Indian Institute of Technology Jodhpur

CSL6010 – Cyber Security
LAB 7 Report

Name- Aman Srivastava
Roll No.- B20CS100

Date-6th Apr 2023

1-Specifying protocol and properties:

In this protocol, there are three entities involved: A, B, and a Trusted Third Party T. The objective is for A to create a shared symmetric session key (AKB) with B.

The protocol for establishing a symmetric session key involves three entities:A, B, and Trusted Third Party T. Here's how it works:

- A sends a request message to T, encrypted with ANT, asking for a session key with B.
- T decrypts A's message using ANT and sends a new nonce to A, encrypted with ANT.
- A encrypts the nonce received from T using the shared key AKB and sends it back to T, encrypted with ANT.
- T decrypts the message from A using ANT and forwards the encrypted nonce to B, encrypted with BAT.
- B decrypts the message from T using BAT and encrypts the nonce with the shared key AKB and sends it back to T, encrypted with BAT.
- T decrypts the message from B using BAT and forwards the encrypted nonce to A, encrypted with ANT.
- A decrypts the message from T using ANT and responds to B with a message encrypted with AKB to confirm the establishment of the session key.

Here are the properties of the protocol:

Secrecy: The shared session key AKB is not accessible to any attacker or eavesdropper, ensuring confidentiality.

Authenticity: Each agent involved in the protocol can verify the identity of the other agents, ensuring that they are communicating with the intended parties.

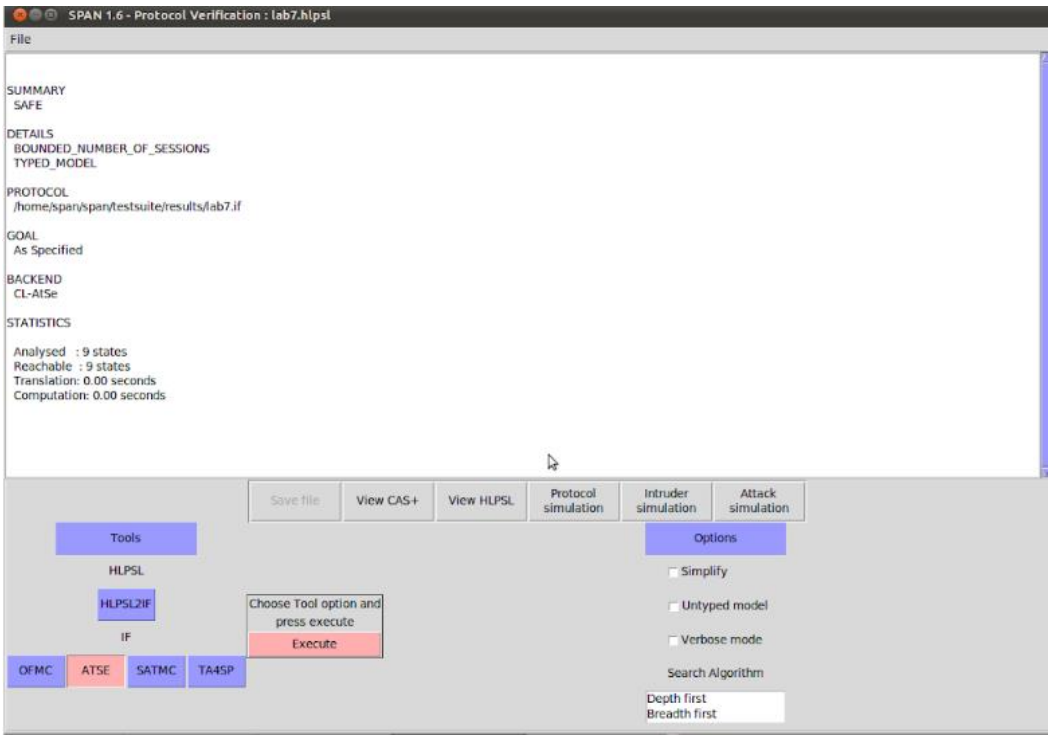
Freshness: To prevent replay attacks, the nonces exchanged between the agents are recent, ensuring that they cannot be reused.

CODE:

```
role role_A(A:agent,B:agent,T:agent,ANT:symmetric_key,SND,RCV:channel(dy))
played_by A
def=
        local
                State:nat,
                AKB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV(start) =|> State':=1 /\ AKB':=new() /\
SND({A.B.AKB'}_ANT) /\ secret(AKB',sec_1,{A,B})
end role

role role_T(T:agent,A:agent,B:agent,ANT,BAT:symmetric_key,SND,RCV:channel(dy))
played_by T
def=
        local
                State:nat,
                AKB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV({A.B.AKB'}_ANT) =|> State':=1
        /\        SND({B.A.AKB'}_BAT)
end role


role role_B(B:agent,A:agent,T:agent,BAT:symmetric_key,SND,RCV:channel(dy))
played_by B
def=
        local
                State:nat,
                AKB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV({B.A.AKB'}_BAT) =|> State':=1
end role


role session(A:agent,B:agent,T:agent,ANT,BAT:symmetric_key)
def=
        local
                SND3,RCV3,SND2,RCV2,SND1,RCV1:channel(dy)
        composition
                role_A(A,B,T,ANT,SND1,RCV1) /\
                role_B(B,A,T,BAT,SND2,RCV2) /\
                role_T(T,A,B,ANT,BAT,SND3,RCV3)
end role
```

```
role environment()
def=
        const
                kat,kbt,kit:symmetric_key,      %% we add a symmetric key: kit shared
between the intruder and T
                 alice,bob,trusted:agent,
                sec_1,auth_1:protocol_id
        intruder_knowledge = {alice,bob,kit}    %% ... and we give it to the intruder
        composition
                session(alice,bob,trusted,kat,kbt)
                /\ session(alice,bob,trusted,kat,kbt)
                /\ session(i,bob,trusted,kit,kbt)

                /\ session(alice,i,trusted,kat,kit)
end role
goal
      secrecy_of sec_1
end goal
environment()
```

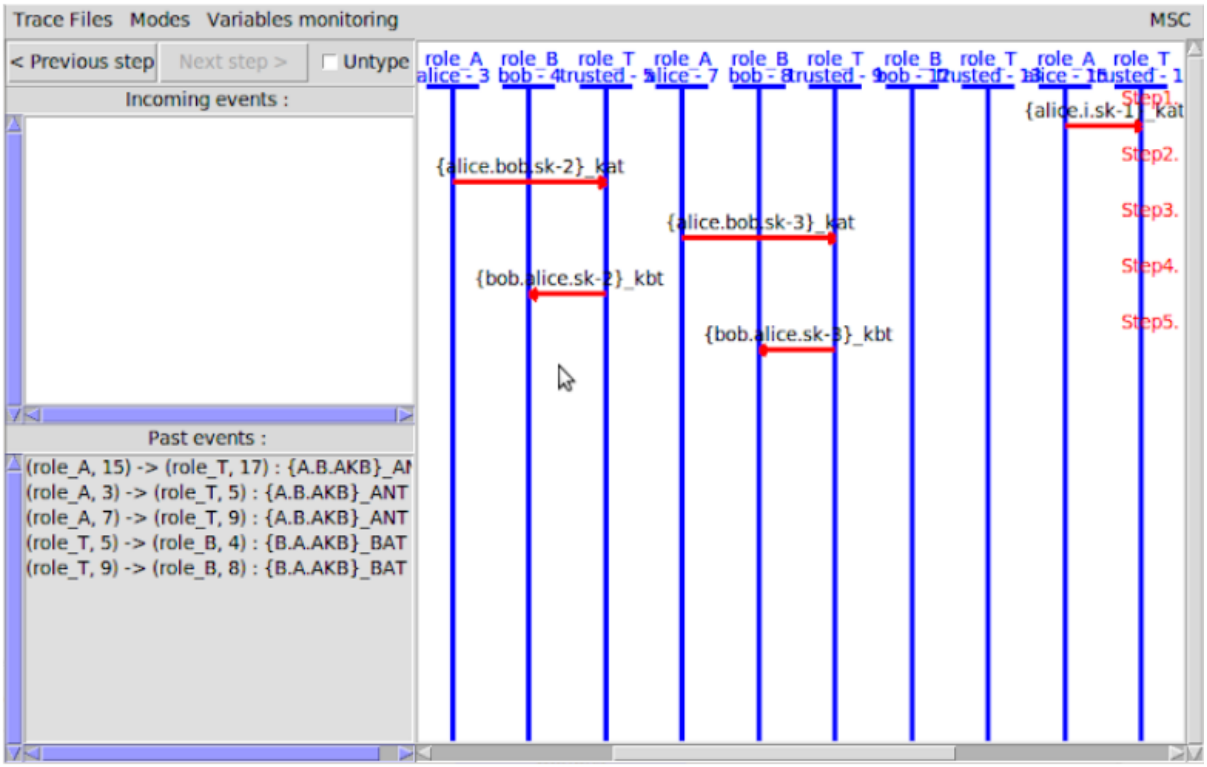
ScreenShots:



2- Debugging specification using animation: Find the blocking transition, monitor the variables:

To ensure that the protocol functions properly and satisfies the mentioned properties, it can be animated using SPAN+AVISPA. This animation can help to identify any upcoming transitions and monitor the variables involved in the protocol. During the animation, the variables

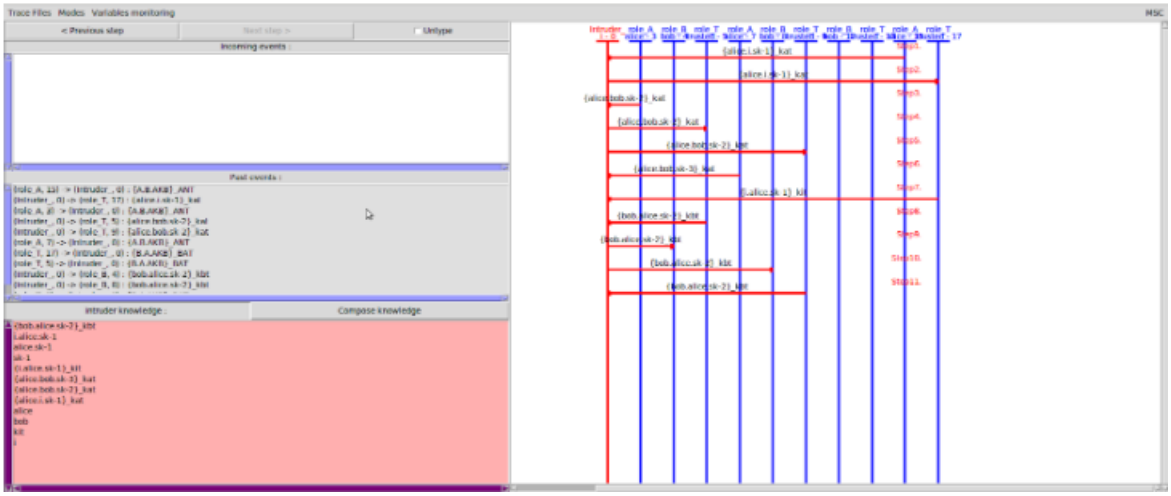
exchanged between the agents can be monitored for any discrepancies, and the protocol's adherence to the listed characteristics can also be verified.



3 - Attack discovery, strengthening the protocol

During the animation, various attacks on the protocol, such as replay attacks, man-in-the-middle attacks, or key compromise attacks, may be identified. To prevent these attacks, the protocol can be strengthened by adding extra messages and checks. For example, digital signatures can be used to ensure authenticity, timestamps can be included to ensure the freshness of nonces, and a message can be included to verify the fidelity of the session key. These measures can help to prevent potential attacks and ensure the security and reliability of the protocol.

ScreenShot:



4- Tuning and optimizing the protocol

Once the protocol has been strengthened to prevent potential attacks, it can be optimized to reduce the number of messages exchanged between the agents or the computational load of the protocol. For example, some messages can be combined or eliminated, or the key size can be reduced to improve efficiency. Formal proof techniques can also be utilized to demonstrate the correctness of the protocol. These optimization measures can improve the performance and efficiency of the protocol while maintaining its security.

CODE:

```
role role_A(A:agent,B:agent,T:agent,ANT:symmetric_key,SND,RCV:channel(dy))
played_by A
def=
        local
                State:nat,
           Na,Nb:text,
           KAB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV(start) =|>
             State':=1  /\  Na':=new()  /\  AKB':=new()  /\  SND({B.AKB'}_ANT)  /\
secret(AKB',sec_1,{A,B,T})

                2. State=1 /\ RCV({B.Nb'}_AKB) =|> State':=2  /\ SND({Nb'}_AKB)
             /\ request(A,B,auth_1,AKB)
             /\ witness(A,B,auth_2,Nb')
end role

role role_T(T:agent,A:agent,B:agent,ANT,BAT:symmetric_key,SND,RCV:channel(dy))
played_by T
```

```
def=
        local
                State:nat,Na:text,AKB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV({B.AKB'}_ANT) =|>

            State':=1 /\ SND({A.AKB'}_BAT)
end role


role role_B(B:agent,A:agent,T:agent,BAT:symmetric_key,SND,RCV:channel(dy))
played_by B
def=
        local
                State:nat,Na,Nb:text,AKB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV({A.AKB'}_BAT) =|>

            State':=1 /\ Nb':= new() /\ SND({B.Nb'}_AKB')

            %% B hopes that Kab will permit to authenticate him
            /\ witness(B,A,auth_1,AKB')

           2. State=1 /\ RCV({Nb}_AKB) =|> State':=2
           /\ request(B,A,auth_2,Nb)
end role



role session(A:agent,B:agent,T:agent,ANT,BAT:symmetric_key)
def=
        local
                SND3,RCV3,SND2,RCV2,SND1,RCV1:channel(dy)
        composition
           role_A(A,B,T,ANT,SND1,RCV1) /\
                role_B(B,A,T,BAT,SND2,RCV2) /\
           role_T(T,A,B,ANT,BAT,SND3,RCV3)
end role

role environment()
def=
        const
                kat,kbt,kit:symmetric_key,      %% we add a symmetric key: kit shared between
the intruder and T
           alice,bob,trusted:agent,
           sec_1,auth_1,auth_2:protocol_id
```

```
        intruder_knowledge = {alice,bob,kit}     %% ... and we give it to the intruder
        composition
                %% We run the regular session
                  session(alice,bob,trusted,kat,kbt)
                %% in parallel with another regular session
              /\ session(alice,bob,trusted,kat,kbt)

                %% and a session between the intruder (with key kit) and bob
              /\ session(i,bob,trusted,kit,kbt)
                %% and a session between alice and the intruder (with key kit)
              /\ session(alice,i,trusted,kat,kit)
end role

goal
        secrecy_of sec_1
      authentication_on auth_1
      authentication_on auth_2
end goal

environment()
```