Sécurité des Systèmes d'Informations

Activation d'une autorisation d'accès à un réseau wifi privé

Guillaume Zablot

Antoine Courtil

Rôles intervenant dans le protocole

Pour notre projet de protocole, nous avons défini trois rôles distincts.

Le premier rôle est celui du Client. Ce client désire s'authentifier sur le réseau afin d'y accéder. Il veut donc tenter d'intégrer le réseau en fournissant le mot de passe adéquat. En échange, il veut récupérer sa nouvelle adresse IP.

Le second rôle est celui du Serveur d'authentification. Le serveur est l'unité qui gère tout le système de connexion d'une nouvelle machine sur le réseau. Pour ce faire, ce dernier veille à ce que la connexion soit correcte en vérifiant si le mot de passe fourni est valide.

Le troisième rôle est celui de la borne d'accès Wifi. Son rôle est uniquement d'assurer la liaison entre une client et le serveur afin de faire transiter les informations. De plus, elle émet en tout temps des informations utiles pour la connexion au réseau.

Informations principales échangées

Avant toute chose, la première information échangée est le certificat du serveur d'authentification. Ce certificat est composé de diverses informations du serveur notamment sa clé publique. Une signature est aussi incluse dans ce certificat, qui est signée avec la clé privée d'une autorité de certification connue.

Par la suite, le client envoie son adresse MAC et sa clé publique afin d'être connu par le serveur.

Ce dernier va lui répondre par une clé de session afin de pouvoir effectuer des échanges avec un chiffrement symétrique.

Afin de s'authentifier, le client envoie le hash du mot de passe réseau.

Si la connexion est validée par le serveur, il répondra donc en envoyant une clé réseau, qui permettra de chiffrer toutes les communications sur le réseau.

Propriétés de sécurité à satisfaire

Afin d'assurer la sécurité de ce protocole, il est important de satisfaire les points clés suivants :

La clé publique du serveur doit être irréfutable.

Malgré le fait qu'elle soit diffusée en clair, cette clé publique est extrêmement importante car c'est avec elle que va commencer le dialogue entre le client et le serveur. C'est pourquoi nous utilisons un certificat afin d'éviter de diffuser un faux certificat. Pour vérifier l'authenticité, le client va hashé les informations du certificat, et dans le même temps va déchiffrer la signature avec la clé publique de l'autorité de certification. Si les deux résultats sont égaux, alors le certificat est authentique et donc la clé publique du serveur aussi.

Le mot de passe ne doit pas être transmis.

Lors de l'authentification du client sur le réseau, ce dernier va envoyer le mot de passe. Si ce dernier est faux, cela ne pose pas de problème. Cependant, si celui-ci est correct, il ne doit pas transité sur le réseau, même s'il est dans un message chiffré avec une clé de session, par sécurité supplémentaire. Pour ce faire, le client va envoyer le hash du mot de passe, et le serveur va donc le comparer avec le réel mot de passe, mais en le hachant localement.

Ebauche(s) du protocole - version 1

Sous forme d'échange de messages à la Alice-Bob, en précisant les hypothèses (ex: connaissances préalables pour chaque rôle), propriétés "a priori" satisfaite

Connaissances préalables pour chaque rôle

Client :

- Clé publique de l'autorité de certification (CA)
- Son adresse MAC
- Son mot de passe

Borne d’accès :

- Certificat du Serveur (contient informations sur le serveur, ces même informations signées par la clé privé d’une autorité de certification (CA))
- Son SSID

Spécification du protocole et de ses propriétés en HLPSL - version 1

```
%% PROTOCOL: Accès WIFI
%%
%% CLIENT_BORNE_SERVEUR :
%%
%% 01. C  <--- {CertificatServeur.SSID} --- B --- --- S
%% 02. C  --- {addrMAC.nonceClient.pkC}_pkS ---> B --- --- S
%% 03. C  --- --- B --- {AddrMAC.nonceClient.pkC}_pkS ---> S
%% 04. C  --- --- B <--- {cleSession.nonceClient.nonceServeur}_pkC --- S
%% 05. C  <--- {cleSession.nonceClient.nonceServeur}_pkC --- B --- --- S
%% 06. C  --- {h(mdp).nonceServeur.addrMac}_cleSession ---> B --- --- S
%% 07. C  --- --- B --- {h(mdp).nonceServeur.addrMac}_cleSession ---> S
%%
%%    Cas où le mdp est valide :
%% 08a. C  --- --- B <--- {ok.addrIP.cleReseau}_cleSession --- S
%% 09a. C  <--- {ok.addrIP.cleReseau}_cleSession --- B --- --- S
%% 10a. C  --- {ok}_cleReseau ---> B --- --- S
%% 11a. C  --- --- B --- {ok}_cleReseau ---> S
%%
%%    Cas où le mdp est faux :
%% 08b. C  --- --- B <--- {faux}_cleSession --- S
%% 09b. C  <--- {faux}_cleSession --- B --- --- S
%%    Retour à l'étape 06.
%%
%%
%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Client
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role client (C, B, S: agent,
             PKc, PKs: public_key,
             SND, RCV: channel(dy))
played_by C def=

  local State: nat,
        AddrMAC, NonceClient, MdpClient, SSID, CertificatServeur, NonceServeur,
AddrIP: text,
        CleSession, CleReseau : symmetric_key

  const ok: text

  init State:=0

  transition

    02.  State=0 /\ RCV(SSID'.CertificatServeur') =|>
            State':=1 /\
            %%Verification du Certificat avec un CA por valider la PKs à partir du
certificat
            AddrMAC':=new() /\
            NonceClient':=new() /\
            secret(NonceClient', nonceClient, {C,S}) /\
            SND({AddrMAC'.NonceClient'.PKc}_PKs)

    06.  State=1 /\ RCV({CleSession'.NonceClient'.NonceServeur'}_PKc) =|>
            State':=2 /\
            MdpClient':=new() /\
            secret(MdpClient', mdpClient, {C,S}) /\
            SND({h(MdpClient').NonceServeur'}_CleSession')

    10.  State=2 /\ RCV({ok.AddrIP'.CleReseau'}_CleSession) =|>
            State':=3 /\
            SND({ok}_CleReseau')



end role
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Borne, initiant le protocole
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role borne (C, B, S: agent,
            PKc, PKs: public_key,
            SND, RCV: channel(dy))
played_by B def=

  local State: nat,
        AddrMAC, NonceClient, MdpClient, SSID, CertificatServeur, NonceServeur,
AddrIP: text,
        CleSession, CleReseau : symmetric_key

  init State:=0

  transition

    01.  State=0 /\ RCV(start) =|>
            State':=1 /\
            SSID':=new() /\
            CertificatServeur':=new() /\
            SND(SSID'.CertificatServeur')

    03.  State=1 /\ RCV({AddrMAC'.NonceClient'.PKc}_PKs) =|>
            State':=2 /\
            SND({AddrMAC'.NonceClient'.PKc}_PKs)

    05.  State=2 /\ RCV({CleSession'.NonceClient'.NonceServeur'}_PKc) =|>
            State':=3 /\
            request(S, C, c_s_CleSession, CleSession') /\
            SND({CleSession'.NonceClient'.NonceServeur'}_PKc)

    07.  State=3 /\ RCV({h(MdpClient').NonceServeur'}_CleSession') =|>
            State':=4 /\
            SND({h(MdpClient').NonceServeur'}_CleSession')

    09.  State=4 /\ RCV({ok.AddrIP'.CleReseau'}_CleSession) =|>
            State':=5 /\
            SND({ok.AddrIP'.CleReseau'}_CleSession)

    11.  State=5 /\ RCV({ok.AddrIP}_CleReseau') =|>
            State':=6 /\
            SND({ok.AddrIP}_CleReseau')

end role
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Serveur
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role serveur (C, B, S: agent,
            PKc, PKs: public_key,
            SND, RCV: channel(dy))
played_by S def=

  local State: nat,
        NonceServeur, AddrIP, AddrMAC, NonceClient, MdpClient, MdpReseau: text,
        CleSession, CleReseau : symmetric_key

  const ok, mdpReseau: text

  init State:=0

  transition

    4.  State=0 /\ RCV({AddrMAC'.NonceClient'.PKc}_PKs) =|>
            State':=1 /\
            CleSession':=new() /\
            NonceServeur':=new() /\
            secret(NonceServeur', nonceServeur, {C,S}) /\
            secret(CleSession', cleSession, {C,S}) /\
            witness(C, S, c_s_CleSession, CleSession') /\
            SND({CleSession'.NonceClient'.NonceServeur'}_PKc)

    8.  State=1 /\ RCV({h(MdpClient').NonceServeur'}_CleSession) =|>
            State':=2 /\
            equal(h(MdpClient'),h(mdpReseau)) /\
            AddrIP':=new() /\
            CleReseau':=new() /\
            secret(CleReseau', cleReseau, {C,S}) /\
            SND({ok.AddrIP'.CleReseau'}_CleSession)

    12.  State=2 /\ RCV({ok.AddrIP}_CleReseau') =|>
            State':=3




end role
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Session
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(C, B, S: agent, PKc, PKs: public_key) def=

  local SC, RC, SB, RB, SS, RS: channel(dy)

  composition

        client(C,B,S,PKc,PKs,SC,RC) /\
        borne(C,B,S,PKc,PKs,SB,RB) /\
        serveur(C,B,S,PKc,PKs,SS,RS)
end role


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du Scenario
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=

    const c, b, s: agent,
          pkc, pks, pki: public_key,
          cleSession, cleReseau : symmetric_key,
          c_s_CleSession, nonceClient, nonceServeur : protocol_id,
          mdpClient : text,
          h : hash_func

    intruder_knowledge = {c, b, s, pkc, pki, inv(pki), h}

    composition

        session(c,b,s,pkc,pks)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition des Propriétés à vérifier
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal
    secrecy_of mdpClient
    secrecy_of nonceClient
    secrecy_of nonceServeur
    secrecy_of cleSession
    secrecy_of cleReseau
    authentication_on c_s_CleSession
end goal


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% lancement du rôle principal
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

environment()
```

Résultats des analyses avec AVISPA - version 1

```
%% Translation of wifi.hlpsl
%% IF output in ./wifi.if
%% Constraint Logic-based ATtack SEarcher (CL-ATSE) Version 2.5-18 (2012-septembre-
26).

SUMMARY
  UNSAFE

DETAILS
  TYPED MODEL

PROTOCOL
  wifi.if

GOAL
  Authentication attack on (s,c,c_s_CleSession,CleSession(9))

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 8 states
  Reachable  : 4 states
  Translation: 0.02 seconds
  Computation: 0.00 seconds

ATTACK TRACE
 i -> (c,3):  SSID(1).CertificatServeur(1)
 (c,3) -> i:  {n1(AddrMAC).n1(NonceClient).pkc}_pks
              & Secret(n1(NonceClient),(),set_120);  Add c to set_120;
              & Add s to set_120;
              & Built from step_0

 i -> (b,4):  start
 (b,4) -> i:  n7(SSID).n7(CertificatServeur)
              & Built from step_3

 i -> (b,4):  {n1(AddrMAC).n1(NonceClient).pkc}_pks
 (b,4) -> i:  {n1(AddrMAC).n1(NonceClient).pkc}_pks
              & Built from step_4

 i -> (b,4):  {CleSession(9).NonceClient(9).NonceServeur(9)}_pkc
 (b,4) -> i:  {CleSession(9).NonceClient(9).NonceServeur(9)}_pkc
              & Request(s,c,c_s_CleSession,CleSession(9));
              & Built from step_5
```

On remarque alors que les secrets de nos variables mdpClient, nonceClient, nonceServeur, cleSession, cleReseau sont bel et bien gardés puisque l'attaquant ne peut les récupérer.

Cependant, on remarque bien que l'intrus peut demander à générer une nouvelle clé de session à un utilisateur qui est en train de se connecter.

Ce genre d'attaque oblige un timing parfait de l'attaquant car cette clé de session est très temporaire. D'autre part, cela peut être bloqué directement dans le software du serveur d'authentification. En effet, si une connexion avec un client est ouverte, alors on ne lui génère pas de nouvelle clé de session.

C’est pourquoi en parallèle nous avons pensé à une deuxième version “facile” du protocole. Cette fois-ci, le serveur a en sa possession une liste de clients connus. Cela permet donc d’éviter qu’un attaquant inconnu ne puisse effectuer cette attaque.

Ebauche(s) du protocole - version 2

Sous forme d'échange de messages à la Alice-Bob, en précisant les hypothèses (ex: connaissances préalables pour chaque rôle), propriétés "a priori" satisfaite

Connaissances préalables pour chaque rôle

Client :

- Clé publique de l'autorité de certification (CA)
- Son adresse MAC
- Son mot de passe

Borne d’accès :

- Certificat du Serveur (contient informations sur le serveur, ces même informations signées par la clé privé d’une autorité de certification (CA))
- Son SSID

Serveur :

- Une liste de clients connus

Spécification du protocole et de ses propriétés en HLPSL - version 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Serveur
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role serveur (C, B, S: agent,
            PKc, PKs: public_key,
            SND, RCV: channel(dy),
            Clients: nat set)
played_by S def=


    4.  State=0 /\ RCV({IdClient.AddrMAC'.NonceClient'.PKc}_PKs) /\
        in(IdClient, Clients) =|> %% Verifie si c'est un client connu par le serveur
            State':=1 /\
            CleSession':=new() /\
            NonceServeur':=new() /\
            secret(NonceServeur', nonceServeur, {C,S}) /\
            secret(CleSession', cleSession, {C,S}) /\
            witness(C, S, c_s_CleSession, CleSession') /\
            SND({CleSession'.NonceClient'.NonceServeur'}_PKc)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Session
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(C, B, S: agent, PKc, PKs: public_key, Clients: nat set) def=

  local SC, RC, SB, RB, SS, RS: channel(dy)

  composition

        client(C,B,S,PKc,PKs,SC,RC) /\
        borne(C,B,S,PKc,PKs,SB,RB) /\
        serveur(C,B,S,PKc,PKs,SS,RS,Clients)
end role


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du Scenario
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=

    local Clients: nat set

    const c, b, s: agent,
          pkc, pks, pki: public_key,
          cleSession, cleReseau : symmetric_key,
          c_s_CleSession, nonceClient, nonceServeur : protocol_id,
          mdpClient : text,
          h : hash_func

    %% Ensemble des clients autorises a se connecter au serveur
    %% (Seul le serveur connait cet ensemble)
    init Clients := {1,2,3}

    intruder_knowledge = {c, b, s, pkc, pks, pki, inv(pki), h}


    composition

        session(c,b,s,pkc,pks,Clients)

end role
```

La seule différence avec la première version réside dans le fait que le serveur connaît “Clients” qui est une liste de plusieurs clients autorisés à pouvoir se connecter.
Cette liste variable est initiée dans l’environnement et retransmis au serveur par le rôle “Session”.

Résultats des analyses avec AVISPA - version 2

```
%% Translation of wifi.hlpsl
%% IF output in ./wifi.if
%% Constraint Logic-based ATtack SEarcher (CL-ATSE) Version 2.5-18 (2012-septembre-
26).

SUMMARY
  SAFE

DETAILS
  BOUNDED NUMBER OF SESSIONS
  TYPED MODEL

PROTOCOL
  wifi.if

GOAL
  As specified

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 6 states
  Reachable  : 2 states
  Translation: 0.03 seconds
  Computation: 0.00 seconds
```

On remarque alors que notre problème sur la clé de session a disparu. En effet, si un attaquant n'est pas autorisé à communiquer avec le serveur, alors il ne peut initier de nouvelle clé de session.

Cependant, cette méthode avec une liste de clients déjà connus est peut-être une contrainte trop forte pour pouvoir être exploitée dans un large domaine.

Pour ce faire, le client communique son identifiant unique.

Nous avons donc décidé de garder l'architecture de la première version, à savoir que n'importe quel client puisse se connecter, tout en évitant l'attaque sur la clé de session.

Ebauche(s) du protocole - version 3

Sous forme d'échange de messages à la Alice-Bob, en précisant les hypothèses (ex: connaissances préalables pour chaque rôle), propriétés "a priori" satisfaite

Connaissances préalables pour chaque rôle

Client :

- Clé publique de l'autorité de certification (CA)
- Son adresse MAC
- Son mot de passe

Borne d’accès :

- Certificat du Serveur (contient informations sur le serveur, ces même informations signées par la clé privé d’une autorité de certification (CA))
- Son SSID

Serveur :

- Une liste de clients en train de s’authentifier (ClientsOnConnecting)

Spécification du protocole et de ses propriétés en HLPSL - version 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Serveur
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role serveur (C, B, S: agent,
              PKc, PKs: public_key,
              SND, RCV: channel(dy),
              ClientsOnConnecting: nat set)
played_by S def=

  local State,IdClient: nat,
        NonceServeur, AddrIP, AddrMAC, NonceClient, MdpClient, MdpReseau: text,
        CleSession, CleReseau : symmetric_key

  const ok, mdpReseau: text

  init State:=0

  transition

    1.  State=0 /\ RCV({IdClient.AddrMAC'.NonceClient'.PKc}_PKs) /\
        not(in(IdClient, ClientsOnConnecting)) =|> %% Verifie si c'est un client
connu par le serveur
              State':=1 /\
              ClientsOnConnecting' := cons(IdClient,ClientsOnConnecting) /\
              CleSession':=new() /\
              NonceServeur':=new() /\
              secret(NonceServeur', nonceServeur, {C,S}) /\
              secret(CleSession', cleSession, {C,S}) /\
              witness(C, S, c_s_CleSession, CleSession') /\
              SND({CleSession'.NonceClient'.NonceServeur'}_PKc)

    2.  State=1 /\ RCV({IdClient.h(MdpClient').NonceServeur'}_CleSession) /\
        in(IdClient, ClientsOnConnecting) =|>
              State':=2 /\
              equal(h(MdpClient'),h(mdpReseau)) /\
              AddrIP':=new() /\
              CleReseau':=new() /\
              secret(CleReseau', cleReseau, {C,S}) /\
              SND({ok.AddrIP'.CleReseau'}_CleSession)

    3.   State=2 /\ RCV({IdClient.ok.AddrIP}_CleReseau') /\
        in(IdClient, ClientsOnConnecting)=|>
              State':=3 /\
              ClientsOnConnecting' := delete(IdClient, ClientsOnConnecting)

end role
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du rôle Session
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(C, B, S: agent, PKc, PKs: public_key, ClientsOnConnecting: nat set)
def=

  local SC, RC, SB, RB, SS, RS: channel(dy)

  composition

        client(C,B,S,PKc,PKs,SC,RC) /\
        borne(C,B,S,PKc,PKs,SB,RB) /\
        serveur(C,B,S,PKc,PKs,SS,RS,ClientsOnConnecting)
end role


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% définition du Scenario
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=

    local ClientsOnConnecting: nat set

    const c, b, s: agent,
          pkc, pks, pki: public_key,
          cleSession, cleReseau : symmetric_key,
          c_s_CleSession, nonceClient, nonceServeur : protocol_id,
          mdpClient : text,
          h : hash_func

    %% Ensemble des ClientsOnConnecting actuellement en cours de connexion
    %% (Seul le serveur connait cet ensemble)
    init ClientsOnConnecting := {}

    intruder_knowledge = {c, b, s, pkc, pks, pki, inv(pki), h}


    composition

        session(c,b,s,pkc,pks,ClientsOnConnecting)

end role
```

On remarque donc ici que le serveur, comparé à la version deux qui connaissait une liste de clients autorisés, connaît cette fois-ci une liste de client en train de se connecter.

Pour ce faire, le client communique à chaque fois son IdClient. Celui-ci encrypté à chaque fois, l'attaquant ne peut le deviner.

Puis lorsque une demande d'authentification arrive au serveur, ce dernier vérifie s'il n'est pas déjà en train de s'authentifier. Si c'est bon, alors on procède à la suite du protocole. De ce fait, à chaque nouvelle action du serveur, il vérifiera que le client est bien en train de se connecter. Puis lorsque l'authentification du client est réussie, il retire son id de sa liste.

C'est cette différence qui permet qu'un attaquant ne peut initier une nouvelle clé de session, est donc de compromettre la sécurité.

Résultats des analyses avec AVISPA - version 3

```
%% Translation of wifi.hlpsl
%% IF output in ./wifi.if
%% Constraint Logic-based ATtack SEarcher (CL-ATSE) Version 2.5-18 (2012-septembre-
26).

SUMMARY
  SAFE

DETAILS
  BOUNDED NUMBER OF SESSIONS
  TYPED MODEL

PROTOCOL
  wifi.if

GOAL
  As specified

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 9 states
  Reachable  : 2 states
  Translation: 0.03 seconds
  Computation: 0.00 seconds
```

Effectivement, tous ces correctifs ont permis d’éviter les attaques sur la clé de session.