Cybersecurity Lab 06

B21CS098 (Ritik Tiwari)

Define Three agents: A, B, and a Trusted Third-Party T Were,

● A and T share a symmetric key A*T

● B and T share a symmetric key B*T

● A want to establish a symmetric session key A*B shared with B

Here, in keys replace * with the third letter of your first name. For example, if your full name is CYBER SECURITY, then your key will be ABT, BBT, ABB. But in my case the * will be T.

1 - Specify 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 is how it works:

• A sends a request message to T, encrypted with ATT, asking for a session key with B.

• T decrypts A's message using ATT and sends a new nonce to A, encrypted with ATT.

• 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 ATT and forwards the encrypted nonce to B, encrypted with BTT.

• B decrypts the message from T using BTT and encrypts the nonce with the shared key AKB and sends it back to T, encrypted with BTT.

• T decrypts the message from B using BTT and forwards the encrypted nonce to A, encrypted with ATT.

• A decrypt the message from T using ATT 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, ATT: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'}_ATT)
/\ secret(AKB', sec_1, {A, B})
end role

role role_T(T:agent, A:agent, B:agent, ATT, BTT: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'}_ATT) =|> State':=1 /\ SND({B.A.AKB'}_BTT)
end role

role role_B(B:agent, A:agent, T:agent, BTT: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'}_BTT) =|> State':=1
end role

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

role environment()
        def=
        const
                kat, kbt, kit: symmetric_key,
                alice, bob, trusted, i: agent,
                sec_1, auth_1: protocol_id
        intruder_knowledge = {alice, bob, kit}
        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()

Screenshot’s:

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /home/span/span/testsuite/results/keyExchange2.if

GOAL
  As Specified
```

GOAL
As Specified

BACKEND
CL-AtSe

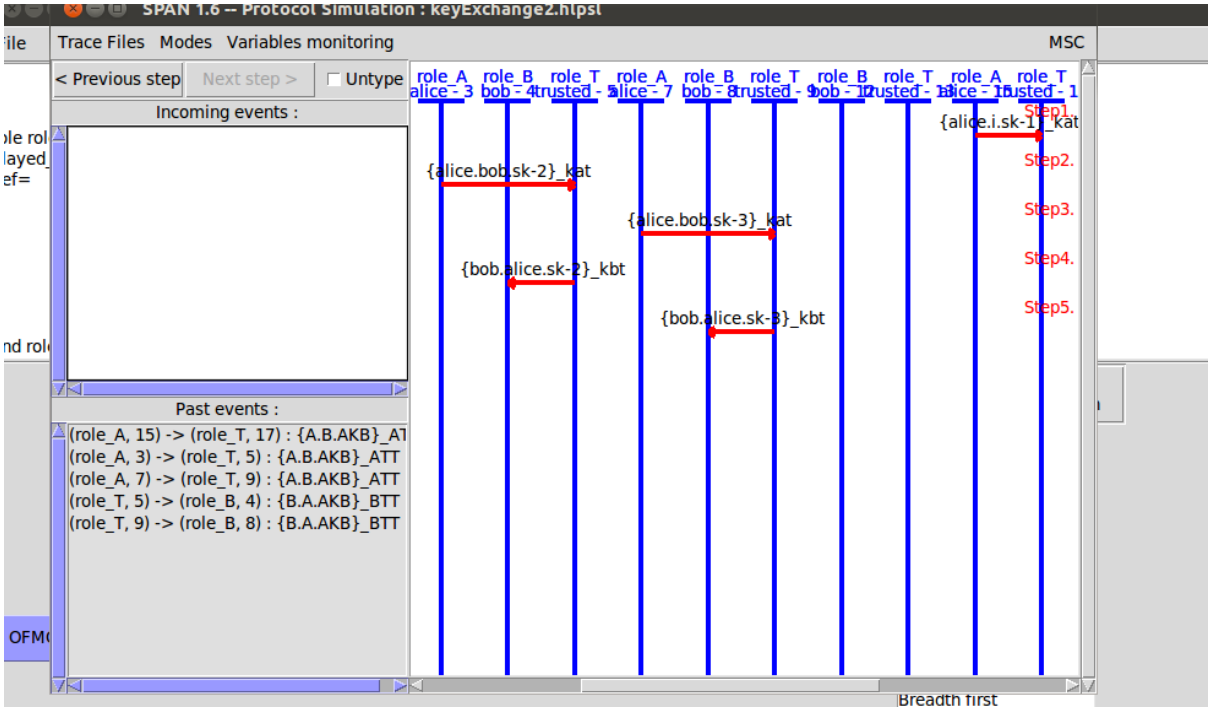
STATISTICS

Analysed : 9 states
Reachable : 9 states
Translation: 0.00 seconds
Computation: 0.00 seconds

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.

Screenshot's:

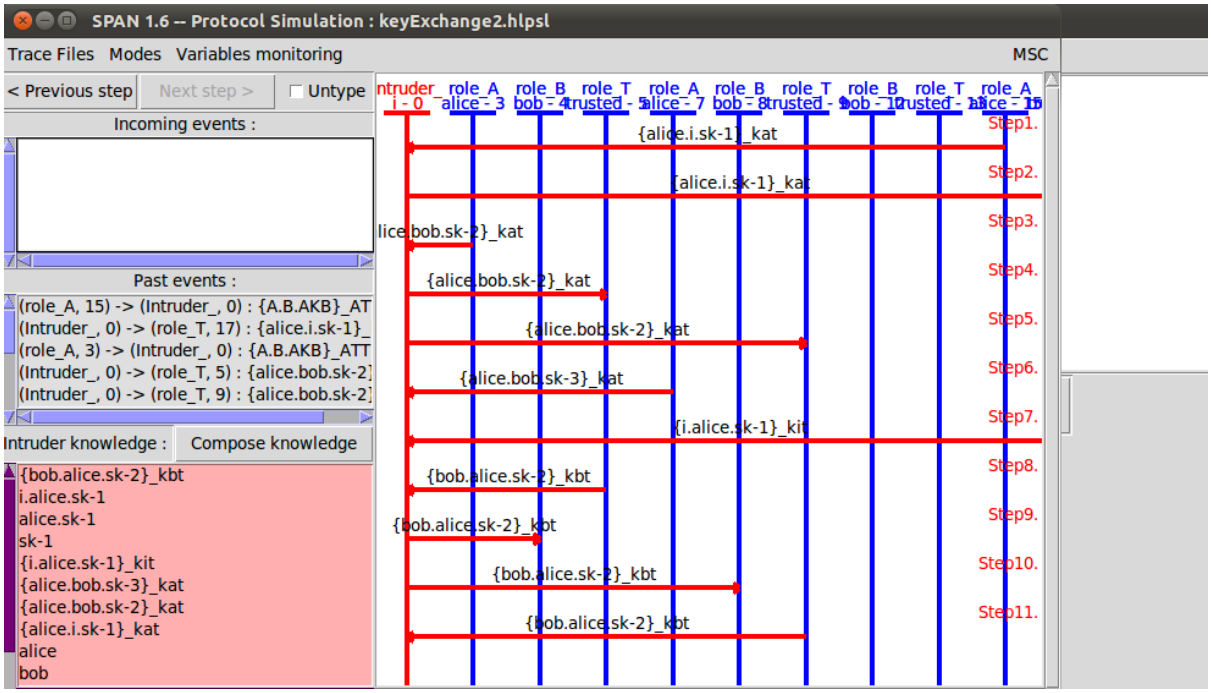


3 - Attack discovery and type, strengthening the protocol

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's:



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,KTT:symmetric_key,SND,RCV:channel(dy))

played_by A

def=

        local

                State:nat,

          Na,Nb:text,

          AKB:symmetric_key
```

```
	init
		State := 0
	transition
		1. State=0 /\ RCV(start) =|>
	    State':=1 /\ Na':=new() /\ AKB':=new() /\ SND({B.AKB'}_KTT) /\
secret(AKB',sec_1,{A,B,T})

		2. State=1 /\ RCV({B.Nb'}_AKB) =|> State':=2  /\ SND({Nb'}_AKB)

	    %% A checks that B uses the same key
	    %% that he sent at step 1.
	    /\ request(A,B,auth_1,AKB)

	    %% A hopes that Nb will permit to authenticate him
	    /\ witness(A,B,auth_2,Nb')
end role

role role_T(T:agent,A:agent,B:agent,KTT,BTT: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'}_KTT) =|>

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

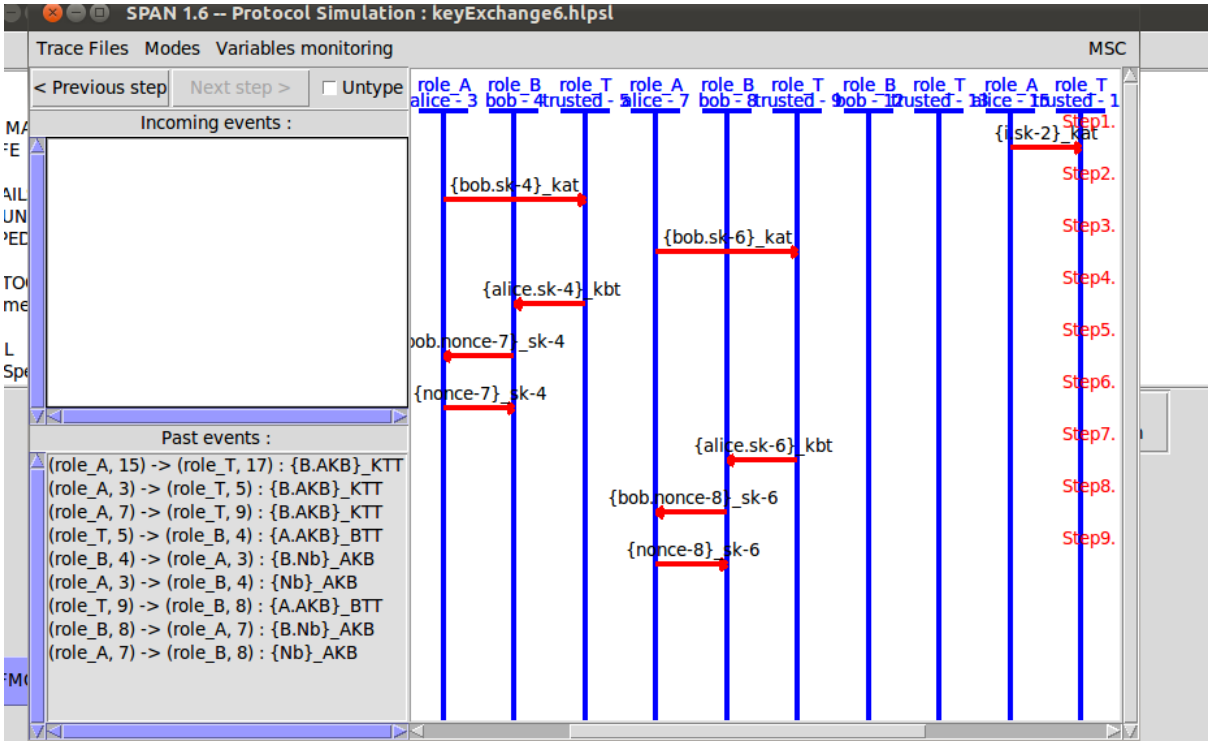
```
role role_B(B:agent,A:agent,T:agent,BTT:symmetric_key,SND,RCV:channel(dy))
played_by B
def=
        local
                State:nat,Na,Nb:text,AKB:symmetric_key
        init
                State := 0
        transition
                1. State=0 /\ RCV({A.AKB'}_BTT) =|>

            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

          %% B checks that he receives the same nonce
          %% that he sent at step 1.

          /\ request(B,A,auth_2,Nb)
end role



role session(A:agent,B:agent,T:agent,KTT,BTT:symmetric_key)
def=
        local
```

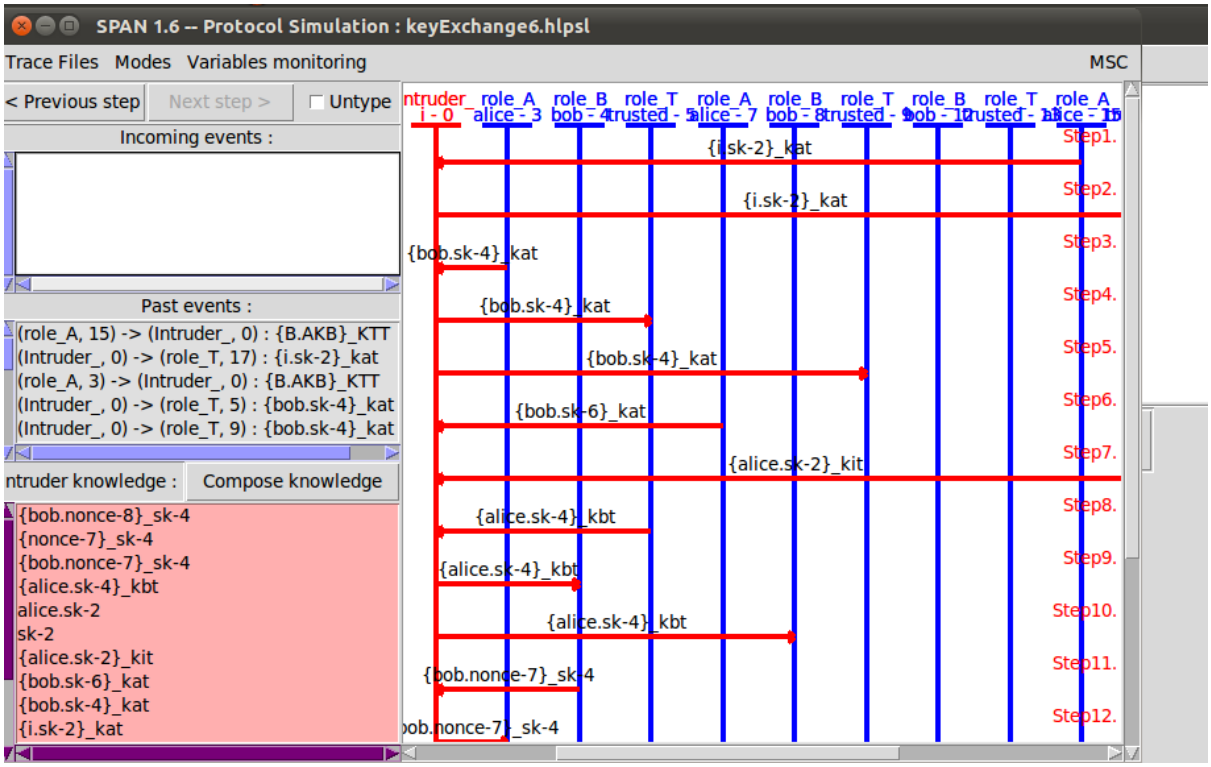
```
                SND3,RCV3,SND2,RCV2,SND1,RCV1:channel(dy)
        composition
          role_A(A,B,T,KTT,SND1,RCV1) /\
                role_B(B,A,T,BTT,SND2,RCV2) /\
          role_T(T,A,B,KTT,BTT,SND3,RCV3)
end role

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()




SPAN 1.6 -- Protocol Simulation : keyExchange6.hlpsl
Trace Files
Modes
Variables monitoring
MSC
< Previous step
Next step >
Untype
Incoming events :
Past events :
(role_A, 15) -> (Intruder_, 0) : {B.AKB}_KTT
(Intruder_, 0) -> (role_T, 17) : {i.sk-2}_kat
(role_A, 3) -> (Intruder_, 0) : {B.AKB}_KTT
(Intruder_, 0) -> (role_T, 5) : {bob.sk-4}_kat
(Intruder_, 0) -> (role_T, 9) : {bob.sk-4}_kat
ntruder knowledge :
Compose knowledge
{bob.nonce-8}_sk-4
{nonce-7}_sk-4
{bob.nonce-7}_sk-4
{alice.sk-4}_kbt
alice.sk-2
sk-2
{alice.sk-2}_kit
{bob.sk-6}_kat
{bob.sk-4}_kat
{i.sk-2}_kat
role_A
role_B
role_T
role_A
role_B
role_T
role_B
role_T
role_A
{i.sk-2}_kat
{i.sk-2}_kat
{bob.sk-4}_kat
{bob.sk-4}_kat
{bob.sk-4}_kat
{bob.sk-6}_kat
{alice.sk-2}_kit
{alice.sk-4}_kbt
{alice.sk-4}_kbt
{alice.sk-4}_kbt
{bob.nonce-7}_sk-4
Step1.
Step2.
Step3.
Step4.
Step5.
Step6.
Step7.
Step8.
Step9.
Step10.
Step11.
Step12.