CYBERSECURITY

LAB ASSIGNMENT-11

Ritik Tiwari

B21CS098

Aim: Automated Verification of an LCDP Security Protocol using SPAN+AVISPA.

LCDP Security Protocol:

The LCDP (Low-Cost Distributed Protocol) Security Protocol is a cryptographic protocol designed to provide security guarantees for communication in distributed systems while being resource-efficient. It aims to address security concerns such as confidentiality, integrity, and authenticity in a distributed computing environment without imposing significant overhead on the system's resources.

Key features of the LCDP Security Protocol may include:

Confidentiality: Ensuring that sensitive information exchanged between distributed entities remains confidential and cannot be accessed by unauthorized parties.

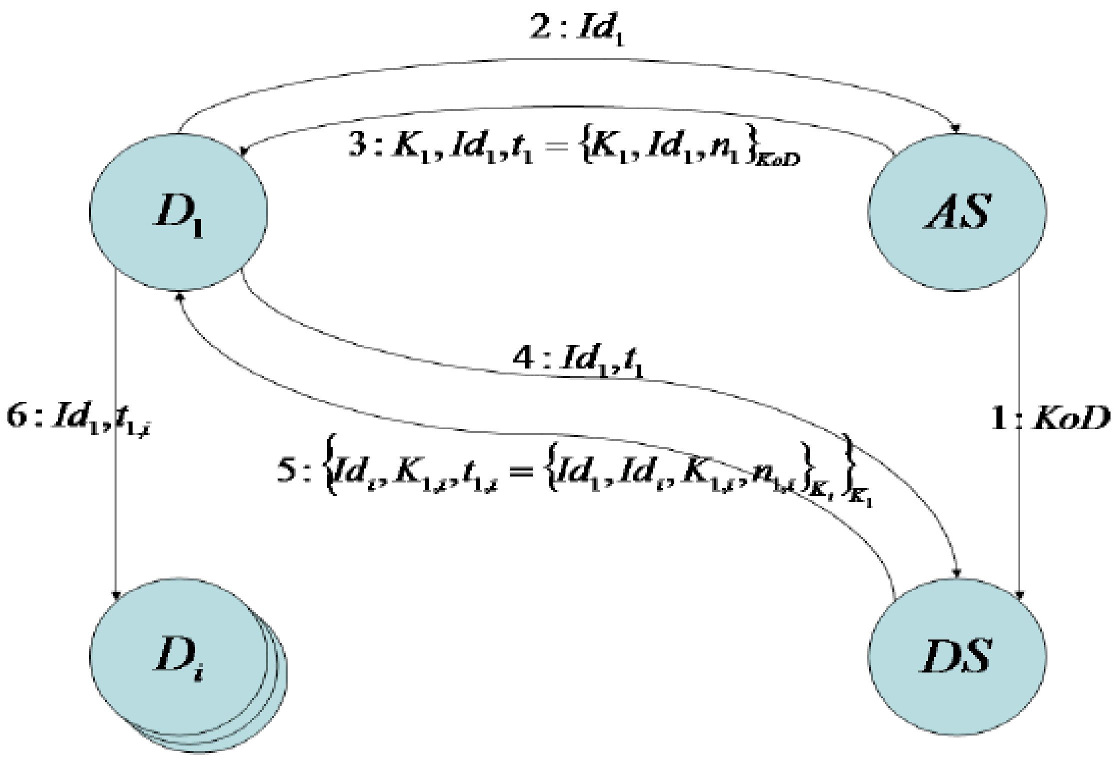
Integrity: Verifying that the data transmitted between nodes remains unchanged and has not been tampered with during transmission.

Authentication: Authenticating the identity of communicating entities to prevent impersonation or unauthorized access.

Efficiency: Minimizing computational and communication overhead to ensure that the protocol is suitable for resource-constrained environments typical of distributed systems.

The LCDP Security Protocol may utilize various cryptographic techniques such as symmetric and asymmetric encryption, digital signatures, and hash functions to achieve its security objectives. It may also incorporate mechanisms for key management and secure session establishment to facilitate secure communication between distributed entities.

Here is the generic diagram of the LCDP Protocol



SPAN+AVISPA:

SPAN (Security Protocol ANalyzer) is a tool designed for the automated analysis and verification of security protocols. It allows users to specify the behavior of cryptographic protocols using a formal language and then automatically analyzes them for security properties.

AVISPA (Automated Validation of Internet Security Protocols and Applications) is a framework for the analysis and validation of security protocols. It includes several tools, including the OFMC (Ottavio's Finite Model Checker) and CL-AtSe (Constraint Logic Analyzer for Security Protocols), which can automatically analyze protocols specified in various formal languages and check them against security properties.

SPAN+AVISPA combines the capabilities of both SPAN and AVISPA, providing a comprehensive platform for the automated analysis and verification of security protocols. It allows users to model security protocols using a high-level language, specify security properties that the protocols should satisfy, and then automatically analyze them for vulnerabilities or security flaws.

By using SPAN+AVISPA, researchers and developers can systematically analyze security protocols, identify potential weaknesses, and refine their designs to ensure robustness and resilience against various security threats. This automated approach can significantly reduce the time and effort required for protocol verification and help improve the overall security of distributed systems.

Here the term “**Automated**” refers to the process of performing tasks or operations with minimal or no human intervention.

In the context of security protocol verification:

Automated Verification: This means using software tools or algorithms to analyze and check the correctness and security properties of a security protocol. Instead of manually inspecting the protocol for flaws or vulnerabilities, automated verification tools can systematically analyze the protocol's behavior and determine whether it meets specified security criteria.

Automated Analysis: This involves using computer algorithms or tools to examine the behavior of the protocol, simulate its execution under different scenarios, and detect potential security weaknesses or vulnerabilities. Automated analysis tools can explore a wide range of possible protocol executions more efficiently than manual inspection, helping to uncover security issues that might be difficult to identify through manual review alone.

LCDP is an symmetric key authentication

LCDP is an adaptation of an existing symmetric key authentication scheme in the case of large communities of devices. The principle is to allow devices to progressively acquire a set of trusted other devices. Once trust is established, devices can securely communicate on a one-to-one basis. To achieve this, LCDP uses a mix of several techniques and components. The components are: one authentication server, at least one directory server and the devices. The techniques are: Public Key Infrastructure (PKI) between devices and authentication server, symmetric cryptography between devices and directory servers, symmetric cryptography again between devices.

The notations used to describe LCDP are: AS for the authentication server, DS for the directory server, D1...Di for devices having identities Id1...Idi Respectively. In a typical session, a device D1 first connects to AS, using its PKI credentials. If D1 is authorized, then AS sends it a cryptographic ticket t1. D1 presents t1 to DS in order to register. DS informs D1 about other devices {Di,...} willing to communicate. For each Di, DS also provides a specific ticket t1,i. Only then may D1 securely communicate with Di by first presenting t1,i.

Screenshot of the Code

```
Role
device1(D1,Di,A,M:agent,Kssl1:symmetric_key,Id1:message,
SND,RCV:channel(dy)) played_by D1 def=
local
State:nat,N1,N2,N:text,Idi,Cred,Ticket:message,K1,K1i:symme
tric_key
init State:=0
transition
1. State=0 /\ RCV(start) =|> State':=1 /\ N':= new()
/\SND({Id1.N'}_Kssl1)
2. State=1 /\ RCV({K1'.Id1.N.Cred'}_Kssl1) =|> State':=2 /\
SND(Id1.Cred')
3. State=2 /\ RCV({Idi'.K1i'.Ticket'}_K1) =|> State':= 3 /\
SND(Id1.Ticket')
4. State=3 /\ RCV({N2'}_K1i) =|> State':= 4
end role
role
devicei(Di,D1:agent,Kssli:symmetric_key,Idi:message,SND,RCV
:channel(dy))
played_by Di def=
local
State:nat,Msg,N2,N:text,Id1,Cred,Tcred:message,Ki,K1i:symme
tric_key
init State:=0
transition
1. State=0 /\ RCV(start) =|> State':=1 /\ N':= new()
/\SND({Idi.N'}_Kssli)
2. State=1 /\ RCV({Ki'.Idi.N.Cred'}_Kssli) =|> State':=2 /\
SND(Idi.Cred')
3. State=2 /\ RCV(Id1'.{Id1'.Idi.K1i'}_Ki) =|> State':=3 /\
```

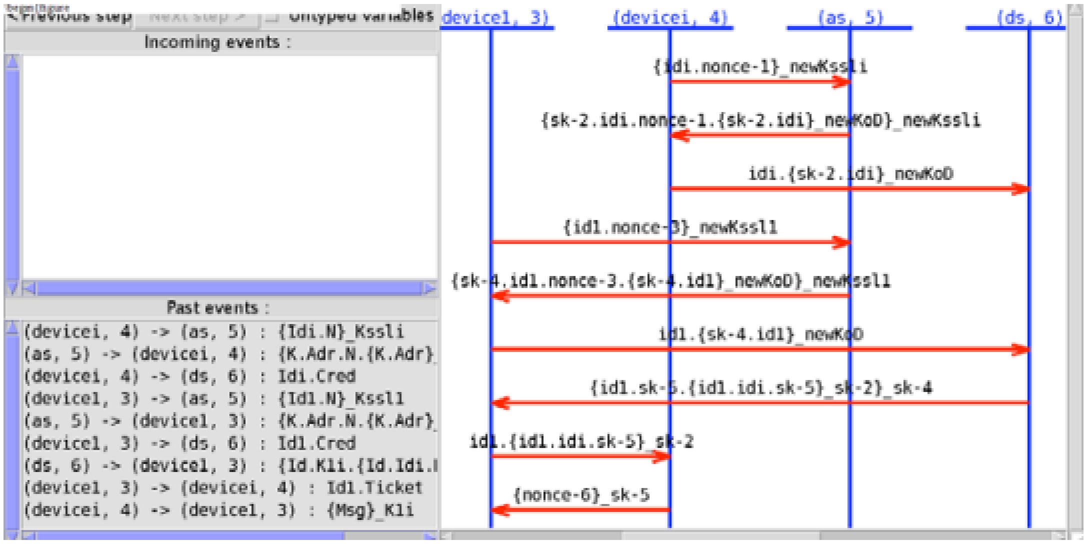
```
Msg':= new() /\
SND({Msg'}_K1i') /\ secret(Msg',secret_msg,{D1,Di})
end role
role as(
A:agent,Kssl1,Kssli,KoD:symmetric_key,SND,RCV:channel(dy))
played_by A def=
local State:nat,N:text,K:symmetric_key,Adr:message
init State:=0
transition
1. State=0 /\ RCV({Adr'.N'}_Kssli) =|>
State':=1 /\ K':= new() /\
SND({K'.Adr'.N'.{K'.Adr'}_KoD}_Kssli)
2. State=1 /\ RCV({Adr'.N'}_Kssl1) =|>
State':=2 /\ K':= new() /\
SND({K'.Adr'.N'.{K'.Adr'}_KoD}_Kssl1)
end role
role ds( M:agent,KoD:symmetric_key,SND,RCV:channel(dy))
played_by M def=
local State:nat,Idi,Id:message,K,Ki,K1i:symmetric_key
init State:=0
transition
1. State=0 /\ RCV(Idi'.{K'.Idi'}_KoD) =|> State':=1
2. State=1 /\ RCV(Id'.{Ki'.Id'}_KoD) =|> State':=2 /\
K1i':= new() /\
SND({Id'.K1i'.{Id'.Idi.K1i'}_K}_Ki')
end role


role
session(D1,Di,A,M:agent,Kssl1,Kssli,KoD:symmetric_key,Id1,
Idi:message) def=
local SD1,SDi,SA,SM,RD1,RDi,RC3,RC4,RA,RM:channel(dy)
```

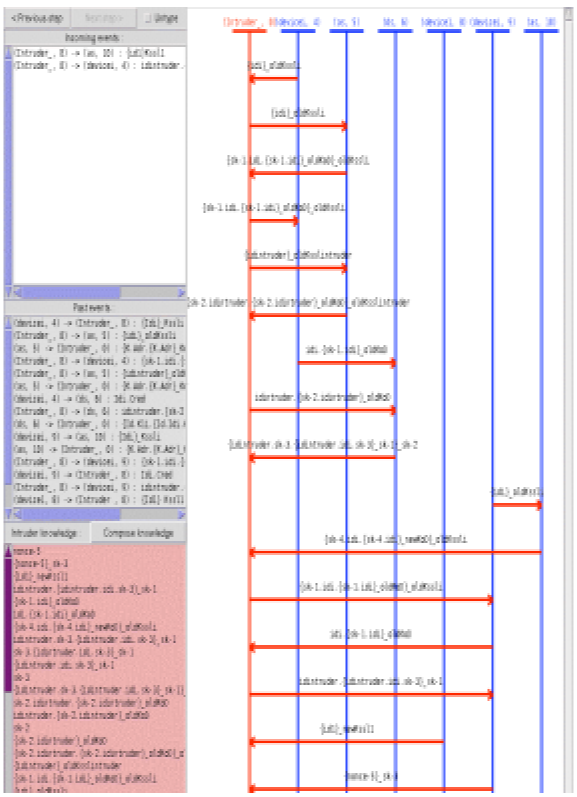
```
composition
device1(D1,Di,A,M,Kssl1,Id1,SD1,RD1) /\
devicei(Di,D1,Kssli,Idi,SDi,RDi)
/\ as(A,Kssl1,Kssli,KoD,SA,RA) /\ ds(M,KoD,SM,RM)
end role
role environment() def=
const d1,di,i,as,ds:agent,
oldKssli,newKssl1,newKssli,oldKsslintruder,oldKoD,newKoD:sy
mmetric_key,
id1,idi,idintruder:message, secret_msg:protocol_id
intruder_knowledge={i,d1,di,as,ds,id1,idi,idintruder,oldKss
lintruder}
composition
session(i,di,as,ds,oldKsslintruder,oldKssli,oldKoD,idintrud
er,idi) /\
session(d1,di,as,ds,newKssl1,newKssli,newKoD,id1,idi)
end role
goal secrecy_of secret_msg end goal
environment()
```

Screenshot of the Simulations

Simple session of LCDP

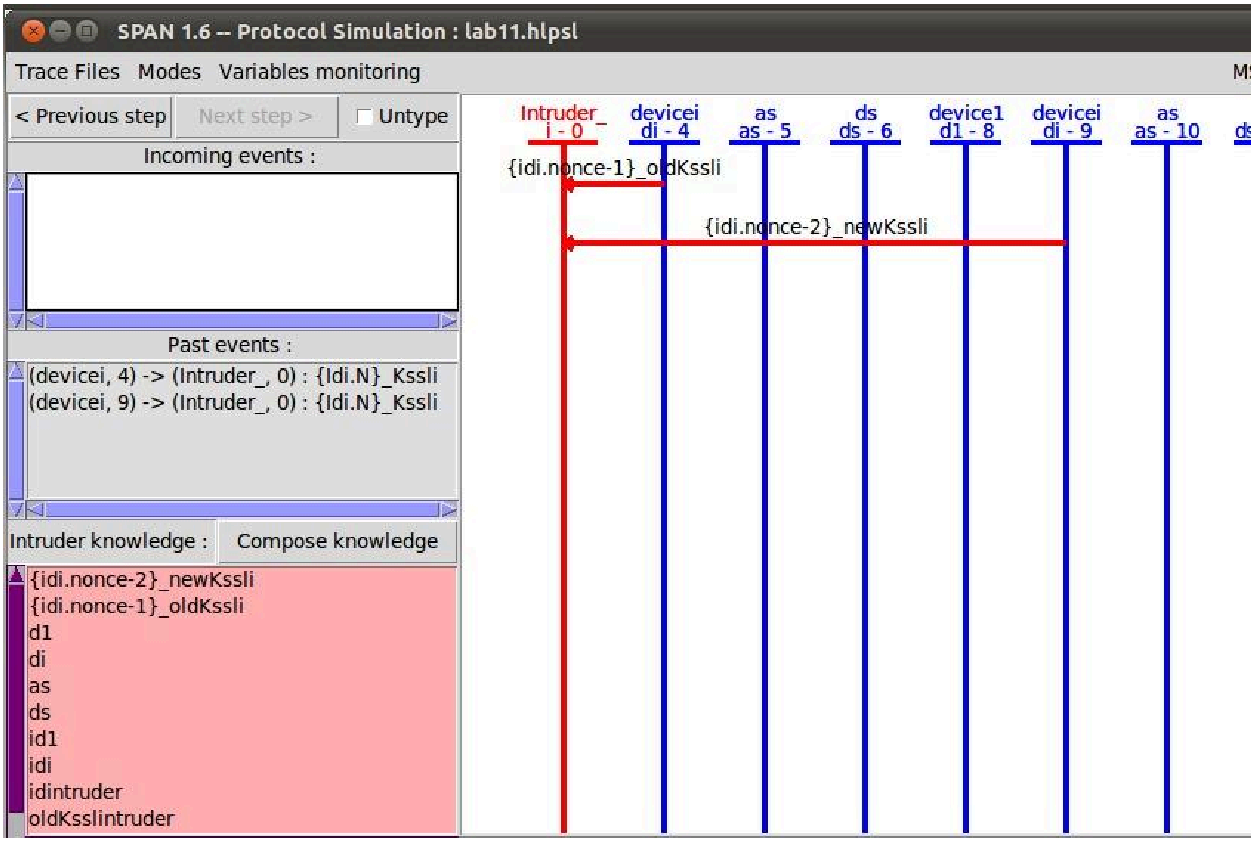


Attack against a slightly downgraded version of LCDP



Two sessions with LCDP

Intruder Simulation



Screenshot of the Results

```
SPAN 1.6 - Protocol Verification : lab11.hlpsl
File

%% IF specification of /home/span/Desktop/lab11.hlpsl

section signature:

state_device1: agent * agent * agent * agent * symmetric_key * message * nat * text * text * text * message * message * message * symmetric_key * symmetric_key * nat -> fact
state_ds: agent * symmetric_key * nat * message * message * symmetric_key * symmetric_key * symmetric_key * nat -> fact
state_as: agent * symmetric_key * symmetric_key * symmetric_key * nat * text * symmetric_key * message * nat -> fact
state_devicei: agent * agent * symmetric_key * message * nat * text * text * text * message * message * message * symmetric_key * symmetric_key * set(agent) * nat -> fact


section types:

secret_msg: protocol_id
State, 11, 10, 9, 4, 3, 2, 8, SID, 1, 0, 6, 5, Dummy_State: nat
set_130, set_106: set
oldKssli, newKssl1, newKssli, oldKsslintruder, oldKoD, newKoD, K, Ki, K1i, KoD, Kssl1, Kssli, Dummy_K1, K1, Dummy_K, Dummy_K1i, Dummy_Ki, dummy_sk: symmetric_key
N, Msg, N2, Dummy_N2, N1, Dummy_Msg, Dummy_N, dummy_nonce: text
d1, di, M, ds, A, as, Di, D1, i: agent
Set_45, Dummy_Set_45, ASGoal: set(agent)
id1, idi, idintruder, Idi, Id, Adr, Id1, Cred, Tcred, Dummy_Ticket, start, Dummy_Id, Dummy_Idi, Dummy_Adr, Ticket, Dummy_Id1, Dummy_Cred, dummy_msg, MGoal: message
```

```
section inits:

initial_state init1 :=
 iknows(start).
 iknows(d1).
 iknows(di).
 iknows(as).
 iknows(ds).
 iknows(id1).
 iknows(idi).
 iknows(idintruder).
 iknows(oldKsslintruder).
 iknows(i).
 state_devicei(di,i,oldKssli,idi,0,dummy_nonce,dummy_nonce,dummy_nonce,dummy_msg,dummy_msg,dummy_msg,dummy_sk,dummy_sk,set_106,4).
 state_as(as,oldKsslintruder,oldKssli,oldKoD,0,dummy_nonce,dummy_sk,dummy_msg,5).
 state_ds(ds,oldKoD,0,dummy_msg,dummy_msg,dummy_sk,dummy_sk,dummy_sk,6).
 state_device1(d1,di,as,ds,newKssl1,id1,0,dummy_nonce,dummy_nonce,dummy_nonce,dummy_msg,dummy_msg,dummy_msg,dummy_sk,dummy_sk,8).
 state_devicei(di,d1,newKssli,idi,0,dummy_nonce,dummy_nonce,dummy_nonce,dummy_msg,dummy_msg,dummy_msg,dummy_sk,dummy_sk,set_130,9).
 state_as(as,newKssl1,newKssli,newKoD,0,dummy_nonce,dummy_sk,dummy_msg,10).
 state_ds(ds,newKoD,0,dummy_msg,dummy_msg,dummy_sk,dummy_sk,dummy_sk,11)
```

```
section rules:

step step_0 (Di,D1,Kssli,Idi,Msg,N2,Dummy_N,Id1,Cred,Tcred,Ki,K1i,Set_45,SID,N) :=
 state_devicei(Di,D1,Kssli,Idi,0,Msg,N2,Dummy_N,Id1,Cred,Tcred,Ki,K1i,Set_45,SID).
 iknows(start)
=[exists N]=>
 state_devicei(Di,D1,Kssli,Idi,1,Msg,N2,N,Id1,Cred,Tcred,Ki,K1i,Set_45,SID).
 iknows(scrypt(Kssli,pair(Idi,N)))

step step_1 (Di,D1,Kssli,Idi,Msg,N2,N,Id1,Dummy_Cred,Tcred,Dummy_Ki,K1i,Set_45,SID,Cred,Ki) :=
 state_devicei(Di,D1,Kssli,Idi,1,Msg,N2,N,Id1,Dummy_Cred,Tcred,Dummy_Ki,K1i,Set_45,SID).
 iknows(scrypt(Kssli,pair(Ki,pair(Idi,pair(N,Cred)))))
 =>
 state_devicei(Di,D1,Kssli,Idi,2,Msg,N2,N,Id1,Cred,Tcred,Ki,K1i,Set_45,SID).
 iknows(pair(Idi,Cred))

step step_2 (Di,D1,Kssli,Idi,Dummy_Msg,N2,N,Dummy_Id1,Cred,Tcred,Ki,Dummy_K1i,Dummy_Set_45,SID,Msg,Id1,K1i) :=
 state_devicei(Di,D1,Kssli,Idi,2,Dummy_Msg,N2,N,Dummy_Id1,Cred,Tcred,Ki,Dummy_K1i,Dummy_Set_45,SID).
 iknows(pair(Id1,scrypt(Ki,pair(Id1,pair(Idi,K1i)))))
=[exists Msg]=>
 state_devicei(Di,D1,Kssli,Idi,3,Msg,N2,N,Id1,Cred,Tcred,Ki,K1i,Dummy_Set_45,SID).
 iknows(scrypt(K1i,Msg)).
 secret(Msg,secret_msg,Dummy_Set_45).
 contains(D1,Dummy_Set_45).
 contains(Di,Dummy_Set_45)
```

```
step step_3 (A,Kssl1,Kssli,KoD,Dummy_N,Dummy_K,Dummy_Adr,SID,N,K,Adr) :=
 state_as(A,Kssl1,Kssli,KoD,0,Dummy_N,Dummy_K,Dummy_Adr,SID).
 iknows(scrypt(Kssli,pair(Adr,N)))
=[exists K]=>
 state_as(A,Kssl1,Kssli,KoD,1,N,K,Adr,SID).
 iknows(scrypt(Kssli,pair(K,pair(Adr,pair(N,scrypt(KoD,pair(K,Adr)))))))

step step_4 (A,Kssl1,Kssli,KoD,Dummy_N,Dummy_K,Dummy_Adr,SID,N,K,Adr) :=
 state_as(A,Kssl1,Kssli,KoD,1,Dummy_N,Dummy_K,Dummy_Adr,SID).
 iknows(scrypt(Kssl1,pair(Adr,N)))
=[exists K]=>
 state_as(A,Kssl1,Kssli,KoD,2,N,K,Adr,SID).
 iknows(scrypt(Kssl1,pair(K,pair(Adr,pair(N,scrypt(KoD,pair(K,Adr)))))))

step step_5 (M,KoD,Dummy_Idi,Id,Dummy_K,Ki,K1i,SID,Idi,K) :=
 state_ds(M,KoD,0,Dummy_Idi,Id,Dummy_K,Ki,K1i,SID).
 iknows(pair(Idi,scrypt(KoD,pair(K,Idi))))
 =>
 state_ds(M,KoD,1,Idi,Id,K,Ki,K1i,SID)

step step_6 (M,KoD,Idi,Dummy_Id,K,Dummy_Ki,Dummy_K1i,SID,Id,Ki,K1i) :=
 state_ds(M,KoD,1,Idi,Dummy_Id,K,Dummy_Ki,Dummy_K1i,SID).
 iknows(pair(Id,scrypt(KoD,pair(Ki,Id))))
=[exists K1i]=>
 state_ds(M,KoD,2,Idi,Id,K,Ki,K1i,SID).
 iknows(scrypt(Ki,pair(Id,pair(K1i,scrypt(K,pair(Id,pair(Idi,K1i)))))))
```

```
step step_7 (D1,Di,A,M,Kssl1,Id1,N1,N2,Dummy_N,Idi,Cred,Ticket,K1,K1i,SID,N) :=
 state_device1(D1,Di,A,M,Kssl1,Id1,0,N1,N2,Dummy_N,Idi,Cred,Ticket,K1,K1i,SID).
 iknows(start)
=[exists N]=>
 state_device1(D1,Di,A,M,Kssl1,Id1,1,N1,N2,N,Idi,Cred,Ticket,K1,K1i,SID).
 iknows(scrypt(Kssl1,pair(Id1,N)))

step step_8 (D1,Di,A,M,Kssl1,Id1,N1,N2,N,Idi,Dummy_Cred,Ticket,Dummy_K1,K1i,SID,Cred,K1) :=
 state_device1(D1,Di,A,M,Kssl1,Id1,1,N1,N2,N,Idi,Dummy_Cred,Ticket,Dummy_K1,K1i,SID).
 iknows(scrypt(Kssl1,pair(K1,pair(Id1,pair(N,Cred)))))
 =>
 state_device1(D1,Di,A,M,Kssl1,Id1,2,N1,N2,N,Idi,Cred,Ticket,K1,K1i,SID).
 iknows(pair(Id1,Cred))

step step_9 (D1,Di,A,M,Kssl1,Id1,N1,N2,N,Dummy_Idi,Cred,Dummy_Ticket,K1,Dummy_K1i,SID,Idi,Ticket,K1i) :=
 state_device1(D1,Di,A,M,Kssl1,Id1,2,N1,N2,N,Dummy_Idi,Cred,Dummy_Ticket,K1,Dummy_K1i,SID).
 iknows(scrypt(K1,pair(Idi,pair(K1i,Ticket))))
 =>
 state_device1(D1,Di,A,M,Kssl1,Id1,3,N1,N2,N,Idi,Cred,Ticket,K1,K1i,SID).
 iknows(pair(Id1,Ticket))
```

```
step step_10 (D1,Di,A,M,Kssl1,Id1,N1,Dummy_N2,N,Idi,Cred,Ticket,K1,K1i,SID,N2) :=
 state_device1(D1,Di,A,M,Kssl1,Id1,3,N1,Dummy_N2,N,Idi,Cred,Ticket,K1,K1i,SID).
 iknows(scrypt(K1i,N2))
 =>
 state_device1(D1,Di,A,M,Kssl1,Id1,4,N1,N2,N,Idi,Cred,Ticket,K1,K1i,SID)


section properties:

property secrecy_of_secret_msg (MGoal,ASGoal) :=
 [] ((secret(MGoal,secret_msg,ASGoal) /\ iknows(MGoal))
    => contains(i,ASGoal))


section attack_states:

attack_state secrecy_of_secret_msg (MGoal,ASGoal) :=
 iknows(MGoal).
 secret(MGoal,secret_msg,ASGoal) &
 not(contains(i,ASGoal))
```

Observations

Both the academic team and the industrial team observe that this experiment has positive results:

1) It brings better confidence in the security of LCDP in simple cases. Neither OFMC nor ATSE found attacks in the test cases. Because ATSE is proven complete this means that, under the simplifying assumptions , there is no attack on a finite number of sessions in the DolevYao model.
2) It provides precise justifications for countermeasures that may otherwise be embedded on a rather prophylactic basis. By just removing some countermeasures, we easily get corresponding attacks.

3) It produces precise specification and execution diagrams. Both are useful for the understanding and further implementation of LCDP. Of course, because of many simplifying assumptions, we are still far away from a complete proof of the protocol. The next step is to progressively relax some of the assumptions, especially the one about the number of directory servers.