Crypto Engineering - verifying security protocols
2023/2024

Exercises

All files needed for this session are available at:

`http://www-verimag.imag.fr/~ene/m2p/examples`

The Tamarin web page is

`https://tamarin-prover.github.io/`

Exercise 1
Install the latest version of Tamarin following the instructions from website of the course:

`http://www-verimag.imag.fr/~ene/m2p/read_Me.txt`

or using the official website:

`https://tamarin-prover.github.io/`

Installation instructions for Linux (various distributions) and MacOS can be found in the manual:

`https://tamarin-prover.github.io/manual/book/002_installation.html`

Exercise 2
Make your way through the tutorial file `examples/tutorial.spthy` to familiarize yourself with the syntax.

Load the file in Tamarin using the interactive mode and try to prove the lemmas.

Exercise 3
In this exercise you have to load in Tamarin some 'spthy' files and to prove the specified lemmas. You may modify and try to understand for each example the new feature of Tamarin that is highlighted and how the lemmas capture interesting properties.

1. Download and load in Tamarin the file s1_1.spthy . Without the prefix '!', facts are considered linear : any such linear fact can be consumed only once by a rewriting rule. In the same time, any rule where the left side is empty can by applied undefinetely.
2. Download and load in Tamarin the file s1_2.spthy . Without the prefix '!', facts are considered linear : any such linear fact can be consumed only once by a rewriting rule. Moreover, the GenA rule requires now to consume each time a fresh a.

3. Download and load in Tamarin the file s1_3.spthy . When marked with prefix '!', facts are persistent. The GenA rule will always generate facts of the form !A(x) with a new 'x' value each time. But the '!' mark ensures that now we dispose of an unbounded number of 'copies' of A(x) for a generated fact !A(x).

4. Download and load in Tamarin the file s1_4.spthy . Check that the only way to prove the lemma 'Bs_Ba' is by induction. Add a lemma stating that there exists a trace with events Bs(x) and Bs(y) such that x$\neq$y. Add a lemma stating that for any trace, an event Bs(x) can not occure twice with the same x.

5. Download and load in Tamarin the file s1_5.spthy . Now the adversary is active: he can receive and send messages to the system.

6. Download and load in Tamarin the file s1_6.spthy . Now the adversary is active: he can receive and send messages to the system. Moreover, he can apply function symbols in the signature in order to compute new messages. Check that all lemmas can be proven and try to understand the properties that they encode. Comment the rules GenA and Rev and the lemmas Ba_Rev and Ca_Rev and check that all remaining lemmas are correct. What can you conclude?

7. Download and load in Tamarin the file s1_7.spthy . In this example, the adversary can send messages to the system. Moreover, he can apply function symbols in the signature in order to compute new messages. Notice how the equations allow to an adversary to improve his knowledge and to compute "the secret" 'a'. On the other side, he can not learn the secret 'b'. Remark the different ways to state that a value is secret or not.

8. Download and load in Tamarin the file ind1.spthy . Lemma Origin states that any trace containing a Bigger() event must contain also a Start() event. Firstly, remark the fact that in order to prove this lemma you need to use **induction**. Secondly, enforce the assertion of lemma Origin, to state that any Bigger() event must be **preceded** by a Start() event.

9. Download and load in Tamarin the file ind2_rest.spthy . Lemma Origin states that any trace containing a Bigger() event must contain also a Start() event. Because of the restriction 'once', we limit now the set of possible traces only to traces having at most one Bigger() event. Hence, this time yo do not need to use induction in order to prove the lemma.

10. Download the file eq2.spthy and rerun Tamarin with the option '–diff". The function **penc** corresponds to probabilistic encryption. Since this function satisfies only the equation **pdec(penc(m, pk(k), r), k)=m**, this means that an adversary that obtained an encryption of 'm', not only he is not able to learn the entire message 'm' , but moreover, he is not able to learn anything about 'm'. Lemma 'Secret' states that the plaintext remains secret (weak secrecy). Lemma 'Observational_equivalence',

automatically generated by Tamarin with the option '–diff", corresponds to the strong secrecy property, that is, the adversary cannot distinguish the encryption of the known plaintext 'x' (sent in clear), from the encryption of an unknown plaintext 'y'. The fact **Out(diff(penc(x , pk(ltk), r), penc(y , pk(ltk), r)))** in the rule "Exemple1" allows to check the behavioural equivalence of two systems: in the first one, **x** is sent to the network together with the message **penc(x , pk(ltk), r)**, in the second one, **x** is sent to the network together with the message **penc(y , pk(ltk), r)** .

Exercise 4
Model the protocol from slide 150 (from the file `http://www-verimag.imag.fr/~ene/m2p/m2p_s.pdf`) in Tamarin.

```
1. A -> B: {|na|}k(A,B)       //this is symmetric encryption,
2. A <- B: {|nb|}k(A,B), na   //so you should use "builtins: symmetric-encryption"
```

1. Check that your model is executable using an exists-trace lemma.
2. Formalize the Aliveness property and show that the protocol does not satisfy this property (A thinks that she is talking to B, but B did not necessarily executed a session of the protocol as initiator or responder).

Exercise 5
Model the protocol from slide 144 (from the file `http://www-verimag.imag.fr/~ene/m2p/m2p_s.pdf`) in Tamarin.

```
1. A -> B: A,{na}pk(B) //this is asymmetric encryption,
2. A <- B: na          //so you should use "builtins: asymmetric-encryption"
```

1. Check that your model is executable using an exists-trace lemma.
2. Formalize the Aliveness property as in the previous exercise and show that now, this protocol satisfies this property. Before proving this property, you need a "sources lemma" which asserts that any message "A,{na}pk(B)" received by 'B' either was sent by 'A' or 'na' was known by the adversary (see

   `https://tamarin-prover.github.io/manual/book/007_precomputation.html`

   for more explanations) . An alternative is to use the option "–auto-sources" for launching Tamarin:
   tamarin-prover interactive –auto-sources YourExample.spthy
3. Formalize the Weak Agreement property and show that the protocol does not satisfy it (A thinks that she is talking to B, but this is false, B did not necessarily executed a session of the protocol with A).

Exercise 6
Model the following protocol in Tamarin.

```
1. A -> B: {B}sk(A)  //this is signing,
1. A <- B: {A}sk(B)  //so you should use  "builtins: signing"
```

1. Check that your model is executable using an exists-trace lemma.
2. Formalize the Weak Agreement property from the perspective of the initiator as in the previous exercise and show that now, this protocol satisfies this property.
3. Formalize the Non-Injective agreement property from the perspective of the initiator and show that the protocol does not satisfy it (the initiator A thinks that she is talking to the **responder** B, but this is false, B did not necessarily executed as a **responder**, a session of the protocol with A).

Exercise 7
Model the following protocol from slide 140 (from the file `http://www-verimag.imag.fr/~ene/m2p/m2p_s.pdf`) in Tamarin.

```
1. A -> B: {A,B}sk(A)  //this is signing,
                       //so you should use  "builtins: signing"
```

1. Formalize the Non-Injective agreement property from the perspective of the responder and show that now, this protocol satisfies this property.
2. Formalize the Injective agreement property from the perspective of the responder and show that the protocol does not satisfy it (there may be several sessions for a responder B talking to an initiator A that can be matched by the same session of the initiator A).

Exercise 8

1. Model the following protocol from Ex. 3 from TD1 (from the file `http://www-verimag.imag.fr/~ene/m2p/td1.pdf`) in Tamarin.

   ```
   1. A -> B: {'1', A, NA}pk(B) //this is asymmetric encryption
   2. B -> A: {'2', A, K}pk(A), {'3', NA}K
                 //symmetric encryption in the second message
   3. A -> B: {'4', A, B, K}pk(B)
            //use "builtins: symmetric-encryption, asymmetric-encryption"
   ```

   (a) Check that your model is executable using an exists-trace lemma.
   (b) Formalize the Secrecy property both for NA and K and show that the protocol does not satisfy this property.
   (c) Formalize the Injective agreement from the perspective of both the initiator and the responder and show that the protocol does not satisfy it.

2. Model the following corrected version of the previous protocol in Tamarin.

```
1. A -> B: {'1', A, NA}pk(B)
2. B -> A: {'2', B, K}pk(A), {'3', NA}K
3. A -> B: {'4', A, B, K}pk(B)
```

   (a) Check that your model is executable using an exists-trace lemma.

   (b) Formalize the Secrecy property both for NA and K and show that the protocol satisfies this property. Before proving this property, you need a "sources lemma" which asserts that any message "{'1', 'A', na}pk(B)" received by 'B' either was sent by 'A' or 'na' was known by the adversary, and a similar property for any message "{'2', B, K}pk(A), {'3', NA}K" received by 'A'.

   (c) Formalize the Injective agreement from the perspective of both the initiator and the responder and show that the protocol satisfies it.

Exercise 9
Model the Yahalom protocol from slide 134 (from the file `http://www-verimag.imag.fr/~ene/m2p/m2p_s.pdf`) in Tamarin.

$A \to B : A, N_A$
$B \to S : B, \{'1', A, N_A, N_B\}_{K_{BS}}$
$S \to A : \{'2', B, K_{AB}, N_A, N_B\}_{K_{AS}}, \{'3', A, K_{AB}\}_{K_{BS}}$
$A \to B : \{'4', A, K_{AB}\}_{K_{BS}}, \{'5', N_B\}_{K_{AB}}$

1. Formalize the Secrecy property for N_A, N_B and K_{AB} and check if the protocol satisfies this property.

2. Formalize the NonInjective Agreement property from the perspective of both the initiator and the responder and check if the protocol satisfies it.

3. Formalize the Injective Agreement property from the perspective of both the initiator and the responder and check if the protocol satisfies it.