Module 1: Formal Verification of Security Protocols

Lab 1: Password Based Authentication

David Basin, Sofia Giampietro, Felix Linker, and Xenia Hofmeier

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

Autumn Semester 2025

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

Overview

In this lab, you will familiarize yourself with the Tamarin prover by analyzing password-based authentication. In particular we will model different ways that a client C can use a password to log in to a service (e.g. a website) provided by a server S. To keep the focus on learning Tamarin, the protocols in this lab are intentionally simplified and **not** realistic for actual deployment!

We will always assume that the client C already has a registered password pwd_C with the server. We also assume that the server stores the pair $\langle C, pwd_C \rangle$ for each client registered. Note that, in real-world systems, servers never store passwords in plaintext. Instead, they store a salted hash, see `https://en.wikipedia.org/wiki/Salt_(cryptography)`. This defends against attacks that use precomputed tables and prevents attackers who compromise the server from directly reading user passwords. Throughout this first lab assignment we will instead assume that the server is *honest* and cannot be compromised.

For each protocol, you will write a Tamarin model, specify some expected properties, and use Tamarin to verify whether these properties hold or not. We provide some rules for the submission at the end of this document and recommend you read them in advance.

We provide a skeleton file for each protocol we will study. You can download them on Moodle at:

`https://moodle-app2.let.ethz.ch/mod/folder/view.php?id=1262129`.

To guide you, these files already contain the rule templates required for modelling each protocol. In the second lab, which addresses a real-world protocol, such guidance will not be provided.

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

Task

This box indicates tasks that must 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 with **"Exercise sheet"**. These questions are part of the weekly (not-graded) 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 some very basic properties. The most basic property is that it is executable: the parties executing it can complete all their steps, without the help of the adversary. This helps you to detect typos and modelling errors, which may make some of the rewriting rules that model the protocol impossible to apply. In this lab, all files will include an executability check, in the form of a lemma stating that the client `C` and the server `S` have finished their respective runs of the protocol using the shared password `pwd`.

```
lemma executable:
  exists-trace "Ex #i #j C S pwd.
    FinishedC(C, S, pwd)@i & FinishedS(S, C, pwd)@j & not (S = C)"
```

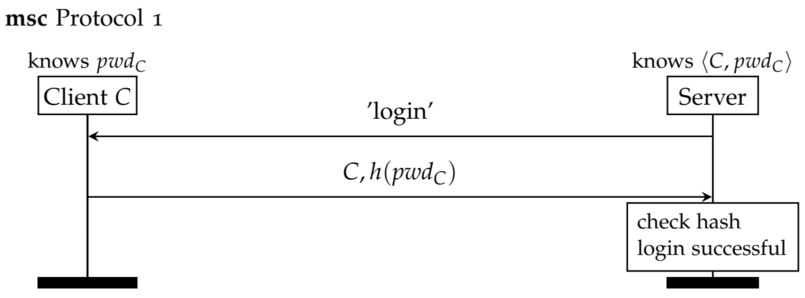
More precisely, the *last* rule of the client's role must emit an action fact `FinishedC` and the *last* rule of the server's role must emit an action fact `FinishedS`, and Tamarin will attempt to prove that there exists an execution where these actions are effectively emitted.

Task

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

It is best practice to check that the trace provided by Tamarin corresponds to the expected protocol execution. Note that Tamarin outputs the first valid trace it finds, which might not be the execution you expect. For example, the same agent might be running the protocol with itself, there might be multiple participants running the same protocol role or the adversary might be actively taking part in the protocol execution. You can add additional, stricter executability lemmas to find the desired trace. Such a lemma could for example restrict the number of protocol participants.

Task 1.1. Tamarin warm-up. To login, the client sends the server the hash of his password. The server checks if this matches the hash of the password he has stored for this client. If successful, the server grants the client access. This protocol is as follows:



You will need to write a rule that creates a shared password for the client and server. We model passwords as fresh terms. Passwords should be reusable across multiple sessions. We do **not** consider the case where a client or server gets compromised and reveals the password.

To model the hash function, use Tamarin's built-in `hashing` theory, which defines the function symbol `h/1` (with no associated equations). This theory can be used in a Tamarin file by adding the line

```
builtins:  hashing
```

at the beginning of the file.

Task

The skeleton file `Protocol1.spthy` already includes the `hashing` built-in theory. Use this file to model Protocol 1. Follow the instructions in the skeleton file and annotate the rules with the action facts as indicated by the comments. Remember to add all the action facts used in the executability lemma.

Make sure that the server checks that the hash it receives matches the hash of the password corresponding to that client! 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"
```

Alternatively, you may check equality via pattern matching.

Expected results

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

Security properties

Now that we have modelled a protocol, we need to define what properties it should guarantee beyond mere executability. Note that so far, we have only checked (via the lemma `executable`) that the protocol actually works: i.e. the client can successfully log in. This is a sanity check and is crucial to exclude major modelling errors. However it is *not* a security property.

In this lab, we will only focus on (1) secrecy of the password, and (2) authentication properties from the *server's* point of view: the server should have the guarantee that only the client who knows the password can login.

Exercise sheet. What authentication guarantees would be desirable from the client's point of view?

Task 1.2. Secrecy

First, we focus on secrecy, i.e., the protocol should guarantee that no party other than the client C and the server can obtain the password pwd_C. 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 action fact to the appropriate rule:

- `Secret(pwd)`: Indicates that, once the client finishes authenticating to the server, it believes his password *pwd* to be secret.

Task

In your model `Protocol1.spthy`, add the action fact `Secret` in the appropriate rules based on the above description. Write a lemma `secrecy` stating that when the client finishes authenticating to the server, then the password is a secret, in that it is not known to the adversary. You must only use the action facts `Secret` and `K` to formulate your lemma.

Expected results

The lemmas should have the following results:

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

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

Task 1.3. Authentication

In this task, we will concentrate on the protocol's main goal, authentication: the protocol should guarantee the server that no party other than the client knowing the password should be able to log in. We consider two variants of the authentication property:

- `non_inj_auth`: if the server grants a client access with a password, then the client has previously authenticated to the server with this password.
- `inj_auth`: if the server grants a client access with a password, the client has previously authenticated to the server with this password and the server has *not* granted the client access with that password at any other time.

Add the following two action facts to the appropriate rules:

- `AuthGiven(C,S,pwd)`: the client C authenticates to the server S with the password pwd.
- `AuthSuccess(S,C,pwd)`: the server S grants the client C access using the password pwd.

Use these action facts to define the above properties. If you need a hint for this, have a look at the message authentication section in the Tamarin manual `https://tamarin-prover.com/manual/master/book/007_property-specification.html`

Task

In your model `Protocol1.spthy`, add the action facts `AuthGiven` and `AuthSuccess` in the appropriate rules and write the lemmas `non_inj_auth` and `inj_auth` according to the above descriptions. You must only use the action facts `AuthGiven`, and `AuthSuccess` to formulate your lemmas.

Expected results

The lemmas should have the following results:

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

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

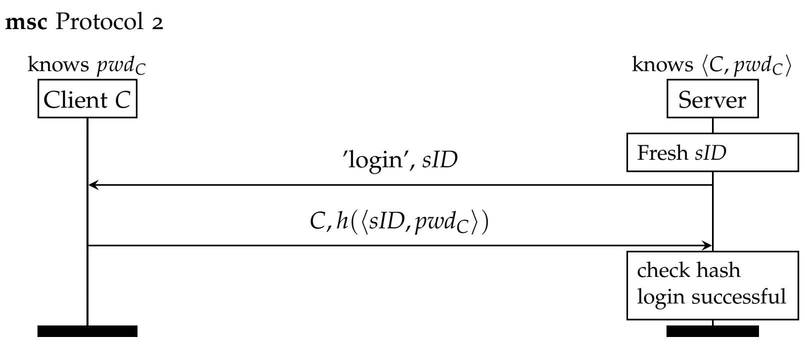
– Tamarin should **falsify** `inj_auth` from `Protocol1.spthy`.

Exercise sheet. Use interactive mode to inspect the attack graph for the property `inj_auth` and explain the counterexample by drawing a message sequence chart.

Task 1.4. Improved protocol.

As we have seen, the previous protocol (as expected) is not very safe. Indeed, anyone can replay a client's message $h(pwd)$ and login as that client.

To resolve the above issue, the server can identify each login session with a random nonce sID. The client then adds the identifier sID within the hash with his password. The server checks if this matches the hash of the generated nonce and the password he has stored for this client. If successful, the server grants the client access. This protocol is as follows:



Task

Use the skeleton file `Protocol2.spthy` to model Protocol 2. Follow the instructions in the skeleton file and annotate the rules with the action facts as indicated by the comments.

We will consider the same properties as in the previous protocol. Modify the `AuthGiven` and `AuthSuccess` action facts to refer to the session identifier sID:

- `AuthGiven(C,S,sID)`: Indicates that the client C authenticated to the server S for the session sID.
- `AuthSuccess(S,C,sID)`: Indicates that the server S grants the client C access for the session sID.

Task

In your Tamarin model `Protocol2.spthy` add, in the appropriate rules, the action facts `AuthGiven` and `AuthSuccess` as described above, and the action fact `Secret` from Task 2. Add the secrecy lemma `secrecy` and the two agreement lemmas, `non_inj_auth` and `inj_auth` defined in the previous task.
You must only use the action facts `Secret`, and `K` for the secrecy lemma and the modified action facts `AuthGiven` and `AuthSuccess` for the agreement lemmas.

Expected results

The lemmas should have the following results:

- Tamarin should **verify** `executable`.
- Tamarin should **verify** `secrecy`.
- Tamarin should **verify** `non_inj_auth`.
- Tamarin should **verify** `inj_auth`.

Exercise Sheet. While Protocol 2 satisfies all the properties we defined, what are some of its drawbacks? Would you use it?

Adversary Capabilities

Our results are so far not so surprising, as the adversary we modelled is very limited. We will consider a more powerful adversary that can perform dictionary attacks on passwords.

A dictionary attack consists in exhaustively enumerating likely possibilities for the password, often obtained from lists of past security breaches (`https://en.wikipedia.org/wiki/Dictionary_attack`). Such attacks often succeed because people tend to choose short passwords that are ordinary words or common variations, such as substituting numbers for similar-looking letters, see `https://en.wikipedia.org/wiki/List_of_the_most_common_passwords`. Note that while dictionary attacks often succeed against human-generated passwords they generally fail against machine generated keys or nondescript, which instead have very high entropy.

For a dictionary attack to succeed, the adversary must be able to check which guess from his list of possible passwords is correct. In this assignment, we assume that trying to login to the server with each possible password guess is infeasible, as the server would notice too many log-in attempts. We assume the attacker can only learn the password if he has a way to check his guesses *without* interacting with the server. For example, if the attacker knows $h(pwd)$, he can check offline for which guess pwd_{guess} the equality $h(pwd_{guess}) = h(pwd)$ holds, implying that $pwd_{guess} = pwd$.

To model this attack scenario in Tamarin, we represent that whenever the adversary gains access to a term containing the password, they can learn the password—representing an offline brute-force capability. Specifically, we introduce a rule that grants the adversary access to *any* registered password, but we restrict this rule to traces where the adversary has previously obtained knowledge of a term that includes the corresponding password.

In particular, every time a term constructed using the password `pwd` and possibly another term `other` is sent over the network we will mark the rule emitting this term by the action fact `PasswordTerm(other, pwd)`. In other words, we mark every emitted term of the form $f(other, pwd)$ for any function symbol f. The attacker can perform a dictionary attack only if it also knows *other*, as in that case it can check for all guesses pwd_{guess} if

$$f(other, pwd_{guess}) \overset{?}{=} f(other, pwd),$$

which would imply $pwd_{guess} = pwd$. We do not assume that the adversary can check his guess without knowing the other terms involved, as these are assumed to be of high entropy.

Finally, we introduce a rule `LeakPassword` that allows the adversary to learn any password `pwd` shared between the client and the server and we mark this rule with the action fact `DictionaryAttack(pwd)`. To model that the adversary can execute this rule only in the case described above, we add the following restriction:

```
restriction DictOffline:
"All pwd #i.  DictionaryAttack(pwd)@i ==>
             Ex other #j #l . PasswordTerm(other, pwd)@j & (#j < #i)
                              & K(other)@l & (#l < #i)"
```

Recall that restrictions ensure that Tamarin only considers traces where the property specified in the restriction holds.

Task 1.5. Dictionary attacks

The skeleton file `Protocol2Dict.spthy` already includes the above restriction. Copy your previous model and lemmas (from `Protocol2.spthy`) in this new file.

Task

In your Tamarin model `Protocol2Dict.spthy` add a rule `LeakPassword` allowing the adversary to learn a client's password `pwd`. This rule should be annotated by an action fact `DictionaryAttack(pwd)`.
Furthermore, add the action facts `PasswordTerm(other, pwd)` in the appropriate rules, where `other` needs to be instantiated appropriately.

Expected results

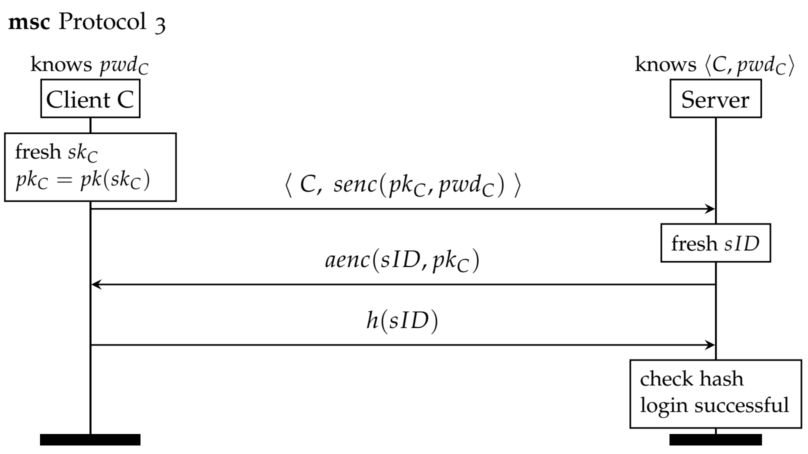
Have the results for your properties changed? The lemmas should now have the following results:

- Tamarin should **verify** `executable`.
- Tamarin should **falsify** `secrecy`.
- Tamarin should **falsify** `non_inj_auth`.
- Tamarin should **falsify** `inj_auth`.

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 graph for the property `non_inj_auth` and explain the counterexample by drawing a message sequence chart.

Task 1.6. Resistance to dictionary attacks

We will now try to improve the above protocol to be resistant to dictionary attacks. We assume that the client and server have access to a symmetric and an asymmetric encryption scheme.



Tamarin already supports both symmetric and asymmetric encryption with the built-in theories `symmetric-encryption` and `asymmetric-encryption`. The first one provides the function symbols `senc/2` and `sdec/2`, obeying the equation

`sdec(senc(m,k),k) = m.`

The asymmetric encryption theory provides the function symbols `pk/1`, `aenc/2` and `adec/2`, obeying the equation

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

The section *Built-in message theories and other built-in features* of the Tamarin manual `https://tamarin-prover.com/manual/master/book/004_cryptographic-messages.html` contains more details about these theories.

Task

Model Protocol 3 in the Tamarin file `Protocol3.spthy`. The skeleton file already loads the necessary built-in theories. Follow the instructions in the skeleton file and add the indicated action facts.

We will now consider the same security properties from the previous task.

Task

Add the rule `LeakPassword` to your model allowing the adversary to learn the client's password, and add the `PasswordTerm` action facts in the appropriate rules. The `DictOffline` restriction is already included in the skeleton file.
Add the security lemmas (`secrecy`, `non_inj_auth`, and `inj_auth`) from the previous task and the action facts `Secret(pwd)`, `AuthGiven(C, S, sID)`, and `AuthSuccess(S, C, sID)` to the rules as described in the above tasks.

Expected results

The lemmas should have the following results:

- Tamarin should **verify** `executable`.
- Tamarin should **verify** `secrecy`.
- Tamarin should **verify** `non_inj_auth`.
- Tamarin should **verify** `inj_auth`.

Submission

Your task is to complete the four provided skeleton files following the tasks above and submit them on Moodle at:

`https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=1262131`.

The submission deadline is **Monday 13.10.2025 15:00**.

You may submit as many times as you want. We will grade your **latest** submission and do not have access to earlier submissions. Thus, check carefully that you submitted the correct files and the correct versions. We will also use the same submission page for the second part of the lab next week.

The skeleton files are available at

`https://moodle-app2.let.ethz.ch/mod/folder/view.php?id=1262129`.

Rules for Submission

- Develop the solution yourself, **without the help of others**.
- Name your solution `.spthy` files – **four** files in total – as follows:
  - `Protocol1.spthy`
  - `Protocol2.spthy`
  - `Protocol2Dict.spthy`
  - `Protocol3.spthy`
- Do not submit folders or zip files.
- Use Tamarin version 1.10.0.
- Do **not** modify the executability lemma, rules, or any uncommented code that is already provided in the skeleton files (unless we explicitly mention to do so).
- Do not change the theory name, lemma or action fact names or the order of arguments for action facts.

- Make sure that you precisely follow the instructions in this lab description and the instructions in the skeleton files. Don't forget to add the action facts to each rule as indicated by the skeleton file and the action facts required for your lemmas. It is crucial for the automatic grading that you do so.
- tamarin–prover ––prove <model>.spthy must not raise any warnings or errors and must terminate.
- Test your submission using the script as explained below.

Testing your Submission

We provide a script that checks for the above requirements. Please run this baseline test on each of your solutions 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 at

`https://moodle-app2.let.ethz.ch/mod/resource/view.php?id=1280076`.

The script requires python3. 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> $(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. The script might ask for your path to your Maude installation as a third argument.

Evaluation

The grading will be fully automatic. For this first lab, you can receive 40 points. We test your lemmas against different models and we test your models against different lemmas. Each lemma receives points if it passes all our tests and each model receives points if it passes all our tests. We will not publish the test lemmas and models until after the submission deadline.

The point distribution is as follows:

	Model	Lemmas
`Protocol1.spthy`	7	3
`Protocol2.spthy`	7	3
`Protocol2Dict.spthy`	2	3
`Protocol3.spthy`	12	3
`Second Lab`	36	24