Energy Efficient Authentication Scheme for Industrial Smart Grid Environments

C. Formal security validation using AVISPA tool

In this section, we verify our claim of mathematical proof of security strength of the proposed scheme using widely accepted Automated Validation of Internet Security Protocols and Applications (AVISPA) tool [1]. This tool test security of proposed scheme against all active and passive attack includes both man-in-the-middle and replay attacks. The AVISPA tool is a push-button method for automatic validation of Internet-safety-sensitive protocols and programs and has been commonly adopted in recent years for standardized security verification [2]. We conducted AVISPA simulation experiment for proposed protocol on SPAN Ubuntu 10.10 Virtual machine consume CPU RAM=2048 MB installed originally on Intel Core i5-8500, 6-core 3.10 GHz machine as operating environments which is originally by service provider. This simulation in AVISPA used SPAN application to check the security strength of the proposed scheme against both active and passive attacks using AVISPA toolset [1]. This AVISPA program modules is written in one of the power languages called as High-Level Specification of Protocol (HLPSL). This languages involves the role that each participating entity plays. The role performed by each entity represent their action for particular scenario. Since for each scenario, role is varying and unique that in other scenarios. For proposed scheme, during authentication phase, node perform the first role of parameter creation $< I_i, X_i, Y_i, P_{xi} >$ (i.e., H$(ID_i.PSW_i).Xi', Yi'.Pxi'$) and sending of parameter to other node. Each role receives the primary information regarding there role action from a parameter which communicates through channels with other roles. An HLPSL2IF translator used in order to converter the HLPSL protocol into the intermediate format specification. Then, intermediate format specification feed as input to one of the four backs ends (i.e., on-the-fly model-checker (OFMC), tree automata based on automatic approximations for the examination of security protocols, constraint-logic-based attack searcher (CL-AtSe), and SAT-based model-checker) to generate output. The numerous important types are used to describe the requirements for every role (See Figs. 5 and 6). This type as follows: 1) agent: it identifies the intruder's principal name by means of a specific identifier i 2) *public_key*: it indicate the proposed scheme public key ($P_{kei} = Psw_i.G,$) for node 3) symmetric key: it represents encryption key for communication. However proposed scheme used open public communication. Then, session key (SK') for encrypting session after successful authentication. 4) const: it specifies the constant defined in role, 5) text: it represents the nonce which is fresh and often new for every session (i.e., for e.g. session key (SK)), used to provide security against attacker. 6) hash func: it represents the one-way hash cryptographic function used in role modeling; 7) nat: its represent message context in the form of natural number.

In our implementation, we used OFMC and CL-AtSe backend, for analysis in AVISPA tool which receive input and carefully analyzed it and precisely generated the output which indicate whether the protocol was in a safe or unsafe condition. The communication channel used during communication of parameters is supposed to be controlled by the Dolev–Yao attacker. It implies the attacker is modeled using the Dolev – Yao model with the

```
role nodedevice_Init (Ui, Uj, TS: agent,
               H:hash_func,
               Send,Recv: channel(dy))
played_by Ui
def=

  local State   : nat,
        U, Xi, V, G, Ri, Rj, Yi,Pxi, S, Pyi, Ii,PKi, PKj:text,
        IDi, IDj, Pswi, Pswj, PKs, Ij, Xj, Yj, YjG, YjGc, SK:text,
        F:hash_func

  const sp1, sp2, ui_uj_xi, uj_ui_xj : protocol_id

  init  State := 0

  transition

   1. State = 0
      /\ Recv(start)
      =|>
%Authentication Phase
      State' := 1
       /\ U' := new()
       /\ Xi':= F(U'.G)
       /\ Pxi':= F(Ri.G)
       /\ Pyi':= xor(Ri, S.H(H(IDi.Pswi).F(Pswi.G)))
       /\ Yi':=  xor(U', Pyi'.H(Ij.U'.PKj))
       /\ secret({Ri,S}, sp1, {TS})
       /\ secret({IDi,Pswi}, sp2, {Ui})
       /\ secret({Ij}, sp3, {Uj})
%Send authentication request message <Ii, Xi, Yi, Pxi> to Uj
% via public channel
      /\ Send(H(IDi.Pswi).Xi'.Yi'.Pxi')
% Ui has freshly generated the values xi' for the Uj
     /\ witness(Ui, Uj, ui_uj_xi, U')
% Recive authentication response message <Xj, Yj, Pxj>
% from Uj via public channel

   2. State = 1
      /\ Recv(F(V'.G).xor(V', xor(Rj, S.H(H(IDj.Pswj).F(Pswj.G)))).
F(Rj.G))
      =|>
      State' := 2
       /\U' := new()
%Compute and Checks :
%Computed one
       /\ YjG' := xor(F(V'.G), xor(F(Rj.G),
F(S.G).H(H(IDj.Pswj).F(Pswj.G))).H(H(IDi.Pswi).F(V'.G).Pswi))
%Recived one
       /\ YjGc':= xor(V', xor(Rj,
S.H(H(IDj.Pswj).F(Pswj.G)).H(H(IDi.Pswi).V'.F(Pswi.G)))).G
% Authentication successful if challenge YjG==YjGc'
       /\ SK':= U'.Pswi.F(V'.G).F(Pswj.G)
% Send authentication acknowledgment message <SK'>
% Ui is acceptance of the values xj' generated for Ui by Uj
   /\ request(Uj, Ui, uj_ui_xj, V')
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Fig. 5. Role specification for node devices

probability of the attacker having a legitimate role in running a protocol. Many of the basic roles are specified by the session role. The role environment is also called as starting point for execution which is top-level role. This role environment instantiates session role using distinctive basic roles to simulate various possible scenarios. Finally, in the goal section, we define identified requirements and appropriate goals according to our preconditions of the designed protocol. We wrote two primary roles when writing the code in AVISPA: first for the node devices, and second for the session. Afterwards, we wrote three further roles: first for the session, second for the environment and third for the goal. The last three roles represent the execution environment of the first two roles. The Fig. 5 describe the specific role going to be performed by node devices. Initially, when we used tamper proof node devices, registration processes are done by expert of company. Therefore, device Ui receive and store two challenges $< Pxi', Pyi' >$ after their successful registration performed by expert. Then Ui is hand over to the owner. As soon as node Ui receive start signal, the Ui update its state from 0 to 1. This indicate the starting of role action for Ui. When Ui initiative communication with Uj, it compute challenge and send $<$ H$(ID_i.PSW_i).Xi', Yi'.Pxi' >$ to the Uj through open public Send channel; We term it as authentication process. The channel used for communication <Send, Recv>, is

```
role session(Ui, Uj, TS: agent,
              H:hash_func)
def=

  local S1, R1, S2, R2: channel (dy)

  composition
     nodedevice_Init(Ui, Uj, TS,H,S1,R1)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


role environment()
def=
  const ui, uj,ts : agent,
        h, f: hash_func,
        sp1, sp2, sp3, ui_uj_xi, uj_ui_xj : protocol_id

  intruder_knowledge={h,f}

  composition
      session(ui,uj,ts, h)
   /\ session(i,uj, ts, h)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal
 secrecy_of sp1, sp2, sp3
 authentication_on ui_uj_xi
 authentication_on uj_ui_xj

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

environment()
```

Fig. 6. Role specification for the session, goal, and environment

insecure medium that is used to transmit message of a Dolev – Yao threat model type; it helps the intruder to modify or erase the content of transmitted message. Then in answer to the Ui authentication request, Ui receive response message $< X_j, Y_j, P_{xj} >$ (i.e., *<F(V'.G).xor(V', xor(Rj,S.H(H(IDj.Pswj).F(Pswj.G)))). F(Rj.G)>*) through open public Recv channel (see Fig. 5). A declaration of knowledge located at the top of each role is used to determine the intruder's initial awareness. The transition to immediate reaction comes from the process X= | >Y, which connects event X to actions Y. The term witness < *(Ui, Uj, ui_uj_xi, U')*> is indicate declaration witness generated by node Ui, where node Ui freshly generate random nonce $xi1$ for the node Ui. Subsequently, term request < *(Uj, Ui, uj_ui_xj, V')*> indicates the node Ui acceptance of random nonce $xj1$, created for Ui by Uj. Since, role of both node Ui and Uj are similar while initiating authentication request.

Finally, we implemented role of the session, goal and environment for the proposed scheme (). Those roles in the role session and goal and environment contain instances with solid arguments. The attacker (i) in the HLPSL protocol for proposed scheme, was also involved in protocol execution in a specific session in the HLPSL procedure, as seen in Fig. 6. We also defined three secrete goal and two authentication goals. The secrete goal (secrecy_of sp1,) define parameters { R_i , s} kept secretes to TS only. The authentication goal authentication_on ui_uj_xi (i.e., authentication_on alice_bob_ xi) means that ui uniquely generate challenge U' for node uj, where U' known only to ui and ui hide it inside <Xi',Yi'>. Then another node uj receive challenge U' indirectly thought <Xi',Yi'> of ui message. Then uj perform authentication verification step and verify ui authenticity. Then in response, uj choose unique challenge V' for node ui and uj hide it inside < F(V'.G).xor(V', xor(Rj, S.H(H(IDj.Pswj).F(Pswj.G)))> (i.e., < Xj',Yj'>) and send it back to ui. Then ui verify the authententicity of node uj. For successful authentication, node ui compute session key (SK':= U'.Pswi.F(V'.G).F(Pswj.G)).

For the execution tests we selected widely accepted CL-AtSe and OFMC backend for the protocol review. By looking for the passive as well as active attacker, the CL-AtSe and OMFC backend analyzes whether the legal agents will execute the specific scheme.

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 /home/span/span/testsuite/results/paper4.if

GOAL
 As Specified

BACKEND
 CL-AtSe

STATISTICS

 Analysed   : 1 states
 Reachable  : 1 states
 Translation: 0.00 seconds
 Computation: 0.00 seconds
```

```
% OFMC
% Version of 2006/02/13
SUMMARY
 SAFE
DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
 /home/span/span/testsuite/results/paper4.if
GOAL
 as_specified
BACKEND
 OFMC
COMMENTS
STATISTICS
 parseTime: 0.00s
 searchTime: 0.02s
 visitedNodes: 2 nodes
 depth: 1 plies
```

Fig. 7. Analysis result using the CL-AtSe and OFMC backend

Consequently, backend results consequently provide the intruders the information of certain typical sessions among the legal agents or nodes (see Fig. 7). The overview section in the backend of CL-AtSe and OFMC shows whether the protocol is SAFE, UNSAFE, or INCONCLUSIVE. The output section presents SAFE SUMMARY, which indicates that the proposed protocol is protected from significant attacks. In addition, the section details state that the situation in which the suggested scheme is secure or has been used to detect an attack, and therefore also explain the reason to which the analysis was not inconclusive. In addition, the section regarding protocol specifies the protocol name. The section on goals shows the key objective of the analysis. The section on the backend represents the name of backend used by analysis. The section Statistics reflects the time that the backend takes to execute the protocol. The attack-trace section determines whether an attack is found; if attack found then the record of the attack is written in the traditional Alice – Bob format. Therefore, based on an interpretation of the outcome of the simulation, we conclude that the proposed protocol is secure as follows:

1. Replay attack: The CL-AtSe and OFMC backend confirm if the legitimate devices will execute the specified protocol by inquiring through an inactive attacker for the replay-attack test. This backend offers the interloper the details between the legitimate nodes of certain normal sessions. The test result is safe. The outcome conclude that our scheme is secure against the replay attack.

2. Active and passive attack check: The outline results for CL-AtSe and OFMC backend reveals that the scheme proposed is SAFE, indicating that the proposed scheme is secure against all active and passive attacks [3].

3. Dolev–Yao model check: The CL-AtSe and OFMC backend confirms if there is some man-in - the-middle attack possible by an attacker for performing the Dolev – Yao model test [3]. Analysis result in Fig. 7 conclude that proposed scheme satisfies the secure design architecture.

D. Formal security validation using ProVerif tool

The ProVerif is another widely accepted automatic cryptographic protocol verifier simulation tool in the symbolic model (i.e., so-called Dolev–Yao model) used in verifying the mutual authentication and session key security for an authentication scheme [4]. In ProVerif simulation, the protocol verification is done on the basis of Horn clauses representing the protocol. We conduct the Proverif simulation for the proposed protocol on Intel Core i5-8500, 6-core 3.10 GHz CPU, with a service provider's Window 10 operating system. This simulation used a binary package of ProVerif version 2.00 [5] to trigger the phase of registration, mutual authentication and session key agreement with node devices. The detail description regarding the ProVerif simulation tool can be found in the official site (See

Figs. 8 and 9). The model output for the proposed scheme in Fig. 10 is generated in ProVerif. This model consists of three important fragments, which includes parameter declaration, query execution and principal processes defining. The parameter-declaration fragment presents several channels, constants, variables, and factors other than the cryptographic capacities outlined as constructors and conditions. The definitions of the principal process and sub-process are clarified in the process section while in the query-execution section the layout of the scheme evaluation is provided. In addition, the query execution fragment characterizes the start and finish of the participating node events. Since, the execution process for both the participating nodes are run parallel. Toward starting of execution (See Fig. 10), first two queries executed successfully which specify the successful interaction and mutual authentication between node devices. The successful execution of last query for proposed scheme specify that proposed scheme is secure against all kind of active and passive attacks on session key confirm its secrecy because of an unsuccessful query attack on the session key.

```
(*********Declaration of parameters**************)

(************** Channels************************)
free SCh: channel [private].     (*Secure Channel*)
free PCh: channel.               (*Public Channel*)
(************Constants & Varibles **************)
const G: bitstring.
free IDi: bitstring.
free PSWi: bitstring  [private].
free IDj: bitstring.
free PSWj: bitstring  [private].
free PKs: bitstring.
free s: bitstring  [private].
(**************Constructor**********************)
fun h1(bitstring, bitstring) : bitstring.
fun h2(bitstring, bitstring) : bitstring.
fun h3(bitstring) : bitstring.
fun ECPM(bitstring, bitstring) : bitstring.
fun MULT(bitstring, bitstring) : bitstring.
fun XOR(bitstring, bitstring) : bitstring.
fun CONCAT(bitstring, bitstring) : bitstring.

(*********End of declaration of parameters*******)

(**********Events and queries executation********)

(************** Events************************)
event beginDeviceUi(bitstring).
event endDeviceUi(bitstring).
event beginDeviceUj(bitstring).
event endDeviceUj(bitstring).
(************** Queries************************)
free SK: bitstring [private].
query id : bitstring; inj-event(endDeviceUi(id)) ==>
     inj-event(beginDeviceUi(id)).
query id: bitstring; inj-event(endDeviceUj(id)) ==>
     inj-event(beginDeviceUj(id)).
query attacker(SK).

(*****End of events and eueries executation*****)
```

Fig. 8. Declaration of parameters with event and query execution

```
(***************Processs************************)
(***************DeviceUi************************)
let DeviceUi=
(***************Registraion*********************)
let Ii = h3(CONCAT(IDi,G)) in
let PKei = ECPM(PSWi,G) in
out(SCh, (Ii, PKei));
in(SCh, (xPix:bitstring, xPiy:bitstring));
(***************Login and Authentication*********)
event beginDeviceUi(Ii);
new xi:bitstring;
new PKej:bitstring;
new Ij:bitstring;
let Xi = ECPM(xi,G) in
let Yi = XOR(xi, MULT(xPiy, h1(Ij, ECPM(Xi, PKej)))) in
out(PCh, (Ii, Xi, Yi, xPix));
in(PCh, (xXj:bitstring, xYj:bitstring, xPjx:bitstring));
let YjG = ECPM(xYj,G) in
let YjG'= MULT(XOR(xXj, (xPjx, MULT(h2(Ij, PKej),
ECPM(s,G)))), h1(Ii, ECPM(xXj,PSWi))) in
if (YjG = YjG') then
let (=SK)= MULT(MULT(xi,xXj), MULT(PSWi,PKej)) in
event endDeviceUi(Ii)
else 0.
(***************DeviceUj************************)
let DeviceUj=
(***************Registraion*********************)
let Ij = h3(CONCAT(IDj,G)) in
let PKej = ECPM(PSWj,G) in
out(SCh, (Ij, PKej));
in(SCh, (xPjx:bitstring, xPjy:bitstring));
(***************Login and Authentication*********)
event beginDeviceUj(Ij);
in(PCh, (xIi:bitstring, xXi:bitstring, xYi:bitstring,
xxPix:bitstring));
new PKei:bitstring;
new Ii:bitstring;
let YiG = ECPM(xYi,G) in
let YiG' = MULT(XOR(xXi,(xxPix, MULT(h2(Ii,PKei),
ECPM(s,G)))), h2(Ij, ECPM(xXi, PSWj))) in
if (YiG = YiG') then
new xj:bitstring;
let Xj = ECPM(xj,G) in
let Yj = XOR(xj, MULT(xPjy, h1(Ii, ECPM(xj, PKei)))) in
let (=SK)= MULT(MULT(xj,xXi), MULT(PSWj,PKei)) in
out(PCh, (Xj, Yj, xPjx));
event endDeviceUj(Ij)
else 0.
(*********************TTP************************)
let TTP=
(***************Registration********************)
in(SCh, (xIi:bitstring,xPKei:bitstring));
new Ri:bitstring;
let Pix = ECPM(Ri,G) in
let Piy = XOR(Ri,MULT(s, h2(xIi, xPKei))) in
out(SCh, (Pix, Piy));
in(SCh, (xIj:bitstring,xPKej:bitstring));
new Rj:bitstring;
let Pjx = ECPM(Rj,G) in
let Pjy = CONCAT(Rj,MULT(s, h2(xIj, xPKej))) in
out(SCh, (Pjx, Pjy));
0.
process ((!DeviceUi) | (!TTP) | (!DeviceUj))
```

Fig. 9. Process specification for embedded device and server

ProVerif text output:

```
-- Query inj-event(endDeviceUi(id)) ==> inj-event(beginDeviceUi(id))
nounif mess(SCh[],(xIi_450,xPKei_451))/-5000
Completing...
Starting query inj-event(endDeviceUi(id)) ==> inj-event(beginDeviceUi(id))
RESULT inj-event(endDeviceUi(id)) ==> inj-event(beginDeviceUi(id)) is true.
-- Query inj-event(endDeviceUj(id_16)) ==> inj-event(beginDeviceUj(id_16))
nounif mess(SCh[],(xIi_1282,xPKei_1283))/-5000
Completing...
Starting query inj-event(endDeviceUj(id_16)) ==> inj-event(beginDeviceUj(id_16))
RESULT inj-event(endDeviceUj(id_16)) ==> inj-event(beginDeviceUj(id_16)) is true.
-- Query not attacker(SK[])
nounif mess(SCh[],(xIi_2063,xPKei_2064))/-5000
Completing...
Starting query not attacker(SK[])
RESULT not attacker(SK[]) is true.
```

Fig. 10. Execution output of query

REFERENCES

[1] Avispa-project.org. (2020). AVISPA Web Tool. [online] Available at: http://www.avispa-project.org/webinterface/basic.php [Accessed 16 Mar. 2020].

[2] David Von Oheimb. The high-level protocol specification language helps developed in the eu project avispa. InProceedings of APPSEM 2005workshop, pages 1–17, 2005

[3] Wazid, M., Das, A.K., Bhat, V. and Vasilakos, A.V., 2020. LAMCIoT: Lightweight authentication mechanism in a cloudbased IoT environment. Journal of Network and Computer Applications, 150, p.102496.

[4] Maitra, T., Obaidat, M.S., Amin, R., Islam, S.H., Chaudhry, S.A.and Giri, D., 2017. A robust ElGamal based password-authentication protocol using smart card for client-server communication. International Journal of Communication Systems, 30(11), p.e3242.

[5] "Cryptographic Protocol Verifier in the Formal Model." ProVerif, prosecco.gforge.inria.fr/personal/bblanche/proverif/.