Module 03: Introduction to Tamarin proofs

Week-6: The IKEv1 protocols

David Basin, Xenia Hofmeier and Sofia Giampietro

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

October 2022

Welcome to the lab session for week 6 of the Information Security Lab. This session is part of the Module-03 on *Introduction to Tamarin Proofs*.

Overview

This lab is focused on modelling and analysing one of the IKEv1 (Internet Key Exchange) key exchange protocols [3]. These protocols are based on the Diffie-Hellman key exchange, and are used to set up secure, authenticated channels in the IPsec protocol suite.

In this lab, we will start by modelling the basic unsigned Diffie-Hellman protocol in Tamarin. We will then progressively strengthen it, to obtain protocols that provide stronger security guarantees – which we will prove using Tamarin. In the end, we will reach one of the IKEv1 protocols, and show that it provides forward secrecy of the shared key.

For this lab session, we provide four Tamarin files containing skeletons of the models for each of the four protocols we will study. You can download them on Moodle at:
`https://moodle-app2.let.ethz.ch/mod/folder/view.php?id=815492`.
One of these files is a complete example and hence need not be handed in.

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, in particular that it is executable. This helps detecting typos and modelling errors, which may make some of the rewriting rules impossible to execute. In this lab, all files will include executability checks, in the form of lemmas stating that there must exist a trace in which the last rule of each role is executed. More precisely, the last rule of each role emits an action `FinishedI` (for the initiator) or `FinishedR` (for the responder), and Tamarin will attempt to prove the existence of an execution where this action is effectively emitted. **Please make sure to include these actions in all your Tamarin models, so that the executability lemmas can be properly checked.**

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 UnsignedDH.spthy`. Tamarin should report that all wellformedness checks were successful.
- It is vital that the sanity lemmas are proved successfully. 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 [2] contains a lot of helpful advice, so consulting it is highly recommended. `https://tamarin-prover.github.io/manual/`

Diffie-Hellman key exchange

We will start by modelling the basic Diffie-Hellman key exchange protocol. In this protocol, all participants must first agree on a finite cyclic group $(G, *)$ of order n, and a generator g. We will assume that these parameters are selected publicly in advance, so that all participants, as well as the attacker, know their values. The protocol between an initiator Alice and a responder Bob is then as follows:

1. Alice first picks a random number x with $1 < x < n$, which she keeps secret. She then computes $X = g^x$, and sends it to Bob.
2. Bob picks a random number y, with $1 < y < n$, which he keeps secret. He computes $Y = g^y$, and sends it to Alice.
3. Alice computes Y^x, and Bob computes X^y. Both obtain the same value $K = g^{xy}$, which constitutes the shared key.

A passive attacker would only learn g^x and g^y from this exchange. The security on the protocol relies on the assumption that computing g^{xy} from these two "partial" keys is computationally hard in the chosen group. This appears to be a reasonable assumption, but only for some specific groups. The Diffie-Hellman key exchange typically uses $(\mathbb{Z}/p\mathbb{Z})^\star$ for prime numbers p, or groups constructed on elliptic curves.

In our models, we will abstract details such as the choice of parameters, and only consider an idealised version of the group operations. Tamarin has a built-in model of Diffie-Hellman group operations, which can be used in a theory file by adding the line

```
builtins:  diffie-hellman
```

This model uses a constant 1 for the group's neutral element, and functions $*$, `^`, and `inv`, respectively for multiplication, exponentiation, and inversion. It also includes the following axioms, defining the behaviour of these function symbols:

$$\begin{array}{rclcrcl} (x\hat{}y)\hat{}z & = & x\hat{}(y*z) & & x\hat{}1 & = & x \\ x*y & = & y*x & & (x*y)*z & = & x*(y*z) \\ x*1 & = & x & & x*\mathtt{inv}(x) & = & 1 \end{array}$$

We will use a public constant `'g'` to model the group generator we need. Note, in particular, that in this idealised model, there is indeed no way to compute $g\hat{}(x*y)$ from $g\hat{}x$ and $g\hat{}y$, or x from $g\hat{}x$.

Example: basic Diffie-Hellman

We will first model the basic Diffie-Hellman protocol presented in the previous section:

$$\begin{array}{lll} A \to B: & A, g^x & \text{x fresh} \\ B \to A: & B, g^y & \text{y fresh} \\ A, B \text{ agree on } & g^{x*y} & \end{array}$$

The provided file `UnsignedDH.spthy` contains a model of the protocol described by the diagram above.

Observe that we have included the action facts `FinishedR` and `FinishedI`, to make the executability lemmas in the file meaningful.

Also note that the initiator role, once it has computed the shared key K, declares it secret – the corresponding rule is labelled with the action `SecretI(A,B,K)`, expressing A's (the initiator) belief that she shares the secret key K with B (the responder). Similarly the action `SecretR(A,B,K)` in the responder's rule indicates the belief that B shares the secret key K with the initiator A.

The file includes the two following lemmas:

```
lemma keySecrecyI:
"All #i A B k. (SecretI(A,B,k) @ i) ==> not (Ex #j. K(k) @ j)"

lemma keySecrecyR:
"All #i A B k. (SecretR(A,B,k) @ i) ==> not (Ex #j. K(k) @ j)"
```

These constitute a rough model of the key secrecy property: they state that, for any messages A, B, k, if at any point i in time the action `SecretI(A,B,k)` (resp. `SecretI(A,B,k)`) is recorded, then the attacker cannot know the value k at any instant j.

Introductory example. Run Tamarin on the file to attempt to prove these lemmas: Tamarin should tell you that it has actually disproved them.

Indeed, the protocol as stated does not satisfy these two properties, for two reasons:

- First, one of the agents involved (Alice and Bob) could be dishonest, i.e. controlled by the attacker. If for instance the role of Alice was played by the attacker, Bob would mark the shared key as secret, but the attacker (through Alice) would of course know its value.
- In addition, even if both agents are honest, since they send their messages in an unauthenticated way, the attacker could modify them, for instance replacing g^x with g^z for some z he knows in the message sent to Bob. Bob would then declare g^{yz} as a secret, which the attacker can compute as $(g^y)^z$.

We will now address these two issues, respectively by incorporating a more detailed model of the honesty and dishonesty of agents, and by extending the protocol so that agents encrypt the messages they send with the intended receivers' public keys.

Task 1: Public-key Diffie-Hellman

For this task, we will be working on the `EncryptedDH.spthy` skeleton file. Consider the following variant of the previous protocol:

$$
\begin{array}{lll}
A \to B: & A, aenc_{pk_B}\{g^x\} & \text{x fresh} \\
B \to A: & B, aenc_{pk_A}\{g^y, h(g^x)\} & \text{y fresh} \\
A \to B: & A, aenc_{pk_B}\{h(g^y)\} & \\
A, B \text{ agree on } & g^{x*y} &
\end{array}
$$

Alice and Bob (in the roles of `Initiator` and `Responder` respectively) only communicate using asymmetric encryption. When Bob receives Alice's first message, he adds to his response the hash of the message he received, to prove that he indeed has the secret key that allows him to decrypt the message. Alice takes similar actions with the message she receives from Bob. **Both Alice and Bob verify that the hash they receive matches the hash of the message they sent.**

To model hash functions, we will use the built-in `hashing` Tamarin primitive, which defines a public function symbol `h/1`, that satisfies no additional property and hence represents a hash function that is perfectly hiding.

For the asymmetric encryption, we will use the built-in `asymmetric-encryption` Tamarin primitive, which defines public function symbols `aenc/2`, `adec/2` and `pk/1`, respectively for encrypting, decrypting and public keys. The term `pk(k)` represents the public key associated to private key `k`, while `aenc(m,k)` represents the encryption of the message `m` with key `k`, and `adec(m,k)` the decryption of message `m` with key `k`. They are related by the equation

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

Finally Tamarin provides special syntax for pairs and tuples, i.e. an n-ary tuple `<t1,...,tn>` is parsed as an n-ary, right-associative application of pairing.

Note that we assume here that a public key infrastructure (PKI) has been previously established, so that all participants have private keys, and the value of all corresponding public keys are publicly known. As you can see in the provided skeleton file, we model this PKI by the rule `GenKey`:

```
[Fr(~kA)] --> [!Key($A,~kA), !Pk($A,pk(~kA)), Out(pk(~kA))]
```

which allows the generation of fresh public and private keys for any agent `A`, records these values as persistent facts `!Key` and `!Pk`, and publishes the public key.

Question 1. Complete the skeleton file with multiset rewriting rules modelling the initiator and responder roles in this protocol. You can of course adapt the models from the previous sections.

As before, make sure that the last rule of each role emits a `FinishedI` or `FinishedR` action (for the executability lemmas), and declares the key as secret using the `SecretI` and `SecretR` facts as actions.

Recall that Alice and Bob must check that the hash they receive matches the hash they compute themselves. To check whether two terms `t1` and `t2` are equal, you can add the action fact `Eq(t1,t2)` in the rule where the check must be made. The restriction, which is already included in the skeleton files,

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

will restrict the search for attacks to executions where these two terms match.

Remark: There are other ways to check whether two terms match (e.g. using pattern matching etc.). If you don't want to use this restriction, you may comment it out to avoid warnings about this restriction not being used. This holds also for the following tasks.

As announced, we will now extend our model with a representation of honest and dishonest agents.

Question 2. Write (in the skeleton file) a rule allowing the attacker to compromise honest agents. From any fact `!Key(A,k)`, this rule will publish to the attacker the secret key `k`, and record as an action `Compromised(A)` the identity of the compromised agent `A`.

As you can see in the skeleton file, the two key secrecy lemmas have been updated with a new condition: the key is now only required to be secret provided neither of the two agents involved are compromised.

Question 3. Run Tamarin on your file, to attempt to prove these two lemmas. The proofs should now go through successfully.

We now have a protocol that guarantees the secrecy of a key established between two honest agents.

However, it has a flaw that our current analysis misses. Despite the use of public keys, it does not allow the agents to reliably authenticate each other: in particular, it does not prevent man-in-the-middle attacks. In the following, we will use Tamarin to find these attacks.

The property that we are interested in is that the participants agree on the key they exchange, *and on each other's identity*.

In general, this is done by proving agreement in both directions, i.e., that if the responder Bob believes that the key K has been established between him and the initiator Alice, then Alice also believes so, and vice versa. We will focus on one direction here (from the responder's point of view), as the second is similar.

Uncomment the Tamarin lemma `agreementR` that is already included in the skeleton file. As you can see, this property is expressed using two actions called `CommitR(B,A,k)` and `RunningI(A,B,k)`. The first one expresses the fact that the responder B is now convinced he is sharing a key k with an initiator A. It should be emitted by the responder role, once it has confirmed that the initiator has correctly received the key – i.e. when confirming that the received hash matches the hash of g^y. The second one expresses the fact that the initiator A has started a session with a responder B, that will in the end produce the key k. It should be emitted by the initiator role, as soon as she is able to compute the shared key, i.e. when receiving the responder's first message.

The agreement property is then specified by requiring that whenever an action `CommitR(B,A,k)` is emitted, there must exist a previous action `RunningI(A,B,k)`. That is, whenever a responder B declares he believes he shares a key k with an initiator A, A has previously recorded that she has started a session to share key k with B.

The agreement property from the point of view of the initiator is analogous: uncomment the Tamarin lemma `agreementI` that is already included in the skeleton file. Observe that we now use action facts `CommitI(A,B,k)` and `RunningR(B,A,k)` which should be emitted by the initiator and responder roles respectively.

The agreement property in this direction is then specified by requiring that whenever an action `CommitI(A,B,k)` is emitted, there must exist a previous action `RunningR(B,A,k)`. That is, whenever an initiator A declares he believes he shares a key k with a responder B, B has previously recorded that he has started a session

to share key k with A.

Question 4. Add the `CommitR` and `RunningI` (respectively `CommitI` and `RunningR`) actions to the relevant rules of your Tamarin model to check that Tamarin disproves both lemmas.

More precisely, to disprove the lemma `agreeementR`, a compromised agent C can perform an attack between two honest agents A and B, so that B believes he is exchanging a key with A, while A believes she is exchanging a key with C. This is because C can decrypt messages he receives from A with his own key, and re-encrypt them with B's public key, pretending to be A.

Note that, as we have proved, this does not let the compromised agent C learn the key g^{xy}. Nevertheless, A and B disagree on the participants of the key exchange: they have not correctly authenticated each other.

To disprove `agreeementI`, a similar man-in-the-middle attack is possible, in which A believes she is exchanging a key with B, while B believes he is exchanging a key with a compromised agent C.

Question 5. Have a look at the attack graphs found and reconstruct the message sequence chart of the concrete attack found by Tamarin. Ask TAs if you have trouble understanding the graphs.

In the next section, we will study a variant of the protocol that fixes this problem, and prove agreement between the two parties.

Task 2. Public-key Diffie-Hellman, second attempt

For this task, we will be working on the `EncryptedDH2.spthy` file. The previous issue is caused by the fact that the attacker can forward to Alice a message that Bob intended to send to someone else, without this being detected. To fix it, we now consider the following variant of the previous protocol, where the identity of the sender of each message is instead added *inside* the encrypted message:

$$
\begin{array}{lll}
A \to B: & aenc_{pk_B}\{A, g^x\} & \text{x fresh} \\
B \to A: & aenc_{pk_A}\{B, g^y, h(g^x)\} & \text{y fresh} \\
A \to B: & aenc_{pk_B}\{A, h(g^y)\} & \\
A, B \text{ agree on } & g^{x*y} &
\end{array}
$$

Again, upon receiving answers, both the initiator and responder roles verify that the received hashes match the hashes of the values they sent.

Question 6. Give a model of this new protocol, including, as in the previous section, a rule letting the attacker compromise agents. You can of course just adapt the model from the previous section.

Do not forget the `SecretI` and `SecretR` actions. Using Tamarin, prove that, as

before, the secrecy of the key is guaranteed between two honest agents. Furthermore, add the `CommitI`, `CommitR`, `RunningI` and `RunningR` actions to the relevant rules of your Tamarin model and this time prove the agreement property for both parties: Tamarin should now succeed.

IKEv1 protocol

Finally, we will complete the `IKEv1.spthy` file. In the previous section, we analysed a Diffie-Hellman key exchange protocol that uses public keys to achieve authentication. We will now study one of the key exchange protocols in the IKEv1 protocols that is based on the same idea, but does things slightly differently. In particular we will be modelling the IKEv1 Phase 1 protocol that uses public key encryption for authentication [3].

We first define the following quantities:

$$\begin{aligned} skID &= prf(\langle h(N_a|N_b), A, B\rangle) \\ HASH_A &= prf(\langle skID, g^x, g^y, A, B\rangle) \\ HASH_B &= prf(\langle skID, g^y, g^x, B, A\rangle), \end{aligned}$$

where prf is a pseudo-random function and h is a hash function.

The IKEv1 Phase 1 protocol is then as follows:

$A \to B:$ $enc_{pk_B}(A), enc_{pk_B}(N_a), g^x$ N_a fresh nonce, x fresh
$B \to A:$ $enc_{pk_A}(B), enc_{pk_A}(N_b), g^y, HASH_B$ N_b fresh nonce, y fresh
$A \to B:$ $HASH_A$
A, B agree on the shared key $K_{AB} = prf(\langle skID, g^{x*y}\rangle)$

As in previous protocols, Alice checks that $HASH_B$ she receives is the same as $prf(\langle skID, g^y, g^x, B, A\rangle)$, which she can compute herself, and similarly for Bob with $HASH_A$.

As you can see in the skeleton file, to model the pseudo-random function `prf`, we can define a public function symbol `prf/1`, that satisfies no additional property. Notice that this function symbol has arity 1. Hence when giving multiple arguments `t1,...,tn` to the function `prf`, you should put them in a tuple: `prf(<t1,...,tn>)`.

Question 7. Write a Tamarin model of the IKE protocol described above. As in previous sections, it must include a rule letting the attacker corrupt agents. Do not forget to include the actions used in the previous models to express secrecy and agreement, as well as the executability checks. As usual, the lemmas are already included in the skeleton file. Using Tamarin, prove that this protocol satisfies all of the these properties.

An interesting feature of the IKE protocol, that it shares with many protocols based on Diffie-Hellman, is that it provides a very strong secrecy guarantee on the key, called *forward secrecy*. This property is a stronger version of the secrecy property we have already proved: it states that a key established between two agents remains secret, *even if in the future the attacker learns the agents' long-term private keys corresponding to their public keys*. In other words, as long as Alice and Bob's long-term private keys k_A, k_B corresponding to their public keys, were not compromised at the time they ran the protocol, an attacker can never learn the exchanged secret g^{xy}, even if he later compromises k_A and k_B. This is an attractive security property: it means that, for instance, if Alice and Bob used the protocol to establish a temporary session key that they then used to encrypt and exchange sensitive data, even if someone manages later on to steal Alice's laptop containing her private key, the key will remain secret and the data secure.

As you can see in the skeleton file provided, we express this property by modifying the condition in the secrecy lemma: we now require that a key k is not known by the attacker as soon as it has been declared a secret between two agents A and B at some time point i and A and B were not compromised before that time.

Question 8. Run Tamarin on your file to check that the protocol provides forward secrecy.

You can come back to your previous models to check whether they also provided this property: unsigned Diffie-Hellman should not, but the two versions of signed Diffie-Hellman should (checking this is entirely optional and it will not contribute to your grade).

Evaluation

Your task is to complete the three provided skeleton files by adding the models for each of the protocols from the previous sections, following questions 1 to 8.

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 and theory names.

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

- `EncryptedDH_<leginumber>.spthy`
- `EncryptedDH2_<leginumber>.spthy`
- `IKEv1_<leginumber>.spthy`

where `<leginumber>` should be replaced by your matriculation (legi) number *without* dashes, for example: `EncryptedDH_15821606.spthy`

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.

Please submit these three files on Moodle:

`https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=782460`

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.

References

1. *The Tamarin Prover.*
   `https://tamarin-prover.github.io/`
2. *The Tamarin Prover Manual.*
   `https://tamarin-prover.github.io/manual/index.html`
3. *IKE Protocols v1. RFC 2409*
   `https://datatracker.ietf.org/doc/html/rfc2409`