Technical report:
SeVe implementation

Luu Anh Tuan[1], Jun Sun[2], Yang Liu[1], and Jin Song Dong[1]

[1] School of Computing, National University of Singapore
{tuanluu,liuyang,dongjs}@comp.nus.edu.sg
[2] Singapore University of Technology and Design
sunjun@sutd.edu.sg

1 Secerity language comparasion

There are many languages to describe the security protocols so far; however, most of them are complicated to use or only suitable for one specific studies but not for all cases. In this section, we will investigate three well-known security languages which are commonly used : Casper, ProVerif and HLPSL. For easy comparison, I give an example description of Needham Schroeder public key protocol in three languages as well as our security language SeVe.

Casper: Casper was proposed by Gavin Lowe [3] and aim to specify the security protocols in an easy way. This Casper description is then transformed into CSP specification and transfered to FDR for verifying security goals. There are some advantages of this language:

– The language is simple and easy to use.
– The protocol declaration is as close as informal description.
– The user does not need to describe the intruder behavior except some initial knowledge.

However, there are still some short-coming of Casper:

– The ability of intruder is set by default (they are inject and deflect ability as in our definition) and can not be changed. In many cases, the intruder ability can be restricted or extended based on evironment but cannot specify in Casper.
– The property descriptions only focus on secrecy and authentication properties. If the user want to verify other properties, he cannot specify it in Casper.
– The time specification is restricted to session level, but not in message level. For example, the user cannot specify the time used to transfer a message from A to B. He can only specify something like: the session between A and B will take 1 time unit (this time specification is used to specify time protocols which has timestamp inside).

HLPSL: HLPSL is the security language which is proposed in AVISPA project [1]. The user can specify the security protocol in HLPSL language and the translator HLPSL2IF (was developed in this project) will translate it into IF language. However, as the aim is translation to IF language, the HLPSL description is proposed toward it:

the protocol is specified in transition states, each state with some conditions will trigger another state. However, from the view of user, it is difficult to described correctly those transition from the informal specification of security protocols. Moreover, only secrecy and authentication properties checking are provided. The intruder ability can not also changed. In the authentication checking, the user need to manually add some signals at the start and end position ofrole runs.

ProVerif: ProVerif [2] is a security protocol language proposed by Bruno Blanchet and used in ProVerif tool. This language is based on pi calculus. The ProVerif description is then translated into an abstract representation by Horn clauses and then using Horn logic technique to verify. However, the protocol's description in ProVerif language is far away from informal specification with many rules and reduction that the users need to define by themselves. Moreover, the user also needs to define the rules for attacker. The timestamp is defined using constant term which is not correct in semantic.

SeVe language From the advantage and disadvantages of other security protocol languages, we define SeVe language which can be considered as the extension of Casper language with some improvements. First of all, we support many kinds of security properties. Secondly, SeVe allow the flexibility in intruder ability. Thirdly, the user are free from specifying intruder rules and behaviors. In addition, SeVe support time specification in message level and in verification.

The next section will show the structure of a SeVe language and the meaning of each keyword. The last section introduces the grammar of SeVe language.

2 SeVe structure

In this section, we demonstrate the structure of a SeVe specification. However, a typical protocol may not need all of these elements. The keyword with obvious name will have the self-explanation meaning.

#Variables —— declare the terms used in protocol
  Timestamps —— the name of timestamps used in protocol
  Time_allow —— the time tolerance in timestamp checking
  Agents —— declare the trusted agents
  Server —— declare the server
  Nonces —— declare the nonces
  Server_keys —— declare the server keys
  Session_keys
  Signature_keys
  Constants ——declare the constant value used in protocol
  Functions ——declare the name of function used in protocol

#Initial —— declare the initial knowledge of participants
  Agent **knows** {knowledge} —— at initial, each agent have some knowledge

#Protocol_description —— main part declare the messages tranfer in protocol
  AgentA $\rightarrow$ AgentB : term (**within**[number])? —— agentA sends agentB a term

which takes number time unit to come

#System —— declare the actual name and number of initiators, and responders
Initiator —— declare the real name and number of initiators in protocols.
Responder —— declare the real name and number of responders in protocols.
Server —— declare the server name.
Repeat —— declare the number of repeated time for each transaction.

#Intruder —— declare the information of intruder here
Intruder —— declare the intruder name
Intruder_knowledge —— declare the initial knowledge of intruder.
Intruder_prepare —— declare the message intruder prepares for agent to send
(used in coercion resistence privacy type)
Intruder_ability —— specify the ability of intruder
(inject, deflect, eavesdrop or jam)

#Verification —— declare the verification properties
Data_secrecy —— secrecy properties
Authentication —— authentication properties
Non_repudiation —— non repudiation properties
Integrity —— Integrity properties
Fairness —— Fairness properties
Privacy —— simple privacy properties
Receipt_freeness —— receiptfreeness privacy properties
Coercion$_resistance||coercionresistanceprivacyproperties$

We also have specification for time checking, such as "if agentA sends message then agentB evetually/always receives message within n time units" (for future verification).

3 SeVe Grammar

Program section

$Program$ ::= **#Variables**
$Variables\_declare$
#Initial
$Initial\_declare$
#Protocol
$Protocol\_declare$
#System
$System\_declare$
[**#Intruder**
$Intruder\_declare$]
[**#Verification**
$Verification\_declare$]

Declaration section

$$
\begin{aligned}
Variables\_declare ::= & [\textbf{Timestamps: } List\_Id]\\
& [\textbf{Time allow: } Number]\\
& \textbf{Agents: } List\_Id\\
& [\textbf{Server: } List\_Id]\\
& [\textbf{Nonces: } List\_Id]\\
& [\textbf{Public\_keys: } List\_Id]\\
& [\textbf{Server\_keys: } List\_Id]\\
& [\textbf{Signature\_keys: } List\_Id]\\
& [\textbf{Session\_keys: } List\_Id]\\
& [\textbf{Constants: } List\_Id]\\
& [\textbf{Functions: } List\_Id]
\end{aligned}
$$

$$
\begin{aligned}
List\_Id \quad ::= & \; Id\\
\mid & \; Id, List\_Id
\end{aligned}
$$

Initial knowledge section

$$
\begin{aligned}
Initial\_declare ::= & \; Id \textbf{ knows } \{msg\}\\
\mid & \; Id \textbf{ knows } \{msg\}, Initial\_declare
\end{aligned}
$$

Protocol description section

$$
\begin{aligned}
Protocol\_declare ::= & \; Id \rightarrow Id : message\\
\mid & \; Id \rightarrow Id : message,\\
& \; Protocol\_declare
\end{aligned}
$$

$$
\begin{aligned}
message \quad ::= & \; msg \textbf{ within}[number]\\
\mid & \; msg
\end{aligned}
$$

$$
\begin{aligned}
msg \quad ::= & \; msg1\\
\mid & \; msg1, msg\\
\mid & \; \{msg\}Id\\
\mid & \; msg \textbf{ + } msg\\
\mid & \; Id(msg) \quad \%function\ declare
\end{aligned}
$$

$$
\begin{aligned}
msg1 ::= & \; Id\\
\mid & \; Id, msg1
\end{aligned}
$$

Actual system section

$$
\begin{aligned}
System\_declare ::= & [\textbf{Initiator: } List\_Id]\\
& [\textbf{Responder: } List\_Id]\\
& [\textbf{Server: } List\_Id]
\end{aligned}
$$

Intruder section

$Intruder\_declare$ ::= **Intruder:** Id
[**Intruder_knowledge:** $msg1$]
[**Intruder_ability:** $List\_ability$]
[**Intruder_prepare:** $List\_prepare$]

$List\_ability$ ::= [**Inject**],
[**Deflect**]
[**Transmit**]
[**Eavesdrop**]
[**Jam**]

$List\_prepare$ ::= $\{msg\}$ **for** Id;
Verification section

$Verification\_declare$ ::= $spec | temporal\_spec$
| $spec, Verification\_declare$
| $temporal\_spec, Verification\_declare$

$spec$ ::= [**Data_secrecy:** $list\_secrecy$]
[**Authentication:** $list\_auth$]
[**Non_repudiation:** $list\_condition1$]
[**Fairness:** $list\_condition2$]
[**Privacy:** Id]
[**Receipt_freeness:** Id]
[**Coercion_resistance:** Id]

$list\_secrecy$ ::= $msg1$ **of** Id
| $msg1$ **of** $Id, list\_secrecy$

$list\_auth$ ::= Id **is authenticated with** Id [**using**$\{Id\}$]
| Id **is authenticated with** Id [**using**$\{msg\}$]$, list\_auth$

$list\_condition1$ ::= $\{Id, Id, msg\}$
| $\{Id, Id, msg\}, list\_condition1$

$list\_condition2$ ::= $\{Id,$ **if** msg **then** $msg\}$
| $\{Id,$ **if** msg **then** $msg\}, list\_condition2$

$temporal\_spec$ ::= **if** $temp\_formula$ **then** $temp\_formula$
[**within**[$number$]]

$temp\_formula$ ::= Id **send** msg
| Id **receive** msg

Basic definition

$$\begin{array}{ll} Identifier & ::= letter\{letter|digit\}^* \\ Number & ::= '1'..'9'\ digit* \\ \\ letter & ::= 'a'..'z'|'A'..'Z'|'\_' \\ digit & ::= '0'..'9' \end{array}$$

References

1. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Modersheim, D. von Oheimb, M. R., J. Santiago, M. Turuani, L. Vigano, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proceedings of CAV'2005*, 2005.
2. B. Blanchet. Automatic Verification of Correspondences for Security Protocols. volume 19, pages 363–434. Journal of Computer Security, 2009.
3. L. Gavin. Casper : A Compiler for the Analysis of Security Protocols. In *Journal of Computer Security*, volume 6, pages 53–84, 1998.