Technical report: 
SeVe implementation

Luu Anh Tuan¹, Jun Sun², Yang Liu¹, and Jin Song Dong¹

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 Security 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 speci-
fication and transferred 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 knowl-
edge.

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 environment 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 → 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 resitance 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_repuadiation —— 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 eventually/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

Variables declare ::= [Timestamps: List_Id] [Time allow: Number] 
Agents: List_Id [Server: List_Id] [Nonces: List_Id] 
Public_keys: List_Id [Server_keys: List_Id] 
Signature_keys: List_Id [Session_keys: List_Id] 
Constants: List_Id [Functions: List_Id]

List_Id ::= Id | Id, List_Id

Initial knowledge section

Initial declare ::= Id knows {msg} | Id knows {msg}, Initial declare

Protocol description section

Protocol declare ::= Id → Id : message | Id → Id : message, Protocol declare

message ::= msg within [number] | msg

msg ::= msg1 | msg1, msg | {msg} Id | msg + msg | Id(msg) % function declare

msg1 ::= Id | Id, msg1

Actual system section

System declare ::= [Initiator: List_Id] [Responder: List_Id] [Server: List_Id]

Intruder section
Intruder \texttt{declare} ::= Intruder: \texttt{Id} \\
\hspace{1cm}[\texttt{Intruder\_knowledge: msg1}] \\
\hspace{1cm}[\texttt{Intruder\_ability: List\_ability}] \\
\hspace{1cm}[\texttt{Intruder\_prepare: List\_prepare}] \\

List\_ability ::= [\texttt{Inject}], \\
\hspace{1cm}\texttt{Deflect}] \\
\hspace{1cm}[\texttt{Transmit}] \\
\hspace{1cm}[\texttt{Eavesdrop}] \\
\hspace{1cm}[\texttt{Jam}] \\

List\_prepare ::= \{msg\} \texttt{for Id;}

\textbf{Verification section}

\begin{align*}
\text{Verification \texttt{declare}} & ::= \texttt{spec|temporal\_spec} \\
\hspace{1cm}\texttt{| spec, Verification\_declare} \\
\hspace{1cm}\texttt{| temporal\_spec, Verification\_declare} \\

\text{spec} & ::= [\texttt{Data\_secrecy: list\_secrecy}] \\
\hspace{1cm}[\texttt{Authentication: list\_auth}] \\
\hspace{1cm}[\texttt{Non\_repudiation: list\_condition1}] \\
\hspace{1cm}[\texttt{Fairness: list\_condition2}] \\
\hspace{1cm}[\texttt{Privacy: Id}] \\
\hspace{1cm}[\texttt{Receipt\_freeness: Id}] \\
\hspace{1cm}[\texttt{Coercion\_resistance: Id}] \\

\text{list\_secrecy} & ::= \texttt{msg1 of Id} \\
\hspace{1cm}| \texttt{msg1 of Id, list\_secrecy} \\

\text{list\_auth} & ::= \texttt{Id is authenticated with Id [using\{Id\}]} \\
\hspace{1cm}| \texttt{Id is authenticated with Id [using\{msg\}], list\_auth} \\

\text{list\_condition1} & ::= \{Id, Id, msg\} \\
\hspace{1cm}| \{Id, Id, msg\}, list\_condition1 \\

\text{list\_condition2} & ::= \{Id, if msg then msg\} \\
\hspace{1cm}| \{Id, if msg then msg\}, list\_condition2 \\

\text{temporal\_spec} & ::= \texttt{if temp\_formula then temp\_formula} \\
\hspace{1cm}[\texttt{within[number]}] \\

\text{temp\_formula} & ::= \texttt{Id send msg} \\
\hspace{1cm}| \texttt{Id receive msg}
Basic definition

Identifier ::= letter{letter|digit}*
Number ::= '1'..'9' digit*

letter ::= 'a'..'z'|'A'..'Z'|
      digit ::= '0'..'9'

References