CAS+

Ronan Saillard and Thomas Genet

March 21, 2011

We present the CAS+ language designed for the easy specification and verification of security protocols. The objective of CAS+ is to have a language as simple as CASRUL [JRV00] and leading to specifications as precise as HLSPL [CCC+04]. The syntax is essentially the one of CASRUL complemented with a verification goal declaration close to what is used in HLPSL. Since syntax of CAS+ is very similar to CASRUL the following manual essentially consists of adapted excerpts of [JRV00].

1 Structure of a CAS+ protocol specification

The specification of a protocol P comes in six parts. These parts respectively declare identifiers, message sequences, agent knowledge, session instances, intruder knowledge and verification goals. See Figure 1 for examples. Note that more examples are available once SPAN is installed on you system in the directory:

```
<SPAN installation directory>/testsuite/CAS_Protocols
```

2 Identifiers declarations

The identifiers used in the description of a protocol P have to be declared to belong to one of the following types: `user` (principal name), `public_key`, `symmetric_key`, `function`, `number`. The type `number` is an abstraction for any kind of data (numeric, text or record ...) not belonging to one of the other types (`user`, `key`, etc.). An identifier F of type `function` is a one-way (hash) function. This means that one cannot retrieve X from the digest $F(X)$. The unary postfix function symbol $'$ is used to represent the private key associated to some public key. For instance, in Figure 1, Kd' is the private key of D.

3 Messages

The core of the protocol description is a list of lines specifying the rules for sending messages,

$$(i.\ S_i \rightarrow_i R_i : M_i)_{1 \leq i \leq n}$$

For each $i \leq n$, the components i is the step number, S_i and R_i are users (respectively sender and receiver of the message) and M_i (message) is a ground term over a signature $\mathbb{F}$ defined as follows. The declared identifiers as well as integers are nullary function symbols of $\mathbb{F}$. The symbols of $\mathbb{F}$ with arity greater than 0 are $\_'$, $\_[\_]$ (for tables access), $\_(\_)$ (for one-way functions access), $\_,\_$ (pairing), $\{\_\}\_$ (encryption), $\_\hat{}\_$ (exponential) and $\_\#\_$ (xor). We assume that multiple arguments in $\_,...,\_$ are right associated. We use the same notation for public key and symmetric key encryption, i.e $\{\_\}$ is an overloaded operator. Which function is really employed shall be determined unambiguously by the type of the key.

The arrow $\rightarrow_i$ represents the type of channel used to send the message. It can be one of the following ascii symbols: `->` for usual Dolev-Yao channel, `=>` for read and write protected channel and `~>` for write protected channel.

Example 1 *Throughout the paper, we illustrate our method on two toy examples of protocols inspired from the one of [JRV00] and presented in Figure 1. These protocols describe messages exchanges in a home cable tv set made of a decoder D and a smartcard C. C is in charge of recording and checking subscription rights to channels of the user. In the first rule of the symmetric key version, the decoder D transmits his name together with an instruction Ins to the smartcard C. The instruction Ins, summarised in a number, may be of the form "(un)subscribe to channel n" or also "check subscription right for channel n". It is encrypted using a symmetric key K known by C and D. The smartcard C executes the instruction Ins and if everything is fine (e.g. the subscription rights are paid for channel n), he acknowledges to D, with a message containing C, D and the instruction Ins encrypted with K. In the public key version, the privates keys of D and C respectively are used for signing messages instead of K.*

4 Knowledge

At the beginning of a protocol execution, each principal needs some initial knowledge to compose his messages. The field following knowledge associates to each user a list of identifiers describing all the data (names, keys, function, etc.) he knows before the protocol starts. We assume that the own name of every user is always implicitly included in his initial knowledge.

Example 2 *In Figure 1, D needs the name of the smartcard C to start communication. In the symmetric key version, both C and D know the shared key K. In the public key version, they both know the public keys Kc and Kd. Note that the number Ins is not declared in D's knowledge. This value may indeed vary from one protocol execution to one another, because it is created by D at the beginning of a protocol execution. The identifier Ins is therefore called a*

fresh number, or nonce (for oNly once), as opposite to persistent identifiers like C, D or K.

5 Session instances

This field proposes some possible values to be assigned to the persistent identifiers (e.g. *tv* for D in Figure 1) and thus describes the different systems running the protocol. It is possible to instanciate a session with the intruder by using the persistant identifier i. The different sessions can take place concurrently or sequentially an arbitrary number of times.

Example 3 *In Figure 1, the public key version of the protocol contains two concurrent sessions, one between an agent* scard *and an agent* tv *and another one between the intruder* i *and agent* tv*.*

6 Intruder knowledge

The intruder knowledge is a set of values introduced in session instance, but not a set of identifiers.

7 Goals

This is the kind of properties we want to prove. There are three families of goals: secrecy, authentication and weak authentication all coming from the $HLPSL$ language and AVISPA [ABB+05] verification tools. The secrecy is related to one identifier and one set of users which must be given in the declaration. The identifier represents the value to keep secret and the set of users represents the agents allowed to learn the secret. The (weak) authentication is related to two users and an identifier. The first user (weak) authenticates the second on the last identifier.
Warning: authentication HLPSL flags are generated but not always conveniently placed. The HLPSL file has to be checked and, possibly, witness/request flags moved!

8 Acknowledgements

Many thanks to Laurent Vigneron for giving us the source code of the CASRUL parser.

References

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko,

```
protocol TV; % symmetric key
identifiers
C,D : user;
Ins : number;
K : symmetric_key;

messages
1. D -> C : D,{Ins}K
2. C -> D : C,D,{Ins}K

knowledge
D : C,K;
C : K;

session_instances
 [D:tv,C:scard,K:onekey];

intruder_knowledge
 scard;

goal
 C authenticates D on Ins;
```

```
protocol TV; % public key
identifiers
C,D : user;
Ins : number;
Kc,Kd : public_key;

messages
1. D -> C : D,{Ins}Kd'
2. C -> D : C,{Ins}Kc'

knowledge
D : C,Kc,Kd;
C : D,Kc,Kd;

session_instances
 [C:scard,D:tv,Kc:kc,Kd:kd]
 [C:i,D:tv,Kc:ki,Kd:kd];

intruder_knowledge
 tv,scard,ki,ki',kc,kd;

goal
 D authenticates C on Ins;
 secrecy_of Ins [C,D];
```

Figure 1: Cable TV toy example

J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In *CAV'2005*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[CCC+04] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS)*, Linz, Austria, 2004.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proceedings of 7th Conference on Logic for Programming and Automated Reasoning*, volume 1955 of *LNAI*. Springer-Verlag, 2000.