www.avispa-project.org

IST-2001-39252

Automated Validation of Internet Security Protocols and Applications

Deliverable D6.2: Specification of the Problems in the High-Level Specification Language

Abstract

This document presents the specifications of the protocols and security problems that we have modelled in the HLPSL and analysed with the AVISPA tool. This set of protocols is a large subset of those described in Deliverable 6.1. For each of the protocols, we describe its purpose, the message exchanges in the Alice&Bob notation, the corresponding security problems, and any attacks found, and we also give the actual HLPSL code. Where appropriate, we add further explanations and comments.

Deliverable details

Deliverable version: *v1.0*
Date of delivery: *31.01.2005*
Classification: *public*

Person-months required: *14*
Due on: *31.12.2004*
Total pages: *295*

Project details

Start date: *January 1st, 2003*
Duration: *30 months*
Project Coordinator: *Alessandro Armando*
Partners: *Università di Genova, INRIA Lorraine, ETH Zürich, Siemens AG*

Project funded by the European Community under the *Information Society Technologies* Programme (1998-2002)

Contents

Part I
Introduction

The goal of this deliverable is to give the specifications of the protocols and security problems that we have modelled in the High-Level Protocol Specification Language (HLPSL). We have specified a large number of protocols, including several variants of generic protocols like Kerberos and EAP, and for some protocols that can be attacked, for instance the H.530 protocol, we give both the original and a fixed version. For each of these protocols, we formulate between one and seven security problems. All protocols are presented using the following scheme.

- The section name gives the common name of the protocol (or protocol suite).
- For each variant or version specified, if any, there is a corresponding subsection.
- Then there is a series of subsubsections:

  Protocol Purpose

  states the overall purpose of the protocol.

  Definition Reference

  points to the official specification(s) and further documentation.

  Model Authors

  gives the author(s) of the HLPSL specification and its documentation.

  Alice&Bob style

  describes the message flow in the well-known semi-formal way.

  Model Limitations

  lists and explains those simplifications and other deviations (with respect to the official protocol reference specification) that were carried out during the modelling process, and which may have a negative effect on the outcome of the analysis, i.e. may lead to attacks missed. This typically includes abstractions from certain notions and details like time, message format, concrete algorithms, protocol options not considered, algebraic properties, etc.

Problems considered:

gives the number of problems, i.e. security goals, tackled for the given protocol. This is the number of authentications, plus one if there are secrecy goals (which all count as one), plus any other goals expressed using the authentication mechanism. Then follows a list of the problems in an abstract semi-formal notation similar to the one typically given in the goals section at the end of the actual HLPSL code.

Attacks Found

states if the back-ends found one or more attacks, and if so, gives the attack trace and/or a verbal description which properties are not satisfied and why.

Further Notes

contains any other issues the modeller(s) decided to point out, e.g. further explanations, justifications, comments on problems with the tools, etc.

HLPSL Specification

gives the plain HLPSL source of the specification.

Part II
The IETF Protocols

1 AAA Mobile IP

Protocol Purpose

This document specifies a Diameter application that allows a Diameter server to authenticate, authorise and collect accounting information for Mobile IPv4 services rendered to a mobile node.

Definition Reference

- [Per03, CJP03]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Paul Hankes Drielsma, Information Security Group, ETH Zürich, December 2003
- Sebastian Mödersheim, Information Security Group, ETH Zürich, January 2004
- Luca Compagna, AI-Lab DIST, University of Genova, December 2004

Alice&Bob style

```
1. FA   -> MN:   FA,N_FA
2. MN   -> FA:   N_FA,MN,AAAH,
                 {N_FA,MN,AAAH}_K_MnAAAH
3. FA   -> AAAL: N_FA,MN,AAAH,
                 {N_FA,MN,AAAH}_K_MnAAAH
4. AAAL -> AAAH: N_FA,MN,AAAH,
                 {N_FA,MN,AAAH}_K_MnAAAH
5. AAAH -> HA:   MN,
                 {K_MnHa,K_FaHa}_KAAAHHa,
                 {K_MnFa,K_MnHa}_K_MnAAAH,
                 {MN,
                  {K_MnHa,K_FaHa}_KAAAHHa,
                  {K_MnFa,K_MnHa}_K_MnAAAH
                 }_K_AAAHHa
6. HA   -> AAAH: {K_MnFa,K_MnHa}_K_MnAAAH,
                 {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa,
                 {{K_MnFa,K_MnHa}_K_MnAAAH,
                  {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa
                 }_K_AAAHHa
```

```
 7. AAAH -> AAAL: N_FA,
                  {K_MnFa,K_FaHa}_K_AAAHAAAL,
                  {K_MnFa,K_MnHa}_K_MnAAAH,
                  {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa,
                  {N_FA,
                   {K_MnFa,K_FaHa}_K_AAAHAAAL,
                   {K_MnFa,K_MnHa}_K_MnAAAH,
                   {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa
                  }_K_AAAHAAAL
 8. AAAL -> FA:   N_FA,
                  {K_MnFa,K_FaHa}_K_FaAAAL,
                  {K_MnFa,K_MnHa}_K_MnAAAH,
                  {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa,
                  {N_FA,
                   {K_MnFa,K_FaHa}_K_FaAAAL,
                   {K_MnFa,K_MnHa}_K_MnAAAH,
                   {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa
                  }_K_FaAAAL
 9. FA   -> MN:   {K_MnFa,K_FaHa}_K_FaAAAL,
                  {K_MnFa,K_MnHa}_K_MnAAAH,
                  {{K_MnFa,K_MnHa}_K_MnAAAH}_K_MnHa
```

Problems considered: 7

- secrecy of `K_MnFa, K_FaHa, K_MnFa`
- `AAA_MIP_FA` weakly authenticates `AAA_MIP_AAAH` on `k_faha1`
- `AAA_MIP_FA` weakly authenticates `AAA_MIP_AAAH` on `k_mnfa1`
- `AAA_MIP_HA` weakly authenticates `AAA_MIP_AAAH` on `k_faha2`
- `AAA_MIP_HA` weakly authenticates `AAA_MIP_AAAH` on `k_mnha1`
- `AAA_MIP_MN` weakly authenticates `AAA_MIP_AAAH` on `k_mnha2`
- `AAA_MIP_MN` weakly authenticates `AAA_MIP_AAAH` on `k_mnfa2`

Attacks Found

```
 i      -> (mn,3): fa,fa
 (mn,3) -> i:      fa,mn,aaah,{fa,mn,aaah}k_mn_aaah
 i       -> (mn,3): {fa,mn,aaah}k_mn_aaah,{{fa,mn,aaah}k_mn_aaah}(mn,aaah)
```

In this type-flaw attack, the intruder replays the message `{fa,mn,aaah}k_mn_aaah` to the mobile node, which expects to receive a message of the form `{fa,NewKey}k_mn_aaah` where NewKey is the new key, which is thus matched with the pair of agent names `mn,aah`. Since the intruder knows these two agent names, he can also produce a message encrypted with this new key as required.

HLPSL Specification

```
role AAA_MIP_MN (MN, AAAH, FA : agent,
                 Snd, Rcv     : channel(dy),
                 K_MnAAAH     : symmetric_key) played_by MN def=

  local  S              : nat,
         K_MnFa,K_MnHa : symmetric_key

  init  S = 0

  knowledge(MN)={MN,AAAH,K_MnAAAH,FA}

  transition

   1. S=0 /\ Rcv(FA.FA) =|>
         Snd(FA.MN.AAAH.{FA.MN.AAAH}_K_MnAAAH)
      /\ S'=1

   2. S=1 /\ Rcv( {K_MnFa'.K_MnHa'}_K_MnAAAH.
                 {{K_MnFa'.K_MnHa'}_K_MnAAAH}_K_MnHa')
      =|>
         S'=2
      /\ wrequest(MN,AAAH,k_mnha2,K_MnHa')
      /\ wrequest(MN,AAAH,k_mnfa2,K_MnFa')

end role
```

```
role AAA_MIP_FA (FA,AAAL,AAAH,MN: agent,
                 Snd, Rcv: channel(dy),
                 K_FaAAAL: symmetric_key) played_by FA def=

  local  S             : nat,
    K_MnFa, K_FaHa     : symmetric_key,
    SignedRegReq       : {agent.(agent.agent)}_symmetric_key,
    KeyMnHaKeyMnFa     : {symmetric_key.symmetric_key}_symmetric_key,
    SignKeyMnHaKeyMnFa :
         {{symmetric_key.symmetric_key}_symmetric_key}_symmetric_key

  init S= 0

  knowledge(FA)={FA,AAAL,MN,K_FaAAAL}

  transition

   1. S=0 /\ Rcv(start) =|>
         Snd(FA.FA)
      /\ S'=1

   2. S=1 /\ Rcv(FA.MN.AAAH.SignedRegReq') =|>
         Snd(FA.MN.AAAH.SignedRegReq')
      /\ S'=2

   3. S=2 /\ Rcv( FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
                     KeyMnHaKeyMnFa'.SignKeyMnHaKeyMnFa'.
                 {FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
                     KeyMnHaKeyMnFa'.SignKeyMnHaKeyMnFa'}_K_FaAAAL)
      =|>    Snd(KeyMnHaKeyMnFa'.SignKeyMnHaKeyMnFa')
          /\ S'=3
          /\ wrequest(FA,AAAH,k_faha1,K_FaHa')
          /\ wrequest(FA,AAAH,k_mnfa1,K_MnFa')

end role
```

```
role AAA_MIP_AAAL (AAAL,AAAH,FA,MN: agent,
                  Snd, Rcv: channel(dy),
                  K_FaAAAL,K_AAAHAAAL: symmetric_key) played_by AAAL def=

  local  S               : nat,
    K_MnFa,K_FaHa        : symmetric_key,
    SignedRegReq         : {agent.(agent.agent)}_symmetric_key,
    KeyMnFaKeyMnHa       : {symmetric_key.symmetric_key}_symmetric_key,
    SignedKeyMnFaKeyMnHa :
           {{symmetric_key.symmetric_key}_symmetric_key}_symmetric_key

  init S = 0

  knowledge(AAAL)={AAAL,AAAH,FA,MN,K_FaAAAL,K_AAAHAAAL}

  transition

   1. S=0 /\ Rcv(FA.MN.AAAH.SignedRegReq') =|>
         Snd(FA.MN.AAAH. SignedRegReq')
      /\ S'=1

   2. S=1 /\ Rcv( FA.{K_MnFa'.K_FaHa'}_K_AAAHAAAL.
                     KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'.
                 {FA.{K_MnFa'.K_FaHa'}_K_AAAHAAAL.
                     KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'}_K_AAAHAAAL)
      =|>
         Snd( FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
                 KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'.
             {FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
                 KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'}_K_FaAAAL)
      /\ S'=2

end role
```

```
role AAA_MIP_AAAH (AAAH,AAAL,HA,FA,MN : agent,
           Snd, Rcv : channel(dy),
           K_MnAAAH,
           K_AAAHAAAL,
           KAAAHHa : symmetric_key) played_by AAAH def=
  local  S                     : nat,
         K_FaHa,K_MnHa,K_MnFa : symmetric_key (fresh)

  init S = 0

  knowledge(AAAH)={AAAH,AAAL,HA,FA,MN,K_MnAAAH,K_AAAHAAAL,KAAAHHa}

  transition

   1. S=0 /\ Rcv(FA.MN.AAAH.{FA.MN.AAAH}_K_MnAAAH) =|>
         Snd( MN.{K_MnHa'.K_FaHa'}_KAAAHHa.{K_MnFa'.K_MnHa'}_K_MnAAAH.
             {MN.{K_MnHa'.K_FaHa'}_KAAAHHa.{K_MnFa'.K_MnHa'}_K_MnAAAH}_KAAAHHa)
      /\ S'=1
      /\ witness(AAAH,FA,k_faha1,K_FaHa')
      /\ witness(AAAH,HA,k_faha2,K_FaHa')
      /\ witness(AAAH,FA,k_mnfa1,K_MnFa')
      /\ witness(AAAH,MN,k_mnfa2,K_MnFa')
      /\ witness(AAAH,MN,k_mnha2,K_MnHa')
      /\ witness(AAAH,HA,k_mnha1,K_MnHa')

   2. S=1 /\ Rcv( {K_MnFa.K_MnHa}_K_MnAAAH.
                  {{K_MnFa.K_MnHa}_K_MnAAAH}_K_MnHa.
                 {{K_MnFa.K_MnHa}_K_MnAAAH.
                  {{K_MnFa.K_MnHa}_K_MnAAAH}_K_MnHa}_KAAAHHa)
      =|>
         Snd( FA.{K_MnFa.K_FaHa}_K_AAAHAAAL.{K_MnFa.K_MnHa}_K_MnAAAH.
                {{K_MnFa.K_MnHa}_K_MnAAAH}_K_MnHa.
             {FA.{K_MnFa.K_FaHa}_K_AAAHAAAL.{K_MnFa.K_MnHa}_K_MnAAAH.
                {{K_MnFa.K_MnHa}_K_MnAAAH}_K_MnHa}_K_AAAHAAAL)
      /\ S'=2
      /\ secret(K_FaHa,FA)
      /\ secret(K_FaHa,HA)
      /\ secret(K_MnFa,FA)
      /\ secret(K_MnFa,MN)
      /\ secret(K_MnHa,MN)
      /\ secret(K_MnHa,HA)

end role
```

```
role AAA_MIP_HA (HA,AAAH,MN: agent,
                 Snd,Rcv: channel(dy),
                 K_AAAHHa: symmetric_key) played_by HA def=

  local  S                 : nat,
    K_MnFa,K_FaHa, K_MnHa : symmetric_key,
    KeyMnFaKeyMnHa        : {symmetric_key.symmetric_key}_symmetric_key

  init S = 0

  knowledge(HA)={HA,AAAH,MN,K_AAAHHa}

  transition

   1. S=0 /\ Rcv( MN.{K_MnHa'.K_FaHa'}_K_AAAHHa.KeyMnFaKeyMnHa'.
                 {MN.{K_MnHa'.K_FaHa'}_K_AAAHHa.KeyMnFaKeyMnHa'}_K_AAAHHa)

      =|>
         Snd( KeyMnFaKeyMnHa'.{KeyMnFaKeyMnHa'}_K_MnHa'.
             {KeyMnFaKeyMnHa'.{KeyMnFaKeyMnHa'}_K_MnHa'}_K_AAAHHa)
      /\ S'=1
      /\ wrequest(HA,AAAH,k_faha2,K_FaHa')
      /\ wrequest(HA,AAAH,k_mnha1,K_MnHa')
end role
```

```
role Session(MN,FA,AAAL,AAAH,HA: agent,
             Kmn3ah,Kfa3al,K3ah3al,Kha3ah: symmetric_key) def=

   local     MNs,MNr,
             FAs,FAr,
             Ls, Lr,
             Hs, Hr,
             HAs, HAr: channel(dy)

   composition

           AAA_MIP_MN(MN,AAAH,FA,MNs,MNr,Kmn3ah)

        /\ AAA_MIP_FA(FA,AAAL,AAAH,MN,FAs,FAr,Kfa3al)
        /\ AAA_MIP_AAAL(AAAL,AAAH,FA,MN,Ls,Lr,Kfa3al,K3ah3al)

        /\ AAA_MIP_AAAH(AAAH,AAAL,HA,FA,MN,Hs,Hr,Kmn3ah,K3ah3al,Kha3ah)

        /\ AAA_MIP_HA(HA,AAAH,MN,HAs,HAr,Kha3ah)
end role
```

```
role Environment() def=

  const k_mnha1, k_mnfa1, k_faha1                    : protocol_id,
        k_mnha2, k_mnfa2, k_faha2                    : protocol_id,
        mn, fa, aaal, aaah, ha                       : agent,
        k_mn_aaah, k_fa_aaal, k_aaah_aaal, k_ha_aaah : symmetric_key

  knowledge(i) = {mn,fa,aaal,aaah,ha}

  composition

        Session(mn,fa,aaal,aaah,ha,
                k_mn_aaah,k_fa_aaal,k_aaah_aaal,k_ha_aaah)

end role

goal

  secrecy_of K_MnFa, K_FaHa, K_MnFa
  AAA_MIP_FA weakly authenticates AAA_MIP_AAAH on k_faha1
  AAA_MIP_FA weakly authenticates AAA_MIP_AAAH on k_mnfa1
  AAA_MIP_HA weakly authenticates AAA_MIP_AAAH on k_faha2
  AAA_MIP_HA weakly authenticates AAA_MIP_AAAH on k_mnha1
  AAA_MIP_MN weakly authenticates AAA_MIP_AAAH on k_mnha2
  AAA_MIP_MN weakly authenticates AAA_MIP_AAAH on k_mnfa2

end goal

Environment()
```

2 H.530: Symmetric security procedures for H.323 mobility in H.510

2.1 Original version

Protocol Purpose

Establish an authenticated (Diffie-Hellman) shared-key between a mobile terminal (MT) and a visited gate-keeper (VGK), who do not know each other in advance, but who have a "mutual friend", an authentication facility (AuF) in the home domain of MT.

Definition Reference

http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.530 (original version without "corrigendum")

Model Authors

Sebastian Mödersheim, ETH Zürich

Alice&Bob style

```
  Macros
  M1 = MT,VGK,NIL,CH1,exp(G,X)
  M2 = M1,F(ZZ,M1),VGK,exp(G,X) XOR exp(G,Y)
  M3 = VGK,MT,F(ZZ,VGK),F(ZZ,exp(G,X) XOR exp(G,Y))
  M4 = VGK,MT,CH1,CH2,exp(G,Y),F(ZZ,exp(G,X) XOR exp(G,Y)),F(ZZ,VGK)
  M5 = MT,VGK,CH2,CH3
  M6 = VGK,MT,CH3,CH4
------------------------------------------------------------------
  1. MT  -> VGK : M1,F(ZZ,M1)
  2. VGK -> AuF : M2,F(ZZ_VA,M2)
  3. AuF -> VGK : M3,F(ZZ_VA,M3)
  4. VGK -> MT  : M4,F(exp(exp(G,X),Y),M4)
  5. MT  -> VGK : M5,F(exp(exp(G,X),Y),M5)
  6. VGK -> MT  : M6,F(exp(exp(G,X),Y),M6)
```

Problems considered: 3

- `MobileTerminal` authenticates `VisitedGateKeeper` on `key`
- `VisitedGateKeeper` authenticates `MobileTerminal` on `key1`
- secrecy of `Key`

Attacks Found

A replay attack, as AuF's reply to the authentication request from VGK does not contain enough information that VGK can read. The attack works by first observing a session between honest agents and then replaying messages from this session to VGK, posing both as MT and AuF. Use option sessco to find this attack with OFMC. Another attack recently discovered with OFMC is based on the fact that VGK cannot distinguish messages (2) and (3).

Further Notes

The fixed version, also included in this library, is not vulnerable to the attacks.

In the original protocol description there is a chain of intermediate hops between VGK and AuF, where the length of this chain depends on the concrete setting. Each of the hops shares a symmetric key with its neighbouring hops and forwards messages in the chain decrypting and re-encrypting them accordingly. All the hops and AuF have to be honest, since if one of them modifies messages or inserts new ones, the protocol trivially cannot provide authentication. In our formalisation we have modelled no intermediate hops (so VGK and AuF directly share a key) and a simple reduction proof shows that all attacks possible in a setting with an arbitrary number of intermediate hops can be simulated in our model with no intermediate hops. Note, however, that it is not possible to take this idea further and "merge" an honest VGK with AuF, as demonstrated by the attacks we have discovered where the intruder eavesdrops and replays messages (that he cannot decrypt) exchanged between VGK and AuF.

HLPSL Specification

```
role MobileTerminal (
    MT,VGK,AuF : agent,
    SND,RCV    : channel(dy),
    F          : function,
    ZZ         : symmetric_key,
    NIL,G       : text)
played_by MT def=

  local
    State       : nat,
    X,CH1,CH3   : text (fresh),
    CH2,CH4     : text,
    GY,Key      : message

  init
    State = 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State' = 1 /\ SND(MT.VGK.NIL.CH1'.exp(G,X').F(ZZ.MT.VGK.NIL.CH1'.exp(G,X')))

 2. State  = 1 /\ RCV(VGK.MT.CH1.CH2'.GY'.
                        F(ZZ.xor(exp(G,X),GY')).
                        F(ZZ.VGK).
                        F(exp(GY',X).VGK.MT.CH1.CH2'.GY'.
                          F(ZZ.xor(exp(G,X),GY')).
                          F(ZZ.VGK)))
              =|>
    State' = 2 /\ Key'=exp(GY',X)
               /\ SND(MT.VGK.CH2'.CH3'.F(Key'.MT.VGK.CH2'.CH3'))
               /\ witness(MT,VGK,key1,Key')

 3. State  = 2 /\ RCV(VGK.MT.CH3.CH4'.F(Key.VGK.MT.CH3.CH4')) =|>
    State' = 3 /\ request(MT,VGK,key,Key)
               /\ secret(Key,VGK)
               /\ secret(Key,AuF) % this is not actually necessary, since
                                  % AuF must be honest anyway...

end role
```

```
role VisitedGateKeeper (
    MT,VGK,AuF : agent,
    SND,RCV    : channel(dy),
    F           : function,
    ZZ_VA       : symmetric_key,
    NIL,G       : text)
played_by VGK def=

  local
    State           : nat,
    GX,Key,Key1     : message,
    FM1,FM2,FM3,M2  : message,
    Y,CH2,CH4       : text (fresh),
    CH1,CH3         : text

  init
    State = 0

  transition

  1. State = 0 /\ RCV(MT.VGK.NIL.CH1'.GX'.FM1') =|>
     State'= 1 /\ Key'=exp(GX',Y')
               /\ M2' = MT.VGK.NIL.CH1'.GX'.FM1'.VGK.xor(GX',exp(G,Y'))
               /\ SND(M2'.F(ZZ_VA.M2'))
               /\ witness(VGK,MT,key,Key')

  2. State = 1 /\ RCV(VGK.MT.FM2'.FM3'.F(ZZ_VA.VGK.MT.FM2'.FM3')) =|>
     State'= 2 /\ SND(  VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'.
                  F(Key.VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'))

  3. State = 2 /\ RCV(MT.VGK.CH2.CH3'.F(Key.MT.VGK.CH2.CH3')) =|>
     State'= 3 /\ SND(VGK.MT.CH3'.CH4'.F(Key.VGK.MT.CH3'.CH4'))
               /\ request(VGK,MT,key1,Key)
               /\ secret(Key,MT)

end role
```

```
role AuthenticationFacility(
    MT,VGK,AuF : agent,
    SND,RCV    : channel(dy),
    F          : function,
    ZZ,ZZ_VA   : symmetric_key,
    NIL,G      : text)
played_by AuF def=

  local
    State         : nat,
    GX,GY         : message,
    CH1           : text

  init
    State = 0

  transition

  1. State = 0 /\ RCV(        MT.VGK.NIL.CH1'.GX'.
                         F(ZZ.MT.VGK.NIL.CH1'.GX').
                              VGK.xor(GX',GY').
                      F(ZZ_VA.MT.VGK.NIL.CH1'.GX'.
                         F(ZZ.MT.VGK.NIL.CH1'.GX').
                              VGK.xor(GX',GY')))  =|>

     State'= 1 /\ SND(        VGK.MT.F(ZZ.VGK).F(ZZ.xor(GX',GY')).
                      F(ZZ_VA.VGK.MT.F(ZZ.VGK).F(ZZ.xor(GX',GY'))))

end role
```

```
role Session(
    MT,VGK,AuF : agent,
    SND,RCV    : channel (dy),
    F          : function,
    ZZ,ZZ_VA   : symmetric_key,
    NIL,G      : text)
def=

  composition
    MobileTerminal(MT,VGK,AuF,SND,RCV,F,ZZ,NIL,G)
 /\ AuthenticationFacility(MT,VGK,AuF,SND,RCV,F,ZZ,ZZ_VA,NIL,G)
 /\ VisitedGateKeeper(MT,VGK,AuF,SND,RCV,F,ZZ_VA,NIL,G)

end role
```

```
role Environment() def=

  local
    S1,R1,S2,R2,
    S3,R3,S4,R4: channel (dy)

  const
    a,b,auf      : agent,
    f            : function,
    key,key1     : protocol_id,
    zz_a_auf,zz_b_auf,zz_i_auf
                 : symmetric_key,
    nil,g        : text

  knowledge(i) = {a,b,auf,f,zz_i_auf}

  composition
    Session(a,b,auf,S1,R1,f,zz_a_auf,zz_b_auf,nil,g)
 /\ Session(b,a,auf,S2,R2,f,zz_b_auf,zz_a_auf,nil,g)
 /\ Session(i,b,auf,S3,R3,f,zz_i_auf,zz_b_auf,nil,g)
 /\ Session(a,i,auf,S4,S4,f,zz_a_auf,zz_i_auf,nil,g)

end role
```

```
goal

  MobileTerminal    authenticates VisitedGateKeeper on key
  VisitedGateKeeper authenticates MobileTerminal    on key1
  secrecy_of Key

end goal
```

```
Environment()
```

2.2 Fixed version

Protocol Purpose

Establish an authenticated (Diffie-Hellman) shared-key between a mobile terminal (MT) and a visited gate-keeper (VGK), who do not know each other in advance, but who have a "mutual friend", an authentication facility (AuF) in the home domain of MT.

Definition Reference

http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.530 (with "corrigendum")

Model Authors

Sebastian Mödersheim, ETH Zürich

Alice&Bob style

```
  Macros
  M1 = MT,VGK,NIL,CH1,exp(G,X)
  M2 = M1,F(ZZ,M1),VGK,exp(G,X) XOR exp(G,Y)
  M3 = VGK,MT,F(ZZ,VGK),F(ZZ,exp(G,X) XOR exp(G,Y)),
       exp(G,X) XOR exp(G,Y)  %%% this is the very term added
                              %%% to fix the protocol
  M4 = VGK,MT,CH1,CH2,exp(G,Y),F(ZZ,exp(G,X) XOR exp(G,Y)),F(ZZ,VGK)
  M5 = MT,VGK,CH2,CH3
  M6 = VGK,MT,CH3,CH4
-------------------------------------------------------------------
  1. MT  -> VGK : M1,F(ZZ,M1)
  2. VGK -> AuF : M2,F(ZZ_VA,M2)
  3. AuF -> VGK : M3,F(ZZ_VA,M3)
  4. VGK -> MT  : M4,F(exp(exp(G,X),Y),M4)
  5. MT  -> VGK : M5,F(exp(exp(G,X),Y),M5)
  6. VGK -> MT  : M6,F(exp(exp(G,X),Y),M6)
```

Problems considered: 3

- `MobileTerminal` authenticates `VisitedGateKeeper` on `key`

- `VisitedGateKeeper` authenticates `MobileTerminal` on `key1`
- secrecy of `Key`

Attacks Found

None

Further Notes

This is the fixed version.

HLPSL Specification

```
role MobileTerminal (
    MT,VGK,AuF : agent,
    SND,RCV    : channel(dy),
    F          : function,
    ZZ         : symmetric_key,
    NIL,G      : text)
played_by MT def=

  local
    State       : nat,
    X,CH1,CH3   : text (fresh),
    CH2,CH4     : text,
    GY,Key      : message

  init
    State = 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State' = 1 /\ SND(MT.VGK.NIL.CH1'.exp(G,X').F(ZZ.MT.VGK.NIL.CH1'.exp(G,X')))

 2. State  = 1 /\ RCV(VGK.MT.CH1.CH2'.GY'.
                      F(ZZ.xor(exp(G,X),GY')).
                      F(ZZ.VGK).
                      F(exp(GY',X).VGK.MT.CH1.CH2'.GY'.
                        F(ZZ.xor(exp(G,X),GY')).
                        F(ZZ.VGK)))
              =|>
    State' = 2 /\ Key'=exp(GY',X)
               /\ SND(MT.VGK.CH2'.CH3'.F(Key'.MT.VGK.CH2'.CH3'))
               /\ witness(MT,VGK,key1,Key')

 3. State  = 2 /\ RCV(VGK.MT.CH3.CH4'.F(Key.VGK.MT.CH3.CH4')) =|>
    State' = 3 /\ request(MT,VGK,key,Key)
               /\ secret(Key,VGK)
               /\ secret(Key,AuF) % this is not actually necessary, since
                                  % AuF must be honest anyway...

end role
```

```
role VisitedGateKeeper (
    MT,VGK,AuF : agent,
    SND,RCV    : channel(dy),
    F          : function,
    ZZ_VA      : symmetric_key,
    NIL,G      : text)
played_by VGK def=

  local
    State          : nat,
    GX,Key         : message,
    FM1,FM2,FM3,M2 : message,
    Y,CH2,CH4      : text (fresh),
    CH1,CH3        : text

  init
    State = 0

  transition

  1. State = 0 /\ RCV(MT.VGK.NIL.CH1'.GX'.FM1') =|>
```

```
     State'= 1 /\ Key'=exp(GX',Y')
               /\ M2' = MT.VGK.NIL.CH1'.GX'.FM1'.VGK.xor(GX',exp(G,Y'))
               /\ SND(M2'.F(ZZ_VA.M2'))
               /\ witness(VGK,MT,key,Key')

  2. State = 1 /\ RCV(VGK.MT.FM2'.FM3'.
                      xor(GX,exp(G,Y)).
                      F(ZZ_VA.VGK.MT.FM2'.FM3'.xor(GX,exp(G,Y)))) =|>
     State'= 2 /\ SND(  VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'.
                  F(Key.VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'))

  3. State = 2 /\ RCV(MT.VGK.CH2.CH3'.F(Key.MT.VGK.CH2.CH3')) =|>
     State'= 3 /\ SND(VGK.MT.CH3'.CH4'.F(Key.VGK.MT.CH3'.CH4'))
               /\ request(VGK,MT,key1,Key)
               /\ secret(Key,MT)

end role
```

```
role AuthenticationFacility(
    MT,VGK,AuF : agent,
    SND,RCV    : channel(dy),
    F          : function,
    ZZ,ZZ_VA   : symmetric_key,
    NIL,G      : text)
played_by AuF def=

  local
    State         : nat,
    GX,GY         : message,
    CH1           : text

  init
    State = 0

  transition

  1. State = 0 /\ RCV(        MT.VGK.NIL.CH1'.GX'.
                         F(ZZ.MT.VGK.NIL.CH1'.GX').
                              VGK.xor(GX',GY').
```

```
                      F(ZZ_VA.MT.VGK.NIL.CH1'.GX'.
                         F(ZZ.MT.VGK.NIL.CH1'.GX').
                              VGK.xor(GX',GY')))  =|>

     State'= 1 /\ SND(        VGK.MT.F(ZZ.VGK).F(ZZ.xor(GX',GY')).xor(GX',GY').
                      F(ZZ_VA.VGK.MT.F(ZZ.VGK).F(ZZ.xor(GX',GY')).xor(GX',GY')))

end role
```

```
role Session(
    MT,VGK,AuF : agent,
    SND,RCV    : channel (dy),
    F          : function,
    ZZ,ZZ_VA   : symmetric_key,
    NIL,G      : text)
def=

  composition
    MobileTerminal(MT,VGK,AuF,SND,RCV,F,ZZ,NIL,G)
 /\ AuthenticationFacility(MT,VGK,AuF,SND,RCV,F,ZZ,ZZ_VA,NIL,G)
 /\ VisitedGateKeeper(MT,VGK,AuF,SND,RCV,F,ZZ_VA,NIL,G)

end role
```

```
role Environment() def=

  local
    S1,R1,S2,R2,S3,R3: channel (dy)

  const
    a,b,auf                      : agent,
    f                            : function,
    key, key1                    : protocol_id,
    zz_a_auf,zz_b_auf,zz_i_auf : symmetric_key,
    nil,g                        : text

  knowledge(i) = {a,b,auf,f,zz_i_auf}
  composition
    Session(a,b,auf,S1,R1,f,zz_a_auf,zz_b_auf,nil,g)
 /\ Session(i,b,auf,S2,R2,f,zz_i_auf,zz_b_auf,nil,g)
 /\ Session(a,i,auf,S3,R3,f,zz_a_auf,zz_i_auf,nil,g)

end role
```

```
goal

  MobileTerminal    authenticates VisitedGateKeeper on key
  VisitedGateKeeper authenticates MobileTerminal    on key1
  secrecy_of Key

end goal
```

```
Environment()
```

3 SPKM-LIPKEY

3.1 Known initiator

Protocol Purpose

Provide a secure channel between a client and server, authenticating the client with a password, and a server with a public key certificate.

Definition Reference

RFC 2847, http://www.faqs.org/rfcs/rfc2847.html

Model Authors

- Boichut Yohan, LIFC-INRIA Besancon, May 2004
- Sebastian Mödersheim, ETH Zürich, January 2005

Alice&Bob style

```
1. A -> S:  A.S.Na.exp(G,X).{A.S.Na.exp(G,X)}_inv(Ka)
2. S -> A:  A.S.Na.Nb.exp(G,Y).{A.S.Na.Nb.exp(G,Y)}_inv(Ks)
3. A -> S: {login.pwd}_K where K= exp(exp(G,Y),X) = exp(exp(G,X),Y)
```

Model Limitations

In reality, the messages 1 and 2 contain respectively the two following items lists.

- the initiator and target names,
- a fresh random number,
- a list of available confidentiality algorithms,
- a list of available integrity algorithms,
- a list of available key establishment algorithms,
- a context key (or half key) corresponding to the first key establishment algorithm given in the previous list,

- GSS context options/choices (such as unilateral or mutual authentication, use of sequencing and replay detection, and so on).

and

- the initiator and target names,
- the random number sent by the initiator,
- a fresh random number,
- the subset of offered confidentiality algorithms which are supported by the target,
- the subset of offered integrity algorithms which are supported by the target,
- an alternative key establishment algorithm (chosen from the offered list) if the first one offered is unsuitable,
- the half key corresponding to the initiator's key establishment algorithm (if necessary), or a context key (or half key) corresponding to the key establishment algorithm above,
- GSS context options/choices (such as unilateral or mutual authentication, use of sequencing and replay detection, and so on).

The sets of algorithms agreed are not used by LIPKEY, indeed LIPKEY only uses SPKM for key establishment. Thus they are not modelled. Furthermore, the key establishment modelled is à la Diffie-Hellman and GSS context options are not modelled.

Problems considered: 2

- `Target` authenticates `Initiator` on `k`
- `Initiator` authenticates `Target` on `ktrgtint`
- secrecy of `Login, Pwd`

Attacks Found

None

Further Notes

Here, we consider the case of mutual authentication, but there exists a unilateral (one-way) authentication version.

HLPSL Specification

```
role Initiator (
        A: agent,
        S: agent,
        G: nat,
        H: function,
        Ka: public_key,
        Ks: public_key,
        Login_A_S: message,
        Pwd_A_S:   message,

        SND, RCV: channel (dy))
played_by A def=

  local
        State        : nat,
        Na           : text(fresh),
        Nb           : text,
        Rnumber1     : text(fresh),
        X            : message,
        Keycompleted : message,
        W            : nat,
        K            : text.text

  owns Ka

  init State=0
  knowledge(A) = {Ks,Login_A_S,Pwd_A_S,G,S, inv(Ka)}
  transition

  1.  State = 0 /\  RCV(start) =|>
```

```
      State'= 1 /\ SND(A.S.Na'.exp(G,Rnumber1').
                         {A.S.Na'.exp(G,Rnumber1')}_inv(Ka))

  2.  State = 1 /\ RCV(A.S.Na.Nb'.X'.{A.S.Na.Nb'.X'}_inv(Ks)) =|>
      State'= 2 /\ Keycompleted'=exp(X',Rnumber1)
                /\ SND({Login_A_S.Pwd_A_S}_Keycompleted' )
                /\ secret(Login_A_S,S)
                /\ secret(Pwd_A_S,S)
                /\ K'= Login_A_S.Pwd_A_S
                /\ request(A,S,ktrgtint,Keycompleted')
                /\ witness(A,S,k,Keycompleted')

end role
```

```
role Target(
        A,S             : agent,
        G             : nat,
        H             : function,
        Ka,Ks            : public_key,
        Login, Pwd    : function,
        SND, RCV      : channel (dy))
played_by S def=

  local State         : nat,
        Na            : text,
        Nb            : text (fresh),
        Rnumber2      : text(fresh),
        Y             : message,
        Keycompleted  : message,
        W             : nat,
        K             : text.text

  owns Ks
  init State=0

  knowledge(S) = { Login, Pwd, A, S, Ka, Ks, H, inv(Ks) }
  transition

  1. State = 0 /\ RCV(A.S.Na'.Y'.{A.S.Na'.Y'}_inv(Ka)) =|>
```

```
     State'= 1 /\ SND(A.S.Na'.Nb'.exp(G,Rnumber2').
                     {A.S.Na'.Nb'.exp(G,Rnumber2')}_inv(Ks))
                /\ Keycompleted'=exp(Y',Rnumber2')
                /\ secret(Login(A,S),A)
                /\ secret(Pwd(A,S),A)
                /\ witness(S,A,ktrgtint,Keycompleted')

  21. State = 1 /\ RCV({Login(A,S).Pwd(A,S)}_Keycompleted) =|>
      State'= 2 /\ K'=Login(A,S).Pwd(A,S)
                /\ request(S,A,k,Keycompleted)

end role
```

```
role Session(
          A,S : agent,
          Login, Pwd: function,
          Ka: public_key,
          Ks: public_key,
          H: function,
          G: nat,
          Si,Ri,St,Rt: channel) def=

  composition
     Initiator(A,S,G,H,Ka,Ks,Login(A,S),Pwd(A,S),Si,Ri) /\
     Target(A,S,G,H,Ka,Ks,Login,Pwd,St,Rt)
end role
```

```
role Environment() def=

  const s_a, r_a, s_i, r_i, s_s, r_s, r_b,s_b : channel (dy),
        a,s,i,b: agent,
        ka, ki, kb, ks: public_key,
        login, pwd : function,
        h: function,
        g: nat,
        k,ktrgtint: protocol_id
  knowledge(i) = { ki,i, inv(ki),a,b,s,h,g,ks,login(i,s),pwd(i,s),ka}

  composition
            Session(a,s,login,pwd,ka,ks,h,g,s_a,r_a,s_s,r_s)
        /\  Session(b,s,login,pwd,kb,ks,h,g,s_b,r_b,s_s,r_s)
        /\  Session(i,s,login,pwd,ki,ks,h,g,s_i,r_i,s_s,r_s)

end role
```

```
goal
  Target authenticates Initiator on k
  Initiator authenticates Target on ktrgtint
  secrecy_of Login, Pwd
end goal
```

```
Environment()
```

3.2 unknown initiator

Protocol Purpose

Provide a method to supply a secure channel between a client and server, authenticating the client with a password, and a server with a public key certificate.

Definition Reference

RFC 2847, http://www.faqs.org/rfcs/rfc2847.html

Model Authors

- Boichut Yohan, LIFC-INRIA Besancon, May 2004
- Sebastian Mödersheim, ETH Zürich, January 2005

Alice&Bob style

```
1. A -> S: S.Na.exp(G,X).H(S.Na.exp(G,X))
2. S -> A: S.Na.Nb.exp(G,Y).{S.Na.Nb.exp(G,Y)}_inv(Ks)
3. A -> S: {login.pwd}_K where K= exp(exp(G,Y),X) = exp(exp(G,X),Y)
```

Model Limitations

In real life, the messages 1 and 2 contain respectively the two following items lists.

- target names,
- a fresh random number,
- a list of available confidentiality algorithms,
- a list of available integrity algorithms,
- a list of available key establishment algorithms,
- a context key (or key half) corresponding to the first key estb. alg. given in the previous list,
- GSS context options/choices (such as unilateral or mutual authentication, use of sequencing and replay detection, and so on).

and

- target names,
- the random number sent by the initiator,
- a fresh random number,
- the subset of offered confidentiality algorithms which are supported by the target,
- the subset of offered integrity algorithms which are supported by the target,
- an alternative key establishment algorithm (chosen from the offered list) if the first one offered is unsuitable,
- the key half corresponding to the initiator's key estb. alg. (if necessary), or a context key (or key half) corresponding to the key estb. alg. above,

- GSS context options/choices (such as unilateral or mutual authentication, use of sequencing and replay detection, and so on).

The sets of algorithms agreed are not used by LIPKEY, indeed LIPKEY only uses SPKM for a key establishment. Thus they are not modelled. Furthermore, the key establishment modelled is a la Diffie-Hellman and GSS context options are not modelled.

Problems considered: 1

- `Target` authenticates `Initiator` on `k`
- secrecy of `Login, Pwd`

Attacks Found

None

Further Notes

Here, this unilateral (one-way) authentication version of this protocol. I have a problem to specify authentication since the server does not know A. Then, how to express witness(S,A,...)

HLPSL Specification

```
role Initiator (
        A,S: agent,
        G: nat,
        H: function,
        Ka,Ks: public_key,
        Login_A_S: message,
        Pwd_A_S:   message,

        SND, RCV: channel (dy))
played_by A def=

  local
        State        : nat,
        Na           : text(fresh),
        Nb           : text,
        Rnumber1     : text(fresh),
        X            : message,
        Keycompleted : message,
        W            : nat,
        K            : text.text

  owns Ka

  init State=0
  knowledge(A) = {Ks,Login_A_S,Pwd_A_S,G,S, inv(Ka)}
  transition

  1.  State = 0 /\ RCV(start) =|>
      State'= 1 /\ SND(S.Na'.exp(G,Rnumber1').
                     H(S.Na'.exp(G,Rnumber1')))

  2.  State = 1 /\ RCV(S.Na.Nb'.X'.{S.Na.Nb'.X'}_inv(Ks)) =|>
      State'= 2 /\ Keycompleted'=exp(X',Rnumber1)
                /\ SND({Login_A_S.Pwd_A_S}_Keycompleted' )
                /\ secret(Login_A_S,S)
                /\ secret(Pwd_A_S,S)
                /\ K'= Login_A_S.Pwd_A_S
                /\ witness(A,S,k,Keycompleted')

end role
```

```
role Target(
        A,S           : agent,
        G             : nat,
        H             : function,
        Ka,Ks         : public_key,
        Login, Pwd    : function,
        SND, RCV      : channel (dy))
played_by S def=

  local State        : nat,
        Na           : text,
        Nb           : text (fresh),
        Rnumber2     : text(fresh),
        Y            : message,
        Keycompleted : message,
        W            : nat,
        K            : text.text

  owns Ks
  init State=0

  knowledge(S) = { Login, Pwd, S, Ka, Ks, H, inv(Ks) }
  transition

  1.  State = 0 /\ RCV(S.Na'.Y'.H(S.Na'.Y')) =|>
      State'= 1 /\ SND(S.Na'.Nb'.exp(G,Rnumber2').
                      {S.Na'.Nb'.exp(G,Rnumber2')}_inv(Ks))
                /\ Keycompleted'=exp(Y',Rnumber2')
                /\ secret(Login(A,S),A)
                /\ secret(Pwd(A,S),A)

  21. State = 1 /\ RCV({Login(A,S).Pwd(A,S)}_Keycompleted) =|>
      State'= 2 /\ K'=Login(A,S).Pwd(A,S)
                /\ request(S,A,k,Keycompleted)

end role
```

```
role Session(
          A,S : agent,
          Login, Pwd: function,
          Ka: public_key,
          Ks: public_key,
          H: function,
          G: nat,
          Si,Ri,St,Rt: channel) def=

  composition
     Initiator(A,S,G,H,Ka,Ks,Login(A,S),Pwd(A,S),Si,Ri) /\
     Target(A,S,G,H,Ka,Ks,Login,Pwd,St,Rt)
end role
```

```
role Environment() def=

  const s_a, r_a, s_i, r_i, s_s, r_s, r_b,s_b : channel (dy),
        a,s,i,b: agent,
        ka, ki, kb, ks: public_key,
        login, pwd : function,
        h: function,
        g: nat,
        k: protocol_id

  knowledge(i) = { ki,i, inv(ki),a,b,s,h,g,ks,login(i,s),pwd(i,s),ka}

  composition
            Session(a,s,login,pwd,ka,ks,h,g,s_a,r_a,s_s,r_s)
        /\  Session(b,s,login,pwd,kb,ks,h,g,s_b,r_b,s_s,r_s)
        /\  Session(i,s,login,pwd,ki,ks,h,g,s_i,r_i,s_s,r_s)

end role
```

```
goal
  Target authenticates Initiator on k
  secrecy_of Login, Pwd
end goal
```

```
Environment()
```

4 (MS-)CHAPv2

Challenge/Response Authentication Protocol, version 2

Protocol Purpose

Mutual authentication between a server and a client who share a password. CHAPv2 is the authentication protocol for the Point-to-Point Tunneling Protocol suite (PPTP).

Definition Reference

[Zor00]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Paul Hankes Drielsma, ETH Zürich

Alice&Bob style

We assume that the server `B` and client `A` share password `k(A,B)` in advance. The server and client generate nonces `Nb` and `Na`, respectively.

```
1. A -> B : A
2. B -> A : Nb
3. A -> B : Na,H(k(A,B),(Na,Nb,A))
4. B -> A : H(k(A,B),Na)
```

Model Limitations

Issues abstracted from:

- Message structure: As is standard, we abstract away from the concrete details of message structure such as bit lengths, etc. What is left after this abstraction contains several redundancies, however (at least in the Dolev-Yao model). We therefore eliminate these redundancies, retaining the core of the data dependencies of the protocol.

Problems considered: 3

- secrecy of `Kab`
- `CHAP_Init` authenticates `CHAP_Resp` on `nb`
- `CHAP_Resp` authenticates `CHAP_Init` on `na`

Attacks Found

None

Further Notes

A cryptanalysis of this protocol in its full complexity can be found in [SMW99].

HLPSL Specification

```
role CHAP_Init (A,B   : agent,
                Kab   : symmetric_key,
                H     : function,
                Snd,Rcv: channel(dy)) played_by A def=

  local State :nat,
        Na: text (fresh),
        Nb: text

  init   State = 0
  knowledge(A) = {A,B,Kab,Na}

  transition
   1. State=0 /\ Rcv(start) =|> Snd(A) /\ State'=1

   2. State=1 /\ Rcv(Nb') =|>
        Snd(Na'.H(Kab.Na'.Nb'.A)) /\
        State'=2 /\
        witness(A,B,na,Na') /\
        secret(Kab,A) /\ secret(Kab,B)

   3. State=2 /\ Rcv(H(Kab.Na)) =|>
        State'=3 /\
        request(A,B,nb,Nb)

end role
```

```
role CHAP_Resp (     B,A : agent,
                     Kab : symmetric_key,
                     H: function,
                     Snd,Rcv: channel(dy)) played_by B def=

  local State : nat,
         Nb: text(fresh),
         Na: text

  init  State = 0
  knowledge(B) = {A,B,Kab,Nb}

  transition
   1. State=0 /\ Rcv(A') =|>
        Snd(Nb') /\
        State'=1 /\
        witness(B,A,nb,Nb')

   2. State=1 /\ Rcv(Na'.H(Kab.Na'.Nb.A)) =|>
        Snd(H(Kab.Na')) /\
        State'=2 /\
        secret(Kab,A) /\ secret(Kab,B) /\
        request(B,A,na,Na')

end role
```

```
role Session(A,B: agent,
             Kab: symmetric_key,
             H: function,
             SA, SB, RA, RB: channel (dy)) def=
  composition
           CHAP_Init(A, B, Kab, H, SA, RA)
       /\  CHAP_Resp(B, A, Kab, H, SB, RB)
end role
```

```
role Environment() def=

  local Sa1, Sa2, Sa3,
        Sb1, Sb2, Sb3,
        Ra1, Ra2, Ra3,
        Rb1, Rb2, Rb3: channel (dy)
  const na, nb: protocol_id
  knowledge(i) = {a, b, h, kai, kbi }

  composition
        Session(a,b,kab,h,Sa1,Sb1,Ra1,Rb1) /\
        Session(a,i,kai,h,Sa2,Sb2,Ra2,Rb2) /\
        Session(b,i,kbi,h,Sa3,Sb3,Ra3,Rb3)

end role
```

```
goal
```

secrecy of the shared key

```
 secrecy_of Kab

 CHAP_Init authenticates CHAP_Resp on nb
 CHAP_Resp authenticates CHAP_Init on na

end goal
```

```
Environment()
```

5 APOP: Authenticated Post Office Protocol

Protocol Purpose

Secure mechanism for origin authentication and replay protection.

Definition Reference

- RFC 1939 : http://www.faqs.org/rfcs/rfc1939.html

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004

Alice&Bob style

```
S -> C : Hello.Timestamp
C -> S : C.MD5(Timestamp.K_CS)
S -> C : Success
```

Problems considered: 1

- `Server` authenticates `Client` on `timestamp`

Attacks Found

None

Further Notes

The following protocol models part of the POP3 Post Office Protocol. POP3 is used to allow a workstation (client) to retrieve mail that a server is holding for it. After the POP3 server has sent a greeting, the session enters the AUTHORISATION state in which the client has to identify itself to the server. After successful identification the session enters the TRANSACTION state and the client may request actions from the server, e.g. for delivering mail. When the client issues the QUIT command, the session enters the UPDATE state, i.e. the server releases acquired resources and says goodbye. The modelled part of the POP3 protocol covers the greeting and the AUTHORISATION phase. There are several ways for server identification one of which is the APOP method which provides origin authentication and replay protection: The APOP method assumes server and client to share a common secret `K_CS`. The POP3 server includes a fresh timestamp in its greeting message. The client answers with his identity and a digest calculated by applying the

MD5 algorithm to the timestamp followed by the shared secret. On successful verification of the digest, the server issues a positive response and the session enters the TRANSACTION state.

HLPSL Specification

```
role Client(
    C,S            : agent,
    K_CS           : symmetric_key,
    MD5            : function,
    Hello, Success : text,
    SND,RCV        : channel(dy))
played_by C def=

  local
    State     : nat,
    Timestamp : text

  const
    timestamp : protocol_id

  init
    State = 0

  transition

 1. State  = 0 /\ RCV(Hello.Timestamp') =|>
    State' = 1 /\ SND(C.MD5(Timestamp'.K_CS))
               /\ witness(C,S,timestamp,Timestamp')

 2. State  = 1 /\ RCV(Success) =|>
    State' = 2

end role
```

```
role Server (
    C,S            : agent,
    K_CS           : symmetric_key,
    MD5            : function,
    Hello, Success : text,
    SND,RCV        : channel(dy))
played_by S def=

  local
    State     : nat,
    Timestamp : text(fresh)

  const
    timestamp : protocol_id

  init
    State = 10

  transition

 1. State  = 10 /\ RCV(start) =|>
    State' = 11 /\ SND(Hello.Timestamp')

 2. State  = 11 /\ RCV(C.MD5(Timestamp.K_CS)) =|>
    State' = 12 /\ SND(Success)
                /\ request(S,C,timestamp,Timestamp)
end role
```

```
role Session (
    C,S            : agent,
    K_CS           : symmetric_key,
    MD5            : function,
    Hello, Success : text)
def=

  local
    S1, S2: channel (dy),
    R1, R2: channel (dy)

  composition
```

```
    Client(C,S,K_CS,MD5,Hello,Success,S1,R1)
 /\ Server(C,S,K_CS,MD5,Hello,Success,S2,R2)

end role
```

```
role Environment() def=

  const
    c,s             : agent,
    md5             : function,
    k_cs,k_is       : symmetric_key,
    hello,success : text

  knowledge(i) = {c,s,i,k_is,md5,hello,success}

  composition

    Session(c,s,k_cs,md5,hello,success)
 /\ Session(c,s,k_cs,md5,hello,success)
 /\ Session(i,s,k_is,md5,hello,success)
 /\ Session(i,s,k_is,md5,hello,success)

end role
```

```
goal
  Server authenticates Client on timestamp
end goal
```

```
Environment()
```

6 CRAM-MD5 Challenge-Response Authentication Mechanism

Protocol Purpose

CRAM-MD5 is intended to provide an authentication extension to IMAP4 that neither transfers passwords in cleartext nor requires significant security infrastructure in order to function. To this end, the protocol assumes a shared password (which we model, without loss of generality, as a shared cryptographic key) between the IMAP4 server (called S in our model) and each client A. Only a hash value of the shared password is ever sent over the network, thus precluding plaintext transmission.

Definition Reference

RFC 2195 [KCK97]

Model Authors

Paul Hankes Drielsma, ETH Zürich, July 2004

Alice&Bob style

```
Alice-Bob Notation:
 1. A -> S: A
 2. S -> A: Ns.T.S
 3. A -> S: F(SK.T)
 where
     Ns is a nonce generated by the server;
     T is a timestamp (currently abstracted with a nonce)
     SK is the shared key between A and S
     F is a cryptographic hash function (MD5 in practice, but this is
      unimportant for our purposes).  The use of F
      is intended to ensure that only a digest of the shared
      key is transmitted, with T assuring freshness of the
      generated hash value.
```

Model Limitations

Issues abstracted from:

- We abstract away from the timestamp `T` using a standard nonce.

Problems considered: 2

- secrecy of `SK`
- `Server` authenticates `Client` on `auth`

Attacks Found

None

Further Notes

RFC 2195 [KCK97] states that the first message from the server S begins with a ”presumptively arbitrary string of random digits”; that is, a nonce. Unspecified, however, is what the client should do with this nonce. It does not appear in subsequent protocol message. We therefore presume it is intended to ensure replay protection, but our HLPSL specification at present does not explicitly model that the client should maintain a list of nonces previously received from the server.

HLPSL Specification

```
role Client(A, S: agent,
            SK: message,
            F: function,
            SND, RCV: channel (dy)) played_by A def=
  local State: nat,
         T, Ns : text
  init State = 0
  transition
   1. State = 0 /\ RCV(start)
   =|>
      State' = 1 /\ SND(A)

  2. State = 1 /\ RCV(Ns'.T'.S)
  =|>
     State' = 2 /\ SND(F(SK.T'))
     /\ witness(A,S,auth,F(SK.T'))
     /\ secret(SK,S)
```

```
end role

role Server(S : agent,
            K,F: function,
            SND, RCV: channel (dy)) played_by S def=
  local State : nat, A: agent, T, Ns : text (fresh), Auth: message
  init State = 0
  transition
   1. State = 0 /\ RCV(A')
   =|>
      State' = 1 /\ SND(Ns'.T'.S)

   2. State = 1 /\ RCV(F(K(A.S).T))
   =|>
      State' = 2
    /\ Auth' = F(K(A.S).T)
    /\ request(S,A,auth,F(K(A.S).T))

end role

role Session(A, S: agent,
             K, F: function,
             SNDA, SNDS, RCVA, RCVS: channel (dy)) def=
  local SK: message
  init SK = K(A.S)
  composition
    Client(A,S,SK,F,SNDA,RCVA) /\
    Server(S,K,F,SNDS,RCVS)
end role

role Environment() def=
 local S1, S2, S3, S4,
       R1, R2, R3, R4,
       S5, S6, R5, R6: channel(dy)
 const a, s : agent,
       k, f : function,
       auth : protocol_id
 knowledge(i) = {a,s,i,f}
 composition
  Session(a,s,k,f,S1,S2,R1,R2) /\
  Session(i,s,k,f,S3,S4,R3,R4) /\
  Session(a,s,k,f,S5,S6,R5,R6)
end role
```

```
goal
  secrecy_of SK
  Server authenticates Client on auth
end goal
```

```
Environment()
```

7 DHCP-Delayed-Auth

Protocol Purpose

Delayed entity and message authentication for DHCP

Definition Reference

RFC 3118, http://www.faqs.org/rfcs/rfc3118.html

Model Authors

- Graham Steel, University of Edinburgh, July 2004
- Luca Compagna, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
1. C -> S : C, delayedAuthReq, Time1
2. S -> C : S, delayedAuthReq, succ(Time1), KeyID(K),
            H(S, delayedAuthReq, succ(Time1), K)
```

Model Limitations

The RFC describes different options and checks in terms of MAY and MUST etc.[1] This model is of the minimum protocol, i.e. only the MUST checks. In real life, message looks like

- 90 (auth requested),
- length,
- 1 (for delayed auth),
- 1 (to indicate standard HMAC algorithm),
- 0 (standard Replay Detection Mechanism, monotonically increasing counter),
- counter value.

[1]**Fix:** LV: why MAY and MUST in capitals? Moreover, what is really meant here?

We ignore length field (as it cannot be, yet, expressed in HLPSL), use fresh nonce to model RDM, and assume 'DelayedAuthReq' token is enough to specify algorithm, type of auth, and type of RDM.

The server returns the nonce + 1 (or `succ(nonce)` to be exact) instead of a timestamp with a higher value.

Problems considered: 2

- secrecy of `K`
- `DHCP_Delayed_Client` authenticates `DHCP_Delayed_Server` on `sig`

Attacks Found

None

Further Notes

Client is the initiator. Sends a DHCP discover and requests authentication

HLPSL Specification

```
role DHCP_Delayed_Client (
        C, S     : agent,    % C client, S server
        H        : function, % HMAC hash func.
        KeyID    : function, % get a key id from a key
        K        : text,     % K is the pre-existing shared secret
        Snd, Rcv : channel(dy)) played_by C def=

  local State : nat,
        Time1 : text(fresh),
        Sig   : message

  const  delayedAuthReq : protocol_id,
         succ           : function   % Successor function

  init  State = 0
  knowledge(C) = {C,S,K,delayedAuthReq,KeyID,H,succ}

  transition
```

```
   1. State = 0 /\ Rcv(start) =|>
      State'= 1 /\ Snd(C.delayedAuthReq.Time1')

   2. State = 1 /\ Rcv(S.delayedAuthReq.succ(Time1).KeyID(K).
                      H(S,delayedAuthReq,succ(Time1),K)) =|>
      State'= 2 /\ Sig' = H(S,delayedAuthReq,succ(Time1),K)
                /\ request(C,S,sig,Sig')
                /\ secret(K,S)

end role
```

```
role DHCP_Delayed_Server (
        S,C      : agent,
        H        : function,  % HMAC hash func.
        KeyID    : function,  % get a key id from a key
        K        : text,
        Snd, Rcv : channel (dy)) played_by S def=

  local State : nat,
        Time1 : text,
        Sig   : message

 const  delayedAuthReq : protocol_id,
        succ           : function   % Successor function

  init State = 0
  knowledge(S) = {S,C,K,KeyID,H,delayedAuthReq,succ}

  transition

   1. State = 0 /\ Rcv(C.delayedAuthReq.Time1') =|>
      State'= 1 /\ Sig' = H(S,delayedAuthReq,succ(Time1'),K)
                /\ Snd(S.delayedAuthReq.succ(Time1').KeyID(K).Sig')
                /\ witness(S,C,sig,Sig')

end role
```

```
role Session(C, S           : agent,
             H, KeyID       : function,
             K              : text,
             SA, RA, SB, RB : channel (dy)) def=
  composition
        DHCP_Delayed_Server(S,C,H,KeyID,K,SA,RA) /\
        DHCP_Delayed_Client(C,S,H,KeyID,K,SB,RB)

end role
```

```
role Environment() def=

 local Sa1,Sa2,Sa3,Ra1,Ra2,Ra3,
       Sb1,Rb1,Si2,Ri2,Si3,Ri3 : channel (dy)

 const a, b     : agent,
       k1, k2   : text,
       h, keyid : function,
       sig      : protocol_id

 knowledge(i) = {a,b,k2,i,delayedAuthReq}

 composition
        Session(a,b,h,keyid,k1,Sa1,Ra1,Sb1,Rb1)
    /\  Session(a,i,h,keyid,k2,Sa2,Ra2,Si2,Ri2)
    /\  Session(i,b,h,keyid,k3,Sa3,Ra3,Si3,Ri3)

end role
```

```
goal
  secrecy_of K
  DHCP_Delayed_Client authenticates DHCP_Delayed_Server on sig
end goal
```

```
Environment()
```

8 EAP: Extensible Authentication Protocol

8.1 With AKA method (Authentication and Key Agreement)

Protocol Purpose

Mutual Authentication, replay protection, confidentiality, Key Derivation.
EAP-AKA is developed from the UMTS AKA authentication and key agreement protocol.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-arkko-pppext-eap-aka-15.txt

Model Authors

- Jing Zhang for Siemens CT IC 3, 2004
- Peter Warkentin, Siemens CT IC 3
- Vishal Sankhla, University of Southern California

Alice&Bob style

```
Before the protocol starts, the authenticator has obtained an authentication
vector AV from the home environment (HE) of the peer. AV is given as
AV = (AT_RAND,AT_AUTN,AT_RES,IK,CK). For constructing AV, HE and P share
a secret symmetric key SK and a sequence number SQN (which changes for each
session), and a list of functions F1,F2 for authentication purposes and
F3,F4,F5 for key generation.

Shared data/functions of A and P:
SK          : symmetric key
SQN         : sequence number
F1, F2      : authentication functions
F3, F4, F5 : key generation functions

A -> P: Identity
P -> A: Identity.NAI
        % NAI is Network Address Identifier.
        % A uses
        %   nonce          AT_RAND
        %   mac           AT_MAC  = F1(SK,SQN,AT_RAND)
        %   anonymity key AK      = F5(SK,AT_RAND)
        %   auth. token   AT_AUTN = {SQN}_AK . AT_MAC
 A -> P: AT_RAND.AT_AUTN
        % P checks AT_AUTN and generates
        %   auth. result  AT_RES  = F2(SK,AT_RAND)
 P -> A: AT_RES
        % A checks AT_RES
 A -> P: Success
        % A,P agree on
        %   session key for encryption      CK = F3(SK,AT_RAND)
        %   session key for integrity check IK = F4(SK,AT_RAND)
```

Model Limitations

- The authenticator A should only use the components of the authentication vector and thus should have no knowledge about F1,F2,F3,F4,F5,SK,SQN. But to perform checks, these data must be included in the parameters (and thus in the knowledge) of A, e.g. to compare the answer RES of the peer to the given challenge RAND with XRES from AV).
- The modeller has to take care that each session gets a unique sequence number SQN (see notes below).
- EAP-specific MAC values in 3rd and 4th message ignored.

Problems considered: 2

- `Peer` authenticates `Authenticator` on `at_rand`
- `Authenticator` authenticates `Peer` on `at_rand2`

Attacks Found

None

Further Notes

The mechanism is based on challenge-response and uses symmetric cryptography. AKA typically runs in a UMTS Subscriber Identity Module (USIM). In AKA, the pre-shared credential is stored in the USIM and in the user's home server. The authentication process starts when the user

attaches to the home environment where an authentication vector from the secret key and a sequence number is generated. The authentication vector contains a random number (RAND), an authenticator part (AUTN) for authenticating the network, the expected result (XRES) for authenticating the peer, a session key for integrity (IK), and a session key for encryption (CK). After the authentication vector is delivered, the authentication starts the protocol by sending a challenge (RAND) and authentication data (AUTN) to USIM. USIM verifies the AUTN based on the secret key and the sequence number to authenticate the network. If the AUTN is valid, the USIM generates the authentication result RES itself and sends this to the authentication server. The authentication server verifies the RES from the USIM. If it is valid, the user is authenticated and IK and CK will be used in key derivation of both peers.
Attention: the modelling of the sequence number SQN is problematic. The purpose of SQN is to avoid replay attacks. To achieve this, SQN must be unique for each session (like a unique session id). By using OFMC with the sessco option this uniqueness is violated — and thus there will result an invalid replay attack involving two sessions with identical SQN.

HLPSL Specification

```
role Peer (
    P,A              : agent,
    F1,F2,F3,F4,F5   : function,
    SK               : symmetric_key,
    Success,Identity : text,
    SQN              : text,
    SND,RCV          : channel (dy))
played_by P def=

  local
    AT_RAND  : text,
    NAI      : text (fresh),
    State    : nat

  init
    State = 0

  knowledge(P) = {F1,F2,F3,F4,F5,Success,SK,Identity,SQN}

  transition

  1. State  = 0 /\ RCV(Identity) =|>
     State' = 2 /\ SND(Identity.NAI')
  2. State  = 2 /\ RCV(AT_RAND'.
                       {SQN}_F5(SK.AT_RAND').
                        F1(SK.SQN.AT_RAND')) =|>
     State' = 4 /\ SND(F2(SK.AT_RAND'))
                /\ request(P,A,at_rand,AT_RAND')
                /\ witness(P,A,at_rand2,AT_RAND')

  3. State  = 4 /\ RCV(Success) =|>
     State' = 6

end role
```

```
role Authenticator (
    A,P               : agent,
    F1,F2,F3,F4,F5    : function,
    SK                : symmetric_key,
    Success,Identity  : text,
    SQN               : text,
    SND,RCV           : channel (dy))
played_by A def=

  local
    AT_RAND         : text(fresh),
    NAI             : text,
    State           : nat

  init
    State = 1

  knowledge(A)= {F1,F2,F3,F4,F5,SK,SuccessIdentity,SQN}
  transition

  1. State  = 1 /\ RCV(start) =|>
     State' = 3 /\ SND(Identity)

  2. State  = 3 /\ RCV(Identity.NAI') =|>
     State' = 5 /\ SND(AT_RAND'.
                       {SQN}_F5(SK.AT_RAND').
```

```
                      F1(SK.SQN.AT_RAND’))
                /\ witness(A,P,at_rand,AT_RAND’)

  3. State  = 5 /\ RCV(F2(SK.AT_RAND)) =|>
     State’ = 7 /\ SND(Success)
                /\ request(A,P,at_rand2,AT_RAND)

end role
```

```
role Session(
    P,A                     : agent,
    F1,F2,F3,F4,F5          : function,
    SK                      : symmetric_key,
    Success,Identity        : text,
    SQN                     : text,
    S1,R1,S2,R2             : channel (dy))
def=
const
    at_rand,at_rand2        : protocol_id

  composition
     Peer(         P,A,F1,F2,F3,F4,F5,SK,Success,Identity,SQN,S1,R1)
  /\ Authenticator(A,P,F1,F2,F3,F4,F5,SK,Success,Identity,SQN,S2,R2)

end role

role Environment() def=

 local
    Sa,Sp,Ra,Rp,Si,Ri,Si2,Ri2,Sa2,Ra2,Sp2,Rp2      : channel

  const
    a,p                       : agent,
    kap,kip                   : symmetric_key, % !!one per user   !!
    sqna1, sqna2, sqni        : text,          % !!one per session!!
    f1,f2,f3,f4,f5            : function,
    success,identity          : text

  knowledge(i) = {p,a,i,f1,f2,f3,f4,f5,kip}
  composition
     Session(a,p,f1,f2,f3,f4,f5,kap,success,identity,sqna1,
             Sa,Ra,Sp,Rp)
  /\ Session(a,p,f1,f2,f3,f4,f5,kap,success,identity,sqna2,
             Sa,Ra,Sp,Rp)
  /\ Session(i,p,f1,f2,f3,f4,f5,kip,success,identity,sqni,
             Si,Ri,Sp2,Rp2)

end role
```

```
goal
  Peer authenticates Authenticator on at_rand
  Authenticator authenticates Peer on at_rand2
end goal
```

```
Environment()
```

8.2 With Archie method

Protocol Purpose

Mutual authentication, Key Derivation

EAP-Archie is a native EAP authentication method [20]. Therefore, there is no defined stand-alone version of Archie outside EAP. Archie is one of the symmetric cryptography methods that use a pre-shared secret key. The Archie 512-bit shared secret key consists of two 128-bit keys called key-confirmation key (KCK), key-encryption key (KEK), and the 256-bit key-derivation key (KDK). The key-confirmation key is used for mutual authentication. The key-encryption key is used for distributing the secret nonces for session key derivation. The key-derivation key is used for deriving the session keys.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-jwalker-eap-archie-02.txt

(expired)

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004
- Vishal Sankhla, University of Southern California, 2004

Alice&Bob style

```
S -> P: S.SessionID
P -> S: SessionID.P.{nonceP}_KEK.Bind.MAC1
S -> P: SessionID.{nonceA}_KEK.Bind.MAC2
P -> S: SessionID.MAC3
```

Problems considered: 5

- `Peer` authenticates `Server` on `sd`
- `Peer` authenticates `Server` on `na`
- `Server` authenticates `Peer` on `bind`
- `Server` authenticates `Peer` on `np`
- secrecy of `Na, Np`

Attacks Found

None

Further Notes

- P wants to be sure that S sent SessionID and nonceA.
- S wants to be sure that P sent nonceP and Bind.
- Secrecy of nonceA and nonceP, which are used for key derivation.

```
- SessionID: Nonce
- KCK: Shared Key used for Authentication
- KEK: Shared Key used for Encryption
- KDK: Shared Key used for Key Derivation
```

```
 - EMK: EAP Master Key: PRF(KDK.nonceA.nonceP)
 - MAC1: MAC(KCK.S.SessionID.P.{nonceP}_KEK.Bind)
 - MAC2: MAC(KCK.P.{nonceP}_KEK.SessionID.{nonceA}_KEK.Bind)
 - MAC3: MAC(KCK.SessionID)
```

HLPSL Specification

```
role Peer (
    P,S          : agent,
    MAC          : function,
    KEK,KCK,KDK  : symmetric_key,
    SND,RCV      : channel (dy))
played_by P def=

  local
    Np,Bind      : text (fresh),
    Na,Sd        : text,           % Sd (=SessionID)
    State        : nat,
    EMK          : message

  init
    State = 0

  knowledge(P)= {S,P,KEK,MAC,KCK,KDK}
  transition

 1. State =  0 /\ RCV(S.Sd') =|>
    State' = 2 /\ SND(Sd'.P.
                      {Np'}_KEK.Bind'.
                      MAC(KCK.S.Sd'.P.{Np'}_KEK.Bind'))
               /\ secret(Np',P)
               /\ secret(Np',S)
               /\ witness(P,S,np,Np')
               /\ witness(P,S,bind,Bind')

 2. State  = 2 /\ RCV(Sd.{Na'}_KEK.Bind.
                      MAC(KCK.P.{Np}_KEK.Sd.{Na'}_KEK.Bind)) =|>
    State' = 4 /\ SND(Sd.MAC(KCK.Sd))
               /\ request(P,S,sd,Sd)
               /\ request(P,S,na,Na')
```

```
              /\ secret(Na',P)
              /\ secret(Na',S)

end role
```

```
role Server  (
    S,P             : agent,
    MAC             : function,
    KEK,KCK,KDK     : symmetric_key,
    SND,RCV         : channel (dy))
played_by S def=

  local
    Np,Bind         : text,
    Na,Sd           : text (fresh),
    State           : nat,
    EMK             : message

  init
    State = 1

  knowledge(S) ={S,P,KEK,KCK,KDK}
  transition

 1. State  = 1 /\ RCV(start) =|>
    State' = 3 /\ SND(S.Sd')
               /\ witness(S,P,sd,Sd')

 2. State  = 3 /\ RCV(Sd.P.{Np'}_KEK.Bind'.
                        MAC(KCK.S.Sd.P.{Np'}_KEK.Bind')) =|>
    State' = 5 /\ SND(Sd.{Na'}_KEK.Bind'.
                        MAC(KCK.P.{Np'}_KEK.Sd.{Na'}_KEK.Bind'))
               /\ witness(S,P,na,Na')
               /\ request(S,P,np,Np')
               /\ request(S,P,bind,Bind')
               /\ secret(Na',P)
               /\ secret(Np',P)
               /\ secret(Na',S)
               /\ secret(Np',S)
 3. State  = 5 /\ RCV(Sd.MAC(KCK.Sd)) =|>
    State' = 7

end role
```

```
role Session (
    S,P           : agent,
    MAC,PRF       : function,
    KEK,KCK,KDK   : symmetric_key,
    Speer,Rpeer,Sserver,Rserver  : channel (dy))
def=

  composition
    Peer(P,S,MAC,KEK,KCK,KDK,Speer,Rpeer)
 /\ Server(S,P,MAC,KEK,KCK,KDK,Sserver,Rserver)

end role
```

```
role Environment() def=

  local Sp0,Rp0,Ss0,Rs0,Sp1,Rp1,Ss1,Rs1 : channel
  const
    s,p                     : agent,
    mac,prf                 : function,
    kek,kck,kdk             : symmetric_key,
    kek_is,kck_is,kdk_is    : symmetric_key,
    kek_ip,kck_ip,kdk_ip    : symmetric_key,
    sd,na,np,bind  : protocol_id

  knowledge(i) = {s,p,mac,prf}

  composition

     Session(s,p,mac,prf,kek,kck,kdk,Sp0,Rp0,Ss0,Rs0)
  /\ Session(s,p,mac,prf,kek,kck,kdk,Sp1,Rp1,Ss1,Rs1)
```

```
end role
```

```
% - P wants to be sure that S sent SessionID and nonceA.
% - S wants to be sure that P sent nonceP and Bind.
% - Secrecy of nonceA and nonceP, which are used for key derivation.

goal

  Peer authenticates Server on sd
  Peer authenticates Server on na
  Server authenticates Peer on bind
  Server authenticates Peer on np
  secrecy_of Na, Np

end goal
```

```
Environment()
```

8.3 With IKEv2 method

Protocol Purpose

Mutual authentication, key establishment, replay protection, confidentiality

EAP-IKEv2 is an EAP method which reuses the cryptography and the payloads of IKEv2, creating a flexible EAP method that supports both symmetric and asymmetric authentication, as well as a combination of both. This EAP method offers the security benefits of IKEv2 authentication and key agreement without the goal of establishing IPsec security associations.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-tschofenig-eap-ikev2-05.txt

Model Authors

- Wolfgang Bücker, Siemens CT IC 3, 2004

- Jing Zhang for Siemens CT IC 3
- Vishal Sankhla, University of Southern California

Alice&Bob style

```
P:   Peer
NAS: Authenticator (Network Access Server)
AS:  (Back-end) Authentication Server

P -- NAS -- AS

P <- NAS         EAP-Req/Identity
P -> NAS         EAP-Res/Identity(Id)
     NAS -> AS   Identity(Id)
     NAS <- AS   HDR(A,0), SAi1, KEi, Ni)
P <- NAS         EAP-Req/EAP-IKEv2(HDR(A,0), SAi1, KEi, Ni)
P -> NAS         EAP-Res/EAP-IKEv2(HDR(A,B), SAr1, KEr, Nr)
     NAS -> AS   HDR(A,B), SAr1, KEr, Nr
     NAS <- AS   HDR(A,B), {AS, AUTH}SK_e_i
P <- NAS         EAP-Req/EAP-IKEv2(HDR(A,B), {AS, AUTH}SK_e_i)
P -> NAS         EAP-Res/EAP-IKEv2(HDR(A,B), {P, AUTH}SK_e_r)
     NAS -> AS   HDR(A,B), {P, AUTH}SK_e_r
     NAS <- AS   Success
P <- NAS         EAP-Success
```

Problems considered: 1

- secrecy of `SK`

Attacks Found

None

HLPSL Specification

```
role Peer(P,NAS: agent,
          G: text,
          PRF: function,
          Kp, Kas: public_key,
          SND_NAS, RCV_NAS: channel (dy))
played_by P def=

  local
    Ni, SAi1 : text,
    Nr,DHp   : text (fresh),
    KEi      : text,
    SK       : message,
    AS       : agent,
    State    : nat

  const
    eap_req_id, eap_success, eap_failure : text

  init
    State = 0

  knowledge(P)={inv(Kp)}

  transition

    1. State = 0 /\ RCV_NAS( eap_req_id ) =|>
         SND_NAS( P ) /\ State' = 2

    2. State = 2 /\ RCV_NAS(SAi1'.KEi'.Ni') =|>
         SND_NAS(SAi1'.exp(G,DHp').Nr') /\
         SK'=PRF(Ni'.Nr'.exp(KEi',DHp')) /\
         State' = 4

    % As opposed to IKEv2, in EAP-IKEv2 there is no negotiation of a
    % CHILD_SA => no second SA payload and no traffic selector payload
    3. State = 4 /\ RCV_NAS({AS'.{SAi1.KEi.Ni.Nr}_inv(Kas)}_SK) =|>
         SND_NAS({P.{SAi1.exp(G,DHp').Nr.Ni}_inv(Kp)}_SK) /\
         State' = 6 /\
         secret(SK,AS')

end role
```

```
% The NAS is agnostic of the EAP method. It only relays messages between
% Peer and AS. The only EAP related actions are issuing the EAP-Identity
% Request and recognizing success and failure messages from the AS
role NetworkAccessServer(
       NAS, P, AS: agent,
       SND_P, RCV_P, SND_AS, RCV_AS: channel (dy))
played_by NAS def=

  local
    ID_P: text,
    EAP_Req_Mess, EAP_Res_Mess: text,
    State: nat

  const
    eap_req_id, eap_success, eap_failure : text

  init
    State = 20

  transition

    1. State = 20 /\ RCV_P( start )  =|>
         SND_P( eap_req_id ) /\ State' = 21

    2. State = 21 /\ RCV_P( ID_P' )  =|>
         SND_AS( ID_P' ) /\ State' = 22

    %  Pass messages from AS and Peer
    %  If it is not a success or failure message, remain in state
    3. State = 22 /\ RCV_AS( EAP_Req_Mess' ) /\
       EAP_Req_Mess' /= eap_success /\ EAP_Req_Mess' /= eap_failure  =|>
         SND_P( EAP_Req_Mess')

    %  Pass message from Peer to AS
    4. State = 22 /\ RCV_P( EAP_Res_Mess' )  =|>
         SND_AS( EAP_Res_Mess' )

    %  On receiving a success message from AS change state to final state
    5. State = 22 /\ RCV_AS( eap_success ) =|>
         SND_P( eap_success ) /\ State' = 23
    %  On receiving a failure message from AS change state to final state
    6. State = 22 /\ RCV_AS( eap_failure ) =|>
         SND_P( eap_failure ) /\ State' = 23

end role
```

```
role AuthenticationServer(
        AS, NAS: agent,
        G: text,
        PRF: function,
        Kas, Kp: public_key,
        SND_NAS, RCV_NAS: channel (dy))
played_by AS def=

  local
    ID_P, Nr: text,
    SAi1, DHi, Ni: text(fresh),
    KEr: text,
    SK: message,
    State: nat

  init
    State = 40

  transition

    1. State = 40 /\ RCV_NAS( ID_P') =|>
         SND_NAS( SAi1'.exp(G,DHi').Ni') /\ State' = 42

    % As opposed to IKEv2, in EAP-IKEv2 there is no negotiation of a
    % CHILD_SA => no second SA payload and no traffic selector payload
    2. State = 42 /\ RCV_NAS(SAi1.KEr'.Nr') =|>
         SK'= PRF(Ni.Nr'.exp(KEr', DHi)) /\
         SND_NAS({AS.{SAi1.exp(G,DHi).Ni.Nr}_inv(Kas)}_SK ) /\
         State' = 44

    3. State = 44 /\ RCV_NAS({ID_P.{SAi1.KEr.Nr.Ni}_inv(Kp)}_SK) =|>
         State' = 46
```

```
end role
```

```
role Session(
       P, NAS, AS: agent,
       G: text,
       PRF: function,
       Kp, Kas: public_key,
       S_PN, R_PN, S_NP, R_NP, S_NA, R_NA, S_AN, R_AN: channel (dy))
def=

  composition
       Peer(P, NAS, G, PRF, Kp, Kas, S_PN, R_PN)
    /\ NetworkAccessServer(NAS, P, AS, S_NP, R_NP, S_NA, R_NA)
    /\ AuthenticationServer(AS, NAS, G, PRF, Kas, Kp, S_AN, R_AN)

end role
```

```
role Environment() def=

  local
    S1,R1,S2,R2,S3,R3,S4,R4: channel
  const
    p, nas, as               : agent,
    g                        : text,
    f                        : function,
    kp, kas                  : public_key

  composition
    Session(p,nas,as,g,f,kp,kas,S1,R1,S2,R2,S3,R3,S4,R4)

end role
```

```
goal
  secrecy_of SK
%  secrecy_of ID_P
```

```
end goal
```

```
Environment()
```

8.4 With SIM

Protocol Purpose

Mutual authentication, key establishment, integrity protection, replay protection, confidentiality.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-haverinen-pppext-eap-sim-16.txt

Model Authors

- Jing Zhang for Siemens CT IC 3, 2004
- Peter Warkentin, Siemens CT IC 3
- Vishal Sankhla, University of Southern California, 2004

Alice&Bob style

```
 A -> P: request_id
 P -> A: respond_id.UserID
         % actually, UserID = IMSI/TMSI
         % here, we use UserID = P
 A -> P: request_sim_start.Version
         % actually, A sends a list of supported versions
         % and P selects one of them
         % here, we assume only one version
 P -> A: respond_sim_start.Version.NonceP
 A -> P: request_sim_challenge.Rand.Mac1
         % A selects Rand from an authentication vector (Rand.SRES.Kc)
         % which he has obtained from some AuC.
         % Mac1 is some message authentication code computed via
```

```
        %   Kc   = A8(Ki,Rand)
        %   SRES = A3(Ki,Rand)
        %   MK   = SHA1(P,Kc,NonceP,Ver)
        %   Mac1 = MAC1(MK,Np)
        % Here, Ki is some secret key only known to A,P.
        % MK is a master key which will be used for generating keys for
        % encryption, authentication and data-integrity.
        % On receipt, P checks Mac1 and thus authenticates A.
 P -> A: respond_sim_challenge.Mac2
        %   Mac2 = MAC2(MK,SRES)
        % On receipt, A checks Mac2 and thus authenticates P.
 A -> P: request_success
```

Model Limitations

- One RAND-challenge: the original version uses 2-3 authentication vectors for generating keying material.
- Simplified id handling (instead of IMSI/TMSI).
- Simplified version handling: only one version for algorithm selection is negotiated.

Problems considered: 3

- secrecy of `MK`
- `Peer` authenticates `Authenticator` on `mac1`
- `Authenticator` authenticates `Peer` on `mac2`

Attacks Found

None

Further Notes

EAP-SIM (Subscriber Identity Module) provides an authentication and encryption mechanism based on the existing method of Global System for Mobile communications (GSM). GSM authentication algorithms run on SIM, a smart card device inserted into the GSM user device. This card stores the shared secret between the user and the Authentication Center (AuC) in the mobile operator network the user is subscribed to. From the AuC, EAP-SIM gathers the ”triplet” (RAND, SRES, Kc) and generates the secure session key.

HLPSL Specification

```
role Peer (P, A                   : agent,
           Ki                     : symmetric_key,
           SHA1,A3,A8,MAC1,MAC2 : function,
           SND, RCV               : channel (dy))
played_by P def=

  local  State       : nat,
         Np          : text(fresh),  % nonce
         Kc, MK      : message,      % keys
         Mac1, Mac2 : message,      % mac's
         Ver         : text,         % version
         SRES        : message,      % signed response
         Rand        : text          % random number

  const  mac1, mac2                                    : protocol_id,
         request_id,             respond_id            : text,
         request_sim_start,      respond_sim_start     : text,
         request_sim_challenge, respond_sim_challenge : text,
         request_success                               : text

  init State = 0

  knowledge(P)={Ki,SHA1,A3,A8,MAC1,MAC2}

  transition

  1. State  = 0 /\ RCV(request_id) =|>
     State' = 2 /\ SND(respond_id.P)

  2. State  = 2 /\ RCV(request_sim_start.Ver') =|>
```

```
     State' = 4 /\ SND(respond_sim_start.Ver'.Np')

  4. State  = 4 /\ RCV(request_sim_challenge.Rand'.Mac1')
                /\ Kc'   = A8(Ki,Rand')
                /\ MK'   = SHA1(P,Kc',Np',Ver)
                /\ Mac1' = MAC1(MK',Np) =|>
     State' = 6 /\ SRES' = A3(Ki,Rand')
                /\ Mac2' = MAC2(MK',SRES')
                /\ SND(respond_sim_challenge.Mac2')
                /\ request(P,A,mac1,Mac1')
                /\ witness(P,A,mac2,Mac2')
                /\ secret(MK', A) /\ secret(MK', P)

  6. State  = 6 /\ RCV(request_success) =|>
     State' = 8

end role
```

```
role Authenticator (P, A                 : agent,
                    Ki                   : symmetric_key,
                    SHA1,A3,A8,MAC1,MAC2 : function,
                    SND, RCV             : channel (dy))
played_by A def=

  local  State        : nat,
         Np           : text,         % nonce
         Kc, MK       : message,      % keys
         Mac1, Mac2   : message,      % mac's
         Ver          : text,         % version
         SRES         : message,      % signed response
         Rand         : text (fresh)  % random number

  const  mac1, mac2                                    : protocol_id,
         request_id,            respond_id             : text,
         request_sim_start,     respond_sim_start      : text,
         request_sim_challenge, respond_sim_challenge  : text,
         request_success                               : text

  init State = 1
  knowledge(A)={Ki,SHA1,A3,A8,MAC1,MAC2}

  transition

  1. State  = 1 /\ RCV(start) =|>
     State' = 3 /\ SND(request_id)

  3. State  = 3 /\ RCV(respond_id.P) =|>
     State' = 5 /\ SND(request_sim_start.Ver')

  5. State  = 5 /\ RCV(respond_sim_start.Ver.Np') =|>
     State' = 7 /\ Kc'   = A8(Ki,Rand')
                /\ MK'   = SHA1(P,Kc',Np',Ver)
                /\ Mac1' = MAC1(MK',Np')
                /\ SND(request_sim_challenge.Rand'.Mac1')
                /\ witness(A,P,mac1,Mac1')

  7. State  = 7 /\ RCV(respond_sim_challenge.Mac2')
                /\ Mac2' = MAC2(MK,SRES')
                /\ SRES' = A3(Ki,Rand) =|>
     State' = 9 /\ SND(request_success)
                /\ secret(MK, A) /\ secret(MK, P)
                /\ request(A,P,mac2,Mac2')

end role
```

```
role Session(P, A                 : agent,
             Kpa                  : symmetric_key,
             SHA1,A3,A8,MAC1,MAC2 : function,
             SP, SA, RP, RA       : channel (dy))
def=

  composition
           Peer(         P,A,Kpa,SHA1,A3,A8,MAC1,MAC2,SP,RP)
        /\ Authenticator(P,A,Kpa,SHA1,A3,A8,MAC1,MAC2,SA,RA)

end role
```

```
role Environment() def=

   local
     S1,S2,S3,S4,S5,S6 : channel,
     R1,R2,R3,R4,R5,R6 : channel

   const
     p, a               : agent,
     kpa, kpi, kia      : symmetric_key,
     sha1,a3,a8,mc1,mc2 : function

   knowledge(i) = {p, a, kpi, kia}

   composition
        Session(p,a,kpa, sha1,a3,a8,mc1,mc2, S1,S2,R1,R2)
    /\  Session(p,i,kpi, sha1,a3,a8,mc1,mc2, S3,S4,R3,R4)
    /\  Session(i,a,kia, sha1,a3,a8,mc1,mc2, S5,S6,R5,R6)

end role
```

```
goal
        secrecy_of MK
        Peer authenticates Authenticator on mac1
        Authenticator authenticates Peer on mac2

end goal
```

```
Environment()
```

8.5 With TLS method

Protocol Purpose

Mutual authentication, key establishment, replay protection, confidentiality. EAP-TLS [10] is based on TLS as a mechanism designed for providing authentication and encryption scheme over TCP transport. The EAP-TLS method is developed to use the concept of TLS handshake over EAP.

Definition Reference

- http://www.ietf.org/rfc/rfc2716.txt

Model Authors

- Jing Zhang for Siemens CT IC 3, 2004
- Peter Warkentin, Siemens CT IC 3
- Vishal Sankhla, University of Southern California, 2004

Alice&Bob style

```
Let S/Ks/Ns denote id/public-key/nonce respectively of the server.
Similarly, P/Kp/Np for the Peer. Furthermore, let Kca denote the public
key of a certification authority. Then set

client_hello              : Vers.SessionID.Np.CipherSuite
server_hello              : Vers.SessionID.Ns.Cipher
client_certificate        : {C.Kc}_inv(Kca)
server_certificate        : {S.Ks}_inv(Kca)
server_key_exchange       : Ks
client_key_exchange       : {PMS}_Ks
client_certificate_verify : {H(Ns.S.PMS)}_inv(Kp)

S -> P: Identity
P -> S: UserId
S -> P: Start
P -> S: client_hello
S -> P: server_hello,
        server_certificate,
        server_key_exchange,
        server_certificate_request,
        server_hello_done
 P -> S: client_certificate,
        client_key_exchange,
        client_certificate_verify,
        change_cipher_spec,
        finished
 S -> P: change_cipher_spec,
        finished
```

Model Limitations

- no modelling of session-resumption, i.e setting session-id=0
- no explicit modelling of ciphersuite-selection

Problems considered: 3

- secrecy of `ClientK, ServerK`
- `Peer` authenticates `Server` on `nps1`
- `Server` authenticates `Peer` on `nps2`

Attacks Found

None

Further Notes

This protocol sets up the communication between two agents, in the following called Peer and Server. It is used to authenticate the Server and (optionally) the Peer. Furthermore, a set of keys is established for future encryption and data integrity. Initially, in `client_hello` and `server_hello`, the Peer and Server exchange and agree on versions-numbers, cipher-suites, session-ids. Furthermore, they exchange nonces Np, Nc which are used later on for key generation. The Server sends a certificate to the Peer for authentication. The Server may (optionally) ask the Peer to authenticate himself. On receipt of the Server's message, the Peer checks the Server's certificate and (if asked) sends his own certificate together with verify-data to the Server. The Peer generates a new secret PMS and sends it (encrypted) to the Server. Based on Np, Ns, PMS both parties are now able to compute the new session keys. They both close the protocol by sending a final message "Finished" encrypted with the new keys.

HLPSL Specification

```
role Peer (P, S           : agent,
           H, PRF, KeyGen : function,
           Kp, Kca        : public_key,
           SND_S, RCV_S   : channel (dy))
played_by P def=

  local  Np, Csus, PMS                 : text (fresh),
         SeID                          : text,
         Ns, TNo, Csu, Sh, Rcert       : text,
         Sc, Ske, Cke, Cv, Shd, Ccs    : text,
         State                         : nat,
         Finished, ClientK, ServerK    : message,
         Ks                            : public_key,
         Nps                           : text.text

  const nps1, nps2 : protocol_id,
        sid0       : text,  % session id = 0
        id_request : text,
        start_tls  : text

  init State = 0

  knowledge(P)={Kp,inv(Kp),Kca,{P.Kp}_inv(Kca)}

  transition

  0. State = 0 /\ RCV_S( id_request ) =|>
     State'= 2 /\ SND_S(P)

  2. State = 2 /\ RCV_S(start_tls) =|>
     State'= 4 /\ SND_S( TNo'.sid0.Np'.Csus' )    % client hello (SeID=0)

  % with client authentication
  41. State = 4 /\ RCV_S(
                TNo.SeID'.Ns'.Csu'.         % server hello
                {S.Ks'}_inv(Kca).           % server certificate
                Ske'.                       % server key exchange
                Rcert'.                     % server certificate request
                Shd')                       % server hello done
     =|>
      State'= 6
       /\ Finished' = H(PRF(PMS'.Np.Ns').P.S.Np.Csu'.SeID')
       /\ ClientK'  = KeyGen(P.Np.Ns'.PRF(PMS'.Np.Ns'))
       /\ ServerK'  = KeyGen(S.Np.Ns'.PRF(PMS'.Np.Ns'))
       /\ SND_S({P.Kp}_inv(Kca).            % client certificate
                {PMS'}_Ks'.                 % client key exchange
                {H(Ns'.S.PMS')}_inv(Kp).    % client certificate verify
                Ccs'.                       % change cipher spec
                {Finished'}_ClientK')       % finished
       /\ witness(P,S,nps2,Np.Ns')

  % without client authentication
  42. State = 4 /\ RCV_S(
                TNo.SeID'.Ns'.Csu'.         % server hello
                {S.Ks'}_inv(Kca).           % server certificate
                Ske'.                       % server key exchange
                Shd')                       % server hello done
     =|>
      State'= 6
       /\ Finished' = H(PRF(PMS'.Np.Ns').P.S.Np.Csu'.SeID')
       /\ ClientK'  = KeyGen(P.Np.Ns'.PRF(PMS'.Np.Ns'))
       /\ ServerK'  = KeyGen(S.Np.Ns'.PRF(PMS'.Np.Ns'))
       /\ SND_S({PMS'}_Ks'.                 % client key exchange
               %{H(Ns'.S.PMS')}_inv(Kp).    % client certificate verify
                Ccs'.                       % change cipher spec
                {Finished'}_ClientK')       % finished
       /\ witness(P,S,nps2,Np.Ns')

  6. State = 6 /\ RCV_S(Ccs.{Finished}_ServerK) =|>
     State'= 8 /\ secret(ClientK,S) /\ secret(ServerK,S)
               /\ request(P,S,nps1,Np.Ns)

end role
```

```
role Server (P, S           : agent,
             H, PRF, KeyGen : function,
             Ks, Kca        : public_key,
             SND_P, RCV_P   : channel (dy))
played_by S def=

  local  Ns, SeID                        : text (fresh),
         PMS                             : text,
         Np, Csus, TNo, Csu, Sh, Sc, Ske : text,
         Cke, Cv, Ccs, Rcert,Shd         : text,
         State                           : nat,
         Finished, ClientK, ServerK      : message,
         Kp                              : public_key

  const nps1, nps2 : protocol_id,
        sid0       : text,  % session id = 0
        id_request : text,
        start_tls  : text

  init State = 1

  knowledge(S)={Ks,inv(Ks),Kca,{S.Ks}_inv(Kca)}

  transition

  1. State = 1 /\ RCV_P(start) =|>
     State'= 3 /\ SND_P(id_request)

  3. State = 3 /\ RCV_P(P) =|>
     State'= 5 /\ SND_P(start_tls)

  % with client authentication
  51. State = 5 /\ RCV_P(TNo'.sid0.Np'.Csus')     % client hello
     =|>
      State'= 7
       /\ SND_P(TNo'.SeID'.Ns'.Csu'.      % server hello
                {S.Ks}_inv(Kca).          % server certificate
                Ske'.                     % server key exchange
                Rcert'.                   % server certificate request
```

```
              Shd')                     % server hello done
      /\ witness(S,P,nps1,Np'.Ns')

  % without client authentication
  52. State = 5
      /\ RCV_P(TNo'.sid0.Np'.Csus')     % client hello
    =|>
     State'= 9
      /\ SND_P(TNo'.SeID'.Ns'.Csu'.     % server hello
              {S.Ks}_inv(Kca).          % server certificate
              Ske'.                     % server key exchange
              Shd')                     % server hello done
      /\ witness(S,P,nps1,Np'.Ns')

  % with client authentication
  7. State = 7
      /\ RCV_P({P.Kp'}_inv(Kca).        % client certificate
              {PMS'}_Ks.                % client key exchange
              {H(Ns.S.PMS')}_inv(Kp').  % client certificate verify
              Ccs'.                     % change cipher spec
              {Finished'}_ClientK'      % finished
             )
      /\ Finished' = H(PRF(PMS'.Np.Ns).P.S.Np.Csu.SeID)
      /\ ClientK'  = KeyGen(P.Np.Ns.PRF(PMS'.Np.Ns))
    =|>
    State' = 11
      /\ ServerK'  = KeyGen(S.Np.Ns.PRF(PMS'.Np.Ns))
      /\ SND_P(Ccs'.{Finished'}_ServerK')
      /\ request(S,P,nps2,Np.Ns)

  % without client authentication
  9. State = 9
      /\ RCV_P({PMS'}_Ks.               % client key exchange
              %{H(Ns.S.PMS')}_inv(Kp).  % client certificate verify
              Ccs'.                     % change cipher spec
              {Finished'}_ClientK'      % finished
             )
      /\ Finished' = H(PRF(PMS'.Np.Ns).P.S.Np.Csu.SeID)
      /\ ClientK'  = KeyGen(P.Np.Ns.PRF(PMS'.Np.Ns))
    =|>
    State' = 11
```

```
       /\ ServerK'  = KeyGen(S.Np.Ns.PRF(PMS'.Np.Ns))
       /\ SND_P(Ccs'.{Finished'}_ServerK')
      %/\ request(S,P,nps2,Np.Ns)

end role
```

```
role Session(P, S           : agent,
             Kp, Ks, Kca    : public_key,
             H, PRF, KeyGen : function,
             SP, SS, RP, RS : channel (dy)) def=

  composition
           Peer(  P,S,H,PRF,KeyGen,Kp,Kca,SP,RP)
        /\ Server(P,S,H,PRF,KeyGen,Ks,Kca,SS,RS)

end role
```

```
role Environment() def=

   local S1,S2,S3,S4,S5,S6,R1,R2,R3,R4,R5,R6 : channel (dy)

   const p,s             : agent,
         kp, ks, ki, kca : public_key,
         h,prf,keygen    : function

   knowledge(i) = { p, s, kp, ks, kca, ki, inv(ki) }

   composition
        Session(p,s,kp,ks,kca,h,prf,keygen,S1,S2,R1,R2)
    /\  Session(p,i,kp,ki,kca,h,prf,keygen,S3,S4,R3,R4)
    /\  Session(i,s,ki,ks,kca,h,prf,keygen,S5,S6,R5,R6)

end role
```

```
goal
```

```
        secrecy_of ClientK, ServerK
        Peer authenticates Server on nps1
        Server authenticates Peer on nps2
end goal
```

```
Environment()
```

8.6 With TTLS authentication via Tunneled CHAP

Protocol Purpose

Mutual authentication, key establishment

EAP-TTLS has been defined as an authentication protocol. It extends EAP-TLS to improve some weak points. This protocol makes use of the handshake phase in TLS to establish a secure tunnel in order to pass the identity of the user and perform the authentication protocol between client and server. The information in the tunnel is exchanged through the use of encrypted attribute-value-pairs (AVPs). In EAP-TLS, the TLS handshake may achieve mutual authentication, or it may be one-way where the server is authenticated to the client. After the secure connection is established, the server can authenticate the client by using the existing authentication infrastructure such as a back-end authentication server accessible through RADIUS. The protocol may be EAP, or any other authentication protocol, e.g. PAP, CHAP, MS-CHAP or MS-CHAP-V2. Therefore, EAP-TTLS supports the legacy password-based authentication protocols while protecting the security of these legacy protocols against eavesdropping, dictionary attack and other cryptographic attacks.

Unlike other methods, EAP-TTLS is the only method that offers the Data-Cipher-suite negotiation of the client and the TTLS Server (inside the method), to secure the link layer between the client and the authenticator while, typically, the link layer security uses the keying material derived from EAP methods.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-pppext-eap-ttls-05.txt

Model Authors

- Jing Zhang and Peter Warkentin for Siemens CT IC 3
- Vishal Sankhla (University of Southern California), 2004

Alice&Bob style

```
C <- P          : id_request
C -> P          : UserId
     P -> S     : UserId, TTLScsuP

C <- P <- S     : start_ttls
C -> P -> S     : Version.SessionID.NonceC.EAPcsuC   % client_hello
C <- P <- S     : Version.SessionID.NonceS.EAPcsuS   % server_hello
                  {S.Ks}_inv(Kca)                    % certificate
                  Ske                                % server_key_exchange
                  Shd                                % server_hello_done
C -> P -> S     : {PMS}_Ks                           % client_key_exchange
                  Ccs                                % change_cipher_spec
                  {Finished}_ClientK                 % finished
C <- P <- S     : Ccs                                % change_cipher_spec
                  {Finished}_ServerK                 % finished

C -> P -> S     : {UserName,
                   CHAP_challenge,
                   CHAP_Password,
                   TTLScsuC
                  }_ClientK
          S -> A: UserName,
                  CHAP_challenge,
                  CHAP_Password
          S <- A: Access_accept
C <- P <- S     : {TTLScsuS}_ServerK
C -> P -> S     : null_ttls
     P <- S     : TTLScsuS,
                  Data_Keying_Material,
                  eap_success
C <- P          : eap_success

with
EAPcsuC:        set of cipher suites supplied by C (for EAP)
EAPcsuS:        cipher suite selected by S (from EAPcsuC)
NonceC:         nonce created by C
NonceS:         nonce created by S
Ks:             secret symmetric key shared by C and S
Kca:            public key of certification authority
```

```
PMS:            pre-master-secret created by C (nonce)
MS:             master-secret ( =PRF(PMS,Nc,Ns) )
Finished:       H(MS.C.S.Nc.EAPcsuS.SessionID)
ClientK:        session key for client =KeyGen(C.Nc.Ns.MS)
ServerK:        session key for server =KeyGen(S.Nc.Ns.MS)
TTLScsuP:       set of cipher suites supplied by P (for ttls)
TTLScsuC:       set of cipher suites supplied by C (for ttls)
TTLScsuS:       cipher suite selected by S (for ttls)
CHAP_challenge: Tranc(CHAP_PRF(M.Txt.Nc.Ns).1.16)
ChapId:         Tranc(CHAP_PRF(M.Txt.Nc.Ns).17.17)
CHAPRs:         Chap response
CHAP_Password:  ChapId + ChapRs
```

Model Limitations

- Unprotected communication between Server and AAA.
- Abstract communication and selection of cipher suites.

Problems considered: 2

- secrecy of `ClientK, ServerK`
- `Client` authenticates `Server` on `k`

Attacks Found

None

Further Notes

- In the current modelling the communication between Server and AAA sends UName in cleartext. This violates the secrecy of UName. Furthermore, UName can be used for replay attacks. Therefore the secrecy and authentication goals in HLPSL are commented out since they lead to obvious attacks.
- The Point role is basically redundant since it only forwards received messages. In the current modelling there are only two situations where Point deviates from forwarding messages: it takes part in the negotiation of a ciphersuite and finally receives keying material for deriving keys to be used at some later time.

In a model which only uses Dolev-Yao channels forwarding transitions may be skipped: All messages come from and go to the intruder. The intruder does not gain new knowledge from forwarding transitions! Furthermore, the intruder can receive and send on all channels and thus he can bridge any forwarding transition. Therefore, in the Point role, all forwarding transitions have been skipped.

- The transitions 1 and 10 in role Point (i.e. in this case, sending and receiving of cipher suites) cause highly increased runtimes:
  ```
  OFMC     with: 6min   without 30sec
  CL_ATSE  with: 8h     without 40sec
  ```
  Since these transitions are not essential, they are skipped.

HLPSL Specification

```
role Client(C, P, S, A                         : agent,
            Kca                                : public_key,
            H, PRF, CHAP_PRF, Tranc, KeyGen : function,
            SND_PC, RCV_PC                     : channel (dy))
played_by C def=

  local
    UserId   : text,          % should not reveal user
    Version  : text,          % version of TLS protocol, presently v1.0
    SeID     : text,          % session id
    Nc       : text (fresh),  % nonce from client
    Ns       : text,          % nonce from server
    EAPcsuC  : text,          % eap-tls ciphersuites supplied by the client
    EAPcsuS  : text,          % eap-tls ciphersuite selected by server

    Ks       : public_key,    % from server

    Ske      : text,          % server-key-exchange (null)
    Shd      : text,          % server-hello-done
    Ccs      : text,          % change-cipher-spec
```

```
    PMS      : text (fresh),  % pre-master-secret
    MS       : message,       % master-secret

    Finished : message,
    ClientK  : message,       % client session key for encryption
    ServerK  : message,       % server session key for encryption

    Txt      : text (fresh),  % string init. with "ttls challenge"

    UName    : text,          % NAI of client e.g. andy@realm
    TTLScsuC : text,          % ttls data-cipher-suite from client for server
    TTLScsuS : text,          % ttls data-cipher-suite selected by server

    ChapRs   : text,          % CHAP response

    State    : nat

  const
    id_request  : text,
    start_ttls  : text,
    null_ttls   : text,
    eap_success : text,
    k, uname    : protocol_id

  init State = 0

  knowledge(C)={Kca}

  transition

  0. State = 0 /\ RCV_PC( id_request ) =|>
     State'= 1 /\ SND_PC( UserId' )

  1. State = 1 /\ RCV_PC(start_ttls) =|>
     State'= 2 /\ SND_PC(Version'.SeID'.Nc'.EAPcsuC')

  2. State = 2 /\ RCV_PC(Version.SeID'.Ns'.EAPcsuS'.{S.Ks'}_inv(Kca).Ske'.Shd')
     =|>
     State'= 3 /\ MS' = PRF(PMS'.Nc.Ns')
               /\ Finished' = H(PRF(PMS'.Nc.Ns').C.S.Nc.EAPcsuS'.SeID)
               /\ ClientK' = KeyGen(C.Nc.Ns'.PRF(PMS'.Nc.Ns'))
               /\ ServerK' = KeyGen(S.Nc.Ns'.PRF(PMS'.Nc.Ns'))
               /\ SND_PC({PMS'}_Ks'.Ccs'.{Finished'}_ClientK')
               /\ secret(ClientK',C) /\ secret(ClientK',P) /\ secret(ClientK',S)
               /\ secret(ServerK',C) /\ secret(ServerK',P) /\ secret(ServerK',S)

  3. State = 3 /\ RCV_PC(Ccs.{Finished}_ServerK) =|>
     State'= 4 /\ SND_PC({UName'.
                         Tranc(CHAP_PRF(MS.Txt'.Nc.Ns).1.16).
                         Tranc(CHAP_PRF(MS.Txt'.Nc.Ns).17.17).
                         ChapRs'.
                         TTLScsuC'
                        }_ClientK)
%       /\ secret(UName',P) /\ secret(UName',S) /\ secret(UName',A)
               /\ request(C,S,k,Nc.Ns)
               /\ witness(C,A,uname,UName')

  4. State = 4 /\ RCV_PC({TTLScsuS'}_ServerK) =|>
     State'= 5 /\ SND_PC(null_ttls)

  5. State = 5 /\ RCV_PC(eap_success) =|>
     State'= 6

end role
```

```
role Point (C, P, S, A                      : agent,
            SND_CP, RCV_CP, SND_SP, RCV_SP  : channel (dy))
played_by P def=

  local
    UserId   : text,          % should not reveal user

    Ks       : public_key,    % from server
    Kca      : public_key,    % from AuC

    TTLScsuP : text,          % ttls data-cipher-suite from point  for server
    TTLScsuS : text,          % ttls data-cipher-suite selected by server

    Dkm      : text,          % data-keying-material
    State    : nat

  const
    id_request   : text,
    eap_success  : text,
    eap_failure  : text

  init State = 20

  transition

  0. State = 20 /\ RCV_CP(start) =|>
     State'= 21 /\ SND_CP(id_request)

% 1. State = 21
%      /\ RCV_CP(UserId')
%    =|>
%    State' = 30
%      /\ SND_SP(UserId'.TTLScsuP')

%10. State = 30
%      /\ RCV_SP(TTLScsuS'.Dkm'.eap_success)
%    =|>
%    State' = 31
%      /\ SND_CP(eap_success)

end role
```

```
role Server (C, P, S, A                        : agent,
             Ks, Kca                           : public_key,
             H, PRF, CHAP_PRF, Tranc, KeyGen : function,
             SND_PS, RCV_PS, SND_AS, RCV_AS  : channel (dy))
played_by S def=

  local
    UserId   : text,          % should not reveal user
    Version  : text,          % version of TLS protocol, presently v1.0
    SeID     : text,          % session id
    Nc       : text,          % nonce from client
    Ns       : text (fresh),  % nonce from server
    EAPcsuC  : text,          % eap-tls ciphersuites supplied by the client
    EAPcsuS  : text,          % eap-tls ciphersuite selected by server

    Ske      : text,          % server-key-exchange (null)
    Shd      : text,          % server-hello-done
    Ccs      : text,          % change-cipher-spec

    PMS      : text,          % pre-master-secret
    MS       : message,       % master-secret

    Finished : message,
    ClientK  : message,       % client session key for encryption
    ServerK  : message,       % server session key for encryption

    Txt      : text,          % string init. with "ttls challenge"

    UName    : text,          % NAI of client e.g. andy@realm
    TTLScsuC : text,          % ttls data-cipher-suite from client for server
    TTLScsuP : text,          % ttls data-cipher-suite from point  for server
    TTLScsuS : text,          % ttls data-cipher-suite selected by server

    ChapRs   : text,          % CHAP response
    Dkm      : text,          % data-keying-material

    State    : nat

  const
    k, uname      : protocol_id,
    start_ttls    : text,
    null_ttls     : text,
    eap_success   : text,
    eap_failure   : text,
    access_accept : text,
    access_reject : text

  init State = 40

  knowledge(S)={inv(Ks),{S.Ks}_inv(Kca)}

  transition
  0. State = 40 /\ RCV_PS( UserId'.TTLScsuP' ) =|>
     State'= 41 /\ SND_PS(start_ttls)

  1. State = 41 /\ RCV_PS(Version'.SeID'.Nc'.EAPcsuC') =|>
     State'= 42 /\ SND_PS(Version'.SeID'.Ns'.EAPcsuS'.{S.Ks}_inv(Kca).Ske'.Shd')
                /\ witness(S,C,k,Nc'.Ns')

  2. State = 42 /\ RCV_PS({PMS'}_Ks.Ccs'.{Finished'}_ClientK')
                /\ Finished' = H(PRF(PMS'.Nc.Ns).C.S.Nc.EAPcsuS'.SeID)
                /\ ClientK'  = KeyGen(C.Nc.Ns.PRF(PMS'.Nc.Ns)) =|>
     State'= 43 /\ MS' = PRF(PMS'.Nc.Ns)
                /\ ServerK'  = KeyGen(S.Nc.Ns.PRF(PMS'.Nc.Ns))
                /\ SND_PS(Ccs'.{Finished'}_ServerK')

  3. State = 43 /\ RCV_PS({UName'.
                        Tranc(CHAP_PRF(MS.Txt'.Nc.Ns).1.16).
                        Tranc(CHAP_PRF(MS.Txt'.Nc.Ns).17.17).
                        ChapRs'.TTLScsuC'}_ClientK) =|>

     State'= 44 /\ SND_AS(UName'.
                       Tranc(CHAP_PRF(MS.Txt'.Nc.Ns).1.16).
                       Tranc(CHAP_PRF(MS.Txt'.Nc.Ns).17.17).
                       ChapRs')

  4. State = 44 /\ RCV_AS(access_accept) =|>
     State'= 45 /\ SND_PS({TTLScsuS'}_ServerK)

  5. State = 45 /\ RCV_PS(null_ttls) =|>
     State'= 46 /\ SND_PS(TTLScsuS.Dkm'.eap_success)

% 4b. State = 44
%       /\ RCV_AS(access_reject)
%     =|>
%     State' = 46
%       /\ SND_PS(eap_failure)

end role
```

```
role AAA (C, P, S, A           : agent,
          PRF, CHAP_PRF, Tranc : function,
          SND_SA, RCV_SA       : channel (dy))
played_by A def=

  local
    Nc       : text,          % nonce from client
    Ns       : text,          % nonce from server

    PMS      : text,          % pre-master-secret
    Txt      : text,          % string init. with "ttls challenge"

    UName    : text,          % NAI of client e.g. andy@realm

    ChapRs   : text,          % CHAP response

    State    : nat

  const
    uname         : protocol_id,
    access_accept : text,
    access_reject : text

  init State = 1

  transition

  1. State = 1 /\ RCV_SA(UName'.
                        Tranc(CHAP_PRF(PRF(PMS'.Nc'.Ns').Txt'.Nc'.Ns').1.16).
                        Tranc(CHAP_PRF(PRF(PMS'.Nc'.Ns').Txt'.Nc'.Ns').17.17).
                        ChapRs') =|>
     State'= 3 /\ SND_SA(access_accept)
%              /\ request(A,C,uname,UName')

% 1b. State = 1
%      /\ RCV_SA(UName'.
%                Tranc(CHAP_PRF(PRF(PMS'.Nc'.Ns').Txt'.Nc'.Ns').1.16).
%                Tranc(CHAP_PRF(PRF(PMS'.Nc'.Ns').Txt'.Nc'.Ns').17.17).
%                ChapRs' )
%      =|>
%    State' = 3
%       /\ SND_SA(access_reject)
%       /\ request(A,C,uname,UName')

end role
```

```
role Session(C, P, S, A: agent,
             Ks, Kca: public_key,
             H, PRF, CHAP_PRF, Tranc, KeyGen: function)
def=

  local
    SND_PC, RCV_PC,                     % channels Client -> Point
    SND_CP, RCV_CP,                     % channels Point  -> Client
    SND_SP, RCV_SP,                     % channels Point  -> Server
    SND_PS, RCV_PS,                     % channels Server -> Point
    SND_AS, RCV_AS,                     % channels Server -> AAA
    SND_SA, RCV_SA  : channel (dy)      % channels AAA    -> Server

  composition
           Client(C,P,S,A,Kca,H,PRF,CHAP_PRF,Tranc,KeyGen,SND_PC,RCV_PC)
        /\ Point( C,P,S,A,SND_CP,RCV_CP,SND_SP,RCV_SP)
        /\ Server(C,P,S,A,Ks,Kca,H,PRF,CHAP_PRF,Tranc,KeyGen,
                  SND_PS,RCV_PS,SND_AS,RCV_AS)
        /\ AAA(   C,P,S,A,PRF,CHAP_PRF,Tranc,SND_SA,RCV_SA)

end role
```

```
role Environment() def=

   const c, p, s, a                      : agent,
         ks, kca                         : public_key,
         h, prf, chapprf, tranc, keygen : function

   knowledge(i) = { c, p, s, a, ks, kca,
                    h, prf, chapprf, tranc, keygen }

   composition
```

```
        Session(c,p,s,a,ks,kca,h,prf,chapprf,tranc,keygen)
%    /\ Session(c,p,s,a,ks,kca,h,prf,chapprf,tranc,keygen)
     /\ Session(i,p,s,a,ks,kca,h,prf,chapprf,tranc,keygen)

end role
```

```
goal

        secrecy_of ClientK, ServerK
       %secrecy_of                 UName
        Client authenticates Server on k
       %AAA authenticates Client on uname

end goal
```

```
Environment()
```

8.7 Protected with MS-CHAP authentication

Protocol Purpose

Mutual authentication, key establishment

Similar to EAP-TTLS, PEAP performs two phases of authentication. The first phase is to create the TLS secure channel. The server is authenticated by certificate in this phase and optionally the client can be authenticated also based on a client certificate. In the second phase, within the TLS secured tunnel, a complete EAP conversation is carried out. The user, which is not authenticated in the first phase, will be authenticated securely inside a TLS channel by EAP method. If the user is already authenticated in the first phase, PEAP does not run EAP method to authenticate the user. In PEAP, it runs only EAP methods, e.g. EAP-MD5, EAP-SIM, to authenticate the client inside the secure tunnel but does not supports non-EAP methods like PAP, CHAP. In case the authentication is held through the access point, it does not need to have any knowledge of the TLS master secret derived between the client and back-end authentication server. The access point simply then acts as the pass-through device and cannot decrypt the PEAP conversation. However, the access point obtains the master session keys, derived from the

TLS master secret.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-josefsson-pppext-eap-tls-eap-10.txt

Model Authors

- Jing Zhang for Siemens CT IC 3
- Vishal Sankhla (University of Southern California), 2004

Alice&Bob style

```
PEAP Phase 1:
S -> P: id_request
P -> S: P
S -> P: start_peap
P -> S: client_hello
S -> P: server_hello, certificate
P -> S: certificate_verify, change_cipher_spec
S -> P: change_cipher_spec, finished

PEAP Phase 2:
P -> S: {P}_ClientK
S -> P: {Rand_S}_ServerK
P -> S: {Rand_P,Hash(k(P,S),(Rand_P,Rand_S,P)}_ClientK
S -> P: {Hash(k(P,S),Rand_P)}_ServerK
P -> S: {Ack}_ClientK
S -> P: {Eap_Success}_ServerK

client_hello = {TlsVNo, SessionID, NonceC, CSu}
server_hello = {TlsVNo, SessionID, NonceS, CSu}
CSu:  a set of eap-tls ciphersuites supplied by the client
    or  a eap-tls ciphersuite selected by the server
certificate = {S.Ks}_inv(Kca)
SessionID+Rand_S is the MS challenge packet
```

Problems considered: 3

- secrecy of `ClientK, ServerK`
- `Peer` authenticates `Server` on `np_ns`
- `Server` authenticates `Peer` on `ns`

Attacks Found

None

HLPSL Specification

```
role Peer ( P, S: agent,
            H1, H2, PRF, KeyGen: function,
            Pw: symmetric_key,
            Kca: public_key,
            SND_S, RCV_S : channel (dy)) played_by P def=

  local Np, PMS: text (fresh),
        SeID, Csu, Ns: text,
        Ccs: text,
        %Ccs, change-cipher-spec, value=1 means cipher suites changed

        M, Finished, ClientK, ServerK: message,
        %M, master secret, calculated by both from PMS and nonces

        Ks: public_key,
        Np_Ns: text.text,
        State: nat

  const np_ns, ns                     : protocol_id,
        id_request, start_peap        : text,
        ack_message, eap_success      : text

  %owns SND_S
  init State = 0
  knowledge(P)={Kca}

  transition

  1. State = 0 /\ RCV_S(id_request) =|>
     State'= 2 /\ SND_S(P)

  2. State = 2 /\ RCV_S(start_peap) =|>
     State'= 4 /\ SND_S(Np'.SeID'.Csu')

  3. State = 4 /\ RCV_S(Ns'.SeID'.Csu'.{S.Ks'}_inv(Kca)) =|>
        SND_S({PMS'}_Ks'.Ccs') /\
        M' = PRF(PMS'.Np.Ns') /\
        Finished' = H1(PRF(PMS'.Np.Ns').P.S.Np.Csu'.SeID') /\
        ClientK' = KeyGen(P.Np.Ns'.PRF(PMS'.Np.Ns')) /\
        ServerK' = KeyGen(S.Np.Ns'.PRF(PMS'.Np.Ns')) /\
        Np_Ns' = Np.Ns'/\
        State' = 6

  4. State = 6 /\ RCV_S(Ccs.{Finished}_ServerK) =|>
     State'= 8 /\ SND_S({P}_ClientK)
               /\ secret(ClientK,S) /\ secret(ServerK,S)
               /\ request(P,S,np_ns,Np_Ns)
%here we assume both of peer and server have finished
%negotiation of authentication method, that is Ms-chap
%An attacker will also not be able to determine which
%EAP method was negotiated.

  5. State = 8 /\ RCV_S({Ns'}_ServerK) =|>
     State'= 10/\ SND_S({Np'.H2(Pw.Np'.Ns'.P)}_ClientK)
               /\ witness(P,S,ns,Ns')

  6. State = 10 /\ RCV_S({H2(Pw.Np)}_ServerK) =|>
     State'= 12 /\ SND_S( ack_message )

% 7. State = 10 /\ RCV_S(eap_failure) =|>
%    State'= 14

  8. State = 12 /\ RCV_S(eap_success) =|>
     State'= 14
```

```
end role
```

```
role Server (S, P : agent,
             H1, H2, PRF, KeyGen: function,
             Pw: symmetric_key,
             Ks, Kca: public_key,
             SND_P, RCV_P : channel (dy))
             played_by S def=

  local Ns: text (fresh),
        Np, SeID, Csu, PMS: text,
        Ccs: text,
        M, Finished, ClientK, ServerK: message,
        Np_Ns: text.text,
        State: nat

  const np_ns, ns                   : protocol_id,
        id_request, start_peap      : text,
        ack_message, eap_success    : text

  %owns SND_P
  init State = 1

  knowledge(S)={inv(Ks),{S.Ks}_inv(Kca)}

  transition

  1. State = 1 /\ RCV_P(start) =|>
     State'= 3 /\ SND_P(id_request)

  2. State = 3 /\ RCV_P(P) =|>
     State'= 5 /\ SND_P(start_peap)

  3. State = 5 /\ RCV_P( Np'.SeID'.Csu' ) =|>
     State'= 7 /\ SND_P(Ns'.SeID'.Csu'.{S.Ks}_inv(Kca))
               /\ Np_Ns' = Np'.Ns'
               /\ witness(S,P,np_ns,Np_Ns')
  4. State = 7 /\ RCV_P({PMS'}_Ks.Ccs') =|>
        SND_P(Ccs'.{H1(PRF(PMS'.Np.Ns).P.S.Np.Csu.SeID)}_
               KeyGen(S.Np.Ns.PRF(PMS'.Np.Ns))) /\
        M' = PRF(PMS'.Np.Ns) /\
        Finished' = H1(PRF(PMS'.Np.Ns).P.S.Np.Csu.SeID) /\
        ServerK' = KeyGen(S.Np.Ns.PRF(PMS'.Np.Ns)) /\
        ClientK' = KeyGen(P.Np.Ns.PRF(PMS'.Np.Ns)) /\
        State' = 9

  5. State = 9  /\ RCV_P({P}_ClientK) =|>
     State'= 11 /\ SND_P({Ns'}_ServerK)

  6. State = 11 /\ RCV_P({Np'.H2(Pw.Np'.Ns.P)}_ClientK) =|>
     State'= 13 /\ SND_P({H2(Pw.Np')}_ServerK)
                /\ request(S,P,ns,Ns)

% 7. State = 11 /\ RCV_P({Np'.H2(Pw.Np'.Ns.P)}_ClientK) =|>
%    State'= 15 /\ SND_P(eap_failure)

   7. State = 13 /\ RCV_P(ack_message) =|>
      State'= 15 /\ SND_P(eap_success)

end role
```

```
role Session(P, S: agent,
             Pw: symmetric_key,
             Ks, Kca: public_key,
             H1,H2, PRF, KeyGen: function,
             S_SP, R_SP, S_PS, R_PS: channel (dy)) def=

  composition
           Peer(P,S,H1,H2,PRF,KeyGen,Pw,Kca,S_SP,R_SP)
        /\ Server(S,P,H1,H2,PRF,KeyGen,Pw,Ks,Kca,S_PS,R_PS)

end role
```

```
role Environment() def=
   const p,s,i                                 : agent,
         kpi,kps,kis                           : symmetric_key,
         ks,ki,kca                             : public_key,
         h1,h2,prf,keygen                      : function,
         s1,s2,s3,s4,s5,s6,r1,r2,r3,r4,r5,r6: channel

   knowledge(i) = { p, s, kca, ks, ki, inv(ki) }
   composition
        Session(p,s,kps,ks,kca,h1,h2,prf,keygen,s1,r1,s2,r2)
    /\  Session(p,i,kpi,ki,kca,h1,h2,prf,keygen,s3,r3,s4,r4)
    /\  Session(i,s,kis,ks,kca,h1,h2,prf,keygen,s5,r5,s6,r6)

end role
```

```
goal
        secrecy_of ClientK, ServerK
        Peer authenticates Server on np_ns
        Server authenticates Peer on ns
end goal
```

```
Environment()
```

9 S/Key One-Time Password System

Protocol Purpose

Mechanism for providing replay protection, authentication and secrecy by generating a sequence of one-time passwords.

Definition Reference

- RFC 1760: http://www.faqs.org/rfcs/rfc1760.html

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004

Alice&Bob style

Given:

```
 - Passwd   : password only known to client
 - Seed     : a nonce supplied by server
 - MD4      : one-way hash function
 - Secret   : secret generated by client (=MD4(Passwd.Seed))
 - Nmax     : maximal number of one-time passwords (here Nmax=6)
 - OTP(N)   : N-th one-time password (N=1,2,..Nmax)
              obtained by applying MD4 (Nmax-N)-times to Secret,
              i.e. OTP(N) = MD4^(Nmax-N)(Secret).
 Initially, S knows OTP(1) = MD4^5(Secret) (here: Nmax = 6).

 C -> S : C
 S -> C : N.Seed
          % challenge of S to C for authentication:
          % C is asked to send N-th OTP (wrt Seed)
          % here: C is asked for next OTP
 C -> S : OTP(N)
          % S knows previous one-time password OTP(N-1)
          % and checks validity, i.e MD4(OTP(N)) = OTP(N-1)
 S -> C : Success
```

Model Limitations

- maximal number Nmax of one-time passwords limited (Nmax = 6)
- no re-initialisation if one-time passwords exhausted
- challenge always concerns current OTP

Problems considered: 1

- `Server` authenticates `Client` on `m`

Attacks Found

None

Further Notes

The protocol consists of two agents: a client and a server. The client computes a secret based on a seed (supplied by the server) and his own password, i.e. `Secret = MD4(Passwd.Seed)`. For a given Nmax, the client further computes a sequence of Nmax one-time passwords OTP(1),..., OTP(Nmax) by repeatedly applying the hash function to this secret (see above). Initially, the server is given the first one-time password OTP(1) and stores it as the current OTP. In following protocol steps, whenever the client is asked to authenticate himself to the server, he sends the next unused OTP. The server checks the validity of the received OTP by applying MD4 and comparing the result with the previously sent OTP - these must coincide! Thereafter, the server stores the obtained OTP as the current one.

The server may ask for a the N-th OTP by supplying N in his challenge. This cannot be easily modelled within the current framework.

HLPSL Specification

```
role Client(
    C,S        : agent,
    MD4        : function,
    Secret     : message,
    SEED       : text,
    SUCCESS   : text,
    SND, RCV  : channel(dy))
played_by C def=

  local
    State  : nat,
    M      : message

  const
    m      : protocol_id

  init
    State = 0

  transition

 0. State  = 0 /\ RCV(start) =|>
    State' = 1 /\ SND(C)

 1. State  = 1 /\ RCV(SEED) =|>
    State' = 2 /\ M' = MD4(MD4(MD4(MD4(Secret))))
               /\ SND(M')
               /\ witness(C,S,m,M')

 2. State  = 2 /\ RCV(SUCCESS) =|>
    State' = 3 /\ SND(C)

 3. State  = 3 /\ RCV(SEED) =|>
    State' = 4 /\ M' = MD4(MD4(MD4(Secret)))
               /\ SND(M')
               /\ witness(C,S,m,M')

 4. State  = 4 /\ RCV(SUCCESS) =|>
    State' = 5 /\ SND(C)

 5. State  = 5 /\ RCV(SEED) =|>
    State' = 6 /\ M' = MD4(MD4(Secret))
               /\ SND(M')
               /\ witness(C,S,m,M')

 6. State  = 6 /\ RCV(SUCCESS) =|>
```

```
    State' = 7

end role
```

```
role Server(
    C,S       : agent,
    MD4       : function,
    OTP       : message,
    SEED      : text,
    SUCCESS   : text,
    SND,RCV   : channel(dy))
played_by S def=

  local
    State     : nat,
    M         : message

  const
    m         : protocol_id

  init
    State = 10

  transition

 1. State  = 10 /\ RCV(C) =|>
    State' = 11 /\ SND(SEED)

 2. State  = 11 /\ RCV(M') /\ OTP = MD4(M') =|>
    State' = 10 /\ OTP' = M'
                /\ SND(SUCCESS)
                /\ request(S,C,m,M')

end role
```

```
role Session (
    C,S     : agent,
    MD4     : function,
    Passwd  : text,
    SUCCESS : text,
    SEED    : text)
def=

  local
    OTP      : message,
    Secret   : message,
    S1, S2   : channel (dy),
    R1, R2   : channel (dy)

  init
    OTP    = MD4(MD4(MD4(MD4(MD4(MD4(Passwd.SEED))))))
 /\ Secret = MD4(Passwd.SEED)

  composition

    Client(C,S,MD4,Secret,SEED,SUCCESS,S1,R1)
 /\ Server(C,S,MD4,OTP,   SEED,SUCCESS,S2,R2)

end role
```

```
role Environment() def=

  const
    c1,s1    : agent,
    c2,s2    : agent,
    md4      : function,
    passwd1  : text,
    passwd2  : text,
    success  : text,
    seed1    : text,
    seed2    : text

  knowledge(i) = {c1,s1,c2,s2,md4,success}

  composition
```

```
    Session(c1,s1,md4,passwd1,success,seed1)
 /\ Session(i, s1,md4,passwd2,success,seed2)

end role
```

```
goal

  Server authenticates Client on m

end goal
```

```
Environment()
```

10 EKE: Encrypted Key Exchange

10.1 basic

Protocol Purpose

Encrypted key exchange

Definition Reference

http://citeseer.ist.psu.edu/bellovin92encrypted.html

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Sebastian Mödersheim, ETH Zürich, December 2003

Alice&Bob style

```
A -> B : {Ea}_Kab        |    Key exchange part
B -> A : {{K}_Ea}_Kab    |
A -> B : {Ca}_K          |
B -> A : {Ca,Cb}_K       |    Challenge/Response
A -> B : {Cb}_K          |    Authentication part
```

Model Limitations

None

Problems considered: 3

- secrecy of `K`
- `EKE_Init` authenticates `EKE_Resp` on `nb`
- `EKE_Resp` authenticates `EKE_Init` on `na`

Attacks Found

```
 i -> (a,3): start
 (a,3) -> i: {Ea(1)}_kab
 i -> (a,6): {Ea(1)}_kab
 (a,6) -> i: {{K(2)}_Ea(1)}_kab
 i -> (a,3): {{K(2)}_Ea(1)}_kab
 (a,3) -> i: {Na(3)}_K(2) witness(a,b,na,Na(3))
 i -> (a,6): {Na(3)}_K(2)
 (a,6) -> i: {Na(3),Nb(4)}_K(2) witness(a,b,nb,Nb(4))
 i -> (a,3): {Na(3),Nb(4)}_K(2)
 (a,3) -> i: {Nb(4)}_K(2)  request(a,b,nb,Nb(4))
```

Parallel session attack, man-in-the-middle between A as initiator and A as responder, attacker masquerades as B, but no secret nonces are exposed.

HLPSL Specification

```
role EKE_Init (A,B: agent,
               Kab: symmetric_key,
               Snd,Rcv: channel(dy)) played_by A def=

  local State : nat,
        Ea    : public_key (fresh),
        Na    : text (fresh),
        Nb,K  : text

  init  State = 0
  knowledge(A)={A,B,Kab}

  transition

   1. State = 0 /\ Rcv(start) =|>
      State'= 1 /\ Snd({Ea'}_Kab)

   2. State = 1 /\ Rcv({{K'}_Ea}_Kab) =|>
      State'= 2 /\ Snd({Na'}_K')
                /\ secret(K',A) /\ secret(K',B)
                /\ witness(A,B,na,Na')
   3. State = 2 /\ Rcv({Na.Nb'}_K) =|>
      State'= 3 /\ Snd({Nb'}_K)
                /\ request(A,B,nb,Nb')

end role
```

```
role EKE_Resp (B,A: agent,
               Kab: symmetric_key,
               Snd,Rcv: channel(dy)) played_by B def=

  local State : nat,
        Nb,K : text (fresh),
          Na : text,
          Ea : public_key

  init State = 0
  knowledge(B) = {A,B,Kab}

  transition

   1. State = 0 /\ Rcv({Ea'}_Kab) =|>
      State'= 1 /\ Snd({{K'}_Ea'}_Kab)
                /\ secret(K',A) /\ secret(K',B)

   2. State = 1 /\ Rcv({Na'}_K) =|>
      State'= 2 /\ Snd({Na'.Nb'}_K)
                /\ witness(B,A,nb,Nb')

   3. State = 2 /\ Rcv({Nb}_K) =|>
      State'= 3 /\ request(B,A,na,Na)

end role
```

```
role Session(A,B: agent,
             Kab: symmetric_key,
             SA, RA, SB, RB: channel (dy)) def=
  composition
     EKE_Init(A,B,Kab,SA,RA)
  /\ EKE_Resp(B,A,Kab,SB,RB)
end role
```

```
role Environment() def=

  local Sa1,Ra1,Sb1,Rb1,
        Sa2,Ra2,Sb2,Rb2 : channel (dy)

  const na, nb: protocol_id

  knowledge(i)={a,b}

  composition
      Session(a,b,kab,Sa1,Ra1,Sb1,Rb1)
   /\ Session(b,a,kab,Sa2,Ra2,Sb2,Rb2)

end role
```

```
goal

 secrecy_of K
 EKE_Init authenticates EKE_Resp on nb
 EKE_Resp authenticates EKE_Init on na

end goal
```

```
Environment()
```

10.2 EKE2 (with mutual authentication)

Protocol Purpose

Encrypted key exchange with mutual authentication

Definition Reference

http://citeseer.ist.psu.edu/bellare00authenticated.html

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Sebastian Mödersheim, ETH Zürich, December 2003

Alice&Bob style

```
1. A -> B : A.{exp(g,X)}_K(A,B)

    B computes master key MK
    MK = H(A,B,exp(g,X),exp(g,Y),exp(g,XY))

2. B -> A : {exp(g,Y)}_K(A,B), H(MK,1)

     A computes master key MK

3. A -> B : H(MK,2)

     Session key K = H(MK,0)

H : hash function
K(A,B): password (shared key)
```

Model Limitations

None

Problems considered: 3

- secrecy of `MK`
- `Eke2_Init` authenticates `Eke2_Resp` on `mk_a`
- `Eke2_Resp` authenticates `Eke2_Init` on `mk_b`

Attacks Found

None

Further Notes

For information, this protocol is an example of the proposition done in http://citeseer.ist.psu.edu/bellare00authenticated.html showing that any secure AKE (Authentication Key Exchange) protocol can be easily improved to also provide MA (Mutual Authentication).

HLPSL Specification

```
role Eke2_Init (A,B : agent,
                G: text,
                H: function,
                Kab : symmetric_key,
                Snd,Rcv: channel(dy)) played_by A def=

  local State   : nat,
        X         : text (fresh),
        GY        : message,
        MK_A,MK_B : message

  const two : text
  init   State = 0
  knowledge(A) = {A,B,Kab,two}

  transition

   1. State = 0 /\ Rcv(start) =|>
      State'= 1 /\ Snd(A.{exp(G,X')}_Kab)
   2. State = 1 /\ Rcv({GY'}_Kab.H(H(A.B.exp(G,X).GY'.exp(GY',X)).one)) =|>
      State'= 2 /\ MK_A' = A.B.exp(G,X).GY'.exp(GY',X)
                /\ MK_B' = MK_A'
                /\ Snd(H(H(MK_A').two))
                /\ secret(MK_A',A) /\ secret(MK_A',B)
                /\ request(A,B,mk_a,MK_A')
                /\ witness(A,B,mk_b,MK_B')
end role
```

```
role Eke2_Resp (B,A : agent,
                G: text,
                H: function,
                Kab : symmetric_key,
                Snd,Rcv : channel(dy)) played_by B def=

  local State     : nat,
         Y         : text (fresh),
         GX        : message,
         MK_A,MK_B : message

  const one : text
  init  State = 0
  knowledge(B) = {A,B,Kab,one}

  transition

   1. State = 0 /\ Rcv(A.{GX'}_Kab) =|>
      State'= 1 /\ MK_B' = A.B.GX'.exp(G,Y').exp(GX',Y')
                /\ Snd({exp(G,Y')}_Kab.H(H(MK_B').one))
                /\ secret(MK_B',A) /\ secret(MK_B',B)
                /\ witness(B,A,mk_a,MK_A') /\ MK_A' = MK_B'

   2. State = 1 /\ Rcv(H(H(MK_B).two)) =|>
      State'= 2 /\ request(B,A,mk_b,MK_B)

end role
```

```
role Session (A,B: agent,
              G: text,
              H: function,
              Kab: symmetric_key,
              SA,RA,SB,RB: channel(dy) ) def=

   composition

           Eke2_Init(A,B,G,H,Kab,SA,RB)  /\
           Eke2_Resp(B,A,G,H,Kab,SB,RA)

end role
```

```
role Environment() def=

  const mk_a, mk_b: protocol_id

  knowledge(i)  = {a,b,c}

  composition

        Session(a,b,g,h,kab,sa1,ra1,sb1,rb1) /\
        Session(a,i,g,h,kai,sa3,ra3,sa3,ra3) /\
        Session(i,b,g,h,kib,sb4,rb4,sa4,ra4)

end role
```

```
goal

  secrecy_of MK
  Eke2_Init authenticates Eke2_Resp on mk_a
  Eke2_Resp authenticates Eke2_Init on mk_b

end goal
```

```
Environment()
```

10.3 SPEKE (with strong password-only authentication)

Protocol Purpose

Strong Password-Only Authenticated Key Exchange

Definition Reference

http://citeseer.ist.psu.edu/jablon96strong.html

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Sebastian Mödersheim, ETH Zürich, December 2003

Alice&Bob style

```
 A -> B : exp(S(A,B), Na)     |    key exchange part
 B -> A : exp(S(A,B), Nb)     |

               both A and B compute
               K = exp(exp(S(A,B),Na), Nb) = exp(exp(S(A,B),Nb), Na)

 A -> B : {Ca}_K              |
 B -> A : {Cb,Ca}_K           |    challenge/response
 A -> B : {Cb}_K              |    authentication part

 S(A,B): password (shared key)
```

Model Limitations

None

Problems considered: 3

- secrecy of `Ca, Cb`
- `SPEKE_Init` authenticates `SPEKE_Resp` on `cb`
- `SPEKE_Resp` authenticates `SPEKE_Init` on `ca`

Attacks Found

None

Further Notes

None

HLPSL Specification

```
role SPEKE_Init (A,B: agent,
                 Kab: symmetric_key,
                 Snd,Rcv: channel(dy)) played_by A def=

  local  State: nat,
         Na,Ca: text(fresh),
         Cb   : text,
         X,K  : message

  init  State = 0
  knowledge(A) = {A,B,Kab}

  transition

   1. State = 0 /\ Rcv(start) =|>
      State'= 1 /\ Snd(exp(Kab, Na'))

   2. State = 1 /\ Rcv(X') =|>
      State'= 2 /\ K'=exp(X',Na)
                /\ Snd({Ca'}_exp(X',Na))
                /\ secret(Ca',A) /\ secret(Ca',B)
                /\ witness(A,B,ca,Ca')
   3. State = 2 /\ Rcv({Cb'.Ca}_K ) =|>
      State'= 3 /\ Snd({Cb'}_K)
                /\ secret(Cb',A) /\ secret(Cb',B)
                /\ request(A,B,cb,Cb')

end role
```

```
role SPEKE_Resp (A,B: agent,
                 Kab: symmetric_key,
                 Snd,Rcv: channel(dy)) played_by B def=

  local State: nat,
        Nb,Cb: text(fresh),
        Ca   : text,
        Y,K  : message

  init  State = 0
  knowledge(B) = {A,B,Kab}

  transition

   1. State = 0 /\ Rcv(Y') =|>
      State'= 1 /\ Snd(exp(Kab, Nb'))
                /\ K' = exp(Y', Nb')

   2. State = 1 /\ Rcv({Ca'}_K) =|>
      State'= 2 /\ Snd({Cb'.Ca'}_K)
                /\ secret(Ca',A) /\ secret(Ca',B)
                /\ secret(Cb',A) /\ secret(Cb',B)
                /\ witness(B,A,cb,Cb')
                /\ request(B,A,ca,Ca')

   3. State = 2 /\ Rcv({Cb}_K) =|>
      State'= 3

end role
```

```
role Session (A,B: agent,
              Kab: symmetric_key,
              SA,RA,SB,RB: channel (dy)) def=

   composition

           SPEKE_Init(A,B,Kab,SA,RA)
        /\ SPEKE_Resp(A,B,Kab,SB,RB)

end role
```

```
role Environment() def=

  local Sa1,Ra1,Sb1,Rb1,Sa2,Ra2,
        Si2,Ri2,Si3,Ri3,Sb3,Rb3 : channel (dy)

  const ca,cb: protocol_id

  knowledge(i) = {a, b, kai, kbi}

  composition
        Session(a,b,kab,Sa1,Ra1,Sb1,Rb1)
    /\  Session(a,i,kai,Sa2,Ra2,Si2,Ri2)
    /\  Session(i,b,kbi,Si3,Ri3,Sb3,Rb3)

end role
```

```
goal

   secrecy_of Ca, Cb
   SPEKE_Init authenticates SPEKE_Resp on cb
   SPEKE_Resp authenticates SPEKE_Init on ca

end goal
```

```
Environment()
```

11 SRP: Secure remote passwords

Protocol Purpose

A client and a server authenticate each other based on a password such that the password remains secret, even if it is guessable.

Definition Reference

- http://srp.stanford.edu
- RFC 2945 [Wu00]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Sebastian Mödersheim, ETH Zürich

Alice&Bob style

We have a password *p* initially shared between the participants and a random number *s*, the *salt* (which at least the server knows initially). Original protocol, according to RFC:

```
identifiers & macros:
U = <username>
p = <raw password>
s = <salt from passwd file> (see notes section below)
N = <modulus>
x = SHA(s | SHA(U | ":" | p))
v = g^x mod N, the "password verifier"
a = <random number, chosen by U>
b = <random number, chosen by the server>
A = g^a mod N
B = v + g^b mod N
u = H(A,B)
S = (B - g^x) ^ (a + u * x) mod N
  = (A * v^u) ^ b mod N
K = SHA_Interleave(S)
M = H(H(N) XOR H(g),H(U),s,A,B,K)
-----------------------------------------------------------------
Client -> Host   : U,A
```

```
Host   -> Client : s,B
Client -> Host   : M
Host   -> Client : H(A,M,K)
-----------------------------------------------------------------
```

Simplified version:

```
Macros:
K = H(V.(G^Na)^Nb)
M = H(H(G),H(A).Salt.G^Na.{G^Nb}V.K)
-----------------------------------------------------------------
A -> B : A, G^Na
B -> A : Salt, {G^Nb}V
A -> B : M
B -> A : H(G^Na,M,K)
```

Problems considered: 3

- secrecy of `K`
- `SRP_Init` authenticates `SRP_Resp` on `k`
- `SRP_Resp` authenticates `SRP_Init` on `k`

Attacks Found

None

Model Limitations

Note that the protocol is slightly simplified as in the original version a full-scale algebraic theory is required.

Further Notes

A salt is a commonly-used mechanism to render dictionary (i.e. guessing) attacks more difficult. Standard UNIX password files, for instance, store a hash of each password prepended with a two-character salt. In this way, each possible password can map to 4096 different hash values, as there are 4096 possible values for the salt. This therefore greatly increases the computing power required for an intruder to mount a password guessing attack based on a precomputed dictionary of passwords and corresponding hash values.

HLPSL Specification

```
role SRP_Init (A,B : agent,
               Password : symmetric_key,
               H : function,
               G : text,
               Snd,Rcv:channel(dy)) played_by A def=

  local State : nat,
         Na:text (fresh),
         Salt : message,
         DHY, V, K, M : message

  init  State = 0
  knowledge(A) = {Password}

  transition

   1. State = 0 /\  Rcv(start) =|>
           Snd(A.exp(G,Na'))
        /\ State'=1

   2. State = 1 /\  Rcv(Salt'.{DHY'}_(exp(G,H(Salt'.H(A.Password))))) =|>
           State' = 2
        /\ V' = exp(G,H(Salt'.H(A.Password)))
        /\ K' = H( V'.exp(DHY',Na) )
        /\ M' = H(H(G).H(A).Salt'.exp(G,Na).{DHY'}_V'.K' )
        /\ Snd( M' )
        /\ witness(A,B,k,K')
        /\ secret(K',A) /\ secret(K',B)

   3. State = 2 /\ Rcv(H(exp(G,Na).M.K)) =|>
           State' = 3
        /\ request(A,B,k,K)

end role
```

```
role SRP_Resp (B,A : agent,
               Password : symmetric_key,
               Salt : message,
               H: function,
               G: text,
               Snd, Rcv:channel(dy)) played_by B def=

  local State: nat,
          Nb :text(fresh),
          M, K, DHX, V: message

  init State = 0
  knowledge(B) = {Password}

  transition

   1. State = 0 /\ Rcv(A.DHX') =|>
          Snd(Salt.{exp(G,Nb')}_(exp(G,H(Salt.H(A.Password)))))
       /\ V' = exp(G,H(Salt.H(A.Password)))
       /\ State' = 1
       /\ K' = H( V'.exp(DHX',Nb') )
       /\ M' = H(H(G).H(A).Salt.DHX'.{exp(G,Nb')}_V'.K')
       /\ witness(B,A,k,K')
       /\ secret(K',A) /\ secret(K',B)

   2. State = 1 /\ Rcv(M) =|>
          Snd(H(DHX.M.K))
       /\ State' = 3
       /\ request(B,A,k,K)
end role
```

```
role Session(A,B: agent,
             Password: symmetric_key,
             Salt: message,
             H: function,
             G: text,
             SA,RA,SB,RB: channel (dy)) def=
   composition
           SRP_Init(A,B,Password,H,G,SA,RA)  /\
           SRP_Resp(B,A,Password,Salt,H,G,SB,RB)

end role
```

```
role Environment() def=

  local S1,R1,S2,R2,
        S3,R3,S4,R4,
        S5,R5,S6,R6 : channel (dy)

  const k: protocol_id,
        a,b,i: agent,
        h:function,
        g:text

  knowledge(i) = {i, kai, kbi, s_kai, s_kbi}
  composition
           Session(a,b,kab,s_ab,h,g,S1,R1,S2,R2)
        /\ Session(a,i,kai,s_ai,h,g,S3,R3,S4,R4)
        /\ Session(b,i,kbi,s_bi,h,g,S5,R5,S6,R6)

end role
```

```
goal
    secrecy_of K
    SRP_Init authenticates SRP_Resp on k
    SRP_Resp authenticates SRP_Init on k
end goal
```

```
Environment()
```

12 IKEv2: Internet Key Exchange, version 2

12.1 authentication based on digital signatures

Protocol Purpose

IKE is designed to perform mutual authentication and key exchange prior to setting up an IPsec connection.

IKEv2 exists in several variants, the defining difference being the authentication method used.

This variant, which we call IKEv2-DS, uses digital signatures.

Definition Reference

[Kau03]

Model Authors

- Sebastian Mödersheim, ETH Zürich, December 2003
- Paul Hankes Drielsma, ETH Zürich, December 2003

Alice&Bob style

IKEv2-DS proceeds in two so-called exchanges. In the first, called IKE_SA_INIT, the users exchange nonces and perform a Diffie-Hellman exchange, establishing an initial security association called the IKE_SA. The second exchange, IKE_SA_AUTH, then authenticates the previous messages, exchanges the user identities, and establishes the first so-called "child security association" or CHILD_SA which will be used to secure the subsequent IPsec tunnel. A (respectively B) generates a nonce Na and a Diffie-Hellman half key KEa (respectively KEb). In addition, SAa1 contains A's cryptosuite offers and SAb1 B's preference for the establishment of the IKE_SA. Similarly SAa2 and SAb2 for the establishment of the CHILD_SA.

```
IKE_SA_INIT
1. A -> B: SAa1, KEa, Na
2. B -> A: SAb1, KEb, Nb
IKE_SA_AUTH
3. A -> B: {A, AUTHa, SAa2}K
   where K = H(Na.Nb.SAa1.g^KEa^KEb) and
     AUTHa = {SAa1.g^KEa.Na.Nb}inv(Ka)
4. B -> A: {B, AUTHb, SAb2}K
   where
```

```
    AUTHb = {SAb1.g^KEb.Na.Nb}inv(Kb)
```

Note that because we abstract away from the negotiation of cryptographic algorithms, we have SAa1 = SAb1 and SAa2 = SAb2.

Model Limitations

Issues abstracted from:

- The parties, Alice and Bob, should negotiate mutually acceptable cryptographic algorithms. This we abstract by modelling that Alice sends only a single offer for a crypto-suite, and Bob must accept this offer.
- There are goals of IKEv2 which we do not yet consider. For instance, identity hiding.
- IKEv2-DS includes provisions for the optional exchange of public-key certificates. This is not included in our model.
- We do not model the exchange of traffic selectors, which are specific to the IP network model and would be meaningless in our abstract communication model.

Problems considered: 3

- secrecy of `SK`
- `Alice` authenticates `Bob` on `sk1`
- `Bob` authenticates `Alice` on `sk2`

Attacks Found

With this variant of IKEv2, we find an attack analogous to the one that Meadows reports on in [Mea99]. In essence, the intruder is able to mount a man-in-the-middle attack between agents *a* and *b*. The trace below illustrates how the intruder convinces *b* that he was talking with *a*, when in fact *a* has not participated in the same session. Rather, the intruder has merely relayed messages from a different session with *a*, a session in which *a* expects to talk to the intruder.

```
i -> (a,6): start
(a,6) -> i: SA1(1),exp(g,DHX(1)),Ni(1)
i -> (b,3): SA1(1),exp(g,DHX(1)),Ni(1)
(b,3) -> i: SA1(1),exp(g,DHY(2)),Nr(2)
i -> (a,6): SA1(1),exp(g,DHY(2)),Nr(2)
(a,6) -> i: {a,{SA1(1),exp(g,DHX(1)),Ni(1),Nr(2)}inv(ka),
                SA2(3)}(f(Ni(1),Nr(2),SA1(1),exp(exp(g,DHY(2)),DHX(1))))
i -> (b,3): {a,{SA1(1),exp(g,DHX(1)),Ni(1),Nr(2)}inv(ka),
                SA2(3)}(f(Ni(1),Nr(2),SA1(1),exp(exp(g,DHX(1)),DHY(2))))
(b,3) -> i: {b,{SA1(1),exp(g,DHY(2)),Nr(2),Ni(1)}inv(kb),
                SA2(3)}(f(Ni(1),Nr(2),SA1(1),exp(exp(g,DHX(1)),DHY(2))))
```

This attack is of questionable validity, as the intruder has not actually learned the key that *b* believes to have established with *a*. Thus, the intruder cannot exploit the authentication flaw to further purposes. The attack can be precluded if we add key confirmation to the protocol. That is, if we extend the protocol to include messages in which the exchanged key is actually used, then this attack is no longer possible. In specification IKEv2-DSX we do just this.

HLPSL Specification

```
role Alice(A,B:agent,
           G: text,
           F: function,
           Ka,Kb: public_key,
           SND_B, RCV_B: channel (dy)) played_by A def=

  local Ni, SA1, SA2, DHX: text (fresh),
        Nr: text,
        KEr: message, %% more specific: exp(text,text)
        SK: message,
        State: nat

  init State = 0

  knowledge(A)={A,B,G,F,Ka,Kb,inv(Ka)}

  transition

  %% The IKE_SA_INIT exchange:
  %% We have abstracted away from the negotiation of cryptographic
  %% parameters.  Alice sends a nonce SAi1, which is meant to
  %% model Alice sending only a single crypto-suite offer.  Bob must
  %% then respond with the same nonce.
  1. State = 0 /\ RCV_B(start) =|>
        SND_B( SA1'.exp(G,DHX').Ni' ) /\ State' = 2

  %% Alice receives message 2 of IKE_SA_INIT, checks that Bob has
  %% indeed sent the same nonce in SAr1, and then sends the first
  %% message of IKE_AUTH.
  %% As authentication Data, she signs her first message and Bob's nonce.
  2. State = 2 /\ RCV_B(SA1.KEr'.Nr') =|>
        SK' = F(Ni.Nr'.SA1.exp(KEr',DHX)) /\
        SND_B( {A.{SA1.exp(G,DHX).Ni.Nr'}_(inv(Ka)).SA2'}_SK' ) /\
        State' = 4
        /\ witness(A,B,sk2,F(Ni.Nr'.SA1.exp(KEr',DHX)))

  3. State = 4 /\
        RCV_B({B.{SA1.KEr.Nr.Ni}_(inv(Kb)).SA2}_SK)
        =|>
        State' = 9
        /\ secret(SK,B)
        /\ request(A,B,sk1,SK)

end role

role Bob (B,A:agent,
          G: text,
          F: function,
          Kb, Ka: public_key,
          SND_A, RCV_A: channel (dy)) played_by B def=

  local Ni, SA1, SA2: text,
         Nr, DHY: text (fresh),
         SK, KEi: message,
         State: nat

  init State = 1

  transition

  1. State = 1 /\ RCV_A( SA1'.KEi'.Ni' ) =|>
        SND_A(SA1'.exp(G,DHY').Nr') /\
        SK' = F(Ni'.Nr'.SA1'.exp(KEi',DHY')) /\
        State' = 3
        /\ witness(B,A,sk1,F(Ni'.Nr'.SA1'.exp(KEi',DHY')))

  2. State = 3 /\ RCV_A( {A.{SA1.KEi.Ni.Nr}_(inv(Ka)).SA2'}_SK ) =|>
        SND_A( {B.{SA1.exp(G,DHY).Nr.Ni}_(inv(Kb)).SA2'}_SK ) /\
        State' = 9
        /\ secret(SK,A)
        /\ request(B,A,sk2,SK)

end role

role Session(A, B: agent,
             Ka, Kb: public_key,
             G: text, F: function,
             SA, RA, SB, RB: channel (dy)) def=

  composition
           Alice(A,B,G,F,Ka,Kb,SA,RA)
        /\ Bob(B,A,G,F,Kb,Ka,SB,RB)
end role

role Environment() def=

  local S1,R1,S2,R2,
        S3,R3,S4,R4,
        S5,R5,S6,R6: channel (dy)
  const sk1,sk2 : protocol_id
  knowledge(i) = {g,f,a,b,ka,kb,i,ki,inv(ki)}

  composition

        Session(a,b,ka,kb,g,f,S1,R1,S2,R2)
     /\ Session(a,i,ka,ki,g,f,S3,R3,S4,R4)
     /\ Session(i,b,ki,kb,g,f,S5,R5,S6,R6)

end role

goal
        secrecy_of SK
        Alice authenticates Bob on sk1
        Bob authenticates Alice on sk2
```

```
end goal

Environment()
```

12.2 authentication based on digital signatures, extended

Protocol Purpose

IKE is designed to perform mutual authentication and key exchange prior to setting up an IPsec connection. IKEv2 exists in several variants, the defining difference being the authentication method used.

This variant, which we call IKEv2-DSx, uses digital signatures and contains a slight extension in order to provide key confirmation, thus precluding the attack possible on the previous variant, IKEv2-DS.

Definition Reference

[Kau03]

Model Authors

- Sebastian Mödersheim, ETH Zürich, December 2003
- Paul Hankes Drielsma, ETH Zürich, December 2003

Alice&Bob style

IKEv2-DSx proceeds in three so-called exchanges. In the first, called IKE_SA_INIT, the users exchange nonces and perform a Diffie-Hellman exchange, establishing an initial security association called the IKE_SA. The second exchange, IKE_SA_AUTH, then authenticates the previous messages, exchanges the user identities, and establishes the first so-called "child security association" or CHILD_SA which will be used to secure the subsequent IPsec tunnel. A (respectively B) generates a nonce Na and a Diffie-Hellman half key KEa (respectively KEb). In addition, SAa1 contains A's cryptosuite offers and SAb1 B's preference for the establishment of the IKE_SA. Similarly SAa2 and SAb2 for the establishment of the CHILD_SA. We extend these standard two exchanges with a third which we call EXTENSION. It consists of two messages, each containing a nonce (MA and MB, respectively) and a distinguished constant (0 and 1, respectively) encrypted with the IKE_SA key K. This is sufficient to preclude the attack that is possible on IKEv2-DS, as it provides key confirmation.

```
IKE_SA_INIT
1. A -> B: SAa1, KEa, Na
2. B -> A: SAb1, KEb, Nb
IKE_SA_AUTH
3. A -> B: {A, AUTHa, SAa2}K
   where K = H(Na.Nb.SAa1.g^KEa^KEb) and
     AUTHa = {SAa1.g^KEa.Na.Nb}inv(Ka)
4. B -> A: {B, AUTHb, SAb2}K
   where
     AUTHb = {SAb1.g^KEb.Na.Nb}inv(Kb)
EXTENSION
5. A -> B: {MA, 0}K
6. B -> A: {MB, 1}K
```

Note that because we abstract away from the negotiation of cryptographic algorithms, we have SAa1 = SAb1 and SAa2 = SAb2.

Model Limitations

Issues abstracted from:

- The parties, Alice and Bob, should negotiate mutually acceptable cryptographic algorithms. This we abstract by modelling that Alice sends only a single offer for a crypto-suite, and Bob must accept this offer.
- There are goals of IKEv2 which we do not yet consider. For instance, identity hiding.
- IKEv2-DSx includes provisions for the optional exchange of public-key certificates. This is not included in our model.
- We do not model the exchange of traffic selectors, which are specific to the IP network model and would be meaningless in our abstract communication model.

Problems considered: 3

- secrecy of `SK`
- `Alice` authenticates `Bob` on `sk1`
- `Bob` authenticates `Alice` on `sk2`

Attacks Found

None

HLPSL Specification

```
role Alice(A,B:agent,
           G: text,
           F: function,
           Ka,Kb: public_key,
           SND_B, RCV_B: channel (dy)) played_by A def=

  local Ni, SA1, SA2, DHX: text (fresh),
         Nr: text,
         KEr: message, %% more specifically: exp(text,text)
         SK: message,
         State: nat,
         MA: text (fresh),
         MB: text,
         AUTH_B: message

  init State = 0

  knowledge(A)={A,B,G,F,Ka,Kb,inv(Ka)}

  transition

  %% The IKE_SA_INIT exchange:
  %% I have abstracted away from the negotiation of cryptographic
  %% parameters.  Alice sends a nonce SAi1, which is meant to
  %% model Alice sending only a single crypto-suite offer.  Bob must
  %% then respond with the same nonce.
  1. State = 0 /\ RCV_B(start) =|>
        SND_B( SA1'.exp(G,DHX').Ni' ) /\ State' = 2

  %% Alice receives message 2 of IKE_SA_INIT, checks that Bob has
  %% indeed sent the same nonce in SAr1, and then sends the first
  %% message of IKE_AUTH.
  %% As authentication Data, she signs her first message and Bob's nonce.
  2. State = 2 /\ RCV_B(SA1.KEr'.Nr') =|>
        SK' = F(Ni.Nr'.SA1.exp(KEr',DHX)) /\
        SND_B( {A.{SA1.exp(G,DHX).Ni.Nr'}_(inv(Ka)).SA2'}_SK' ) /\
        State' = 4

  3. State = 4 /\
        RCV_B({B.{SA1.KEr.Nr.Ni}_(inv(Kb)).SA2}_SK)
        =|>
        State' = 6 /\ SND_B({MA'.zero}_SK) /\
        AUTH_B' = {SA1.KEr.Nr.Ni}_(inv(Kb))
        /\ secret(SK,B)
        /\ witness(A,B,sk2,SK)

  4. State = 6 /\ RCV_B({MB'.one}_SK) =|> State' = 8
        /\ request(A,B,sk1,SK)

end role

role Bob (B,A:agent,
          G: text,
          F: function,
          Kb, Ka: public_key,
          SND_A, RCV_A: channel (dy)) played_by B def=

  local Ni, SA1, SA2: text,
        Nr, DHY: text (fresh),
        SK, KEi: message,
        State: nat,
        MA: text,
        MB: text (fresh),
        AUTH_A: message

  init State = 1

  transition

  1. State = 1 /\ RCV_A( SA1'.KEi'.Ni' ) =|>
        SND_A(SA1'.exp(G,DHY').Nr') /\
        SK' = F(Ni'.Nr'.SA1'.exp(KEi',DHY')) /\
        State' = 3

  2. State = 3 /\ RCV_A( {A.{SA1.KEi.Ni.Nr}_(inv(Ka)).SA2'}_SK ) =|>
        SND_A( {B.{SA1.exp(G,DHY).Nr.Ni}_(inv(Kb)).SA2'}_SK ) /\
        State' = 5 /\
        AUTH_A' = {SA1.KEi.Ni.Nr}_(inv(Ka))
        /\ witness(B,A,sk1,SK)
        /\ secret(SK,A)

  3. State = 5 /\ RCV_A({MA'.zero}_SK) =|>
        SND_A({MB'.one}_SK) /\ State' =7
        /\ request(B,A,sk2,SK)

end role

role Session(A, B: agent,
             Ka, Kb: public_key,
             G: text, F: function,
             SA, RA, SB, RB: channel (dy)) def=

  composition
           Alice(A,B,G,F,Ka,Kb,SA,RA)
        /\ Bob(B,A,G,F,Kb,Ka,SB,RB)
end role

role Environment() def=

  local S1,R1,S2,R2,
        S3,R3,S4,R4,
        S5,R5,S6,R6: channel (dy)

  const sk1, sk2: protocol_id,
        zero, one: protocol_id
  knowledge(i) = {g,f,a,b,ka,kb,i,ki,inv(ki),zero,one}

  composition

        Session(a,b,ka,kb,g,f,S1,R1,S2,R2)
     /\ Session(a,i,ka,ki,g,f,S3,R3,S4,R4)
     /\ Session(i,b,ki,kb,g,f,S5,R5,S6,R6)
```

```
end role

goal
        secrecy_of SK
        Alice authenticates Bob on sk1
        Bob authenticates Alice on sk2
end goal

Environment()
```

12.3 authentication based on MACs

Protocol Purpose

IKE is designed to perform mutual authentication and key exchange prior to setting up an IPsec connection. IKEv2 exists in several variants, the defining difference being the authentication method used.

This variant, which we call IKEv2-MAC, is based on exchanging the MAC of a pre-shared secret that both nodes possess.

Definition Reference

[Kau03]

Model Authors

- Sebastian Mödersheim, ETH Zürich, December 2003
- Paul Hankes Drielsma, ETH Zürich, December 2003

Alice&Bob style

IKEv2-MAC proceeds in two so-called exchanges. In the first, called IKE_SA_INIT, the users exchange nonces and perform a Diffie-Hellman exchange, establishing an initial security association called the IKE_SA. The second exchange, IKE_SA_AUTH, then authenticates the previous messages, exchanges the user identities, and establishes the first so-called "child security association" or CHILD_SA which will be used to secure the subsequent IPsec tunnel. A (respectively B) generates a nonce Na and a Diffie-Hellman half key KEa (respectively KEb). In addition, SAa1 contains A's cryptosuite offers and SAb1 B's preference for the establishment of the IKE_SA.

Similarly SAa2 and SAb2 for the establishment of the CHILD_SA. The two parties share a secret in advance, the so-called PSK or pre-shared key. The authenticator message is built by taking a hash of the PSK and the previously exchanged messages.

```
IKE_SA_INIT
1. A -> B: SAa1, KEa, Na
2. B -> A: SAb1, KEb, Nb
IKE_SA_AUTH
3. A -> B: {A, AUTHa, SAa2}K
   where K = H(Na.Nb.SAa1.g^KEa^KEb) and
     AUTHa = F(PSK.SAa1.KEa.Na.Nb)
4. B -> A: {B, AUTHb, SAb2}K
   where
     AUTHb = F(PSK.SAa1.KEr.Na.Nb)
```

Note that because we abstract away from the negotiation of cryptographic algorithms, we have SAa1 = SAb1 and SAa2 = SAb2.

Model Limitations

Issues abstracted from:

- The parties, Alice and Bob, should negotiate mutually acceptable cryptographic algorithms. This we abstract by modelling that Alice sends only a single offer for a crypto-suite, and Bob must accept this offer.
- There are goals of IKEv2 which we do not yet consider. For instance, identity hiding.
- We do not model the exchange of traffic selectors, which are specific to the IP network model and would be meaningless in our abstract communication model.

Problems considered: 3

- secrecy of `SK`
- `Alice` authenticates `Bob` on `sk1`
- `Bob` authenticates `Alice` on `sk2`

Attacks Found

None. Note that the use of MAC-based authentication precludes the man-in-the-middle attack that is possible on the first variant, IKEv2-DS.

HLPSL Specification

```
role Alice(A,B: agent,
               G: text,
               F: function,
               PSK: symmetric_key,
               SND_B, RCV_B: channel (dy)) played_by A def=

  local Ni, SA1, SA2, DHX: text (fresh),
         Nr: text,
         KEr: message, %% more spefic: exp(text,text)
         SK: message,
         State: nat,
         MA: text (fresh),
         MB: text,
         AUTH_B: message

  init State = 0

  knowledge(A)={A,G,F,PSK}

  transition

  %% The IKE_SA_INIT exchange:
  1. State = 0 /\ RCV_B(start) =|>
        SND_B( SA1'.exp(G,DHX').Ni' ) /\ State' = 2

  %% Alice receives message 2 of IKE_SA_INIT, checks that Bob has
  %% indeed sent the same nonce in SAr1, and then sends the first
  %% message of IKE_AUTH.
  %% As authentication Data, she signs her first message and Bob's nonce.
  2. State = 2 /\ RCV_B(SA1.KEr'.Nr') =|>
        SK' = F(Ni.Nr'.SA1.exp(KEr',DHX)) /\
        SND_B( {A.F(PSK.SA1.exp(G,DHX).Ni.Nr').SA2'}_SK' ) /\
        State' = 4
        /\ witness(A,B,sk2,F(Ni.Nr'.SA1.exp(KEr',DHX)))

  3. State = 4 /\
        RCV_B({B.F(PSK.SA1.KEr.Ni.Nr).SA2}_SK)
        =|>
        State' = 6
        /\ AUTH_B' = F(PSK.SA1.KEr.Ni.Nr)
        /\ secret(SK,B)
        /\ request(A,B,sk1,SK)

end role

role Bob(B,A:agent,
            G: text,
            F: function,
            PSK: symmetric_key,
            SND_A, RCV_A: channel (dy)) played_by B def=

  local Ni, SA1, SA2: text,
         Nr, DHY: text (fresh),
         SK, KEi: message,
         State: nat,
         MA: text,
         MB: text (fresh),
         AUTH_A: message

  init State = 1

  knowledge(B)={B,G,F,PSK}

  transition

  1. State = 1 /\ RCV_A( SA1'.KEi'.Ni' ) =|>
        SND_A(SA1'.exp(G,DHY').Nr') /\
        SK' = F(Ni'.Nr'.SA1'.exp(KEi',DHY')) /\
        State' = 3

  2. State = 3 /\ RCV_A( {A.F(PSK.SA1.KEi.Ni.Nr).SA2'}_SK ) =|>
        SND_A( {B.F(PSK.SA1.exp(G,DHY).Ni.Nr).SA2'}_SK ) /\
        State' = 5 /\
        AUTH_A' = F(PSK.SA1.KEi.Ni.Nr)
        /\ witness(B,A,sk1,SK)
        /\ secret(SK,A)
        /\ request(B,A,sk2,SK)

end role

role Session(A, B: agent,
             PSK: symmetric_key,
             G: text, F: function,
             SA, RA, SB, RB: channel (dy)) def=

  composition
           Alice(A,B,G,F,PSK,SA,RA)
        /\ Bob(B,A,G,F,PSK,SB,RB)
end role

role Environment() def=

  local S1,R1,S2,R2,
        S3,R3,S4,R4,
        S5,R5,S6,R6: channel (dy)

  const sk1, sk2 : protocol_id
  knowledge(i) = {g,f,a,b,i,kai,kbi}

  composition

        Session(a,b,kab,g,f,S1,R1,S2,R2)
     /\ Session(a,i,kai,g,f,S3,R3,S4,R4)
     /\ Session(i,b,kbi,g,f,S5,R5,S6,R6)

end role

goal
        secrecy_of SK
        Alice authenticates Bob on sk1
        Bob authenticates Alice on sk2
end goal
```

```
Environment()
```

12.4 authentication based on MACs, extended

Protocol Purpose

IKE is designed to perform mutual authentication and key exchange prior to setting up an IPsec connection. IKEv2 exists in several variants, the defining difference being the authentication method used.

This variant, which we call IKEv2-MACx, is based on exchanging the MAC of a pre-shared secret that both nodes possess. Analogous to the IKEv2-DSx variant, it also contains a slight extension in order to provide key confirmation.

Definition Reference

[Kau03]

Model Authors

- Sebastian Mödersheim, ETH Zürich, December 2003
- Paul Hankes Drielsma, ETH Zürich, December 2003

Alice&Bob style

IKEv2-MACx proceeds in three so-called exchanges. In the first, called IKE_SA_INIT, the users exchange nonces and perform a Diffie-Hellman exchange, establishing an initial security association called the IKE_SA. The second exchange, IKE_SA_AUTH, then authenticates the previous messages, exchanges the user identities, and establishes the first so-called "child security association" or CHILD_SA which will be used to secure the subsequent IPsec tunnel. A (respectively B) generates a nonce Na and a Diffie-Hellman half key KEa (respectively KEb). In addition, SAa1 contains A's cryptosuite offers and SAb1 B's preference for the establishment of the IKE_SA. Similarly SAa2 and SAb2 for the establishment of the CHILD_SA. The two parties share a secret in advance, the so-called PSK or pre-shared key. The authenticator message is built by taking a hash of the PSK and the previously exchanged messages. We extend these standard two exchanges with a third which we call EXTENSION. It consists of two messages, each containing a nonce (MA and MB, respectively) and a distinguished constant (0 and 1, respectively) encrypted with the IKE_SA key K.

```
IKE_SA_INIT
1. A -> B: SAa1, KEa, Na
2. B -> A: SAb1, KEb, Nb
IKE_SA_AUTH
3. A -> B: {A, AUTHa, SAa2}K
   where K = H(Na.Nb.SAa1.g^KEa^KEb) and
     AUTHa = F(PSK.SAa1.KEa.Na.Nb)
4. B -> A: {B, AUTHb, SAb2}K
   where
     AUTHb = F(PSK.SAa1.KEr.Na.Nb)
EXTENSION
5. A -> B: {MA, 0}K
6. B -> A: {MB, 1}K
```

Note that because we abstract away from the negotiation of cryptographic algorithms, we have SAa1 = SAb1 and SAa2 = SAb2.

Model Limitations

Issues abstracted from:

- The parties, Alice and Bob, should negotiate mutually acceptable cryptographic algorithms. This we abstract by modelling that Alice sends only a single offer for a crypto-suite, and Bob must accept this offer.
- There are goals of IKEv2 which we do not yet consider. For instance, identity hiding.
- We do not model the exchange of traffic selectors, which are specific to the IP network model and would be meaningless in our abstract communication model.

Problems considered: 3

- secrecy of `SK`
- `Alice` authenticates `Bob` on `sk`
- `Bob` authenticates `Alice` on `sk`

Attacks Found

None.

HLPSL Specification

```
role Alice(A,B: agent,
               G: text,
               F: function,
               PSK: symmetric_key,
               SND_B, RCV_B: channel (dy)) played_by A def=

  local Ni, SA1, SA2, DHX: text (fresh),
         Nr: text,
         KEr: message, %% more specifically: exp(text,text)
         SK: message,
         State: nat,
         MA: text (fresh),
         MB: text,
         AUTH_B: message

  init State = 0

  knowledge(A)={A,G,F,PSK}

  transition

  %% The IKE_SA_INIT exchange:
  1. State = 0 /\ RCV_B(start) =|>
        SND_B( SA1'.exp(G,DHX').Ni' ) /\ State' = 2

  %% Alice receives message 2 of IKE_SA_INIT, checks that Bob has
  %% indeed sent the same nonce in SAr1, and then sends the first
  %% message of IKE_AUTH.
  %% As authentication Data, she signs her first message and Bob's nonce.
  2. State = 2 /\ RCV_B(SA1.KEr'.Nr') =|>
        SK' = F(Ni.Nr'.SA1.exp(KEr',DHX)) /\
        SND_B( {A.F(PSK.SA1.exp(G,DHX).Ni.Nr').SA2'}_SK' ) /\
        State' = 4
  3. State = 4 /\
        RCV_B({B.F(PSK.SA1.KEr.Ni.Nr).SA2}_SK)
        =|>
        State' = 6 /\ SND_B({MA'.zero}_SK) /\
        AUTH_B' = F(PSK.SA1.KEr.Ni.Nr)
        /\ witness(A,B,sk,SK)

  4. State = 6 /\ RCV_B({MB'.one}_SK) =|> State' = 8
        /\ secret(SK,B)
        /\ request(A,B,sk,SK)

end role

role Bob(B,A:agent,
             G: text,
             F: function,
             PSK: symmetric_key,
             SND_A, RCV_A: channel (dy)) played_by B def=

  local Ni, SA1, SA2: text,
         Nr, DHY: text (fresh),
         SK, KEi: message,
         State: nat,
         MA: text,
         MB: text (fresh),
         AUTH_A: message

  init State = 1
  knowledge(B)={B,G,F,PSK}

  transition

  1. State = 1 /\ RCV_A( SA1'.KEi'.Ni' ) =|>
        SND_A(SA1'.exp(G,DHY').Nr') /\
        SK' = F(Ni'.Nr'.SA1'.exp(KEi',DHY')) /\
        State' = 3

  2. State = 3 /\ RCV_A( {A.F(PSK.SA1.KEi.Ni.Nr).SA2'}_SK ) =|>
        SND_A( {B.F(PSK.SA1.exp(G,DHY).Ni.Nr).SA2'}_SK ) /\
        State' = 5 /\
        AUTH_A' = F(PSK.SA1.KEi.Ni.Nr)
        /\ witness(B,A,sk,SK)

  3. State = 5 /\ RCV_A({MA'.zero}_SK) =|>
        SND_A({MB'.one}_SK) /\ State' =7
        /\ secret(SK,A)
        /\ request(B,A,sk,SK)

end role

role Session(A, B: agent,
             PSK: symmetric_key,
             G: text, F: function,
             SA, RA, SB, RB: channel (dy)) def=

  composition
           Alice(A,B,G,F,PSK,SA,RA)
        /\ Bob(B,A,G,F,PSK,SB,RB)
end role

role Environment() def=

  local S1,R1,S2,R2,
        S3,R3,S4,R4,
        S5,R5,S6,R6: channel (dy)

  const sk : protocol_id,
        zero, one : text
  knowledge(i) = {g,f,a,b,i,kai,kbi,zero,one}

  composition

        Session(a,b,kab,g,f,S1,R1,S2,R2)
     /\ Session(a,i,kai,g,f,S3,R3,S4,R4)
     /\ Session(i,b,kbi,g,f,S5,R5,S6,R6)

end role

goal
        secrecy_of SK
        Alice authenticates Bob on sk
        Bob authenticates Alice on sk
```

```
end goal

Environment()
```

12.5 subprotocol for the establishment of child SAs

Protocol Purpose

IKE is designed to perform mutual authentication and key exchange prior to setting up an IPsec connection.

This subprotocol of IKE, known as CREATE_CHILD_SA, is used to establish child security associations once an initial SA has been set up using the two initial exchanges of IKEv2.

Definition Reference

[Kau03]

Model Authors

- Sebastian Mödersheim, ETH Zürich, December 2003
- Paul Hankes Drielsma, ETH Zürich, December 2003

Alice&Bob style

IKEv2-CHILD consists of a single exchange called CREATE_CHILD_SA. Given a previously set up security association with key K, the users exchange two messages encrypted with K. These messages exchanges nonces and perform a Diffie-Hellman exchange, establishing a new security association called. A (respectively B) generates a nonce Na and a Diffie-Hellman half key KEa (respectively KEb). In addition, SAa contains A's cryptosuite offers and SAb B's preference for the establishment of the new SA. Authentication is provided based on the use of K, which is assumed to be known only to A and B.

```
CREATE_CHILD_SA
1. A -> B: {SAa, Na, KEa}K
2. B -> A: {SAb, Nr, KEb}K
```

Note that because we abstract away from the negotiation of cryptographic algorithms, we have SAa = SAb.

Model Limitations

Issues abstracted from:

- The parties, Alice and Bob, should negotiate mutually acceptable cryptographic algorithms. This we abstract by modelling that Alice sends only a single offer for a crypto-suite, and Bob must accept this offer.
- There are goals of IKEv2 which we do not yet consider. For instance, identity hiding.
- We do not model the exchange of traffic selectors, which are specific to the IP network model and would be meaningless in our abstract communication model.

Problems considered: 3

- secrecy of `CSK`
- `Alice` authenticates `Bob` on `nr`
- `Bob` authenticates `Alice` on `ni`

Attacks Found

None.

HLPSL Specification

```
role Alice(A,B:agent,
           G: text,
           F: function,
           SK: symmetric_key,
           SND_B, RCV_B: channel (dy)) played_by A def=

  local Ni, SA, DHX: text (fresh),
        Nr: text,
        KEr: message, % more specifically: exp(text,text)
        CSK: message, % CHILD_SA to be established.
        State: nat,
        MA: text (fresh),
        MB: text

  init State = 0

  knowledge(A)={A,B,G,F}

  transition

  1. State = 0 /\ RCV_B(start) =|>
     State' = 2 /\ SND_B( {SA'.Ni'.exp(G,DHX')}_SK )
                /\ witness(A,B,ni,Ni')

  2. State = 2  /\ RCV_B({SA.Nr'.KEr'}_SK) =|>
     State' = 4  /\ CSK' = F(Ni.Nr'.SA.exp(KEr',DHX))
                 /\ SND_B( {MA'.zero}_CSK' )

  4. State = 4 /\ RCV_B({MB'.one}_CSK) =|>
     State' = 6 /\ request(A,B,nr,Nr)
                /\ secret(CSK,B)

end role

role Bob (B,A:agent,
         G: text,
         F: function,
         SK: symmetric_key,
         SND_A, RCV_A: channel (dy)) played_by B def=

  local Ni, SA: text,
        Nr, DHY: text (fresh),
        KEi, CSK: message,
        State: nat,
        MA: text,
        MB: text (fresh)

  init State = 1

  knowledge(B)={A,B,G,F}

  transition
  1. State = 1 /\ RCV_A( {SA'.Ni'.KEi'}_SK ) =|>
     State' = 3 /\ CSK' = F(Ni'.Nr'.SA'.exp(KEi',DHY'))
                /\ SND_A( {SA'.Nr'.exp(G,DHY')}_SK )
                /\ witness(B,A,nr,Nr')

  2. State = 3 /\ RCV_A( {MA'.zero}_CSK ) =|>
     State' = 5 /\ SND_A( {MB'.one}_CSK )
                /\ request(B,A,ni,Ni)
                /\ secret(CSK,A)

end role

role Session(A, B: agent,
             SK: symmetric_key,
             G: text, F: function,
             SAC, RA, SB, RB: channel (dy)) def=

  composition
            Alice(A,B,G,F,SK,SAC,RA)
         /\ Bob(B,A,G,F,SK,SB,RB)
end role

role Environment() def=

  local S1,R1,S2,R2,
        S3,R3,S4,R4,
        S5,R5,S6,R6: channel (dy)

  const ni,nr : protocol_id,
        zero, one: protocol_id

  knowledge(i) = {g,f,a,b,i,kai,kbi,zero,one}

  composition

        Session(a,b,kab,g,f,S1,R1,S2,R2)
     /\ Session(a,i,kai,g,f,S3,R3,S4,R4)
     /\ Session(i,b,kbi,g,f,S5,R5,S6,R6)

end role
```

```
goal
        secrecy_of CSK
        Alice authenticates Bob on nr
        Bob authenticates Alice on ni
end goal

Environment()
```

12.6 using the EAP-Archie method

Protocol Purpose

The protocol should provide fresh key agreement, 3P-authorisation and DoS resilience.

Definition Reference

http://www.ietf.org/internet-drafts/draft-tschofenig-eap-ikev2-05.txt

Model Authors

Daniel Plasto for Siemens CT IC 3, 2004

Alice&Bob style

```
  1. A -> B: SAi1.KEi.Ni
  2. B -> A: SAr1.KEr.Nr

  3. A -> B: {IDi}_SK_e_i
  4. B -> A: {IDr.AUTH2.S.SessionID}_SK_e_r

  5. A -> B: {SessionID.P.{nonceP}_KEK.Binding.MAC1}_SK_e_i
  6. B -> A: {SessionID.{nonceS}_KEK.Binding.MAC2}_SK_e_r

  7. A -> B: {SessionID.MAC3.AUTH3}_SK_e_i
  8. B -> A: {Success.AUTH4}_SK_e_r

 - SAi1: ciphersuite (actually a single nonce)
 - SAr1: ciphersuite response (actually the nonce returned)
 - KEi: DH message 1. exp(G,DHX)
 - KEr: DH message 2. exp(G,DHY)
```

```
- Ni: nonce
- Nr: nonce
- SK: key derived from DH plus nonces.
     PRF(Ni.Nr.SAi1.exp(KEr,DHX)) for A
     PRF(Ni.Nr.SAr1.exp(KEi,DHY)) for B
- SK_e_i: key derived from SK for the initiator’s encryption PRFP1(SK)
- SK_e_r: key derived from SK for the initiator’s encryption PRFP2(SK)
- Idi: initiator’s identity
- Idr: responder’s identity
- AUTH2: {message2.Ni.PRF(SK_a_r,IDr)},signed with Kb
- AUTH3: PRF(EMK,message1.Nr.PRF(SK_a_i,IDi))
- AUTH4: PRF(EMK,message2.Ni.PRF(SK_a_r,IDr))
- SK_a_i: key derived from SK for the initiator’s
          authentication operations PRFP3(SK)
- SK_a_r: key derived from SK for the responder’s
          authentication operations PRFP4(SK)
- Ka: public key of A
- Kb: public key of B
- SessionID: Nonce
- KCK: Shared Key used for Authentication
- KEK: Shared Key used for Encryption
- KDK: Shared Key used for Key Derivation
- EMK: EAP Master Key: PRF(KDK.nonceS.nonceP)
- Binding: a nonce
- MAC1: MAC(KCK.S.SessionID.P.{nonceP}_KEK.Binding)
- MAC2: MAC(KCK.P.{nonceP}_KEK.SessionID.{nonceS}_KEK.Binding)
- MAC3: MAC(KCK.SessionID)
```

Model Limitations

- The optional certificates are excluded for now.
- The `CREATE_CHILD_SA` exchange is excluded, as are related fields.
- The ciphersuite is modelled as a nonce which must be returned by B. Similar to only having one option available.

Problems considered: 3

- secrecy of `SK,EMK`
- `Alice` authenticates `Bob` on `ker_nr_sid__nonces`

- `Bob` authenticates `Alice` on `kei_ni_binding_noncep`

Attacks Found

None

Further Notes

- In this version, the AUTH payloads are included in messages 7 and 8. Note that this is the first possible place to include them. Three other variations on this have been modelled, which change the position for the AUTH messages.
- The EAP Master Key is used as the Session Key, instead of applying another transform to get the Master Session Key.

HLPSL Specification

```
role Alice(
    A,B                                : agent,
    G                                  : text,
    Success                            : message,
    PRF,PRFP1,PRFP2,PRFP3,PRFP4,MAC : function,
    Ka,Kb                              : public_key,
    KCK,KEK,KDK                        : symmetric_key,
    SND, RCV                           : channel (dy))
played_by A def=

  local
    Ni, SAi1, DHX                      : text (fresh),
    Nr                                 : text,
    SK                                 : message,
    KEr                                : message,
    SID_                               : text,
    State                              : nat,
    Binding,NonceP                     : text (fresh),
    NonceS                             : text,
    EMK                                : message,
    KEr_Nr_SID__NonceS,
    KEi_Ni_Binding_NonceP           : message

  init
    State = 0

  knowledge(A) = {inv(Ka)}

  transition

 1. State  = 0  /\ RCV(start) =|>
    State' = 2  /\ SND(SAi1'.exp(G,DHX').Ni')

 2. State  = 2  /\ RCV(SAi1.KEr'.Nr') =|>
    State' = 4  /\ SK' = PRF(Ni.Nr'.SAi1.exp(KEr',DHX))
                /\ SND({A}_PRFP1(SK'))

 3. State  = 4  /\ RCV({ B.
                        {SAi1.KEr.Nr.Ni.PRF(PRFP4(SK),B)}_inv(Kb).
                        B.SID_'
                      }_PRFP2(SK)) =|>
    State' = 6  /\ SND({ SID_'.A.
                        {NonceP'}_KEK.
                        Binding'.
                        MAC(KCK.B.SID_'.A.{NonceP'}_KEK.Binding')
                      }_PRFP1(SK))
                /\ KEi_Ni_Binding_NonceP' = exp(G,DHX).Ni.Binding'.NonceP'
                /\ witness(A,B,kei_ni_binding_noncep,KEi_Ni_Binding_NonceP')

 4. State  = 6  /\ RCV({ SID_.
                        {NonceS'}_KEK.
                        Binding.
                        MAC(KCK.A.{NonceP}_KEK.SID_.{NonceS'}_KEK.Binding)
                      }_PRFP2(SK)) =|>
    State' = 8  /\ EMK' = PRF(KDK.NonceS'.NonceP)
                /\ SND({ SID_.
                        MAC(KCK.SID_).
                        PRF(EMK'.SAi1.exp(G,DHX).Ni.Nr.PRF(PRFP3(SK).A))
                      }_PRFP1(SK))

 5. State  = 8  /\ RCV({ Success.
                         PRF(EMK.SAi1.KEr.Nr.Ni.PRF(PRFP4(SK).B))
                       }_PRFP2(SK)) =|>
    State' = 10
                /\ secret(SK,B)
                /\ secret(EMK,B)
                /\ KEr_Nr_SID__NonceS' = KEr.Nr.SID_.NonceS
                /\ request(A,B,ker_nr_sid__nonces,KEr_Nr_SID__NonceS')

end role
```

```
role Bob (
    A,B                               : agent,
    G                                 : text,
    Success                           : message,
    PRF,PRFP1,PRFP2,PRFP3,PRFP4,MAC : function,
    Ka, Kb                            : public_key,
    KCK,KEK,KDK                       : symmetric_key,
    SND, RCV                          : channel (dy))
played_by B def=

  local
    Ni,SAr1                           : text,
    Nr,DHY,SID_,NonceS                : text (fresh),
    KEi                               : message,
    SK,EMK                            : message,
    NonceP,Binding                    : text,
    State                             : nat,
    KEr_Nr_SID__NonceS,
    KEi_Ni_Binding_NonceP             : message

  init
    State = 1

  knowledge(B) = {inv(Kb)}

  transition

 1. State  = 1  /\ RCV(SAr1'.KEi'.Ni') =|>
    State' = 3  /\ SND(SAr1'.exp(G,DHY').Nr')
```

```
                /\ SK' = PRF(Ni'.Nr'.SAr1'.exp(KEi',DHY'))

 2. State  = 3  /\ RCV({A}_PRFP1(SK)) =|>
    State' = 5  /\ SND({ B.
                      {SAr1.exp(G,DHY).Nr.Ni.PRF(PRFP4(SK),B)}_inv(Kb).
                      B.SID_'
                    }_PRFP2(SK))

 3. State  = 5  /\ RCV({ SID_.A.
                      {NonceP'}_KEK.
                      Binding'.
                      MAC(KCK.B.SID_.A.{NonceP'}_KEK.Binding')
                    }_PRFP1(SK)) =|>
    State' = 7  /\ SND({ SID_.
                      {NonceS'}_KEK.
                      Binding'.
                      MAC(KCK.A.{NonceP'}_KEK.SID_.{NonceS'}_KEK.Binding')
                    }_PRFP2(SK))
                /\ EMK' = PRF(KDK.NonceS'.NonceP')
                /\ KEr_Nr_SID__NonceS' = exp(G,DHY).Nr.SID_.NonceS'
                /\ witness(B,A,ker_nr_sid__nonces,KEr_Nr_SID__NonceS')

 4. State  = 7  /\ RCV({ SID_.
                      MAC(KCK.SID_).
                      PRF(EMK.SAr1.KEi.Ni.Nr.PRF(PRFP3(SK).A))
                    }_PRFP1(SK)) =|>
    State' = 9  /\ SND({ Success.
                      PRF(EMK.SAr1.exp(G,DHY).Nr.Ni.PRF(PRFP4(SK).B))
                    }_PRFP2(SK))
                /\ KEi_Ni_Binding_NonceP' = KEi.Ni.Binding.NonceP
                /\ request(B,A,kei_ni_binding_noncep,KEi_Ni_Binding_NonceP')

end role
```

```
role Session(
    A,B                             : agent,
    G                               : text,
    Success                         : message,
    PRF,PRFP1,PRFP2,PRFP3,PRFP4,MAC : function,
    Ka,Kb                            : public_key,
    KCK,KEK,KDK                      : symmetric_key)
def=

  local
     S1, S2 : channel (dy),
     R1, R2 : channel (dy)

  composition
     Alice(A,B,G,Success,
           PRF,PRFP1,PRFP2,PRFP3,PRFP4,MAC,Ka,Kb,KCK,KEK,KDK,S1,R1)
  /\ Bob(  A,B,G,Success,
           PRF,PRFP1,PRFP2,PRFP3,PRFP4,MAC,Ka,Kb,KCK,KEK,KDK,S2,R2)

end role
```

```
role Environment() def=

  const
    ker_nr_sid__nonces,
    kei_ni_binding_noncep : protocol_id,
    a,b                          : agent,
    ka,kb,ki1,ki2                : public_key,
    kck,kek,kdk                  : symmetric_key,
    kck_ib,kek_ib,kdk_ib         : symmetric_key,
    kck_ia,kek_ia,kdk_ia         : symmetric_key,
    g                            : text,
    success                      : message,
    prf,prfp1,prfp2,prfp3,prfp4 : function,
    mac                          : function

  knowledge(i) = {prf,prfp1,prfp2,prfp3,prfp4,
                  g,mac,a,b,i,
                  ka,kb,ki1,inv(ki1),ki2,inv(ki2),
                  success}

  composition

%   Session(a,b,g,success,prf,prfp1,prfp2,prfp3,prfp4,
%           mac,ka,kb,kck,kek,kdk)
% /\
    Session(a,b,g,success,prf,prfp1,prfp2,prfp3,prfp4,
            mac,ka,kb,kck,kek,kdk)
 /\ Session(i,b,g,success,prf,prfp1,prfp2,prfp3,prfp4,
            mac,ki1,kb,kck_ib,kek_ib,kdk_ib)
 /\ Session(a,i,g,success,prf,prfp1,prfp2,prfp3,prfp4,
            mac,ka,ki2,kck_ia,kek_ia,kdk_ia)

end role
```

```
goal

  secrecy_of SK,EMK
  Alice authenticates Bob on ker_nr_sid__nonces
  Bob authenticates Alice on kei_ni_binding_noncep

end goal
```

```
Environment()
```

13 RADIUS: Remote Authentication Dial In User Service

Protocol Purpose

A protocol for carrying authentication, authorisation, and configuration information between a Network Access Server which desires to authenticate its links and a shared Authentication Server.

Definition Reference

- RFC 2865: `http://www.faqs.org/rfcs/rfc2865.html`

Model Authors

- Vishal Sankhla, University of Southern California, August 2004

Alice&Bob style

```
1. Client -> Server : Access-Request
     where Access-Request = NAS_ID, NAS_PORT, {Secret_Key}MD5
2. Server -> Client : Access-Accept | Access-Reject | Access-Challenge
3. Client -> Server : Access-Chall-Request
     where Access-Chall-Request = {Message}Secret_Key
4. Server -> Client : Access-Accept
5. Client -> Server : Success
```

In (`2.`): If Client is authorised, the connection is accepted in which case a Success is returned. If Client is not authorised a failure message is sent out. If Challenge-Response is required to further authenticate the Client, the Server sends an access challenge to the Client.

Problems considered: 2

- secrecy of `Kcs`
- `Server` authenticates `Client` on `kcs`

Attacks Found

None

HLPSL Specification

```
role Client(C,S                         : agent,
            Kcs                         : symmetric_key,
            Md5                         : function,
            Success, Failure            : text,
            Access_accept,Access_reject : text,
            SND, RCV                    : channel(dy))
played_by C def=

      exists State             : nat,
             NAS_ID , NAS_Port : text,
             Chall_Message     : text

      init State = 0
      knowledge (C) = {C,S,Kcs,Md5,Success,Failure,SND,RCV}

      transition

      s1.   State = 0 /\ RCV(start) =|>
            State'= 1 /\ SND(NAS_ID'.NAS_Port'.Md5(Kcs))
                      /\ secret(Kcs,S)

      s2.   State = 1 /\ RCV(NAS_ID.Access_accept) =|>
            State'= 2 /\ SND(NAS_ID.Success)

      s3.   State = 1 /\ RCV(NAS_ID.Access_reject) =|>
            State'= 3 /\ SND(NAS_ID.Failure)

      s4.   State = 1 /\ RCV(NAS_ID.Chall_Message') =|>
            State'= 4 /\ SND(NAS_ID.{Chall_Message'}_Kcs)
                      /\ witness(C,S,kcs,Kcs)

      s5.   State = 4 /\ RCV(NAS_ID.Access_accept) =|>
            State'= 5 /\ SND(NAS_ID.Success)

end role
```

```
role Server(C,S                         : agent,
            Kcs                           : symmetric_key,
            Md5                           : function,
            Success, Failure              : text,
            Access_accept,Access_reject : text,
            SND, RCV                      : channel(dy))
played_by S def=

     exists State             : nat,
            NAS_ID , NAS_Port : text,
            Chall_Message     : text

     init State = 11
     knowledge (S) = {C,S,Kcs,Md5,Success,Failure,SND,RCV}

     transition

     s1.   State = 11 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
           State'= 12 /\ SND(NAS_ID'.Access_accept)
                       /\ secret(Kcs,C)

     s2.    State = 12 /\ RCV(NAS_ID.Success) =|>
            State'= 13

     s3.    State = 11 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
            State'= 14 /\ SND(NAS_ID'.Access_reject)

     s4.    State = 14 /\ RCV(NAS_ID.Failure) =|>
            State'= 15

     s5.    State = 11 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
            State'= 16 /\ SND(NAS_ID'.Chall_Message')

     s6.    State = 16 /\ RCV(NAS_ID.{Chall_Message}_Kcs) =|>
            State'= 17 /\ SND(NAS_ID.Access_accept)
                        /\ request(S,C,kcs,Kcs)

     s7.    State = 17 /\ RCV(NAS_ID.Success) =|>
            State'= 18

end role
```

```
role Session(C,S                         : agent,
             Kcs                         : symmetric_key,
             Md5                         : function,
             Success, Failure            : text,
             Access_accept,Access_reject : text)
def=

     local
       S1, S2 : channel (dy),
       R1, R2 : channel (dy)

     composition

         Client(C,S,Kcs,Md5,Success,Failure,Access_accept,Access_reject,S1,R1)
     /\  Server(C,S,Kcs,Md5,Success,Failure,Access_accept,Access_reject,S2,R2)

end role
```

```
role Environment() def=

     const c1,s1             : agent,
           md5               : function,
           succs, fails      : text,
           acc_acp, acc_rej  : text,
           kcsk , kisk, kcik : symmetric_key,
           c2,s2             : agent,
           kcs               : protocol_id

    knowledge(i) = {c1,s1,md5,kisk,kcik}

    composition
          Session(c1,s1,kcsk,md5,succs,fails,acc_acp,acc_rej)
       /\ Session(i, s1,kisk,md5,succs,fails,acc_acp,acc_rej)

end role
```

```
goal

  secrecy_of Kcs
  Server authenticates Client on kcs

end goal
```

```
Environment()
```

14 IEEE802.1x - EAPOL: EAP over LAN authentication

(IEEE 802.1X RADIUS: Remote Authentication Dial In User Service)

Protocol Purpose

The 802.1X (EAPOL) protocol provides effective authentication regardless of whether one implements 802.11 WEP keys or no encryption at all. If configured to implement dynamic key exchange, the 802.1X authentication server can return session keys to the access point along with the accept message. The access point uses the session keys to build, sign and encrypt an EAP key message that is sent to the client immediately after sending the success message. The client can then use contents of the key message to define applicable encryption keys.

Definition Reference

- RFC 3580: http://www.faqs.org/rfcs/rfc3580.html

Model Authors

- Vishal Sankhla, University of Southern California, August 2004

Alice&Bob style

```
Client -> Authenticator : EAPOL_Start
Auth -> Client : EAPOL_Request_Identity
Client -> Auth : EAPOL_Response (= NAS_ID, NAS_PORT, {Secret_Key}MD5)
Auth -> Server : Access-Request (= NAS_ID, NAS_PORT, {Secret_Key}MD5)
Server -> Auth : Access-Challenge
Auth -> Client : Access-Challenge
          where  Access-Challenge = Message
Client -> Auth : Access-Chall-Response
          where  Access-Chall-Response : {Message}Secret_Key
Auth -> Server : Access-Chall_Response
Server -> Auth : Access_Accept
Auth -> Client : EAPOL_Success
```

Problems considered: 2

- secrecy of `Kcs`
- `Server` authenticates `Client` on `kcs`

Attacks Found

None

Further Notes

Agents involved: Client, Authenticator, Radius Server

HLPSL Specification

```
role Client(C,A,S               : agent,
            Kcs                 : symmetric_key,
            Md5                 : function,
            EAPOL_Success,
            EAPOL_Start,
            EAPOL_Req_Identity : text,
            Success             : text,
            SND, RCV            : channel(dy))
played_by C def=

            exists State              : nat,
                   NAS_ID , NAS_Port : text,
                   Chall_Message      : text

            const kcs: protocol_id

            init State = 0
            knowledge (C) = {C,A,S,Kcs,Md5,SND,RCV}

            transition

            1. State = 0 /\ RCV(start) =|>
               State'= 1 /\ SND(EAPOL_Start)

            2. State = 1 /\ RCV( EAPOL_Req_Identity) =|>
               State'= 2 /\ SND(NAS_ID'.NAS_Port'.Md5(Kcs))
                         /\ secret(Kcs,S)
            3. State = 2 /\ RCV(NAS_ID.Chall_Message') =|>
               State'= 3 /\ SND(NAS_ID.{Chall_Message'}_Kcs)
                         /\ witness(C,S,kcs,Kcs)

            4. State = 3 /\ RCV(NAS_ID.EAPOL_Success) =|>
               State'= 4 /\ SND(NAS_ID.Success)
end role
```

```
role Auth(  C,A,S              : agent,
            Kcs                : symmetric_key,
            Md5                : function,
            EAPOL_Success,
            EAPOL_Start,
            EAPOL_Req_Identity : text,
            Success            : text,
            Access_accept      : text,
            SND, RCV           : channel(dy))
played_by A def=

            exists State              : nat,
                   NAS_ID , NAS_Port  : text,
                   Chall_message      : text,
                   Client_chall_reply : {text}_symmetric_key % ??? message

            const kcs: protocol_id

            init State = 11
            knowledge (A) = {C,S,Md5,SND,RCV}

            transition

            1. State = 11 /\ RCV(EAPOL_Start) =|>
               State'= 12 /\ SND(EAPOL_Req_Identity)

            2. State = 12 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
               State'= 13 /\ SND(NAS_ID'.NAS_Port'.Md5(Kcs))

            3. State = 13 /\ RCV(NAS_ID.Chall_message') =|>
               State'= 14 /\ SND(NAS_ID.Chall_message')

            4. State = 14 /\ RCV(NAS_ID.Client_chall_reply') =|>
```

```
               State'= 15 /\ SND(NAS_ID.Client_chall_reply')

            5. State = 15 /\ RCV(NAS_ID.Access_accept) =|>
               State'= 16 /\ SND(EAPOL_Success)

            6. State = 16 /\ RCV(NAS_ID.Success) =|>
               State'= 17

end role
```

```
role Server(C,A,S                         : agent,
            Kcs                           : symmetric_key,
            Md5                           : function,
            Success, Failure              : text,
            Access_accept,Access_reject : text,
            SND, RCV                      : channel(dy))
played_by S def=

            exists State             : nat,
                   NAS_ID , NAS_Port : text,
                   Chall_Message     : text

            const kcs: protocol_id

            init State = 21
            knowledge (S) = {C,A,S,Kcs,Md5,Success,Failure,SND,RCV}

    transition

     s5.    State = 21 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
            State'= 26 /\ SND(NAS_ID'.Chall_Message')
                       /\ secret(Kcs,C)
     s6.    State = 26 /\ RCV(NAS_ID.{Chall_Message}_Kcs) =|>
            State'= 27 /\ SND(NAS_ID.Access_accept)
                       /\ request(S,C,kcs,Kcs)

     s7.    State = 27 /\ RCV(NAS_ID.Success) =|>
            State'= 28

end role
```

```
role Session(C,A,S              : agent,
            Kcs                 : symmetric_key,
            Md5                 : function,
            Success,
            Failure             : text,
            Access_accept,
            Access_reject       : text,
            EAPOL_Success,
            EAPOL_Start,
            EAPOL_Req_Identity : text)
def=

     local
        S1, S2, S3 : channel (dy),
        R1, R2, R3 : channel (dy)

            composition

            Client(C,A,S,Kcs,Md5,
                   EAPOL_Success,EAPOL_Start,EAPOL_Req_Identity,
                   Success,S1,R1)
        /\    Auth(C,A,S,Kcs,Md5,
                   EAPOL_Success,EAPOL_Start,EAPOL_Req_Identity,
                   Success,Access_accept,S2,R2)
        /\  Server(C,A,S,Kcs,Md5,
                   Success,Failure,Access_accept,Access_reject,
                   S3,R3)

end role
```

```
role Environment() def=

     const c1,a1,s1    : agent,
     md5               : function,
     succs, fails      : text,
     acc_acp, acc_rej  : text,
     eap_succ,
     eap_start,
     eap_req_id        : text,
     kcsk , kisk, kcik : symmetric_key,
     c2,s2             : agent
     knowledge(i) = {c1,a1, s1,md5,kisk,kcik}

     composition
     Session(c1,a1,s1,kcsk,md5,succs,fails,acc_acp,acc_rej, eap_succ,
             eap_start,eap_req_id)
     %/\ Session(i,a1,s1,kisk,md5,succs,fails,acc_acp,acc_rej,
     %           eap_succ, eap_start,eap_req_id)

end role
```

```
goal

  secrecy_of Kcs
  Server authenticates Client on kcs

end goal
```

```
Environment()
```

15 HIP: Host Identity Protocol

Protocol Purpose

4-way hand-shake protocol that provides mobility enhanced with security, in particular authentication and authorisation.

Definition Reference

- http://homebase.htt-consult.com/~hip/
- http://www.ietf.org/internet-drafts/draft-ietf-hip-base-01.txt

Model Authors

- Murugaraj Shanmugam for Siemens CT IC 3, January 2005
- David von Oheimb, Siemens CT IC 3, January 2005

Alice&Bob style

S chooses HIP_Trans and PUZZLE

```
 0. C -> S: Hash(HI_S).Hash(HI_C)
 1. S -> C: {PUZZLE.HI_S.DH_S.HIP_Trans.ESP_Trans} Sig(S)
 2. C -> S: {Soln.LSI_C.SPI_C.HIP_Trans.ESP_Trans.DH_C.{HI_C}key}Sig(C)
 3. S -> C: {LSI_S.SPI_S.HMAC}Sig(S)
```

where

HI – Host Identity/Public key
DH – Diffie-Hellman
HIP_Trans – Algorithms for HIP key Generation
ESP_Trans – Algorithms for ESP key Generation
Soln – function to solve puzzle
key – generated using the HIP_Trans and DH
SPI – Security Parameter Index Value
LSI – Local Index Value
HMAC – one of the hash chains derived from DH

Model Limitations

- We assume that there is a Certificate Authority for verifying the certificates.
- Generation of hash chains to protect the further Base Exchange is not shown in the model.
- The key for hashing cannot be specified directly; we gave it as an extra field to `Hash`.
- Since HLPSL does not support arithmetic, we are unable to use the counter mechanism on puzzles, which prevents the replay attack. We incorporated the puzzle and the counter mechanism into a single fresh `PUZZLE` variable.

Problems considered: 2

- secrecy of `Hash(exp(EGX,Y))`
- `Initiator` authenticates `Responder` on `auth_jr`

Attacks Found

None

Further Notes

This protocol does not guarantee client authentication for the server, because there is no DNS lookup for the responder to verify the identity of the claimer.

Since the Certificate packet follows the I2 Packet, we just combined I2 and Certificate into a single packet. This does not pose a new attack, except for potential Denial of Service attacks, which we do not consider (yet).

HLPSL Specification

```
role Initiator (
        J,R     :agent,         % Initiator and Responder
        SND,RCV :channel(dy),   % Send, Receive Channel
        Hash    :function,      % Hash Function
        Soln    :function,      % Solution
        HI_I,HI_R:public_key,   % Public key of the Initiator and Responder
        G:nat)                   % Diffie Hellman's public G value
        played_by J def=

        local
        State:nat,
        X       :text(fresh),    % Initiator's Diffie Hellman Value
        SPI_I   :text(fresh),    % Initiator's Security Parameter Index Value
        LSI_I   :text,           % Initiator's Local Scope Index Value
        SPI_R   :text(fresh),    % Responder's Security Parameter Index Value
        LSI_R   :text,           % Responder's Local Scope Index Value
        PUZZLE  :text,           % Puzzle
        HIP_Trans :text,         % HIP Transform sent by the Responder
        ESP_Trans :text,         % ESP Transform of the Responder
        EGY     :message,        % Responder's Diffie Hellman Value
        R2      :message         % R2 Packet

        const
        hit_r   :text,           % HIT of the Responder (Hash of HI_R)
        cert    :text            % Certificate Packet

        init State = 0

        knowledge(J)={J,R,Hash,Soln,HI_R,G,HI_I,inv(HI_I)}
        % Knowledge of Initiator for this communication
                transition

 0.     State = 0 /\ RCV (start)=|>
                  State'= 1 /\ SND (Hash(HI_R).Hash(HI_I))

 1.     State = 1 /\ RCV((PUZZLE'.HI_R.EGY'.HIP_Trans'.ESP_Trans').
                    {Hash(PUZZLE'.HI_R.EGY'.HIP_Trans'.ESP_Trans')
                    }_inv(HI_R)) =|>
        State'= 3 /\ R2'=Hash(exp(EGY',X'))
                  /\ SND(Soln(PUZZLE').SPI_I'.LSI_I.choose(HIP_Trans').
                              choose(ESP_Trans').exp(G,X').{HI_I}_R2'.
                        {Hash(Soln(PUZZLE').SPI_I'.LSI_I.choose(HIP_Trans').
                              choose(ESP_Trans').exp(G,X').{HI_I}_R2')
                        }_inv(HI_I).
                         cert.{Hash(cert)}_inv(HI_I))
                  /\ secret(Hash(exp(EGY',X')),J) /\secret(Hash(exp(EGY',X')),R)
```

```
3.      State = 3 /\ RCV(Hash(SPI_R'.LSI_R'.Hash(R2))
                           .{SPI_R'.LSI_R'.Hash(R2)}_inv(HI_R)) =|>
        State'= 5 /\ request(J,R,auth_jr,R2)

end role
```

```
role Responder (
                J,R     :agent,         % Initiator and Responder
                SND,RCV :channel(dy),   % Send, Receive Channel
                Hash    :function,      % Hash Function
                Soln    :function,      % Solution
                HI_R    :public_key,    % Public key of the Responder
                G:nat)                  % Diffie Hellman's public G value
        played_by R def=

        local
        State:nat,
        Y       :text(fresh),   % Responder's Diffie Hellman parameter
        SPI_R   :text(fresh),   % Responder's Security Parameter Index Value
        LSI_R   :text,          % Responder's Local Scope Index Value
        SPI_I   :text(fresh),   % Initiator's Security Parameter Index Value
        LSI_I   :text,          % Initiator's Local Scope Index Value
        Puzzle  :text(fresh),   % Responder's Puzzle
        HI_I    :public_key,    % Public key of the Initiator
        Hj_I    :message,       % Hash (Public key)  of the Initiator
        Chosen_HIP_Trans :message,   % chosen HIP Transform
        Chosen_ESP_Trans :message,   % chosen HIP Transform
        I1      :text,          % I1 Packet
        CERT    :text,          % Certificate Packet
        EGX     :message,       % Initiator's Diffie-Hellman value
        R2      :message        % R2 Packet

        const
        hIP_Trans       :text,  % HIP Transform of the Responder
        eSP_Trans       :text   % HIP Transform of the Responder

        init State = 2

        knowledge(R)={J,R,Hash,Soln,HI_R,G,inv(HI_R)}
        transition

2.       State = 2 /\ RCV (Hash(HI_R).Hj_I') =|>
         State'= 4 /\ SND ((Puzzle'.HI_R.exp(G,Y').hIP_Trans.eSP_Trans).
                     {Hash((Puzzle'.HI_R.exp(G,Y').hIP_Trans.eSP_Trans))
                     }_inv(HI_R))

4.       State = 4 /\ RCV((Soln(Puzzle).SPI_I'.LSI_I'.Chosen_HIP_Trans'.
                            Chosen_ESP_Trans'.EGX'.{HI_I'}_Hash(exp(EGX',Y))).
                     {Hash(Soln(Puzzle).SPI_I'.LSI_I'.Chosen_HIP_Trans'.
                            Chosen_ESP_Trans'.EGX'.{HI_I'}_Hash(exp(EGX',Y)))
                     }_inv(HI_I').
                        CERT'.{Hash(CERT')}_inv(HI_I'))
                   /\ (Hash(HI_I')= Hj_I) =|>
         State'= 6 /\ R2'=Hash(exp(EGX',Y))
                   /\ SND(Hash(SPI_R'.LSI_R.Hash(R2')).
                              {SPI_R'.LSI_R.Hash(R2')}_inv(HI_R))
                   /\ secret(Hash(exp(EGX',Y)),J) /\ secret(Hash(exp(EGX',Y)),R)
                   /\ witness(R,J,auth_jr,R2')

end role
```

```
role Session (
        J,R     :agent,         % Initiator and Responder
        IR,RI   :channel(dy),   % Send, Receive Channel
        Hash    :function,      % Hash Function
        Soln    :function,      % Solution
        HI_I,HI_R  :public_key, % Public key of the Initiator,Responder
        G:nat)                  % Diffie Hellman's public G value
           def=

       composition
                Initiator(J,R,IR,RI,Hash,Soln,HI_I,HI_R,G)
             /\ Responder(J,R,RI,IR,Hash,Soln,HI_R,G)

end role
```

```
role Environment() def=

        const
          c,s     :agent,       % Initiator and Responder
          hash_   :function,    % Hash Function
          soln_   :function,    % Solution
          hi_i,hi_r:public_key, % Public key of the Initiator,Responder
          g_      :nat,         % Diffie Hellman's public G value
          auth_jr :protocol_id  % Protocol ID

        knowledge(i)={c,s,cs,sc,hash_,hi_i,hi_r,g_}
        % intruder won't solve puzzles

        composition

        Session(c,s,cs,sc,hash_,soln_,hi_i,hi_r,g_)
% /\    Session(i,s,cs,sc,hash_,soln_,hi_i,hi_r,g_)
%       Adding this session yields a spurious authentication failure because
%       the client of the first session talks to the server of the second,
%       but in exactly the same way as he would do within the first session.
% /\    Session(c,i,cs,sc,hash_,soln_,hi_i,hi_r,g_)
%       adding this session yields a spurious authentication failure for the
%       first session because the intruder knows the private key of the server
%       since he plays the server's role.

end role
```

```
goal

        secrecy_of Hash(exp(EGX,Y))
        Initiator authenticates Responder on auth_jr

end goal
```

```
Environment()
```

16 PBK: Purpose Built Keys Framework

16.1 original version

Protocol Purpose

Sender invariance (authentication assuming first message is not tampered with)

Definition Reference

http://www.ietf.org/internet-drafts/draft-bradner-pbk-frame-06.txt

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004
- Sebastian Mödersheim, ETH Zürich

Alice&Bob style

```
A -> B: A, PK_A, hash(PK_A)
A -> B: {Msg}_inv(PK_A), hash(PK_A)
B -> A: Nonce
A -> B: {Nonce}_inv(PK_A)
```

Problems considered: 1

- `Alice` is authenticated on `msg`

Attacks Found

The initiator shall sign a random challenge received from the responder. This can easily be exploited to make agents sign whatever the intruder wishes:

```
i      -> (a,3) : start
(a,3)  -> i     : {Msg(1)}inv(pk_a),f(pk_a)
i      -> (a,12): start
(a,12) -> i     : {Msg(2)}inv(pk_a),f(pk_a)
i      -> (a,3) : x71
(a,3)  -> i     : {x71}inv(pk_a)
```

```
   i      -> (b,3) : {x71}inv(pk_a),f(pk_a)
   (b,3)  -> i     : Nonce(4)
   i      -> (a,12): Nonce(4)
   (a,12) -> i     : {Nonce(4)}inv(pk_a)
   i      -> (b,3) : {Nonce(4)}inv(pk_a)
```

Further Notes

The protocol is so far only roughly described in natural language, and this file represents a verbatim translation to HLPSL as an "early prototype" and the AVISPA tool can identify a potential source for attacks which protocol designers should be aware of when implementing a protocol (see paragraph "Attacks"). A fixed version (with tagging the challenge before signing it) is also provided in this library.

The assumption is that the intruder cannot modify (or intercept) the first message is modelled by a compression-technique. Also, the authentication must be specified in a slightly different way, as A does not say for whom it signs the message (and anybody can act as responder).

HLPSL Specification

```
role Alice (
    A,B         : agent,
    SND,RCV     : channel(dy),
    Hash        : function,
    PK_A        : public_key
) played_by A def=

  local
    State       : nat,
    Msg         : text (fresh),
    Nonce       : text

  init State = 0

  transition

 1. State  = 0 /\ RCV(start) =|>
```

```
    State' = 2 /\ SND({Msg'}_inv(PK_A).Hash(PK_A))
               /\ witness(A,A,msg,Msg')

 3. State  = 2 /\ RCV(Nonce') =|>
    State' = 4 /\ SND({Nonce'}_inv(PK_A))

end role
```

```
role Bob (
    B,A         : agent,
    SND,RCV     : channel(dy),
    Hash        : function,
    PK_A        : public_key
) played_by B def=

  local
    State       : nat,
    Nonce       : text (fresh),
    Msg         : text

  init State = 1

  transition

 1. State  = 1 /\ RCV({Msg'}_inv(PK_A).Hash(PK_A)) =|>
    State' = 5 /\ SND(Nonce')

 3. State  = 5 /\ RCV({Nonce}_inv(PK_A)) =|>
    State' = 7 /\ request(A,A,msg,Msg)

end role
```

```
role Session(
    A,B         : agent,
    SNDA,RCVA,
    SNDB,RCVB   : channel (dy),
    Hash        : function,
    PK_A       : public_key
) def=

  composition
    Alice(A,B,SNDA,RCVA,Hash,PK_A)
 /\ Bob(B,A,SNDB,RCVB,Hash,PK_A)

end role
```

```
role Environment() def=

  local
    S1,R1,S2,R2,
    S3,R3,S4,R4,
    S5,R5,S6,R6,
    S7,R7,S8,R8: channel (dy)

  const
    a,b          : agent,
    f            : function,
    msg          : protocol_id,
    pk_a, pk_b,pk_i: public_key

  knowledge(i) = {a,b,f,pk_a,pk_b,pk_i,inv(pk_i)}

  composition
    Session(a,b,S1,R1,S5,R5,f,pk_a)
 /\ Session(b,a,S2,R2,S6,R6,f,pk_b)
 /\ Session(i,b,S3,R3,S7,R7,f,pk_i)
 /\ Session(a,i,S4,R4,S8,R8,f,pk_a)

end role
```

```
goal

  Alice authenticates Alice on msg
```

```
end goal
```

```
Environment()
```

16.2 fixed version

Protocol Purpose

Sender invariance (authentication assuming that the first message is not tampered with)

Definition Reference

http://www.ietf.org/internet-drafts/draft-bradner-pbk-frame-06.txt

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004
- Sebastian Mödersheim, ETH Zürich

Alice&Bob style

```
A -> B: A, PK_A, hash(PK_A)
A -> B: {***tag1***,Msg}inv(PK_A), hash(PK_A)
B -> A: Nonce
A -> B: {***tag2***,Nonce}inv(PK_A)
```

Problems considered: 1

- `Alice` is authenticated on `msg`

Attacks Found

Initially, we demanded (strong) authentication, but this does of course not hold as there is nothing that guarantees freshness, until the agent generates a new public key, as in the following replay attack, which is possible after observing a session between honest agents a and b using $Msg(1)$ as the exchanged message.

```
i -> (a,3): start
(a,3) -> i: b,{tag1,Msg(1)}inv(pk_a),f(pk_a)
i -> (b,3): b,{tag1,Msg(1)}inv(pk_a),f(pk_a)
(b,3) -> i: Nonce(3)
i -> (a,3): Nonce(3)
(a,3) -> i: {tag2,Nonce(3)}inv(pk_a)
i -> (b,3): {tag2,Nonce(3)}inv(pk_a)

i -> (a,6): start
(a,6) -> i: b,{tag1,Msg(4)}inv(pk_a),f(pk_a)
i -> (b,6): b,{tag1,Msg(1)}inv(pk_a),f(pk_a)
(b,6) -> i: Nonce(6)
i -> (a,6): Nonce(6)
(a,6) -> i: {tag2,Nonce(6)}inv(pk_a)
i -> (b,6): {tag2,Nonce(6)}inv(pk_a)
```

Further Notes

Prevents the attack of the initial version by tagging the nonce before signing it. This version was only provide to demonstrate that the protocol cannot ensure strong authentication.

HLPSL Specification

```
role Alice (
    A,B         : agent,
    SND,RCV     : channel(dy),
    Hash        : function,
    PK_A        : public_key,
    Tag1,Tag2   : text
) played_by A def=

  local
    State       : nat,
    Msg         : text (fresh),
    Nonce       : text
  init  State = 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State' = 2 /\ SND(B.{Tag1.Msg'}_inv(PK_A).Hash(PK_A))
               /\ witness(A,A,msg,Msg')

 3. State  = 2 /\ RCV(Nonce') =|>
    State' = 4 /\ SND({Tag2.Nonce'}_inv(PK_A))

end role
```

```
role Bob (
    B,A         : agent,
    SND,RCV     : channel(dy),
    Hash        : function,
    PK_A        : public_key,
    Tag1,Tag2   : text
) played_by B def=

  local
    State       : nat,
    Nonce       : text (fresh),
    Msg         : text

  init State = 1

  transition

 1. State  = 1 /\ RCV(B.{Tag1.Msg'}_inv(PK_A).Hash(PK_A)) =|>
    State' = 5 /\ SND(Nonce')

 3. State  = 5 /\ RCV({Tag2.Nonce}_inv(PK_A)) =|>
    State' = 7 /\ request(A,A,msg,Msg)

end role
```

```
role Session(
    A,B        : agent,
    SNDA,RCVA,
    SNDB,RCVB  : channel (dy),
    Hash       : function,
    PK_A       : public_key,
    Tag1,Tag2        : text
) def=

  composition
    Alice(A,B,SNDA,RCVA,Hash,PK_A,Tag1,Tag2)
 /\ Bob(B,A,SNDB,RCVB,Hash,PK_A,Tag1,Tag2)

end role
```

```
role Environment() def=

  local
    S1,R1,S2,R2,
    S3,R3,S4,R4,
    S5,R5,S6,R6,
    S7,R7,S8,R8: channel (dy)

  const
    a,b          : agent,
    f            : function,
    msg          : protocol_id,
    pk_a, pk_b,pk_i: public_key,
    tag1,tag2     : text

  knowledge(i) = {a,b,f,pk_a,pk_b,pk_i,inv(pk_i)}

  composition
    Session(a,b,S1,R1,S2,R2,f,pk_a,tag1,tag2)
 /\ Session(a,b,S3,R3,S4,R4,f,pk_a,tag1,tag2)

end role
```

```
goal

  Alice authenticates Alice on msg

end goal
```

```
Environment()
```

16.3 fixed version with weak authentication

Protocol Purpose

Sender invariance (authentication assuming that the first message is not tampered with)

Definition Reference

http://www.ietf.org/internet-drafts/draft-bradner-pbk-frame-06.txt

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004
- Sebastian Mödersheim, ETH Zürich

Alice&Bob style

```
  A -> B: A, PK_A, hash(PK_A)
  A -> B: {***tag1***,Msg}inv(PK_A), hash(PK_A)
  B -> A: Nonce
  A -> B: {***tag2***,Nonce}inv(PK_A)
```

Problems considered: 1

- `Alice` is weakly authenticated on `msg`

Attacks Found

None

Further Notes

Same as before, but specifying only weak authentication.

HLPSL Specification

```
role Alice (
    A,B         : agent,
    SND,RCV     : channel(dy),
    Hash        : function,
    PK_A        : public_key,
    Tag1,Tag2   : text
) played_by A def=

  local
    State       : nat,
    Msg         : text (fresh),
    Nonce       : text

  init State = 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State' = 2 /\ SND(B.{Tag1.Msg'}_inv(PK_A).Hash(PK_A))
               /\ witness(A,A,msg,Msg')

 3. State  = 2 /\ RCV(Nonce') =|>
    State' = 4 /\ SND({Tag2.Nonce'}_inv(PK_A))

end role
```

```
role Bob (
    B,A         : agent,
    SND,RCV     : channel(dy),
    Hash        : function,
    PK_A        : public_key,
    Tag1,Tag2   : text
) played_by B def=

  local
    State       : nat,
    Nonce       : text (fresh),
    Msg         : text

  init State = 1

  transition

 1. State  = 1 /\ RCV(B.{Tag1.Msg'}_inv(PK_A).Hash(PK_A)) =|>
    State' = 5 /\ SND(Nonce')

 3. State  = 5 /\ RCV({Tag2.Nonce}_inv(PK_A)) =|>
    State' = 7 /\ wrequest(A,A,msg,Msg)

end role
```

```
role Session(
    A,B         : agent,
    SND,RCV,
    SNDA,RCVA   : channel (dy),
    Hash        : function,
    PK_A        : public_key,
    Tag1,Tag2         : text
) def=

  composition
    Alice(A,B,SND,RCV,Hash,PK_A,Tag1,Tag2)
 /\ Bob(B,A,SND,RCV,Hash,PK_A,Tag1,Tag2)

end role
```

```
role Environment() def=

  local
    S1,R1,S2,R2,
    S3,R3,S4,R4,
    S5,R5,S6,R6,
    S7,R7,S8,R8: channel(dy)

  const
    a,b          : agent,
    f            : function,
    msg          : protocol_id,
    pk_a, pk_b,pk_i: public_key,
    tag1,tag2     : text

  knowledge(i) = {a,b,f,pk_a,pk_b,pk_i,inv(pk_i)}

  composition
    Session(a,b,S1,R1,S5,R5,f,pk_a,tag1,tag2)
 /\ Session(b,a,S2,R2,S6,R6,f,pk_b,tag1,tag2)
 /\ Session(i,b,S3,R3,S7,R7,f,pk_i,tag1,tag2)
 /\ Session(a,i,S4,R4,S8,R8,f,pk_a,tag1,tag2)

end role
```

```
goal

  Alice weakly authenticates Alice on msg

end goal
```

```
Environment()
```

17 Kerberos Network Authentication Service (V5)

17.1 basic (core)

Protocol Purpose

Authentication, Authorisation, Key Exchange

Kerberos is a distributed authentication service that allows a process (a client) running on behalf of a principal (a user) to prove its identity to a verifier (an application server, or just server) without sending data across the network that might allow an attacker or the verifier to subsequently impersonate the principal. Kerberos optionally provides integrity and confidentiality for data sent between the client and server.

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-07.txt

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Sebastian Mödersheim, Computer Security Group, ETH Zürich, January 2004
- AVISPA team (since then)

Alice&Bob style

```
C: Client
A: Authentication Server
G: Ticket Granting Server
S: Server (that the client wants to talk to)

K_AB: key shared or intended to be shared between A and B
      Initially shared: K_CA, K_AG, K_GS
      Established during protocol: K_CG, K_CS

All things marked * are timestamp-related and will be simply replaced
with fresh text.

Macros:
```

```
Ticket_1 := { C,G, K_CG, Tstart*, Texpire* }K_AG
Ticket_2 := { C,S, K_CS, Tstart2*, Texpire2* }K_GS

1. C -> A : C,G,Lifetime_1*,N_1
2. A -> C : C, Ticket_1, { G, K_CG, Tstart*, Texpire*, N_1 }K_CA

3. C -> G : S,Lifetime_2*,N_2,Ticket_1, { C,T* }K_CG
4. G -> C : C, Ticket_2, { S, K_CS, Tstart2*, Texpire2*, N_2 }K_CG

5. C -> S : Ticket_2, { C, T2* }K_CS
6. S -> C : { T2* }K_CS
```

Model Limitations

Ticket Caching is not performed, so only weak authentication is provided. It is rumoured that implementations do not perform ticket caching.

Problems considered: 8

- secrecy of K_CG, K_CS
- Kerberos_C weakly authenticates Kerberos_A on k_cg
- Kerberos_G weakly authenticates Kerberos_A on k_cg
- Kerberos_C weakly authenticates Kerberos_G on k_cs
- Kerberos_S weakly authenticates Kerberos_G on k_cs
- Kerberos_C weakly authenticates Kerberos_S on t2a
- Kerberos_S weakly authenticates Kerberos_C on t2b
- Kerberos_G weakly authenticates Kerberos_C on t1

Attacks Found

None

Further Notes

Agents involved: Client, Authentication Server (AS), Ticket Granting server (TGS), Server where the client needs to authenticate (Server)

HLPSL Specification

```
% Authentication Server
role Kerberos_A (A, C, G : agent,
                 Snd, Rcv   : channel (dy),
                 K_CA, K_AG : symmetric_key) played_by A def=

  local St              : nat,
        K_CG            : symmetric_key (fresh),
        N1, Lifetime_1  : text,
        Tstart, Texpire : text (fresh)

  const  k_cg : protocol_id

  init St = 0

  knowledge(A) = {A,C,G,K_CA,K_AG}

  transition

   1. St = 0 /\ Rcv(C.G.Lifetime_1'.N1') =|>
      St'= 1 /\ Snd(C.{C.G.K_CG'.Tstart'.Texpire'}_K_AG.
                      {G.K_CG'.Tstart'.Texpire'.N1'}_K_CA)
             /\ witness(A,C,k_cg,K_CG')
             /\ secret(K_CG',A) /\ secret(K_CG',C) /\ secret(K_CG',G)

end role
```

```
% Ticket Granting Server
role Kerberos_G (G, A, S, C  : agent,
                 Snd, Rcv    : channel (dy),
                 K_AG, K_GS  : symmetric_key) played_by G def=

  local St                                 : nat,
        K_CG                               : symmetric_key,
        K_CS                               : symmetric_key (fresh),
        Lifetime_2, Tstart, Texpire, T, N2 : text,
        Tstart2, Texpire2                  : text (fresh)

  const  k_cs : protocol_id

  init St = 0

  knowledge(G) = {A,C,G,S,K_AG,K_GS}

  transition

   1. St = 0 /\
       Rcv(S.Lifetime_2'.N2'.{C.G.K_CG'.Tstart'.Texpire'}_K_AG.{C.T'}_K_CG') =|>
      St'= 1 /\ Snd(C.
                   {C.S.K_CS'.Tstart2'.Texpire2'}_K_GS.
                     {S.K_CS'.Tstart2'.Texpire2'.N2'}_K_CG')
             /\ wrequest(G,C,t1,T')
             /\ wrequest(G,A,k_cg,K_CG')
             /\ witness(G,S,k_cs,K_CS')
             /\ secret(K_CG',A) /\ secret(K_CG',C) /\ secret(K_CG',G)
             /\ secret(K_CS',G) /\ secret(K_CS',C) /\ secret(K_CS',S)

end role
```

```
% Server
role Kerberos_S (S, G, C  : agent,
                 Snd, Rcv : channel (dy),
                 K_GS     : symmetric_key) played_by S def=

  local St                    : nat,
        Tstart2, Texpire2, T2 : text,
        K_CS                  : symmetric_key

  const  t2a, t2b : protocol_id
  init St = 0

  knowledge(S) = {S,G,C,K_GS}

  transition

   1. St = 0 /\ Rcv({C.S.K_CS'.Tstart2'.Texpire2'}_K_GS.{C.T2'}_K_CS') =|>
      St'= 1 /\  Snd({T2'}_K_CS')
             /\ witness(S,C,t2a,T2')
             /\ wrequest(S,G,k_cs,K_CS')
             /\ wrequest(S,C,t2b,T2')
             /\ secret(K_CS',G) /\ secret(K_CS',C) /\ secret(K_CS',S)

end role
```

```
% Client
role Kerberos_C (C, A, G, S : agent,
                 Snd, Rcv   : channel (dy),
                 K_CA       : symmetric_key) played_by C def=

  local St                                 : nat,
      K_CG, K_CS                           : symmetric_key,
      T, T2 : text (fresh),
      Tstart, Texpire, Tstart2, Texpire2    : text,
      Ticket_1, Ticket_2 : {agent.agent.symmetric_key.text.text}_symmetric_key,
      N1, N2  : text (fresh)

  const  k_cg, k_cs, t2a, t2b : protocol_id,
         cLifetime_1, cLifetime_2: text

  init St= 0

  knowledge(C) = {C,A,G,S,K_CA}

  transition

   1. St = 0 /\ Rcv(start) =|>
      St'= 1 /\ Snd(C.G.cLifetime_1.N1')
   2. St = 1 /\ Rcv(C.Ticket_1'.{G.K_CG'.Tstart'.Texpire'.N1}_K_CA) =|>
      St'= 2 /\ Snd(S.cLifetime_2.N2'.Ticket_1'.{C.T'}_K_CG')
             /\ witness(C,G,t1,T')
             /\ wrequest(C,A,k_cg,K_CG')
             /\ secret(K_CG',A) /\ secret(K_CG',C) /\ secret(K_CG',G)

   3. St = 2 /\ Rcv(C.Ticket_2'.{S.K_CS'.Tstart2'.Texpire2'.N2}_K_CG)  =|>
      St'= 3 /\ Snd(Ticket_2'.{C.T2'}_K_CS')
             /\ witness(C,S,t2b,T2')
             /\ wrequest(C,G,k_cs,K_CS')
             /\ secret(K_CS',G) /\ secret(K_CS',C) /\ secret(K_CS',S)

   4. St = 3 /\ Rcv({T2}_K_CS) =|>
      St'= 4 /\ wrequest(C,S,t2a,T2)

end role
```

```
role Session( C, A, G, S                             : agent,
              K_CA, K_AG, K_GS                       : symmetric_key) def=

   local S_C, R_C, S_A, R_A, S_G, R_G, S_S, R_S : channel (dy)

   composition

        Kerberos_C(C,A,G,S,S_C,R_C,K_CA)
     /\ Kerberos_A(A,C,G,S_A,R_A,K_CA,K_AG)
     /\ Kerberos_G(G,A,S,C,S_G,R_G,K_AG,K_GS)
     /\ Kerberos_S(S,G,C,S_S,R_S,K_GS)

end role
```

```
role Environment() def=

  const  c, a, g, s, i          : agent,
         kca, kag, kgs, kia     : symmetric_key

  knowledge(i) = {c,a,g,s,kia
```

```
                 }

  composition
        Session(c,a,g,s,kca,kag,kgs)
 /\     Session(i,a,g,s,kia,kag,kgs)

end role
```

```
goal

  secrecy_of K_CG, K_CS

  Kerberos_C weakly authenticates Kerberos_A on k_cg
  Kerberos_G weakly authenticates Kerberos_A on k_cg

  Kerberos_C weakly authenticates Kerberos_G on k_cs
  Kerberos_S weakly authenticates Kerberos_G on k_cs

  Kerberos_C weakly authenticates Kerberos_S on t2a
  Kerberos_S weakly authenticates Kerberos_C on t2b

  Kerberos_G weakly authenticates Kerberos_C on t1

end goal
```

```
Environment()
```

17.2 with ticket caching

Protocol Purpose

Strong mutual authentication

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-07.txt

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004

Alice&Bob style

```
C -> A: U,G,N1
A -> C: U,Tcg,{G,Kcg,T1start,T1expire,N1}_Kca

where Tcg := {U,C,G,Kcg,T1start,T1expire}_Kag
      A := Authentication Server

C -> G: S,N2,Tcg,Acg
G -> C: U,Tcs,{S,Kcs,T2start,T2expire,N2}_Kcg

where Acg := {C,T1}_Kcg  (T1 is a timestamp)
      Tcs := {U,C,S,Kcs,T2start,T2expire}_Kgs

C -> S: Tcs,Acs
S -> C: {T2'}_Kcs

where Acs := {C,T2'}_Kcs  (T2 is a timestamp)
```

Problems considered: 6

- secrecy of `Kcg,Kcs`
- `Client` authenticates `KeyDistributionCentre` on `n1`
- `Client` authenticates `TicketGrantingServer` on `n2`
- `Client` authenticates `Server` on `t2a`
- `Server` authenticates `Client` on `t2b`
- `TicketGrantingServer` weakly authenticates `Client` on `t1`

Attacks Found

None

Further Notes

Both the TGS and S cache the timestamps they have received in order to prevent replays as specified in RFC 1510.

HLPSL Specification

```
role KeyDistributionCentre(
             A,C,G    : agent,
             Kca,Kag  : symmetric_key,
             SND, RCV : channel(dy)
           ) played_by A def=

  local State    : nat,
        N1       : text,
        U        : text,
        Kcg      : symmetric_key (fresh),
        T1start  : text (fresh),
        T1expire: text (fresh)

  init State = 11

  knowledge(A) = {A,C,G,Kca,Kag}

  transition
    1. State  = 11 /\ RCV(U'.G.N1') =|>
       State' = 12 /\ SND(U'.{U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                          {G.Kcg'.T1start'.T1expire'.N1'}_Kca)
                   /\ witness(A,C,n1,N1')
                   /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)
end role
```

```
role TicketGrantingServer (
            G,S,C,A      : agent,
            Kag,Kgs      : symmetric_key,
            SND,RCV      : channel(dy),
            L            : text set
          ) played_by G def=

  local  State   : nat,
         N2      : text,
         U       : text,
         Kcg     : symmetric_key,
         Kcs     : symmetric_key (fresh),
         T1start, T1expire : text,
         T2start, T2expire : text (fresh),
         T1      : text

  init State = 21

  knowledge(G) = {G,S,C,A,Kag,Kgs}

  transition
    1. State  = 21 /\ RCV( S.N2'.
                           {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                           {C.T1'}_Kcg')
                           /\ not(in(T1',L))
                            =|>

       State' = 22 /\ SND(  U'.
                           {U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.
                           {S.Kcs'.T2start'.T2expire'.N2'}_Kcg')
                   /\ L' = cons(T1',L)
                   /\ wrequest(G,C,t1,T1')
                   /\ witness(G,C,n2,N2')
                   /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)
                   /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

end role
```

```
role Server(
               S,C,G    : agent,
               Kgs      : symmetric_key,
               SND, RCV : channel(dy),
               L        : text set
             ) played_by S def=

   local  State   : nat,
          U       : text,
          Kcs     : symmetric_key,
          T2expire: text,
          T2start : text,
          T2      : text

   init State = 31

   knowledge(S) = {S,G,C,Kgs}

   transition
     1. State  = 31 /\ RCV({U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.{C.T2'}_Kcs')
                    /\ not(in(T2',L)) =|>
        State' = 32 /\ SND({T2'}_Kcs')
                    /\ L' = cons(T2',L)
                    /\ witness(S,C,t2a,T2')
                    /\ request(S,C,t2b,T2')
                    /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

 end role
```

```
 role Client(
               U            : text,
               C,G,S,A       : agent,
               Kca           : symmetric_key,
               SND,RCV       : channel(dy)
             ) played_by C def=

   local  State   : nat,
          Kcs,Kcg : symmetric_key,
          T1expire: text,
          T2expire: text,
          T1start : text,
          T2start : text,
          Tcg,Tcs : {text.agent.agent.symmetric_key.text.text}_symmetric_key,
          T1,T2   : text (fresh),
          N1,N2   : text (fresh)

   init State = 1

   knowledge(C) = {U,C,G,S,A,Kca}

   transition
     1. State  = 1 /\ RCV(start)  =|>
          State' = 2 /\ SND(U.G.N1')

     2. State  = 2 /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Kca) =|>
          State' = 3 /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                     /\ witness(C,G,t1,T1')
                     /\ request(C,A,n1,N1)
                     /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

     3. State  = 3 /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
          State' = 4 /\ SND(Tcs'.{C.T2'}_Kcs')
                     /\ witness(C,S,t2b,T2')
                     /\ request(C,G,n2,N2)
                     /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

     4. State  = 4 /\ RCV({T2}_Kcs) =|>
          State' = 5 /\ request(C,S,t2a,T2)

 end role
```

```
role Session(
         U                           : text,
         A,G,C,S                     : agent,
         Kca,Kgs,Kag                 : symmetric_key,
         LS,LG                       : text set
             ) def=

  local
```

```
         SendC,ReceiveC            : channel (dy),
         SendS,ReceiveS            : channel (dy),
         SendG,ReceiveG            : channel (dy),
         SendA,ReceiveA            : channel (dy)

  composition
           Client(U,C,G,S,A,Kca,SendC,ReceiveC)
      /\   Server(S,C,G,Kgs,SendS,ReceiveS,LS)
      /\   TicketGrantingServer(G,S,C,A,Kag,Kgs,SendG,ReceiveG,LG)
      /\   KeyDistributionCentre(A,C,G,Kca,Kag,SendA,ReceiveA)

end role
```

```
role Environment() def=

  local LS, LG : text set

  const
        u1,u2                   : text,
        a,g,c,s                 : agent,
        k_ca,k_gs,k_ag,k_ia     : symmetric_key,
        t1,t2a,t2b,n1,n2        : protocol_id

  init LS = {} /\ LG = {}

  knowledge(i) = {u1,u2,a,g,c,s,k_ia
                 }

  composition

        Session(u1,a,g,c,s,k_ca,k_gs,k_ag,LS,LG)
 /\     Session(u2,a,g,i,s,k_ia,k_gs,k_ag,LS,LG)

end role
```

```
goal
```

```
  secrecy_of Kcg,Kcs
  Client authenticates KeyDistributionCentre on n1
  Client authenticates TicketGrantingServer on n2
  Client authenticates Server on t2a
  Server authenticates Client on t2b
  TicketGrantingServer weakly authenticates Client on t1

end goal
```

```
Environment()
```

17.3 cross realm version

Protocol Purpose

The Kerberos protocol is designed to operate across organisational boundaries. A client in one organisation can be authenticated to a server in another. Each organisation wishing to run a Kerberos server establishes its own "realm".

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-07.txt

Model Authors

- Vishal Sankhla, University of Southern California, August 2004

Alice&Bob style

```
1. C -> ASlocal   : C, TGSlocal, N1
2. ASlocal -> C   : C, Ticket1,
                    {TGSlocal, KC_TGSlocal, Tstart1, Texpire1, N1
                    }_KC_ASlocal
   where Ticket1 : {C, TGSlocal, KC_TGSlocal, Tstart1, Texpire1
                    }_KASlocal_TGSlocal
3. C -> TGSlocal : TGSremote, N2, Ticket1, {C, T1}_KC_TGSlocal
```

```
4. TGSlocal -> C : C, Ticket2b,
                   {TGSremote, KC_TGSremote, Tstart2b, Texpire2, N2
                   }_KC_TGSlocal
   where Ticket2b: {C,TGSremote,KC_TGSremote,Tstart2b,Texpire2
                   }_KTGSlocal_TGSremote
5. C -> TGSremote: S,N3,Ticket2b, {C, T2B}_KC_TGSremote
6. TGSremote -> C: C, Ticket3,
                   {Sremote, KC_Sremote, Tstart3,Texpire3}_KC_TGSremote
   where Ticket3 : {C, Sremote, KC_Sremote, Tstart3, Texpire3
                   }_KTGSremote_Sremote
7. C -> Sremote  : Ticket3, {C,T3}_KC_Sremote
8. Sremote -> C  : {T3}_KC_Sremote
```

Problems considered: 8

- secrecy of `KC_TGSlocal, KC_TGSremote, KC_Sremote, T3`
- `Client` authenticates `ASlocalRole` on `n1`
- `Client` authenticates `TGSlocalRole` on `n1r`
- `Client` authenticates `TGSremoteRole` on `n2`
- `Client` authenticates `SremoteRole` on `t2a`
- `SremoteRole` authenticates `Client` on `t2b`
- `TGSlocalRole` weakly authenticates `Client` on `t1`
- `TGSremoteRole` weakly authenticates `Client` on `t1r`

Attacks Found

None

Further Notes

Agents involved: Client, Local Authentication Server (ASLocal), Local Ticket Granting server (TGSlocal), Remote Ticket Granting server (TGSRemote), Remote Server where the client needs to authenticate (ServerRemote)

HLPSL Specification

```
role Client(C,
            ASlocal,
            TGSlocal,
            TGSremote,
            Sremote    : agent,
            KC_ASlocal : symmetric_key,
            SND, RCV   : channel(dy))
played_by C def=

  local State      :nat,
        T1,T2B,T3  : text(fresh),
        KC_TGSlocal,
        KC_TGSremote,
        KC_Sremote : symmetric_key,
        Ticket1,
        Ticket2b,
        Ticket3    : {agent.agent.symmetric_key.text.text}_symmetric_key,
        Tstart1,
        Texpire1,
        Tstart2b,
        Texpire2,
        Tstart3,
        Texpire3   : text,
        N1,N2,N3   : text (fresh)

  init State = 0
  knowledge(C) = {C, ASlocal, TGSlocal, TGSremote, Sremote, KC_ASlocal}

  transition

  step1.
    State = 0 /\ RCV(start) =|>
    State'= 1 /\ SND(C.TGSlocal.N1')

  step2.
    State = 1 /\ RCV(C.Ticket1'.
                  {TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'.N1}_KC_ASlocal) =|>
    State'= 2 /\ SND(TGSremote.N2'.Ticket1'.{C.T1'}_KC_TGSlocal')
              /\ witness(C,TGSlocal,t1,T1')
              /\ request(C,ASlocal,n1,N1)
              /\ secret(KC_TGSlocal',ASlocal)
              /\ secret(KC_TGSlocal',C)
              /\ secret(KC_TGSlocal',TGSlocal)

  step3.
    State = 2 /\ RCV(C.Ticket2b'.
               {TGSremote.KC_TGSremote'.Tstart2b'.Texpire2'.N2}_KC_TGSlocal) =|>
    State'= 3 /\ SND( Sremote.N3'.Ticket2b'.{C.T2B'}_KC_TGSremote')
              /\ witness(C,TGSremote,t1r,T2B')
              /\ request(C,TGSlocal,n1r,N2)
              /\ secret(KC_TGSremote',TGSlocal)
              /\ secret(KC_TGSremote',C)
              /\ secret(KC_TGSremote',TGSremote)

  step4.
    State = 3 /\ RCV(C.Ticket3'.
                  {Sremote.KC_Sremote'.Tstart3'.Texpire3'.N3}_KC_TGSremote ) =|>
    State'= 4 /\ SND (Ticket3'.{C.T3'}_KC_Sremote')
              /\ witness(C,Sremote,t2b,T3')
              /\ request(C,TGSremote,n2,N3)
              /\ secret(T3',C) /\ secret(T3',Sremote)
              /\ secret(KC_Sremote',TGSremote)
              /\ secret(KC_Sremote',C)
              /\ secret(KC_Sremote',Sremote)

  step5.
    State = 4 /\ RCV( {T3}_KC_Sremote ) =|>
    State'= 5 /\ request(C,Sremote,t2a,T3)

end role
```

```
role ASlocalRole(C,
                 ASlocal,
                 TGSlocal          : agent,
                 KC_ASlocal,
                 KASlocal_TGSlocal : symmetric_key,
                 SND ,RCV          : channel(dy))
played_by ASlocal def=
  local State            : nat,
        N1               : text,
        Tstart1,Texpire1 : text(fresh),
        KC_TGSlocal      : symmetric_key(fresh)

  init State = 6
  knowledge(ASlocal) = {C, ASlocal,TGSlocal,KC_ASlocal,KASlocal_TGSlocal}

  transition

  step1.
    State = 6 /\ RCV( C.TGSlocal.N1') =|>
    State'= 7 /\ SND(C.
             {C.TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'}_KASlocal_TGSlocal.
             {TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'.N1'}_KC_ASlocal)
              /\ witness(ASlocal,C,n1,N1')
              /\ secret(KC_TGSlocal',ASlocal)
              /\ secret(KC_TGSlocal',C)
              /\ secret(KC_TGSlocal',TGSlocal)

end role
```

```
role TGSlocalRole(C,
                  ASlocal,
                  TGSlocal,TGSremote  : agent,
                  KASlocal_TGSlocal,
                  KTGSlocal_TGSremote : symmetric_key,
                  SND ,RCV            : channel(dy),
                  L                   : text set)
played_by TGSlocal def=

  local State             : nat,
        N2                : text,
        Tstart1,  Texpire1 : text,
        Tstart2b, Texpire2 : text(fresh),
        KC_TGSlocal       : symmetric_key,
        KC_TGSremote      : symmetric_key(fresh),
        T1                : text
  init State = 8
  knowledge(TGSlocal) = {C,ASlocal,TGSlocal,TGSremote,
                         KTGSlocal_TGSremote,KASlocal_TGSlocal}

  transition

  step1.
    State = 8 /\ RCV(TGSremote.N2'.
             {C.TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'}_KASlocal_TGSlocal.
             {C.T1'}_KC_TGSlocal')
              /\ not(in(T1',L)) =|>
    State'= 9 /\ SND(C.
            {C.TGSremote.KC_TGSremote'.Tstart2b'.Texpire2'}_KTGSlocal_TGSremote.
            {TGSremote.KC_TGSremote'.Tstart2b'.Texpire2'.N2'}_KC_TGSlocal')
              /\ L' = cons(T1',L)
              /\ wrequest(TGSlocal,C,t1,T1')
              /\ witness(TGSlocal,C,n1r,N2')
              /\ secret(KC_TGSlocal',ASlocal)
              /\ secret(KC_TGSlocal',C)
              /\ secret(KC_TGSlocal',TGSlocal)
              /\ secret(KC_TGSremote',TGSlocal)
              /\ secret(KC_TGSremote',C)
              /\ secret(KC_TGSremote',TGSremote)

end role
```

```
role TGSremoteRole(C,
                   TGSlocal,
                   TGSremote,
                   Sremote             : agent,
                   KTGSlocal_TGSremote,
                   KTGSremote_Sremote  : symmetric_key,
                   SND ,RCV            : channel(dy),
                   L                   : text set )
played_by TGSremote def=

  local State            : nat,
        N3               : text,
        Tstart2b, Texpire2 : text,
        Tstart3,  Texpire3 : text(fresh),
        KC_TGSremote,
        KC_Sremote         : symmetric_key,
        T2B                : text

  init State = 10
  knowledge(TGSremote) = {C,TGSlocal,TGSremote, Sremote,
                          KTGSlocal_TGSremote,KTGSremote_Sremote}

  transition

  step1.
    State = 10 /\ RCV(Sremote.N3'.
            {C.TGSremote.KC_TGSremote'.Tstart2b'.Texpire2'}_KTGSlocal_TGSremote.
            {C.T2B'}_KC_TGSremote')
      /\ not(in(T2B',L)) =|>
    State'= 11 /\ SND(C.
             {C.Sremote.KC_Sremote'.Tstart3'.Texpire3'}_KTGSremote_Sremote.
             {Sremote.KC_Sremote'.Tstart3'.Texpire3'.N3'}_KC_TGSremote')
               /\ L' = cons(T2B',L)
               /\ wrequest(TGSremote,C,t1r,T2B')
               /\ witness(TGSremote,C,n2,N3')
               /\ secret(KC_Sremote',TGSremote)
               /\ secret(KC_Sremote',C)
               /\ secret(KC_Sremote',Sremote)
               /\ secret(KC_TGSremote',TGSlocal)
               /\ secret(KC_TGSremote',C)
               /\ secret(KC_TGSremote',TGSremote)

end role
```

```
role SremoteRole(C,
                 TGSremote,
                 Sremote            : agent,
                 KTGSremote_Sremote : symmetric_key,
                 SND ,RCV           : channel(dy),
                 L                  : text set )
played_by Sremote def=
  local State             : nat,
        Tstart3, Texpire3 : text,
        KC_Sremote        : symmetric_key,
        T3                : text

  init State = 12
  knowledge(Sremote) = {C,TGSremote, Sremote,KTGSremote_Sremote}

  transition

  step1.
    State = 12 /\
        RCV({C.Sremote.KC_Sremote'.Tstart3'.Texpire3'}_KTGSremote_Sremote.
             {C.T3'}_KC_Sremote')
               /\ not(in(T3',L)) =|>
    State'= 13 /\ SND({T3'}_KC_Sremote')
               /\ L' = cons(T3',L)
               /\ witness(Sremote,C,t2a,T3')
               /\ request(Sremote,C,t2b,T3')
               /\ secret(KC_Sremote',TGSremote)
               /\ secret(KC_Sremote',C)
               /\ secret(KC_Sremote',Sremote)
               /\ secret(T3',C) /\ secret(T3',Sremote)

end role
```

```
role Session(C,ASlocal,TGSlocal,TGSremote,Sremote   : agent,
             KC_ASlocal,KASlocal_TGSlocal           : symmetric_key,
             KTGSlocal_TGSremote,KTGSremote_Sremote : symmetric_key,
             LTGSlocal, LTGSremote, LSremote        : text set )
def=

  local Send1, Send2, Send3, Send4, Send5,
        Receive1, Receive2, Receive3, Receive4, Receive5: channel (dy)

  composition
    Client(C,ASlocal,TGSlocal,TGSremote,Sremote,KC_ASlocal,Send1,Receive1)
    /\  ASlocalRole(C,ASlocal,TGSlocal,
                    KC_ASlocal, KASlocal_TGSlocal,Send2,Receive2)
    /\  TGSlocalRole(C,ASlocal,TGSlocal,TGSremote,
                     KASlocal_TGSlocal, KTGSlocal_TGSremote,
                     Send3,Receive3,LTGSlocal)
    /\  TGSremoteRole(C,TGSlocal,TGSremote,Sremote,
                      KTGSlocal_TGSremote,KTGSremote_Sremote,
                      Send4,Receive4,LTGSremote)
    /\  SremoteRole(C,TGSremote,Sremote,KTGSremote_Sremote,
                    Send5,Receive5,LSremote)

end role
```

```
role Environment() def=

  local LTGSL, LTGSR, LS : text set

  const c, asl, tgsl, tgsr, s : agent,
        ki_aslocal,
        kc_aslocal,
        kaslocal_tgslocal,
        ktgslocal_tgsremote,
        ktgsremote_sremote    : symmetric_key,

        t1,t1r,t2a,t2b,n1,n1r,n2: protocol_id

  init LTGSL = {} /\ LTGSR = {} /\ LS = {}

  knowledge(i) = {c,asl,tgsl,tgsr,s,ki_aslocal
                 }

  composition

        Session(c,asl,tgsl,tgsr,s,
                kc_aslocal,kaslocal_tgslocal,ktgslocal_tgsremote,
                ktgsremote_sremote,LTGSL,LTGSR,LS)
 /\     Session(i,asl,tgsl,tgsr,s,
                ki_aslocal,kaslocal_tgslocal,ktgslocal_tgsremote,
                ktgsremote_sremote,LTGSL,LTGSR,LS)
```

```
end role
```

```
goal

  secrecy_of KC_TGSlocal, KC_TGSremote, KC_Sremote, T3
  Client authenticates ASlocalRole   on n1
  Client authenticates TGSlocalRole  on n1r
  Client authenticates TGSremoteRole on n2
  Client authenticates SremoteRole   on t2a
  SremoteRole   authenticates Client   on t2b
  TGSlocalRole  weakly authenticates Client  on t1
  TGSremoteRole weakly authenticates Client on t1r

end goal
```

```
Environment()
```

17.4 with forwardable ticket

Protocol Purpose

Mutual authentication

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-07.txt

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004
- Vishal Sankhla, University of Southern California, 2004

Alice&Bob style

```
  C -> A: U,G,N1
  A -> C: U,Tcg,{G,Kcg,T1start,T1expire,N1}_Kca

  where Tcg := {U,C,G,Kcg,T1start,T1expire}_Kag
        A := Authentication Server

  C -> G: IP-ADDR,S,N2,Tcg,Acg,FORWARDABLE
  G -> C: U,Tcs1,{S,Kcs,T2start,T2expire,N2}_Kcg

  where Acg  := {C,T1}_Kcg  (T1 is a timestamp)
        Tcs1 := {IP-ADDR,U,C,S,Kcs,T2start,T2expire,FORWARDABLE}_Kgs

  C -> G: IP-ADDR,S,N2,Tcs1,Acg
  G -> C: U,Tcs2,{S,Kcs,T2start,T2expire,N2}_Kcg

  where Acg  := {C,T1}_Kcg  (T1 is a timestamp)
        Tcs2 := {IP-ADDR,U,C,S,Kcs,T2start,T2expire,FORWARDABLE}_Kgs

  C -> S: Tcs2,Acs
  S -> C: {T2'}_Kcs

  where Acs := {C,T2'}_Kcs  (T2 is a timestamp)

**********************************************************************

 An alternative instance of the protocol in action.  The
 client does not request a forwardable ticket, and does not
 change IP address.

  C -> A: U,G,N1
  A -> C: U,Tcg,{G,Kcg,T1start,T1expire,N1}_Kca

  where Tcg := {U,C,G,Kcg,T1start,T1expire}_Kag
        A := Authentication Server

  C -> G: IP-ADDR,S,N2,Tcg,Acg,NOT_FORWARDABLE
  G -> C: U,Tcs1,{S,Kcs,T2start,T2expire,N2}_Kcg

  where Acg  := {C,T1}_Kcg  (T1 is a timestamp)
```

```
        Tcs1 := {IP-ADDR,U,C,S,Kcs,T2start,T2expire,NOT_FORWARDABLE}_Kgs

  C -> S: Tcs1,Acs
  S -> C: {T2'}_Kcs

  where Acs := {C,T2'}_Kcs  (T2 is a timestamp)
```

Problems considered: 6

- secrecy of `Kcg, Kcs`
- `Client` authenticates `AuthenticationServer` on `n1`
- `Client` authenticates `TicketGrantingServer` on `n2`
- `Client` authenticates `Server` on `t2a`
- `Server` authenticates `Client` on `t2b`
- `TicketGrantingServer` weakly authenticates `Client` on `t1`

Attacks Found

None

Further Notes

- Same as plain Kerberos V except that if the client requests a forwardable ticket from the TGS, then sends this back to the TGS to get a ticket for a new IP address.
- IP address is a local nonce to client, and is included in requests and tickets.
- The IP address is also changed before requesting a new ticket, naturally.

HLPSL Specification

```
role AuthenticationServer(
    A,C,G    : agent,
    Kca,Kag   : symmetric_key,
    SND, RCV : channel(dy))
played_by A def=

  local
    State    : nat,
    N1       : text,
    U        : text,
    Kcg      : symmetric_key (fresh),
    T1start  : text (fresh),
    T1expire : text (fresh)

  init
    State = 11

  knowledge(A) = {A,C,G,Kca,Kag}

  transition

 1. State  = 11 /\ RCV(U'.G.N1') =|>
    State' = 12 /\ SND(U'.{U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                       {G.Kcg'.T1start'.T1expire'.N1'}_Kca       )
                /\ witness(A,C,n1,N1')
                /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)
end role
```

```
role TicketGrantingServer (
    G,S,C,A            : agent,
    Kag,Kgs            : symmetric_key,
    SND,RCV            : channel(dy),
    L                  : text set)
played_by G def=

  local
    State    : nat,
    N2       : text,
    U        : text,
    Kcg      : symmetric_key,
    Kcs      : symmetric_key (fresh),
    T1start  : text,
    T2start  : text (fresh),
    T1expire : text,
    T2expire : text (fresh),
    T1       : text,
    IP_ADDR  : text,
    Forwardable_or_not  : protocol_id

  const  forwardable    : protocol_id

  init
    State = 21

  knowledge(G) = {G,S,C,A,Kag,Kgs,L}

  transition

 1. State  = 21
      /\ RCV(IP_ADDR'.S.N2'.
            {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
            {C.T1'}_Kcg'.
            Forwardable_or_not')
      %% T1' should not have been received before
      /\ not(in(T1',L))
    =|>
    State' = 22
      /\ SND(U'.
            {IP_ADDR'.U'.C.S.Kcs'.T2start'.T2expire'.Forwardable_or_not'}_Kgs.
            {S.Kcs'.T2start'.T2expire'.N2'}_Kcg')
      /\ L' = cons(T1',L)
      /\ wrequest(G,C,t1,T1')
      /\ witness(G,C,n2,N2')
      /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)
      /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

 3. State  = 22
      /\ RCV(IP_ADDR.S.N2.
             {IP_ADDR.U.C.S.Kcs.T2start.T2expire.forwardable}_Kgs.
             {C.T1}_Kcg)
      /\ Forwardable_or_not = forwardable
    =|>
    State' = 23
      /\ SND(U.
             {IP_ADDR.U.C.S.Kcs.T2start.T2expire.forwardable}_Kgs.
             {S.Kcs.T2start.T2expire.N2}_Kcg)

end role
```

```
role Server(
    S,C,G     : agent,
    Kgs       : symmetric_key,
    SND, RCV : channel(dy),
    L         : text set)
played_by S def=

  local
    State    : nat,
    U        : text,
    Kcs      : symmetric_key,
    T2expire: text,
    T2start : text,
    T2       : text,
    IP_ADDR : text,
    Forwardable_or_not : protocol_id

  init
    State = 31

  knowledge(S) = {S,G,C,Kgs,L}

  transition

 1. State  = 31
      /\ RCV({IP_ADDR'.U'.C.S.Kcs'.T2start'.T2expire'.Forwardable_or_not'}_Kgs.
             {C.T2'}_Kcs')
      /\ not(in(T2',L)) =|>
    State' = 32
      /\ SND({T2'}_Kcs')
      /\ L' = cons(T2',L)
      /\ witness(S,C,t2a,T2')
      /\ request(S,C,t2b,T2')
      /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)
end role
```

```
role Client(
    C,G,S,A         : agent,
    U               : text,
    Kca             : symmetric_key,
    SND,RCV         : channel(dy))
played_by C def=

  local
    State   : nat,
    Kcs     : symmetric_key,
    T1expire: text,
    T2expire: text,
    T1start : text,
    T2start : text,
    Kcg     : symmetric_key,
    T1      : text (fresh),
    T2      : text (fresh),
    IP_ADDR : text (fresh),
    Tcg     : {text.agent.agent.symmetric_key.text.text}_symmetric_key,
    Tcs1, Tcs2:
     {text.text.agent.agent.symmetric_key.text.text.protocol_id}_symmetric_key,
    N1, N2  : text (fresh)

  const forwardable, un_forwardable : protocol_id

  init
    State = 1

  knowledge(C) = {C,G,S,A,U,Kca}

  transition
```

```
 1. State  = 1 /\ RCV(start)  =|>
    State' = 2 /\ SND(U.G.N1')

21. State  = 2 /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Kca) =|>
    State' = 3 /\ SND(IP_ADDR'.S.N2'.Tcg'.{C.T1'}_Kcg'.forwardable)
               /\ witness(C,G,t1,T1')
               /\ request(C,A,n1,N1)
               /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

22. State  = 2 /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Kca) =|>
    State' = 4 /\ SND(IP_ADDR'.S.N2'.Tcg'.{C.T1'}_Kcg'.un_forwardable)
               /\ witness(C,G,t1,T1')
               /\ request(C,A,n1,N1)
               /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

 3. State  = 3 /\ RCV(U.Tcs1'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
    State' = 4 /\ SND(IP_ADDR.S.N2.Tcs1'.{C.T1}_Kcg)
               /\ request(C,G,n2,N2)
               /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

 4. State  = 4 /\ RCV(U.Tcs2'.{S.Kcs'.T2start.T2expire.N2}_Kcg) =|>
    State' = 5 /\ SND(Tcs2'.{C.T2'}_Kcs')
               /\ witness(C,S,t2b,T2')

 5. State  = 5 /\ RCV({T2}_Kcs) =|>
    State' = 6 /\ request(C,S,t2a,T2)

end role
```

```
role Session(
    A,G,C,S                     : agent,
    U                           : text,
    Kca,Kgs,Kag                 : symmetric_key,
    LS,LG                       : text set) def=

  local
    SendC,ReceiveC              : channel (dy),
    SendS,ReceiveS              : channel (dy),
    SendG,ReceiveG            : channel (dy),
    SendA,ReceiveA            : channel (dy)

  composition
    Client(C,G,S,A,U,Kca,SendC,ReceiveC)
 /\ Server(S,C,G,Kgs,SendS,ReceiveS,LS)
 /\ TicketGrantingServer(G,S,C,A,Kag,Kgs,SendG,ReceiveG,LG)
 /\ AuthenticationServer(A,C,G,Kca,Kag,SendA,ReceiveA)

end role
```

```
role Environment() def=

  local LS, LG : text set

  const
    a,g,c,s                         : agent,
    u1,u2                           : text,
    k_ca,k_gs,k_ag,k_ia             : symmetric_key,
    t1,t2a,t2b,n1,n2                : protocol_id,
    forwardable, un_forwardable     : protocol_id

  init LS = {} /\ LG = {}

  knowledge(i) = {a,g,c,s,k_ia,forwardable,u1,u2
                 }

  composition

        Session(a,g,c,s,u1,k_ca,k_gs,k_ag,LS,LG)
 /\     Session(a,g,i,s,u2,k_ia,k_gs,k_ag,LS,LG)

end role
```

```
goal

  secrecy_of Kcg, Kcs
  Client authenticates AuthenticationServer on n1
  Client authenticates TicketGrantingServer on n2
  Client authenticates Server on t2a
  Server authenticates Client on t2b
  TicketGrantingServer weakly authenticates Client on t1

end goal
```

```
Environment()
```

17.5 public key initialisation

Protocol Purpose

Mutual Authentication with Public Key initialisation (in case the Authentication Server and Client don't share a key)

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-22.txt

Model Authors

- Vishal Sankhla, University of Southern California, August 2004
- Daniel Plasto for Siemens CT IC 3, 2004

Alice&Bob style

```
  C -> A: U,G,N1,{Kca,T0,N1,hash(U,G,N1)}inv(Kca)

    In PKINIT, the first message contains additional
    information in the pre-authentication field:
    The public key of U, a timestamp, the nonce repeated,
    and a checksum of the message body.  This is all signed
    with the private key of U.
  A -> C: U,Tcg,{G,Kcg,T1start,T1expire,N1}Ktemp,{{Ktemp}Kca}inv(Pka)

  where Tcg := {U,C,G,Kcg,T1start,T1expire}Kag

    A replies as usual, except the reply is encrypted
    with a random key, and this key is included in the
    pre-authentication field and encrypted with the U's
    public key and signed with the A's private key.

  C -> G: S,N2,Tcg,Acg
  G -> C: U,Tcs,{S,Kcs,T2start,T2expire,N2}Kcg

  where Acg := {C,T1}Kcg  (T1 is a timestamp)
        Tcs := {U,C,S,Kcs,T2start,T2expire}Kgs

  C -> S: Tcs,Acs
  S -> C: {T2'}Kcs

  where Acs := {C,T2'}Kcs  (T2 is a timestamp)
```

The AS, TGS and S cache the timestamps they have received in order to prevent replays as specified in RFC 1510.

We assume that the Key Distribution Centre (KDC) is the certifying authority here.

Problems considered: 7

- secrecy of Kcg,Kcs
- Client authenticates AuthenticationServer on n1
- Client authenticates TicketGrantingServer on n2
- Client authenticates Server on t2a
- Server authenticates Client on t2b
- TicketGrantingServer weakly authenticates Client on t1
- AuthenticationServer weakly authenticates Client on t0

Attacks Found

None

HLPSL Specification

```
role AuthenticationServer(
            A,C,G    : agent,
            Kca      : public_key,
            Kag      : symmetric_key,
            SND, RCV : channel(dy),
            L        : text set,
            Pka      : public_key,
            Hash     : function
          ) played_by A def=

  local State   : nat,
        N1      : text,
        U       : text,
        T0      : text,
        Kcg     : symmetric_key (fresh),
        T1start  : text (fresh),
        T1expire : text (fresh),
        Ktemp    : symmetric_key(fresh)

  init State = 11

  knowledge(A) = {A,C,G,Kca,Kag,L,Pka,inv(Pka)}

  transition
    1. State  = 11 /\ RCV(U'.G.N1'.
                          {Kca.T0'.N1'.Hash(U'.G.N1')}_inv(Kca))
                   /\ not(in(T0',L)) =|>
       State' = 12 /\ SND(U'.
                          {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                          {G.Kcg'.T1start'.T1expire'.N1'}_Ktemp'.
                          {{Ktemp'}_Kca}_inv(Pka))
                   /\ L' = cons(T0',L)
```

```
                    /\ witness(A,C,n1,N1')
                    /\ wrequest(A,C,t0,T0')
                    /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

end role
```

```
role TicketGrantingServer (
             G,S,C,A      : agent,
             Kag,Kgs      : symmetric_key,
             SND,RCV      : channel(dy),
             L            : text set
           ) played_by G def=

  local State   : nat,
        N2      : text,
        U       : text,
        Kcg     : symmetric_key,
        Kcs     : symmetric_key (fresh),
        T1start,T1expire : text,
        T2start,T2expire : text (fresh),
        T1      : text

  init State = 21

  knowledge(G) = {G,S,C,A,Kag,Kgs,L}

  transition
    1. State  = 21 /\ RCV( S.N2'.
                           {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                           {C.T1'}_Kcg')
                           /\ not(in(T1',L)) =|>
       State' = 22 /\ SND(  U'.
                           {U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.
                           {S.Kcs'.T2start'.T2expire'.N2'}_Kcg'
                         )
                    /\ L' = cons(T1',L)
                    /\ wrequest(G,C,t1,T1')
                    /\ witness(G,C,n2,N2')
                    /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)
```

```
                     /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

end role
```

```
role Server(
             S,C,G    : agent,
             Kgs   : symmetric_key,
             SND, RCV : channel(dy),
             L       : text set
           ) played_by S def=

  local  State   : nat,
         U       : text,
         Kcs     : symmetric_key,
         T2expire: text,
         T2start : text,
         T2      : text

  init State = 31

  knowledge(S) = {S,G,C,Kgs,L}

  transition
    1. State  = 31 /\ RCV({U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.{C.T2'}_Kcs')
                   /\ not(in(T2',L)) =|>
       State' = 32 /\ SND({T2'}_Kcs')
                   /\ L' = cons(T2',L)
                   /\ witness(S,C,t2a,T2')
                   /\ request(S,C,t2b,T2')
                   /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

end role
```

```
role Client(
             C,G,S,A      : agent,
             SND,RCV      : channel(dy),
             Kca,Pka      : public_key,
             U            : text,
             Hash         : function

           ) played_by C def=

  local State    : nat,
         Kcs     : symmetric_key,
         T1expire: text,
         T2expire: text,
         T1start : text,
         T2start : text,
         Kcg     : symmetric_key,
         Tcg,Tcs : {text.agent.agent.symmetric_key.text.text}_symmetric_key,
         T0      : text (fresh),
         T1      : text (fresh),
         T2      : text (fresh),
         Ktemp   : symmetric_key,
         N1, N2  : text (fresh)

  init State = 1

  knowledge(C) = {C,G,S,A,U,Pka,Kca,inv(Kca)}

  transition
    1. State  = 1 /\ RCV(start) =|>
       State' = 2 /\ SND(U.G.N1'.{Kca.T0'.N1'.Hash(U.G.N1')}_inv(Kca))
                  /\ witness(C,A,t0,T0')

    2. State  = 2 /\ RCV(U.Tcg'.
                         {G.Kcg'.T1start'.T1expire'.N1}_Ktemp'.
                         {{Ktemp'}_Kca}_inv(Pka)) =|>
       State' = 3 /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                  /\ witness(C,G,t1,T1')
                  /\ request(C,A,n1,N1)
                   /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

    3. State  = 3 /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
       State' = 4 /\ SND(Tcs'.{C.T2'}_Kcs')
                  /\ witness(C,S,t2b,T2')
                  /\ request(C,G,n2,N2)
                  /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)
    4. State  = 4 /\ RCV({T2}_Kcs) =|>
       State' = 5 /\ request(C,S,t2a,T2)

end role
```

```
role Session(

         A,G,C,S                      : agent,
         Kag,Kgs                      : symmetric_key,
         LS                           : text set,
         Hash                         : function,
         U                            : text,
         Kca,Pka                      : public_key
             ) def=

  local
         SendC,ReceiveC               : channel (dy),
         SendS,ReceiveS               : channel (dy),
         SendG,ReceiveG               : channel (dy),
         SendA,ReceiveA               : channel (dy)

  composition
           Client(C,G,S,A,SendC,ReceiveC,Kca,Pka,U,Hash)
       /\  Server(S,C,G,Kgs,SendS,ReceiveS,LS)
       /\  TicketGrantingServer(G,S,C,A,Kag,Kgs,SendG,ReceiveG,LS)
       /\  AuthenticationServer(A,C,G,Kca,Kag,SendA,ReceiveA,LS,Pka,Hash)

end role
```

```
role Environment() def=

  local LS : text set

  const a,g,c,s             : agent,
        k_ag,k_gs           : symmetric_key,
        kia,kca,pka         : public_key,
        hash_              : function,
        u1,u2              : text,
        t0,t1,t2a,t2b,n1,n2 : protocol_id

  init LS = {}

  knowledge(i) = {a,g,c,s,pka,hash_,k_gi,u1,u2
}

  composition
        Session(a,g,c,s,k_ag,k_gs,LS,hash_,u1,kca,pka)
 /\     Session(a,g,i,s,k_ag,k_gs,LS,hash_,u2,kia,pka)

end role
```

```
goal

  secrecy_of Kcg,Kcs
  Client authenticates AuthenticationServer on n1
  Client authenticates TicketGrantingServer on n2
  Client authenticates Server on t2a
  Server authenticates Client on t2b
  TicketGrantingServer weakly authenticates Client on t1
  AuthenticationServer weakly authenticates Client on t0

end goal
```

```
Environment()
```

17.6 with PA-ENC-TIMESTAMP pre-authentication method

Protocol Purpose

Mutual authentication

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-preauth-framework-02.txt

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004
- Vishal Sankhla, University of Southern California, 2004

Alice&Bob style

```
C -> A: U,G,N1,{C,T0}_Kca
A -> C: U,Tcg,{G,Kcg,T1start,T1expire,N1}_Kca

where Tcg := {U,C,G,Kcg,T1start,T1expire}_Kag
      A := Key Distribution Centre

C -> G: S,N2,Tcg,Acg
G -> C: U,Tcs,{S,Kcs,T2start,T2expire,N2}_Kcg

where Acg := {C,T1}_Kcg  (T1 is a timestamp)
      Tcs := {U,C,S,Kcs,T2start,T2expire}_Kgs

C -> S: Tcs,Acs
S -> C: {T2'}_Kcs

where Acs := {C,T2'}_Kcs  (T2 is a timestamp)
```

Problems considered: 7

- secrecy of `Kcg,Kcs`
- `Client` authenticates `AuthenticationServer` on `n1`
- `Client` authenticates `TicketGrantingServer` on `n2`
- `Client` authenticates `Server` on `t2b`
- `Server` authenticates `Client` on `t2a`
- `TicketGrantingServer` weakly authenticates `Client` on `t1`

- `AuthenticationServer` weakly authenticates `Client` on `t0`

Attacks Found

None

Further Notes

The AS, TGS and S cache the timestamps they have received in order to prevent replays as specified in RFC 1510.

HLPSL Specification

```
role AuthenticationServer(
             A,C,G    : agent,
             Kca,Kag   : symmetric_key,
             SND, RCV : channel(dy),
             L        : text set
           ) played_by A def=

  local State     : nat,
        N1        : text,
        U         : agent,
        T0        : text,
        Kcg       : symmetric_key (fresh),
        T1start   : text (fresh),
        T1expire  : text (fresh)

  init State = 11

  knowledge(A) = {A,C,G,Kca,Kag,L}

  transition
    1. State  = 11 /\ RCV(U'.G.N1'.{C.T0'}_Kca)
                   /\ not(in(T0',L)) =|>
       State' = 12 /\ SND(U'.
                          {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
```

```
                             {G.Kcg'.T1start'.T1expire'.N1'}_Kca)
                     /\ L' = cons(T0',L)
                     /\ witness(A,C,n1,N1')
                     /\ wrequest(A,C,t0,T0')
                     /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

end role
```

```
role TicketGrantingServer (
             G,S,C,A      : agent,
             Kag,Kgs      : symmetric_key,
             SND,RCV      : channel(dy),
             L            : text set
           ) played_by G def=

  local State     : nat,
        N2        : text,
        U         : agent,
        Kcg       : symmetric_key,
        Kcs       : symmetric_key (fresh),
        T1start,T1expire  : text,
        T2start, T2expire : text (fresh),
        T1        : text

  init State = 21

  knowledge(G) = {G,S,C,A,Kag,Kgs,L}

  transition
    1. State  = 21 /\ RCV(S.N2'.
                             {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                             {C.T1'}_Kcg')
                     /\ not(in(T1',L))
       =|>
       State' = 22 /\ SND(U'.
                             {U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.
                             {S.Kcs'.T2start'.T2expire'.N2'}_Kcg')
                     /\ L' = cons(T1',L)
                     /\ wrequest(G,C,t1,T1')
```

```
                    /\ witness(G,C,n2,N2')
                    /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)
                    /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

end role
```

```
role Server(
            S,C,G    : agent,
            Kgs      : symmetric_key,
            SND, RCV : channel(dy),
            L        : text set
          ) played_by S def=

  local State    : nat,
        U        : agent,
        Kcs      : symmetric_key,
        T2expire : text,
        T2start  : text,
        T2       : text

  init State = 31

  knowledge(S) = {S,G,C,Kgs,L}

  transition
    1. State  = 31 /\ RCV({U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.
                          {C.T2'}_Kcs')
                   /\ not(in(T2',L)) =|>
       State' = 32 /\ SND({T2'}_Kcs')
                   /\ L' = cons(T2',L)
                   /\ request(S,C,t2a,T2')
                   /\ witness(S,C,t2b,T2')
                   /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

end role
```

```
role Client(
            C,G,S,A      : agent,
            U            : agent,
            Kca          : symmetric_key,
            SND,RCV      : channel(dy)
          ) played_by C def=

local State     : nat,
       Kcs      : symmetric_key,
       T1expire : text,
       T2expire : text,
       T1start  : text,
       T2start  : text,
       Kcg      : symmetric_key,
       Tcg,Tcs  : {agent.agent.agent.symmetric_key.text.text}_symmetric_key,
       T0       : text (fresh),
       T1       : text (fresh),
       T2       : text (fresh),
       N1, N2   : text (fresh)

init State = 1

knowledge(C) = {C,G,S,A,U,Kca}

transition
  1. State  = 1 /\ RCV( start )  =|>
     State' = 2 /\ SND(U.G.N1'.{C.T0'}_Kca)
                /\ witness(C,A,t0,T0')

  2. State  = 2 /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Kca) =|>
     State' = 3 /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                /\ witness(C,G,t1,T1')
                /\ request(C,A,n1,N1)
                /\ secret(Kcg',A) /\ secret(Kcg',C) /\ secret(Kcg',G)

  3. State  = 3 /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
     State' = 4 /\ SND(Tcs'.{C.T2'}_Kcs')
                /\ witness(C,S,t2a,T2')
                /\ request(C,G,n2,N2)
                /\ secret(Kcs',G) /\ secret(Kcs',C) /\ secret(Kcs',S)

  4. State  = 4 /\ RCV({T2}_Kcs) =|>
```

```
         State' = 5
                    /\ request(C,S,t2b,T2)

  end role
```

```
role Session(A,G,C,S                    : agent,
             U                          : agent,
             Kca,Kgs,Kag                : symmetric_key,
             LS,LG,LA                   : text set) def=

        local
             SendC,ReceiveC             : channel (dy),
             SendS,ReceiveS             : channel (dy),
             SendG,ReceiveG             : channel (dy),
             SendA,ReceiveA             : channel (dy)
  composition
           Client(C,G,S,A,U,Kca,SendC,ReceiveC)
       /\  Server(S,C,G,Kgs,SendS,ReceiveS,LS)
       /\  TicketGrantingServer(G,S,C,A,Kag,Kgs,SendG,ReceiveG,LG)
       /\  AuthenticationServer(A,C,G,Kca,Kag,SendA,ReceiveA,LA)

end role
```

```
role Environment() def=

  local LS, LG, LA : text set

  const a,g,c,s                  : agent,
        kca,kgs,kag              : symmetric_key,
        kia                      : symmetric_key,
        u1,u2                    : agent,
        t0,t1,t2a,t2b,n1,n2      : protocol_id

  init LS = {} /\ LG = {} /\ LA = {}

  knowledge(i) = {a,g,c,s,u1,u2,kia
                 }
  composition

        Session(a,g,c,s,u1,kca,kgs,kag,LS,LG,LA) % normal session
 /\     Session(a,g,i,s,u2,kia,kgs,kag,LS,LG,LA) % i is Client
 /\     Session(a,g,c,i,u3,kca,kgi,kag,LS,LG,LA) % i is Server

end role
```

```
goal

  secrecy_of Kcg,Kcs
  Client authenticates AuthenticationServer on n1
  Client authenticates TicketGrantingServer on n2
  Client authenticates Server on t2b
  Server authenticates Client on t2a
  TicketGrantingServer weakly authenticates Client on t1
  AuthenticationServer weakly authenticates Client on t0

end goal
```

```
Environment()
```

18 TESLA: Timed Efficient Stream Loss-tolerant Authentication

Protocol Purpose

Secure source authentication mechanism for multicast or broadcast data streams

Definition Reference

http://www.ece.cmu.edu/~adrian/tesla.html

Model Authors

David von Oheimb, Siemens CT IC 3, August 2004

Alice&Bob style

`S` chooses `N` (the number of messages to broadcast) and a random symmetric key `K_N`.

```
 0. S -> R: {N.K_0}inv(k_S)
 i. S -> R: M_i.hash(K_i,M_i).K_i-1
```

where
`F` is a one-way function
`K_0 = F^N(K_N)` is the `N`-th power of `F` on `K_N`
`K_i = F^i(K_N)` is the `i`-th power of `F` on `K_N`

Note that the last message `M_N` cannot be authenticated because the corresponding key `K_N` is never revealed.

Model Limitations

Issues abstracted from:

- Real-time issues including initial synchronisation (may be critical)
- Delay other than 1 (should not make a difference wrt security here)
- Any number of packets per interval (should not be critical)
- Multiple receivers (no problem, since receivers are independent of each other)

The count of rounds, N should be a parameter, but is hard-wired to be 3 here.

The current model assumes that the sender sends messages one per time interval and the receiver receives these messages one per time interval - with the possibility of gaps, i.e. he may miss a message. The current model does not include the possibility of messages being delayed, i.e. being received in a later time interval.

Problems considered: 1

- Sender is authenticated on msgstream

Attacks Found

None

Further Notes

Since function exponentiation $F^N(X)$ (N-times repeated application of F on X), is not directly expressible, we have to model this via loops.

A variant with lazy generation of one-way chains is commented out.

We send artificial time ticks to keep the Sender synchronised with the Receiver.

Since protocol_ids are used in the goals section and the third argument of witness and request must be atomic, we use the single constant msgstream to identify the whole message stream rather than individual messages. Yet the check for authentication is fine since the matching of witness and request also takes the fourth argument of witness and request into account.

HLPSL Specification

```
role Sender (S: agent,
             SND, RCV: channel(dy),
             F: function,
             K_S: public_key)
played_by S def=

  local State: nat,
        Time, N: message, % current time and final time, should be: text,
        K_prev, K: message, % should be: symmetric_key,
        M: message (fresh)
  const k_N: symmetric_key
  init State = 0

  knowledge(S) = {inv(K_S),tick,t_0}

  transition

  0. State  = 0 /\ RCV(start)
                /\ K_prev' = F(F(F(k_N))) =|>                       % 3 rounds
%               /\ K_prev' = K_prev' =|>  % lazy generation of one-way chain!
     State' = 1 /\ Time' = t_0 /\ N' = tick(tick(tick(t_0))) %       3 rounds
                /\ SND({tick(N').F(K_prev')}_inv(K_S)) % send tick(N') instead
    % of N' to prevent the intruder from replaying N' before Receiver sends N'

  1. State  = 1 /\ RCV(Time) % keeps the Sender synchronised with the Receiver
%t_0 and tick must not be known to the intruder in order to be used as a signal
%that can only be generated by the Receiver
                /\ K_prev = F(K') /\ Time /= N =|>
     State' = 1 /\ SND(M'.hash_(K',M').F(K'))
/\ K_prev' = K'
                /\ Time' = tick(Time)
                /\ witness(S,S,msgstream,M') %msgstream should be: tick(Time)

%this transition is not really necessary; it just closes the lazy one-way chain.
% 2. State  = 1 /\ RCV(Time) /\ Time = N /\ K_prev = k_N =|>
%    State' = 2

end role
```

```
role Receiver (R, S: agent,
               SYNC, RCV: channel(dy),
               F: function,
               K_S: public_key)
played_by R def=

  local State: nat,
        Time, N: message, % should be: text,
        T_prev:  message, % time when M_prev was sent, should be: text,
        K_prev_prev, K_prev, K_prev2: message, % should be: symmetric_key,
        M_prev, M: message,
        Hash_prev, Hash: message, % should be: text
        Compare: bool,
        Gap,Gap2: message % should be: nat

  const true, false: bool,
        zero: nat,
        succ: nat -> nat,
        buffered, compared_and_buffered: protocol_id % signals just for tracing

  init State = 3

  knowledge(R) = {hash_,tick,t_0}

  transition

   initialise.
     State  = 3 /\ RCV({tick(N').K_prev_prev'}_inv(K_S)) =|>
     State' = 4 /\ Compare' = false /\ Gap' = zero
                /\ Time' = t_0 /\ SYNC(Time')

   arrive.
     State  = 4 /\ Time /= N
                /\ RCV(M'.Hash'.K_prev') =|>
     State' = 5 /\ K_prev2' = K_prev' /\ Gap2' = zero

   adjust_K_prev2.
                   RCV(start) /\
     State  = 5 /\ Gap2 /= Gap =|>
     State' = 5 /\ K_prev2' = F(K_prev2) /\ Gap2' = succ(Gap2)

   buffer.
                   RCV(start) /\
     State  = 5 /\ Compare = false /\ Gap2 = Gap
                /\ K_prev_prev = F(K_prev2) =|>
     State' = 4 /\ K_prev_prev' = K_prev
                /\ M_prev' = M /\ Hash_prev' = Hash
                /\ T_prev' = tick(Time)
                /\ Compare' = true /\ Gap' = zero
                /\ Time' = tick(Time) /\ SYNC(Time'.buffered)
   compare_and_buffer.
                   RCV(start) /\
     State  = 5 /\ Compare = true /\ Gap2 = Gap
                /\ Hash_prev = hash_(K_prev2,M_prev) % check previous message
                /\ K_prev_prev = F(K_prev2) =|>
     State' = 4 /\ K_prev_prev' = K_prev
                /\ M_prev' = M /\ Hash_prev' = Hash
                /\ T_prev' = tick(Time)
                /\ Compare' = true /\ Gap' = zero
                /\ Time' = tick(Time) /\ SYNC(Time'.compared_and_buffered)
                /\ request(S,S,msgstream,M_prev) %msgstream should: be T_prev

   lose.
     State  = 4 /\ Time /= N
                /\ RCV(loss) =|>
     State' = 4 /\ Gap' = succ(Gap)
                /\ Time' = tick(Time) /\ SYNC(Time')

end role
```

```
role Session(S,R: agent,
             SR, SYNC: channel (dy),
             F: function,
             K_S: public_key)
def=
  composition
           Sender  (S,    SR, SYNC, F, K_S)
       /\  Receiver(R, S, SYNC, SR, F, K_S)
end role
```

```
role Environment() def=

  const s,r: agent,
        sr,ir,sync: channel (dy),
        hash_: hash,
        f: function,
        k_S: public_key,
        tick: text -> text,
        t_0: text,
        loss: text,
        msgstream: protocol_id

  knowledge(i) = {s,r,hash_,loss,f,k_S}

  composition
        Session(s,r,sr,sync,f,k_S)
%    /\ Session(i,r,ir,sync,f,k_S)

end role
```

```
goal

  Sender authenticates Sender on msgstream

end goal
```

```
Environment()
```

19 SSH Transport Layer Protocol

Protocol Purpose

Secure authentication mechanism (of server) and key exchange

Definition Reference

http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-22.txt

Model Authors

David von Oheimb, Siemens CT IC 3, August 2004

Alice&Bob style

setting up the transport, including key exchange:

```
1. C -> S: NC
2. S -> C: NS
3. C -> S: exp(g,X)
4. S -> C: k_S.exp(g,Y).{H}_inv(k_S)
with K=exp(exp(g,X),Y), H=hash(NC.NS.k_S.exp(g,X).exp(g,Y).K)
```

user authentication, connections, etc:

```
5. C -> S: {XXX}_KCS with SID=H, KCS=hash(K.H.c.SID)
6. S -> C: {YYY}_KSC with SID=H, KSC=hash(K.H.d.SID)
```

Model Limitations

Issues abstracted from:

- version strings and matching
- algorithm negotiation for encryption, hashing, etc.
- Binary packet protocol/format, MAC, sequence numbers
- message numbers (i.e. message type identifiers)
- `first_kex_packet_follows`

- alternative key exchange algorithms (other than Diffie-Hellman)
- `SSH_MSG_NEWKEYS`, key re-exchange
- `SSH_MSG_DISCONNECT`, `SSH_MSG_DEBUG`, `SSH_MSG_IGNORE`, `SSH_MSG_UNIMPLEMENTED`
- `SSH_MSG_SERVICE_REQUEST`, `SSH_MSG_SERVICE_ACCEPT`

Problems considered: 2

- secrecy of `K, KCS, KSC, secretC %%%, secretS`
- `Client` authenticates `Server` on `k`

Attacks Found

None

Further Notes

Modelling of authentication property:

The common way (see ”standard version” in the HLPSL code) is done by augmenting the Server role with `witness(S,C,n,K')` and the Client-role with `request(C,S,n,K')` where K' is the common (secret!) key. This model yields a spurious attack in which the intruder always forwards the current message. The intruder does not know the common key! Thus, in this attack the intruder plays a dummy role. The attack only results since the intruder is also playing an active role and thus is witness for the final request:

```
Request c s n exp(exp(g,Y(4)),X(3))
Witness s i n exp(exp(g,X(3)),Y(4))
```

To avoid such a dummy attack a different modelling was chosen.

The property is split into two parts. First, assuring that the client has communicated with the server. This is achieved by augmenting the Server role by `witness(S,S,n,K')` and the Client role by `request(S,S,n,K')`. Second, assuring that the common key is only(!) known to the client and the server and not(!) to the intruder. This is achieved by augmenting the Client-role by `secret(K',S)`. Using this modelling no attack results.

The protocol only authenticates the server and not the client. Therefore, messages sent by the server after completion of the protocol may not stay secret.

In the IETF draft (SSH Transport Layer Protocol) it is mentioned that the 'exchange hash SHOULD be kept secret'. This recommendation is violated by the send-operation in the 2nd protocol step in the IETF draft. Here, the 'exchange hash' corresponds to H in role Server and the violation concerns the SND-operation in transition 6 of role Server.

HLPSL Specification

```
role Client (C, S             : agent,
             SND, RCV         : channel(dy),
             Hash             : function,
             HostKey          : function,
             G                : nat,
             LetterC, LetterD : text)
played_by C def=
  local State:    nat,
        NC:       text(fresh),
        NS:       text,
        X:        text(fresh),
        EGY,K:    message, %should be: text
        H,SID_:   message, %should be: text
        KCS, KSC: message, %should be: symmetric_key
        SecretS:  text

  const secretC: text,
        k:       protocol_id

  init   State = 1

% knowledge(C) = {C,S,Hash,HostKey}

  transition

   1. State  = 1 /\ RCV(start) =|>
      State' = 3 /\ SND(NC')

   3. State  = 3 /\ RCV(NS') =|>
      State' = 5 /\ SND(exp(G,X'))
   5. State  = 5 /\ RCV(HostKey(S).EGY'.{H'}_inv(HostKey(S)))
                 /\ H' = Hash(NC.NS.HostKey(S).exp(G,X).EGY'.K')
                 /\ K' = exp(EGY',X) =|>
      State' = 7 /\ SID_' = H'
                 /\ KCS' = Hash(K'.H'.LetterC.SID_')
                 /\ KSC' = Hash(exp(EGY',X).H'.LetterD.H')
                 /\ secret(K',S) /\ secret(KCS',S) /\ secret(KSC',S)
                 /\ SND({secretC}_KCS')
                 /\ secret(secretC,S)
                 %/\ request(C,S,k,K')  % standard version
                 /\ request(S,S,k,K')

   7. State  = 7 /\ RCV({SecretS'}_KSC) =|>
      State' = 9

end role
```

```
role Server (C, S            : agent,
             SND, RCV        : channel(dy),
             Hash            : function,
             HostKey         : function,
             G               : nat,
             LetterC, LetterD : text)
played_by S def=
  local State:    nat,
        NS:       text(fresh),
        NC:       text,
        Y:        text(fresh),
        EGX,K:    message, %should be: text
        H,SID_:   message, %should be: text
        KCS, KSC: message, %should be: symmetric_key
        SecretC:  text

  const k:       protocol_id

  init  State = 2

  knowledge(S) = {inv(HostKey(S))}
  transition
   2. State  = 2 /\ RCV(NC') =|>
      State' = 6 /\ SND(NS')

   6. State  = 6 /\ RCV(EGX') =|>
      State' = 8 /\ K' = exp(EGX',Y')
                 /\ H' = Hash(NC.NS.HostKey(S).EGX'.exp(G,Y').K')
                 /\ SID_' = H'
                 /\ KCS' = Hash(K'.H'.LetterC.SID_')
                 /\ KSC' = Hash(K'.H'.LetterD.SID_')
                 /\ SND(HostKey(S).exp(G,Y').{H'}_inv(HostKey(S)))
                 %/\ witness(S,C,k,K')  % standard version
                 /\ witness(S,S,k,K')

   8. State  = 8 /\ RCV({SecretC'}_KCS) =|>
      State' =10

end role
```

```
role Session(C, S    :   agent,
             CS, SC  : channel (dy),
             Hash    : function,
             HostKey : function,
             G       : nat,
             LetterC, LetterD : text)
def=
  composition
           Client(C, S, CS, SC, Hash, HostKey, G, LetterC, LetterD)
       /\  Server(C, S, SC, CS, Hash, HostKey, G, LetterC, LetterD)
end role
```

```
role Environment() def=

  const
    c,s                 : agent,
    cs,sc,is,si,ci,ic   : channel (dy),
    hash_,host_key      : function,
    g                  : nat,
    letter_c, letter_d : text

  knowledge(i) = {c,s,hash_,host_key,g,letter_c,letter_d}

  composition
        Session(c,s,cs,sc,hash_,host_key,g,letter_c,letter_d)
     /\ Session(i,s,is,si,hash_,host_key,g,letter_c,letter_d)
     /\ Session(c,i,ci,ic,hash_,host_key,g,letter_c,letter_d)

end role
```

```
goal

  secrecy_of K, KCS, KSC, secretC
  Client authenticates Server on k

end goal
```

```
Environment()
```

20 TSP: Time Stamp Protocol

Protocol Purpose

Assertion of proof that a datum existed before a particular time.

Definition Reference

- RFC 3161 : http://www.faqs.org/rfcs/rfc3161.html

Model Authors

- Daniel Plasto for Siemens CT IC 3, 2004

Alice&Bob style

```
C   -> TSA: Hash(Data).NonceC
TSA -> C:   {Hash(Data).Time.NonceC}_inv(PK_TSA)
```

Problems considered: 1

- `TSA` is authenticated on `authdata`

Attacks Found

None

Further Notes

The purpose of this protocol is to assert that a given datum existed before a particular time. For this a trusted time stamping authority (TSA) is used which supplies a unique time stamp. To prove this property the client checks if his datum has really been time stamped by the TSA. This is achieved by the witness/request-pair

```
witness(TSA_,TSA_,authdata,Authdata')
request(TSA_,TSA_,authdata,Authdata')
```

Note that we need not authenticate an agent and therefore use TSA as 1st and 2nd argument in witness/request. Actually, using instead the pair

```
witness(TSA_,C,authdata,Authdata')
request(C,TSA_,authdata,Authdata')
```

will yield an attack where the intruder takes a normal role and simply replays received messages.

```
  i -> (c,3):    start
  (c,3) -> i:    hash_(Data(1)),NonceC(1)
  i -> (tsa,10): hash_(Data(1)),NonceC(1)
  (tsa,10) -> i: {hash_(Data(1)),Time(2),NonceC(1)}inv(pk_tsa)
  i -> (c,3):    {hash_(Data(1)),Time(2),NonceC(1)}inv(pk_tsa)

  Witness tsa i authdata hash_(Data(1)),Time(2)
  Request c tsa authdata hash_(Data(1)),Time(2)
```

This attack does not contradict the intended property since the datum is correctly time-stamped.

HLPSL Specification

```
role Client (
    C,TSA_  : agent,
    Hash    : function,
    PK_TSA  : public_key,
    SND,RCV : channel)
played_by C def=

  local
    State    : nat,
    Data     : text (fresh),
    NonceC   : text (fresh),
    Time     : text

  init
    State = 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State' = 2 /\ SND(Hash(Data').NonceC')

 2. State  = 2 /\ RCV({Hash(Data).Time'.NonceC}_inv(PK_TSA)) =|>
    State' = 4
```

```
               /\ request(TSA_,TSA_,authdata,Hash(Data).Time')

end role
```

```
role TSA (
    C,TSA_      : agent,
    PK_TSA      : public_key,
    SND,RCV     : channel)
played_by TSA_ def=

  local
    State       : nat,
    HashedData : message,
    NonceC      : text,
    Time        : text (fresh)

  init
    State = 1

  transition

 1. State  = 1 /\ RCV(HashedData'.NonceC') =|>
    State' = 3 /\ SND({HashedData'.Time'.NonceC'}_inv(PK_TSA))
               /\ witness(TSA_,TSA_,authdata,HashedData'.Time')

end role
```

```
role Session (
    C,T     : agent,
    Hash    : function,
    PK_TSA  : public_key)
def=

  local
     S1, S2 : channel (dy),
     R1, R2 : channel (dy)
  composition
     Client(C,T,Hash,PK_TSA,S1,R1)
  /\ TSA(   C,T,     PK_TSA,S2,R2)

end role
```

```
role Environment() def=

  const
    c,tsa       : agent,
    hash_       : function,
    pk_tsa      : public_key,
    authdata    : protocol_id

  knowledge(i) = {c,tsa,hash_,pk_tsa}

  composition
     Session(c,tsa,hash_,pk_tsa)
  /\ Session(c,tsa,hash_,pk_tsa)
  /\ Session(i,tsa,hash_,pk_tsa)

end role
```

```
goal

  TSA authenticates TSA on authdata

end goal
```

```
Environment()
```

21 TLS: Transport Layer Security

Protocol Purpose

TLS is intended to provide privacy and data integrity of communication over the Internet.

Definition Reference

- [DA99, Pau99]

Model Authors

- Paul Hankes Drielsma, ETH Zürich, November 2003

Alice&Bob style

The protocol proceeds between a client `A` and a server `B` with respective public keys Ka and Kb. These two agents generate nonces `Na` and `Nb`, respectively. In addition, we assume the existence of a trusted third party (in essence, a certificate authority) `S` whose public key is `Ks`. The agents possess certificates of the form `{X,Kx}inv(Ks)`. Each session is identified by a unique ID `Sid`. The protocol also makes use of a pseudo-random number generator PRF which we model as a hash function.

```
 0. A -> B: A, Na, Sid, Pa         where Pa is a cryptosuite offer
 1. B -> A: Nb, Sid, Pb where Pb is B's counteroffer
 2. B -> A: {B, Kb}inv(Ks) optional certificate exchange
 3. A -> B: {A, Ka}inv(Ks) optional certificate exchange
 4. A -> B: {PMS}Kb where PMS is a nonce generated by A
 5. A -> B: {H(Nb,B,PMS)}inv(Ka) optional certificate verify message
 6. A -> B: {Finished}Keygen(A, Na, Nb, M)
  where  M = PRF(PMS,Na,Nb)
Finished = H(M,messages) for all messages 0 - 5
 7. B -> A: {Finished}Keygen(B, Na, Nb, M)
```

Note that Paulson leaves messages 2., 3., and 5. as optional. We include them in this model. Note also that in order to minimize the number of transitions specified, we have combined the sending of messages 1. and 2. as well as the sending of messages 3. 4. 5. and 6. into single transitions.

Model Limitations

This formalisation is based on the abstracted version of TLS presented by Paulson in [Pau99]. In addition to the abstractions made in this paper, we further abstract away from the negotiation of cryptographic algorithms. Our model assumes that one offer for a crypto suite is made and only that offer will be accepted. This may exclude cipher-suite rollback attacks like the one that was possible on SSLv2.

Problems considered: 3

- secrecy of `ClientK,ServerK`
- `Alice` authenticates `Bob` on `na_nb1`
- `Bob` authenticates `Alice` on `na_nb2`

Attacks Found

None

HLPSL Specification

```
role Alice(A, B : agent,
           H, PRF, KeyGen: function,
           Ka, Ks: public_key,  %% Ks is the public key of a T3P (ie. CA)
           SND, RCV: channel (dy)) played_by A def=

   local Na, Sid, Pa, PMS: text (fresh),
          Nb: text,
          State: nat,
          Finished, ClientK, ServerK: message,
          Kb: public_key,
          M: message

   init State = 0
   knowledge(A) = { inv(Ka), {A.Ka}_(inv(Ks)) }

   transition
   1.  State = 0 /\ RCV(start) =|>
       State'= 2 /\ SND(A.Na'.Sid'.Pa')

   % Since we abstract away from the negotiation
   % of cryptographic algorithms, here I simply assume
   % that the server must send back Pa.  (Essentially
   % modelling that the client makes only one offer.)

   2.   State = 2 /\ RCV(Nb'.Sid.Pa.{B.Kb'}_(inv(Ks))) =|>
        State'= 3 /\
        M' = PRF(PMS'.Na.Nb') /\
        Finished' = H(PRF(PMS'.Na.Nb').A.B.Na.Pa.Sid) /\
        ClientK' = KeyGen(A.Na.Nb'.PRF(PMS'.Na.Nb')) /\
        ServerK' = KeyGen(B.Na.Nb'.PRF(PMS'.Na.Nb')) /\
        SND({PMS'}_Kb'.
            {A.Ka}_(inv(Ks)).
            {H(Nb'.B.PMS')}_(inv(Ka)).
            {H(PRF(PMS'.Na.Nb').
             A.B.Na.Pa.Sid)
            }_KeyGen(A.Na.Nb'.PRF(PMS'.Na.Nb')))
        /\ witness(A,B,na_nb2,Na.Nb')

   4.  State = 3 /\ RCV({Finished}_ServerK) =|> State' = 5
        /\ request(A,B,na_nb1,Na.Nb)
        /\ secret(ClientK,B) /\ secret(ClientK,A)
        /\ secret(ServerK,B) /\ secret(ServerK,A)

end role
```

```
role Bob(A, B : agent,
         H, PRF, KeyGen: function,
         Kb, Ks: public_key,
         SND, RCV: channel (dy)) played_by B def=

   local Nb: text (fresh),
          Na, Sid, Pa, PMS: text,
          State: nat,
          Ka: public_key
   init State = 1
   knowledge(B) = { inv(Kb),{B.Kb}_(inv(Ks)) }

   transition

   1.  State = 1 /\ RCV(A.Na'.Sid'.Pa') =|>
       State'= 3 /\ SND(Nb'.Sid'.Pa'.{B.Kb}_(inv(Ks)))
                 /\ witness(B,A,na_nb1,Na'.Nb')

   2.  State = 3 /\
       RCV({PMS'}_Kb.{A.Ka'}_(inv(Ks)).
           {H(Nb.B.PMS')}_(inv(Ka')).
           {H(PRF(PMS'.Na.Nb).
            A.B.Na.Pa.Sid)
           }_KeyGen(A.Na.Nb.PRF(PMS'.Na.Nb))) =|>
       State'= 5 /\ SND({H(PRF(PMS'.Na.Nb).
                          A.B.Na.Pa.Sid)
                         }_KeyGen(B.Na.Nb.PRF(PMS'.Na.Nb)))
        /\ request(B,A,na_nb2,Na.Nb)

end role
```

```
role Session(A,B: agent,
             Ka, Kb, Ks: public_key,
             H, PRF, KeyGen: function,
             SA, SB, RA, RB: channel (dy)) def=
   composition
                Alice(A,B,H,PRF,KeyGen,Ka,Ks,SA,RA)
           /\   Bob(A,B,H,PRF,KeyGen,Kb,Ks,SB,RB)

end role

role Environment() def=

   local S1,S2,R1,R2,
         S3,S4,R3,R4,
         S5,S6,R5,R6: channel (dy)
   const na_nb1, na_nb2: protocol_id
   knowledge(i) = { a, b, ka, kb, ks, ki, inv(ki) }
   composition
        Session(a,b,ka,kb,ks,h,prf,keygen,S1,S2,R1,R2)
    /\  Session(a,i,ka,ki,ks,h,prf,keygen,S3,S4,R3,R4)
    /\  Session(i,b,ki,kb,ks,h,prf,keygen,S5,S6,R5,R6)

end role

goal

        secrecy_of ClientK,ServerK
        Alice authenticates Bob on na_nb1
        Bob authenticates Alice on na_nb2

end goal
```

```
Environment()
```

Part III
e-Business

22 FairZG

Protocol Purpose

Fair Zhou Gollmann Non-Repudiation

Definition Reference

- http://citeseer.ist.psu.edu/62704.html

Model Authors

- Sebastian Mödersheim, ETH Zürich, December 2003
- Luca Compagna, AI-Lab DIST University of Genova, 2004

Alice&Bob style

```
A   -> B: fNRO,B,L,C,NRO
B   -> A: fNRR,A,L,NRR
A   -> S: fSUB,B,L,K,subK
A <-> S: fCON,A,B,L,K,conK
B <-> S: fCON,A,B,L,K,conK

S    = Trusted Third Party (TTP)
C    = {M}_K
L    = Hash(M,K)
NRO  = {fNRO,B,L,C}_inv(Ka)
NRR  = {fNRR,A,L,C}_inv(Kb)
SubK = {fSUB,B,L,K}_inv(ka)
ConK = {fCON,A,B,L,K}_inv(ks)
```

Model Limitations

The last two exchange of messages between the Server and the agents are ftp-gets. The agents are supposed to search a certificate on the server and they should be able to eventually get the certificate even if the server is temporarily down. The server is supposed to not be down forever.

Problems considered: 5

- `Alice` authenticates `Bob` on `nrr`
- `Bob` authenticates `Alice` on `nro`
- `Server` authenticates `Alice` on `sub`
- `Alice` authenticates `Server` on `con`
- `Bob` authenticates `Server` on `con`

Attacks Found

None

HLPSL Specification

```
role Alice ( A, B, S    : agent,
             Ka, Kb, Ks : public_key,
             K          : symmetric_key,
             Hash       : function,
             Snd, Rcv   : channel(dy)) played_by A def=

  local State : nat,
        M     : text (fresh),
        C     : {text}_symmetric_key,
        L     : message

  const fNRO, fNRR, fSUB, fREQ, fCON : protocol_id

  init   State = 0

  knowledge(A) = {A,B,S,Ka,Kb,Ks,K,inv(Ka),fREQ,fNRO,fNRR,fSUB,fCON}

  transition

   1. State=0 /\ Rcv(start)
```

```
      =|>
         C' = {M'}_K
      /\ L' = Hash(M'.K)
      % The message {fNRO.B.L'.C'}_inv(Ka)
      % represents NRO in Alice&Bob-notation
      /\ Snd(fNRO.B.L'.C'.{fNRO.B.L'.C'}_inv(Ka))
      /\ State'=1

      % Non-repudiation of Origin
      /\ witness(A,B,nro,{fNRO.B.L'.C'}_inv(Ka))

   2. State=1 /\

      % The message {fNRR.A.L.C}_inv(Kb)
      % represents NRR in Alice&Bob-notation
      Rcv(fNRR.A.L.{fNRR.A.L.C}_inv(Kb))
      =|>

      % The message {fSUB.B.L.K}_inv(Ka)
      % represents SubK in Alice&Bob-notation
      Snd(fSUB.B.L.K.{fSUB.B.L.K}_inv(Ka)) /\
      State'=2 /\

      % Non-repudiation of Submission
      witness(A,S,sub,{fSUB.B.L.K}_inv(Ka))

   3. State=2
      --|>
      Snd(fREQ.A.B.L) /\
      State'=3

      % The message {fCON.A.B.L.K}_inv(Ks)
      % represents ConK in Alice&Bob-notation
   4. State=3 /\ Rcv(fCON.A.B.L.K.{fCON.A.B.L.K}_inv(Ks))
      =|>
      State'=4 /\

      % Non-repudiation of Delivery
      request(A,S,con,{fCON.A.B.L.K}_inv(Ks)) /\

      % Non-repudiation of Receipt
      % The message {fNRR.A.L.C}_inv(Kb)
      % represents NRR in Alice&Bob-notation
      request (A,B,nrr,{fNRR.A.L.C}_inv(Kb))

end role
```

```
role Bob (B, A, S    : agent,
          Kb, Ka, Ks : public_key,
          Snd, Rcv   : channel (dy)) played_by B def=

  local State : nat,
        M     : text,
        K     : symmetric_key,
        C     : {text}_symmetric_key,
        L     : message

  const fNRO, fNRR, fSUB, fREQ, fCON : protocol_id

  init State = 0

  knowledge(B) = {B,A,S,Ka,Kb,Ks,inv(Kb),fREQ,fNRO,fNRR,fSUB,fCON}

  transition

   1. State=0 /\

      % The message {fNRO.B.L'.C'}_inv(Ka)
      % represents NRO in Alice&Bob-notation
      Rcv(fNRO.B.L'.C'.{fNRO.B.L'.C'}_inv(Ka))
      =|>

      Snd(fNRR.A.L'.{fNRR.A.L'.C'}_inv(Kb))  /\
      State'=1  /\

      % Non-repudiation of Receipt
      % The message {fNRR.A.L'.C'}_inv(Kb)
      % represents NRR in Alice&Bob-notation
      witness (B,A,nrr,{fNRR.A.L'.C'}_inv(Kb))
   2. State=1
      --|>
      Snd(fREQ.A.B.L) /\
      State'=2

   3. State=2 /\

      % The message {fCON.A.B.L.K'}_inv(Ks)
      % represents ConK in Alice&Bob-notation
      Rcv(fCON.A.B.L.K'.{fCON.A.B.L.K'}_inv(Ks)) /\
      C = {M'}_K'
      =|>
      State'=3 /\

      % Non-repudiation of Delivery
      request(B,S,con,{fCON.A.B.L.K'}_inv(Ks)) /\

      % Non-repudiation of Origin
      % The message {fNRO.B.L.C}_inv(Ka)
      % represents NRO in Alice&Bob-notation
      request(B,A,nro,{fNRO.B.L.C}_inv(Ka))

end role
```

```
role Server (S        : agent,
             Ks, Ka   : public_key,
             Snd, Rcv : channel (dy)) played_by S def=

  local State : nat,
        A, B  : agent,
        L     : message,
        K     : symmetric_key

  const fNRO, fNRR, fSUB, fREQ, fCON : protocol_id

  init State = 0

  knowledge(S) = {S,Ks,Ka,inv(Ks),fREQ,fNRO,fNRR,fSUB,fCON}
  transition

   1. State=0 /\

      % The message {fSUB.B'.L'.K'}_inv(Ka)
      % represents SubK in Alice&Bob-notation
      Rcv(fSUB.B'.L'.K'.{fSUB.B'.L'.K'}_inv(Ka))
      =|>
      State'=1

   2. State=1 /\
      Rcv(fREQ.A'.B.L)
      =|>

      % The message {fCON.A.B.L.K}_inv(Ks)
      % represents ConK in Alice&Bob-notation
      Snd(fCON.A'.B.L.K.{fCON.A'.B.L.K}_inv(Ks)) /\
      State'=2 /\

      % Non-repudiation of Delivery
      witness (S,A',con,{fCON.A'.B.L.K}_inv(Ks)) /\
      witness (S,B,con,{fCON.A'.B.L.K}_inv(Ks)) /\

      % Non-repudiation of Submission
      request (S,A',sub,{fSUB.B.L.K}_inv(Ka))

end role
```

```
role Session(A, B, S                : agent,
             Ka, Kb, Ks             : public_key,
             K                      : symmetric_key,
             H                      : function,
             SA, RA, SB, RB, SS, RS : channel (dy)) def=

  composition
    Alice(A,B,S,Ka,Kb,Ks,K,H,SA,RA) /\
    Bob(B,A,S,Kb,Ka,Ks,SB,RB) /\
    Server(S,Ks,Ka,SS,RS)
```

```
end role
```

```
role Environment() def=

 local Sa1,Ra1,Sb1,Rb1,Ss1,Rs1,
       Sa2,Ra2,Si2,Ri2,Ss2,Rs2,
       Si3,Ri3,Sb3,Rb3,Ss3,Rs3 : channel (dy)

 const nro,nrr,con,sub: protocol_id,
       fREQ,fNRO,fNRR,fSUB,fCON: protocol_id

 knowledge(i) = {a,b,s,ka,kb,ks,ki,i,inv(ki),fNRO,fNRR,fSUB,fCON}

 composition

       Session(a,b,s,ka,kb,ks,k,h,Sa1,Ra1,Sb1,Rb1,Ss1,Rs1)
    /\ Session(a,i,s,ka,ki,ks,k,h,Sa2,Ra2,Si2,Ri2,Ss2,Rs2)
    /\ Session(i,b,s,ki,kb,ks,k,h,Si3,Ri3,Sb3,Rb3,Ss3,Rs3)
%   /\ Session(a,b,i,ka,kb,ki,k,h,Sa4,Ra4,Sb4,Rb4,Si4,Ri4)

end role
```

```
goal

   Alice authenticates Bob on nrr
   Bob authenticates Alice on nro
   Server authenticates Alice on sub
   Alice authenticates Server on con
   Bob authenticates Server on con

end goal
```

```
Environment()
```

Part IV
Non IETF Protocols

23 UMTS-AKA

Protocol Purpose

Authentication and Key Agreement

Definition Reference

http://www.3gpp.org/ftp/tsg_sa/WG3_Security/_Specs/33902-310.pdf

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003
- Sebastian Mödersheim, ETH Zürich, December 2003

Alice&Bob style

S is the server, M is the mobile set, they share a secret key k(M).

Both S and M have an own version of a sequence number, that they try to maintain synchonized.

Using k(M), a random number (nonce) r, his sequence number seq, when S receives a request from M (or whenever he wishes this part is not modelled here), S generates:

```
  res = F2(k(M); r)   where F2 hash
  CK =  F3(k(M); r)   where F3 one-way
  IK =  F4(k(M); r)   where F4 one-way
  Ka =  F5(k(M); r)   where F5 one-way
  AUTN = {seq}Ka; F1(k(M); seq; r)  where F1 hash

 M -> S : M
 S -> M : r; {seq}_Ka; F1(k(M); seq; r)

     from r M calculates KA, then seq, then checks if F1(k(M); seq; r) OK
     if yes, M increments his seq number and responds:

 M -> S : F2(k(M); r)
```

The goal is that at the end both authenticate each other and share the value of `CK` and `IK`.

Problems considered: 3

- secrecy of `Seq` `% the shared key`
- `Mobile` weakly authenticates `Server` on `r1` `% the nonce R`
- `Server` weakly authenticates `Mobile` on `r2` `% the nonce R`

Attacks Found

None

HLPSL Specification

```
role Server(S,M : agent,
            Snd, Rec: channel(dy),
            K_M: symmetric_key,
            Seq : text,
            F1,F2,F5: function) played_by S def=

  local State : nat,
        R     : text(fresh)

  init State=1
  knowledge(S) = {S,M,K_M,Seq,F1,F2,F5}

  transition

    1.  State = 1 /\ Rec(M) =|>
        State'= 2 /\ Snd(R'.{Seq}_F5(K_M.R').F1(K_M.Seq.R'))
                  /\ secret(Seq,M) /\ secret(Seq,S)
                  /\ witness(S,M,r1,R')

    2.  State = 2 /\ Rec(F2(K_M.R)) =|>
        State'= 3 /\ Seq' = add(Seq,1)
                  /\ wrequest(S,M,r2,R)

end role
```

```
role Mobile(M,S:agent,
            Snd, Rec: channel(dy),
            K_M: symmetric_key,
            Seq: text,
            F1,F2,F5: function) played_by M def=

  local State :nat,
        R     :text

  init State=1
  knowledge(M) = {M,S,K_M,Seq,F1,F2,F5}

  transition

    1.  State = 1 /\ Rec(start) =|>
        State'= 2 /\ Snd(M)

    2.  State = 2 /\ Rec(R'.{Seq}_F5(K_M.R').F1(K_M.Seq.R')) =|>
        State'= 3 /\ Snd(F2(K_M. R'))
                  /\ secret(Seq,S) /\ secret(Seq,M)
                  /\ wrequest(M,S,r1,R')
                  /\ witness(M,S,r2,R')

end role
```

```
role Session(M,S: agent,
             K_M: symmetric_key,
             Seq: text,
             F1,F2,F5: function,
             SA,RA,SB,RB: channel(dy)) def=

   composition

        Mobile(M,S,SA,RA,K_M,Seq,F1,F2,F5)
     /\ Server(S,M,SB,RB,K_M,Seq,F1,F2,F5)

end role
```

```
role Environment() def=

 local Sa1,Ra1,Ss1,Rs1 : channel (dy)

 const r1, r2                 : protocol_id,
       a, i, s                : agent,
       f1, f2, f5             : function,
       seq_as, seq_is, seq_ai : text

 knowledge(i)={a,s,i,f1,f2,f5}

 composition

        Session(a,s,k_as,seq_as,f1,f2,f5,Sa1,Ra1,Ss1,Rs1)
% /\    Session(i,s,k_is,seq_is,f1,f2,f5,si1,ri1,ss2,rs2)
% /\    Session(a,i,k_ai,seq_ai,f1,f2,f5,sa2,ra2,si2,ri2)

end role
```

```
goal

  secrecy_of Seq  % the shared key
  Mobile weakly authenticates Server on r1  % the nonce R
  Server weakly authenticates Mobile on r2  % the nonce R

end goal
```

```
Environment()
```

24 ISO1 Public Key Unilateral Authentication Protocol

24.1 one-pass unilateral authentication

Protocol Purpose

A client authenticates himself to a server by sending a digital signature.

Definition Reference

- [CJ, ISO97]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003 and
- Luca Compagna et al, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
1. A -> B : {PKa,A}inv(PKs), Na, B, Text,{Na,B,Text}inv(PKa)
```

Problems considered: 1

- `ISO1_Resp` authenticates `ISO1_Init` on `na`

Attacks Found

The intruder can attack this protocol by simple eavesdropping and replaying the digital signatures.

```
i     -> (a,6) : start
(a,6) -> i     : pka,a,{pka,a}inv(pks),na(a,6),b,ctext,
                 {na(a,6),b,ctext}inv(pka)
i     -> (b,4) : pka,a,{pka,a}inv(pks),na(a,6),b,ctext,
                 {na(a,6),b,ctext}inv(pka)
i     -> (b,7) : pka,a,{pka,a}inv(pks),na(a,6),b,ctext,
                 {na(a,6),b,ctext}inv(pka)
```

Further Notes

`inv(PKs)` is the private key of the server `C`; `{PKa,A}inv(PKs)` is the certificate of agent `A`.

If one would like to use the same server public key for each session, then permutation on `Pks` should be avoided.

HLPSL Specification

```
role ISO1_Init ( A,B : agent,
                 Pka, Pks : public_key,
                 Snd, Rcv : channel(dy)) played_by A def=

 local  State: nat,
        Na   : text(fresh)

 init  State = 0

 knowledge(A) = {A,B,Pka,Pks,ctext,inv(Pka)}

 transition

   1. State=0 /\ Rcv(start) =|>
         Snd(Pka.A.{Pka.A}_inv(Pks).Na'.B.ctext.{Na'.B.ctext}_inv(Pka))
      /\ State'=1
      /\ witness(A,B,na,Na')

end role
```

```
role ISO1_Resp (A, B: agent,
                Pks : public_key,
                Rec : channel(dy)) played_by B def=

  local  State     : nat,
         Pka       : public_key,
         Na, Text  : text

  init  State=0

  knowledge(B)={A,B,Pks}

  transition

   1.    State=0
      /\ Rec(Pka'.A.{Pka'.A}_inv(Pks).Na'.B.Text'.{Na'.B.Text'}_inv(Pka')) =|>
         State'= 1
      /\ request(B,A,na,Na')

end role
```

```
role Session (A, B : agent,
              Pka  : public_key,
              Pks  : public_key) def=

  local SA, RA, RB: channel (dy)

  const na : protocol_id

  composition

          ISO1_Init(A,B,Pka,Pks,SA,RA)
       /\ ISO1_Resp(A,B,Pks,RB)

end role
```

```
role Environment() def=

  const ctext    : text,
        a, b     : agent,
        pka, pks : public_key

  knowledge(i)={a,b,pks,pka}
  composition

        Session(a,b,pka,pks)
     /\ Session(a,b,pka,pks)

end role
```

```
goal

   ISO1_Resp authenticates ISO1_Init on na

end goal
```

```
Environment()
```

24.2 two-Pass unilateral authentication

Protocol Purpose

Authentication of a client to a server. This protocol models a situation in which the server wants to verify the client identity and starts the session. The client answers by sending his digital signature.

Definition Reference

- [CJ, ISO97]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003 and
- Luca Compagna et al, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
1. B -> A : Rb, Text1
2. A -> B : {PKa,A}inv(PKs), Ra,Rb, B, Text2,{Ra,Rb,B,Text1}inv(PKa)
```

Problems considered: 1

- `ISO2_Init` authenticates `ISO2_Resp` on `ra`

Attacks Found

None

Further Notes

`inv(PKs)` is the private key of the server `C`; `{PKa,A}inv(PKs)` is the certificate of agent `A`.

HLPSL Specification

```
role ISO2_Init (B,A    : agent,
                Pks    : public_key,
                Snd,Rec: channel(dy)) played_by B def=

  local  State     : nat,
         Pka       : public_key,
         Rb        : text(fresh),
         Ra, Text2 : text

  init State = 0

  knowledge(B)={A,B,ctext1,Pks}

  transition

   1. State=0 /\ Rec(start) =|> Snd(Rb'.ctext1) /\ State'=1
   2.    State=1
      /\ Rec(Pka'.A.{Pka'.A}_inv(Pks).Ra'.Rb.B.Text2'.
                    {Ra'.Rb.B.ctext1}_inv(Pka'))
      =|>
         State'=2
      /\ request(B,A,ra,Ra')

end role
```

```
role ISO2_Resp (A,B    : agent,
                Pka,Pks: public_key,
                Snd,Rec: channel(dy)) played_by A def=

  local  State     : nat,
         Ra        : text(fresh),
         Rb, Text1 : text

  init State = 0

  knowledge(A)={A,B,Pka,Pks,ctext2,inv(Pka),{Pka.A}_inv(Pks)}

  transition

   1. State=0 /\ Rec(Rb'.Text1') =|>
         Snd(Pka.A.{Pka.A}_inv(Pks).Ra'.Rb'.B.ctext2.
                   {Ra'.Rb'.B.Text1'}_inv(Pka))
      /\ State'=2
      /\ witness(A,B,ra,Ra')

end role
```

```
role Session (B, A : agent,
              Pka  : public_key,
              Pks  : public_key) def=

  local SA, RA, SB, RB: channel (dy)
  composition
```

```
          ISO2_Init(B,A,Pks,SB,RB)
       /\ ISO2_Resp(A,B,Pka,Pks,SA,RA)

end role
```

```
role Environment() def=

  const  ctext1,ctext2 : text,
         ra            : protocol_id,
         a,b,i         : agent,
         pks,pki       : public_key

  knowledge(i)={i,a,b,pks,pki,inv(pki),ctext1,ctext2}

  composition

        Session(a,b,pkb,pks)
     /\ Session(a,i,pki,pks)
     /\ Session(i,b,pkb,pks)

end role
```

```
goal

   ISO2_Init  authenticates ISO2_Resp on ra

end goal
```

```
Environment()
```

24.3 two-pass mutual authentication

Protocol Purpose

Two parties authenticate each other. Aim of the Mutual authentication is to make sure to each of the parties of the other's identity. In this protocol authentication should be achieved by a single encrypted message sent from each party.

Definition Reference

- [CJ, ISO97]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003 and
- Luca Compagna et al, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
1. A -> B : PKa,A,{PKa,A}inv(PKs), Na, B, Text2,{Na,B,Text1}inv(PKa)
2. B -> A : PKb,B,{PKb,B}inv(PKs), Nb, A, Text4,{Nb,A,Text3}inv(PKb)
```

- `inv(PKs)` is the private key of the server `C`
- `{PKa,A}inv(PKs)` is the certificate of agent `A`
- `{PKb,B}inv(PKs)` is the certificate of agent `B`

Problems considered: 2

- `ISO3_Init` weakly authenticates `ISO3_Resp` on `nb`
- `ISO3_Resp` weakly authenticates `ISO3_Init` on `na`

Attacks Found

The intruder can attack this protocol by simple eavesdropping and replaying the messages.

```
 i     -> (a,6) : start
 (a,6) -> i     : pka,a,{pka,a}inv(pks),na(a,6),b,ctext2,
                  {na(a,6),b,ctext1}inv(pka)
 i     -> (b,9) : start
 (b,9) -> i     : pkb,b,{pkb,b}inv(pks),na(b,9),a,ctext2,
                  {na(b,9),a,ctext1}inv(pkb)
 i     -> (a,6) : pkb,b,{pkb,b}inv(pks),na(b,9),a,ctext2,
                  {na(b,9),a,ctext1}inv(pkb)
```

Further Notes

HLPSL Specification

```
role ISO3_Init( A, B     : agent,
                Pka, Pks : public_key,
                Snd, Rcv : channel(dy)) played_by A def=

  local  State             : nat,
         Na                : text(fresh),
         Nb, Text3, Text4  : text,
         Pkb               : public_key

  init State = 0

  knowledge(A)={A,B,Pka,Pks,ctext1,ctext2}

  transition

   1.    State=0 /\ Rcv(start)
      =|>
         Snd(Pka.A.{Pka.A}_inv(Pks).Na'.B.ctext2.{Na'.B.ctext1}_inv(Pka))
      /\ State'=1
      /\ witness(A,B,na,Na')

   2.    State=1
      /\ Rcv(Pkb'.B.{Pkb'.B}_inv(Pks).Nb'.A.Text4'.{Nb'.A.Text3'}_inv(Pkb'))
```

```
    =|>
       State'=2
    /\ wrequest(A,B,nb,Nb')

end role
```

```
role ISO3_Resp (B, A     : agent,
                Pkb, Pks : public_key,
                Snd, Rcv : channel(dy)) played_by B def=

   local  State          : nat,
          Nb             : text(fresh),
          Na,Text1,Text2 : text,
          Pka            : public_key

   init State = 0

   knowledge(B)={A,B,Pkb,Pks,ctext3,ctext4}

   transition

   1.    State=0
      /\ Rcv(Pka'.A.{Pka'.A}_inv(Pks).Na'.B.Text2'.{Na'.B.Text1'}_inv(Pka'))
      =|>
         Snd(Pkb.B.{Pkb.B}_inv(Pks).Nb'.A.ctext4.{Nb'.A.ctext3}_inv(Pkb))
      /\ State'=1
      /\ witness(B,A,nb,Nb')
      /\ wrequest(B,A,na,Na')

end role
```

```
role Session (A, B     : agent,
              Pka, Pkb : public_key,
              Pks      : public_key) def=

  local SA, RA, SB, RB: channel (dy)
  composition

          ISO3_Init(A,B,Pka,Pks,SA,RA)
       /\ ISO3_Resp(B,A,Pkb,Pks,SB,RB)

end role
```

```
role Environment() def=

  const ctext1, ctext2, ctext3, ctext4 : text,
        na, nb                         : protocol_id,
        a, b                           : agent,
        pka, pkb, pks, pki             : public_key

  knowledge(i)={a,b,pks,pki,inv(pki)}

  composition

        Session(a,b,pka,pkb,pks)
     /\ Session(a,b,pka,pkb,pks)
     /\ Session(b,a,pkb,pka,pks)

end role
```

```
goal

   ISO3_Init weakly authenticates ISO3_Resp on nb

   ISO3_Resp weakly authenticates ISO3_Init on na

end goal
```

```
Environment()
```

24.4 three-pass mutual authentication

Protocol Purpose

Two parties authenticate each other. Aim of the Mutual authentication is to make sure to each of the parties of the other's identity. In this protocol a confirmation of the successful authentication is sent by the initiator.

Definition Reference

- [CJ, ISO97]

Model Authors

- Haykal Tej, Siemens CT IC 3, 2003 and
- Luca Compagna et al, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
 1. B -> A : Nb, Text1
 2. A -> B : PKa,A,{PKa,A}inv(PKs),Na,Nb,B,Text3,{Na,Nb,B,Text2}inv(PKa)
 3. B -> A : PKb,B,{PKb,B}inv(PKs),Nb,Na,A,Text5,{Nb,Na,A,Text4}inv(PKb)
```

Problems considered: 2

- `ISO4_Resp` authenticates `ISO4_Init` on `nb`
- `ISO4_Init` authenticates `ISO4_Resp` on `na`

Attacks Found

None

Further Notes

`inv(PKs)` is the private key of the server `C`; `{PKa,A}inv(PKs)` is the certificate of agent `A`, and `{PKb,B}inv(PKs)` is the certificate of agent `B`.

HLPSL Specification

```
role ISO4_Init ( A,B: agent,
                 Pkb,Pks: public_key,
                 Snd,Rec: channel(dy)) played_by B def=

  local  State         : nat,
         Pka           : public_key,
         Nb            : text(fresh),
         Na,Text2,Text3: text

  const ctext1,ctext4,ctext5: text

  init State = 0

  knowledge(B)={A,B,Pkb,inv(Pkb),Pks,ctext1,ctext4,ctext5,{Pkb.B}_inv(Pks)}

  transition

   1. State=0 /\ Rec(start)
        =|>
      Snd(Nb'.ctext1)
        /\ State'=1
        /\ witness(B,A,nb,Nb')

   2. State=1 /\ Rec(Pka'.A.{Pka'.A}_inv(Pks).Na'.Nb.B.Text3'.
                     {Na'.Nb.B.Text2'}_inv(Pka'))
      =|>
      Snd(Pkb.B.{Pkb.B}_inv(Pks).Nb.Na'.A.ctext5.{Nb.Na'.A.ctext4}_inv(Pkb))
        /\ State'=2
        /\ request(B,A,na,Na')
end role
```

```
role ISO4_Resp ( B,A: agent,
                 Pka,Pks: public_key,
                 Snd,Rec: channel(dy)) played_by A def=

  local  State              : nat,
         Pkb                : public_key,
         Na                 : text(fresh),
         Nb,Text1,Text4,Text5: text

  const ctext2,ctext3: text

  init State = 0

  knowledge(A)={A,B,Pka,inv(Pka),Pks,ctext2,ctext3,{Pka.A}_inv(Pks)}

  transition

   1. State=0 /\ Rec(Nb'.Text1')
              =|> Snd(Pka.A.{Pka.A}_inv(Pks).
                      Na'.Nb'.B.ctext3.{Na'.Nb'.B.ctext2}_inv(Pka))
               /\ State'=1
               /\ witness(A,B,na,Na')

   2. State=1
        /\ Rec(Pkb'.B.{Pkb'.B}_inv(Pks).
                     Nb.Na.A.Text5'.{Nb.Na.A.Text4'}_inv(Pkb'))
        =|>
        State'=2
        /\ request(A,B,nb,Nb)

end role
```

```
role Session (A,B:agent,
              Pka,Pkb,Pks: public_key) def=

  local SA,RA,SB,RB: channel (dy)

  composition

          ISO4_Init(A,B,Pkb,Pks,SA,RA)
       /\ ISO4_Resp(B,A,Pka,Pks,SB,RB)
```

```
end role
```

```
role Environment() def=

 const na, nb             : protocol_id,
       a, b, i            : agent,
       pka, pkb, pks, pki : public_key

 knowledge(i)={a,b,pki,inv(pki),pks}

 composition

        Session(a,b,pka,pkb,pks)
     /\ Session(a,i,pka,pki,pks)
     /\ Session(i,b,pki,pkb,pks)

end role
```

```
goal

   ISO4_Resp authenticates ISO4_Init on nb

   ISO4_Init authenticates ISO4_Resp on na

end goal
```

```
Environment()
```

25 2pRSA: Two-Party RSA Signature Scheme

Protocol Purpose

Secure signing protocol by including a trusted server as second party in the signing process

Definition Reference

- http://www-cse.ucsd.edu/users/mihir/papers/splitkey.html

Model Authors

- Peter Warkentin, Siemens CT IC 3, December 2004

Alice&Bob style

```
0. BC -> S:  M.SM   with SM = {M}_inv(kc)
                    where S checks if BC has signed, i.e. {SM}_Kbc = M
1. S  -> BC: SSM    with SSM = {SM}_inv(ks)
2. BC -> C:  M.SSM  where C checks if S  has signed, i.e. {{SSM}_Ks}_Kbc = M
```

Model Limitations

Issues abstracted from:

- General public/private keys instead of RSA exponentiation
- Only MCS,HCS (client starts signing process)

Currently, algebraic equations involving exponentiation exp and its inverse, inv, cannot be handled. Therefore we use general public/private keys.

Problems considered: 1

- `Consumer` weakly authenticates `Server` on `m`

Attacks Found

None

Further Notes

The protocol uses the RSA-based signature scheme for signing a message by including a 3rd trusted party (Server) in the signing process. The RSA algorithm defines a modulus N and two exponents e,d such that `m^(ed) = m modulo EulerFct(N)`. Here, e is the publicly known encryption exponent and d the corresponding secret decryption exponent. The signature of a message m is obtained by computing `m^d`. The basic idea now is to split d into dc,ds with `dc*ds = d modulo EulerFct(N)` and to give ds to the server and dc to the client. For computing a signature the client first signs with his part of d yielding `m^dc` and thereafter the server signs the result with ds yielding `(m^dc)^ds = m^d`. Of course, the signing may also be performed the other way round: first server then client. Any agent who knows e can check the signature by computing `signature^e` and by checking if the result is the original message.

The original property is as follows: The (trusted) server S has taken part in all complete signatures which the (possibly) bad client BC can produce. We model the bad client BC as a normal (good) client. Additionally, we define a consumer C to whom BC sends the original message M together with the final signature SSM. The intruder may intercept and modify this last message (and thus play the 'bad' part of BC). The consumer checks if the signature really originated from the server S.

HLPSL Specification

```
role BClient (C,BC,S:   agent,
              Kbc,Ks:   public_key,
              H:        function,
              SND,RCV:  channel(dy))
played_by BC def=

  local State: nat,
        M0:    text (fresh),
        M,SSM: message

  const m:     protocol_id

  init State = 0

  knowledge(BC) = {C,BC,S,Kbc,Ks,H,SND,RCV}
  transition
    1.      State =0
            /\ RCV(start)
        =|>
            State'=1
            /\ M' = H(M0')                    % using   hashed message
            %/\ M' = M0'                      % using unhashed message
            /\ SND(M'.{M'}_inv(Kbc))

    2.      State =1
            /\ RCV(SSM')
            /\ SSM' = {{M}_inv(Kbc)}_inv(Ks)
        =|>
            State'=2
            /\ SND(M.SSM')

end role
```

```
role Consumer(C,BC,S:   agent,
              Kbc,Ks:   public_key,
              H:        function,
              SND,RCV:  channel(dy))
played_by C def=

  local State: nat,
        M,SSM: message

  const m:     protocol_id

  init State = 0

  knowledge(C) = {C,BC,S,Kbc,Ks,H,SND,RCV}

  transition
    1.      State =0
            /\ RCV(M'.SSM')
            /\ SSM' = {{M'}_inv(Kbc)}_inv(Ks)
        =|>
            State'=1
            /\ wrequest(C,S,m,M')

end role
```

```
role Server (C,BC,S:  agent,
             Kbc,Ks:  public_key,
             H:       function,
             SND,RCV: channel(dy))
played_by S def=

  local State: nat,
        M,SM:  message

  const m:     protocol_id

  init State=0

  knowledge(S) = {BC,S,Ks,Kbc,H,SND,RCV}

  transition

    1.     State =0
           /\ RCV(M'.SM')
           /\ SM' = {M'}_inv(Kbc)
       =|>
           State'=1
           /\ SND({SM'}_inv(Ks))
           /\ witness(S,C,m,M')

end role
```

```
role Session(C,BC,S: agent,
             Kbc,Ks: public_key,
             H:      function,
             CS,SC:  channel (dy))
def=
  composition
```

```
           BClient( C, BC, S, Kbc,Ks, H, CS, SC)
        /\ Consumer(C, BC, S, Kbc,Ks, H, CS, SC)
        /\ Server(  C, BC, S, Kbc,Ks, H, SC, CS)
end role
```

```
role Environment() def=

  local
        S1,S2,S3   : channel(dy),
        R1,R2,R3   : channel(dy)

  const c,bc,s     : agent,
        kbc,ks,ki  : public_key,
        h          : function

  knowledge(i) = {c,bc,s,h,kbc,ks,ki,inv(ki)}

  composition
     Session(c,bc,s,kbc,ks,h,S1,R1)
  /\ Session(c,bc,s,kbc,ks,h,S2,R2)
  /\ Session(c,i, s,ki, ks,h,S3,R3)

end role
```

```
goal
  Consumer weakly authenticates Server on m
end goal
```

```
Environment()
```

26 LPD: Low-Powered Devices

26.1 MSR: Modulo Square Root

LPD (Low-Powered Devices) MSR (Modulo Square Root) protocol is a key establishment protocol for secure mobile communications. It has been designed by Beller, Chang, and Yacobi in 1990s. Such a protocol relies on a public key cryptosystem for which encryption is particularly efficient, at least in comparison to other public key cryptosystems. The specific public key cryptosystem employed is due to Rabin, in which encryption and decryption tantamount, respectively, to modulo squaring and extracting a modulo square root (MSR). MSR technique allows public key encryption to be implemented within the computational power of a mobile station.

Protocol Purpose

Key establishment protocol for secure mobile communications.

Definition Reference

- [BM98, page 4]

Model Authors

- Graham Steel, University of Edinburgh, July 2004
- Luca Compagna, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
B, M : agent
PKb  : public key
SCm  : text
X    : symmetric key (fresh)

1. B -> M : B, PKb
2. M -> B : {x}PKb
3. M -> B : {M, SCm}x
```

The object `SCm` denotes the secret certificate of the mobile M which is issued by a trusted central authority.

Upon receiving `B`'s public key `PKb`, the mobile uses it to encrypt the session key `X`, and sends the encrypted message to `B`. The mobile also sends its identity and secret certificate encrypted under `X` to authenticate `X` to the base. The encryption in message 3 is carried out using a symmetric key cryptosystem. Since this encryption is negligible compared to the public key encryption in message 2, the computational effort at the mobile is effectively reduced to that of modulo squaring of the session key.

Model Limitations

The protocol would require the mobile `M` to send two sequential messages to the base station `B` in a row. We model such a situation by sending in one single transition the pair of the two messages.

Problems considered: 2

- secrecy of `X`
- `MSR_Base` weakly authenticates `MSR_Mobile` on `x`

Attacks Found

The public key of `B` is uncertified, thereby allowing anyone to masquerade as `B` (perceived as a serious threat in the emerging standards). Moreover replay of an old compromised session key allows masquerade of `M`. As a matter of fact, the following attack trace:

```
i       -> (b,3) : start
(b,3)   -> i : b,kb
i       -> (m,4) : b,ki
(m,4)   -> i : {x0(m,4)}ki,{m,scm1}x0(m,4)
```

suffices (i) to violate the secrecy of the established session key `X` and (ii) to make the base station `B` to believe talking with the mobile `M` while it is talking with the intruder.

HLPSL Specification

```
role MSR_Base(B, M      : agent,
              PKb       : public_key,
              SCm       : text,
              Snd, Rcv : channel(dy)) played_by B def=

  local  State : nat,
         X     : symmetric_key

  init   State = 0
  accept State = 2
  knowledge(B) = {B,M,PKb,inv(PKb),SCm}

  transition

   1. State=0 /\ Rcv(start) =|>
         Snd(B.PKb)
      /\ State'=1

   2. State=1 /\ Rcv({X'}_PKb.{M.SCm}_X') =|>
         State'=2
      /\ wrequest(B,M,x,X')

end role
```

```
role MSR_Mobile(B, M     : agent,
                SCm      :   text,
                Snd, Rcv : channel (dy)) played_by M def=

  local State : nat,
        PKb   : public_key,
        X     : symmetric_key (fresh)

  init   State = 0
  accept State = 1
  knowledge(M) = {B,M,SCm}

  transition

   1. State=0 /\ Rcv(B.PKb') =|>
         Snd({X'}_PKb'.{M.SCm}_X')
      /\ State'=1
      /\ witness(M,B,x,X')
      /\ secret(X',B)
      /\ secret(X',M)

end role
```

```
role Session(B, M           : agent,
             PKb            : public_key,
             SCm            : text) def=

  local  SA, RA, SB, RB : channel (dy)

  const  x : protocol_id

  composition

           MSR_Base(B,M,PKb,SCm,SA,RA)
        /\ MSR_Mobile(B,M,SCm,SB,RB)

end role
```

```
role Environment() def=

 const b,m                               : agent,
       kb, ki                            : public_key,
       scm1,scm2,scm3                    : text

 knowledge(i) = {b,m,scm2,scm3,i,ki,inv(ki)}

composition

        Session(b,m,kb,scm1)
    /\  Session(b,i,kb,scm2)
    /\  Session(i,m,ki,scm3)

end role
```

```
goal

% The established key X must be a secret between the base and the mobile
  secrecy_of X

% Authentication: base station authenticates mobile
  MSR_Base weakly authenticates MSR_Mobile on x

end goal
```

```
Environment()
```

26.2 IMSR: Improved Modulo Square Root

LPD (Low-Powered Devices) Improved MSR (Modulo Square Root) protocol is a key establishment protocol for secure mobile communications. It has been designed by Beller, Chang, and Yacobi in 1990s as an improvement of MSR. Namely IMSR overcomes a major weakness of MSR by including a certificate of the base station in the first message. Apart from this feature it is identical to the basic MSR protocol, and therefore does not address the problem of replay

Protocol Purpose

Key establishment protocol for secure mobile communications.

Definition Reference

- [BM98, pages 5-6]

Model Authors

- Graham Steel, University of Edinburgh, July 2004
- Luca Compagna, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
 B, M    : agent
```

```
PKb     : public key
SCm     : text
Nb      : text (fresh)
Cert(B) : message
X       : symmetric key (fresh)

1. B -> M : B, Nb, PKb, Cert(B)
2. M -> B : {X}PKb
3. M -> B : {Nb, M, SCm}X
```

The object `SCm` denotes the secret certificate of the mobile `M` which is issued by a trusted central authority. `Cert(B)` is the public certificate previously issued by some server for `B`. We assume `Cert(B) = {B.PKb}inv(PKs)`.

Notice that wrt MSR there is a twofold increase in the complexity of this protocol as compared to the basic MSR protocol. The mobile now calculates an additional modulo square to verify the base's certificate on receiving message 1. Upon receiving the final message, `B` decrypts it using the session key `X`, and checks that the value `Nb` is the same as the random challenge sent in message 1.

Model Limitations

The protocol would require the mobile `M` to send two sequential messages to the base station `B` in a row. We model such a situation by sending in one single transition the pair of the two messages.

Problems considered: 2

- secrecy of `X`
- `IMSR_Base` weakly authenticates `IMSR_Mobile` on `x`

Attacks Found

None

Further Notes

The added public certificate and nonce exchange give some more protection. Boyd et al. [BM98] recommend moving the nonce and `M` into message 2.

HLPSL Specification

```
role IMSR_Base(B, M     : agent,
               SCm      : text,
               PKb      : public_key,
               PKs      : public_key,
               Snd, Rcv : channel (dy)) played_by B def=

  local State   : nat,
        PBk     : public_key (fresh),
        X       : symmetric_key,
        Nb      : text (fresh),
        Package : message

  init   State = 0
  accept State = 2
  knowledge(B) = {B,M,PKb,inv(PKb),SCm,PKs,{B.PKb}_inv(PKs)}

  transition

   1. State=0 /\ Rcv(start) =|>
         Snd(B.Nb'.PKb.{B.PKb}_inv(PKs))
      /\ State'=1

   2. State=1 /\ Rcv({X'}_PKb.{Nb.M.SCm}_X') =|>
         State'=2
      /\ wrequest(B,M,x,X')

end role
```

```
role IMSR_Mobile(B, M     : agent,
                 SCm      : text,
                 PKs      : public_key,
                 Snd, Rcv : channel (dy)) played_by M def=
  local State: nat,
        PKb  : public_key,
        X    : symmetric_key (fresh),
        Nb   : text,
        Cert : message

  init   State = 0
  accept State = 1
  knowledge(M) = {B,M,SCm,PKs}

  transition

   1. State=0 /\ Rcv(B.Nb'.PKb'.Cert') /\ Cert' = {B.PKb'}_inv(PKs)
      =|>
         Snd({X'}_PKb'.{Nb'.M.SCm}_X')
      /\ State'=1
      /\ secret(X',B)
      /\ secret(X',M)
      /\ witness(M,B,x,X')

end role
```

```
role Session(B, M           : agent,
             SCm            : text,
             PKb, PKs       : public_key) def=

  local SA, RA, SB, RB : channel (dy)

  const  x : protocol_id

  composition

        IMSR_Base(B,M,SCm,PKb,PKs,SA,RA)
     /\ IMSR_Mobile(B,M,SCm,PKs,SB,RB)

end role
```

```
role Environment() def=

  const b, m                                      : agent,
        kb, ki, ks                                : public_key,
        scm1, scm2, scm3                          : text

  knowledge(i) = {b,s,scm2,scm3,i,ki,ks,inv(ki)}

  composition

        Session(b,m,scm1,kb,ks)
    /\  Session(b,i,scm2,kb,ks)
    /\  Session(i,m,scm3,ki,ks)

end role
```

```
goal

% The established key X must be a secret between the base and the mobile
  secrecy_of X

% Authentication: base station authenticates mobile
  IMSR_Base weakly authenticates IMSR_Mobile on x

end goal
```

```
Environment()
```

Part V Acknowledgements

Many people have contributed to this large body of specifications – not only members of the actual AVISPA team, but also students from various universities in Europe, the United States, India, China, and Australia. These include David Gümbel, Vishal Sankhla, Murugaraj Shanmugam, Jing Zhang, and Daniel Plasto. Daniel Plasto and Vishal Sankhla contributed to the documentation of a number of protocol models including the Kerberos and EAP suites.

References

[BM98] Colin Boyd and Anish Mathuria. Key establishment protocols for secure mobile communications: A selective survey. *Lecture Notes in Computer Science*, 1438:344ff, 1998.

[CJ] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: www.cs.york.ac.uk/~jac/papers/drareview.ps.gz.

[CJP03] Pat Calhoun, Tony Johansson, and Charles Perkins. Diameter Mobile IPv4 Application, October 2003. Work in Progress.

[DA99] T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0, January 1999. Status: Proposed Standard.

[ISO97] ISO/IEC. ISO/IEC 9798-3: Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using digital signature techniques, 1997.

[Kau03] Charlie Kaufman. Internet Key Exchange (IKEv2) Protocol, October 2003. Work in Progress.

[KCK97] J. Klensin, R. Catoe, and P. Krumviede. RFC 2195: IMAP/POP AUTHorize Extension for Simple Challenge/Response, September 1997. Status: Proposed Standard.

[Mea99] Catherine Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.

[Pau99] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Computer and System Security*, 2(3):332–351, 1999.

[Per03] Charles Perkins. Mobile IPv4 Challenge/Response Extensions (revised), October 2003. Work in Progress.

[SMW99] Bruce Schneier, Mudge, and David Wagner. Cryptanalysis of microsoft's PPTP authentication extensions (MS-CHAPv2). In *CQRE: International Exhibition and Congress on Secure Networking – CQRE [Secure]*, 1999.

[Wu00] T. Wu. RFC 2945: The SRP Authentication and Key Exchange System, September 2000. Status: Proposed Standard.

[Zor00] G. Zorn. RFC 2759: Microsoft PPP CHAP Extensions, Version 2, January 2000. Status: Informational.