[보안 공학 프로젝트 최종 보고서]

학번 : 2018010838
학과 : 컴퓨터과학과
이름 : 공종욱

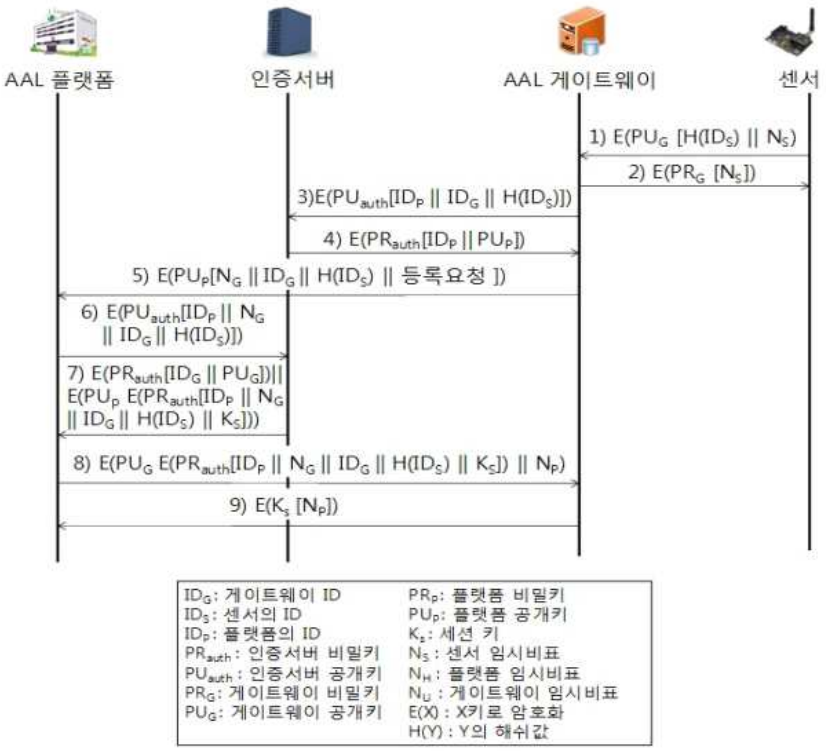
〈 목차 〉

1. 개요

- 검증 대상 프로토콜 소개

논문명 : 전천 후 생활 지원 시스템을 위한 경량 인증 프로토콜 - (이명규, 황보택근)
(A Lightweight Authentication Protocol for Ambient Assisted Living System)

요약 : 향상된 의료 기술과 건강관리 기술의 최근 발전으로 인해 지난 수십 년 동안 기대 수명이 꾸준히 늘게 되었다. 그 결과 세계 인구는 빠르게 고령화되고 있다. 고령자의 복지를 향상시키고 일상생활에서 독립성을 제공하는 해결책을 기반으로 하는 정보통신기술 지원에 대한 다양한 연구가 진행되고 있다. 전천후 보조 생활은 개인의 일상생활 및 근무 환경에서 정보 통신 기술을 사용하여 더 오래 활동성을 유지하고 사회활동을 할 수 있도록 하여 노년기에 독립적으로 살아갈 수 있도록 하는 것으로 정의된다. 전천후 생활 보조 시스템에서 전송되는 정보는 매우 민감한 정보이므로, 이러한 데이터의 보안 및 개인 정보는 중요한 문제로 대두되고 있다. 본 논문에서는 전천후 보조 생활 시스템을 위한 새로운 경량 인증 프로토콜을 제안한다. 제안된 인증 프로토콜은 전천후 생활 지원 시스템에 필요한 몇 가지 중요한 보안 요구 사항을 지원할 뿐만 아니라 다양한 유형의 공격으로부터 안전하다. 또한 보안 분석 결과를 통해 제안된 인증 프로토콜이 기존 프로토콜보다 더 효율적이고 안전하다는 것을 보여준다.



* 제안된 경량 인증 프로토콜

- AAL 시스템 구조

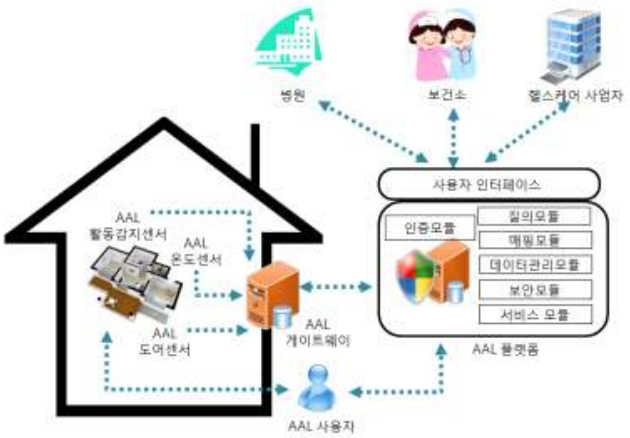
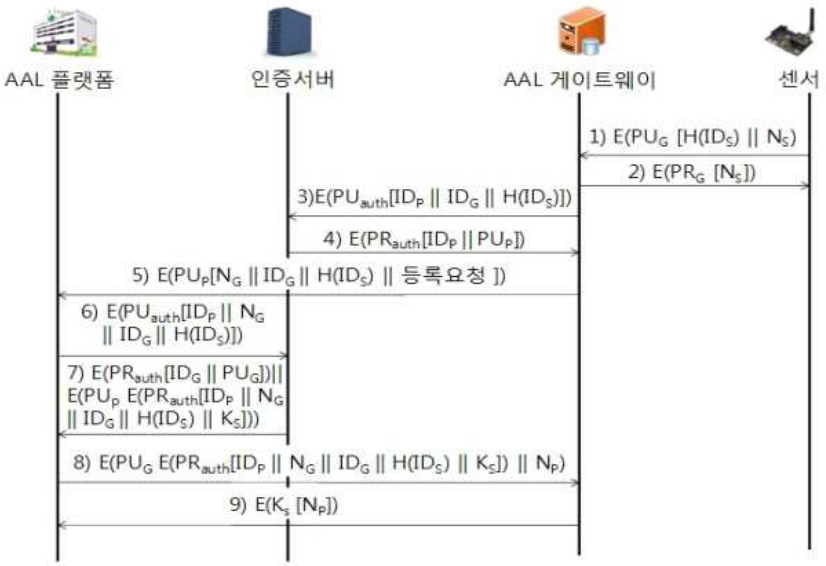


그림 1. AAL 시스템 구성도
Fig. 1. The architecture of the proposed AAL system

AAL 시스템의 구조는 그림 1과 같다. AAL 서비스를 제공하는 대부분의 AAL 시스템은 공통적으로 AAL 센서, AAL 게이트웨이, AAL 서비스를 위한 AAL 플랫폼으로 구성되어 있다. 인증서버의 공개키는 안전한 채널을 통하여 AAL 게이트웨이와 AAL 플랫폼이 공유하고 있다고 가정한다. 또한, AAL 센서의 식별자(IDS)는 AAL 게이트웨이에 미리 등록되어 있으며 AAL 게이트웨이의 공개키는 AAL 센서가 이미 알고 있다고 가정한다. AAL 센서는 사용자의 거주환경으로부터 심전도, 심박수, 호흡 수, 혈압과 같은 사용자의 생체정보와 온도, 습도, 조명과 같은 환경 정보를 수집한다. 다양한 AAL센서로부터 수집된 정보는 AAL 게이트를 통하여 AAL플랫폼으로 전송된다. 기본적으로 AAL 게이트는 사용자의 거주환경 내에 위치하며 인터넷을 통하여 AAL 서비스를 제공하는 AAL 플랫폼과 연결된다. AAL 사용자의 거주환경 내에 위치하는 AAL 센서와 AAL 게이트웨이 간의 데이터 교환은, 위협에 노출되어있는 인터넷을 통하여 데이터 교환이 이루어지는 AAL 게이트웨이와 AAL 플랫폼 구간보다 안전하다고 볼 수 있다.

이러한 특성을 고려하여, 본 논문에서는 AAL 센서와 AAL 게이트웨이 간 인증은 시스템 부하가 적은 임시비표와 해쉬 값을 사용하고, AAL 게이트웨이와 AAL 플랫폼 간의 인증은 해쉬 값, 임시비표, 그리고 인증서버를 통한 공개키 획득을 통해 AAL 시스템 객체 간의 안전한 연결을 설정하고자 한다.

- **제안된 경량 인증 프로토콜**



1. AAL 센서는 자신의 식별자 해쉬 값 H(IDS)과 AAL 센서가 생성한 임시비표 Ns를 AAL 게이트웨이의 공개키로 암호화하여 AAL 게이트웨이에게 전송한다.

2. AAL 게이트웨이는 미리 등록되어 있는 AAL 센서식별자의 해쉬 값을 계산하고, AAL 센서로부터 받은 해쉬 값을 비교해본다. 두 개의 해쉬 값이 동일하면 인증단계를 진행하고, 일치하지 않으면 인가되지 않은 센서로 가정하고 인증단계를 종료한다. AAL 센서와 성공적인 인증이 끝나면 AAL 센서가 생성한 임시비표를 AAL 게이트웨이의 비밀키로 암호화해서 AAL 센서에게 전달해서 인증이 성공적으로 이루어졌음을 알린다.

3) AAL 게이트웨이는 AAL 플랫폼과 안전한 연결을 설정하기 위한 의도를 인증서버에 알리기 위하여, AAL 게이트웨이의 식별자 IDG, AAL 플랫폼의 식별자 IDP, 센서 식별자의 해쉬 값 H(IDS)을 인증서버의 공개키 PUauth 암호화하여 인증서버로 전송한다.

4. 인증서버는 AAL 게이트웨이의 식별자와 센서 식별자의 해쉬 값을 인증서버에 등록한다. 그리고, 인증서버는 AAL 플랫폼의 식별자 IDP와 AAL 플랫폼의 공개키 PUP를 복사한 후 인증서버의 개인키 PRauth로 암호화하여 AAL 게이트웨이에게 반송한다.

5. AAL 게이트웨이는 AAL 플랫폼에게 안전한 연결을 설정하기 위하여 등록요청 메시지를 보낸다. 등록요청 메시지는 AAL 게이트웨이의 식별자와 센서 식별자의 해쉬 값, 그리고 AAL 게이트에서 생성한 임시비표 NG와 함께 AAL 플랫폼의 공개키로 암호화한다.

6. AAL 플랫폼은 인증서버에게 AAL 게이트웨이 공개키와 세션 키 KS를 요청하기 위하여 AAL 플랫폼의 식별자, AAL 게이트웨이 식별자, AAL 센서의 해쉬 값, AAL 게이트웨이의 임시비표를 인증서버의 공개키로 암호화하여 전송한다. AAL 플랫폼은 AAL 게이트웨이에서 발행한 임시비표를 포함함으로써 인증 서버가 AAL 게이트웨이의 임시비표에 세션 키를 서명할 수 있도록 한다.

7. 인증서버는 AAL 플랫폼에게 AAL 게이트웨이의 식별자 IDG와 AAL 게이트웨이의 공개키 PUG를 복사한 후 인증서버의 비밀키로 암호화한다. 또한, AAL 플랫폼 식별자, AAL 게이트웨이의 임시비표, AAL 게이트웨이 식별자, 센서 식별자의 해쉬 값, 세션 키를 인증서버의 비밀키로 암호화한 후 AAL 플랫폼의 공개키로 암호화하여 AAL 플랫폼에 전송한다. 세션 키 Ks는 AAL 플랫폼을 대신하여 인증서버에 의해 만들어지고 AAL 게이트웨이의 비표에 결부된 세션 키이다. 즉, Ks 와 AAL 게이트웨이 비표 NG의 결합은 AAL 게이트웨이에게 세션키 Ks 가 투명하다는 것을 보장한다. 이 세 가지 정보는 인증서버의 비밀키를 이용하여 암호화됨으로 AAL 플랫폼에게 인증서버에 의해서 만들어졌음을 검증한다. 이 정보는 또한 AAL 플랫폼의 공개키를 사용하여 암호화됨으로써 사용자와의 부정한 연결을 설립하는 시도에 이용하지 못하게 한다.

8. AAL 플랫폼은 AAL 게이트웨이에게 세션 키를 전송하기 위해서 AAL 플랫폼 식별자, AAL 게이트웨이의 임시비표, AAL 게이트웨이 식별자, 센서 식별자의 해쉬 값, 세션 키를 인증서버의 비밀키로 암호화한 후 AAL 플랫폼에서 생성된 임시비표 Np와 함께 AAL 게이트웨이의 공개키로 암호화하여 AAL 게이트웨이에 전송한다.

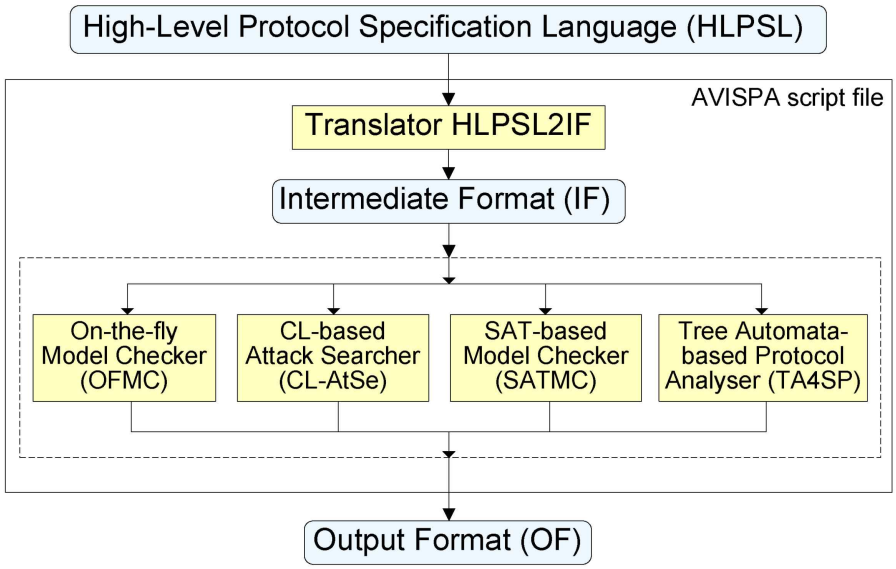
9. AAL 게이트웨이는 세션 키 Ks를 회수하여 AAL플랫폼의 임시비표를 암호화하여 AAL 플랫폼에 반송한다. 이 마지막 메시지는 세션 키에 대한 정보가 안전하다는 것을 보장한다. AAL 플랫폼과 AAL 게이트웨이는 생성된 세션 키를 사용하여 세션기간동안 센서에서 수집된 정보를 전송할 수 있다.

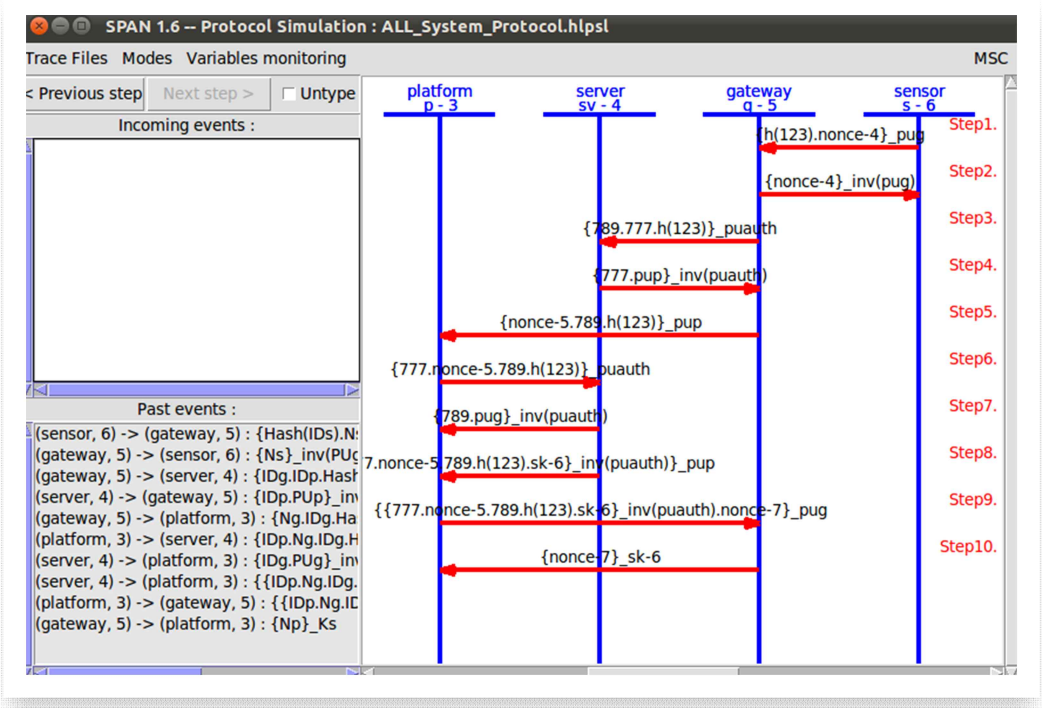
제안된 경량 인증 프토토콜을 통하여 AAL 시스템 간의 객체, 즉 AAL 센서, AAL 게이트웨이, AAL 플랫폼의 인증이 성공적으로 이루어지면 AAL 센서는 AAL 게이트웨이의 공개키로 암호화하여 데이터를 전송하고, AAL 게이트웨이는 세션 키를 이용하여 AAL 플랫폼에게 데이터를 전송할 수 있다.

- 검증에 사용한 Tool

HLPSL를 이용하여 Avispa + SPAN Tool을 사용하여 이번 보안 프로토콜에 대하여 정형검증을 진행하였다.

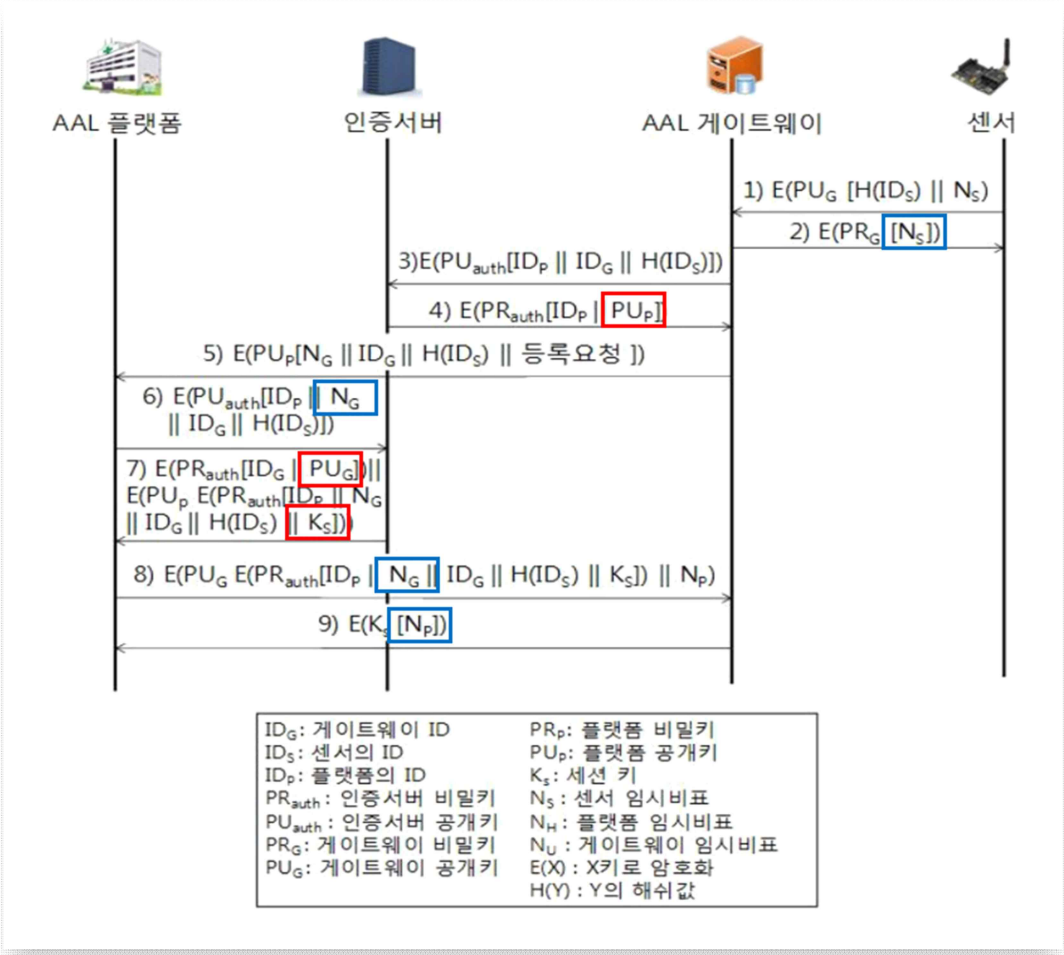
(Avispa 백엔드는 CL-Atse를 사용하여 Simplify Mode로 한번, Verbose 모드로 한번 정형검증을 진행함.)





2. 정형 검증

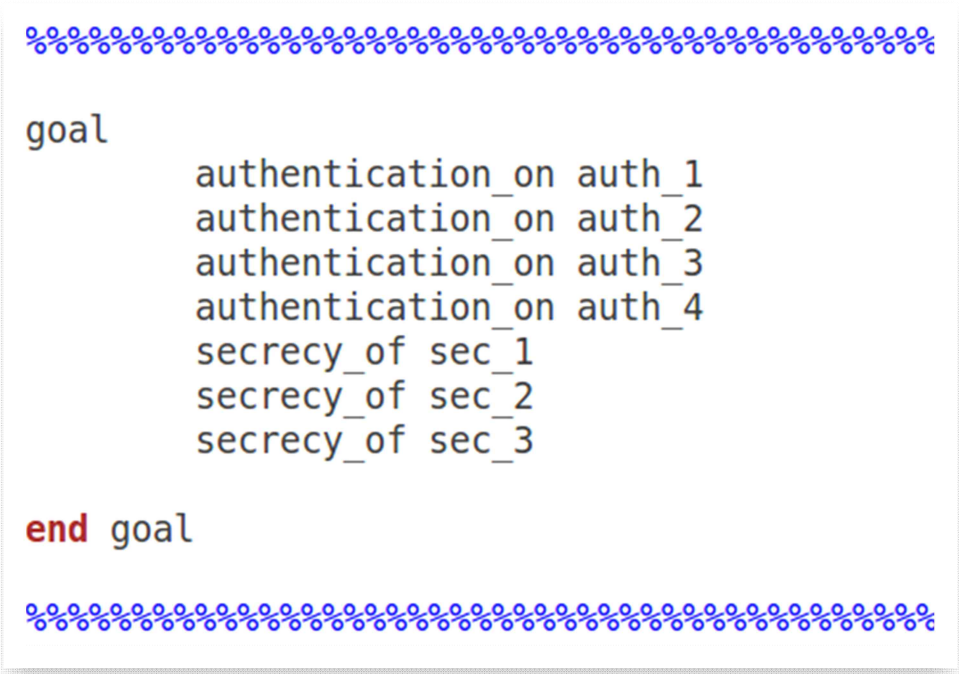
- Goal



[그림 6]

정형 검증 목표는 해당 프로토콜이 안전 한지에 대한 검사를 진행하는 것이다.
그렇기에 논문에 제안된 프로토콜 중 메시지의 무결성 및 기밀성 요소를 체킹하는 목표를 가지고 프로토콜에 대한 정형검증을 진행한다.

그림6에 보이는 직사각형 중 파란색 - 무결성, 빨간색 - 기밀성 검사가 진행된다.



[그림 7]

* 그림7은 HLPSL로 해당 메시지의 기밀성과, 무결성을 검증하기 위하여 작성한 파트이다.

- Environment Role

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

role environment() def=

        const   s,g,sv,p                                        :agent,
                h                                               :hash_func,
                pug,puauth,pup                                  :public_key,
                auth_1,auth_2,auth_3,auth_4,sec_1,sec_2,sec_3   :protocol_id

        intruder_knowledge = {s,g,sv,p,h}

        composition
                session(s,g,sv,p,h,pug,puauth,pup)

end role

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

전역변수는

1. Agent / 2. Hash_func / 3.Public_key / 4.Protocol_id

로 설정해 주었다.

우선 해당 프로토콜은 4명의 상호작용 주체(Sensor, Gateway, Server, Platform)이 존재하기에 해당 주체를 요약하여 4개의 Agent를 설정해 주었고,
모든 주체들은 동일한 Hash 함수를 사용하기에 Hash함수를 설정해 주었다.
또한 각 주체들은 자신의 공개키 및 상대방의 키를 알고 있어야 되기 때문에 공개키를 설정해 주었으며, 정형검증에 필요한 protocol_id를 전역변수로 설정하였다.

공격자의 배경지식은 4개의 주체, 해쉬함수로 설정해 주었다.

- Composed Role

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

role session(Sensor,Gateway,Server,Platform:agent, Hash:hash_func, PUg,PUauth,PUp:public_key)
def=
        local
                SND1,RCV1,SND2,RCV2,SND3,RCV3,SND4,RCV4:channel(dy)

        composition
                platform(Platform,Server,Gateway,Hash,PUauth,PUp,SND4,RCV4)
                /\server(Server,Gateway,Platform,Hash,PUg,PUauth,PUp,SND3,RCV3)
                /\gateway(Sensor,Gateway,Server,Platform,Hash,PUg,PUauth,SND1,RCV1)
                /\sensor(Sensor,Gateway,Hash,PUg,SND2,RCV2)

end role

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

Composed Role 에서는 해당 객체의 파라미터에 알맞게 객체들을 호출해주고, 각 객체에 할당할 채널들을 생성해준다.

- Basic Role

〈 Sensor 〉

```
role sensor(Sensor,Gateway:agent, Hash:hash_func, PUg:public_key, SND,RCV:channel(dy))
played_by Sensor def=

        local
                State   : nat,
                Ns      : text,
                IDs     : nat

        init
                State   :=0 /\ IDs :=123

        transition

        1. State  = 0 /\ RCV(start) =|>
           State':= 3 /\ Ns' := new()
                      /\ SND({Hash(IDs).Ns'}_PUg)


        2. State  = 3 /\ RCV({Ns}_inv(PUg)) =|>
           State':= 4 /\ request(Sensor, Gateway, auth_1, Ns)

end role
```

〈 Gateway 〉

```
role gateway(Sensor,Gateway,Server,Platform:agent, Hash:hash_func, PUg,PUauth:public_key, SND,RCV:channel(dy))
played_by Gateway def=

        local
                State                   : nat,
                Ns,Ng,Np                : text,
                IDs,IDp,IDg             : nat,
                SIDs                    : nat,
                SVIDp                   : nat,
                PUp                     : public_key,
                Ks                      : symmetric_key

        init
                State :=1 /\ IDs:=123 /\ IDg:=789 /\ IDp:=777

        transition

        1. State  = 1 /\ RCV({Hash(SIDs').Ns'}_PUg) =|>
           State':= 3 /\ equal(123,SIDs')
                      /\ SND({Ns'}_inv(PUg))
                      /\ witness(Gateway,Sensor,auth_1,Ns')
                      /\ SND({IDg.IDp.Hash(IDs)}_PUauth)

        2. State  = 3 /\ RCV({SVIDp'.PUp'}_inv(PUauth)) =|>
           State':= 5 /\ Ng':=new()
                      /\ SND({Ng'.IDg.Hash(IDs)}_PUp)

        3. State  = 5 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg) =|>
           State':= 9 /\ request(Gateway,Platform,auth_3,Ng)
                      /\ SND({Np'}_Ks)
                      /\ witness(Gateway,Platform,auth_4,Np')

end role
```

〈 Server 〉

```
role server(Server,Gateway,Platform:agent, Hash:hash_func, PUg,PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Server def=

        local
                State                   : nat,
                Ng                      : text,
                IDp,IDg,IDs,GIDp        : nat,
                PIDp,PIDg,PIDs          : nat,
                Ks                      : symmetric_key

        init
                State :=2 /\ IDp :=777

        transition

        1. State  = 2 /\ RCV({IDg'.GIDp'.Hash(IDs')}_PUauth) =|>
           State':= 4 /\ equal(777,GIDp')
                      /\ SND({IDp.PUp}_inv(PUauth))
                      /\ secret(PUp,sec_1,{Gateway,Server})

        2. State  = 4 /\ RCV({PIDp'.Ng'.PIDg'.Hash(PIDs')}_PUauth) =|>
           State':= 7 /\ equal(789,PIDg')
                      /\ SND({IDg.PUg}_inv(PUauth))
                      /\ secret(PUg,sec_2,{Platform,Server})
                      /\ Ks':=new()
                      /\ SND({{IDp.Ng'.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp)
                      /\ secret(Ks',sec_3,{Platform,Server})
                      /\ witness(Server,Platform,auth_2,Ng')

end role
```

〈 Platform 〉

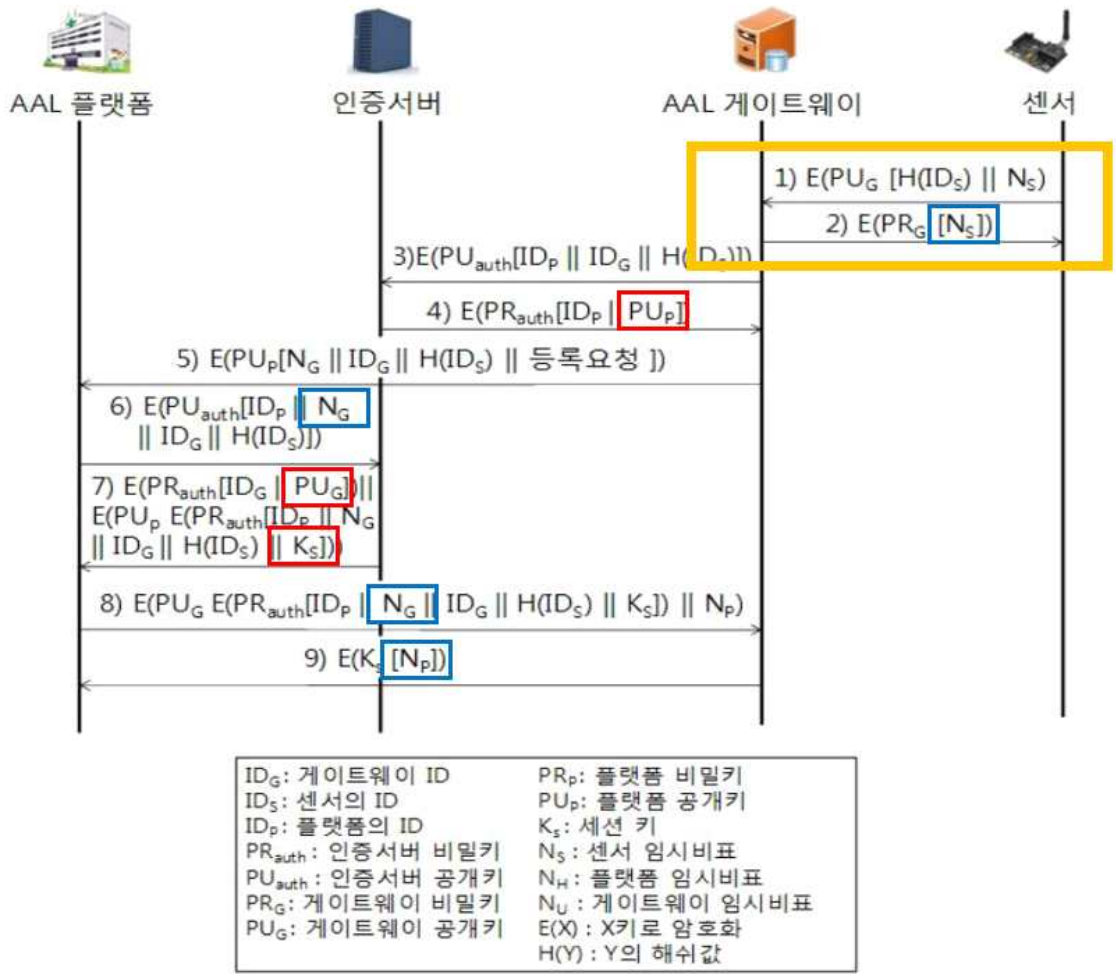
```
role platform(Platform,Server,Gateway:agent, Hash:hash_func, PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Platform def=

        local
                State                   : nat,
                Ng,Np                   : text,
                IDg,IDs,IDp             : nat,
                PUg                     : public_key,
                Ks                      : symmetric_key

        init
                State :=4 /\ IDp:=777

        transition

        1. State  = 4 /\ RCV({Ng'.IDg'.Hash(IDs')}_PUp) =|>
           State':= 5 /\ SND({IDp.Ng'.IDg'.Hash(IDs')}_PUauth)

        2. State  = 5 /\ RCV({IDg.PUg'}_inv(PUauth)) =|>
           State':= 6

        3. State  = 6 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp) =|>
           State':= 8 /\ request(Platform,Server,auth_2,Ng)
                      /\ Np':=new()
                      /\ SND({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg)
                      /\ witness(Platform,Gateway,auth_3,Ng)

        4. State  = 8 /\ RCV({Np}_Ks) =|>
           State':= 9 /\ request(Platform,Gateway,auth_4,Np)

end role
```

- Transition 절차

1st

〈 Sensor -〉 Gateway : SND({Hash(IDs).Ns'}_PUg) 〉

〈 Gateway -〉 Sensor : SND({Ns'}_inv(PUg)) 〉



1. AAL 센서는 자신의 식별자 해쉬 값 H(IDS)과 AAL 센서가 생성한 임시비표 Ns를 AAL 게이트웨이의 공개키로 암호화하여 AAL 게이트웨이에게 전송한다.

2. AAL 게이트웨이는 미리 등록되어 있는 AAL 센서 식별자의 해쉬 값을 계산하고, AAL 센서로부터 받은 해쉬 값을 비교해본다. 두 개의 해쉬 값이 동일하면 인증단계를 진행하고, 일치하지 않으면 인가되지 않은 센서로 가정하고 인증단계를 종료한다. AAL 센서와 성공적인 인증이 끝나면 AAL 센서가생성한 임시비표를 AAL 게이트웨이의 비밀키로 암호화해서 AAL 센서에게 전달해서 인증이 성공적으로 이루어졌음을 알린다.

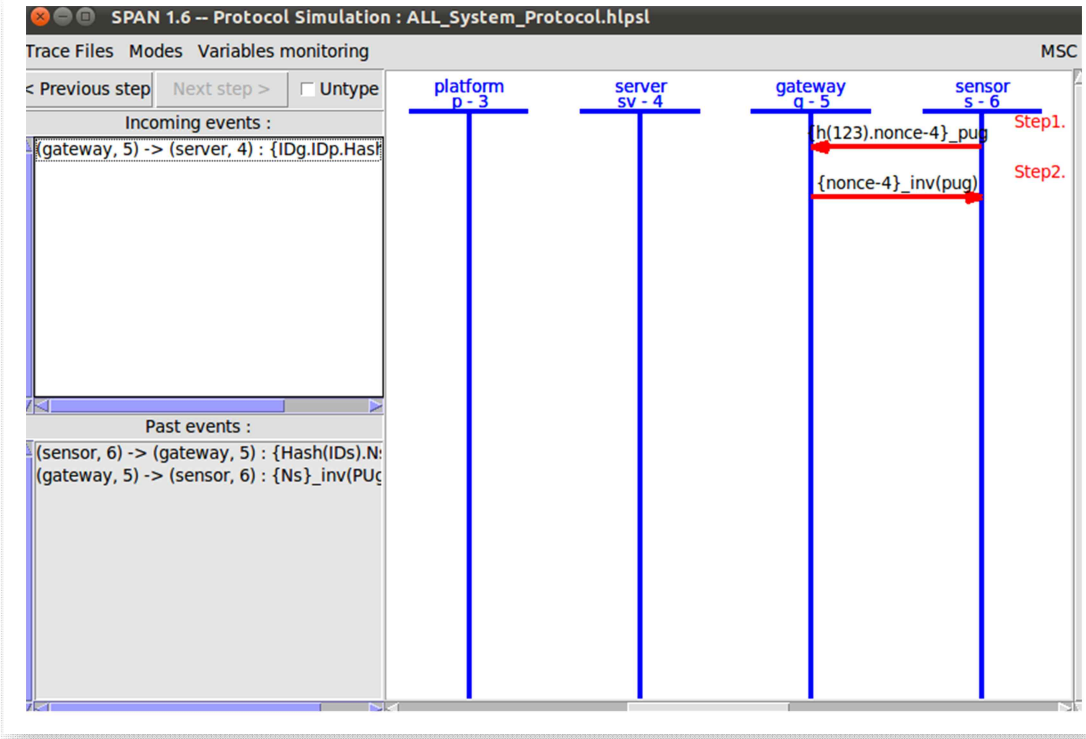
```
role sensor(Sensor,Gateway:agent, Hash:hash_func, PUg:public_key, SND,RCV:channel(dy))
played_by Sensor def=

        local
                State   : nat,
                Ns      : text,
                IDs     : nat

        init
                State   :=0 /\ IDs :=123

        transition

        1. State  = 0 /\ RCV(start) =|>
           State':= 3 /\ Ns' := new()
                      /\ SND({Hash(IDs).Ns'}_PUg)

        2. State  = 3 /\ RCV({Ns} inv(PUg)) =|>
           State':= 4 /\ request(Sensor, Gateway, auth_1, Ns)
end role
```

```
role gateway(Sensor,Gateway,Server,Platform:agent, Hash:hash_func, PUg,PUauth:public_key, SND,RCV:channel(dy))
played_by Gateway def=

        local
                State                   : nat,
                Ns,Ng,Np                : text,
                IDs,IDp,IDg             : nat,
                SIDs                    : nat,
                SVIDp                   : nat,
                PUp                     : public_key,
                Ks                      : symmetric_key

        init
                State :=1 /\ IDs:=123 /\ IDg:=789 /\ IDp:=777

        transition

        1. State  = 1 /\ RCV({Hash(SIDs').Ns'} PUg) =|>
           State':= 3 /\ equal(123,SIDs')
                      /\ SND({Ns'}_inv(PUg))
                      /\ witness(Gateway,Sensor,auth_1,Ns')
                      /\ SND({IDg.IDp.Hash(IDs)}_PUauth)

        2. State  = 3 /\ RCV({SVIDp'.PUp'}_inv(PUauth)) =|>
           State':= 5 /\ Ng':=new()
                      /\ SND({Ng'.IDg.Hash(IDs)}_PUp)

        3. State  = 5 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg) =|>
           State':= 9 /\ request(Gateway,Platform,auth_3,Ng)
                      /\ SND({Np'}_Ks)
                      /\ witness(Gateway,Platform,auth_4,Np')

end role
```

〈 정형검증 〉

Avispa에서 해당 프로토콜 검증 시 센서의 id를 123으로 설정하고 게이트웨이에서 equal 함수를 통하여 기존 존재하는 센서인지 검증을 진행해 줍니다.

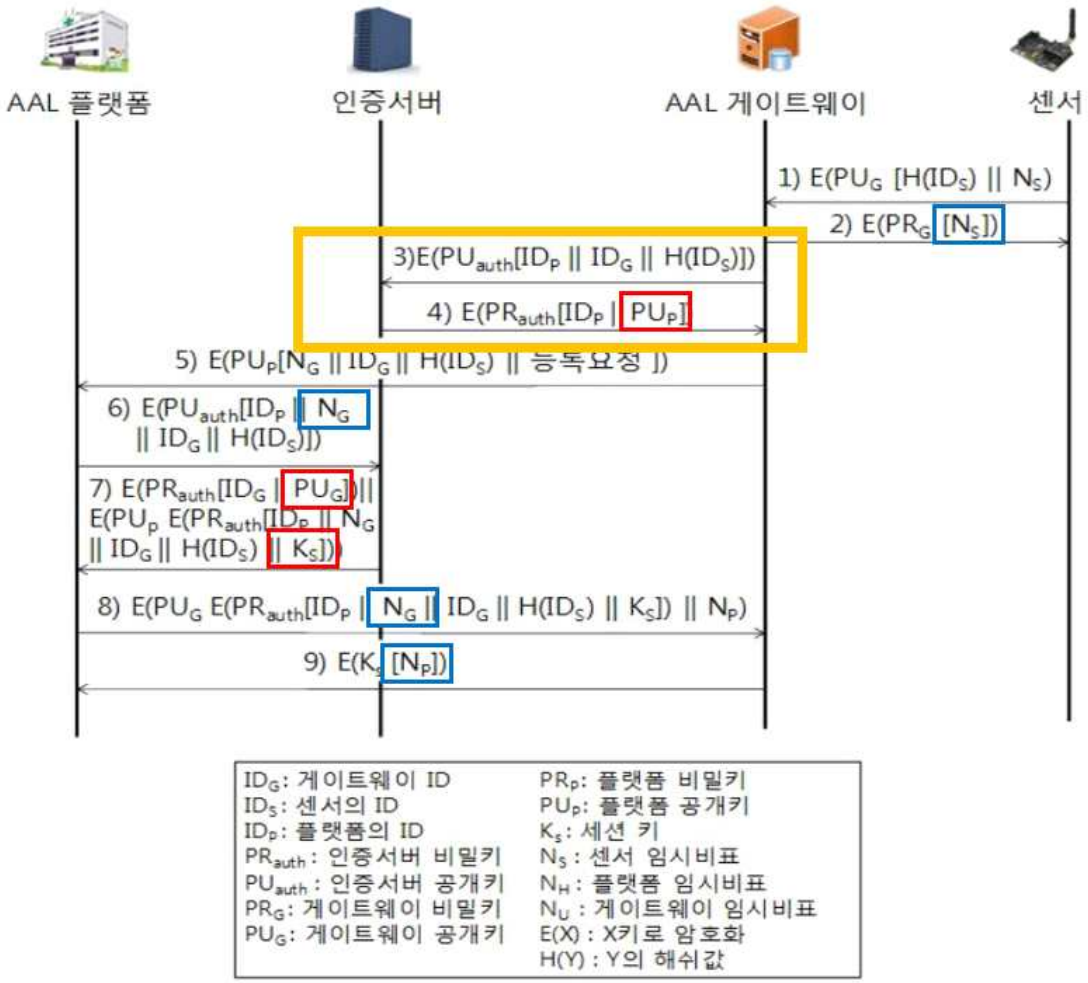
또한 센서는 자신의 임시 비표를 게이트웨이로부터 다시 받을 때 무결성검사를 통해 임시 비표가 자신이 생성한 비표가 맞는지 확인하는 절차를 가집니다.


SPAN 1.6 -- Protocol Simulation : ALL_System_Protocol.hlpsl
Trace Files
Modes
Variables monitoring
MSC
< Previous step
Next step >
Untype
Incoming events :
(gateway, 5) -> (server, 4) : {IDg.IDp.Hash
Past events :
(sensor, 6) -> (gateway, 5) : {Hash(IDs).N
(gateway, 5) -> (sensor, 6) : {Ns}_inv(PUg
platform
p - 3
server
sv - 4
gateway
g - 5
sensor
s - 6
Step1.
h(123).nonce-4}_pug
Step2.
{nonce-4}_inv(pug)

2nd

〈 Gateway -〉 Server : SND({IDg.IDp.Hash(IDs)}_PUauth) 〉

〈 Server -〉 Gateway : SND({SVIDp'.PUp'}_inv(PUauth)) 〉



3. AAL 게이트웨이는 AAL 플랫폼과 안전한 연결을 설정하기 위한 의도를 인증서버에 알리기 위하여, AAL 게이트웨이의 식별자 IDG, AAL 플랫폼의 식별자 IDP, 센서 식별자의 해쉬 값 H(IDS)을 인증서버의 공개키 PUauth 암호화하여 인증서버로 전송한다.

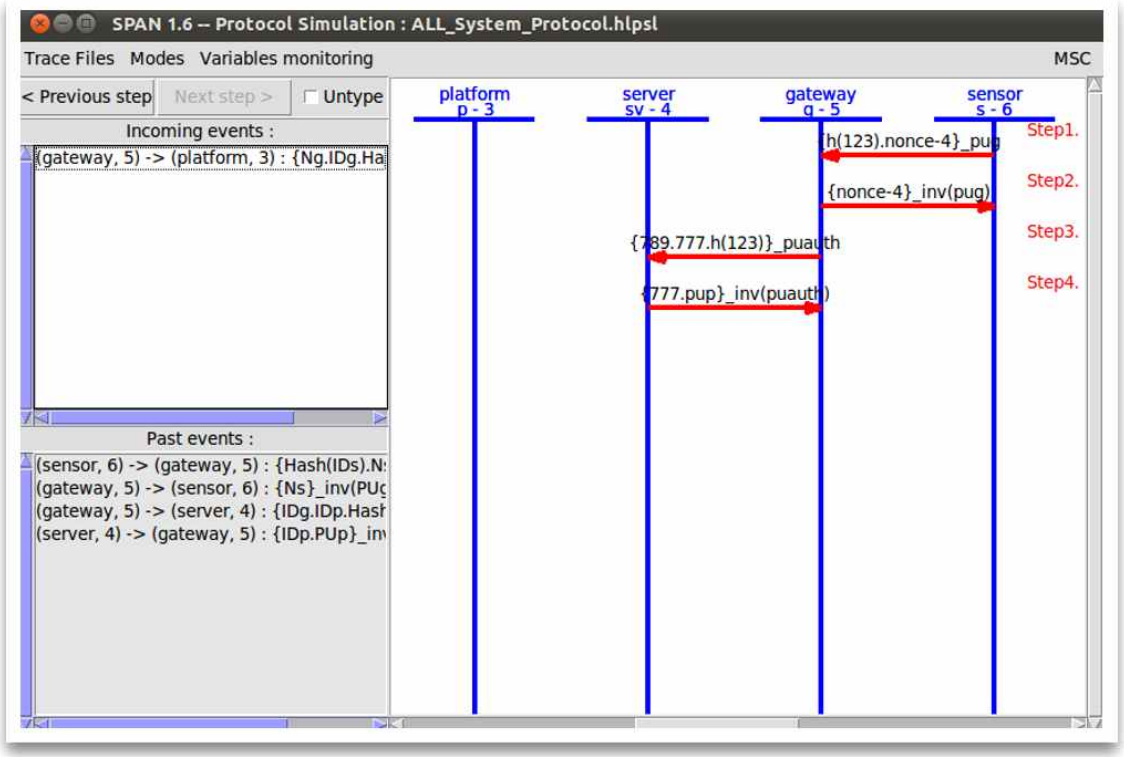
4. 인증서버는 AAL 게이트웨이의 식별자와 센서 식별자의 해쉬 값을 인증서버에 등록한다. 그리고, 인증서버는 AAL 플랫폼의 식별자 IDP와 AAL 플랫폼의 공개키 PUP를 복사한 후 인증서버의 개인키 PRauth로 암호화하여 AAL 게이트웨이에게 반송한다.

```
role gateway(Sensor,Gateway,Server,Platform:agent, Hash:hash_func, PUg,PUauth:public_key, SND,RCV:channel(dy))
played_by Gateway def=

        local
                State                   : nat,
                Ns,Ng,Np                : text,
                IDs,IDp,IDg             : nat,
                SIDs                    : nat,
                SVIDp                   : nat,
                PUp                     : public_key,
                Ks                      : symmetric_key

        init
                State :=1 /\ IDs:=123 /\ IDg:=789 /\ IDp:=777

        transition

        1. State  = 1 /\ RCV({Hash(SIDs').Ns'}_PUg) =|>
           State':= 3 /\ equal(123,SIDs')
                      /\ SND({Ns'}_inv(PUg))
                      /\ witness(Gateway,Sensor,auth_1,Ns')
                      /\ SND({IDg.IDp.Hash(IDs)}_PUauth)

        2. State  = 3 /\ RCV({SVIDp'.PUp'}_inv(PUauth)) =|>
           State':= 5 /\ Ng':=new()
                      /\ SND({Ng'.IDg.Hash(IDs)}_PUp)

        3. State  = 5 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg) =|>
           State':= 9 /\ request(Gateway,Platform,auth_3,Ng)
                      /\ SND({Np'}_Ks)
                      /\ witness(Gateway,Platform,auth_4,Np')

end role
```

```
role server(Server,Gateway,Platform:agent, Hash:hash_func, PUg,PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Server def=

        local
                State                   : nat,
                Ng                      : text,
                IDp,IDg,IDs,GIDp        : nat,
                PIDp,PIDg,PIDs          : nat,
                Ks                      : symmetric_key

        init
                State :=2 /\ IDp :=777

        transition

        1. State  = 2 /\ RCV({IDg'.GIDp'.Hash(IDs')}_PUauth) =|>
           State':= 4 /\ equal(777,GIDp')
                      /\ SND({IDp.PUp}_inv(PUauth))
                      /\ secret(PUp,sec_1,{Gateway,Server})

        2. State  = 4 /\ RCV({PIDp'.Ng'.PIDg'.Hash(PIDs')}_PUauth) =|>
           State':= 7 /\ equal(789,PIDg')
                      /\ SND({IDg.PUg}_inv(PUauth))
                      /\ secret(PUg,sec_2,{Platform,Server})
                      /\ Ks':=new()
                      /\ SND({{IDp.Ng'.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp)
                      /\ secret(Ks',sec_3,{Platform,Server})
                      /\ witness(Server,Platform,auth_2,Ng')

end role
```

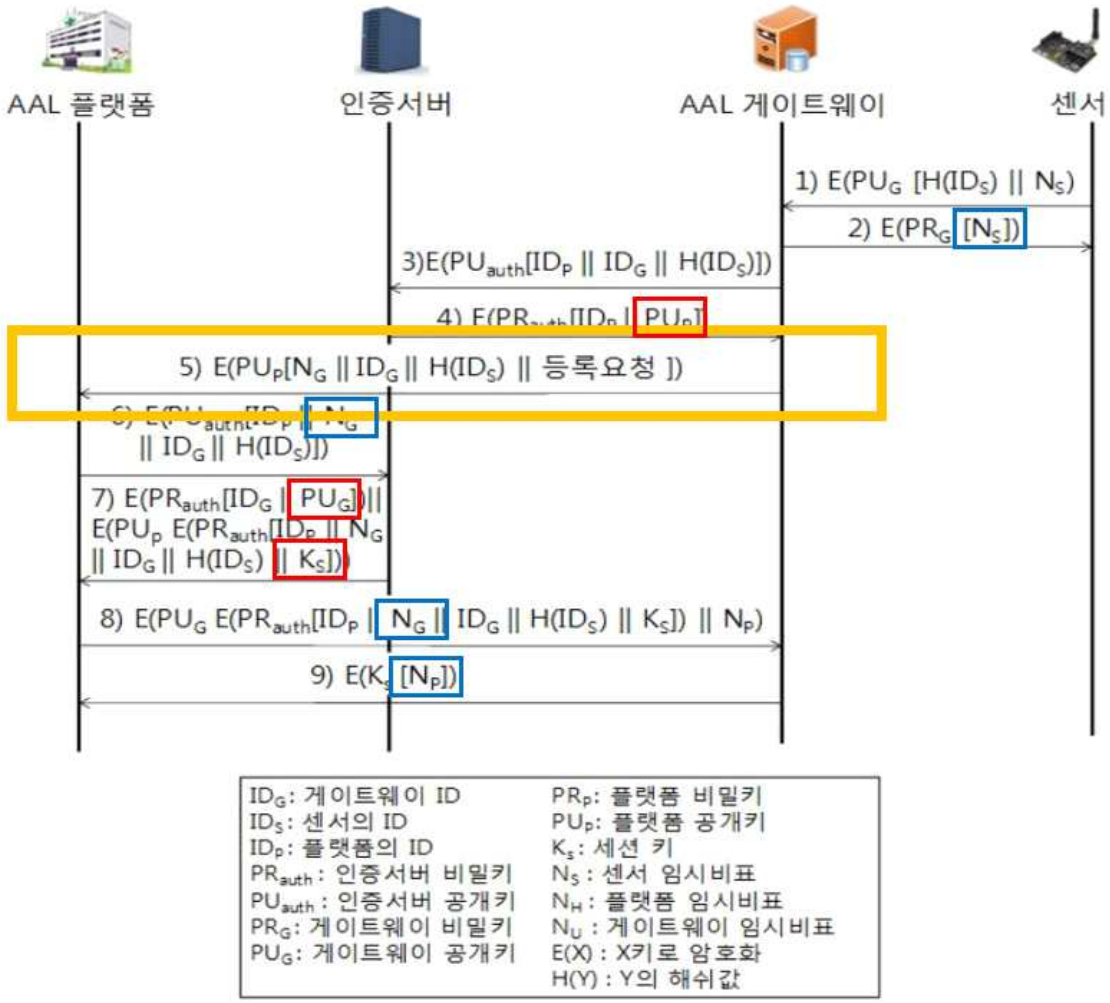
〈 정형검증 〉

프로토콜에서 게이트웨이는 통신을 원하는 플랫폼의 공개키를 인증 서버로부터 요청하여 받게 되는데 이때 키에 대한 기밀성 검사를 진행해줍니다.


SPAN 1.6 -- Protocol Simulation : ALL_System_Protocol.hlpsl
Trace Files
Modes
Variables monitoring
MSC
< Previous step
Next step >
Untype
Incoming events :
(gateway, 5) -> (platform, 3) : {Ng.IDg.Ha
Past events :
(sensor, 6) -> (gateway, 5) : {Hash(IDs).N
(gateway, 5) -> (sensor, 6) : {Ns}_inv(PUg
(gateway, 5) -> (server, 4) : {IDg.IDp.Hash
(server, 4) -> (gateway, 5) : {IDp.PUp}_inv
platform
p - 3
server
sv - 4
gateway
g - 5
sensor
s - 6
Step1.
h(123).nonce-4}_pug
Step2.
{nonce-4}_inv(pug)
Step3.
{789.777.h(123)}_puauth
Step4.
{777.pup}_inv(puauth)

3rd

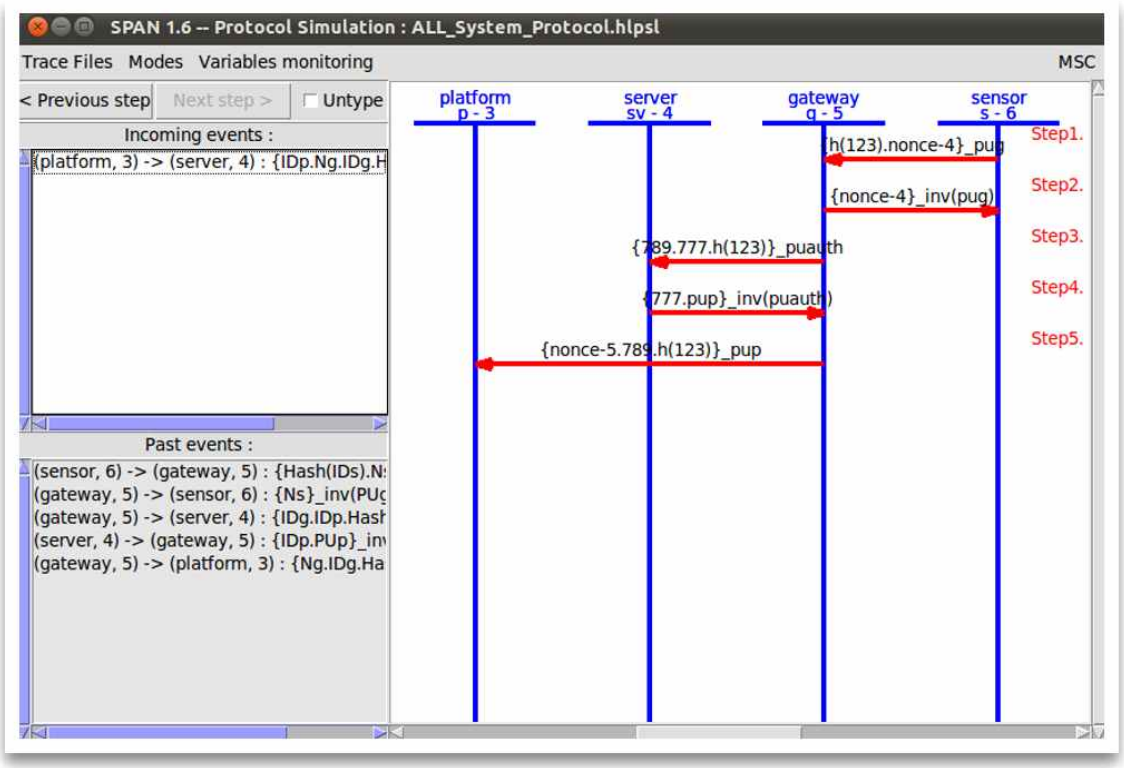
〈 Gateway -〉 Platfrom : SND({Ng'.IDg.Hash(IDs))_PUp 〉



5. AAL 게이트웨이는 AAL 플랫폼에게 안전한 연결을 설정하기 위하여 등록요청 메시지를 보낸다. 등록요청 메시지는 AAL 게이트웨이의 식별자와 센서 식별자의 해쉬 값, 그리고 AAL 게이트에서 생성한 임시비표 NG와 함께 AAL 플랫폼의 공개키로 암호화한다.

```
role gateway(Sensor,Gateway,Server,Platform:agent, Hash:hash_func, PUg,PUauth:public_key, SND,RCV:channel(dy))
played_by Gateway def=

        local
                State                   : nat,
                Ns,Ng,Np                : text,
                IDs,IDp,IDg             : nat,
                SIDs                    : nat,
                SVIDp                   : nat,
                PUp                     : public_key,
                Ks                      : symmetric_key

        init
                State :=1 /\ IDs:=123 /\ IDg:=789 /\ IDp:=777

        transition

        1. State  = 1 /\ RCV({Hash(SIDs').Ns'}_PUg) =|>
           State':= 3 /\ equal(123,SIDs')
                      /\ SND({Ns'}_inv(PUg))
                      /\ witness(Gateway,Sensor,auth_1,Ns')
                      /\ SND({IDg.IDp.Hash(IDs)}_PUauth)

        2. State  = 3 /\ RCV({SVIDp'.PUp'}_inv(PUauth)) =|>
           State':= 5 /\ Ng':=new()
                      /\ SND({Ng'.IDg.Hash(IDs)}_PUp)

        3. State  = 5 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg) =|>
           State':= 9 /\ request(Gateway,Platform,auth_3,Ng)
                      /\ SND({Np'}_Ks)
                      /\ witness(Gateway,Platform,auth_4,Np')

end role
```

```
role platform(Platform,Server,Gateway:agent, Hash:hash_func, PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Platform def=

        local
                State                   : nat,
                Ng,Np                   : text,
                IDg,IDs,IDp             : nat,
                PUg                     : public_key,
                Ks                      : symmetric_key

        init
                State :=4 /\ IDp:=777

        transition

        1. State  = 4 /\ RCV({Ng'.IDg'.Hash(IDs')}_PUp) =|>
           State':= 5 /\ SND({IDp.Ng'.IDg'.Hash(IDs')}_PUauth)

        2. State  = 5 /\ RCV({IDg.PUg'}_inv(PUauth)) =|>
           State':= 6

        3. State  = 6 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp) =|>
           State':= 8 /\ request(Platform,Server,auth_2,Ng)
                      /\ Np':=new()
                      /\ SND({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg)
                      /\ witness(Platform,Gateway,auth_3,Ng)

        4. State  = 8 /\ RCV({Np}_Ks) =|>
           State':= 9 /\ request(Platform,Gateway,auth_4,Np)

end role
```

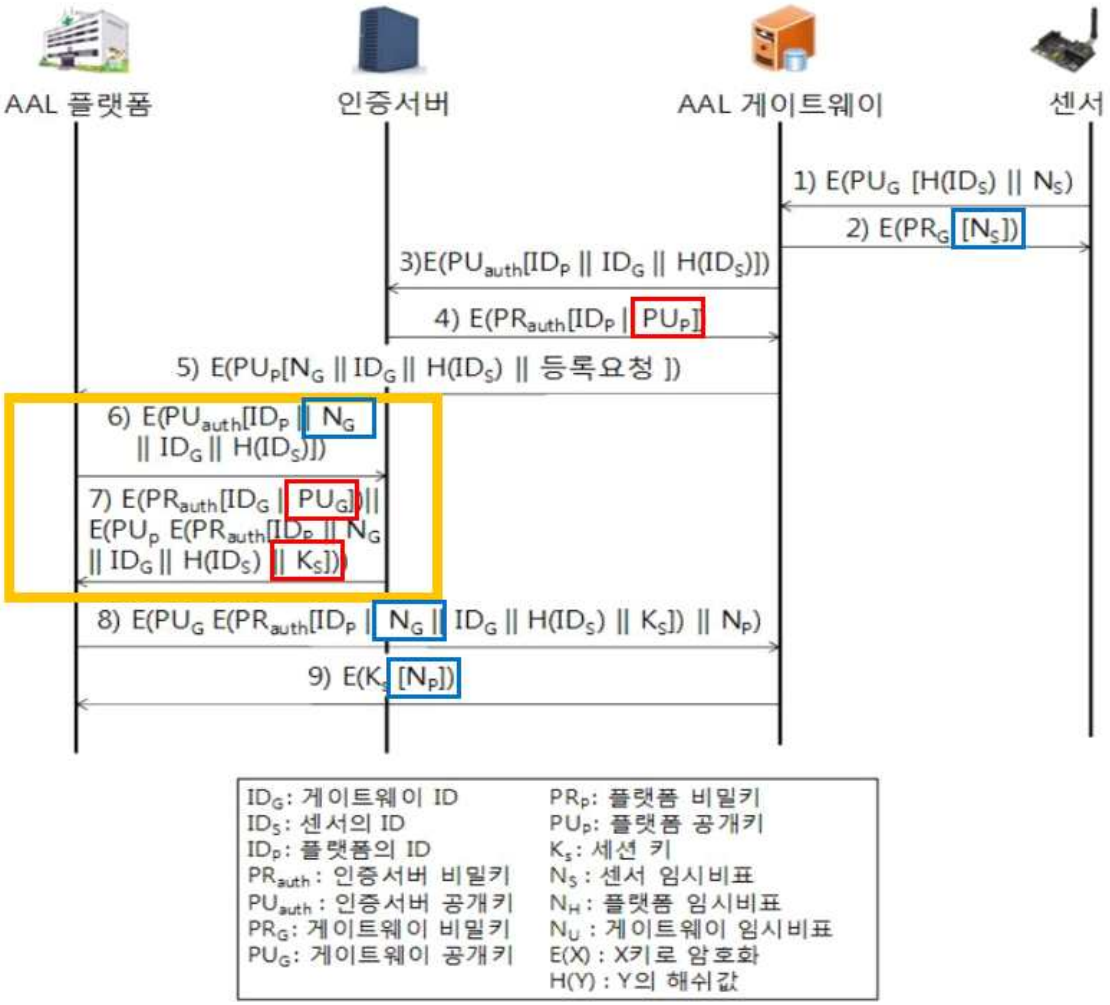

SPAN 1.6 -- Protocol Simulation : ALL_System_Protocol.hlpsl
Trace Files
Modes
Variables monitoring
MSC
< Previous step
Next step >
Untype
Incoming events :
(platform, 3) -> (server, 4) : {IDp.Ng.IDg.H
Past events :
(sensor, 6) -> (gateway, 5) : {Hash(IDs).N
(gateway, 5) -> (sensor, 6) : {Ns}_inv(PUg
(gateway, 5) -> (server, 4) : {IDg.IDp.Hash
(server, 4) -> (gateway, 5) : {IDp.PUp}_inv
(gateway, 5) -> (platform, 3) : {Ng.IDg.Ha
platform
p - 3
server
sv - 4
gateway
g - 5
sensor
s - 6
Step1.
h(123).nonce-4}_pug
Step2.
{nonce-4}_inv(pug)
Step3.
{789.777.h(123)}_puauth
Step4.
{777.pup}_inv(puauth)
Step5.
{nonce-5.789.h(123)}_pup

4th

〈 Platform -〉 Server : SND({IDp.Ng'.IDg'.Hash(IDs')}_PUauth) 〉

〈 Server -〉 Platform : SND({IDg.PUg})_inv(PUauth)}) 〉

〈 Server -〉 Platform : SND({{IDp.Ng'.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp) 〉



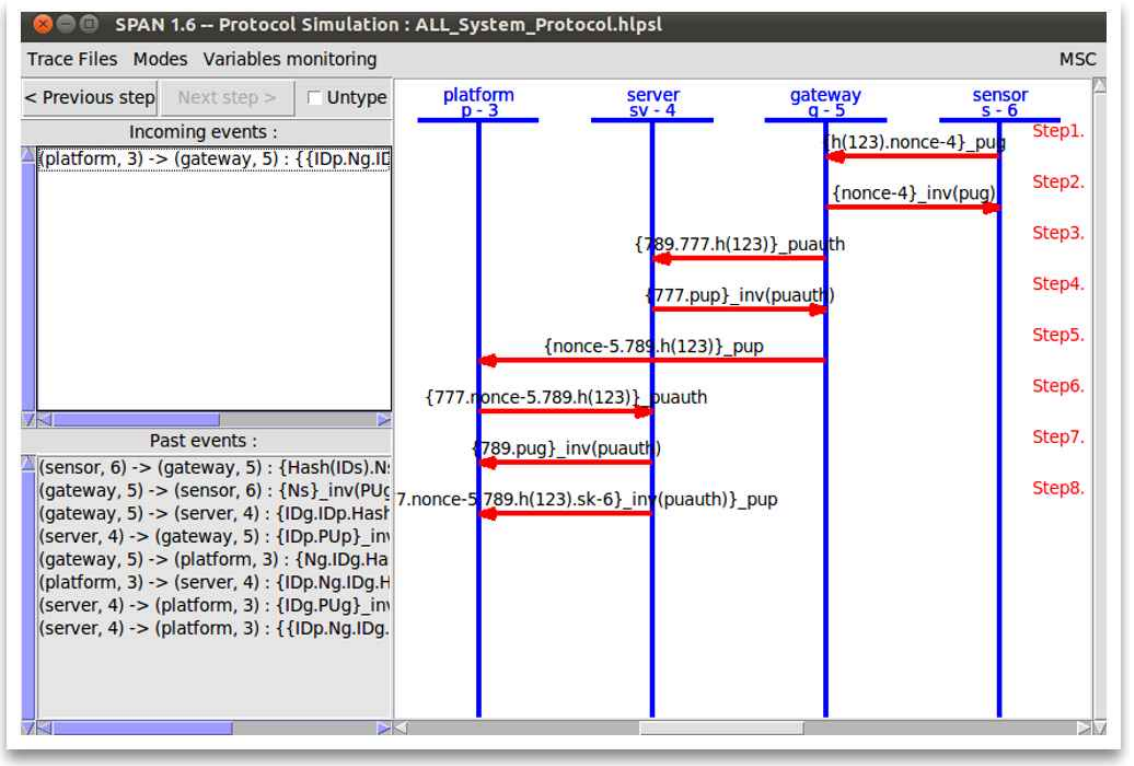
6. AAL 플랫폼은 인증서버에게 AAL 게이트웨이 공개키와 세션 키 KS를 요청하기 위하여 AAL 플랫폼의 식별자, AAL 게이트웨이 식별자, AAL 센서의 해쉬 값, AAL 게이트웨이의 임시비표를 인증서버의 공개키로 암호화하여 전송한다. AAL 플랫폼은 AAL 게이트웨이에서 발행한 임시비표를 포함함으로써 인증 서버가 AAL 게이트웨이의 임시비표에 세션 키를 서명할 수 있도록 한다.

7. 인증서버는 AAL 플랫폼에게 AAL 게이트웨이의 식별자 IDG와 AAL 게이트웨이의 공개키 PUG를 복사한 후 인증서버의 비밀키로 암호화한다. 또한, AAL 플랫폼 식별자, AAL

게이트웨이의 임시비표, AAL 게이트웨이 식별자, 센서 식별자의 해쉬 값, 세션 키를 인증서버의 비밀키로 암호화한 후 AAL플랫폼의 공개키로 암호화하여 AAL 플랫폼에 전송한다. 세션 키 Ks는 AAL 플랫폼을 대신하여 인증서버에 의해 만들어지고 AAL 게이트웨이의 비표에 결부된 세션 키이다. 즉, Ks 와 AAL 게이트웨이 비표 NG의 결합은 AAL 게이트웨이에게 세션키 Ks 가 투명하다는 것을 보장한다. 이 세 가지 정보는 인증서버의 비밀 키를 이용하여 암호화됨으로 AAL 플랫폼에게 인증서버에 의해서 만들어졌음을 검증한다. 이 정보는 또한 AAL 플랫폼의 공개키를 사용하여 암호화됨으로써 사용자와의 부정한 연결을 설립하는 시도에 이용하지 못하게 한다.

```
role platform(Platform,Server,Gateway:agent, Hash:hash_func, PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Platform def=

        local
                State                   : nat,
                Ng,Np                   : text,
                IDg,IDs,IDp             : nat,
                PUg                     : public_key,
                Ks                      : symmetric_key

        init
                State :=4 /\ IDp:=777

        transition

        1. State  = 4 /\ RCV({Ng'.IDg'.Hash(IDs')}_PUp) =|>
           State':= 5 /\ SND({IDp.Ng'.IDg'.Hash(IDs')}_PUauth)

        2. State  = 5 /\ RCV({IDg.PUg'}_inv(PUauth)) =|>
           State':= 6

        3. State  = 6 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp) =|>
           State':= 8 /\ request(Platform,Server,auth_2,Ng)
                      /\ Np':=new()
                      /\ SND({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg)
                      /\ witness(Platform,Gateway,auth_3,Ng)
```

```
role server(Server,Gateway,Platform:agent, Hash:hash_func, PUg,PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Server def=

        local
                State                   : nat,
                Ng                      : text,
                IDp,IDg,IDs,GIDp        : nat,
                PIDp,PIDg,PIDs          : nat,
                Ks                      : symmetric_key

        init
                State :=2 /\ IDp :=777

        transition

        1. State  = 2 /\ RCV({IDg'.GIDp'.Hash(IDs')}_PUauth) =|>
           State':= 4 /\ equal(777,GIDp')
                      /\ SND({IDp.PUp}_inv(PUauth))
                      /\ secret(PUp,sec_1,{Gateway,Server})

        2. State  = 4 /\ RCV({PIDp'.Ng'.PIDg'.Hash(PIDs')}_PUauth) =|>
           State':= 7 /\ equal(789,PIDg')
                      /\ SND({IDg.PUg}_inv(PUauth))
                      /\ secret(PUg,sec_2,{Platform,Server})
                      /\ Ks':=new()
                      /\ SND({{IDp.Ng'.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp)
                      /\ secret(Ks',sec_3,{Platform,Server})
                      /\ witness(Server,Platform,auth_2,Ng')

end role
```



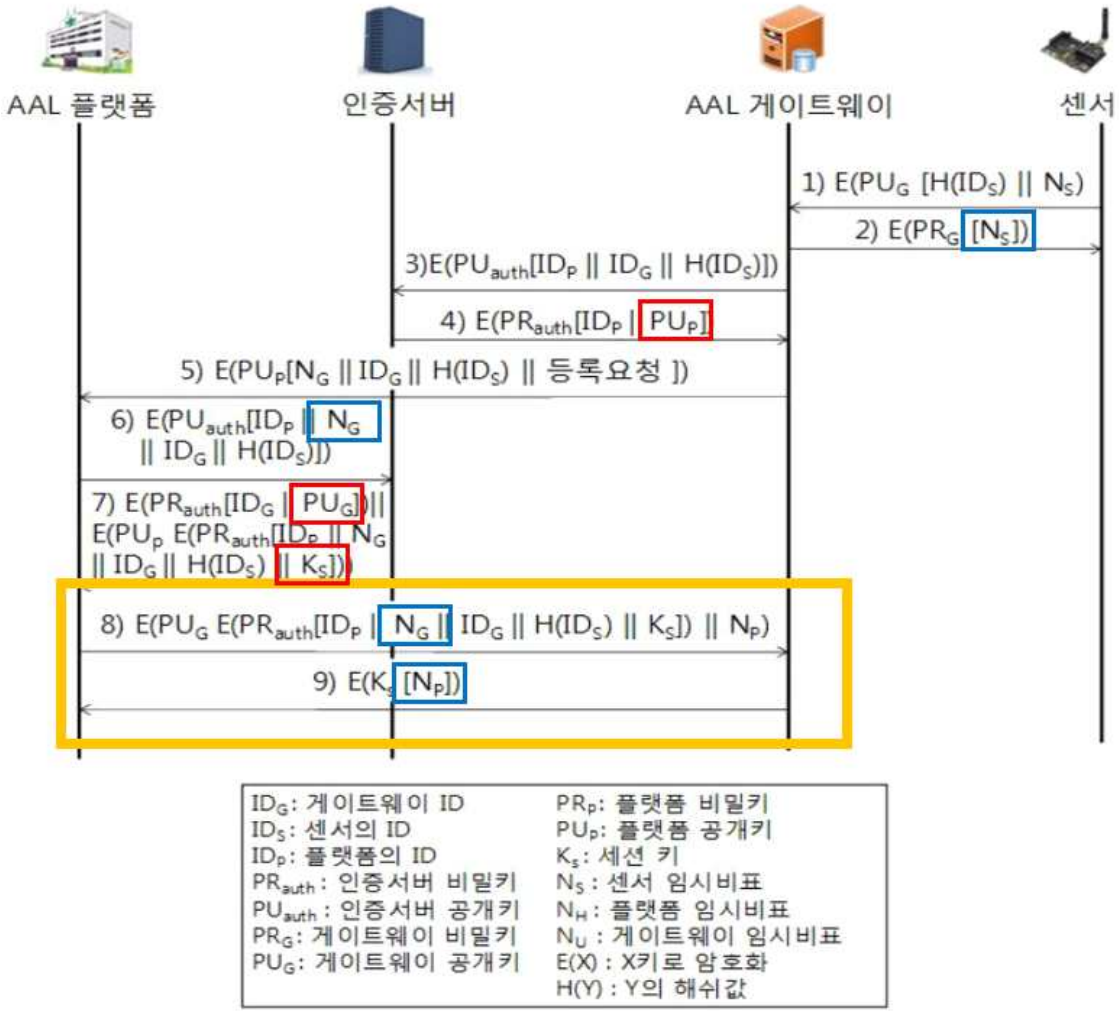
〈 정형검증 〉

플랫폼은 인증서버로부터 해당 게이트웨이의 요청과 공개키를 요청하는 메시지를 보낸다. 서버는 이러한 메시지로부터 해당 게이트 웨이와 통신에 사용되는 세션키를 응답하기에 이에대한 키 기밀성 검사를 진행한다. 또한 게이트웨이에게 세션키 Ks를 전달시 게이트웨이의 임시비표를 통해 세션키가 투명하다를 보장하게끔 프로토콜은 구성이 되어있다. 그러하여 플랫폼과 서버 사이에 게이트웨이의 메시지안 포함되어있는 게이트웨이의 임시비표 무결성 또한 검사를 진행해 준다.

5th

〈 Platform -〉 Gateway : SND({IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'})_PUg) 〉

〈 Gateway -〉 Platform : SND({Np}_Ks) 〉



8. AAL 플랫폼은 AAL 게이트웨이에게 세션 키를 전송하기 위해서 AAL 플랫폼 식별자, AAL 게이트웨이의 임시비표, AAL 게이트웨이 식별자, 센서 식별자의 해쉬 값, 세션 키를 인증서버의 비밀키로 암호화한 후 AAL 플랫폼에서 생성된 임시비표 Np와 함께 AAL 게이트웨이의 공개키로 암호화하여 AAL 게이트웨이에 전송한다.

9. AAL 게이트웨이는 세션 키 Ks를 회수하여 AAL플랫폼의 임시비표를 암호화하여 AAL 플랫폼에 반송한다. 이 마지막 메시지는 세션 키에 대한 정보가 안전하다는 것을 보장한다. AAL 플랫폼과AAL 게이트웨이는 생성된 세션 키를 사용하여 세션기간동안 센서에서 수집된 정보를 전송할 수 있다.

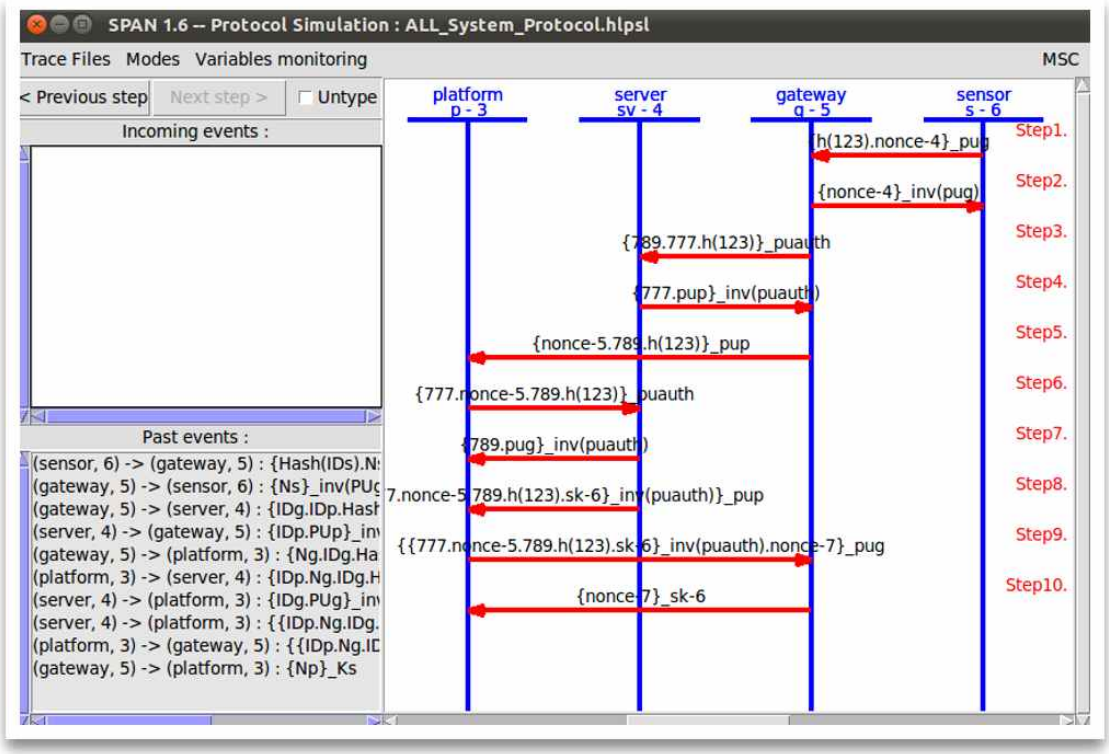
```
role platform(Platform,Server,Gateway:agent, Hash:hash_func, PUauth,PUp:public_key, SND,RCV:channel(dy))
played_by Platform def=

        local
                State                   : nat,
                Ng,Np                   : text,
                IDg,IDs,IDp             : nat,
                PUg                     : public_key,
                Ks                      : symmetric_key

        init
                State :=4 /\ IDp:=777

        transition

        1. State  = 4 /\ RCV({Ng'.IDg'.Hash(IDs')}_PUp) =|>
           State':= 5 /\ SND({IDp.Ng'.IDg'.Hash(IDs')}_PUauth)

        2. State  = 5 /\ RCV({IDg.PUg'}_inv(PUauth)) =|>
           State':= 6

        3. State  = 6 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth)}_PUp) =|>
           State':= 8 /\ request(Platform,Server,auth_2,Ng)
                      /\ Np':=new()
                      /\ SND({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg)
                      /\ witness(Platform,Gateway,auth_3,Ng)

        4. State  = 8 /\ RCV({Np}_Ks) =|>
           State':= 9 /\ request(Platform,Gateway,auth_4,Np)

end role
```

```
role gateway(Sensor,Gateway,Server,Platform:agent, Hash:hash_func, PUg,PUauth:public_key, SND,RCV:channel(dy))
played_by Gateway def=

        local
                State                   : nat,
                Ns,Ng,Np                : text,
                IDs,IDp,IDg             : nat,
                SIDs                    : nat,
                SVIDp                   : nat,
                PUp                     : public_key,
                Ks                      : symmetric_key

        init
                State :=1 /\ IDs:=123 /\ IDg:=789 /\ IDp:=777

        transition

        1. State  = 1 /\ RCV({Hash(SIDs').Ns'}_PUg) =|>
           State':= 3 /\ equal(123,SIDs')
                      /\ SND({Ns'}_inv(PUg))
                      /\ witness(Gateway,Sensor,auth_1,Ns')
                      /\ SND({IDg.IDp.Hash(IDs)}_PUauth)

        2. State  = 3 /\ RCV({SVIDp'.PUp'}_inv(PUauth)) =|>
           State':= 5 /\ Ng':=new()
                      /\ SND({Ng'.IDg.Hash(IDs)}_PUp)

        3. State  = 5 /\ RCV({{IDp.Ng.IDg.Hash(IDs).Ks'}_inv(PUauth).Np'}_PUg) =|>
           State':= 9 /\ request(Gateway,Platform,auth_3,Ng)
                      /\ SND({Np'}_Ks)
                      /\ witness(Gateway,Platform,auth_4,Np')

end role
```

〈 정형검증 〉

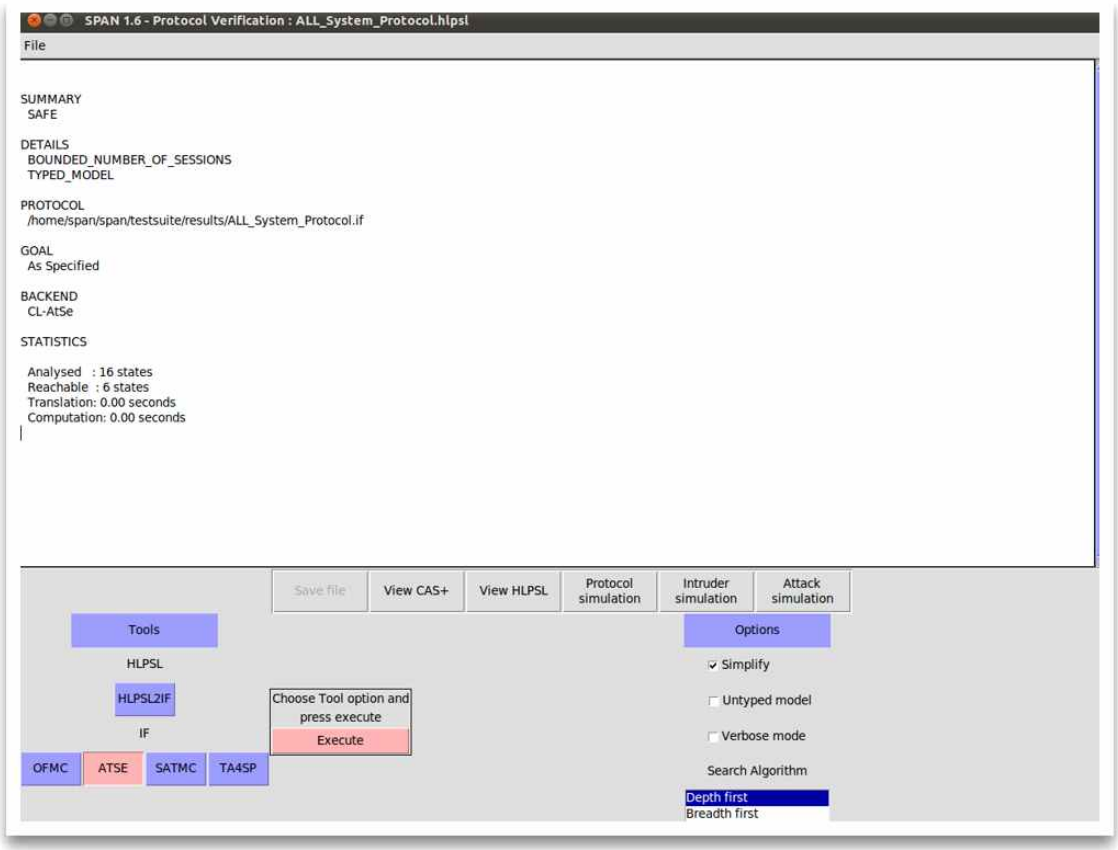
플랫폼은 게이트웨이에게 기존 메시지 + 세션키 + 플랫폼의 임시 비표를 생성하여 전달하고 게이트웨이는 플랫폼의 임시비표를 세션키로 암호화하여 플랫폼에게 전달합니다.

이때 임시비표 값에 대한 무결성 검사를 진행해 줍니다.

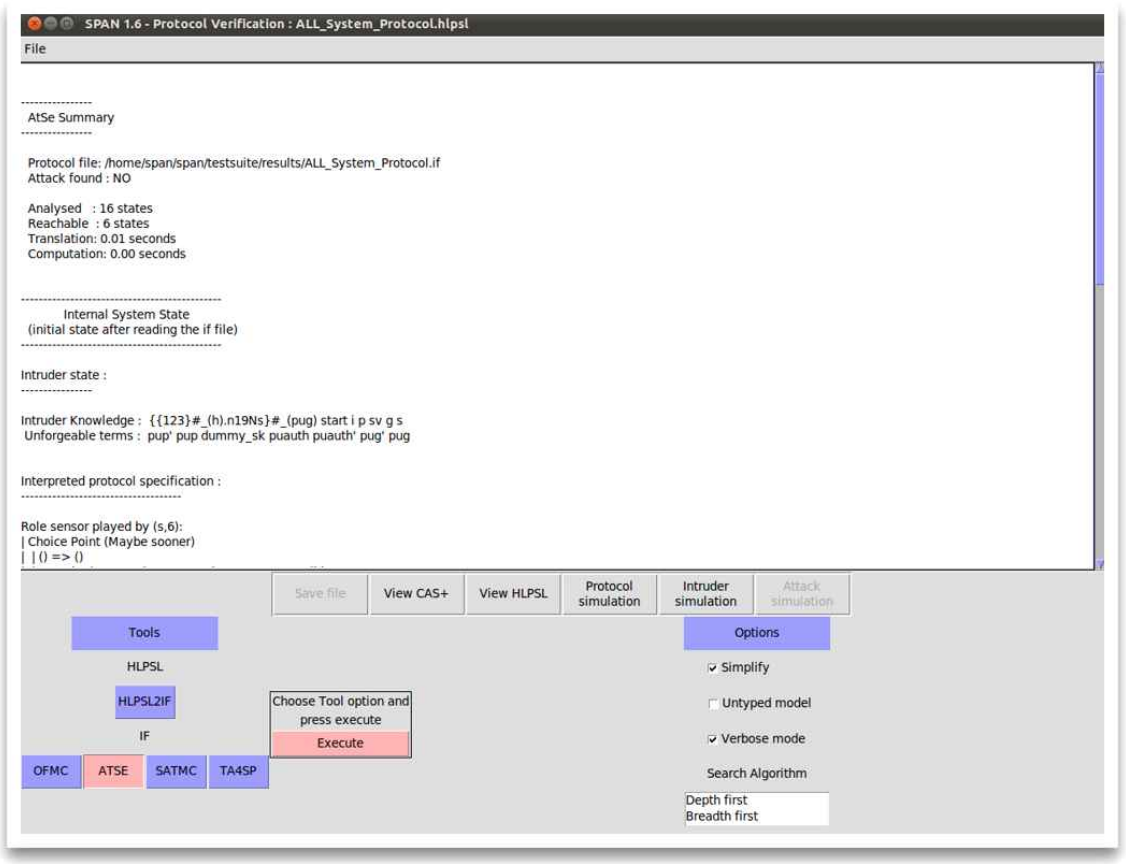

SPAN 1.6 -- Protocol Simulation : ALL_System_Protocol.hlpsl
Trace Files
Modes
Variables monitoring
MSC
< Previous step
Next step >
Untype
Incoming events :
Past events :
(sensor, 6) -> (gateway, 5) : {Hash(IDs).N
(gateway, 5) -> (sensor, 6) : {Ns}_inv(PU
(gateway, 5) -> (server, 4) : {IDg.IDp.Hash
(server, 4) -> (gateway, 5) : {IDp.PUp}_in
(gateway, 5) -> (platform, 3) : {Ng.IDg.Ha
(platform, 3) -> (server, 4) : {IDp.Ng.IDg.H
(server, 4) -> (platform, 3) : {IDg.PUg}_in
(server, 4) -> (platform, 3) : {{IDp.Ng.IDg.
(platform, 3) -> (gateway, 5) : {{IDp.Ng.ID
(gateway, 5) -> (platform, 3) : {Np}_Ks
platform
p - 3
server
sv - 4
gateway
g - 5
sensor
s - 6
Step1.
h(123).nonce-4}_pug
Step2.
{nonce-4}_inv(pug)
Step3.
{789.777.h(123)}_puauth
Step4.
{777.pup}_inv(puauth)
Step5.
{nonce-5.789.h(123)}_pup
Step6.
{777.nonce-5.789.h(123)}_puauth
Step7.
{789.pug}_inv(puauth)
Step8.
7.nonce-5.789.h(123).sk-6}_inv(puauth)}_pup
Step9.
{{777.nonce-5.789.h(123).sk-6}_inv(puauth).nonce-7}_pug
Step10.
{nonce-7}_sk-6

3. 정형 검증 결과 (ATSE)

- Simplify Mode (SAFE)



- Verbose Mode (SAFE)



- 결론

해당 논문(전천 후 생활 지원 시스템을 위한 경량 인증 프로토콜)은 안전하다는 것을 알 수 있다.

- HLPSL 파일 링크

https://github.com/GongWook/Avispa-AAL-System-Protocol.git