from `Protocol2.spthy`.

Exercise sheet. How do the results compare between models? Think about what has changed! One can of course further increase the level of detail in modelling: for example concretely modelling how the asymmetric encryption or the symmetric encryption is performed. Many things could go wrong there too! Give an example of a scenario where, when modelling a given protocol, you are interested in the details of its sub-protocols, and an example in which you are not.

Task 1.4. Agreement. Recall the agreement properties defined in class (see also the Tamarin manual `https://tamarin-prover.github.io/manual/master/book/007_property-specification.html#sec:elsewhere`). Without proving them on Tamarin, can you already predict which properties (aliveness, weak-agreement, non-injective or injective agreement) hold for Alice and for Bob on their exchanged messages? Recall that for now, all parties are honest. We now focus on *non-injective agreement*, in particular:

– `non_injectiveA`: non-injective agreement for Alice with Bob on message m_B. This states that when Alice completes her run of the protocol, apparently with Bob, receiving m_B, then Bob was running the protocol with Alice and sent m_B.

– `non_injectiveB`: non-injective agreement for Bob with Alice on message m_A. This states that when Bob completes his run of the protocol, apparently with Alice, receiving m_A, then Alice was running the protocol with Bob and sent m_A.

Add the following two action facts to the appropriate rules:

• `CommitA($A, $B, mb)`: Indicates that Alice `$A` finishes her role, receiving the message `mb` from Bob `$B`.

• `CommitB($B, $A, ma)`: Indicates that Bob `$B` finishes his role, receiving the message `ma` from Alice `$A`.

- `RunningA($A, $B, ma)`: Indicates that Alice `$A` generated the message `ma` while running the protocol with Bob `$B`.
- `RunningB($B, $A, mb)`: Indicates that Bob `$B` generated the message `mb` while running the protocol with Alice `$A`.

Task

In each of the Tamarin models `Protocol1.spthy` and `Protocol2.spthy`, write the two lemmas, `non_injectiveA` and `non_injectiveB`, described above. You must only use the four action facts `CommitA`, `CommitB`, `RunningA`, and `RunningB`. Add these action facts in the appropriate rules according to the above description.

Expected results

Tamarin should falsify all four lemmas.

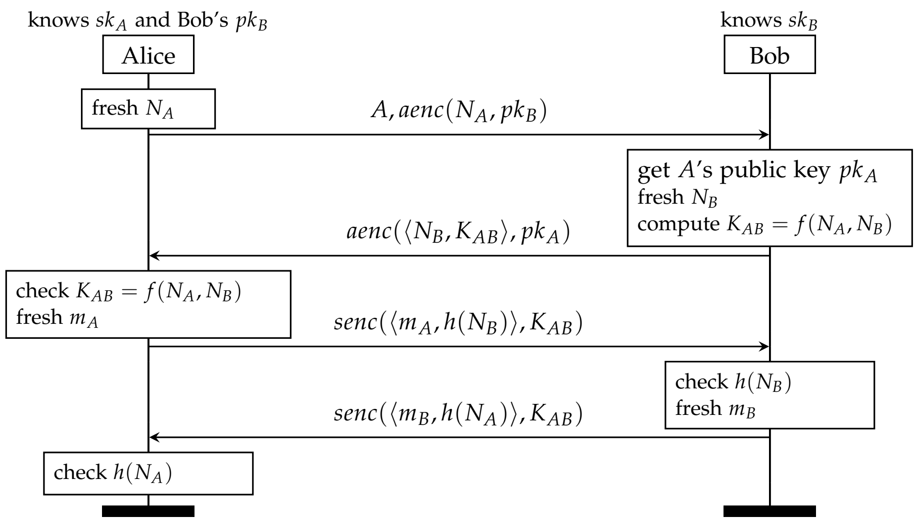
Exercise sheet. Using Tamarin in interactive mode, look at the attack graphs and explain what the attacks represent.

Task 1.5. Concrete key share mechanism revisited

In the previous task, we considered a very basic protocol, which turned out not to guarantee basic security properties such as secrecy or authentication. Why is this?

We will improve the protocol so that Alice and Bob use a key derivation function (KDF) f that produces a key from two random inputs (nonces), which Alice and Bob exchange using asymmetric encryption. Furthermore, for authenticity, Alice and Bob send back hashes of the nonces they receive. Recall that we always assume no agent can be compromised. In particular the protocol is as follows:

msc Protocol 3



knows sk_A and Bob's pk_B
knows sk_B
Alice
Bob
fresh N_A
$A, aenc(N_A, pk_B)$
get A's public key pk_A
fresh N_B
compute $K_{AB} = f(N_A, N_B)$
$aenc(\langle N_B, K_{AB}\rangle, pk_A)$
check $K_{AB} = f(N_A, N_B)$
fresh m_A
$senc(\langle m_A, h(N_B)\rangle, K_{AB})$
check $h(N_B)$
fresh m_B
$senc(\langle m_B, h(N_A)\rangle, K_{AB})$
check $h(N_A)$

To model the KDF use an abstract function symbol f of arity 2. To do so, add the following line in Tamarin.

```
functions:  f/2
```

To ensure that two terms t_1 and t_2 are equal in a given rule, you can add the action fact $Eq(t_1, t_2)$ in the rule actions, and the following restriction in the model

```
restriction Equality:
 "All x y #i. Eq(x,y) @i ==> x = y"
```

Finally we use Tamarin's built-in `hashing` theory, which defines a function `h/1` of arity 1, with no additional properties, representing a hash function.

Task

Model the above protocol in the provided file `Protocol3.spthy`. Add the action facts `SecretA`, `SecretB`, `CommitA`, `CommitB`, `RunningA`, and `RunningB` in the appropriate rules according to their definitions in the previous tasks. Again, follow the instructions in the skeleton file and add the indicated action facts.
In addition to the executability lemma already provided, include the two secrecy lemmas and the two non-injective agreement lemmas defined above. As for the previous tasks, you must only use the action facts `SecretA`, `SecretB`, `K`, `CommitA`, `CommitB`, `RunningA`, and `RunningB` to formulate your lemmas.

Expected results

Tamarin should now verify all five lemmas.

Task 1.6. Adversary capabilities The previous results should be quite surprising. In fact, the above protocol is vulnerable to man-in-the-middle attacks.

This is because, so far, we have only considered the standard Dolev-Yao adversary that is default in Tamarin: the adversary can only interact with the protocol through *In* or *Out* facts. However it cannot "execute" protocol rules. For example, in our previous protocols, we did not allow the adversary to possess a private/public key pair! Depending on our adversarial model, we often want to consider an attacker who can.

Task

In the file `Protocol3.spthy`, add a Tamarin rule `RevealKey` that allows the adversary to compromise an agent. In paticular this rule should allow the adversary to gain knowledge of the agent's secret key, thus allowing it to "pretend" to be that agent. Modify the security properties so that they exclude honest participants being corrupted. You may use the additional action facts `Honest` and `Reveal` in your lemmas, where `Honest(X)` indicates that we assume the agent `X` to be honest and `Reveal(X)` indicates that agent `X` got compromised. Add these action facts accordingly in your rules.

Expected results

Results should be as follows:

- `executable` is verified;
- `secrecyA` is falsified;
- `secrecyB` is verified;
- `non-injectiveA` is falsified;
- `non-injectiveB` is falsified.

Exercise sheet. As you can see, it is important to thoroughly consider which adversarial capabilities you want to take into account and correctly model them. Use interactive mode to inspect the attack graphs for the falsified properties and explain them by drawing a message sequence chart for each counterexample. Can you improve the protocol so that it also guarantees non-injective agreement?

Submission

Your task is to complete the three provided skeleton files following the tasks above. You can download the skeleton files from Moodle:

`https://moodle-app2.let.ethz.ch/mod/folder/view.php?id=954774`.

Do **not** modify the lemmas and any uncommented code that is already provided in the skeleton files (unless we explicitly mention to do so). In particular, please do not change the lemma names, rule names, and theory names. Also, do not add comments within the provided sections.

You must provide the resulting `.spthy` files – **three** files in total:

- `Protocol1_<leginumber>.spthy`
- `Protocol2_<leginumber>.spthy`
- `Protocol3_<leginumber>.spthy`

where `<leginumber>` should be replaced by your matriculation (legi) number *without* dashes, for example: `Protocol1_15821606.spthy`

Use the current release version 1.8.0 of Tamarin for you proofs. Tamarin must be able to process these files and prove or disprove each of the properties automatically, i. e. by running

```
tamarin-prover --prove <model>.spthy
```

Make sure that Tamarin does not raise warnings about wellformedness checks failing or any other errors. We provide a script that checks for the above requirements. Please run this baseline test on your solution before submission, as **we will not accept solutions that do not pass this baseline test,** i.e. you will not receive any points for them. You can find the script with the skeleton files at

`https://moodle-app2.let.ethz.ch/mod/folder/view.php?id=954774.`

You have to provide your path to Tamarin and the path to your submission to the script, i. e. by running:

```
        ./BaseLineScript <path-to-submission> <path-to-tamarin>
```

You can get the path to Tamarin via `which tamarin-prover` .

If all tests pass, the script should output:

```
SUCCESS: You passed all the base line tests. You may submit your theory.
```

Otherwise you will get an error message.

Please submit your three files on Moodle:

`https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=954340`

As usual, you may submit as many times as you want. We will also use the same submission page for the second part of the lab next week.

Evaluation

As stated above, your submissions must pass the baseline test. Submissions that do not pass this test will not receive any points.

The grading will be fully automatic. You will receive points for the following tests:

10 points if the executability lemmas pass for all models.

40 points if your lemmas pass hidden tests: we test them against different models.

50 points if your models pass hidden tests: we test them against different lemmas.

Furthermore, we will check that you adhere to the instructions in the skeleton files (i.e. add the appropriate action facts in the indicated rule). Points will be subtracted if this is not the case.

Finally, we will run anti-plagiarism checks on all submissions.

Good luck!