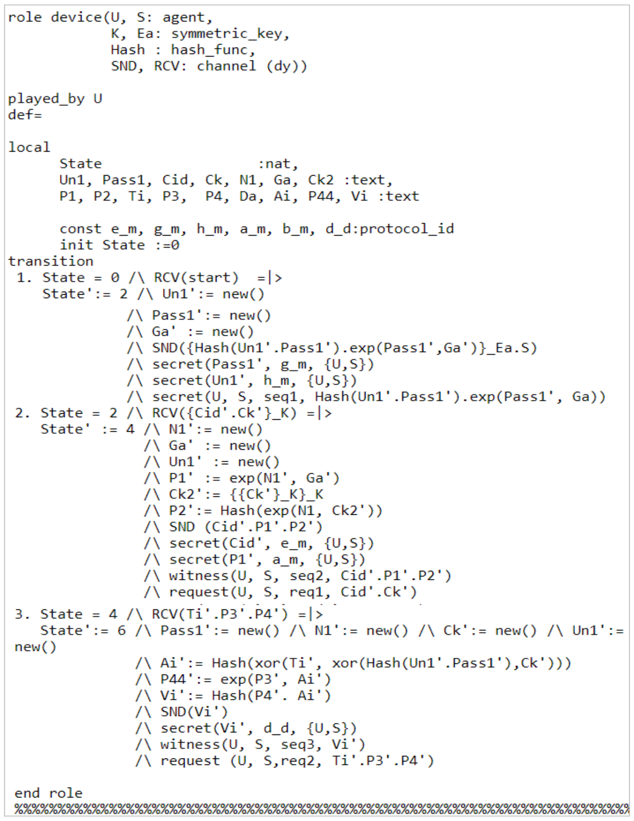
Energy Efficient Authentication Scheme for Industrial Smart Grid Environments

C. Formal security validation using AVISPA tool

In this section, we perform the simulation of the proposed scheme by utilizing the AVISPA tool [1] and ensure that the proposed scheme is secure against both man-in-the-middle and replay attacks. The AVISPA tool is a push-button tool for performing the automated validation of Internet-security-sensitive protocols and applications, and it has become widely accepted for formal security verification, in recent years [2]. The AVISPA tool is coded in one of the power languages (i.e., high-level protocol specification language (HLPSL)). This language comprises the role that represents each participating activity. The role in a scenario is separate/different from that in other scenarios. The role receives the primary information from a parameter that communicates with other roles through channels. Furthermore, the HLPSL protocol is translated to the intermediate format specification by using an HLPSL2IF translator. The input is given 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 create yield. We implemented the proposed scheme code on a SPAN Ubuntu 10.10 virtual machine [1] with RAM=2048 Gb. This experimental set up install on originally window 10 PC with an Intel Core i5-8500,6-core 3.10 GHz CPU provided by the service provider. The Avispa simulation used SPAN software to assess the security strength of the proposed protocol against both active and passive attacks using AVISPA toolset [1].

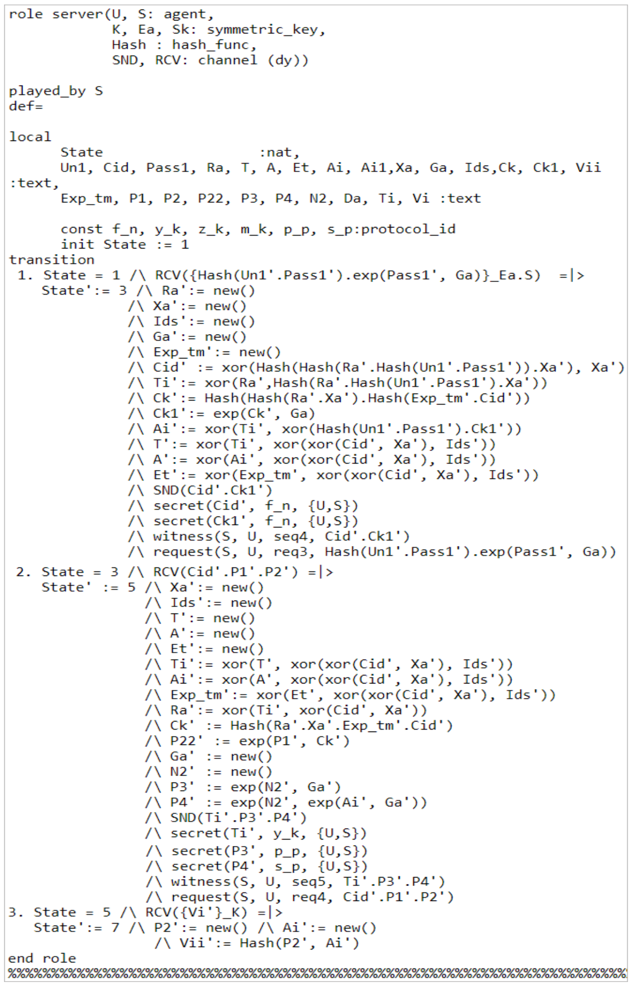
The numerous basic types are used to define specific criteria for the roles. For e.g., agent: the agent determines the principle of the attacker receives the data, analyses it, and provides recommendations for the protocol being designed by correctly defining the state of whether the protocol is in a secure or dangerous state. The different basic types are utilized for defining the specifications of each role. Some of them are as follows: 1) *agent*: it defines the principal name of the intruder by using a special identifier *i*, 2) *public_key*: it represents the public key, 3) *symmetric_key*: it represents the key used for encryption, 4) *const*: it defines the constant declared in roles, 5) *text*: it represents the nonce that is always fresh and unique, and it secures the message from an attacker, 6) *function*: it represents the irreversible one-way hash cryptography function of type hash_func used for modeling, and 7) *nat*: it represents the natural number in a no message context. The AVISPA tool received input as a designed protocol, analyzed it, and precisely generated the output by portraying the state whether the protocol was in a safe or an unsafe state. The channel is used for communication, which is supposed to be controlled by the Dolev–Yao attacker. This means that the intruder is modeled using the Dolev–Yao model with the possibility that the intruder might assume a legitimate role in a protocol run. The session role defines all the basic roles. The role environment is a top-level role, and it is the beginning point for the execution; furthermore, it instantiates a session role utilizing distinctive basic roles to simulate various possible scenarios. Finally, in the goal section, according to our prerequisites of the designed protocol, we define all the necessary and sufficient goals. While writing the code in



```
role device(U, S: agent,
            K, Ea: symmetric_key,
            Hash : hash_func,
            SND, RCV: channel (dy))

played_by U
def=

local
      State                  :nat,
      Un1, Pass1, Cid, Ck, N1, Ga, Ck2 :text,
      P1, P2, Ti, P3,  P4, Da, Ai, P44, Vi :text

      const e_m, g_m, h_m, a_m, b_m, d_d:protocol_id
      init State :=0
transition
 1. State = 0 /\ RCV(start)  =|>
    State':= 2 /\ Un1':= new()
              /\ Pass1':= new()
              /\ Ga' := new()
              /\ SND({Hash(Un1'.Pass1').exp(Pass1',Ga')}_Ea.S)
              /\ secret(Pass1', g_m, {U,S})
              /\ secret(Un1', h_m, {U,S})
              /\ secret(U, S, seq1, Hash(Un1'.Pass1').exp(Pass1', Ga))
 2. State = 2 /\ RCV({Cid'.Ck'}_K) =|>
    State' := 4 /\ N1':= new()
                /\ Ga' := new()
                /\ Un1' := new()
                /\ P1' := exp(N1', Ga')
                /\ Ck2':= {{Ck'}_K}_K
                /\ P2':= Hash(exp(N1, Ck2'))
                /\ SND (Cid'.P1'.P2')
                /\ secret(Cid', e_m, {U,S})
                /\ secret(P1', a_m, {U,S})
                /\ witness(U, S, seq2, Cid'.P1'.P2')
                /\ request(U, S, req1, Cid'.Ck')
 3. State = 4 /\ RCV(Ti'.P3'.P4') =|>
    State':= 6 /\ Pass1':= new() /\ N1':= new() /\ Ck':= new() /\ Un1':=
 new()
               /\ Ai':= Hash(xor(Ti', xor(Hash(Un1'.Pass1'),Ck')))
               /\ P44':= exp(P3', Ai')
               /\ Vi':= Hash(P4'. Ai')
               /\ SND(Vi')
               /\ secret(Vi', d_d, {U,S})
               /\ witness(U, S, seq3, Vi')
               /\ request (U, S,req2, Ti'.P3'.P4')

 end role
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Fig. 1. Role specification for Smart Meter

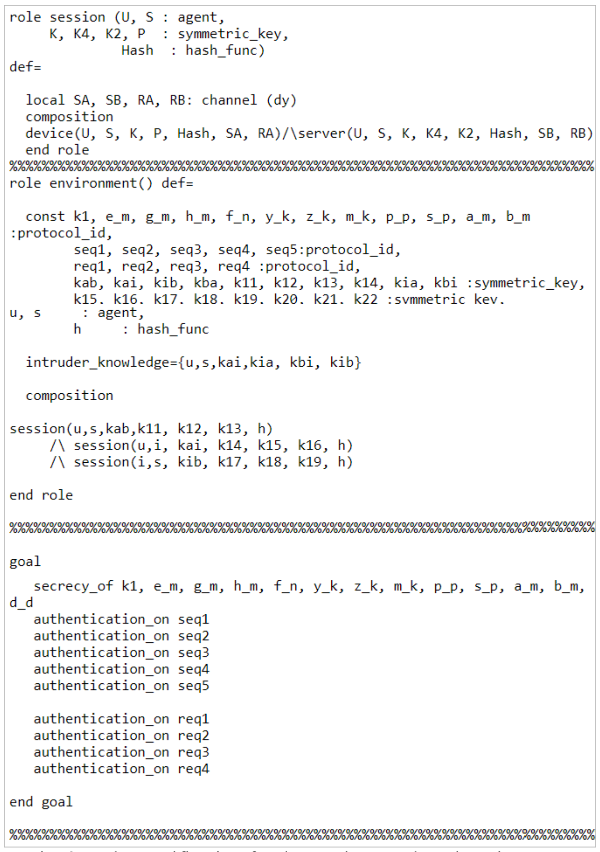
AVISPA, we wrote two primary roles: first for the smart meter (i.e.., embedded device (SM)) and the second for the server (i.e., service provider (SP)). Subsequently, we wrote another three roles: first for the session, second for the environment, and the third for the goal. The last three roles represent the execution environment of the first two roles. Fig. 1 depicts the specific role performed by the agent. Upon receiving the start signal, the embedded device U (i.e., SM_i) updates its state from 0 to 1. This state is retained by a variable state. Subsequently, U sends the registration request <Hash $(Un1'.Pass1')$. exp$(Pass1', Ga')$ > (i.e., <I_i, P_{ke}>) securely to the server through a secure channel (*SND*); we call this the registration phase. The transmission channel <*SND, RCV*>, which is an unreliable channel, is used for the message transmission of the type of Dolev–Yao threat model; it enables an attacker to modify or delete the contents of transmitted messages. Afterward, in response, U receives the message $< Cid', Ck' >$ (i.e., $< CID_i , CK' >$) from the server securely by the help of the secure RCV channel. During the login phase, U sends a message $< Cid', P1', P2' > (i.e., < CID_i, X_i, Y_i >)$ to the server via the public open channel (See Fig. 2). Subsequently, the server responds with a message $< T', P3', P4' >$ (i.e., $< T, X_j, Y_j >$) to U via the public open channel. Finally, SM_i replies with the message <Vi'> (i.e., <C>) to the server via



```
role server(U, S: agent,
            K, Ea, Sk: symmetric_key,
            Hash : hash_func,
            SND, RCV: channel (dy))

played_by S
def=

local
      State                  :nat,
      Un1, Cid, Pass1, Ra, T, A, Et, Ai, Ai1,Xa, Ga, Ids,Ck, Ck1, Vii
:text,
      Exp_tm, P1, P2, P22, P3, P4, N2, Da, Ti, Vi :text

      const f_n, y_k, z_k, m_k, p_p, s_p:protocol_id
      init State := 1
transition
 1. State = 1 /\ RCV({Hash(Un1'.Pass1').exp(Pass1', Ga)}_Ea.S)  =|>
    State':= 3 /\ Ra':= new()
              /\ Xa':= new()
              /\ Ids':= new()
              /\ Ga':= new()
              /\ Exp_tm':= new()
              /\ Cid' := xor(Hash(Hash(Ra'.Hash(Un1'.Pass1')).Xa'), Xa')
              /\ Ti':= xor(Ra',Hash(Ra'.Hash(Un1'.Pass1').Xa'))
              /\ Ck':= Hash(Hash(Ra'.Xa').Hash(Exp_tm'.Cid'))
              /\ Ck1':= exp(Ck', Ga)
              /\ Ai':= xor(Ti', xor(Hash(Un1'.Pass1').Ck1'))
              /\ T':= xor(Ti', xor(xor(Cid', Xa'), Ids'))
              /\ A':= xor(Ai', xor(xor(Cid', Xa'), Ids'))
              /\ Et':= xor(Exp_tm', xor(xor(Cid', Xa'), Ids'))
              /\ SND(Cid'.Ck1')
              /\ secret(Cid', f_n, {U,S})
              /\ secret(Ck1', f_n, {U,S})
              /\ witness(S, U, seq4, Cid'.Ck1')
              /\ request(S, U, req3, Hash(Un1'.Pass1').exp(Pass1', Ga))
 2. State = 3 /\ RCV(Cid'.P1'.P2') =|>
    State' := 5 /\ Xa':= new()
                /\ Ids':= new()
                /\ T':= new()
                /\ A':= new()
                /\ Et':= new()
                /\ Ti':= xor(T', xor(xor(Cid', Xa'), Ids'))
                /\ Ai':= xor(A', xor(xor(Cid', Xa'), Ids'))
                /\ Exp_tm':= xor(Et', xor(xor(Cid', Xa'), Ids'))
                /\ Ra':= xor(Ti', xor(Cid', Xa'))
                /\ Ck' := Hash(Ra'.Xa'.Exp_tm'.Cid')
                /\ P22' := exp(P1', Ck')
                /\ Ga' := new()
                /\ N2' := new()
                /\ P3' := exp(N2', Ga')
                /\ P4' := exp(N2', exp(Ai', Ga'))
                /\ SND(Ti'.P3'.P4')
                /\ secret(Ti', y_k, {U,S})
                /\ secret(P3', p_p, {U,S})
                /\ secret(P4', s_p, {U,S})
                /\ witness(S, U, seq5, Ti'.P3'.P4')
                /\ request(S, U, req4, Cid'.P1'.P2')
3. State = 5 /\ RCV({Vi'}_K) =|>
   State':= 7 /\ P2':= new() /\ Ai':= new()
                 /\ Vii':= Hash(P2', Ai')
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Fig. 2. Role specification for service provider server

the public open channel. A knowledge declaration situated at the top of each role is used to specify the initial knowledge of the intruder. The immediate-reaction transition is of the form X= | >Y, which relates an event X to an action Y. The declaration witness $<U, S, seq2, Cid'.P1'.P2'>$ is generated by U, where the seq2 indicates message sequence (i.e., message $< alice_{bob}$ _ $Cid'.P1'.P2' >$ from alice to bob) and $< P1', P2' >$ is freshly generated for the server (SP). Subsequently again, another declaration request $<S, U, seq5, T'.P3'.P4' >$ shows the U acceptance of the random nonce $< P3', P4' >$ generated for ED_i by the server, where the seq5 indicates message sequence (i.e., message $< bob_{alice}$ _ $T'.P3'.P4' >$ from bob to alice). Similarly, we executed the role of the server during the registration phase (see Fig. 2). Finally, we implemented the role of the session, goal, and environment of the proposed scheme (see Fig. 3). All these roles are the instances with solid arguments in the role session. In the HLPSL protocol, the intruder (i) likewise was interested in the execution of protocol in a concrete session, as depicted in Fig. 3. Furthermore, we defined many secret goals and nine authentication goals. For example, the secret goal (secrecy_of_k1) indicates that ID_i and Psw_i are kept secret from the device U only. The authentication goal: authentication_on seq2 (i.e., authentication_on alice_bob_ $, Cid'.P1'.P2')$ means that means that U generates $< Cid'.P1'.P2' >$, where secrete $N1'$ hidden inside $<P1', P2'>$ is only known to U. When the server receives $< Cid'.P1'.P2' >$ from other messages from the same U, the server performs a strong authentication for the devices based on $<$

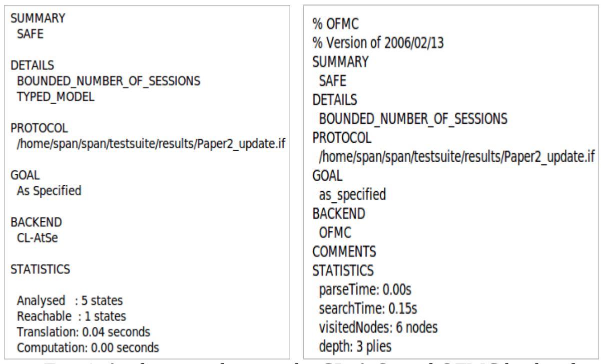


```
role session (U, S : agent,
     K, K4, K2, P  : symmetric_key,
              Hash  : hash_func)
def=

  local SA, SB, RA, RB: channel (dy)
  composition
  device(U, S, K, P, Hash, SA, RA)/\server(U, S, K, K4, K2, Hash, SB, RB)
  end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=

  const k1, e_m, g_m, h_m, f_n, y_k, z_k, m_k, p_p, s_p, a_m, b_m
:protocol_id,
        seq1, seq2, seq3, seq4, seq5:protocol_id,
        req1, req2, req3, req4 :protocol_id,
        kab, kai, kib, kba, k11, k12, k13, k14, kia, kbi :symmetric_key,
        k15. k16. k17. k18. k19. k20. k21. k22 :svmmetric kev.
u, s     : agent,
        h     : hash_func

  intruder_knowledge={u,s,kai,kia, kbi, kib}

  composition

session(u,s,kab,k11, k12, k13, h)
     /\ session(u,i, kai, k14, k15, k16, h)
     /\ session(i,s, kib, k17, k18, k19, h)

end role

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

goal

   secrecy_of k1, e_m, g_m, h_m, f_n, y_k, z_k, m_k, p_p, s_p, a_m, b_m,
d_d
   authentication_on seq1
   authentication_on seq2
   authentication_on seq3
   authentication_on seq4
   authentication_on seq5

   authentication_on req1
   authentication_on req2
   authentication_on req3
   authentication_on req4

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Fig. 3. Role specification for the session, goal, and environment

$P1', P2' >$. For analyzing the protocol, we selected widely accepted CL-AtSe backend and OFMC for the execution tests. Both CL-AtSe and OFMC backend analyzes whether the legal agents can execute the specific scheme by searching for the passive intruder. Subsequently, backend results provide the intruders with the knowledge of some typical sessions among the legitimate agents (see Fig. 4). The section summary in CL-AtSe and OFMC backend indicates whether the protocol is SAFE, UNSAFE, or INCONCLUSIVE. Our output summary section shows the safe terminology, meaning that the proposed protocol is safe against significant attacks. Furthermore, the section details specify that the condition under which the proposed protocol is safe or has been used for finding an attack, or finally, and they also specify why the analysis was inconclusive. Furthermore, the protocol section specifies the name of the protocol. The goal section indicates the main objective of the analysis. The backend section represents the name of the backend used. The statistics section specifies why the analysis was inconclusive. Furthermore, the protocol section specifies the name of the protocol. The goal section indicates the main objective of the analysis. The backend section represents the name of the backend used. The statistics section represents the time required by the backend to execute the protocol. The attack-trace section specifies whether an attack is found; the trace of the attack is printed in the standard seq (i.e., Alice–Bob) format. It also represents how the attack has been performed in the protocol. Therefore, on the basis of analyzing the simulation result, we conclude that the proposed protocol is safe as follows:

1) Replay attack: For the replay-attack check, the CL-AtSe and OFMC backend confirm whether the genuine agents can execute the specified protocol by inquiring an inactive intruder. This backend gives the interloper the information of some normal sessions among the genuine agents. The test outcomes appeared.



```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /home/span/span/testsuite/results/Paper2_update.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 5 states
  Reachable  : 1 states
  Translation: 0.04 seconds
  Computation: 0.00 seconds
```

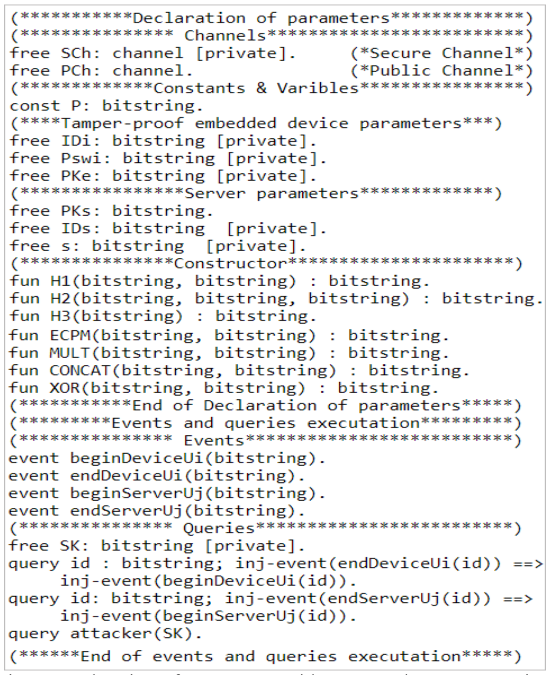
```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/Paper2_update.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.15s
  visitedNodes: 6 nodes
  depth: 3 plies
```

Fig. 4. Analysis result using the CL-AtSe and OFMC backend

Fig. 4 exhibit that our scheme is secure against the replay attack.
2) Active and passive attack check: The outline result for CL-AtSe and OFMC backend shows that the proposed scheme is SAFE, meaning that the proposed scheme is secure against all active and passive attacks [2, 3].
3) Dolev–Yao model check: For performing the Dolev–Yao model check, the CL-AtSe, and OFMC backend additionally confirm whether there is any man-in-the-middle attack conceivable by an intruder. The outcomes in Fig. 4 show that our scheme satisfies the design properties and that it is additionally secure under this backend.

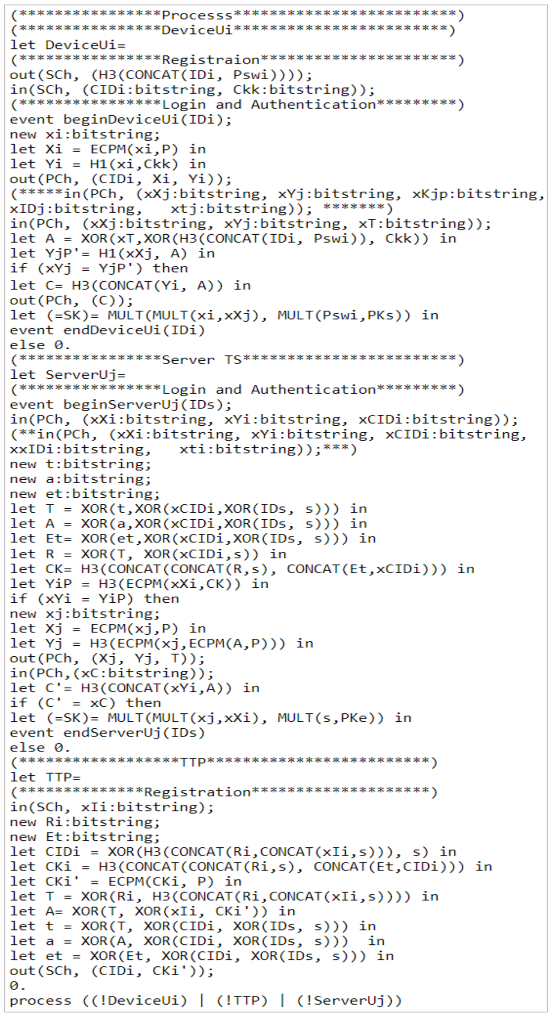
D. Formal security validation using ProVerif tool

ProVerif is widely accepted as an automatic cryptographic protocol verifier tool, in the symbolic model (so-called Dolev–Yao model) [4]. This protocol verifier depends on the portrayal of the protocol by Horn clauses. In this subsection, we implemented the proposed scheme code in the Proverif tool to test its correctness of mutual authentication. We perform this experiment on Intel Core i5-8500, 6-core 3.10 GHz CPU with window 10 operating system provided by a service provider. This simulation used ProVerif version 2.00 binary package [5] to stimulate the registration, mutual authentication, and session key agreement phase between an embedded device. The description of the ProVerif simulation tool can be found in [4] (see Figs. 5 and 6). The model output of the



```
(***********Declaration of parameters*************)
(*************** Channels*************************)
free SCh: channel [private].      (*Secure Channel*)
free PCh: channel.                (*Public Channel*)
(*************Constants & Varibles****************)
const P: bitstring.
(****Tamper-proof embedded device parameters***)
free IDi: bitstring [private].
free Pswi: bitstring [private].
free PKe: bitstring [private].
(****************Server parameters*************)
free PKs: bitstring.
free IDs: bitstring  [private].
free s: bitstring  [private].
(***************Constructor**********************)
fun H1(bitstring, bitstring) : bitstring.
fun H2(bitstring, bitstring, bitstring) : bitstring.
fun H3(bitstring) : bitstring.
fun ECPM(bitstring, bitstring) : bitstring.
fun MULT(bitstring, bitstring) : bitstring.
fun CONCAT(bitstring, bitstring) : bitstring.
fun XOR(bitstring, bitstring) : bitstring.
(***********End of Declaration of parameters*****)
(*********Events and queries executation*********)
(*************** Events**************************)
event beginDeviceUi(bitstring).
event endDeviceUi(bitstring).
event beginServerUj(bitstring).
event endServerUj(bitstring).
(*************** Queries*************************)
free SK: bitstring [private].
query id : bitstring; inj-event(endDeviceUi(id)) ==>
     inj-event(beginDeviceUi(id)).
query id: bitstring; inj-event(endServerUj(id)) ==>
     inj-event(beginServerUj(id)).
query attacker(SK).
(******End of events and queries executation*****)
```

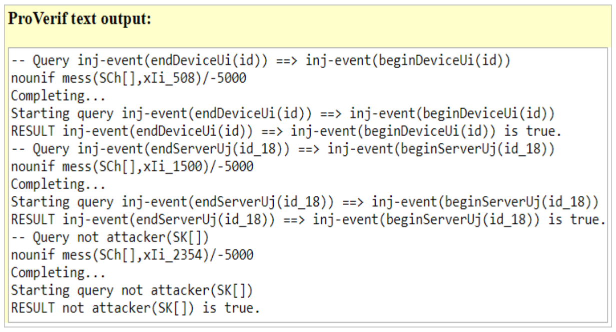
Fig. 5. Declaration of parameters with event and query execution



```
(****************Processs*************************)
(****************DeviceUi************************)
let DeviceUi=
(****************Registraion*********************)
out(SCh, (H3(CONCAT(IDi, Pswi))));
in(SCh, (CIDi:bitstring, Ckk:bitstring));
(****************Login and Authentication*********)
event beginDeviceUi(IDi);
new xi:bitstring;
let Xi = ECPM(xi,P) in
let Yi = H1(xi,Ckk) in
out(PCh, (CIDi, Xi, Yi));
(*****in(PCh, (xXj:bitstring, xYj:bitstring, xKjp:bitstring,
xIDj:bitstring,   xtj:bitstring)); *******)
in(PCh, (xXj:bitstring, xYj:bitstring, xT:bitstring));
let A = XOR(xT,XOR(H3(CONCAT(IDi, Pswi)), Ckk)) in
let YjP'= H1(xXj, A) in
if (xYj = YjP') then
let C= H3(CONCAT(Yi, A)) in
out(PCh, (C));
let (=SK)= MULT(MULT(xi,xXj), MULT(Pswi,PKs)) in
event endDeviceUi(IDi)
else 0.
(****************Server TS************************)
let ServerUj=
(****************Login and Authentication*********)
event beginServerUj(IDs);
in(PCh, (xXi:bitstring, xYi:bitstring, xCIDi:bitstring));
(**in(PCh, (xXi:bitstring, xYi:bitstring, xCIDi:bitstring,
xxIDi:bitstring,   xti:bitstring));***)
new t:bitstring;
new a:bitstring;
new et:bitstring;
let T = XOR(t,XOR(xCIDi,XOR(IDs, s))) in
let A = XOR(a,XOR(xCIDi,XOR(IDs, s))) in
let Et= XOR(et,XOR(xCIDi,XOR(IDs, s))) in
let R = XOR(T, XOR(xCIDi,s)) in
let CK= H3(CONCAT(CONCAT(R,s), CONCAT(Et,xCIDi))) in
let YiP = H3(ECPM(xXi,CK)) in
if (xYi = YiP) then
new xj:bitstring;
let Xj = ECPM(xj,P) in
let Yj = H3(ECPM(xj,ECPM(A,P))) in
out(PCh, (Xj, Yj, T));
in(PCh,(xC:bitstring));
let C'= H3(CONCAT(xYi,A)) in
if (C' = xC) then
let (=SK)= MULT(MULT(xj,xXi), MULT(s,PKe)) in
event endServerUj(IDs)
else 0.
(******************TTP*************************)
let TTP=
(**************Registration*******************)
in(SCh, xIi:bitstring);
new Ri:bitstring;
new Et:bitstring;
let CIDi = XOR(H3(CONCAT(Ri,CONCAT(xIi,s))), s) in
let CKi = H3(CONCAT(CONCAT(Ri,s), CONCAT(Et,CIDi))) in
let CKi' = ECPM(CKi, P) in
let T = XOR(Ri, H3(CONCAT(Ri,CONCAT(xIi,s)))) in
let A= XOR(T, XOR(xIi, CKi')) in
let t = XOR(T, XOR(CIDi, XOR(IDs, s))) in
let a = XOR(A, XOR(CIDi, XOR(IDs, s)))  in
let et = XOR(Et, XOR(CIDi, XOR(IDs, s))) in
out(SCh, (CIDi, CKi'));
0.
process ((!DeviceUi) | (!TTP) | (!ServerUj))
```

Fig. 6. Process specification for embedded device and server

proposed scheme in Fig. 7 is made in ProVerif. The model comprises three fragments, namely, parameter declaration, principal process, and query execution (See Figure. 5). The parameter-declaration fragment depicts variables, names, and channels other than cryptographic capacities. The meanings of the principal process and sub-process are explained in the process segment, although the outline of the assessing scheme is set up in the query-execution segment. A couple of channels, constants, variables, and factors other than cryptographic capacities outlined as constructors and conditions are displayed in the parameter-declaration fragment. Furthermore, the principle-process fragment characterizes the commencement and end of the participating clients. The procedures of the execution of the participating clients are kept parallel. Toward the end of the execution, three queries are executed to amend the



ProVerif text output:

```
-- Query inj-event(endDeviceUi(id)) ==> inj-event(beginDeviceUi(id))
nounif mess(SCh[],xIi_508)/-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(endServerUj(id_18)) ==> inj-event(beginServerUj(id_18))
nounif mess(SCh[],xIi_1500)/-5000
Completing...
Starting query inj-event(endServerUj(id_18)) ==> inj-event(beginServerUj(id_18))
RESULT inj-event(endServerUj(id_18)) ==> inj-event(beginServerUj(id_18)) is true.
-- Query not attacker(SK[])
nounif mess(SCh[],xIi_2354)/-5000
Completing...
Starting query not attacker(SK[])
RESULT not attacker(SK[]) is true.
```

Fig. 7. Output of query execution

correctness and secrecy of the proposed scheme. The consequences of the queries are shown in Fig. 7. The rightness of the proposed scheme is substantiated because the initial two queries are executed successfully, which indicate the successful beginning of initial interaction between devices and server, although its secrecy is affirmed because of an unsuccessful query (i.e., last query in Queries section in Figure 7) attack on the session key.

REFERENCES:

[1] 36Avispa-project.org. (2020). *AVISPA Web Tool*. [online] Available at: http://www.avispa-project.org/web-interface/basic.php [Accessed 16 Mar. 2020].

[2] 13 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] 25Wazid, M., Das, A.K., Bhat, V. and Vasilakos, A.V., 2020. LAMCIoT: Lightweight authentication mechanism in a cloud-based IoT environment. Journal of Network and Computer Applications, 150, p.102496.

[4] 14 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/.