Module 05: Introduction to Tamarin proofs

Week-10: The SIGMA protocols

David Basin, Dennis Jackson, Joseph Lallemand, and Jorge Toro

basin@inf.ethz.ch, djackson@inf.ethz.ch
joseph.lallemand@inf.ethz.ch, jorge.toro@inf.ethz.ch

November 2020

Welcome to the lab session for 10th week of the Information Security Lab. This session is part of the Module-05 on *Introduction to Tamarin Proofs*, or Tamarin in short.

Instructions for Online Lab

We begin with the usual pointers and instructions. The lab sessions will again be held online via Zoom. We ask that all students join the Zoom session via the following link:

`https://ethz.zoom.us/j/91450896871`

The first 30-ish minutes of the Zoom session will consist of a general presentation and discussion about the lab objectives and evaluation criteria. Afterwards, you may choose to stay on in the lab session to discuss specific problems you might be having with the lab. You are also welcome to post your questions and to join ongoing discussions on the Moodle forum at the following link:

`https://moodle-app2.let.ethz.ch/mod/forum/view.php?id=475218`

During the Zoom session, we will be hosting "break-out rooms". These are meant to facilitate one-on-one discussions with one of the TAs. During such discussions, you can share your screen and bring to our attention specific implementation issues that you might be facing. Please be aware that when you are sharing a screen with the session, you are responsible for the content that is presented to any other students in the breakout. In other words, *please be mature*.

Overview

This lab is focused on modelling and analysing some key exchange protocols called the SIGMA (for "Sign and MAC")' protocols [3]. These protocols are based on the Diffie-Hellman key exchange, and form the basis of some widely used key exchange protocols such as IKE (Internet Key Exchange). 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 the SIGMA protocols, and show that they provide forward secrecy of the key.

For this lab session, we provide five Tamarin files containing skeletons of the models for each of the five protocols we will study. You can download them on Moodle at:
`https://moodle-app2.let.ethz.ch/mod/resource/view.php?id=512294`

Getting started with Tamarin

For this module, you will need to use the Tamarin prover. You can either install it directly on your machine, following the instructions on the webpage [1], or use the VM with Tamarin preinstalled which is provided. Instructions on how to install Tamarin, how to get the VM, and how to test that your installation is working can be found in the "Getting Started" document on Moodle, which you can find at:
`https://moodle-app2.let.ethz.ch/mod/resource/view.php?id=512297`.

In case you have any doubts or questions on the syntax and usage of Tamarin, the Tamarin manual [2] is available online and is a great source of information.

When modelling protocols in Tamarin, it is good practice to include some sanity checks, e.g. to ensure that the protocol is executable. This helps detecting typos and modelling errors, which often make some of the rewriting rules impossible to execute. In this lab sessions, all files will include light 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 will emit 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 such an action in the last rule of each role, in all your Tamarin models, so that the executability lemmas can be properly checked.**

Introducing the Diffie-Hellman key exchange

We will first present the basic ideas of the Diffie-Hellman protocol. The objective of this protocol is to allow two participants to establish a secret shared key between them, that they can later on use for other purposes (e.g. to encrypt messages, or use the key as input for another protocol).

To do this, 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 file by adding the line `builtins: diffie-hellman`. This model assumes 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:

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

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 $g\hat{}x$.

Unsigned Diffie-Hellman protocol

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

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

Question 1. The provided skeleton file contains a model of the Initiator role, played by Alice in the diagram above. Complete the model of the protocol by adding a model of the receiver role (Bob). You should only need one single rule to do so. Do not forget, as explained above, to include an action `FinishedR`, to make the executability lemmas in the file meaningful.

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 belief that she shares the secret key K with B. Add a similar action `SecretR(A,B,K)` to the model of the responder.

The skeleton file declares 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.

Question 2. 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 modifying the protocol so that the agents sign the messages they send, to authenticate them.

Signed Diffie-Hellman protocol: first attempt

We will now consider the following variant of the previous protocol:

$$
\begin{array}{lll}
A \to B: & A, g^x & \text{x fresh} \\
B \to A: & B, g^y, sign(\langle g^y, g^x \rangle, k_B) & \text{y fresh} \\
A \to B: & A, sign(\langle g^x, g^y \rangle, k_A) & \\
A, B \text{ agree on } & g^{x*y} &
\end{array}
$$

Compared to earlier, we have added signatures to messages, as well as a final confirmation message. When Bob receives Alice's first message, he adds to his response a signature of both g^y, to prove to Alice that he generated this value, and g^x, to prove to Alice that he correctly received her message. Once Alice receives this response, she confirms to Bob the value of both g^x and g^y by sending their signature with her secret key k_A.

We will use the built-in `signing` Tamarin primitive, which defines public function symbols `sign/2`, `verify/3`, `pk/1`, and `true`, respectively for signing, signature verification, public keys, and the constant true. `pk(k)` represents the public verification key associated to signature key `k`, `sign(m,k)` the signature of message `m` with key `k`, and `verify(s,m,pk)` the operation of verifying that `s` is the signature of message `m` using the verification key `pk`. They are related by the equation

```
verify(sign(m,sk),m,pk(sk)) = true.
```

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

```
[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 3. 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 `Finished` action (for the executability lemmas), and declares the key as secret using the `SecretI` and `SecretR` facts as actions.

You can perform the signature verification by pattern-matching the received message in the `In` fact against the expected signature, e.g. writing `In(<m,sign(m,k)>)` to ensure that only messages with a valid signature will be accepted.

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

Question 4. 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 5. Run Tamarin on your file, to attempt to prove these two lemmas. The proof should now go through successfully.

We have now 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 signatures, it does not allow the agents to reliably authenticate each other. More precisely, a malicious agent C could perform a man-in-the-middle attack between two honest agents A and B, so that A believes she is exchanging a key with B, while B believes he is exchanging a key with C. To do so, C starts a session with A, claiming to be B. He then forwards A's partial key g^x to B in his own name. B replies with his own partial key g^y, as well as a signature. C forwards these to A, who is now convinced that B agrees on g^{xy} with her. C can then sign $\langle g^x, g^y \rangle$ with his own key, which convinces B he agrees on key g^{xy} with C rather than A.

Note that, as we have proved, this does not let 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.

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

Signed Diffie-Hellman protocol done right

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 being detected. To fix it, we now consider the following variant of the previous protocol, where the identity of the intended receiver of each message is added inside the signature:

$$
\begin{array}{lll}
A \to B: & A, g^x & \text{x fresh} \\
B \to A: & B, g^y, sign(\langle A, g^y, g^x \rangle, k_B) & \text{y fresh} \\
A \to B: & A, sign(\langle B, g^x, g^y \rangle, k_A) & \\
A, B \text{ agree on } & g^{x*y} &
\end{array}
$$

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 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.

We will additionally show that this protocol guarantees 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 Bob believes key K has been established between him and Alice, then Alice also

believes so, and vice versa. We will focus on the first direction here, as the second is similar.

As you can see in the skeleton file, this property is expressed using two actions called `Commit('Resp','Init',B,A,k)` and `Running('Resp','Init',B,A,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 receiving the last signed message. 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 `Commit('Resp','Init',B,A,k)` is emitted, there must exist a previous action `Running('Resp','Init',B,A,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.

Question 7. Add the `Commit` and `Running` actions to the relevant rules of your Tamarin model. Run Tamarin on your file to prove the agreement property: Tamarin should succeed.

Optionally, you can add the same actions and lemma to the model from the previous section, to check that Tamarin disproves it (this is entirely optional, and will not be graded).

Basic SIGMA protocol

In the previous section, we have analysed the signed Diffie-Hellman key exchange protocol. We will now study a family of protocols, called the SIGMA protocols, that are based on the same idea, but do things slightly differently. In this section, we focus on the basis of the SIGMA protocol, and we will then consider an extension of it in the next section.

The SIGMA protocol relies on the same basic principle of exchanging g^x and g^y to compute g^{xy}. However, it handles authentication of the agents slightly differently, using a "Sign-and-MAC" strategy, which gives the protocol its name:

$$
\begin{array}{lll}
A \to B: & g^x & \text{x fresh} \\
B \to A: & g^y, B, sign(\langle g^x, g^y \rangle, k_B), mac(B, k_m) & \text{y fresh} \\
A \to B: & A, sign(\langle g^y, g^x \rangle, k_A), mac(A, k_m) & \\
A, B \text{ agree on } & g^{x*y} &
\end{array}
$$

where k_m is a key derived from the shared secret g^{xy}, and used only to generate the MACs.

As we can see, this protocol proceeds is a similar way to the incorrect version of the signed Diffie-Hellman protocol we studied earlier, where the signatures do not contain information on the identity of the intended receiver. However, it adds MACs that can be generated only by parties who know the shared secret g^{xy}, to bind the identities of the participants to the session, and prevent any disagreement.

As you can see in the skeleton file, we will model the MAC in Tamarin as a function symbol `mac/2`, with no associated equations: knowing a message and the key, we can compute the MAC, but the MAC does not reveal anything about the message or the key, and cannot be inverted.

We will, similarly, model the key derivation function used for k_m as a function symbol `kdf_m/1`, with no equations. We will then use `kdf_m(g^(x*y))`, where `g^(x*y)` is the shared key computed by each agent, as the key k_m.

Question 8. Write a Tamarin model of the basic SIGMA 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. Using Tamarin, prove that this protocol still satisfies these properties.

An interesting feature of the SIGMA 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 must remain secret, *even if in the future the attacker learns the agents' long-term keys*. In other words, as long as Alice and Bob's long-term signing keys k_A, k_B 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 the keys k_A, k_B. This is a very interesting 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 provided skeleton file, 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 9. 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 (this is entirely optional, and will not contribute to your grade).

Identity-hiding SIGMA protocol

A feature of the SIGMA protocol from the previous section is that it can be adapted to provide identity protection. The basic SIGMA protocol reveals the identities of the participants because of the signatures: anyone observing the exchange can check a signature with different public keys, to deduce the identity of the agent who sent it. In some scenarios however, the participants may prefer to remain anonymous.

To protect the identity of the agents, so that any third party cannot learn who exactly is exchanging a key, variants of the protocols has been proposed. We will focus on one of them here, called SIGMA-I:

$$
\begin{array}{lll}
A \to B: & g^x & \text{x fresh} \\
B \to A: & g^y, \{\langle B, sign(\langle g^x, g^y\rangle, k_B), mac(B, k_m)\rangle\}_{k_e} & \text{y fresh} \\
A \to B: & \{\langle A, sign(\langle g^y, g^x\rangle, k_A), mac(A, k_m)\rangle\}_{k_e} & \\
A, B \text{ agree on } & g^{x*y} &
\end{array}
$$

Compared to the basic SIGMA protocols, all messages that would reveal the identity of the participants are encrypted using a symmetric key k_e, which is derived from the shared secret g^{xy} independently from k_m. We will model this as using a second key derivation function `kdf_e/1`, also with no associated equation. We will also use the symmetric encryption model built into Tamarin, called `symmetric-encryption`.

By encrypting the messages this way, the SIGMA-I protocol ensures that the identity of the responder is hidden from a passive attacker, and that the identity of the initiator remains secret even from an active attacker. Formally proving this would however require notions that are not in the scope of this course (namely, equivalence properties).

Question 10. Write a model of the SIGMA-I protocol. As in the previous sections, it must include a rule allowing the attacker to corrupt agents. Do not forget to include the actions used in the previous models to express secrecy and agreement in the relevant rules of your model. Use Tamarin to prove that the protocol guarantees forward secrecy of the key, as well as agreement of the responder with the initiator, as before. Note that the lemmas this time are not provided in the skeleton file: you will need to add the relevant lemmas yourself.

Evaluation

Your task is to complete the five provided skeleton files by adding the models for each of the protocols from the previous sections, following questions 1 to 10. Please do not modify the lemmas already provided in the skeleton files.

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

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
```

References

1. *The Tamarin Prover.*
   `https://tamarin-prover.github.io/`
2. *The Tamarin Prover Manual.*
   `https://tamarin-prover.github.io/manual/index.html`
3. *SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols.* H. Krawczyk; CRYPTO 2003.
   `https://webee.technion.ac.il/~hugo/sigma-pdf.pdf`