AVISS — Automated Verification of Infinite State Systems

(IST-2000-26410)

Deliverable D2.1

Specification of the High-Level Protocol Specification Language HLPSL and prototype parser for the language

1 Introduction

The second work-package WP2 of our project AVISS IST-2000-26410 is devoted to the definition of the High-Level Protocol Specification Language HLPSL, a language close to the languages used in text-books and by engineers, of the Intermediate Format IF, and of the translation from HLPSL to IF. In particular, this deliverable D2.1 describes our implementation of task T2.1 by illustrating the specification of the HLPSL and of the prototype parser that we have written for this language.

2 The HLPSL Syntax

We begin with a simple example of a HLPSL specification.

Protocol `Otway_Rees`;
Identifiers
A, B, S :*User*;
Kas, Kbs, Kab:*Symmetric_Key*;
M, Na, Nb, X :*Number*;
Knowledge
A :B, S, Kas;
B :S, Kbs;
S :A, B, Kas, Kbs;
Messages
1. $A \rightarrow B$: $M, A, B, \{Na, M, A, B\}Kas$
2. $B \rightarrow S$: $M, A, B, \{Na, M, A, B\}Kas, \{Nb, M, A, B\}Kbs$
3. $S \rightarrow B$: $M, \{Na, Kab\}Kas, \{Nb, Kab\}Kbs$
4. $B \rightarrow A$: $M, \{Na, Kab\}Kas$
5. $A \rightarrow B$: $\{X\}Kab$

Session_instances
$[A : a; B : b; S : se; Kas : kas; Kbs : kbs]$;
Intruder *Divert*, *Impersonate*;
Intruder_knowledge a;
Goal *Secrecy_Of* X;

Let us comment each part of this specification. A protocol specification can be split into several parts, namely:

- its name;
- the identifiers used in the protocol specification;
- the knowledge that each role in the protocol is supposed to have at the beginning of a protocol session;
- the sequence of messages of the protocol;
- the description of the knowledge of the principals, both in parallel sessions and in the studied session;
- the intruder's capabilities;
- the intruder's knowledge;
- the goal of the intruder, that is the flaw we are looking for.

We shall now illustrate how to specify each of these sections in HLPSL syntax; note that the sections are all mandatory (except for the parallel sessions one) and have to be given in this order.

The *name* of the protocol is given by:

> **Protocol** `name_of_the_protocol`;

The Identifiers used for the description of the protocol are given with:

> **Identifiers**
> *list of identifiers*:`type`;
> ...

A list of identifiers is a list of names separated by commas. Names are simply alphanumeric strings, but beginning neither with x nor X nor with a numeral. The types are among the following: *User*, *Public_Key*, *Symmetric_Key*, *Number*, *Function*. *User* represents roles in the protocol, *Public_Key* is for the names of both public and private key (one only has to define the public key), *Number* is for any kind of data, and *Function* is used for hash functions. We always suppose that hash functions are one-way injective functions; they are represented as free constructors.

We then define the knowledge each role of the protocol has before each session of the protocol as follows:

> **Knowledge**
> *role1*:`list of terms`;
> *role2*:...

The syntax is straightforward. Roles always know at least their name; it is therefore unnecessary to specify it. The terms have to be built with the identifiers previously given, and with the following constructors:

- _, _ is the concatenation of two terms. Functions are all unary functions, with their arguments being concatenated with this *comma* constructor.
- {_}_ expresses the encryption of the first term by the second one. In the current version of HLPSL, the second term may only be a public key, a private key, or a symmetric key. The type of the encryption (public, private, or symmetric) is given by the type of the key used for encryption.
- $(\_)XOR(\_)$ expresses that the two given terms are XOR-ed. We do not consider block cipher properties afterwards. They have to be given in the HLPSL description of the protocol;
- _(_) is valid whenever the first argument is of type *Function*, and it expresses the application of this function with arguments being given by the second term.

The *messages* sent and received during a session of the protocol are specified as follows:

> **Messages**
> 1. $Sender \rightarrow Receiver : Term$
> ...

where messages are numbered from 1 to n, $Sender$ and $Receiver$ are among the identities declared in the Identifiers part of the specification, and $Term$ is a term constructed as described above. It is mandatory that the receiver of message n is the sender of message $n+1$.

Let us now consider the definition of principals in parallel, which is specified by:

Parallel $Role1 : [ID1 : Name1; \ldots], \ldots;$
Secret $List\ of\ Names;$

where $Role1$ is a role in the protocol's specification. Inside the hook, we instantiate the knowledge defined in the knowledge section of the specification, plus the principal acting this role. The list of names is a help that gives names that should remain unknown to the intruder.

Given this, we can then construct a scenario under which we study a particular protocol, and which consists of one or more instances of the protocol. In order to do this, one specifies **Session_instances**, which define who uses the protocol and how. Between square brackets, we define an instance of the protocol with the principals involved and the knowledge they use.

Session_instances
$[A : a1; \ldots]$
$[A : a2; \ldots]$
$\ldots;$

The intruder's possible actions are given in the **Intruder** field of the specification:

Intruder abilities_list;

where the abilities of the intruder are among the following:

- *Impersonate*, meaning the intruder may send a message faking his identity.
- *Divert*, meaning the intruder can intercept the messages sent by honest principals.
- *Eaves_dropping*, meaning the intruder can know the content of messages not sent to him by honest principals.

The most powerful intruder, corresponding to the Dolev-Yao intruder model, is defined with both *Impersonate* and *Divert* since in this case he can always simulate eaves dropping by simply replaying the messages he has intercepted.

The intruder's Knowledge is defined as follows:

Intruder_knowledge $list\_of\_instantiated\_identifiers;$

There are two important points in this definition. By "instantiated identifiers", we mean names given to identifiers in the session instance part of the HLPSL specification. Moreover, one can only give "instantiated identifiers" in this list, and not terms built with these identifiers, though it is planned to do so in the near future.

We have identified two major flaws (and corresponding goals) in protocols, which we express in terms of goals that the protocols should satisfy.

1. We specify a correspondence flaw as a symmetric property between two principals as follows:

   Goal *Correspondence_Between A B*;

2. The flaw resulting from the failure of the Long Term Secrecy of an identifier defined in the Identifiers part of the specification, which is specified as:

   Goal *Secrecy_Of M*;

3 Glossary

- **correspondence flaw**: this is the most frequently occurring flaw, and it happens when a principal *Alice* believes, through her interactions with the environment, that she is talking to another principal *Bob*, whereas *Bob* is not aware that someone tries to communicate with him. This is symmetric among principals.
- **fresh**: a data is said to be fresh whenever it is created by an honest principal during a session of the protocol.
- **goal**: the goal is a property that a protocol should satisfy (and for which we test). It is defined with respect to the session instances.
- **intruder**: the intruder represents a principal who tries to find flaws in, and thus attack, a protocol. He may also participate in the protocol (dishonest principal), and his possibilities are described in the intruder's knowledge and intruder parts of the HLPSL description of the protocol.
- **name**: an instantiation of an identifier.
- **nonces**: this term refers to data given in the identifiers part of the HLPSL description of the protocol, but not known at the beginning of a session. These data are created the first time they are used.
- **parallel**: this refers to weaker principals, acting according to the protocol's description, but not in the studied session, and whose behavior is an approximation of the behavior of standard, real principals memory and who always use the same nonces, that are here in order to help the intruder. The use of these rules is not correct (we do not insure that all flaws found are real flaws), but helps accelerate the search for flaws, and is complete (if there is any flaw, we'll find it).
- **principal**: this term refers to any participant of the protocol. This may include, usually, Alice and Bob, a Server, a Vendor, etc. When a principal follows its role, we speak of honest principals. Dishonest principals are all merged under the identity of the intruder, which is sound as long as dishonest principals collaborate (as is usual in standard intruder models).
- **role**: this term refers to the set of actions a principal has to perform during a session of the protocol. Roles have to be instantiated by either principals or the intruder.
- **runnable**: a protocol is said to be runnable if each message in the protocol can be composed by the honest role who is supposed to send it. It only depends on the knowledge of the principal before he sends a message.

- **secrecy (long term)**: it refers to data that should remain unknown to the intruder, even in the long run.
- **role**: a role defines what a principal carrying out the protocol should know and what message he should receive and send.
- **session**: it refers to a run of the protocol between the principals given in the Session instance part of the HLPSL description of the protocol.
- **session instance**: a session instance specifies the principals who will participate in a session, together with their knowledge.
- **type**: we assign a type to the identifiers to specify how the identifier should be used. Among the common types, one can find `user` for principals names, `number` for each message part with no special properties (text, timestamps,...), `public_key` and `symmetric_key` for encryption, `function`, and so on. `Number`, `public_key` and `symmetric_key` may be used as nonces. In case of public key encryption, both the public key and private one are created by the principal using them.

4 The HLPSL-parser

It is planned, as part of the workpackage 2 to write a compiler (HLPSL2IF) from HLPSL to the Intermediate Format. This compiler should take as input a file containing the HLPSL description of a protocol, and prints on standard output rewrite rules corresponding to the different roles acting in the protocol, as well as honest principals' descriptions.

So far we have implemented a prototype parser that will form the front-end of the compiler. As part of the parser, we also perform a first analysis of the given protocol, namely which data are fresh in each protocol step and if the protocol is runnable. The back end of the compiler, i.e. the generation of the Intermediate Format code, will be part of the next deliverable (D2.2).

In the future, the compiler should have the following usage:

```
HLPSL [options] file
```

where `file` is the file containing the HLPSL specification of the protocol to be analyzed.

The possible options are among the following:

- `--intruder` prints the intruder rules.
- `--rules` prints, for each message in the protocol description, a rule describing the actions a role carries out when it receives the message and sends a reply. For a more detailed description, we refer to the Roles description part of the HLPSL-parser's output.
- `--init` prints the initial state of the protocol description. This includes descriptions of the principals and the intruder's knowledge specification as given in the Intruder_knowledge part of the HLPSL specification of the protocol.
- `--goal` prints a rewrite rule corresponding to the negation of the property given as a goal in the HLPSL description of the protocol. We refer to the goal description for a more accurate description of the possible flaws, and to the role description and the intruder's knowledge description for the semantics of the output.
- `--ident` prints the identifiers' list and type as given in the Identifiers part of the HLPSL description of the protocol. (This is mainly for debugging purposes.)
- `--simplif` prints simplification rules for the intruder's knowledge.

- `--para` prints parallel rules. Parallel principals are an approximation of normal principals. The rewrite rules given in output by this option have to be completed by another tool to get all the possible interactions between the principals in parallel.
- `--knowledge` prints the knowledge of users at the different protocol steps. This is intended for debugging purposes.
- `--all` prints everything.
- `-l path` gives a path for the input file.
- `-help` displays the list of options. The HLPSL-parser defaults to this if the options given are not recognized.
- `--typed` changes the output of the HLPSL-parser.