A
Project Report
On

Four Factor Mutual Authentication in WBAN

Submitted to

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
RK VALLEY

in partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN COMPUTER SCIENCE ENGINEERING

Under the guidance of

Mr. T SANDEEP KUMAR REDDY
Assistant Professor

Submitted By

J.Raja Vinay (R180821)	**C. Navya Sree (S180735)**	**N. Guna Sekhar (R180559)**	**D. Kavitha Bai (R180558)**
S.Sai Teja (S180663)	**S. Naga Malleswaramma (R180826)**	**V. Mounika (R180264)**	**N. Mamatha (S180901)**
J. Sandhya Rani (R180779)	**N. Mahendra (R170335)**	**P. Mohan raju (R180051)**	**Ch.Joy Joshua Paul(R180244)**
P. Jahnavi (R180608)	**P. Dinesh Kumar (R180535)**	**C.SuryaKiran (R181007)**	**A.Mithra (R180582)**
M. Indu (R180271)	**K. Hema latha (R170908)**	**V. Lakshmi Anjali (S180937)**	
J. Krishna Charan (R180399)	**T. Sushma (R180194)**	**T. Lakshmi (R181036)**	

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE

TECHNOLOGIES, RK VALLEY- 516330

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the project report entitled "**Four Factor Mutual Authentication in WBAN**" a bonafide record of the project work done and submitted by J Raja Vinay (R180821), S Sai Teja (S180663), J. Sandhya Rani (R180779), P Jahnavi (R180608), M Indu (R180271), J Krishna Charan (R180399), C Navya Sree (S180735), S. Naga Malleswaramma (R180826), N Mahendra (R170335), P Dinesh Kumar (R180535), K Hemalatha (R170908), T Sushma (R180194), N Guna Sekhar (R180559), V Mounika (R180264), P Mohan Raju (R180051), C SuryaKiran (R181007), V Lakshmi Anjali (S180937), T Lakshmi (R181036), D Kavitha Bai (R180558), N Mamatha (S180901), Ch Joy Joshua Paul (R180244), A Mithra (R180582) for the partial fulfilment of the requirements for the award of B.Tech. Degree in Computer Science and Engineering, Rajiv Gandhi University of Knowledge Technologies, RK Valley.

The report has been not submitted previously in part or full to this university or any other university or institution for the award of any degree or diploma.

INTERNAL GUIDE	**HEAD OF THE DEPARTMENT**
T SANDEEP KUMAR REDDY	N SATYANANDARAM
Assistant Professor, Dept of CSE,	HOD CSE
RGUKT - RK VALLEY	RGUKT - RK VALLEY

DECLARATION

We hereby declare that the project report entitled “Four Factor Mutual Authentication in WBAN” submitted to the Department of COMPUTER SCIENCE ENGINEERING in partial fulfilment of requirements for the award of the degree of BACHELOR OF TECHNOLOGY. This project is the result of our own effort and that it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and who’s constant guidance and encouragement crown all the efforts success.

I would like to express my sincere gratitude to **Mr. T. Sandeep Kumar Reddy**, my project guide for valuable suggestions and keen interest throughout the progress of the project.

I’m grateful to **Mr. N. Satyanandaram , HOD CSE** , for providing excellent computing facilities and congenial atmosphere for progressing the project.

At the outset, I would like to thank **Honourable Director Madam, Mrs. K Sandhya Rani**, for providing all the necessary resources and support for the successful completion of my course work.

TABLE OF CONTENTS

Abstract

Remote health monitoring is a big help in these situations. It uses special apps and wireless devices attached to our bodies to keep track of our health. This is a big step in healthcare. The tricky part is making sure these devices can talk to each other securely. People have come up with many ideas for this, but there are problems with each one. So, we're suggesting a new way to make sure these devices talk safely. It uses four different things to check if everything is okay. This way, we can make sure only the right people can see the data from the devices. We're not just saying this, we tested it using special tools to make sure it's safe. We found that our idea is better than most others when it comes to both safety and how well it works. We studied the Two Factor Authentication and due to security reasons we are moving Four Factor Mutual Authentication in WBAN. The scheme uses four factors (username, password, biometrics, and token) to provide strong authentication. It also uses lightweight cryptographic primitives to make it efficient for resource-constrained WBAN devices. In short, our scheme is a secure and efficient way to authenticate users and sensors in WBANs, which makes it a promising solution for remote health care monitoring. We are applying Four Factor Mutual Authentication in Real Time applications like Financial Institutions, Military Organizations and Agriculture. Finally we need to be study the implementation of Four Factor Mutual Authentication in these real time applications and analyse the security challenges.

INTRODUCTION

Health care is important because it helps people stay healthy and live longer. Wireless body area networks (WBANs) can help people monitor their health from a distance by using small, wearable sensors to collect data on a person's vital signs. This data can then be transmitted to a doctor or other healthcare provider for monitoring and diagnosis. WBANs have advantages over traditional methods of health monitoring, such as blood pressure cuffs and stethoscopes, because they are more convenient and can provide more continuous monitoring. Mutual authentication is a security process in which both parties, such as a user and a system, verify each other's identities before granting access. It ensures a two-way trust relationship, enhancing overall system security. For example, WBANs can be used to monitor a patient's heart rate and blood pressure 24/7. This type of continuous monitoring can help to identify problems earlier and improve the quality of care. WBANs have challenges, such as cost and security. WBAN devices are still relatively expensive, and it is important to ensure that the data collected by WBANs is protected from unauthorized access. However, the cost of WBAN devices is expected to decrease as the technology matures, and there are a number of security protocols that can be used to protect WBAN data. WBANs have the potential to revolutionize health care by making it easier for patients to monitor their own health and by helping to improve the quality of care for patients who are at risk for chronic diseases. For example, WBANs can be used to help patients with diabetes track their blood sugar levels and to help patients with heart disease monitor their heart rate and blood pressure. This type of remote monitoring can help patients to take better care of their health and to avoid costly and unnecessary hospital visits.

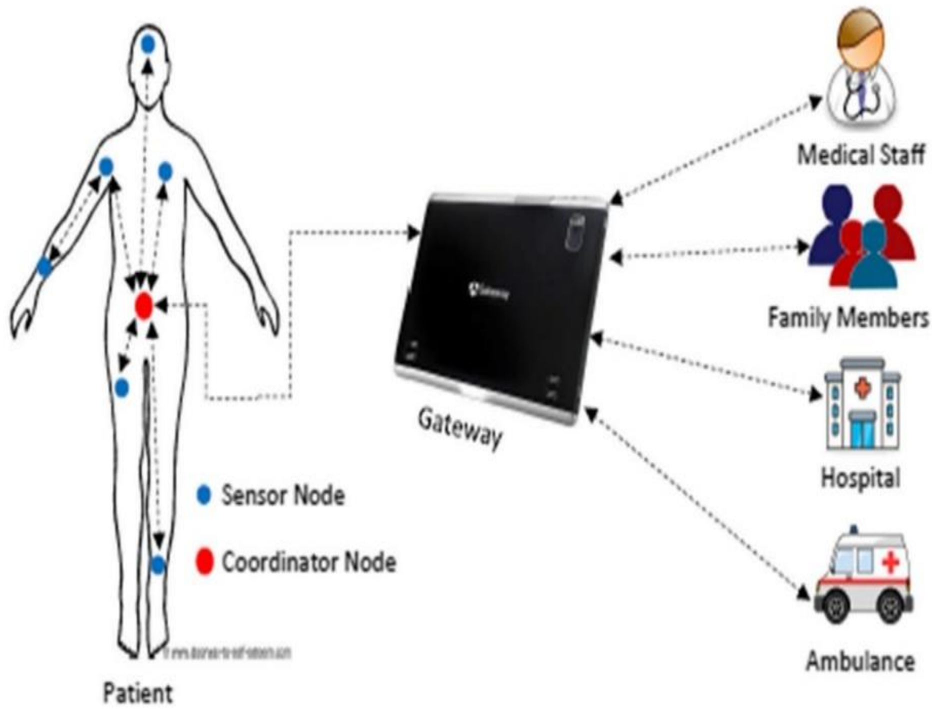


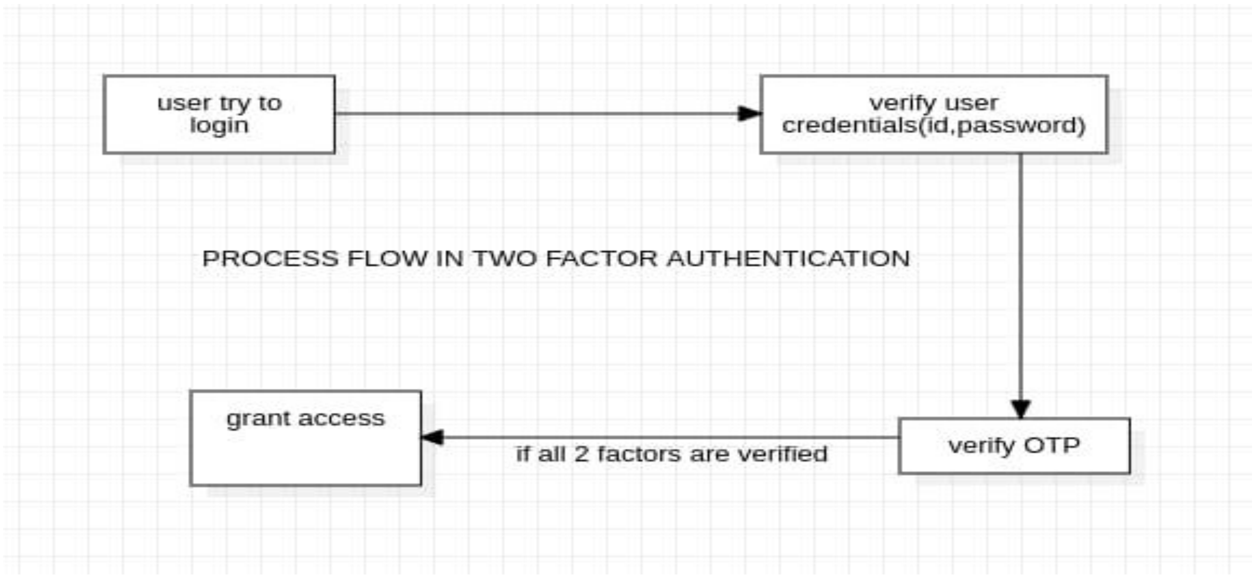
Fig. 1 Architecture of WBAN

i. **Two factor Authentication using AVISPA**

Two-factor authentication (2FA) is a security method that adds an extra layer of protection to online accounts. It requires users to provide two separate forms of verification before gaining access.This involves something the user knows (like a password) and something the user has (like a verification code sent via text message or generated by an authentication app).

Control Flow :

1. *User initiates login:*
   The user enters their username and password.
2. *System requests second factor:*
   After the initial credentials are entered, the system asks for the second factor of authentication.
3. *User provides second factor:*
   The user provides the requested second factor, which could be a code generated by an authentication app, a text message with a code, a fingerprint scan, etc.
4. *Access granted:*
   If both factors are correct, access is granted to the account or system.


user try to login
verify user credentials(id,password)
PROCESS FLOW IN TWO FACTOR AUTHENTICATION
grant access
if all 2 factors are verified
verify OTP

a. IMPLEMENTATION

```
role admin(SA, SN, HN:agent, SK:symmetric_key, SND, RCV:channel(dy), KHN, KN, IDN:text,
H:hash_func)

played
_by SA
def=

     local State:nat, AN, BN:text

     const secKHN, secIDN, secKN,
     sn_hn_beta:protocol_idinit State:=0

   transition
```

1. role admin(SA, SN, HN:agent, SK:symmetric_key, SND, RCV:channel(dy), KHN, KN, IDN:text, H:hash_func)
   - This line defines a role named "admin" that takes several parameters: SA, SN, and HN (agents), SK (a symmetric key), SND and RCV (communication channels with the data type "dy"), KHN, KN, and IDN (text values), and H (a hash function).

2. played_by SA
   - This line specifies that the "admin" role is played by the agent SA.

3. def=
   - The start of the role definition.

4. local State:nat, AN, BN:text
   - This line declares local variables for the "admin" role. The variables include "State" of type natural number, and two text variables AN and BN.

5. const secKHN, secIDN, secKN, sn_hn_beta:protocol_id
   - This line declares constant values secKHN, secIDN, secKN, and sn_hn_beta, which are identifiers for protocol entities.

6. init State:=0
- Initializes the "State" variable to 0.

7. transition
   - Specifies that the following lines describe transitions for this role.

8. 1.State=0 /\ RCV(start) =|> State':=1
- This is the first transition, which can be executed when the current "State" is 0, and the

"RCV"channel receives a message "start". The transition updates the "State" to 1.

9. Within this transition:
   - AN':=xor(IDN,H(KHN.KN)): Calculate the value of "AN'" as the result of the
\BN':=xor(xor(KHN,KN),AN')
    - Calculate the value of "BN'" as the result of the XOR operation between KHN, KN, and AN'.
XOR operation between IDN and the hash of the concatenation of KHN and KN.

10. \SND({IDN.AN'.BN'}_SK)
    - Send the message {IDN.AN'.BN'} encrypted with the symmetric key "SK" through the "SND" channel.

End role

```
role snode(SA, SN, HN:agent, SK:symmetric_key, SND, RCV:channel(dy), H:hash_func)
played_by SN
def=
      local State:nat, IDN, AN, BN, RN, TN, XN, YN, TIDN, ALPHA, BETA, GAMMA, ETA, MU,
FN, ANnew, BNnew, KN, KHN, KS:text
      const secKHN, secIDN, secKN, sn_hn_beta:protocol_id
      init State:=0
```

11. role snode(SA, SN, HN:agent, SK:symmetric_key, SND, RCV:channel(dy), H:hash_func)
   - This line defines a role named "snode" that takes several parameters: SA, SN, and HN (agents), SK (a symmetric key), SND and RCV (communication channels with the data type "dy"), and H (a hash function).

12. played_by SN
   - This line specifies that the "snode" role is played by the agent SN.

13. def=
   - The start of the role definition.

14. local State:nat, IDN, AN, BN, RN, TN, XN, YN, TIDN, ALPHA, BETA, GAMMA, ETA, MU, FN, ANnew, BNnew, KN, KHN, KS:text
   - This line declares local variables for the "snode" role. The variables include "State" of type natural number, and several other variables used in the protocol.

15. const secKHN, secIDN, secKN, sn_hn_beta:protocol_id
   - This line declares constant values secKHN, secIDN, secKN, and sn_hn_beta, which are identifiers for protocol entities.

16. init State:=0
- Initializes the "State" variable to 0.

17. transition
   - Specifies that the following lines describe transitions for this role.

19. 1.State=0 /\ RCV({IDN'.AN'.BN'}_SK) =|> State':=1
   - This is the first transition, which can be executed when the current "State" is 0, and the "RCV" channel receives a message {IDN'.AN'.BN'} encrypted with the symmetric key "SK". The transition updates the "State" to 1.

20. Within this transition:
   - RN':=new() and TN':=new()
     - Generate fresh values for the variables RN' and TN'.

21. \XN':= xor(AN',IDN')
    - Calculate the value of "XN'" as the result of the XOR operation between AN' and IDN'.

22. \YN':=xor(XN',RN')
    - Calculate the value of "YN'" as the result of the XOR operation between XN' and RN'.

23. \TIDN':=H(xor(IDN',TN').RN')
    - Calculate the value of "TIDN'" as the hash of the concatenation of the XOR result between IDN', TN', and RN'.

24. \SND(TIDN'.YN'.AN'.BN'.TN')
    - Send the message {TIDN'.YN'.AN'.BN'.TN'} through the "SND" channel.

```
      2.State=1 /\ RCV(ALPHA'.BETA'.ETA'.MU') =|> State':=2
        /\FN':=xor(XN,ALPHA')
        /\BETA':=H(XN.RN.FN'.ETA'.MU') /\ request(SN,HN,sn_hn_beta,BETA')
        /\GAMMA':=xor(RN,FN') /\ ANnew':=xor(GAMMA',ETA')
        /\BNnew':=xor(GAMMA',MU') /\ KS':=H(IDN.RN.FN'.XN)
        /\AN':=ANnew' /\ BN':=BNnew' /\ secret(KN,secKN,{SA,HN}) /\ secret(KHN,secKHN,
{SA,HN}) /\ secret(IDN,secIDN,{SA,SN,HN})
end role
```

25. 2.State=1 /\ RCV(ALPHA'.BETA'.ETA'.MU') =|> State':=2
    - This is the second transition, which can be executed when the current "State" is 1, and the "RCV" channel receives a message (ALPHA'.BETA'.ETA'.MU'). The transition updates the "State" to 2.

26. Within this transition:
    - \FN':=xor(XN,ALPHA')
      - Calculate the value of "FN'" as the result of the XOR operation between XN and ALPHA'.

    - \BETA':=H(XN.RN.FN'.ETA'.MU') /\ request(SN,HN,sn_hn_beta,BETA')

- Calculate the value of "BETA'" as the hash of the concatenation of XN, RN, FN', ETA', and MU'. Then, send a "request" message to HN for authentication proof, with parameters SN, HN, sn_hn_beta, and BETA'.

- \GAMMA':=xor(RN,FN')
  - Calculate the value of "GAMMA'" as the result of the XOR operation between RN and FN'.

- \ANnew':=xor(GAMMA',ETA') and \BNnew':=xor(GAMMA',MU')
  - Calculate the values of "ANnew'" and "BNnew'" as the results of XOR operations involving GAMMA', ETA', and GAMMA', MU', respectively.

- \KS':=H(IDN.RN.FN'.XN)
  - Calculate the value of "KS'" as the hash of the concatenation of IDN, RN, FN', and XN.

- \AN':=ANnew' and \BN':=BNnew'
  - Update AN' with the value of ANnew' and BN' with the value of BNnew'.

- \secret(KN,secKN,{SA,HN}), \secret(KHN,secKHN,{SA,HN}), and \secret(IDN,secIDN, {SA,SN,HN})
  - Set KN, KHN, and IDN as secrets with identifiers secKN, secKHN, and secIDN, respectively, shared between SA and HN, SA and HN, and SA, SN, and HN.

end role

```
role hnode(SA, SN, HN:agent, SK:symmetric_key, SND, RCV:channel(dy), H:hash_func)
played_by HN
def=
      local State:nat, IDN, TIDN, YN, AN, BN, TN, KN, KHN, XN, RN, FN, ALPHA, BETA,
GAMMA, ETA, MU, TKN, KS:text
      const secKHN, secKN, secIDN, sn_hn_beta:protocol_id
      init State:=0
```

27. role hnode(SA, SN, HN:agent, SK:symmetric_key, SND, RCV:channel(dy), H:hash_func)
   - This line defines a role named "hnode" that takes several parameters: SA, SN, and HN (agents), SK (a symmetric key), SND and RCV (communication channels with the data type "dy"), and H (a hash function).

28. played_by HN
   - This line specifies that the "hnode" role is played by the agent HN.

29. def=
   - The start of the role definition.

30. local State:nat, IDN, TIDN, YN, AN, BN, TN, KN, KHN, XN, RN, FN, ALPHA, BETA,GAMMA, ETA, MU, TKN, KS:text
   - This line declares local variables for the "hnode" role. The variables include "State" of type natural number, and several other variables used in the protocol.

31. const secKHN, secKN, secIDN, sn_hn_beta:protocol_id

- This line declares constant values secKHN, secKN, secIDN, and sn_hn_beta, which are identifiers for protocol entities.

32. init State:=0
   - Initializes the "State" variable to 0.

```
    transition
    1.State=0 /\ RCV(TIDN'.YN'.AN'.BN'.TN') =|> State':=1
      /\KN':=xor(xor(KHN,BN),AN)
      /\XN':=H(KHN.KN')
      /\IDN':=xor(AN',XN') /\ RN':=xor(YN',XN')
      /\TIDN':=H(xor(IDN',TN').RN')
      /\FN':=new() /\ ALPHA':=xor(XN',FN') /\ GAMMA':=xor(RN',FN')
      /\TKN':=new() /\ AN':=xor(IDN',H(KHN.TKN))
      /\BN':=xor(xor(KHN,TKN),AN') /\ ETA':=xor(GAMMA',AN')
      /\MU':=xor(GAMMA',BN') /\ BETA':=H(XN'.RN'.FN'.ETA'.MU')
      /\witness(HN,SN,sn_hn_beta,BETA') /\ KS':=H(IDN'.RN'.FN'.XN')
      /\SND(ALPHA'.BETA'.ETA'.MU') /\ secret(KN,secKN,{SA,HN}) /\ secret(KHN,secKHN,
{SA,HN}) /\ secret(IDN,secIDN,{SA,SN,HN})
```

33. transition
   - Specifies that the following lines describe transitions for this role.

34. 1.State=0 /\ RCV(TIDN'.YN'.AN'.BN'.TN') =|> State':=1
   - This is the first transition, which can be executed when the current "State" is 0, and the "RCV" channel receives a message (TIDN'.YN'.AN'.BN'.TN'). The transition updates the "State" to 1.

35. Within this transition:
   - \KN':=xor(xor(KHN,BN),AN)
     - Calculate the value of "KN'" as the result of the XOR operation between KHN, BN, and AN.

   - \XN':=H(KHN.KN')
     - Calculate the value of "XN'" as the hash of the concatenation of KHN and KN'.

   - \IDN':=xor(AN',XN') /\ RN':=xor(YN',XN')
     - Calculate the values of "IDN'" and "RN'" as the results of XOR operations involving AN', XN', and YN', XN', respectively.

   - \TIDN':=H(xor(IDN',TN').RN')
     - Calculate the value of "TIDN'" as the hash of the concatenation of the XOR result betweenIDN', TN', and RN'.

   - \FN':=new() /\ ALPHA':=xor(XN',FN') /\ GAMMA':=xor(RN',FN')
     - Generate a fresh value for FN', and then calculate the values of ALPHA' and GAMMA' as the results of XOR operations involving XN' and FN', and RN' and FN', respectively.

   - \TKN':=new() /\ AN':=xor(IDN',H(KHN.TKN))
     - Generate a fresh value for TKN', and then calculate the value of AN' as the result of the XOR operation between IDN' and the hash of the concatenation of KHN and TKN'.

- \BN':=xor(xor(KHN,TKN),AN') /\ ETA':=xor(GAMMA',AN')
  - Calculate the values of BN' and ETA' as the results of XOR operations involving KHN, TKN,and AN', and GAMMA' and AN', respectively.

- \MU':=xor(GAMMA',BN') /\ BETA':=H(XN'.RN'.FN'.ETA'.MU')
  - Calculate the values of MU' and BETA' as the results of XOR operations involving GAMMA' and BN', and the hash of the concatenation of XN', RN', FN', ETA', and MU', respectively.

- \witness(HN,SN,sn_hn_beta,BETA')
  - Send a "witness" message to SN with the parameters HN, SN, sn_hn_beta, and BETA'.

- \KS':=H(IDN'.RN'.FN'.XN')
  - Calculate the value of KS' as the hash of the concatenation of IDN', RN', FN', and XN'.

- \SND(ALPHA'.BETA'.ETA'.MU')
  - Send the message (ALPHA'.BETA'.ETA'.MU') through the "SND" channel.

- \secret(KN,secKN,{SA,HN}), \secret(KHN,secKHN,{SA,HN}), and \secret(IDN,secIDN, {SA,SN,HN

})
  - Set KN, KHN, and IDN as secrets with identifiers secKN, secKHN, and secIDN, respectively, shared between SA and HN, SA and HN, and SA, SN, and HN.

End role

```
role session(SA,SN,HN:agent,SK:symmetric_key,KHN,KN,IDN:text,H:hash_func)
def=
    local
        SND3,RCV3,SND2,RCV2,SND1,RCV1:channel(dy)
    composition
        admin(SA,SN,HN,SK,SND3,RCV3,KHN,KN,IDN,H) /\
hnode(SA,SN,HN,SK,SND2,RCV2,H) /\ snode(SA,SN,HN,SK,SND1,RCV1,H)
end role
```

36. role session(SA, SN, HN:agent, SK:symmetric_key, KHN, KN, IDN:text, H:hash_func) def= ...end role
   - This line defines a new role named session.
   - The role has several parameters: SA, SN, and HN (agents), SK (a symmetric key), KHN and KN (protocol identifiers), IDN (a text parameter), and H (a hash function).

37. local SND3, RCV3, SND2, RCV2, SND1, RCV1:channel(dy)
   - This line declares six local variables: SND1, RCV1, SND2, RCV2, SND3, and RCV3.
   - Each of these variables is of type channel(dy), indicating that they are communication channelsfor exchanging dynamic (encrypted) messages.

38. composition
   - This keyword indicates that the following roles (admin, snode, and hnode) are combined to form

39. admin(SA, SN, HN, SK, SND3, RCV3, KHN, KN, IDN, H)
 - This line creates an instance of the role admin with the specified parameters.
 - The instance represents the role admin played by agent SA in the session.

40. hnode(SA, SN, HN, SK, SND2, RCV2, H)
 - This line creates an instance of the role hnode with the specified parameters.
 - The instance represents the role hnode played by agent HN in the session.

41. snode(SA, SN, HN, SK, SND1, RCV1, H)
 - This line creates an instance of the role snode with the specified parameters.
- The instance represents the role snode played by agent SN in the session.

```
role environment()
def=
    const sa,sn,hn:agent,
        khn,kn,idn:text,
        sk:symmetric_key,
        h:hash_func,
        seckn,seckhn,secidn,sn_hn_beta:protocol_id
    intruder_knowledge={sa,sn,hn}
    composition
        session(sa,sn,hn,sk,khn,kn,idn,h)
end role
goal
   secrecy_of secKHN
   secrecy_of secKN
   secrecy_of secIDN
   authentication_on sn_hn_beta
end goal
environment()
```

42. role environment() def= ... end role
 - This line defines a new role named environment.

43. const sa, sn, hn:agent, khn, kn, idn:text, sk:symmetric_key, h:hash_func, seckn, seckhn, secidn,sn_hn_beta:protocol_id
 - This line declares several constants:
  - sa, sn, hn: agents (representing the identities of the three entities in the protocol).
  - khn, kn, idn: text parameters.
  - sk: a symmetric key parameter.
  - h: a hash function parameter.
  - seckn, seckhn, secidn, sn_hn_beta: protocol identifiers.

44. intruder_knowledge={sa, sn, hn}
 - This line specifies the knowledge of the intruder (environment).
 - It indicates that the intruder knows the existence of agents sa, sn, and hn.

45. composition session(sa, sn, hn, sk, khn, kn, idn, h)
 - This line creates an instance of the role session with the specified parameters.

46. goal
    - This keyword indicates that the following lines define the security goals to be achieved.

47. secrecy_of secKHN
    - This goal specifies that the identifier secKHN must remain secret.
    - It means that unauthorized entities should not learn the value of secKHN.

48. secrecy_of secKN
    This goal specifies that the identifier secKN must remain secret.

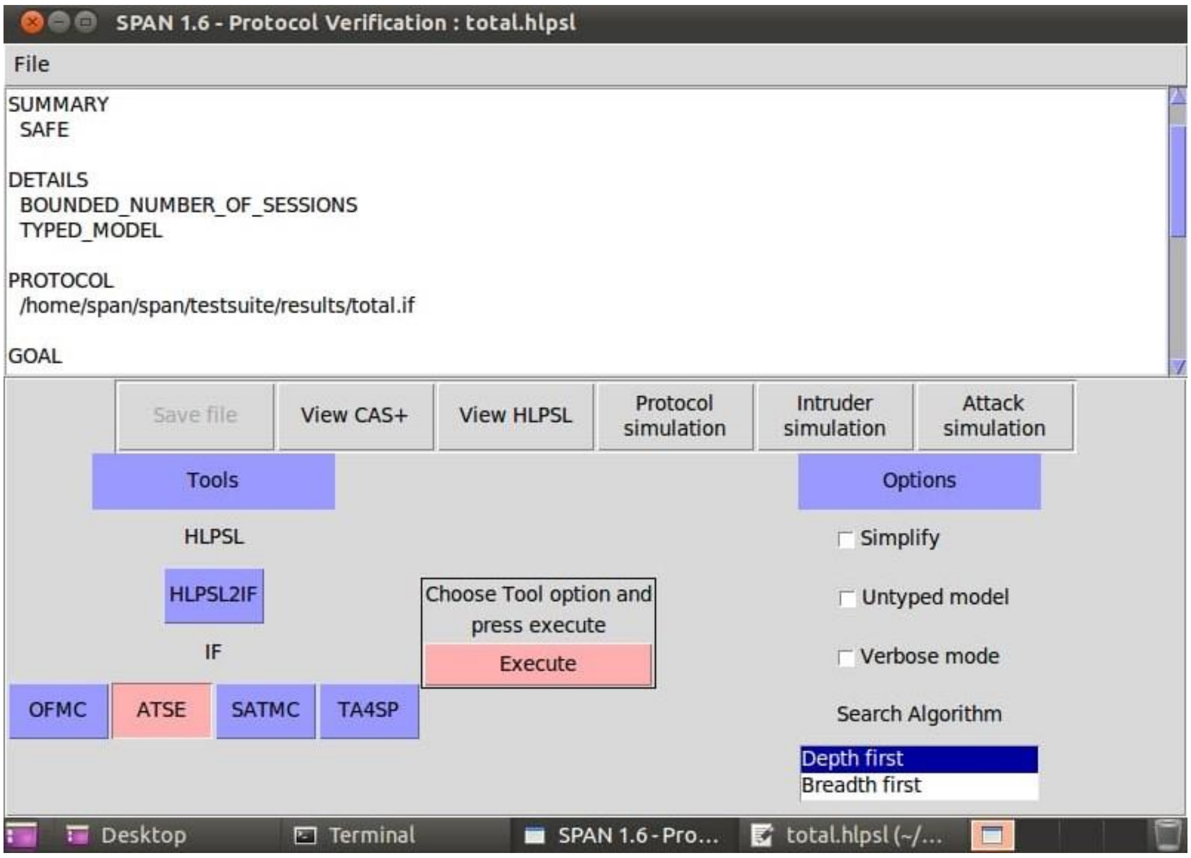
    - It means that unauthorized entities should not learn the value of secKN.

49. secrecy_of secIDN
    - This goal specifies that the identifier secIDN must remain secret.
    - It means that unauthorized entities should not learn the value of secIDN.

50. authentication_on sn_hn_beta
    - This goal specifies that the communication channel sn_hn_beta should be authenticated.
    - It means that only valid agents (entities) should be able to communicate over this channel.

51. environment()
    - This line creates an instance of the role environment.
    - The instance represents the intruder (environment) in the protocol.

b. RESULT:



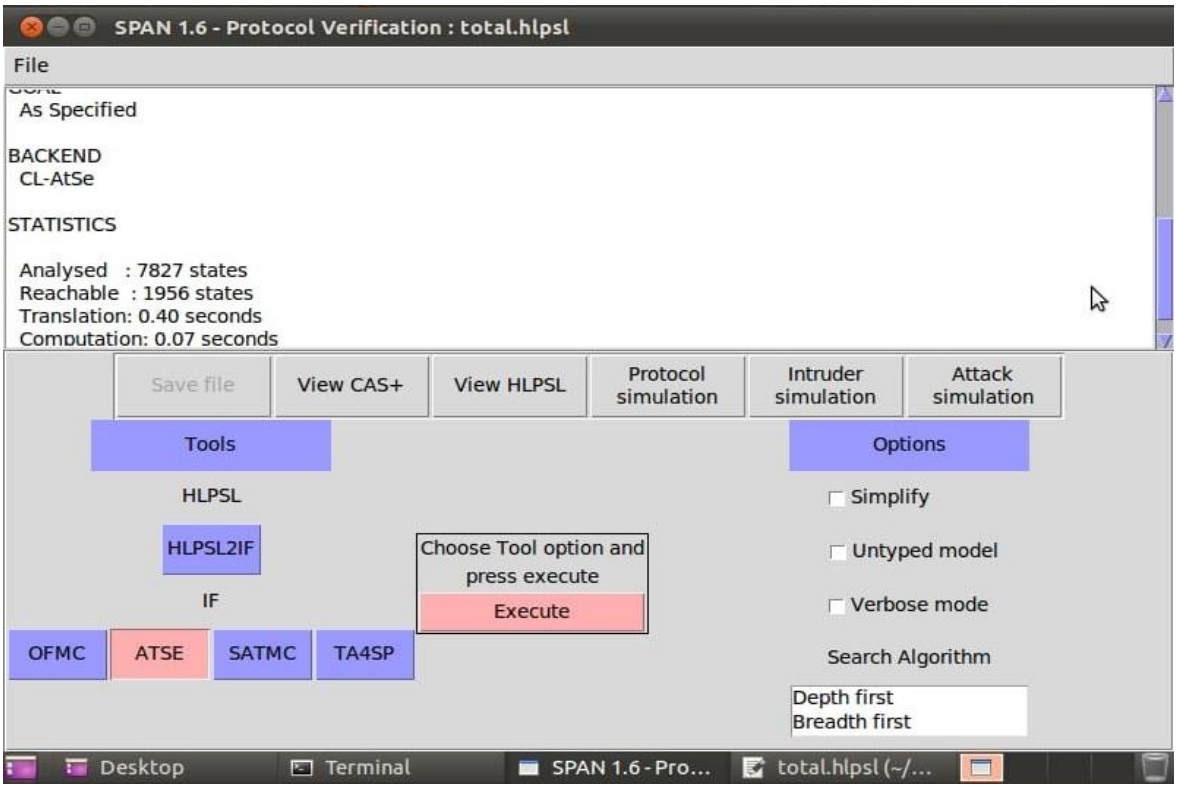
```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

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

GOAL
```

Options: ☐ Simplify, ☐ Untyped model, ☐ Verbose mode; Search Algorithm: [✓ Depth first] Breadth first



```
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 7827 states
  Reachable  : 1956 states
  Translation: 0.40 seconds
  Computation: 0.07 seconds
```

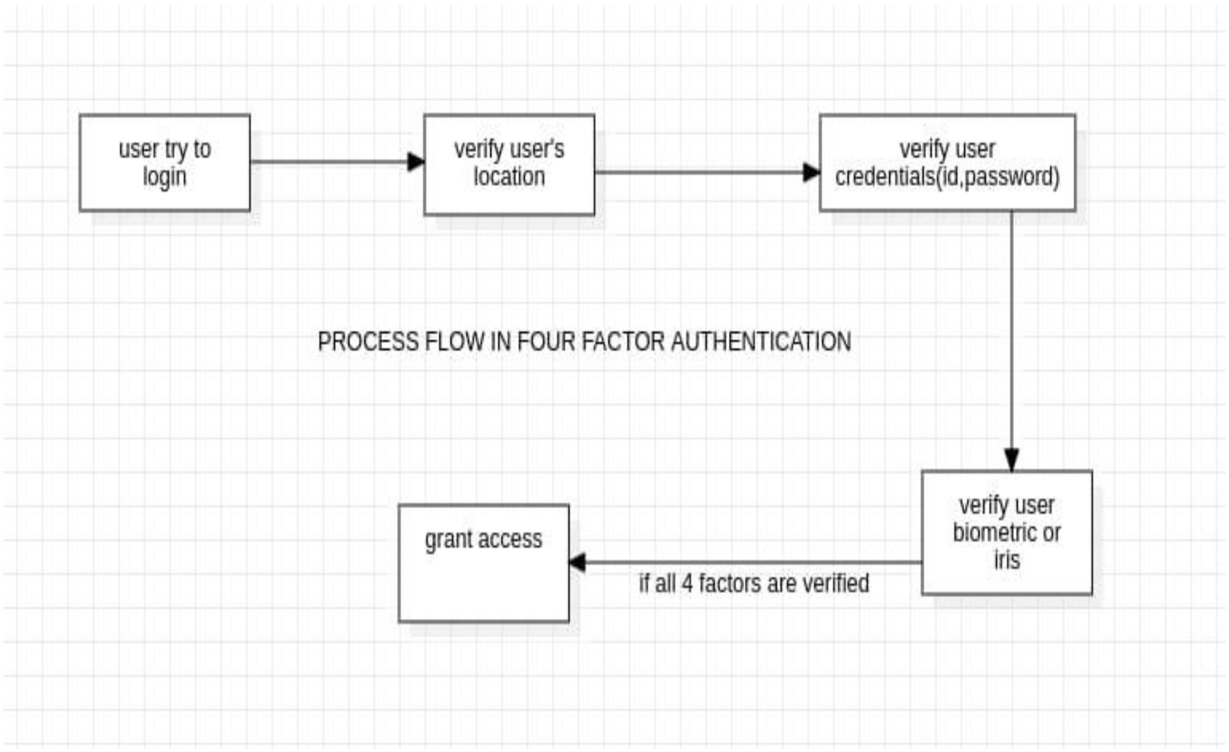
IFig- ATSE RESULT

ii. Four factor authentication (4FA) using AVISPA

- Four-factor authentication (4FA) is an advanced security measure that enhances digital security by requiring the user to provide four distinct types of credentials to access a system or account.
- "Four-factor authentication" is an advanced security concept that combines four different types of authentication factors to ensure a higher level of security for user access. These factors typically include:

  1. Knowledge factor: This involves something only the user knows, like a password or a PIN.
  2. Possession factor: This involves something only the user possesses, like a smartphone, a hardware token, or an ID card.
  3. Inherence factor: This relates to something inherent to the user,such as biometric data like fingerprints,facial recognition, or voice recognition.
  4. Location factor: This factor considers the user's geographical location or the network they are connecting from.

- Avispa (Automated Validation of Internet Security Protocols and Applications) is a framework and toolset for the automated validation of internet security-sensitive protocols and applications.
- It helps in analysing the security of protocols by simulating potential attacks and vulnerabilities in protocols like authentication, key exchange, and access control.
- Four-factor authentication (4FA) is a more advanced approach to security than two-factor authentication (2FA), involving additional layers of verification.


user try to login
verify user's location
verify user credentials(id,password)
PROCESS FLOW IN FOUR FACTOR AUTHENTICATION
verify user biometric or iris
grant access
if all 4 factors are verified

GENERAL PROCESS CONTROL FLOW:

1. Define the four factors:
   Clearly define the four factors you intend to use for authentication (knowledge, possession, inherence, location).
2. Protocol Specification
3. Security Properties
4. Configuration
5. Modeling the Factors
6. Modeling the Communication
7. Analysis
8. Results and Refinement

A HLPSL file starts executing the code from environment() function always. It is similar to the main function of C programming.

When environment function is called the program control jumps to function definition and starts executing the function.

If there are any function calls inside the function the program jumps to that function by pausing the currently running function.

Then after completely executing that function the program control comes back to the function which is paused previously and completes the execution.

After all the roles are executed the program control executes goal block where it checks whether all the goals of the program are satisfied.

The control flow emerges from the interactions between the roles and their corresponding sessions during the execution of the protocol.

Four-factor authentication is a robust approach to enhance network and IoT security by adding an additional layer of verification beyond the traditional three-factor authentication (something you know, something you have, something you are). The fourth factor typically involves the user's location or behavior, making it even harder for unauthorized users to gain access. Here's an outline of an algorithm for four-factor authentication in network IoT security:

Four-factor authentication in network IoT security typically involves using four different types of credentials or factors to verify a user's identity before granting access to the IoT network. The four factors are typically classified as follows:

Something you know: This factor involves knowledge-based credentials that only the user should know. Examples include passwords, PINs, or answers to security questions.

Something you have: This factor involves possession-based credentials, usually a physical device that only the user possesses. Examples include smart cards, mobile phones, or hardware tokens.

Something you are: This factor involves biometric-based credentials, which are unique physical or behavioral traits of the user. Examples include fingerprint scans, iris scans, or voice recognition.

Somewhere you are: This factor involves location-based credentials, which verify the user's presence in a specific geographic location or within the proximity of a particular device or access point.

Designing an algorithm for four-factor authentication in network IoT security can be complex, and it needs to be tailored to the specific requirements of the IoT system. Below is a high-level outline of how the process might work:

Step 1: User Identification
The user initiates a connection to the IoT network and provides a username or identifier.
The IoT system verifies the username against a database to identify the user.

Step 2: First Factor - Something You KnowThe system prompts the user to enter a password or PIN associated with their account.
The entered password or PIN is compared to the stored password or PIN in the database.
Step 3: Second Factor - Something You Have
The system requests the user to present a physical device, such as a mobile phone or smart card, which contains a secure authentication token.
The IoT system validates the token's authenticity and verifies its association with the user's account.
Step 4: Third Factor - Something You Are
The system prompts the user to provide a biometric sample, such as a fingerprint, iris scan, or voice sample.
The provided biometric data is compared to the stored biometric data associated with the user's account.
Step 5: Fourth Factor - Somewhere You Are
The IoT system determines the user's location or proximity using locationbased technologies like GPS, Wi-Fi positioning, or Bluetooth beacons.
The system checks whether the user's location aligns with the expected or authorized location for authentication.
Step 6: Authentication Decision
After evaluating all four factors, the IoT system calculates an overall authentication score.
If the score meets the required threshold, the user is granted access to the IoT network; otherwise, access is denied.
It's important to note that implementing such a complex authentication process
requires careful consideration of security, usability, and scalability. Additionally, the use of biometric data raises privacy concerns, and proper encryption and protection measures should be applied to sensitive user information. Moreover,
IoT devices should also have appropriate security measures in place to prevent unauthorized access to the network itself.

a. IMPLEMENTATION

ROLE USER:

```
role user(Ui,GWj,SNk:agent,
UGKeyij:symmetric_key,
SGKeykj:symmetric_key,
H:hash_func,
Snd,Rcv:channel(dy))
played_by Ui
def=
local State:nat,
UIDi,UPWi,UBIOi,URSi,UBTi,UR2i,GIDj,GPrij,GPubj,UPrii,UPubi,SPrik,SPubk,SSeck,Srk,SIDk,Auth1ij,Auth3jk,Auth5kj,Auth7ji,SKik,Gr2j,SR5k:text,
UR1i,UR3i,Uri,UR4j,UR5j,Gr1j,SR1k,SR2k,SR3j,SR4j,Auth2ij,Auth4jk,Auth6kj,Auth8ji,URandi,GRandj,SRandk,CT1i,CT2j,CT3k,CT4j,Accessi:messa
Inc:hash_func
const
user_gateway_UR3i,gateway_sensor_Auth4jk,sensor_gateway_Auth6kj,gateway_user_Auth8ji,
user_gateway_Auth2ij,gateway_user_UR5j,sensor_gateway_SR2k,gateway_sensor_SR4j,
subs1,subs2,subs3,subs4,subs5,subs6:protocol_id
init State:=0
transition
1.State=0/\Rcv(start)=|>
State':=1/\UBIOi':=new()
/\UPWi':=new()
/\URSi':=new()
/\UBTi':=xor(URSi',UPWi')
/\UR1i':=H(UIDi.UPWi'.UBIOi'.URSi')
/\Uri':=new()
/\UR2i':=H(GPubj.UGKeyij)
/\UR3i':=H(UR1i.UR2i.xor(UGKeyij,Uri))
/\Snd({UIDi.UR1i'.UR3i'.Uri'}_GPubj)
/\witness(Ui,GWj,user_gateway_UR3i,UR3i)
/\secret({UPWi',UBIOi',URSi',UBTi'},subs1,Ui)
/\secret({UGKeyij},subs2,{Ui,GWj})
2.State=1/\Rcv(UR4j'.UR5j'.Gr1j')=|>
State':=2/\UPrii':=H({UR1i.UR5j'.xor(Uri,Gr1j')}_UBTi)




/\UPubi':=H({UPrii.UR2i}_UBTi)
/\CT1i':=new()
/\URandi':=new()
/\Accessi':={UIDi.URandi'.SIDk}_GPubj
/\Auth1ij':=H(UR1i.UGKeyij.UPubi')
/\Auth2ij':=H(Auth1ij.URandi.CT1i.UR5j)
/\Snd(Accessi'.Auth2ij'.URandi.CT1i')
/\witness(Ui,GWj,user_gateway_Auth2ij,Auth2ij)
/\request(GWj,Ui,gateway_user_UR5j,UR5j)
/\secret({UPrii'},subs3,Ui)
3.State=2/\Rcv(Auth8ji'.CT4j'.SRandk'.GRandj')=|>
State':=3/\Snd({xor(xor(UR3i,URandi),SRandk')}_SPubk)
/\request(GWj,Ui,gateway_user_Auth8ji,Auth8ji)
4.State=3/\Rcv({xor(xor(SR1k,SPubk),UPubi)}_UPrii)=|>
State':=4/\SKik':=H(xor(H({xor(SRandk,UR3i)}_SR1k),H({xor(URandi,SR1k)}_UR3i)))
end role
```

The "user" role represents the behavior and actions of a user (Ui) in the network security protocol. The user is an agent interacting with the system and attempting to gain access to resources or services. The "user" role involves communication with the "gateway" and "sensor" roles to establish secure communication and authentication.

Here's a step-by-step explanation of the "user" role:

1. **Initialization (State 0):**
   - The user's initial state is set to 0, indicating the start of the authentication process.
   - The user role listens for the "start" message to begin the authentication.
2. **Knowledge Factor, Possession Factor, and Biometric Factor (State 1):**
   - Upon receiving the "start" message, the user generates and stores new values for knowledge (UPWi), possession (UBIOi), and biometric (URSi) factors.
   - The user also computes the biometric token (UBTi) by XORing the possession factor (UBIOi) with the biometric factor (URSi).
   - A unique user identifier (UR1i) is created using a hash function (H) based on the concatenation of UIDi, UPWi, UBIOi, and URSi.
   - The user generates a random value (Uri) and computes UR2i by hashing the concatenation of GPubj (gateway's public key) and UGKeyij (symmetric key shared between user and gateway).
   - UR3i is calculated by hashing the concatenation of UR1i, UR2i, and the XOR of UGKeyij and Uri.
   - The user sends the encrypted tuple {UIDi, UR1i, UR3i, Uri} to the gateway using the gateway's public key (GPubj).
3. **Gateway Interaction (State 2):**
   - In this state, the user role waits to receive a message from the gateway containing SR3j' and SR4j'.
   - The user computes the private key (UPrii) using a hash function (H) based on the concatenation of UR1i, UR5j', and the XOR of Uri and Gr1j' (random value generated by gateway).
   - UPubi is calculated by hashing the concatenation of UPrii and UR2i.
   - The user generates a new random value (CT1i') and URandi' and forms the access message {UIDi, URandi', SIDk} to be sent to the gateway using the gateway's public key (GPubj).
   - Auth1ij is computed using a hash function (H) on the concatenation

- Auth2ij is calculated by hashing the concatenation of Auth1ij, URandi, CT1i', and UR5j'.
- The user sends the tuple {Accessi', Auth2ij, URandi, CT1i'} to the gateway.

4. **Receiving Authentication (State 3):**

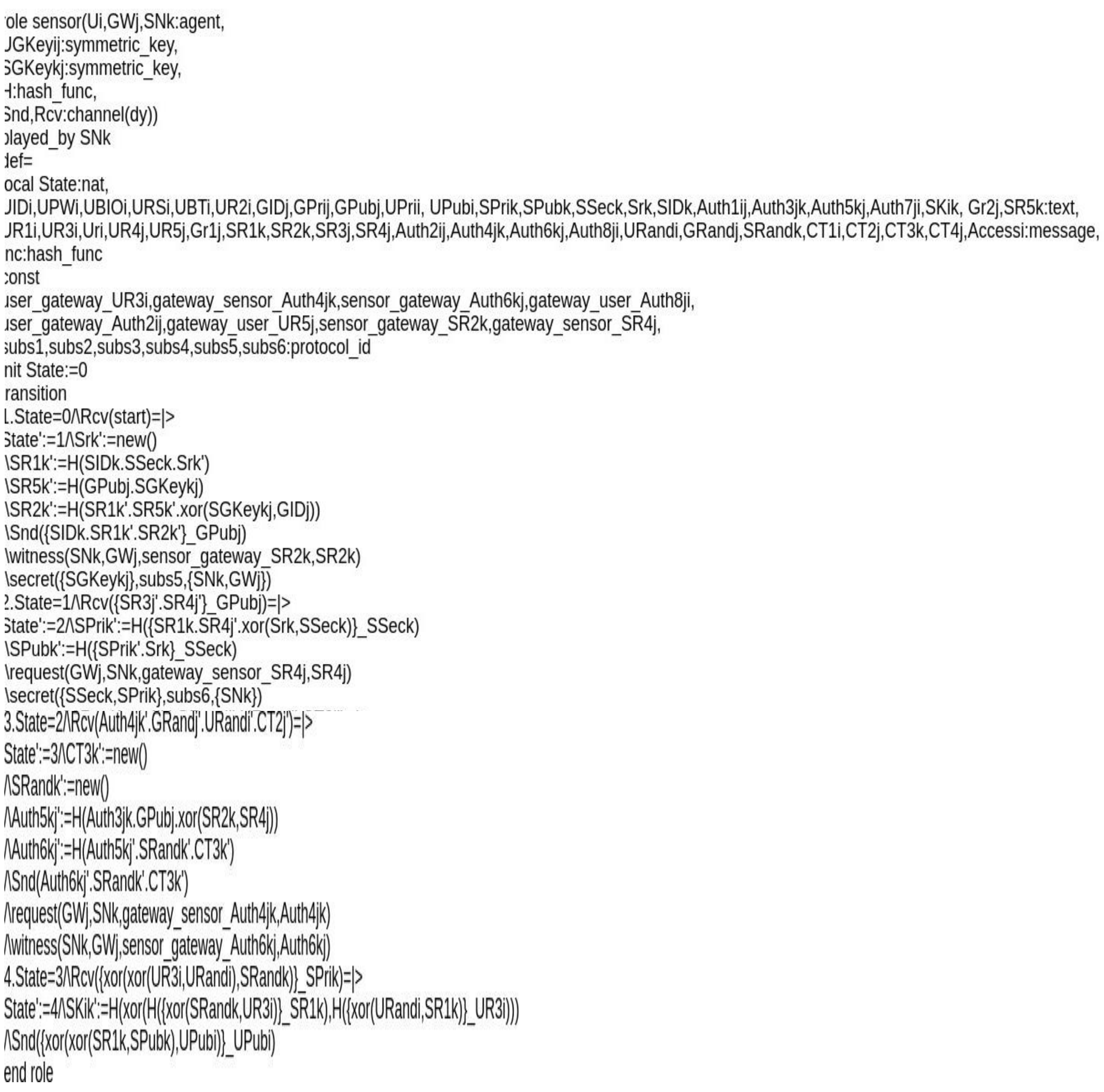
   - In this state, the user receives authentication information from the gateway, including Auth8ji', CT4j', SRandk', and GRandj'.
   - The user computes SKik' (shared session key between user and gateway) by hashing XORed values based on SRandk', UR3i, SR1k, UR3i, and SR1k.
   - The user sends the encrypted tuple {xor(xor(UR3i, URandi), SRandk')} to the sensor using the sensor's public key (SPubk).

5. **Final Authentication (State 4):**

   - In this state, the user receives final authentication information from the sensor, including {xor(xor(SR1k, SPubk), UPubi)}_UPrii.
   - The user computes the shared session key SKik' (again) by hashing XORed values based on SRandk, UR3i, SR1k, UR3i, and SR1k.
   - The user sends the encrypted tuple {xor(xor(SR1k, SPubk), UPubi)}_UPubi to the gateway.

The user role plays a significant part in the authentication process by generating and exchanging various cryptographic values with the gateway and sensor, contributing to secure communication and access to the network's resources.

SENSOR ROLE:



```
ole sensor(Ui,GWj,SNk:agent,
JGKeyij:symmetric_key,
SGKeykj:symmetric_key,
H:hash_func,
Snd,Rcv:channel(dy))
played_by SNk
def=
ocal State:nat,
JIDi,UPWi,UBIOi,URSi,UBTi,UR2i,GIDj,GPrij,GPubj,UPrii, UPubi,SPrik,SPubk,SSeck,Srk,SIDk,Auth1ij,Auth3jk,Auth5kj,Auth7ji,SKik, Gr2j,SR5k:text,
JR1i,UR3i,Uri,UR4j,UR5j,Gr1j,SR1k,SR2k,SR3j,SR4j,Auth2ij,Auth4jk,Auth6kj,Auth8ji,URandi,GRandj,SRandk,CT1i,CT2j,CT3k,CT4j,Accessi:message,
nc:hash_func
const
user_gateway_UR3i,gateway_sensor_Auth4jk,sensor_gateway_Auth6kj,gateway_user_Auth8ji,
user_gateway_Auth2ij,gateway_user_UR5j,sensor_gateway_SR2k,gateway_sensor_SR4j,
subs1,subs2,subs3,subs4,subs5,subs6:protocol_id
nit State:=0
ransition
1.State=0/\Rcv(start)=|>
State':=1/\Srk':=new()
\SR1k':=H(SIDk.SSeck.Srk')
\SR5k':=H(GPubj.SGKeykj)
\SR2k':=H(SR1k'.SR5k'.xor(SGKeykj,GIDj))
\Snd({SIDk.SR1k'.SR2k'}_GPubj)
\witness(SNk,GWj,sensor_gateway_SR2k,SR2k)
\secret({SGKeykj},subs5,{SNk,GWj})
2.State=1/\Rcv({SR3j'.SR4j'}_GPubj)=|>
State':=2/\SPrik':=H({SR1k.SR4j'.xor(Srk,SSeck)}_SSeck)
\SPubk':=H({SPrik'.Srk}_SSeck)
\request(GWj,SNk,gateway_sensor_SR4j,SR4j)
\secret({SSeck,SPrik},subs6,{SNk})
3.State=2/\Rcv(Auth4jk'.GRandj'.URandi'.CT2j')=|>
State':=3/\CT3k':=new()
/\SRandk':=new()
/\Auth5kj':=H(Auth3jk.GPubj.xor(SR2k,SR4j))
/\Auth6kj':=H(Auth5kj'.SRandk'.CT3k')
/\Snd(Auth6kj'.SRandk'.CT3k')
/\request(GWj,SNk,gateway_sensor_Auth4jk,Auth4jk)
/\witness(SNk,GWj,sensor_gateway_Auth6kj,Auth6kj)
4.State=3/\Rcv({xor(xor(UR3i,URandi),SRandk)}_SPrik)=|>
State':=4/\SKik':=H(xor(H({xor(SRandk,UR3i)}_SR1k),H({xor(URandi,SR1k)}_UR3i)))
/\Snd({xor(xor(SR1k,SPubk),UPubi)}_UPubi)
end role
```

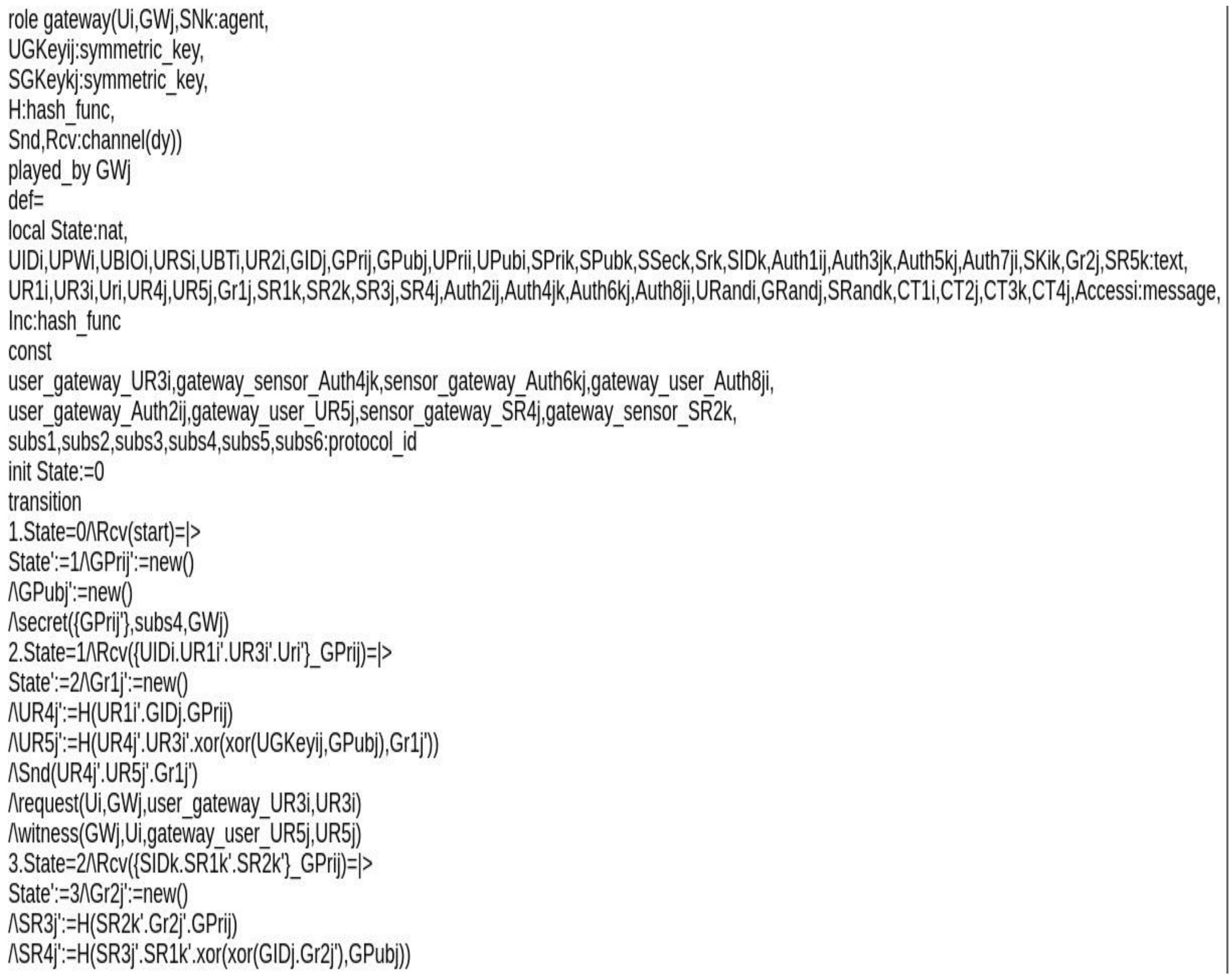
The "sensor" role represents the behavior and actions of a sensor (SNk) in the network security protocol. The sensor is an agent responsible for sensing data and communicating with the user (Ui) and gateway (GWj) to establish secure communication and authentication.

Here's a step-by-step explanation of the "sensor" role:

1. **Initialization (State 0):**
    - The sensor's initial state is set to 0, indicating the start of the authentication process.
    - The sensor role listens for the "start" message to begin the authentication.
2. **Biometric Factor and Possession Factor (State 1):**
    - Upon receiving the "start" message, the sensor generates and stores new values for the biometric (SRk) and possession (SSeck) factors.
    - The sensor computes SR1k by hashing the concatenation of SIDk (sensor's unique identifier), SSeck, and Srk.
    - SR5k is calculated by hashing the concatenation of GPubj (gateway's public key) and SGKeykj (symmetric key shared between sensor and gateway).
    - SR2k is computed by hashing the concatenation of SR1k', SR5k', and the XOR of SGKeykj and GIDj (gateway's ID).
    - The sensor sends the encrypted tuple {SIDk, SR1k', SR2k'} to the gateway using the gateway's public key (GPubj).
3. **Gateway Interaction (State 2):**
    - In this state, the sensor role waits to receive a message from the gateway containing SR3j' and SR4j'.
    - The sensor computes SPrik' (private key shared between sensor and gateway) by hashing the concatenation of SR1k', SR4j', and the XOR of Srk and SSeck.
    - SPubk' is calculated by hashing the concatenation of SPrik' and Srk.
    - The sensor generates a new random value (CT3k') and SRandk' and forms the authentication message Auth5kj' to be sent to the gateway.
    - Auth6kj is computed using a hash function (H) on the concatenation of Auth5kj', SRandk', CT3k', and SR4j'.
    - The sensor sends the tuple {Auth6kj', SRandk', CT3k'} to the gateway.

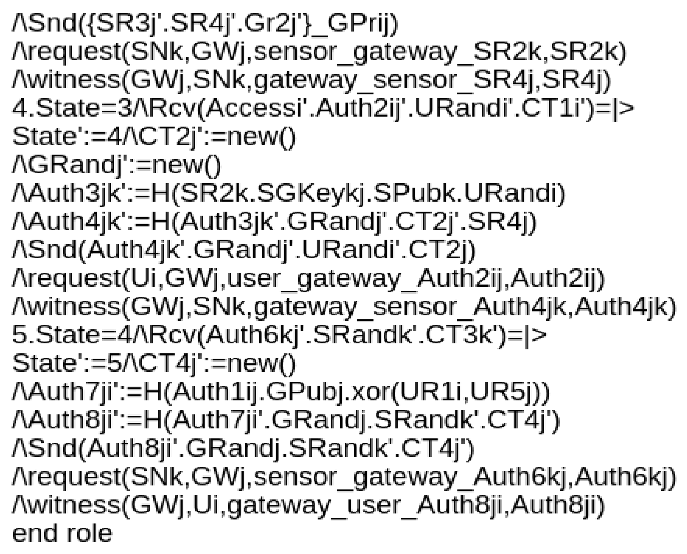
4. **Receiving Authentication (State 3):**
    - In this state, the sensor receives authentication information from the gateway, including {xor(xor(UR3i, URandi), SRandk')}_SPrik.
    - The sensor computes SKik' (shared session key between sensor and gateway) by hashing XORed values based on UR3i, SR1k, and SR4j.
    - The sensor sends the encrypted tuple {xor(xor(SR1k, SPubk'), UPubi)}_UPrii to the gateway.

The sensor role plays a significant part in the authentication process by generating and exchanging various cryptographic values with the gateway and user, contributing to secure communication and access to the network's resources. The sensor's main responsibilities involve generating and verifying biometric and possession factors and exchanging authentication messages with the gateway and user



```
role gateway(Ui,GWj,SNk:agent,
UGKeyij:symmetric_key,
SGKeykj:symmetric_key,
H:hash_func,
Snd,Rcv:channel(dy))
played_by GWj
def=
local State:nat,
UIDi,UPWi,UBIOi,URSi,UBTi,UR2i,GIDj,GPrij,GPubj,UPrii,UPubi,SPrik,SPubk,SSeck,Srk,SIDk,Auth1ij,Auth3jk,Auth5kj,Auth7ji,SKik,Gr2j,SR5k:text,
UR1i,UR3i,Uri,UR4j,UR5j,Gr1j,SR1k,SR2k,SR3j,SR4j,Auth2ij,Auth4jk,Auth6kj,Auth8ji,URandi,GRandj,SRandk,CT1i,CT2j,CT3k,CT4j,Accessi:message,
Inc:hash_func
const
user_gateway_UR3i,gateway_sensor_Auth4jk,sensor_gateway_Auth6kj,gateway_user_Auth8ji,
user_gateway_Auth2ij,gateway_user_UR5j,sensor_gateway_SR4j,gateway_sensor_SR2k,
subs1,subs2,subs3,subs4,subs5,subs6:protocol_id
init State:=0
transition
1.State=0/\Rcv(start)=|>
State':=1/\GPrij':=new()
/\GPubj':=new()
/\secret({GPrij'},subs4,GWj)
2.State=1/\Rcv({UIDi.UR1i'.UR3i'.Uri'}_GPrij)=|>
State':=2/\Gr1j':=new()
/\UR4j':=H(UR1i'.GIDj.GPrij)
/\UR5j':=H(UR4j'.UR3i'.xor(xor(UGKeyij,GPubj),Gr1j'))
/\Snd(UR4j'.UR5j'.Gr1j')
/\request(Ui,GWj,user_gateway_UR3i,UR3i)
/\witness(GWj,Ui,gateway_user_UR5j,UR5j)
3.State=2/\Rcv({SIDk.SR1k'.SR2k'}_GPrij)=|>
State':=3/\Gr2j':=new()
/\SR3j':=H(SR2k'.Gr2j'.GPrij)
/\SR4j':=H(SR3j'.SR1k'.xor(xor(GIDj.Gr2j'),GPubj))
```

GATEWAY ROLE :



```
/\Snd({SR3j'.SR4j'.Gr2j'}_GPrij)
/\request(SNk,GWj,sensor_gateway_SR2k,SR2k)
/\witness(GWj,SNk,gateway_sensor_SR4j,SR4j)
4.State=3/\Rcv(Accessi'.Auth2ij'.URandi'.CT1i')=|>
State':=4/\CT2j':=new()
/\GRandj':=new()
/\Auth3jk':=H(SR2k.SGKeykj.SPubk.URandi)
/\Auth4jk':=H(Auth3jk'.GRandj'.CT2j'.SR4j)
/\Snd(Auth4jk'.GRandj'.URandi'.CT2j)
/\request(Ui,GWj,user_gateway_Auth2ij,Auth2ij)
/\witness(GWj,SNk,gateway_sensor_Auth4jk,Auth4jk)
5.State=4/\Rcv(Auth6kj'.SRandk'.CT3k')=|>
State':=5/\CT4j':=new()
/\Auth7ji':=H(Auth1ij.GPubj.xor(UR1i,UR5j))
/\Auth8ji':=H(Auth7ji'.GRandj.SRandk'.CT4j')
/\Snd(Auth8ji'.GRandj.SRandk'.CT4j')
/\request(SNk,GWj,sensor_gateway_Auth6kj,Auth6kj)
/\witness(GWj,Ui,gateway_user_Auth8ji,Auth8ji)
end role
```

The "gateway" role represents the behavior and actions of a gateway (GWj) in the network security protocol. The gateway acts as an intermediary between the user (Ui) and the sensor (SNk) and is responsible for facilitating secure communication and mutual authentication.

Here's a step-by-step explanation of the "gateway" role:

1. **Initialization (State 0):**
    - The gateway's initial state is set to 0, indicating the start of the authentication process.
    - The gateway role listens for the "start" message to begin the authentication.
2. **Knowledge Factor, Possession Factor, and Biometric Factor (State 1):**
    - Upon receiving the "start" message, the gateway generates and stores new values for knowledge (GPrij), possession (GPubj), and biometric (GIDj) factors.
    - The gateway computes SR1k' by hashing the concatenation of SIDk (sensor's unique identifier), SSeck (sensor's possession factor), and Srk (sensor's biometric factor).
    - SR5k is calculated by hashing the concatenation of GPubj and SGKeykj (symmetric key shared between gateway and sensor).
    - SR2k' is computed by hashing the concatenation of SR1k', SR5k', and the XOR of SGKeykj and GIDj (gateway's ID).

- The gateway sends the encrypted tuple {SIDk, SR1k', SR2k'} to the sensor using the sensor's public key (SPubk).

3. **User Interaction (State 2):**

- In this state, the gateway role waits to receive a message from the user containing UR1i' and UR3i'.
- The gateway computes UPrik' (private key shared between user and gateway) by hashing the concatenation of UR1i', UR5j', and the XOR of Uri and Gr1j' (random value generated by gateway).
- UPubi' is calculated by hashing the concatenation of UPrik' and UR2i.
- The gateway generates a new random value (CT1i') and URandi' and forms the access message {UIDi, URandi', SIDk} to be sent to the sensor using the sensor's public key (SPubk).
- Auth1ij is computed using a hash function (H) on the concatenation of UR1i', UGKeyij, and UPubi'.
- Auth2ij is calculated by hashing the concatenation of Auth1ij, URandi', CT1i', and UR5j'.
- The gateway sends the tuple {Accessi', Auth2ij, URandi', CT1i'} to the sensor.

4. **Receiving Authentication (State 3):**

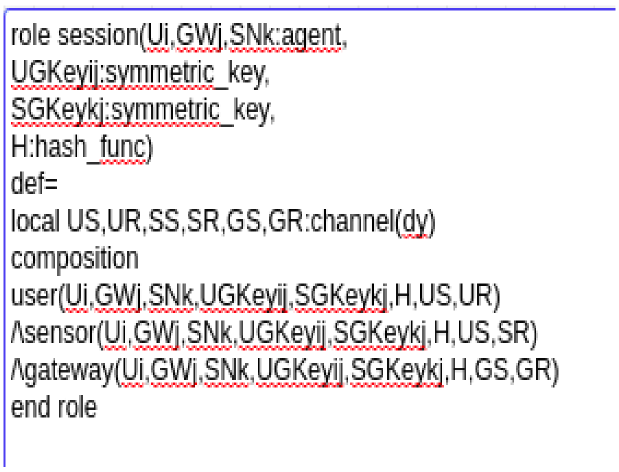
- In this state, the gateway receives authentication information from the sensor, including Auth6kj', SRandk', and CT3k'.
- The gateway computes SKik' (shared session key between user and gateway) by hashing XORed values based on SRandk', UR3i, SR1k', UR3i, and SR1k'.
- The gateway sends the encrypted tuple {xor(xor(UR3i, URandi), SRandk')}_SPrik to the user.

5. **User Interaction and Final Authentication (State 4):**

- In this state, the gateway receives final authentication information from the user, including {xor(xor(SR1k', SPubk'), UPubi)}_UPrii.
- The gateway computes SKik' (shared session key between user and gateway) again by hashing XORed values based on SRandk', UR3i, SR1k', UR3i, and SR1k'.
- The gateway completes the authentication process, and the shared session key SKik' is now established between the user and the gateway.

The gateway role plays a critical role in the authentication process by generating and exchanging various cryptographic values with the user and sensor. It facilitates secure communication and mutual authentication between the user and the sensor, ensuring the network's resources are accessed securely and efficiently.

SESSION ROLE



```
role session(Ui,GWj,SNk:agent,
UGKeyij:symmetric_key,
SGKeykj:symmetric_key,
H:hash_func)
def=
local US,UR,SS,SR,GS,GR:channel(dy)
composition
user(Ui,GWj,SNk,UGKeyij,SGKeykj,H,US,UR)
/\sensor(Ui,GWj,SNk,UGKeyij,SGKeykj,H,US,SR)
/\gateway(Ui,GWj,SNk,UGKeyij,SGKeykj,H,GS,GR)
end role
```

The "session" role represents the interaction between a user (Ui), a gateway (GWj), and a sensor (SNk) in the network security protocol. It combines the behaviors and actions of these three entities to establish secure communication and perform mutual authentication.

Here's a step-by-step explanation of the "session" role:

1. **Initialization and Composition:**
    - The session role composes the behaviors of the user, gateway, and sensor roles into a single session role.
    - Each instance of the session role involves a specific user (Ui), gateway (GWj), and sensor (SNk).
2. **Communication and Authentication:**
    - The session role allows for secure communication and authentication between the user, gateway, and sensor.
3. **User Role (user()):**

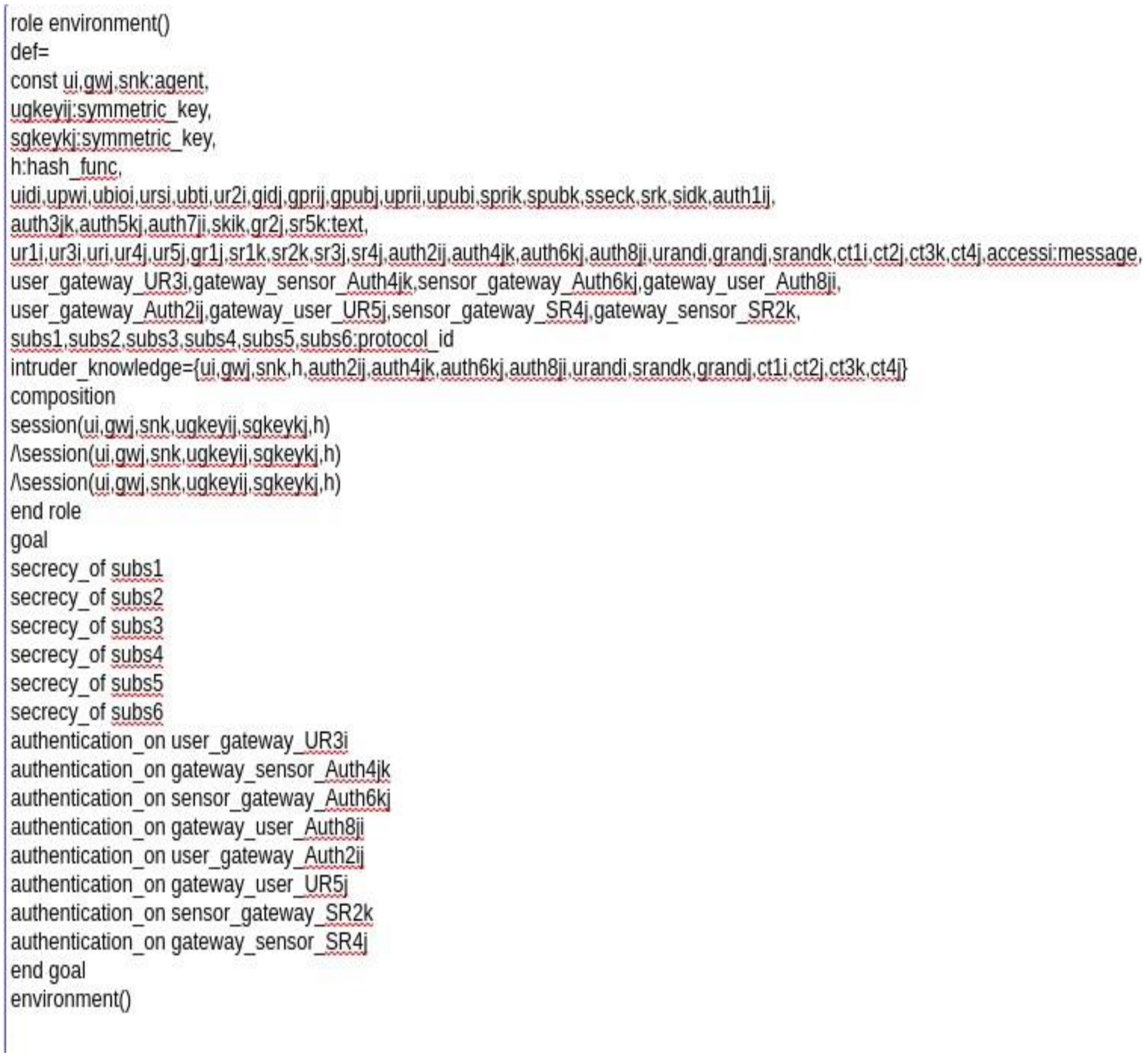
- The user role is responsible for generating and exchanging various cryptographic values with the gateway and sensor to establish secure communication and authentication.

- The user's actions include providing knowledge, possession, and biometric factors, sending messages to the gateway and sensor, and computing shared session keys (SKik').

4. **Gateway Role (gateway()):**
   - The gateway role interacts with the user and sensor to facilitate secure communication and mutual authentication.
   - The gateway's actions include receiving messages from the user and sensor, computing private keys (UPrii) and shared session keys (SKik'), and sending encrypted tuples to the sensor.
5. **Sensor Role (sensor()):**
   - The sensor role interacts with the user and gateway to facilitate secure communication and mutual authentication.
   - The sensor's actions include generating biometric and possession factors, computing private keys (SPrik') and shared session keys (SKik'), and sending authentication messages to the gateway.
6. **Session Composition (composition):**
   - The session role combines the behaviors of the user, gateway, and sensor roles into a single session.
   - The "composition" statement ensures that all three roles (user, gateway, and sensor) work together within the same session.

The session role plays a vital role in orchestrating the communication and authentication between the user, gateway, and sensor, contributing to the overall security of the network protocol. It coordinates the actions of these entities to ensure secure data exchange and access to the network's resources.

ENVIRONMENT ROLE



```
role environment()
def=
const ui,gwj,snk:agent,
ugkeyij:symmetric_key,
sgkeykj:symmetric_key,
h:hash_func,
uidi,upwi,ubioi,ursi,ubti,ur2i,gidj,gprij,gpubj,uprii,upubi,sprik,spubk,sseck,srk,sidk,auth1ij,
auth3jk,auth5kj,auth7ji,skik,gr2j,sr5k:text,
ur1i,ur3i,uri,ur4j,ur5j,gr1j,sr1k,sr2k,sr3j,sr4j,auth2ij,auth4jk,auth6kj,auth8ji,urandi,grandj,srandk,ct1i,ct2j,ct3k,ct4j,accessi:message,
user_gateway_UR3i,gateway_sensor_Auth4jk,sensor_gateway_Auth6kj,gateway_user_Auth8ji,
user_gateway_Auth2ij,gateway_user_UR5j,sensor_gateway_SR4j,gateway_sensor_SR2k,
subs1,subs2,subs3,subs4,subs5,subs6:protocol_id
intruder_knowledge={ui,gwj,snk,h,auth2ij,auth4jk,auth6kj,auth8ji,urandi,srandk,grandj,ct1i,ct2j,ct3k,ct4j}
composition
session(ui,gwj,snk,ugkeyij,sgkeykj,h)
/\session(ui,gwj,snk,ugkeyij,sgkeykj,h)
/\session(ui,gwj,snk,ugkeyij,sgkeykj,h)
end role
goal
secrecy_of subs1
secrecy_of subs2
secrecy_of subs3
secrecy_of subs4
secrecy_of subs5
secrecy_of subs6
authentication_on user_gateway_UR3i
authentication_on gateway_sensor_Auth4jk
authentication_on sensor_gateway_Auth6kj
authentication_on gateway_user_Auth8ji
authentication_on user_gateway_Auth2ij
authentication_on gateway_user_UR5j
authentication_on sensor_gateway_SR2k
authentication_on gateway_sensor_SR4j
end goal
environment()
```

The "environment" role in the provided code represents the surrounding environment in which the network security protocol is deployed. It defines the constant values and interactions that the agents (user, gateway, and sensor) in the network have with the environment. The environment role is an abstract representation of external elements interacting with the agents and the protocol.

Here's an explanation of the "environment" role and its components:

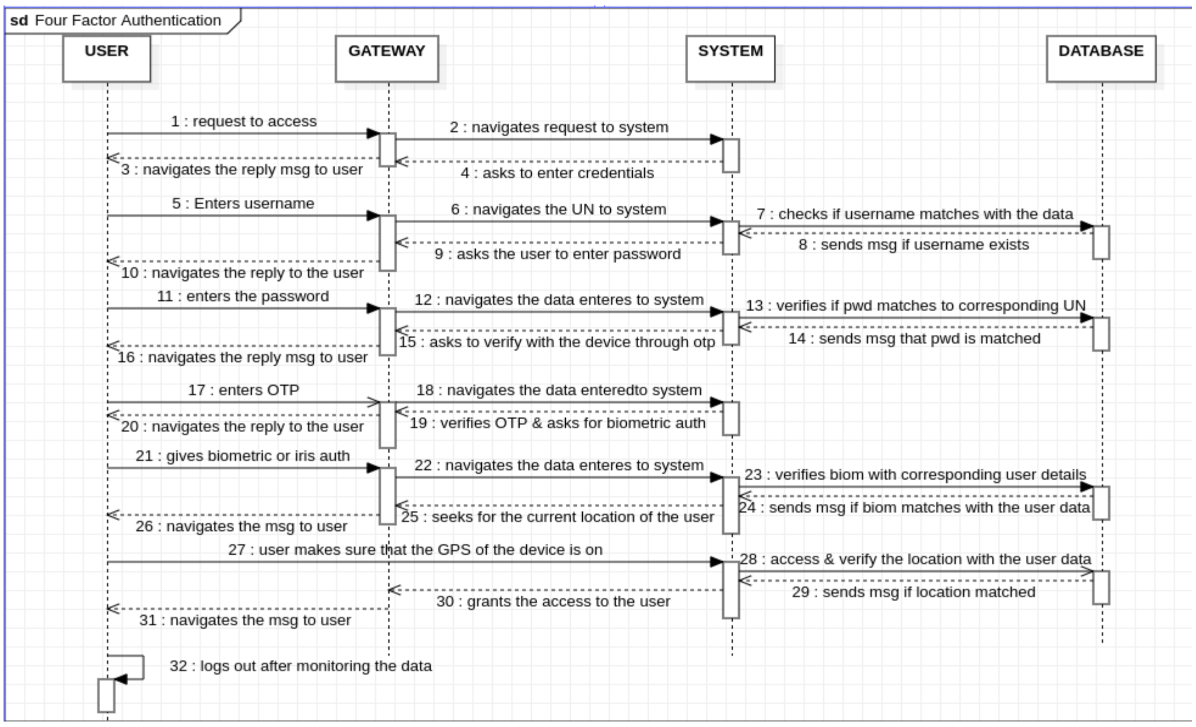
1. **Initialization (const):**
    - The "const" section defines constant values that are known to all the agents in the network. These constants include agent identifiers (ui, gwj, snk), symmetric keys (ugkeyij, sgkeykj), a hash function (h), and various message variables (e.g., uidi, upwi, etc.).
    - These constant values are shared between the agents and are used for various cryptographic operations and authentication.
2. **Intruder Knowledge (intruder_knowledge):**
    - The "intruder_knowledge" section specifies a set of variables and values that the intruder in the network possesses.
    - The intruder has access to knowledge of various cryptographic values such as shared keys (ugkeyij, sgkeykj), hashes (h), and authentication information (auth2ij, auth4jk, auth6kj, auth8ji).
    - The intruder's knowledge represents a security threat to the network and is used to model potential attacks or vulnerabilities in the protocol.
3. **Composition and Interaction (composition):**
    - The "composition" statement combines the instances of the "session" role, representing multiple interactions between users, gateways, and sensors.
    - The "composition" section defines multiple instances of the "session" role involving different combinations of users, gateways, and sensors.
    - Each instance of the "session" role is a specific network session involving a particular user, gateway, and sensor.
4. **Security Goals (goal):**
    - The "goal" section defines the security goals that the network protocol aims to achieve.
    - The defined goals include ensuring secrecy of certain protocol

identifiers (subs1 to subs6) and authentication properties on various

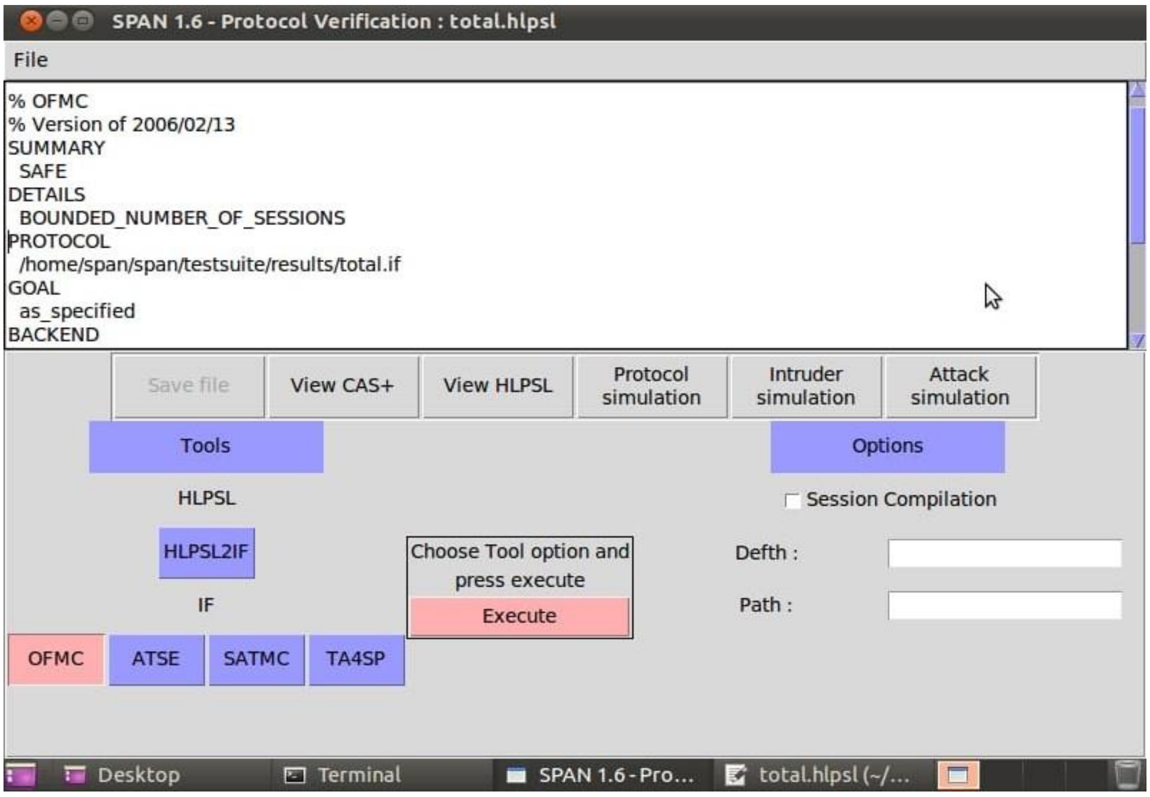
- roles (user_gateway_UR3i, gateway_sensor_Auth4jk, sensor_gateway_Auth6kj, gateway_user_Auth8ji, user_gateway_Auth2ij, gateway_user_UR5j, sensor_gateway_SR2k, gateway_sensor_SR4j).
- The security goals specify what information should remain confidential and what authentication mechanisms should be enforced to maintain network security.

The "environment" role serves as a high-level specification of the external factors and constraints within which the network security protocol operates. It defines the constant values and security goals, contributing to the formal modeling and analysis of the protocol's security properties and vulnerabilities. By including an "intruder_knowledge" section, the environment role allows for the modeling of potential attacks by an adversary with certain knowledge and capabilities.

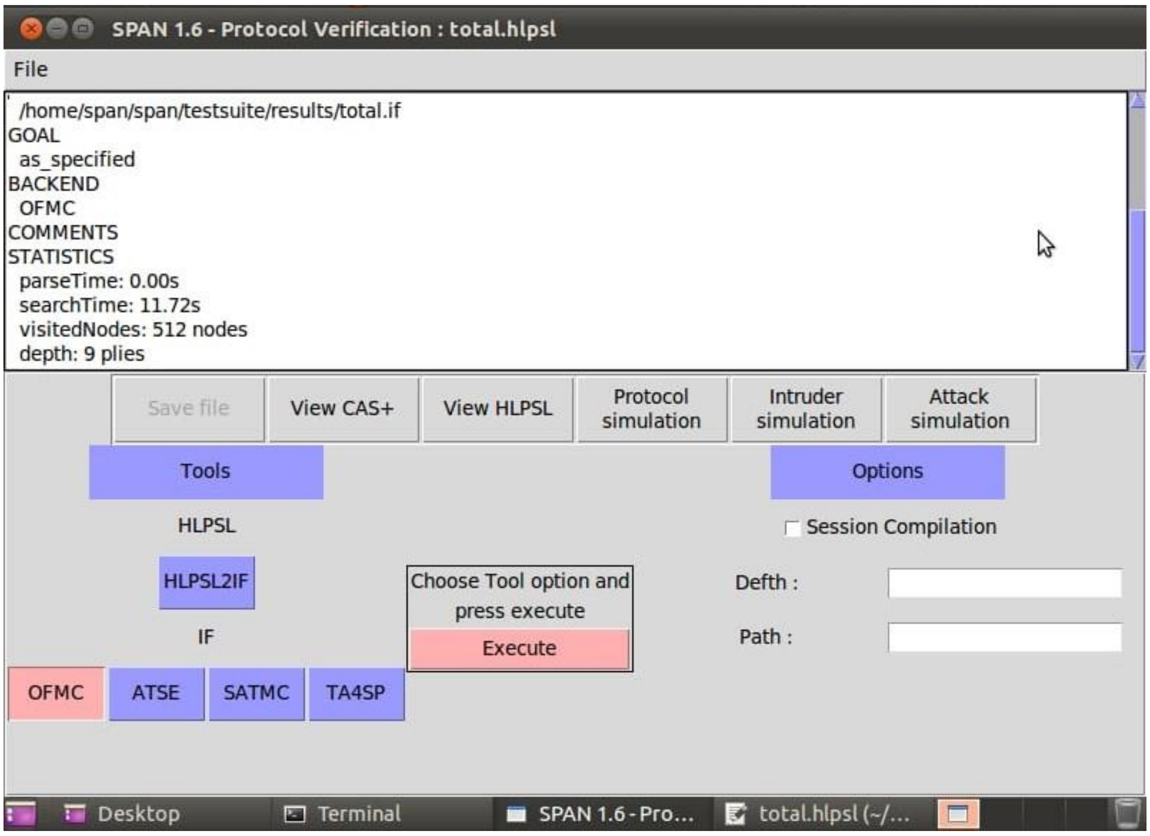
SEQUENCE DIAGRAM


sd Four Factor Authentication
USER
GATEWAY
SYSTEM
DATABASE
1 : request to access
2 : navigates request to system
3 : navigates the reply msg to user
4 : asks to enter credentials
5 : Enters username
6 : navigates the UN to system
7 : checks if username matches with the data
8 : sends msg if username exists
9 : asks the user to enter password
10 : navigates the reply to the user
11 : enters the password
12 : navigates the data enteres to system
13 : verifies if pwd matches to corresponding UN
14 : sends msg that pwd is matched
15 : asks to verify with the device through otp
16 : navigates the reply msg to user
17 : enters OTP
18 : navigates the data enteredto system
19 : verifies OTP & asks for biometric auth
20 : navigates the reply to the user
21 : gives biometric or iris auth
22 : navigates the data enteres to system
23 : verifies biom with corresponding user details
24 : sends msg if biom matches with the user data
25 : seeks for the current location of the user
26 : navigates the msg to user
27 : user makes sure that the GPS of the device is on
28 : access & verify the location with the user data
29 : sends msg if location matched
30 : grants the access to the user
31 : navigates the msg to user
32 : logs out after monitoring the data

b. RESULT:



```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/total.if
GOAL
  as_specified
BACKEND
```



```
  /home/span/span/testsuite/results/total.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 11.72s
  visitedNodes: 512 nodes
  depth: 9 plies
```

Fig- OFMC RESULT

Four Factor Authentication using Ocaml

OCaml (Objective Caml) is a general-purpose programming language that is particularly well- suited for functional programming. It has been used for various purposes due to its unique featuresand advantages. Here are some reasons why OCaml code is used:

1. Strong type system: OCaml has a powerful type system that helps catch many errors at compile-time, reducing bugs and improving code reliability. This makes it an attractivechoice for building critical and robust systems.
2. Functional programming support: OCaml supports functional programming paradigms, which promote immutability and pure functions. This leads to code that is easier to reasonabout, understand, and test.
3. Performance: OCaml is designed to be efficient and can deliver high-performance code. Itsnative compilation produces optimized machine code, making it suitable for performance- sensitive applications.
4. Conciseness and expressiveness: OCaml code is often more concise and expressive thancode in other languages. It allows developers to achieve more with less code, making it easier to maintain and understand.
5. Interoperability: OCaml provides good support for interoperability with other programminglanguages, such as C and C++. This allows developers to leverage existing libraries and systems.
6. Strong community and ecosystem: OCaml has an active and passionate community, contributing to its ecosystem of libraries, tools, and frameworks. The availability of variouspackages helps developers be more productive and avoid reinventing the wheel.
7. Type inference: OCaml features a powerful type inference system that can automaticallydeduce the types of expressions in the code. This reduces the need for explicit type annotations, resulting in

cleaner and more concise code.

8. Pattern matching: OCaml's pattern matching capabilities are particularly powerful and expressive, making it easy to handle complex data structures and perform transformations onthem.
9. Compiler support: OCaml's compiler provides useful error messages and warnings that aidin writing high-quality code and identifying potential issues early in the development process.
10. Security: OCaml's strong type system and functional programming features can help preventmany common security vulnerabilities, such as null pointer dereferences and buffer overflows.

OCaml is often chosen for a range of applications, including language implementations, financial systems, compilers, static analyzers, formal verification tools, and more.It strikes a balance between expressiveness, safety, and performance, making it an excellent choice for various projects.

GENERAL PROCESS FLOW:

1.Ocaml code execution begins at the top level i.e. 1st line of the program.

2.The code is then executed sequentially from top to bottom.Function definitions,type definitions,let bindings etc., are all processed in the order they appear.

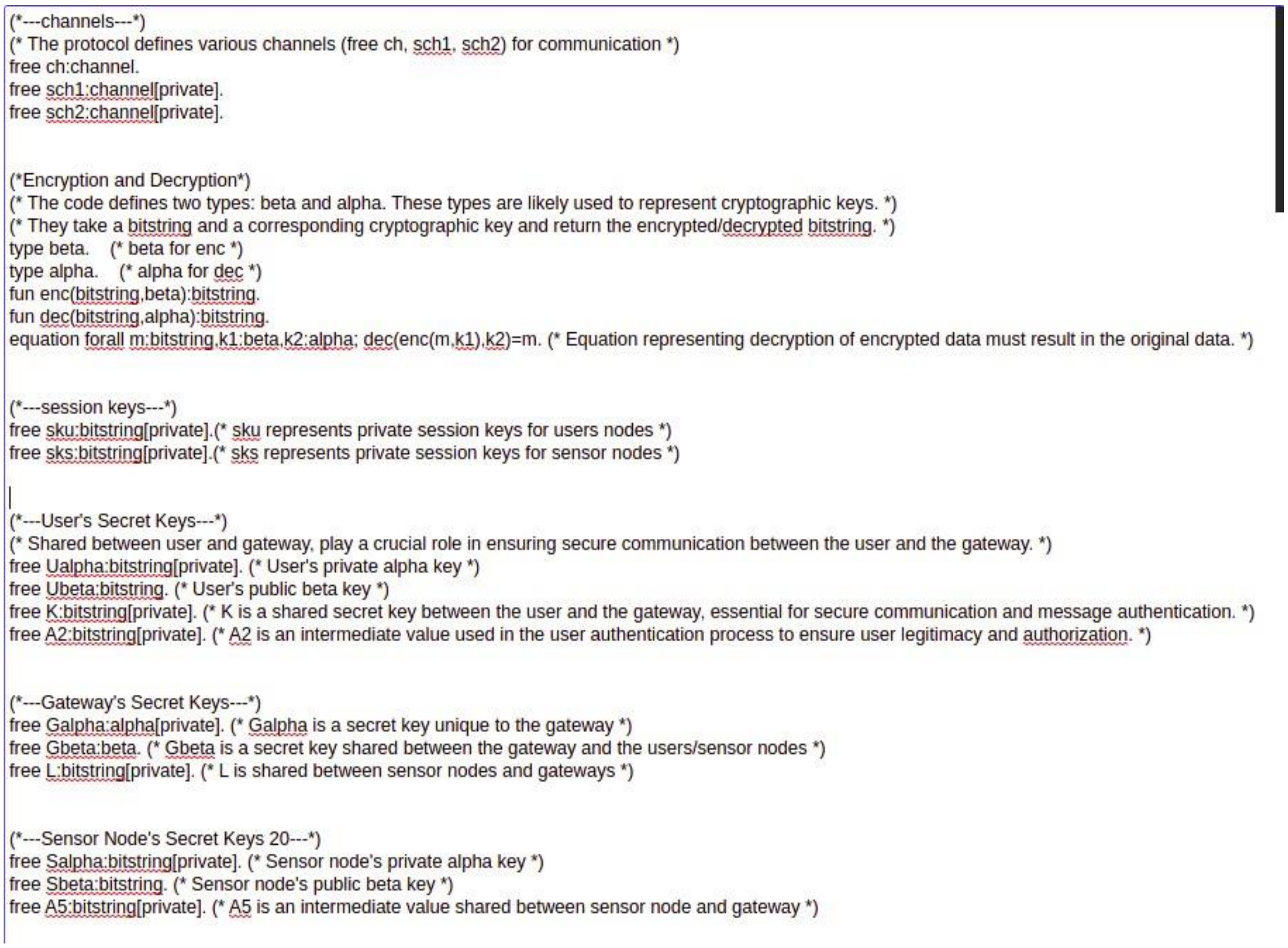
3.If the program encounters a function call, the sequential execution is paused and jumps to the function definition and begins executing that function.

4.Ocaml uses exceptions to handle errors.When an exception is thrown,the control flow is immediately passed to the nearest enclosing exception handler.If no such handler is found, the program terminates

5.Once all the code has been executed the program exits.

a. IMPLEMENTATION

Channels: You have defined three channels - ch, sch1, and sch2. The channel ch is a regular channel, while sch1 and sch2 are private channels. Private channels are



```
(*---channels---*)
(* The protocol defines various channels (free ch, sch1, sch2) for communication *)
free ch:channel.
free sch1:channel[private].
free sch2:channel[private].


(*Encryption and Decryption*)
(* The code defines two types: beta and alpha. These types are likely used to represent cryptographic keys. *)
(* They take a bitstring and a corresponding cryptographic key and return the encrypted/decrypted bitstring. *)
type beta.    (* beta for enc *)
type alpha.    (* alpha for dec *)
fun enc(bitstring,beta):bitstring.
fun dec(bitstring,alpha):bitstring.
equation forall m:bitstring,k1:beta,k2:alpha; dec(enc(m,k1),k2)=m. (* Equation representing decryption of encrypted data must result in the original data. *)


(*---session keys---*)
free sku:bitstring[private].(* sku represents private session keys for users nodes *)
free sks:bitstring[private].(* sks represents private session keys for sensor nodes *)


(*---User's Secret Keys---*)
(* Shared between user and gateway, play a crucial role in ensuring secure communication between the user and the gateway. *)
free Ualpha:bitstring[private]. (* User's private alpha key *)
free Ubeta:bitstring. (* User's public beta key *)
free K:bitstring[private]. (* K is a shared secret key between the user and the gateway, essential for secure communication and message authentication. *)
free A2:bitstring[private]. (* A2 is an intermediate value used in the user authentication process to ensure user legitimacy and authorization. *)


(*---Gateway's Secret Keys---*)
free Galpha:alpha[private]. (* Galpha is a secret key unique to the gateway *)
free Gbeta:beta. (* Gbeta is a secret key shared between the gateway and the users/sensor nodes *)
free L:bitstring[private]. (* L is shared between sensor nodes and gateways *)


(*---Sensor Node's Secret Keys 20---*)
free Salpha:bitstring[private]. (* Sensor node's private alpha key *)
free Sbeta:bitstring. (* Sensor node's public beta key *)
free A5:bitstring[private]. (* A5 is an intermediate value shared between sensor node and gateway *)
```

channels that can only beaccessed by specific participants.

Encryption and Decryption: Two functions enc and dec are defined for encryption and decryption, respectively. The enc function takes a bitstring and a beta as input and returns a bitstring, while the dec function takes a bitstring and an alpha as input and returns a bitstring. Theequation dec(enc(m,k1),k2)=m represents that if you encrypt a message m using k1 and then decrypt it using k2, you should get back the original message m.

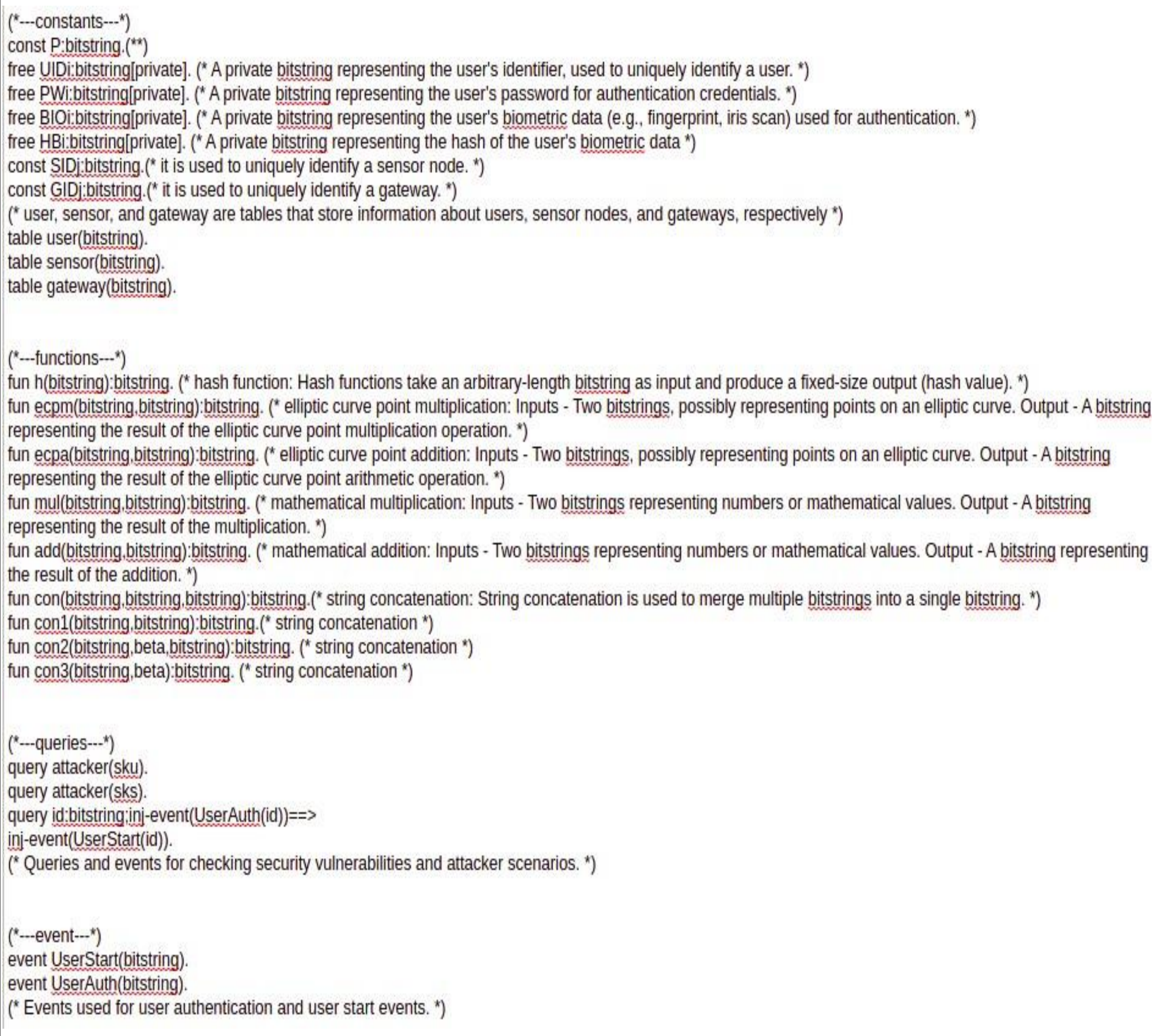
Session Keys: Two private bitstring session keys sku and sks are defined. These session keys areused for cryptographic purposes during the protocol execution.

User's Secret Keys: Three private bitstring variables Ualpha, Ubeta, and K are defined, representing the secret keys of the user. Additionally, A2 is defined as a private bitstring, which isshared between the user and the gateway.

Gateway's Secret Keys: Two private keys Galpha and Gbeta of types alpha and beta, respectively, are defined for the gateway. Additionally, L is defined as a private bitstring, which isshared between the sensor node and the gateway.

Sensor Node's Secret Keys: Two private bitstring keys Salpha and Sbeta are defined for the sensor node. Moreover, A5 is defined as a private bitstring, which is shared between the sensor nodeand the gateway.

Constants: Two constants are defined - P and SIDj, both of type bitstring. Constants have fixedvalues and are useful for defining predefined values in the protocol.



```
(*---constants---*)
const P:bitstring.(**)
free UIDi:bitstring[private]. (* A private bitstring representing the user's identifier, used to uniquely identify a user. *)
free PWi:bitstring[private]. (* A private bitstring representing the user's password for authentication credentials. *)
free BIOi:bitstring[private]. (* A private bitstring representing the user's biometric data (e.g., fingerprint, iris scan) used for authentication. *)
free HBi:bitstring[private]. (* A private bitstring representing the hash of the user's biometric data *)
const SIDj:bitstring.(* it is used to uniquely identify a sensor node. *)
const GIDj:bitstring.(* it is used to uniquely identify a gateway. *)
(* user, sensor, and gateway are tables that store information about users, sensor nodes, and gateways, respectively *)
table user(bitstring).
table sensor(bitstring).
table gateway(bitstring).


(*---functions---*)
fun h(bitstring):bitstring. (* hash function: Hash functions take an arbitrary-length bitstring as input and produce a fixed-size output (hash value). *)
fun ecpm(bitstring,bitstring):bitstring. (* elliptic curve point multiplication: Inputs - Two bitstrings, possibly representing points on an elliptic curve. Output - A bitstring
representing the result of the elliptic curve point multiplication operation. *)
fun ecpa(bitstring,bitstring):bitstring. (* elliptic curve point addition: Inputs - Two bitstrings, possibly representing points on an elliptic curve. Output - A bitstring
representing the result of the elliptic curve point arithmetic operation. *)
fun mul(bitstring,bitstring):bitstring. (* mathematical multiplication: Inputs - Two bitstrings representing numbers or mathematical values. Output - A bitstring
representing the result of the multiplication. *)
fun add(bitstring,bitstring):bitstring. (* mathematical addition: Inputs - Two bitstrings representing numbers or mathematical values. Output - A bitstring representing
the result of the addition. *)
fun con(bitstring,bitstring,bitstring):bitstring.(* string concatenation: String concatenation is used to merge multiple bitstrings into a single bitstring. *)
fun con1(bitstring,bitstring):bitstring.(* string concatenation *)
fun con2(bitstring,beta,bitstring):bitstring. (* string concatenation *)
fun con3(bitstring,beta):bitstring. (* string concatenation *)


(*---queries---*)
query attacker(sku).
query attacker(sks).
query id:bitstring;inj-event(UserAuth(id))==>
inj-event(UserStart(id)).
(* Queries and events for checking security vulnerabilities and attacker scenarios. *)


(*---event---*)
event UserStart(bitstring).
event UserAuth(bitstring).
(* Events used for user authentication and user start events. *)
```

Tables: Three tables user, sensor, and gateway are defined with the type bitstring. Tables are used to store information about the entities in the protocol. They can be used to keep track of data relatedto users, sensor nodes, and gateways.

Functions:

h(bitstring): bitstring: This is a hash function that takes a bitstring as input and returns anotherbitstring.

ecpm(bitstring, bitstring): bitstring: This is a function for elliptic curve point multiplication thattakes two bitstring inputs and returns a bitstring.

ecpa(bitstring, bitstring): bitstring: This is another function for elliptic curve point addition thattakes two bitstring inputs and returns a bitstring.

mul(bitstring, bitstring): bitstring: This function performs mathematical multiplication on twobitstring inputs and returns a bitstring.

add(bitstring, bitstring): bitstring: This function performs mathematical addition on two bitstringinputs and returns a bitstring.

con(bitstring, bitstring, bitstring): bitstring: This function concatenates three bitstring inputs andreturns a bitstring.

con1(bitstring, bitstring): bitstring: This function concatenates two bitstring inputs and returns abitstring.

con2(bitstring, beta, bitstring): bitstring: This function concatenates a bitstring, a beta, andanother bitstring, and returns a bitstring.

con3(bitstring, beta): bitstring: This function concatenates a bitstring and a beta, and returns abitstring.

Queries:

query attacker(sku): This query checks if there is an attacker who can learn the value of sku (thesession key of the user).

query attacker(sks): This query checks if there is an attacker who can learn the value of sks (thesession key of the sensor).

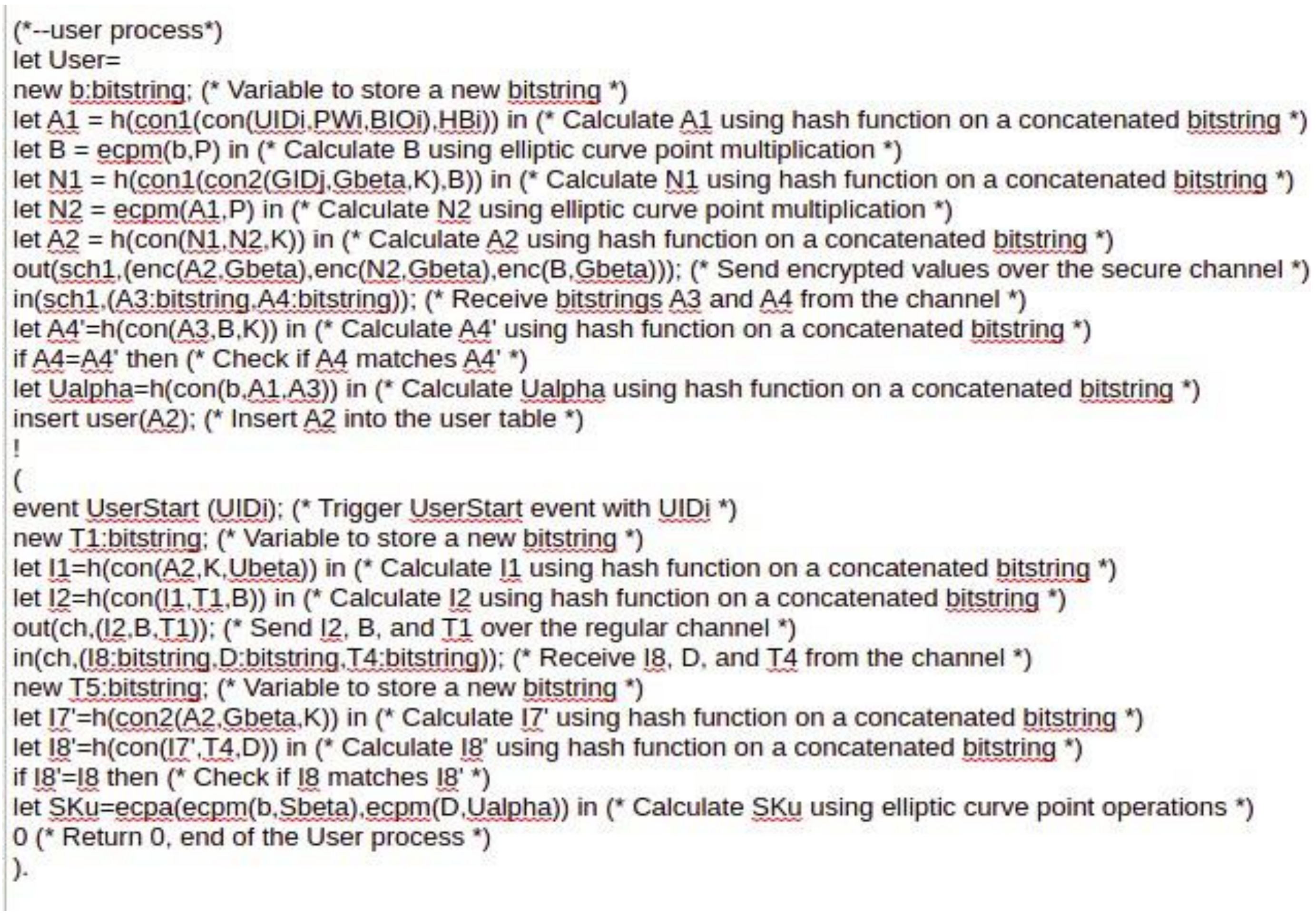
query id: bitstring; inj-event(UserAuth(id)) ==> inj-event(UserStart(id)): This query specifiesthat if an attacker can inject a UserAuth event with id, then the attacker can also inject a corresponding UserStart event with the same id.

Events:

event UserStart(bitstring): This is an event declaration for UserStart, which represents the start ofa user in the protocol.

event UserAuth(bitstring): This is an event declaration for UserAuth

USER PROCESS



```
(*--user process*)
let User=
new b:bitstring; (* Variable to store a new bitstring *)
let A1 = h(con1(con(UIDi,PWi,BIOi),HBi)) in (* Calculate A1 using hash function on a concatenated bitstring *)
let B = ecpm(b,P) in (* Calculate B using elliptic curve point multiplication *)
let N1 = h(con1(con2(GIDj,Gbeta,K),B)) in (* Calculate N1 using hash function on a concatenated bitstring *)
let N2 = ecpm(A1,P) in (* Calculate N2 using elliptic curve point multiplication *)
let A2 = h(con(N1,N2,K)) in (* Calculate A2 using hash function on a concatenated bitstring *)
out(sch1,(enc(A2,Gbeta),enc(N2,Gbeta),enc(B,Gbeta))); (* Send encrypted values over the secure channel *)
in(sch1,(A3:bitstring,A4:bitstring)); (* Receive bitstrings A3 and A4 from the channel *)
let A4'=h(con(A3,B,K)) in (* Calculate A4' using hash function on a concatenated bitstring *)
if A4=A4' then (* Check if A4 matches A4' *)
let Ualpha=h(con(b,A1,A3)) in (* Calculate Ualpha using hash function on a concatenated bitstring *)
insert user(A2); (* Insert A2 into the user table *)
!
(
event UserStart (UIDi); (* Trigger UserStart event with UIDi *)
new T1:bitstring; (* Variable to store a new bitstring *)
let I1=h(con(A2,K,Ubeta)) in (* Calculate I1 using hash function on a concatenated bitstring *)
let I2=h(con(I1,T1,B)) in (* Calculate I2 using hash function on a concatenated bitstring *)
out(ch,(I2,B,T1)); (* Send I2, B, and T1 over the regular channel *)
in(ch,(I8:bitstring,D:bitstring,T4:bitstring)); (* Receive I8, D, and T4 from the channel *)
new T5:bitstring; (* Variable to store a new bitstring *)
let I7'=h(con2(A2,Gbeta,K)) in (* Calculate I7' using hash function on a concatenated bitstring *)
let I8'=h(con(I7',T4,D)) in (* Calculate I8' using hash function on a concatenated bitstring *)
if I8'=I8 then (* Check if I8 matches I8' *)
let SKu=ecpa(ecpm(b,Sbeta),ecpm(D,Ualpha)) in (* Calculate SKu using elliptic curve point operations *)
0 (* Return 0, end of the User process *)
).
```

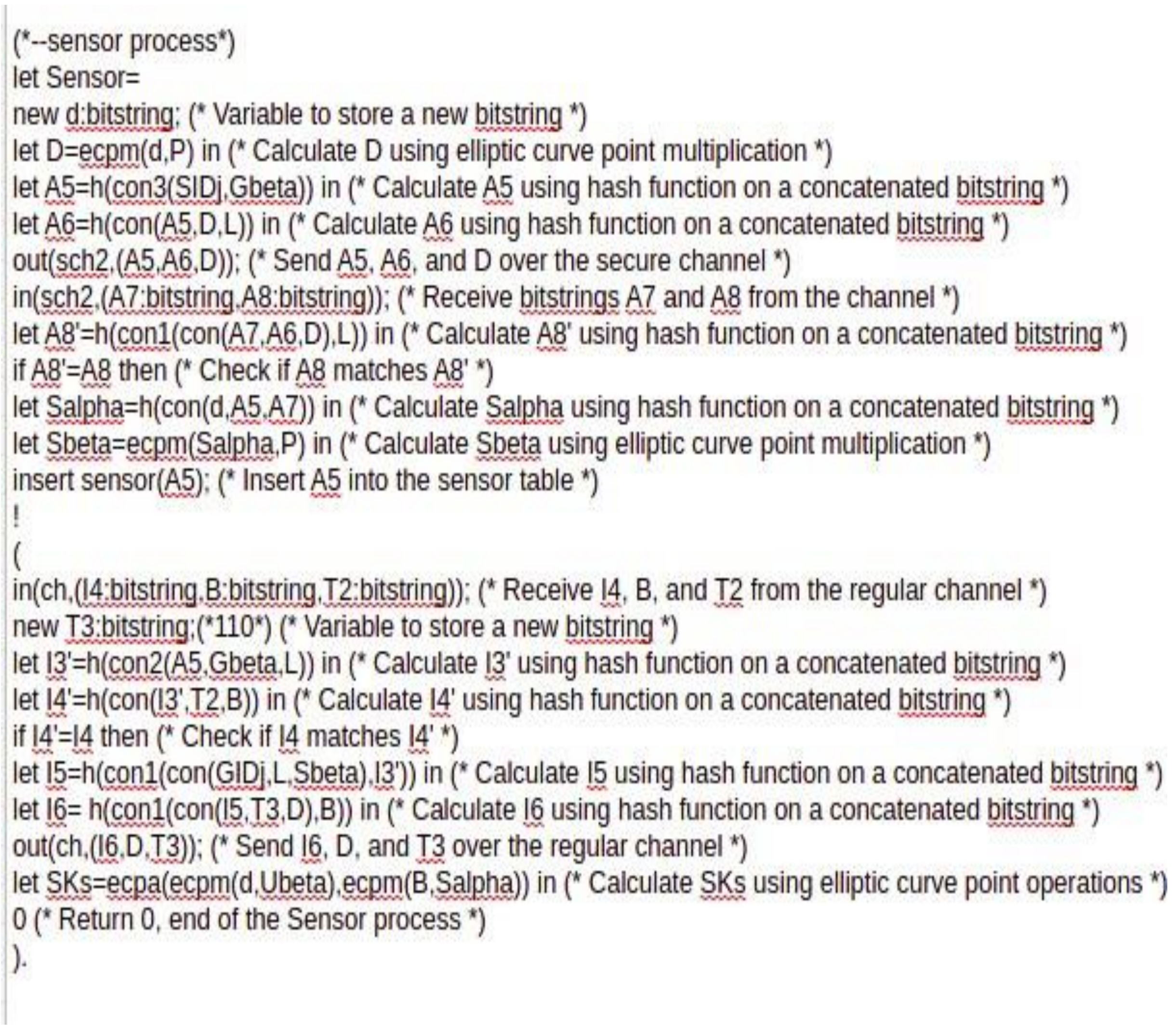
A new bitstring b is generated.
A1 is computed as the hash of the concatenation of con1(con(UIDi, PWi, BIOi), HBi).
B is calculated as the result of elliptic curve point multiplication (ecpm) with inputs b and P.N1 is obtained by hashing the concatenation of con1(con2(GIDj, Gbeta, K), B).
N2 is computed as the result of elliptic curve point multiplication (ecpm) with inputs A1 and P.A2 is obtained by hashing the concatenation of N1, N2, and K.
The tuple (enc(A2, Gbeta), enc(N2, Gbeta), enc(B, Gbeta)) is sent through the private channelsch1.
The process waits to receive a tuple (A3: bitstring, A4: bitstring) on channel sch1.A4' is computed as the hash of con(A3, B, K).

If A4 is equal to A4', then the process continues with the rest of the code inside the if block.Inside the if block (denoted by (*72*)):

Ualpha is calculated as the hash of con(b, A1, A3).The user table is updated with the value A2

An output event UserStart(UIDi) is generated.A new bitstring T1 is generated.
I1 is computed as the hash of con(A2, K, Ubeta).
I2 is obtained by hashing the concatenation of I1, T1, and B.The tuple (I2, B, T1) is sent through the channel ch.
The process waits to receive a tuple (I8: bitstring, D: bitstring, T4: bitstring) on channel ch.A new bitstring T5 is generated.
I7' is computed as the hash of con2(A2, Gbeta, K).
I8' is obtained by hashing the concatenation of I7', T4, and D.
If I8 is equal to I8', then a session key SKu is computed using elliptic curve point addition (ecpa)of the results of two elliptic curve point multiplications (ecpm).
The process returns 0, indicating successful completion.

SENSOR PROCESS:



```
(*--sensor process*)
let Sensor=
new d:bitstring; (* Variable to store a new bitstring *)
let D=ecpm(d,P) in (* Calculate D using elliptic curve point multiplication *)
let A5=h(con3(SIDj,Gbeta)) in (* Calculate A5 using hash function on a concatenated bitstring *)
let A6=h(con(A5,D,L)) in (* Calculate A6 using hash function on a concatenated bitstring *)
out(sch2,(A5,A6,D)); (* Send A5, A6, and D over the secure channel *)
in(sch2,(A7:bitstring,A8:bitstring)); (* Receive bitstrings A7 and A8 from the channel *)
let A8'=h(con1(con(A7,A6,D),L)) in (* Calculate A8' using hash function on a concatenated bitstring *)
if A8'=A8 then (* Check if A8 matches A8' *)
let Salpha=h(con(d,A5,A7)) in (* Calculate Salpha using hash function on a concatenated bitstring *)
let Sbeta=ecpm(Salpha,P) in (* Calculate Sbeta using elliptic curve point multiplication *)
insert sensor(A5); (* Insert A5 into the sensor table *)
!
(
in(ch,(I4:bitstring,B:bitstring,T2:bitstring)); (* Receive I4, B, and T2 from the regular channel *)
new T3:bitstring;(*110*) (* Variable to store a new bitstring *)
let I3'=h(con2(A5,Gbeta,L)) in (* Calculate I3' using hash function on a concatenated bitstring *)
let I4'=h(con(I3',T2,B)) in (* Calculate I4' using hash function on a concatenated bitstring *)
if I4'=I4 then (* Check if I4 matches I4' *)
let I5=h(con1(con(GIDj,L,Sbeta),I3')) in (* Calculate I5 using hash function on a concatenated bitstring *)
let I6= h(con1(con(I5,T3,D),B)) in (* Calculate I6 using hash function on a concatenated bitstring *)
out(ch,(I6,D,T3)); (* Send I6, D, and T3 over the regular channel *)
let SKs=ecpa(ecpm(d,Ubeta),ecpm(B,Salpha)) in (* Calculate SKs using elliptic curve point operations *)
0 (* Return 0, end of the Sensor process *)
).
```

A new bitstring d is generated.
D is computed as the result of elliptic curve point multiplication (ecpm) with inputs d and P.A5 is calculated as the hash of the concatenation of con3(SIDj, Gbeta).
A6 is obtained by hashing the concatenation of A5, D, and L. The tuple (A5, A6, D) is sent through the private channel sch2.
The process waits to receive a tuple (A7: bitstring, A8: bitstring) on channel sch2.A8' is computed as the hash of con1(con(A7, A6, D), L).
If A8 is equal to A8', then the process continues with the rest of the code inside the if block.

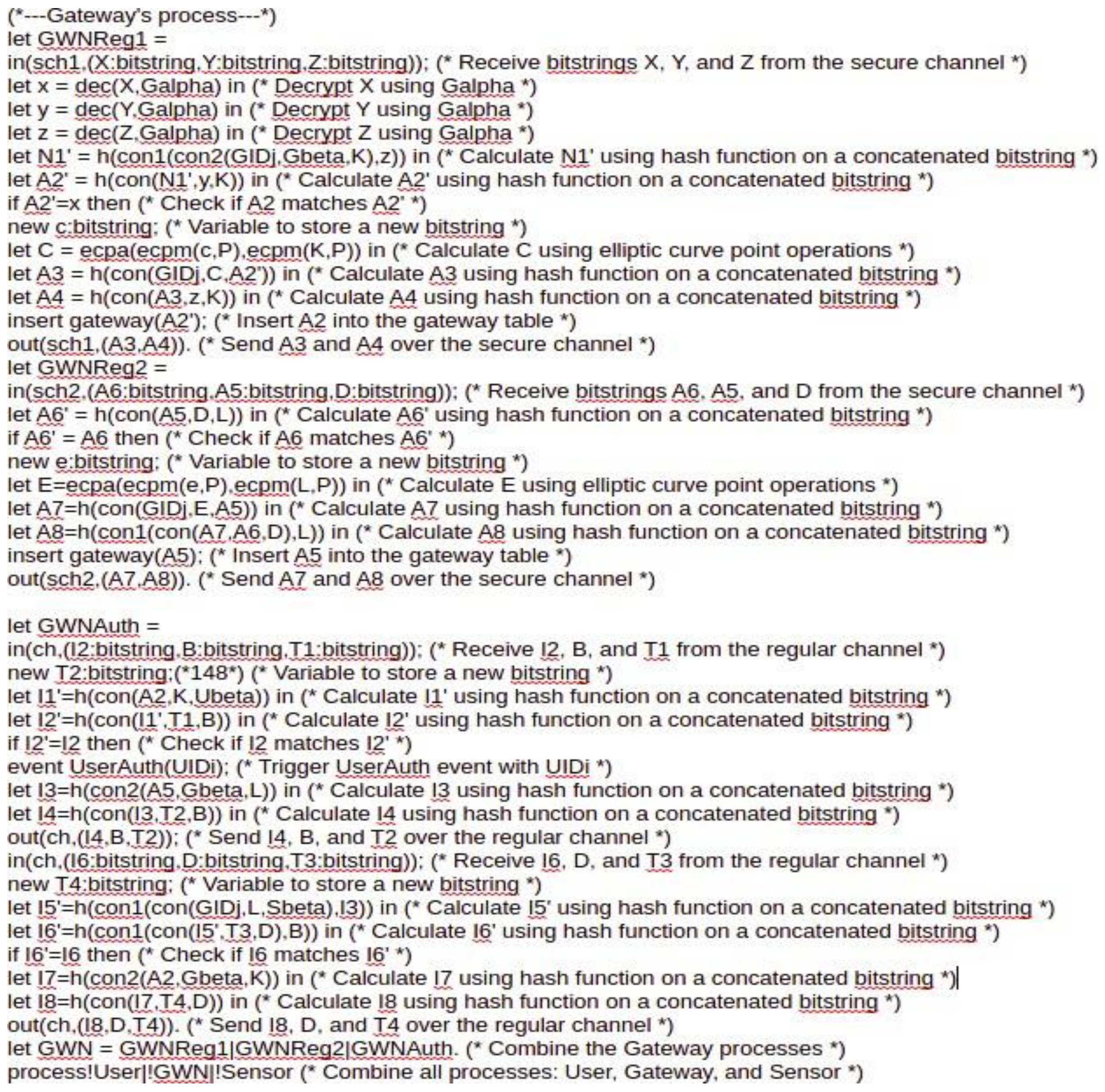
Inside the if block (denoted by (*110*)):

Salpha is calculated as the hash of con(d, A5, A7).
Sbeta is obtained by performing elliptic curve point multiplication (ecpm) with inputs Salpha ,P

The sensor table is updated with the value A5.
The process waits to receive a tuple (I4: bitstring, B: bitstring, T2: bitstring) on channel ch.A new bitstring T3 is generated.
I3' is computed as the hash of con2(A5, Gbeta, L).
I4' is obtained by hashing the concatenation of I3', T2, and B.
If I4 is equal to I4', then the process continues with the rest of the code inside this nested if block.

Continuing inside the nested if block:
I5 is calculated as the hash of con1(con(GIDj, L, Sbeta), I3'). I6 is obtained by hashing the concatenation of I5, T3, D, and B.The tuple (I6, D, T3) is sent through the channel ch.
A session key SKs is computed using elliptic curve point addition (ecpa) of the results of twoelliptic curve point multiplications (ecpm).
The process returns 0, indicating successful completion.

GATEWAY PROCESS:



```
(*---Gateway's process---*)
let GWNReg1 =
in(sch1,(X:bitstring,Y:bitstring,Z:bitstring)); (* Receive bitstrings X, Y, and Z from the secure channel *)
let x = dec(X,Galpha) in (* Decrypt X using Galpha *)
let y = dec(Y,Galpha) in (* Decrypt Y using Galpha *)
let z = dec(Z,Galpha) in (* Decrypt Z using Galpha *)
let N1' = h(con1(con2(GIDj,Gbeta,K),z)) in (* Calculate N1' using hash function on a concatenated bitstring *)
let A2' = h(con(N1',y,K)) in (* Calculate A2' using hash function on a concatenated bitstring *)
if A2'=x then (* Check if A2 matches A2' *)
new c:bitstring; (* Variable to store a new bitstring *)
let C = ecpa(ecpm(c,P),ecpm(K,P)) in (* Calculate C using elliptic curve point operations *)
let A3 = h(con(GIDj,C,A2')) in (* Calculate A3 using hash function on a concatenated bitstring *)
let A4 = h(con(A3,z,K)) in (* Calculate A4 using hash function on a concatenated bitstring *)
insert gateway(A2'); (* Insert A2 into the gateway table *)
out(sch1,(A3,A4)). (* Send A3 and A4 over the secure channel *)
let GWNReg2 =
in(sch2,(A6:bitstring,A5:bitstring,D:bitstring)); (* Receive bitstrings A6, A5, and D from the secure channel *)
let A6' = h(con(A5,D,L)) in (* Calculate A6' using hash function on a concatenated bitstring *)
if A6' = A6 then (* Check if A6 matches A6' *)
new e:bitstring; (* Variable to store a new bitstring *)
let E=ecpa(ecpm(e,P),ecpm(L,P)) in (* Calculate E using elliptic curve point operations *)
let A7=h(con(GIDj,E,A5)) in (* Calculate A7 using hash function on a concatenated bitstring *)
let A8=h(con1(con(A7,A6,D),L)) in (* Calculate A8 using hash function on a concatenated bitstring *)
insert gateway(A5); (* Insert A5 into the gateway table *)
out(sch2,(A7,A8)). (* Send A7 and A8 over the secure channel *)

let GWNAuth =
in(ch,(I2:bitstring,B:bitstring,T1:bitstring)); (* Receive I2, B, and T1 from the regular channel *)
new T2:bitstring;(*148*) (* Variable to store a new bitstring *)
let I1'=h(con(A2,K,Ubeta)) in (* Calculate I1' using hash function on a concatenated bitstring *)
let I2'=h(con(I1',T1,B)) in (* Calculate I2' using hash function on a concatenated bitstring *)
if I2'=I2 then (* Check if I2 matches I2' *)
event UserAuth(UIDi); (* Trigger UserAuth event with UIDi *)
let I3=h(con2(A5,Gbeta,L)) in (* Calculate I3 using hash function on a concatenated bitstring *)
let I4=h(con(I3,T2,B)) in (* Calculate I4 using hash function on a concatenated bitstring *)
out(ch,(I4,B,T2)); (* Send I4, B, and T2 over the regular channel *)
in(ch,(I6:bitstring,D:bitstring,T3:bitstring)); (* Receive I6, D, and T3 from the regular channel *)
new T4:bitstring; (* Variable to store a new bitstring *)
let I5'=h(con1(con(GIDj,L,Sbeta),I3)) in (* Calculate I5' using hash function on a concatenated bitstring *)
let I6'=h(con1(con(I5',T3,D),B)) in (* Calculate I6' using hash function on a concatenated bitstring *)
if I6'=I6 then (* Check if I6 matches I6' *)
let I7=h(con2(A2,Gbeta,K)) in (* Calculate I7 using hash function on a concatenated bitstring *)|
let I8=h(con(I7,T4,D)) in (* Calculate I8 using hash function on a concatenated bitstring *)
out(ch,(I8,D,T4)). (* Send I8, D, and T4 over the regular channel *)
let GWN = GWNReg1|GWNReg2|GWNAuth. (* Combine the Gateway processes *)
process!User|!GWN|!Sensor (* Combine all processes: User, Gateway, and Sensor *)
```

GWNReg1:

The process waits to receive a tuple (X: bitstring, Y: bitstring, Z: bitstring) on channel sch1.
The variables x, y, and z are obtained by decrypting X, Y, and Z, respectively, using the privatekey Galpha.
N1' is computed as the hash of the concatenation of con1(con2(GIDj, Gbeta, K), z).A2' is obtained by hashing the concatenation of N1', y, and K.
If A2' is equal to x, the process continues inside the if block: A new bitstring c is generated.
C is computed as the result of elliptic curve point addition (ecpa) of the results of two ellipticcurve point multiplications (ecpm).
A3 is calculated as the hash of con(GIDj, C, A2').
A4 is obtained by hashing the concatenation of A3, z, and K. The gateway table is updated with the value A2'.
The tuple (A3, A4) is sent through the channel sch1.

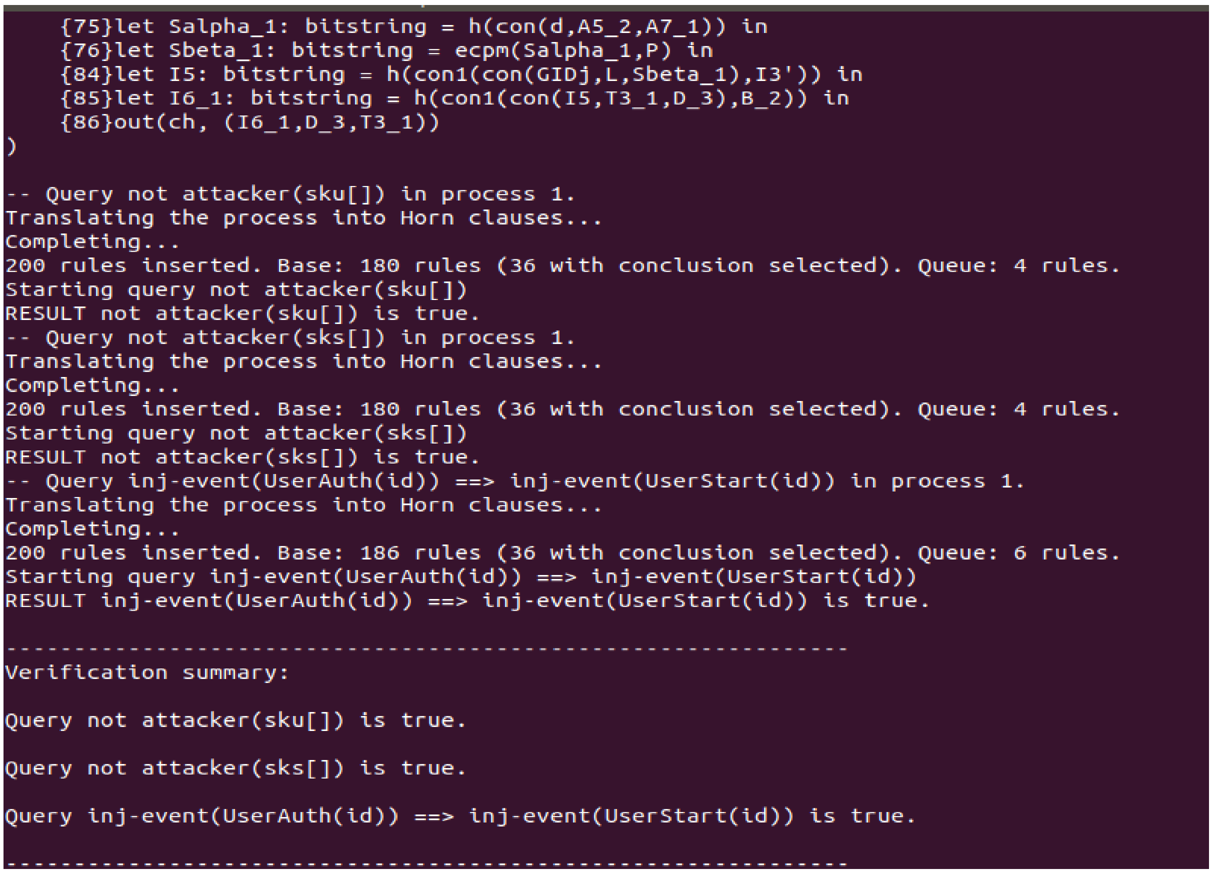
GWNReg2:

The process waits to receive a tuple (A6: bitstring, A5: bitstring, D: bitstring) on channel sch2.A6' is computed as the hash of con(A5, D, L).
If A6' is equal to A6, the process continues inside the if block:A new bitstring e is generated.
E is computed as the result of elliptic curve point addition (ecpa) of the results of two ellipticcurve point multiplications (ecpm).
A7 is calculated as the hash of con(GIDj, E, A5).
A8 is obtained by hashing the concatenation of A7, A6, and D.The gateway table is updated with the value A5.
The tuple (A7, A8) is sent through the channel sch2

GWNAuth:

It defines a process named "GWNAuth," which takes three input channels: "ch," "I2," "B," and"T1." It also creates a new variable "T2" of type "bitstring."
It computes a value "I1'" based on inputs "A2," "K," and "Ubeta," and then computes "I2'" basedon "I1'," "T1," and "B."
It checks if "I2'" is equal to "I2," and if true, emits an event "UserAuth" with the value "UIDi."
It computes a value "I3" based on "A5," "Gbeta," and "L," and then computes "I4" based on "I3,""T2," and "B."
It sends the values "I4," "B," and "T2" through the output channel "ch."
It defines another process named "GWN," which combines processes "GWNReg1,""GWNReg2," and "GWNAuth."
It starts three processes: "User," "GWN," and "Sensor."

b.<u>RESULT:</u>



```
    {75}let Salpha_1: bitstring = h(con(d,A5_2,A7_1)) in
    {76}let Sbeta_1: bitstring = ecpm(Salpha_1,P) in
    {84}let I5: bitstring = h(con1(con(GIDj,L,Sbeta_1),I3')) in
    {85}let I6_1: bitstring = h(con1(con(I5,T3_1,D_3),B_2)) in
    {86}out(ch, (I6_1,D_3,T3_1))
)

-- Query not attacker(sku[]) in process 1.
Translating the process into Horn clauses...
Completing...
200 rules inserted. Base: 180 rules (36 with conclusion selected). Queue: 4 rules.
Starting query not attacker(sku[])
RESULT not attacker(sku[]) is true.
-- Query not attacker(sks[]) in process 1.
Translating the process into Horn clauses...
Completing...
200 rules inserted. Base: 180 rules (36 with conclusion selected). Queue: 4 rules.
Starting query not attacker(sks[])
RESULT not attacker(sks[]) is true.
-- Query inj-event(UserAuth(id)) ==> inj-event(UserStart(id)) in process 1.
Translating the process into Horn clauses...
Completing...
200 rules inserted. Base: 186 rules (36 with conclusion selected). Queue: 6 rules.
Starting query inj-event(UserAuth(id)) ==> inj-event(UserStart(id))
RESULT inj-event(UserAuth(id)) ==> inj-event(UserStart(id)) is true.

--------------------------------------------------------------
Verification summary:

Query not attacker(sku[]) is true.

Query not attacker(sks[]) is true.

Query inj-event(UserAuth(id)) ==> inj-event(UserStart(id)) is true.

--------------------------------------------------------------
```

6. APPLICATIONS

1.FINANCIAL INSTITUTIONS :

- Now a days more activities or transactions are being done online as more and more businesses are going cashless.

- Cybercrimes in digital banking have an impact on both the customers and the banks.

- Protecting the assets like usernames, passwords, credit card numbers etc...of the consumer is the main goal of cybersecurity in digital banking and financial institutions.

- end-consumer suffers losses, card issuers and banks will face the majority of the burden, including refunding the customer, dealing with refund fees/fines, and investigative expenses, which frequentlyresult in reputation loss.

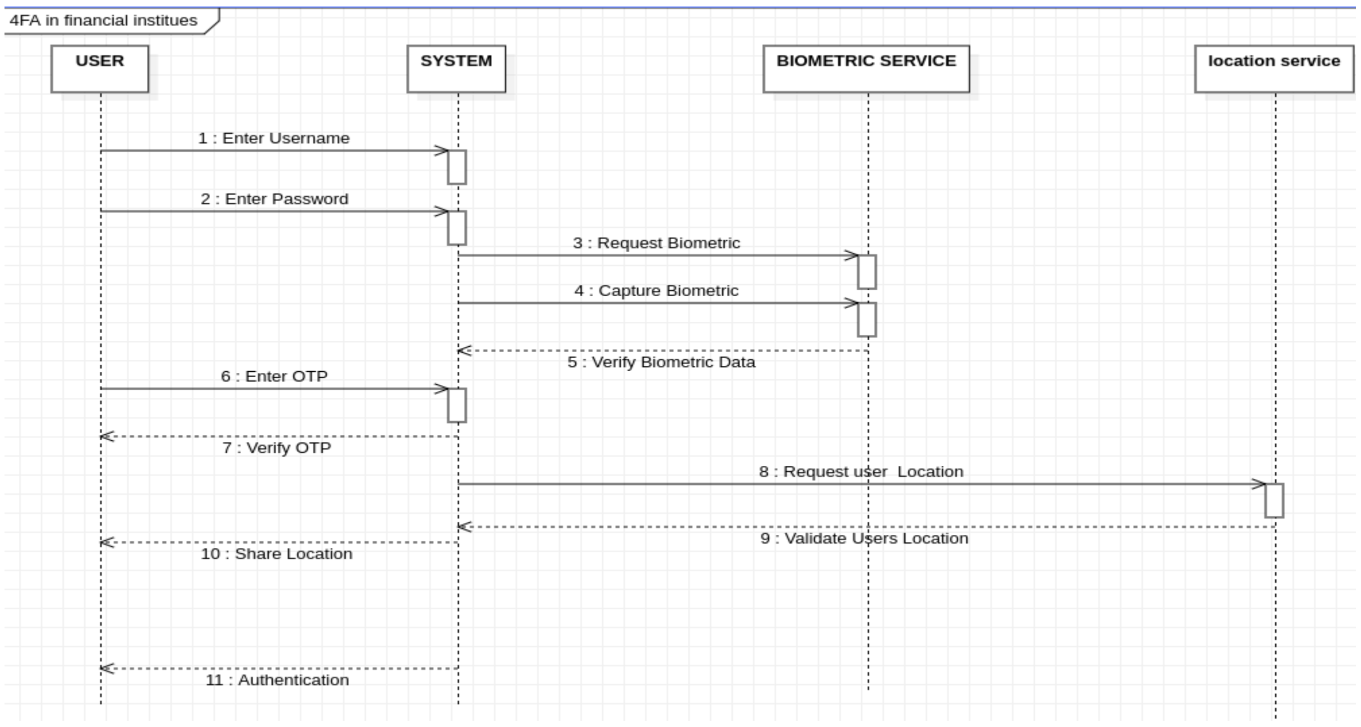
- **Username and Password (Knowledge Factor):** The user provides their username and password to log in, as is the case with traditional authentication methods.

- **One-Time Password (OTP) or Time-based One-Time Password (TOTP) (Possession Factor):** After entering the username and password, the system sends a unique one-time password to the user's registered mobile device via SMS or an authentication app like Google Authenticator. The user must then enter this OTP or TOTP toproceed.

- **Biometric Authentication (Inherence Factor):** For the third factor,the user's biometric data is used for authentication. This could include fingerprint recognition, facial recognition, or even iris scanning. The user needs to provide the required biometric data, andthe system verifies it against the previously enrolled biometric template.

- **Location-Based Authentication (Context Factor):** The fourth factor involves verifying the user's location. The system compares the user's current geographical location (determined through GPS orIP address) with the location history associated with the account. If the location seems inconsistent or suspicious, it could trigger additional verification steps or deny access.

Sequence diagram :



Explanation of the sequence diagram:

a. The user provides their username and password to the system.
b. The system requests biometric data from the Biometric Service.
c The Biometric Service captures and verifies the user's biometric data.
d. The system requests an OTP/TOTP from the user.
e. The user provides the OTP/TOTP to the system, and the systemverifies it.
f. The system requests the user's location from the Location Service

g. The Location Service provides the user's location to the system, and the system validates it.
h. The system performs the authentication process, verifying all four factors (username/password, OTP/TOTP, biometric data, and userlocation).
i. If all factors are successfully authenticated, the system concludes the process with "Authentication Successful."

Flow control for the 4FA in financial institutions :

a. **Input Username and Password:** User provides their username andpassword.
b. **Verify Username and Password:** The system validates the provided username and password.
c. **Username and Password Valid?**
d. **Yes:** Proceed to Step 4.
e. **No:** Show "Authentication Failed" message and end.
f. **Request OTP/TOTP:** If the username and password are valid, the system requests a one-time password (OTP) or time-based one-timepassword (TOTP).
g. **Input OTP/TOTP:** The user receives the OTP/TOTP and enters it into the system.
h. **Verify OTP/TOTP:**
i. **Correct OTP/TOTP?**
j. **Yes:** Proceed to Step 8.
k. **No:** Show "Authentication Failed" message and end.
l. **Capture Biometric Data:** Upon successful OTP/TOTP verification, the system prompts the user to capture their biometric data (e.g., fingerprint, facial scan).
m. **Input Biometric Data:** The user provides the requested biometric data.
n. **Verify Biometric Data:**

o. **Biometric Data Valid?**
    a. **Yes:** Proceed to Step l.
    b. **No:** Show "Authentication Failed" message and end.

p. **Request User's Location:** After successful biometric verification, the system requests the user's current location (e.g.,through GPS or IP address).

q. **Input User's Location:** The user's device shares its location data.

r. **Validate User's Location:**

s. **Location Valid?**
    a. **Yes:** Proceed to Step p
    b. **No:** Show "Authentication Failed" message and end.

t. **Authentication Successful:** If all four factors are successfully authenticated, the user gains access to the financial institution's services.

2.MILITARY ORGANIZATIONS:

In military organizations, security is of paramount importance due to the sensitive nature of the information and operations they handle. Four-factor authentication (4FA) is one of the advanced security measures that can be applied to ensure access to critical systems, equipment, and data is highly restricted and safeguarded.

The four factors typically used in military 4FA include:
1. Knowledge Factor: This involves a traditional username and password combination. Military personnel are provided with unique credentials, including usernames and strong passwords, to access various systems.

2. Possession Factor: Military personnel may be issued physical tokens or smart cards, also known as Common Access Cards (CAC) or Personal Identity Verification (PIV) cards. These cards contain embedded cryptographic keys and certificates that are used for authentication.

3. Inherence Factor: Biometric authentication is a crucial component in military 4FA. Biometric data,such as fingerprints, retina scans, or facial recognition, is used to confirm the identity of the individual.

4. Location Factor: The military may incorporate geolocation tracking as part of 4FA. This factor ensures that access to certain systems or sensitive information is only granted when the user is physically located in an authorized area or within a predefined geographic region.

3. AGRICULTURE:

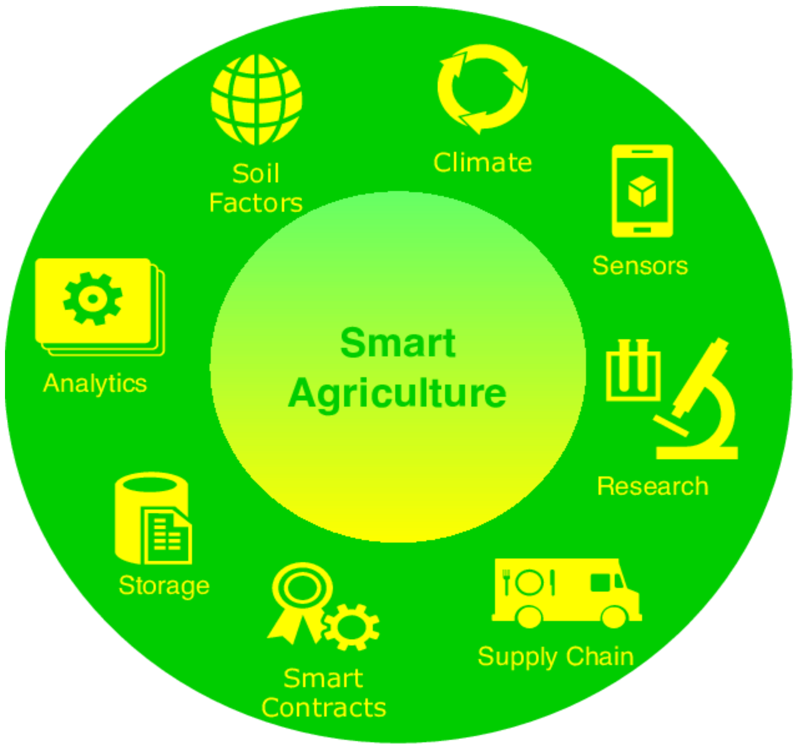
Agricultural data management involves collecting, storing, processing, and analyzing vast amounts of data that can influence farming practices. This data may come from a myriad of sources, including IoT devices, sensors, satellite imagery, drones, and manual inputs. Due to the increased reliance on data-driven practices and the importance of data in making critical farm decisions, securing this data is paramount. Implementing Four-Factor Authentication (4FA) can significantly heighten the security.

The data that is to be secured is :
1.Soil sensors providing data about moisture levels, pH, and nutrient content.
2.Weather stations tracking temperature, rainfall, humidity, and other conditions.
3.IoT devices monitoring water usage or greenhouse conditions.
4.Satellite imagery of fields to assess crop health, predict yields, or monitor for pests and diseases.
5.Drone-captured images and videos for field assessments.
6. Types of crops, planting and harvest dates, yield data, seed variety information, and crop rotation plans.

Here's a step-by-step breakdown of how 4FA might be integrated into agricultural data management systems:
Initial Setup and Data Collection:
Deploy sensors, IoT devices, drones, and other data-collecting tools across the farm.
Ensure the devices have encrypted communication protocols to securely transmit data to the central database.

Data Storage:
Store data in encrypted databases, preferably with regular backup protocols. Access to these databases should be restricted. Only certain authenticated users should be able to read or modify the data.
Implement real-time monitoring to detect any unauthorized or suspicious activity. Users should be immediately alerted on any suspicious login attempts, especially from unrecognized locations or devices.

Data Analysis & Decision Making:
Only authenticated users, having passed the 4FA, should be able to analyze the data or use farm management software tools.
Implementing role-based access can ensure that users only access the information they need, further enhancing security.
The integration of 4FA in agricultural data management not only heightens security but also fosters trust among stakeholders. It reassures farmers and partners that the data, which is crucial for farm productivity and profitability, is well-guarded against threats.



CONCLUSION & FUTURE SCOPE:

In an era of sophisticated cyber threats, single or two-factor authentication is inadequate due to which major areas have opted to implement 4 factor authentication. Few of them are Financial Institutions, Military and Agriculture Data Management. Four-Factor Authentication (4FA) offers a robust solution by combining password/PIN, hardware token, biometrics, and unique metrics like location. In conclusion, the future scope for WBANs in the fields of financial institutions, military organizations and agriculture data management is immense. Finally we studied and applied the implementation of Four Factor Mutual Authentication in these real time applications and analyse the security challenges

REFERENCES:

1. D., & Om, H. (2022). Four-factor mutual authentication scheme for health-care based on wireless body area network. The Journal of Supercomputing, 1-35.
2. Bayat, M., Das, A. K., Pournaghi, M., Far, H. A. N., Fotuhi, M., & Doostari, M. (2020). A lightweight and secure two-factor authentication scheme for wireless body area networks in health-care IoT. Computer Networks: The International Journal of Computer and Telecommunications, 1(1).
3. Li, X., Ibrahim, M. H., Kumari, S., Sangaiah, A. K., Gupta, V., & Choo, K. K. R. (2017). Anonymous mutual authentication and key agreement scheme for wearable sensors in wireless body area networks. Computer Networks, 129, 429-443.
4. Gupta, A., Tripathi, M., & Sharma, A. (2020). A provably secure and efficient anonymous mutual authentication and key agreement protocol for wearable devices in WBAN. Computer Communications, 160, 311-325
5. Kumar, P., Lee, S. G., & Lee, H. J. (2012). E-SAP: Efficient-strong authentication protocol for healthcare applications using wireless medical sensor networks. Sensors, 12(2), 1625-1647.
6. He, D., Kumar, N., Chen, J., Lee, C. C., Chilamkurti, N., & Yeo, S. S. (2015). Robust anonymous authentication protocol for health-care applications using wireless medical sensor networks. Multimedia Systems, 21, 49-60.