Exercise Sheet 4

due 25.11.2024 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 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: Quiz (20 points)

Determine whether the following statements are true or false. For each **false** statement, explain in one sentence why it is incorrect.

(1) If the protocol's traces includes behaviors that are not included in the specified properties, then we have an attack.

(2) Traces of system are defined over the action facts and persistent facts of its multiset rewrite rules.

(3) The inbuilt fact $K(x)$ that Tamarin uses is both, a persistent and an action fact.

(4) Tamarin will always terminate for reachability (exist-trace) lemmas if the defined state is, in fact, reachable.

(5) Given enough time and resources, Tamarin will always find an attack or prove that there are none.

(6) Tamarin's algorithm creates constraints from the security properties and from the negations of system rules.

(7) Dependency graphs solve the problem of undecidability.

(8) For each possible trace, there is a corresponding dependency graph.

(9) Persistent facts can have more than one outgoing edge in a dependency graph.

(10) Tamarin uses constraint systems to prevent the attacker from performing unnecessary actions that may cause loops.

Exercise 2: Trace Properties (50 points)

(a) (8 points) **Lemma Correctness**

For the following lemmas, check whether they are syntactically correct, and if not, write one correct version of them.

(1)
```
lemma executable: exists-trace
      "All A B m #i #j. Send(A,m) @#i & Recv(B,m) @#j"
```

(2)
```
lemma distinct_nonces: exists-trace
    "not All n #i #j. Nonce(n)@i & Nonce(n)@j ==> #i=#j"
```

(3)
```
lemma message_authentication:
      "All b m #j. Authentic(b,m) @#j
      ==> Ex #i. Send(b,m) @#i & #i>#j"
```

(4)
```
lemma key_agreement_reachable:
    "not (Ex #i1 #i2 #i3 s k A B minfo.
          Accept(s, k)  @#i1
        & Sid(s, A, B, minfo) @#i2
        & Match(s, minfo) @#i3
        )"
```

(5)
```
lemma secret:
      "All n #i. Secret(n) @i
      ==> (not (Ex #j. K(n)@j))
      |   (Ex B #j. Reveal(B)@j & sign(n,Honest(B))@i)
      "
```

(b) (17 points) **Writing Lemmas**

Consider the following protocol modeling a version of the NAXOS protocol

```
rule NewKeys:
  [ Fr(~x) ]
  --[  ]->
  [ !KeyPair($ID, ~x,'g'^~x) ]

rule Init_Start:
  let
    hskI = h1(~ea,a)
  in
  [!KeyPair($IDA, a, pubA), Fr(~ea)]
  --[  ]->
  [Out(pubA), Out('g'^hskI), InitKeys($IDA, a, ~ea, hskI)]

rule Resp:
  let
    hskR = h1(~eb,b),
    k = KDF(pubA^hskR, hpkI^b, hpkI^hskR)
  in
  [ !KeyPair($IDB, b, pubB), In(pubA), In(hpkI), Fr(~eb) ]
  --[ FinishResp($IDB) ]->
  [ SharedB($IDB, k), Out(puB), Out('g'^hskR) ]

rule Init_Fin:
  let
    k = KDF(hpkR^a, pubB^hskI, hpkR^hskI)
  in
  [ InitKeys($IDA, a, ea, hskI),In(pubB),In(hpkR), Fr(~m) ]
  --[ FinishInit($IDA) ]->
  [ SharedA($IDA, k), Out(senc(~m,k)) ]
```

Using the correct syntax, formulate

(1) a reachability lemma, which states that both *Init* and *Resp* can finish the protocol. Use the existing action facts.

(2) a lemma stating that if *Init* and *Resp* finish a protocol run with the same key k, the attacker does not learn k. To achieve this, create your own action facts and add them to the existing rules. It is sufficient to write down the action fact and link it to the rule name instead of writing down the full modified rule.

(3) two lemmas that express injective agreement for both, *Init* and *Resp*. For this, add a rule that, taking a KeyPair fact, reveals its key to the network. Add all necessary action facts to the correct rules. It is sufficient to write down the action fact and link it to the rule name instead of writing down the full modified rule.

(c) (25 points) **Modeling**

Consider the following protocol between client and server

$$C \to S : aenc(nC, pk_S)$$
$$S \to C : aenc(nC, nS, pk_C)$$
$$C \to S : senc(payload, K_{CS})$$

Here, K_{CS} is the hash of the two nonces the client and the server exchanged. The client is allowed to send as many message to the server as it wants. In the end, the client can decide to drop the connection and end the conversation with the server.

(1) Model the described protocol using multiset rewrite rules. Also model one (or multiple) rules that setup the long-term private and public keys used. Write an extra rule that lets C send a 'DROP' message to the server to terminate the session. This marks the end of the protocol.

(2) Write a reachability/exist-trace lemma for three of your rules (your choice).

(3) Write a lemma to prove that if the client reaches the end of the protocol, the session key is secret.

(4) Write a lemma to verify that the protocol can be executed as intended – the lemma should ensure that each step of the protocol is performed in the correct order, starting with the generation of the keys and ending with the termination of a session.

Exercise 3: Dependency Graphs (30 points)

(a) (10 points) Consider the following protocol expressed as a set of multiset rewriting rules. Produce a dependency graph that shows how we can reach the action fact instance `Finish(a,c,d,c)`. Include all relevant instances of protocol execution rules, message deduction rules, and fresh rules in the graph.

```
rule r1:
    [ Fr(~a), Fr(~b), Fr(~c) ]
  --[ Initialize(~a, ~b, ~c) ]->
    [ St_1(~a, ~b, ~c), Out(~a) ]

rule r2:
    [ St_1(a, b, c), Fr(~d) ]
  --[ Step(a, b, c, ~d) ]->
    [ St_2(a, c, ~d), Out(c) ]

rule r3:
    [ St_2(a, c, d), In(<a, c>) ]
  --[ Finish(a, c, d, c) ]->
    [ !St(a, c, d) ]
```

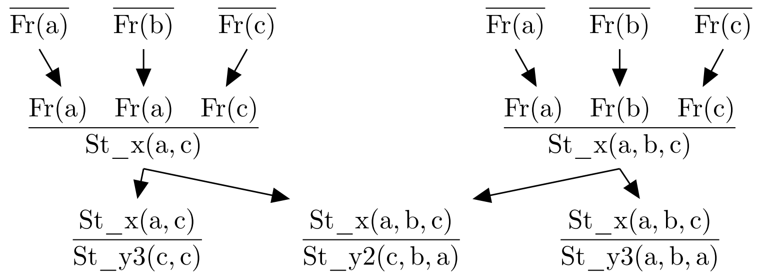
(b) (10 points) Consider the following sequence of multiset rewriting rule instances.

1:	$\dfrac{}{\text{Fr(id)}}$
2:	$\dfrac{}{\text{Fr(m)}}$
3:	$\dfrac{}{\text{Fr(k)}}$
4:	$\dfrac{\text{Fr(id)} \quad \text{Fr(m)} \quad \text{Fr(k)}}{\text{!Secret(id, k)} \quad \text{Msg(id, m)}}[\ \text{StoreSecret(id, k)}\ ]$
5:	$\dfrac{\text{!Secret(id, k)}}{\text{Out(k)}}[\ \text{RevealSecret(id, k)}\ ]$
6:	$\dfrac{\text{Out(k)}}{\text{K(k)}}$
7:	$\dfrac{\text{!Secret(id, k)} \quad \text{Msg(id, m)}}{\text{Out(senc(m, k))} \quad \text{Msg(id, m)}}[\ \text{SendMessage(id, m)}\ ]$
8:	$\dfrac{\text{Out(senc(m, k))}}{\text{K(senc(m, k))}}$
9:	$\dfrac{\text{K(senc(m, k))} \quad \text{K(k)}}{\text{K(m)}}$
10:	$\dfrac{\text{K(senc(m, k))} \quad \text{K(m)}}{\text{In}(\langle \text{senc(m, k), m} \rangle)}$
11:	$\dfrac{\text{!Secret(id, k)} \quad \text{Msg(id, m)} \quad \text{In(senc(m, k), m)}}{}[\ \text{ReceiveMessage(id, m)}\ ]$

(1) Write down the intermediate states before and after each rule instance. See lecture 4, slide 7 for an example.

(2) Write down the corresponding trace. See lecture 3, slide 7 for an example.

(c) (10 points) Consider the following syntactically incorrect dependency graph. Briefly explain each issue you can identify in the graph by referring to the definitions in lecture 4, slide 9.


Fr(a)
Fr(b)
Fr(c)
Fr(a) Fr(a) Fr(c)
St_x(a, c)
Fr(a)
Fr(b)
Fr(c)
Fr(a) Fr(b) Fr(c)
St_x(a, b, c)
St_x(a, c)
St_y3(c, c)
St_x(a, b, c)
St_y2(c, b, a)
St_x(a, b, c)
St_y3(a, b, a)