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: *v2.0*
Date of delivery: *15.07.2005*
Classification: *public*
Person-months required: *14*
Due on: *31.12.2004*
Total pages: *410*

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.

Problem Classification:

lists the goals (according to the list in Deliverable 6.1 [AVI03, §3]) addressed by the model.

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 `secFAHA, secFAMN, secMNHA`
- weak authentication on `k_faha1`
- weak authentication on `k_mnfa1`
- weak authentication on `k_faha2`
- weak authentication on `k_mnha1`
- weak authentication on `k_mnha2`
- weak authentication on `k_mnfa2`

Problem Classification: G1, G7, G10, G12

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  State         : nat,
         K_MnFa,K_MnHa : symmetric_key

  init   State := 0

  transition

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

   2. State = 1
      /\ Rcv( {K_MnFa'.K_MnHa'}_K_MnAAAH.
              {{K_MnFa'.K_MnHa'}_K_MnAAAH}_K_MnHa')
      =|>
      State' := 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
    State              : 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 State := 0

  transition

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

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

   3. State = 2
      /\ Rcv( FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
              KeyMnHaKeyMnFa'.SignKeyMnHaKeyMnFa'.
              {FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
               KeyMnHaKeyMnFa'.SignKeyMnHaKeyMnFa'}_K_FaAAAL)
```

```
      =|>
      State' := 3
      /\ Snd(KeyMnHaKeyMnFa'.SignKeyMnHaKeyMnFa')
      /\ 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
    State                : 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 State := 0

  transition

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

   2. State = 1
      /\ Rcv( FA.{K_MnFa'.K_FaHa'}_K_AAAHAAAL.
              KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'.
              {FA.{K_MnFa'.K_FaHa'}_K_AAAHAAAL.
               KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'}_K_AAAHAAAL)
      =|>
      State' := 2
```

```
      /\ Snd( FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
                KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'.
            {FA.{K_MnFa'.K_FaHa'}_K_FaAAAL.
                KeyMnFaKeyMnHa'.SignedKeyMnFaKeyMnHa'}_K_FaAAAL)

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  State                : nat,
         K_FaHa,K_MnHa,K_MnFa : symmetric_key

  const  secFAHA, secFAMN, secMNHA   : protocol_id

  init   State := 0

  transition

   1. State   = 0
      /\ Rcv(FA.MN.AAAH.{FA.MN.AAAH}_K_MnAAAH)
      =|>
      State' := 1
      /\ K_MnHa' := new()
      /\ K_MnFa' := new()
      /\ K_FaHa' := new()
      /\ 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)
      /\ 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. State = 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)
      =|>
      State' := 2
      /\ 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)
      /\ secret(K_FaHa,secFAHA,{FA,HA})
      /\ secret(K_MnFa,secFAMN,{FA,MN})
      /\ secret(K_MnHa,secMNHA,{MN,HA})

end role
```

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

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

  init State := 0

  transition

   1. State = 0
      /\ Rcv( MN.{K_MnHa'.K_FaHa'}_K_AAAHHa.KeyMnFaKeyMnHa'.
             {MN.{K_MnHa'.K_FaHa'}_K_AAAHHa.KeyMnFaKeyMnHa'}_K_AAAHHa)
      =|>
```

```
      State' := 1
      /\ Snd( KeyMnFaKeyMnHa'.{KeyMnFaKeyMnHa'}_K_MnHa'.
              {KeyMnFaKeyMnHa'.{KeyMnFaKeyMnHa'}_K_MnHa'}_K_AAAHHa)
      /\ 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

  intruder_knowledge = {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
  secrecy_of secFAHA, secFAMN, secMNHA % addresses G12

  %AAA_MIP_FA weakly authenticates AAA_MIP_AAAH on k_faha1
  weak_authentication_on k_faha1 % addresses G1,G7,G10
  %AAA_MIP_FA weakly authenticates AAA_MIP_AAAH on k_mnfa1
  weak_authentication_on k_mnfa1 % addresses G1,G7,G10
  %AAA_MIP_HA weakly authenticates AAA_MIP_AAAH on k_faha2
  weak_authentication_on k_faha2 % addresses G1,G7,G10
  %AAA_MIP_HA weakly authenticates AAA_MIP_AAAH on k_mnha1
  weak_authentication_on k_mnha1 % addresses G1,G7,G10
  %AAA_MIP_MN weakly authenticates AAA_MIP_AAAH on k_mnha2
  weak_authentication_on k_mnha2 % addresses G1,G7,G10
  %AAA_MIP_MN weakly authenticates AAA_MIP_AAAH on k_mnfa2
  weak_authentication_on k_mnfa2 % addresses G1,G7,G10

end goal

environment()
```

2 CTP: Context Transfer Protocol, non-predictive variant

Protocol Purpose

Multiple authentication for mobile communication in PANA,
transmitting context of a client between PANA agents after handover of client

Definition Reference

- `http://www.ietf.org/internet-drafts/draft-ietf-seamoby-ctp-11.txt`
- Handover-Aware Access Control Mechanism: CTP for PANA [BLMTM04]
- Use of Context Transfer Protocol (CTP) for PANA,
  `http://ietfreport.isoc.org/idref/draft-bournelle-pana-ctp/`

Model Authors

Lan Liu, Siemens CT IC 3, February 2005

Alice&Bob style

```
PaC : PANA Client
PPAA : previous PANA Authentication Agent
NPAA : new PANA Authentication Agent
It is assumed that PPAA and NPAA have mutually authenticated each other
before this protocol starts.
1. NPAA ----------------------------- NPAA -----------------------------> PaC
2. PaC  --------              IP_PaC.IP_pPAA.NPaC.
                Hash(CTP_key.IP_PaC.IP_pPAA.NPaC) --------------------> NPAA
3. NPAA --------             {IP_PaC.IP_pPAA.NPaC.
                Hash(CTP_key.IP_PaC.IP_pPAA.NPaC)}_ESP_Key -----------> PPAA
4. PPAA ----------- {AAA_ID.AAA_k_i.PaC}_ESP_Key ----------------------> NPAA
5. NPAA ---------- NSId.NnPAA.Hash(MAC_key.NSId.NnPAA) ----------------> PaC
6. PaC  --------------- NnPAA.Hash(MAC_key.NnPAA) ---------------------> NPAA
```

Problems Considered: 3

- secrecy of `mac_key`

- authentication on `ppaa_pac_ip_pac`
- authentication on `npaa_pac_mac_key`

Problem Classification: G1, G3, G7

Attacks Found: None

Further Notes

HLPSL Specification

```
role new_PANA_Authentication_Agent(
  NPAA, PaC, PPAA : agent,
  ESP_Key         : symmetric_key,
  AAA_ID          : text,
  Hash, Key_f     : function,
  Snd, Rcv        : channel(dy))
played_by NPAA def=

local
  State                       : nat,
  NnPAA, NSId,
  NPaC, IP_PaC, IP_pPAA       : text,
  New_AAA_k, MAC_key, AAA_k_i : message,     % should be symmetric_key
  H1                          : message

const mac_key : protocol_id

init State:=0

transition

  0.  State=0
      /\ Rcv(start)
  =|> State':=2
      /\ Snd(NPAA)
```

```
  2.  State=2
      /\ Rcv(IP_PaC'.IP_pPAA'.NPaC'.H1')
  =|> State':=4
      /\ Snd({IP_PaC'.IP_pPAA'.NPaC'.H1'}_ESP_Key)

  4.  State=4
      /\ Rcv({AAA_ID.AAA_k_i'.PaC}_ESP_Key)
  =|> State':= 6
      /\ NnPAA' := new()
      /\ NSId':=new()
      /\ New_AAA_k':=Key_f(AAA_k_i'.NPaC.NnPAA')
      /\ MAC_key':=Key_f(New_AAA_k'.NPaC.NnPAA'.NSId')
      /\ Snd(NSId'.NnPAA'.Hash(MAC_key'.NSId'.NnPAA'))

  6.  State=6
      /\ Rcv(NnPAA.Hash(MAC_key.NnPAA))
  =|> State':=8
      /\ request(NPAA, PaC, npaa_pac_mac_key, MAC_key)
      % only now PaC has been authenticated and secrecy may be checked!
      /\ secret(MAC_key, mac_key, {PaC, NPAA})

end role
```

```
role pANA_Client(
  NPAA, PaC, PPAA     : agent,
  CTP_KEY, AAA_k      : symmetric_key,
  Hash, Key_f         : function,
  IP_PaC, AAA_ID,
  Session_ID, IP_pPAA : text,
  Snd, Rcv            : channel(dy))
played_by PaC def=

local
  State                       : nat,
  NPaC                        : text,
  New_AAA_k, AAA_k_i, MAC_key : message,  %should be: symmetric_key
  NnPAA, NSId       : text,
  Signature                   : message
```

```
init State:=1

transition

  1.  State=1
      /\ Rcv(NPAA)
  =|> State':=7
      /\ NPaC':=new()
      /\ Snd(IP_PaC.IP_pPAA.NPaC'. Hash(CTP_KEY.IP_PaC.IP_pPAA.NPaC'))
      /\ witness(PaC, PPAA, ppaa_pac_ip_pac, IP_PaC)

  7.  State=7
      /\ Rcv(NSId'.NnPAA'.
             Hash(MAC_key'.NSId'.NnPAA'))
      /\ MAC_key'=Key_f(New_AAA_k'.NPaC.NnPAA'.NSId')
      /\ New_AAA_k'=Key_f(AAA_k_i'.NPaC.NnPAA')
      /\ AAA_k_i'=Key_f(AAA_k.AAA_ID.Session_ID)
  =|> State':=9
      /\ Snd(NnPAA'.Hash(MAC_key'.NnPAA'))
      /\ witness(PaC, NPAA, npaa_pac_mac_key, MAC_key')

end role
```

```
role previous_PANA_Authentication_Agent(
  NPAA, PaC, PPAA         : agent,
  CTP_KEY, ESP_Key, AAA_k : symmetric_key,
  Hash, Key_f             : function,
  IP_PaC, AAA_ID,
  Session_ID, IP_pPAA     : text,
  Snd, Rcv                : channel(dy))
played_by PPAA def=

local State       : nat,
      NPaC, NnPAA : text,
      AAA_k_i     : message    % should be a symmetric_key

init State:=3
```

```
transition

  3.  State=3
      /\ Rcv({IP_PaC.IP_pPAA.NPaC'.
              Hash(CTP_KEY.IP_PaC.IP_pPAA.NPaC')}_ESP_Key)
  =|> State':=5
      /\ AAA_k_i' = Key_f(AAA_k.AAA_ID.Session_ID)
      /\ Snd({AAA_ID.AAA_k_i'.PaC}_ESP_Key)
      /\ request(PPAA, PaC, ppaa_pac_ip_pac, IP_PaC)
end role
```

```
role session(
  NPAA,PaC, PPAA          : agent,
  CTP_KEY, ESP_Key, AAA_k : symmetric_key,
  Hash, Key_f             : function,
  IP_PaC, AAA_ID,
  Session_ID, IP_pPAA     : text)
def=

local SPaC, SNPAA, SPPAA, RPaC, RNPAA, RPPAA : channel(dy)

composition
    previous_PANA_Authentication_Agent(
         NPAA, PaC, PPAA, CTP_KEY, ESP_Key,
         AAA_k, Hash, Key_f, IP_PaC, AAA_ID,
         Session_ID, IP_pPAA, SPPAA, RPPAA)
 /\ new_PANA_Authentication_Agent(
         NPAA, PaC, PPAA, ESP_Key,
         AAA_ID, Hash, Key_f, SNPAA, RNPAA)
 /\ pANA_Client(
         NPAA, PaC, PPAA, CTP_KEY,
         AAA_k, Hash, Key_f, IP_PaC, AAA_ID,
         Session_ID, IP_pPAA, SPaC, RPaC)
end role
```

```
role environment() def=
```

```
const
  ppaa_pac_ip_pac, npaa_pac_mac_key : protocol_id,
  npaa, pac, ppaa                   : agent,
  h, key_f                          : function,
  ctp_key, esp_key, aaa_k, aaa_k_i,
  ctp_key_i                         : symmetric_key,
  ip_pac, aaa_id, ip_pPAA,
  sid1, sidi, ip_i                  : text

intruder_knowledge = {npaa, pac, ppaa, h, key_f, aaa_id, aaa_k_i,
                      ip_pPAA, ctp_key_i, session_id_i, ip_i}

composition
     session(npaa, pac, ppaa, ctp_key, esp_key, aaa_k, h, key_f,
             ip_pac, aaa_id, sid1, ip_pPAA)

  /\ session(npaa, i, ppaa, ctp_key_i, esp_key, aaa_k_i, h, key_f,
             ip_i, aaa_id, sidi, ip_pPAA)
end role
```

```
goal
  secrecy_of mac_key

  authentication_on ppaa_pac_ip_pac  % addresses G1 and G3
  authentication_on npaa_pac_mac_key % addresses G3 and G7
end goal
```

```
environment()
```

3 SIP, Diameter Session Initiation Protocol

Protocol Purpose

This is a Diameter application that allows a Diameter client to request authentication and authorization information to a Diameter server for Session Initiation Protocol (SIP) based IP multimedia services.

Definition Reference

- draft-ietf-aaa-diameter-sip-app-07, IETF Memo available at www.ietf.org/internet-drafts/draft-ietf-aaa-diameter-sip-app-07.txt.
- IETF RFC 2617, available at www.ietf.org/rfc/rfc2617.txt.

Model Authors

- Jacopo Mantovani, AI-Lab, DIST, University of Genova
- Luca Compagna, AI-Lab, DIST, University of Genova

Alice&Bob style

```
UAC   : User Agent Client
SSn   : n-th SIP Server
DS    : Diameter Server
Dest  : Models the requested service
sip401: http unauthorized response
sip200: http authorized response

1.  UAC -> SS1 : sipregister.UAC.Dest
2.  SS1 -> DS : UAC.Dest
3.  DS  -> SS1 : UAC.SS2
4.  SS1 -> SS2 : sipregister.UAC.Dest
5.  SS2 -> DS : Dest.UAC
6.  DS  -> SS2 : Nonce.UAC
7.  SS2 -> SS1 : sip401.Nonce
8.  SS1 -> UAC : sip401.Nonce
9.  UAC -> SS1 : sipregister.UAC.dest.Nonce.H(Nonce.H(UAC.PWD).H(dest))
10. SS1 -> DS : UAC.Dest
```

```
11. DS  -> SS1 : UAC.SS2
12. SS1 -> SS2 : sipregister.UAC.Dest.Nonce.H(Nonce.H(UAC.PWD).H(dest))
13. SS2 -> DS : Dest.UAC.Nonce.H(Nonce.H(UAC.PWD).H(dest))
14. DS  -> SS2 : UAC.success
15. SS2 -> SS1 : sip200
16. SS1 -> UAC : sip200
```

Model Limitations

We model here only one successfull run of the protocol, that is, a first attempt where the authentication fails and the credentials of the User Agent Client are requested (together with a challenge) by the Diameter Server, and a second part where the Client sends his credentials to the Server, for authorization and authentication. The credentials are sent via the HTTP Digest schema, which can safely be modeled in HLPSL by using a hash function. Lastly, we assume that the SIP server and the Diameter client are located in the same node, so that the SIP server is able to receive and process SIP requests and responses which in turn relies on the AAA infrastructure for authenticating the SIP request and authorizing the usage of particular SIP services.

Problems Considered: 1

- authentication on `y`

Problem Classification: G1, G2, G3

Attacks Found: None

HLPSL Specification

```
role sip_server_1(
                SS1,DS : agent,
                CH_UAC_SS1,CH_SS1_UAC,CH_DS_SS1,CH_SS1_DS,
                CH_SS2_SS1,CH_SS1_SS2 : channel(dy))
```

```
played_by SS1 def=

local   State   : nat,
        Dest    : protocol_id,
        SS2,UAC : agent,
        X,Y     : message,
        Nonce   : text

init State := 1

transition

0. State = 1
/\ CH_UAC_SS1(sipregister.UAC'.Dest')
=|>
State' := 2
/\ CH_SS1_DS(UAC'.Dest')

1. State = 2
/\ CH_DS_SS1(UAC.SS2')
=|>
State' := 3
/\ CH_SS1_SS2(sipregister.UAC.Dest)

2. State = 3
/\ CH_SS2_SS1(sip401.Nonce')
=|>
State' := 4
/\ CH_SS1_UAC(sip401.Nonce')

3. State = 4
/\ CH_UAC_SS1(sipregister.UAC.Dest.Nonce.Y')
=|>
State':= 5
/\ CH_SS1_DS(UAC.Dest)

4. State = 5
/\ CH_DS_SS1(UAC.SS2')
=|>
State':= 6
/\ CH_SS1_SS2(sipregister.UAC.Dest.Nonce.Y)
```

```
        5. State = 6
        /\ CH_SS2_SS1(X')
        =|>
        State':= 7
        /\ CH_SS1_UAC(X')

end role
```

```
role sip_server_2(
                SS2,DS : agent,
                CH_DS_SS2,CH_SS2_DS,CH_SS1_SS2,CH_SS2_SS1 : channel(dy))

        played_by SS2 def=

        local State: nat,
              Dest : protocol_id,
              UAC  : agent,
              Nonce: text,
              Y    : message

        init State := 1

        transition

        6. State = 1
        /\ CH_SS1_SS2(sipregister.UAC'.Dest')
        =|>
        State'= 2
        /\ CH_SS2_DS(Dest'.UAC')

        7. State = 2
        /\ CH_DS_SS2(Nonce'.UAC)
        =|>
        State'= 3
        /\ CH_SS2_SS1(sip401.Nonce')

        8. State = 3
        /\ CH_SS1_SS2(sipregister.UAC.Dest.Nonce.Y')
        =|>
```

```
        State':= 4
        /\ CH_SS2_DS(Dest.UAC.Nonce.Y')

        9. State = 4
        /\ CH_DS_SS2(UAC.success)
        =|>
        State':= 5
        /\ CH_SS2_SS1(sip200)

end role
```

```
role diameter_server(
                DS,SS1,SS2 : agent,
                PWD : text,
                H : hash,
                CH_SS1_DS,CH_DS_SS1,CH_SS2_DS,CH_DS_SS2 : channel(dy))

        played_by DS def=

        local   State   : nat,
                UAC     : agent,
                Nonce   : text,
                Y       : message

        init State := 1

        transition

        10. State = 1
        /\ CH_SS1_DS(UAC'.dest)
        =|>
        State' := 2
        /\ CH_DS_SS1(UAC'.SS2)

        11. State = 2
        /\ CH_SS2_DS(dest.UAC)
        =|>
        State':= 3
        /\ Nonce' := new()
        /\ CH_DS_SS2(Nonce'.UAC)
```

```
        12. State = 3
        /\ CH_SS1_DS(UAC.dest)
        =|>
        State':= 4
        /\ CH_DS_SS1(UAC.SS2)

        13. State = 4
        /\ CH_SS2_DS(dest.UAC.Nonce.H(Nonce.H(UAC.PWD).H(dest)))
        =|>
        State':= 5
        /\ CH_DS_SS2(UAC.success)
        /\ request(UAC,UAC,y,H(Nonce.H(UAC.PWD).H(dest)))

end role
```

```
role user_agent_client(
                UAC,SS1 : agent,
                PWD : text,
                H :  hash,
                CH_SS1_UAC,CH_UAC_SS1:channel(dy))

        played_by UAC

        def=

        local   State : nat,
                Nonce : text

        init State := 1

        transition

        14. State = 1
        /\ CH_SS1_UAC(start)
        =|>
        State':=2
        /\ CH_UAC_SS1(sipregister.UAC.dest)

        15. State = 2
```

```
        /\ CH_SS1_UAC(sip401.Nonce')
        =|>
        State':= 3
        /\ CH_UAC_SS1(sipregister.UAC.dest.Nonce'.H(Nonce'.H(UAC.PWD).H(dest)))
        /\ witness(UAC,UAC,y,H(Nonce'.H(UAC.PWD).H(dest)))

        16. State = 3
        /\ CH_SS1_UAC(sip200)
        =|>
        State':= 4

end role

role session(UAC,SS1,SS2,DS:agent,H:hash,PWD:text) def=

        local   SND_SS1A, RCV_SS1A, SND_SS1B, RCV_SS1B, SND_SS1C, RCV_SS1C: channel(dy),
                SND_SS2A, RCV_SS2A, SND_SS2B, RCV_SS2B : channel(dy),
                SND_DSA, RCV_DSA, SND_DSB, RCV_DSB : channel(dy),
                SND_UACA, RCV_UACA  : channel(dy)

        composition

     sip_server_1(SS1,DS,SND_SS1A,RCV_SS1A,SND_SS1B,RCV_SS1B,SND_SS1C,RCV_SS1C)
  /\ sip_server_2(SS2,DS,SND_SS2A, RCV_SS2A, SND_SS2B, RCV_SS2B)
  /\ diameter_server(DS,SS1,SS2,PWD,H,SND_DSA, RCV_DSA, SND_DSB, RCV_DSB)
  /\ user_agent_client(UAC,SS1,PWD,H,SND_UACA, RCV_UACA)

end role
```

```
role environment() def=

        const   uac,ss1,ss2,ds          : agent,
                h                       : hash,
                y                       : protocol_id,
                sipregister,success     : protocol_id,
                sip401,sip200           : protocol_id,
                dest                    : protocol_id,
                pwd                     : text

        intruder_knowledge = {uac,ss1,ss2,ds,sipregister,sip401,sip200,success,h,i}
```

```
        composition

           session(uac,ss1,ss2,ds,h,pwd)

 end role
```

```
goal

authentication_on y % addresses G1, G2, G3

end goal
```

```
environment()
```

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

4.1 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

- authentication on `key`
- authentication on `key1`
- secrecy of `sec_m_Key, sec_v_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,
    CH2,CH4      : text,
    GY,Key       : message

  const sec_m_Key : protocol_id

  init  State := 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State':= 1 /\ X' := new()
               /\ CH1' := new()
               /\ 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 /\ CH3' := new()
               /\ 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,sec_m_Key,{VGK,AuF})  % 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,
    CH1,CH3        : text

  const sec_v_Key : protocol_id

  init  State := 0

  transition

  1. State = 0 /\ RCV(MT.VGK.NIL.CH1'.GX'.FM1') =|>
     State'= 1 /\ Y' := new()
               /\ 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 /\ CH2' := new()
               /\ 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 /\ CH4' := new()
               /\ SND(VGK.MT.CH3'.CH4'.F(Key.VGK.MT.CH3'.CH4'))
               /\ request(VGK,MT,key1,Key)
               /\ secret(Key,sec_v_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,
    F          : function,
    ZZ,ZZ_VA   : symmetric_key,
    NIL,G      : text)
def=

  local SND,RCV : channel (dy)
```

```
  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=

  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

  intruder_knowledge = {a,b,auf,f,g,nil,zz_i_auf}

  composition
     session(a,b,auf,f,zz_a_auf,zz_b_auf,nil,g)
  /\ session(b,a,auf,f,zz_b_auf,zz_a_auf,nil,g)
  /\ session(i,b,auf,f,zz_i_auf,zz_b_auf,nil,g)
  /\ session(a,i,auf,f,zz_a_auf,zz_i_auf,nil,g)

end role
```

```
goal

  % Entity authentication (G1)
  % Message authentication (G2)
  % Replay protection (G3)
  % Authorization (by T3P) (G6)
  % Key authentication (G7)
  authentication_on key
```

```
  authentication_on key1
  secrecy_of sec_m_Key, sec_v_Key

end goal
```

```
environment()
```

4.2 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

- authentication on `key`
- authentication on `key1`
- secrecy of `sec_m_Key, sec_v_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,
    CH2,CH4     : text,
    GY,Key      : message
```

```
  const sec_m_Key : protocol_id

  init  State := 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State':= 1 /\ X' := new()
               /\ CH1' := new()
               /\ 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 /\ CH3' := new()
               /\ 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,sec_m_Key,{VGK,AuF})

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,
    CH1,CH3         : text

  const sec_v_Key : protocol_id

  init  State := 0

  transition

  1. State = 0  /\ RCV(MT.VGK.NIL.CH1'.GX'.FM1') =|>
     State':= 1 /\ Y' := new()
                /\ 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 /\ CH2' := new()
                /\ 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 /\ CH4' := new()
                /\ SND(VGK.MT.CH3'.CH4'.F(Key.VGK.MT.CH3'.CH4'))
                /\ request(VGK,MT,key1,Key)
                /\ secret(Key,sec_v_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,
    F          : function,
    ZZ,ZZ_VA   : symmetric_key,
    NIL,G      : text)
def=

  local SND,RCV : channel (dy)

  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=

  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

  intruder_knowledge = {a,b,auf,f,zz_i_auf}

  composition
    session(a,b,auf,f,zz_a_auf,zz_b_auf,nil,g)
 /\ session(i,b,auf,f,zz_i_auf,zz_b_auf,nil,g)
 /\ session(a,i,auf,f,zz_a_auf,zz_i_auf,nil,g)

end role
```

```
goal

  % Entity authentication (G1)
  % Message authentication (G2)
  % Replay protection (G3)
  % Authorization (by T3P) (G6)
  % Key authentication (G7)
  authentication_on key
  authentication_on key1
  secrecy_of sec_m_Key, sec_v_Key

end goal
```

```
environment()
```

5 NSIS QoS-NSLP Authorization

(Next Steps In Signaling, Quality of Service NSIS Signaling Layer Application, Authorization process of the QoS reservation requests)

Definition Reference

`http://www.ietf.org/internet-drafts/draft-ietf-nsis-qos-nslp-06.txt` [dB03]

Protocol Purpose

Authorization of the QoS resource requestor (Client),
Protection of the 3-party model against misbehavior of the Client (MITM attack), the Router and the Server

Model Authors

Tseno Tsenov for Siemens CT IC 3, June 2005

Alice&Bob style

```
1. R --- {Service.C.R}_K_CR ------------> C
2. R <-- {{Service.C.S}_K_CS.C.R}_K_CR -- C
3. R --------- {{Service.C.S}_K_CS.C.S.R}_inv(K_R) --> S
4. R <-------- {C.R.S}_inv(K_S) --------------------- S
```

Model Limitations

- NSIS QoS NLPS application provides QoS signaling. The current analysis covers only the authorization aspects in its functionality.
- Please consider that the design of the NSIS QoS-NSLP application is ongoing and several functional changes and extensions might occur.

Problems Considered: 2

- weak authentication on `router_server_clientid`
- weak authentication on `server_client_service`

Problem Classification: G2, G20

Attacks Found: None

Further Notes

1. Sessions between the Router and the Client are based on TLS and sessions between the Router and the Server are based on any AAA protocol. These protocols provide authentication, integrity and replay protection of the exchanged messages. The Client and its subscriber profile is known at the Server and is authenticated on the symmetric key shared between the Client and Server.

   The main goal of the model is the authorization aspect and not authentication of the involved peers. Therefore the above protocols are not modeled but services that their sessions provide are directly used. This implies:

   - The Client and the Router share a session symmetric key provided by TLS
   - Considering the features of the AAA protocol, message authentication is sufficient for modeling the Server-Router message exchange, which is modeled with public keys of the Server and the Router.
   - For modeling of the TLS or AAA-protocol sessions, identities of the session peers are included in the messages.

2. There are two authorization goals that are modeled:

   - The Server provides to the Router an authorization decision for the Client, based on Client's authenticated identity. By matching the authenticated identity of the Client with the identity of the authorized peer provided by the Server in the authorization decision, the Router mitigates the vulnerability to MITM attack.
   - The Client trusts the Server that it checks if the Router is authorized to provide the service offered to the Client. This mitigates misbehavior of the Router.

   Since AVISPA does not provide direct proof of authorization, but only proof of authentication verification of the above goals is indirectly modeled by the following approach:

   - proof of authentication of the Server by the Router on the identity of the Client models authorization of the Client.
   - Authorization of the Router for provisioning of a service is modeled by a shared parameter `Service`, known by the Server and the Router. Proof of authentication of the Client by the Server on the value of the parameter `service` models authorization of the Router.

3. Due to the use of a TLS and AAA session, we can assume replay protection. Moreover it is assumed that the Routers authorized to provide the service do not misbehave. Thus only *weak* authentication is specified as goals.

HLPSL Specification

```
role client(C,R,S          : agent,
            K_CR, K_CS     : symmetric_key,
            SND_CR, RCV_CR : channel(dy)
           )

played_by C
def=

  local State   : nat,
        Service : text

  init  State := 1

  transition

    1.  State   = 1 /\ RCV_CR( {Service'.C.R}_K_CR) =|>
        State' := 2 /\ SND_CR({{Service'.C.S}_K_CS.C.R}_K_CR)
                    /\ witness(C,S,server_client_service,Service')

end role
```

```
role router(C,R,S    : agent,
            K_CR     : symmetric_key,
            K_S, K_R : public_key,
            Service  : text,
            SND_CR, RCV_CR, SND_RS, RCV_RS: channel(dy)
            )
```

```
played_by R
def=

  local State     : nat,
        MessageCS : {text.agent.agent}_symmetric_key

  init  State := 1

  transition

    1.  State  = 1 /\ RCV_CR(start) =|>
        State':= 2 /\ SND_CR({Service.C.R}_K_CR)

    2.  State  = 2 /\ RCV_CR({MessageCS'.C.R}_K_CR) =|>
        State':= 3 /\ SND_RS({MessageCS'.C.S.R}_inv(K_R))

    3.  State  = 3 /\ RCV_RS({C.S.R}_inv(K_S)) =|>
        State':= 4 /\ wrequest(R,S,router_server_clientid,C)

end role
```

```
role server(R,S      : agent,
            K_S, K_R : public_key,
            Service  : text,
            KeyRing  : (agent.symmetric_key) set,
            SND_RS, RCV_RS: channel(dy)
            )

played_by S
def=

  local State     : nat,
        C         : agent,
        K_CS      : symmetric_key,
        MessageCS : {text.agent.agent}_symmetric_key

  init  State := 1

  transition
```

```
    1.  State = 1 /\ RCV_RS({MessageCS'.C'.S.R}_inv(K_R))
                  /\ in(C'.K_CS', KeyRing)
                  /\ MessageCS' = {Service.C'.S}_K_CS' =|>
        State'= 2 /\ SND_RS({C'.S.R}_inv(K_S))
                  /\ witness(S,R,router_server_clientid,C')
                  /\ wrequest(S,C',server_client_service,Service)

end role
```

```
role session(C,R,S: agent,
             K_CR,K_CS : symmetric_key,
             K_S, K_R : public_key,
             Service: text,
             KeyRing: (agent.symmetric_key) set
            )
def=
   local
      SCR,RCR,SSR,RSR: channel(dy)

   composition

         client(C,R,S,K_CR,K_CS,SCR,RCR)
      /\ router(C,R,S,K_CR,K_S,K_R,Service,SCR,RCR,SSR,RSR)
      /\ server(R,S,K_S,K_R,Service,KeyRing,SSR,RSR)

end role
```

```
role environment() def=

 local KeyRing                          : (agent.symmetric_key) set

 const c, r, s                          : agent,
       k_cr, k_cs, k_is, k_ic, k_ir : symmetric_key,
       k_s, k_r, k_i                    : public_key,
       service                          : text,
       server_client_service,
```

```
      router_server_clientid       : protocol_id

 init KeyRing = {c.k_cs, i.k_is}

 intruder_knowledge={c,r,s,service,k_is,k_ic,k_ir,k_s,k_r,k_i,inv(k_i)}

 composition

       session(c,r,s,k_cr,k_cs,k_s,k_r,service,KeyRing)
 /\    session(i,r,s,k_ir,k_is,k_s,k_r,service,KeyRing)
 /\    session(c,i,s,k_ic,k_cs,k_s,k_i,service,KeyRing)
 /\    session(c,r,i,k_cr,k_ic,k_i,k_r,service,KeyRing)

end role
```

```
goal

  %client authorization
  weak_authentication_on router_server_clientid % addresses G2: agreeement on C

  %router authorization
  weak_authentication_on server_client_service  % addresses G2 and G20:
          % the router provides the service the client has asked (and payed) for

end goal
```

```
environment()
```

6 Geopriv

6.1 Variant with pseudonym for Location Recipient only

Definition Reference

`http://www.faqs.org/rfcs/rfc3693.html` [CMM+04]

Protocol Purpose

Obtain geographical location information restricted by a privacy policy.
Using a pseudonym, the location recipient is anonymous to the location server.

Model Authors

Lan Liu for Siemens CT IC 3, January 2005

Alice&Bob style

```
MU : Mobile User (= Target) (subsumes the Rule Maker)
LR : Location Recipient
LS : Location Server (subsumes the Location Sighter)
1. LR ------ LR.N_LR.{LR}_K_MU_LR -> MU
2. LR <- {N_LR.Psi.K_Psi}_K_MU_LR -- MU
3.                                  MU -- {MU.Psi.K_Psi DT}_K_MU_LS -> LS
% some time later, LR requests the location of MU:
4. LR ---------------- {LS.MU.Psi.K_Psi.K1}_Pk_LS ------------------> LS
5. LR <------------------------ {DT(Loc)}_K1 ------------------------ LS
DT ("data type") describes the accuracy of the location information.
It is a function projecting/filtering Loc to the accuracy allowed by the MU.
```

Model Limitations

For simplicity we model the Location Sighter as part of the Location Server, which is fine here because the Location Server is allowed to know the identity of the Target.

Problems Considered: 5

- secrecy of `filtered_loc, psi, k_psi, k1`
- authentication on `lr_ls_filtered_loc`

- authentication on `lr_mu_n_lr`
- weak authentication on `ls_mu_psi`
- weak authentication on `mu_lr_lr`

Problem Classification: G1, G2, G3, G12, G20

Attacks Found: None

Further Notes

- The name of LR in the initial contact is modelled as in the clear and encrypted. The encrypted form of the LR information is used by T to authenticate the LR. In reality the inital contact can be part of another protocol, protected via PKI, or unprotected.
- An LR can get a certain `{Psi,K_Psi}` pair from the MU. `K_Psi` is the key related to the pseudonym `Psi` of a LR. `Psi` and `K_Psi` are used for authorization to get location information from the LS. Although `K_Psi` is the password for `Psi` of `LR`, it could be omitted here because the secrecy of `Psi` suffices.
- `K1` is a temporary key of LR, generated by LR for encryption of the location information sent by LS.
- LS cannot authenticate LR because he knows only the pseudoynm of LR, since an important objective of this protocol is the anonymity of LR to LS.
- The secrecy fact for `filtered_loc` is given in the role of the Location Server (where the secret actually is produced). To make this possible, LS has LR as its parameter, but only for technical reasons to state the goal. LS does not make use of this "knowledge", as it should know only LS's pseudonym.
- In the last step, LS does not know to whom to answer. In reality, an IP address is used, but here, one may regard it is a broadcast.

HLPSL Specification

```
role locationRecipient(
        MU, LR, LS       : agent,
        K_MU_LR          : symmetric_key,
        Pk_LS            : public_key,
```

```
        Snd, Rcv        : channel(dy)) played_by LR def=

local
        State           : nat,
        N_LR, Psi       : text,
        K_Psi           : symmetric_key,
                        % password for pseudonym Psi of a certain LR,
                        % generated by MU and stored by LS
        K1              : public_key, % could also be: symmetric_key
        Filtered_Loc    : message

init State := 0

transition

        0. State  = 0 /\ Rcv(start)
       =|> State':= 2 /\ N_LR' := new()
                      /\ Snd(LR.N_LR'.{LR}_K_MU_LR)
                      /\ witness(LR, MU, mu_lr_lr, LR)

        2. State  = 2 /\ Rcv({N_LR.Psi'.K_Psi'}_K_MU_LR)
       =|> State':= 4 /\ K1' := new()
                      /\ secret(K1', k1, {LR, LS})
                      /\ Snd({LS.MU.Psi'.K_Psi'.K1'}_Pk_LS)

        4. State  = 4 /\ Rcv({Filtered_Loc'}_K1)
       =|> State':= 6 /\ request(LR, LS, lr_ls_filtered_loc, Filtered_Loc')
                      /\ request(LR, MU, lr_mu_n_lr, N_LR)

end role
```

```
role mobileUser(
        MU, LR, LS : agent,
        K_MU_LR    : symmetric_key,
        K_MU_LS    : symmetric_key,
        Snd_LR, Snd_LS,
        Rcv        : channel(dy)) played_by MU def=

local
```

```
        State    : nat,
        N_LR     : text,
        Psi      : text,
        K_Psi    : symmetric_key,
        DT       : function

const  psi, k_psi : protocol_id

init State := 1

transition

        1. State  = 1 /\ Rcv(LR.N_LR'. {LR}_K_MU_LR)
       =|> State':= 3 /\   Psi' := new()
                      /\ K_Psi' := new()
                      /\ secret(  Psi,  psi, {MU, LR, LS})
                      /\ secret(K_Psi,k_psi, {MU, LR, LS})
                      /\ Snd_LR({N_LR'.Psi'.K_Psi'}_K_MU_LR)
                      /\ witness(MU, LR, lr_mu_n_lr, N_LR')
                      /\ wrequest(MU, LR, mu_lr_lr, LR)
                      /\ DT' := new()  % chooses some accuracy
                      /\ Snd_LS({MU. Psi'. K_Psi'. DT'}_K_MU_LS)
                      /\ witness(MU, LS, ls_mu_psi, Psi')
end role
```

```
role locationServer(
        MU, LR,  % but LS does not use identity of LR
        LS       : agent,
        Psi_Table: (agent.text.symmetric_key.function) set,
        Pk_LS    : public_key,
        K_MU_LS  : symmetric_key,
        Snd, Rcv : channel(dy)) played_by LS def=

local   State           : nat,
        K1              : public_key,
        Na              : text,
        K_Psi           : symmetric_key,
        Psi             : text,
        DT              : function,
```

```
        Loc             : text

init    State := 7

transition

        7. State  = 7  /\ Rcv({MU. Psi'. K_Psi'. DT'}_K_MU_LS)
                                % actually, LS should learn MU here
        =|>State':= 9  /\ Psi_Table':= cons(MU.Psi'.K_Psi'. DT', Psi_Table)
                       /\ wrequest(LS, MU, ls_mu_psi, Psi')
                                       % need MU here for technical reasons

        9. State  = 9  /\ Rcv({LS. MU'. Psi'. K_Psi'. K1'}_Pk_LS)
                       /\       in(MU'. Psi'. K_Psi'. DT, Psi_Table)
 % LS checks the information MU, Psi and K_Psi, and looks up DT in the table.
        =|>State':= 11 /\ Loc':= new()
                       /\ secret(DT(Loc'),filtered_loc, {LR, LS, MU})
                       % in any case, MU is allowed to know its own location!
                       /\ Snd({DT(Loc')}_K1')
                       /\ witness(LS, LR, lr_ls_filtered_loc, DT(Loc'))

end role
```

```
role session(MU, LR, LS : agent,
             Psi_Table  : (agent.text.symmetric_key.function) set,
             K_MU_LR    : symmetric_key,
             Pk_LS      : public_key,
             K_MU_LS    : symmetric_key
            ) def=

local SLR, SMULR, SMULS, SLS, RMU, RLR, RLS : channel(dy)

composition

           locationRecipient(MU, LR, LS, K_MU_LR, Pk_LS, SLR, RLR)
        /\ mobileUser       (MU, LR, LS, K_MU_LR, K_MU_LS, SMULR, SMULS, RMU)
        /\ locationServer   (MU, LR, LS, Psi_Table,Pk_LS, K_MU_LS, SLS, RLS)

end role
```

```
role environment() def=

local
        Psi_Table: (agent.text.symmetric_key.function) set
        % shared between all instances of LS

const   ls_mu_psi, lr_mu_n_lr, k1, filtered_loc,
        ls_lr_k_psi, lr_ls_filtered_loc, mu_lr_lr: protocol_id,
        mu, lr,ls                : agent,
        k_MU_LR, k_MU_i, k_i_LR : symmetric_key,
        pk_LS                    : public_key,
        k_mu_ls, k_i_ls          : symmetric_key

init    Psi_Table := {}

intruder_knowledge = {mu, lr, ls, pk_LS, k_MU_i, k_i_LR, k_i_ls}

composition

           session(mu, lr, ls, Psi_Table, k_MU_LR, pk_LS, k_mu_ls)
         % repeat session to check for replay attacks
        /\ session(mu, lr, ls, Psi_Table, k_MU_LR, pk_LS, k_mu_ls)
        /\ session(i , lr, ls, Psi_Table, k_i_LR,  pk_LS, k_i_ls)
%       /\ session(mu, i , ls, Psi_Table, k_MU_i,  pk_LS, k_mu_ls)
%       It does not make sense to let the intruder play the role of LR
%       since then the intruder is allowed to know the (secret) location of MU.

end role
```

```
goal
        secrecy_of filtered_loc, psi, k_psi, k1 % addresses G12

        authentication_on lr_ls_filtered_loc    % addresses G2 and G3
        % authentication and integrity of location object

        % additional authentication goals, not in RFC3639:
        authentication_on lr_mu_n_lr            % addresses G1 and G3,
           % and G20: MU authorizes LR to receive the location via LS
```

```
        weak_authentication_on ls_mu_psi        % addresses G1
        weak_authentication_on mu_lr_lr         % addresses G1

end goal
```

```
environment()
```

6.2 Variant with pseudonyms for Location Recicpient and Target

Definition Reference

- http://www.faqs.org/rfcs/rfc3693.html [CMM+04]
- *IETF Geopriv: Geographic Location Privacy.* Talk by Jorge Cuellar at LIF 2002 in Vienna.

Protocol Purpose

Obtain geographical location information restricted by a privacy policy. Using pseudonyms for both the location recipient and the target, to protect their anonymity against the location server.

Model Authors

Lan Liu for Siemens CT IC 3, May 2005

Alice&Bob style

```
 LoSi : Location Sighter
 RM   : Rule Maker
 T    : Target (here we use T to describe the role of LoSi, RM and Mobile User)
 LR   : Location Recipient
 LS   : Location Server
 1. LR -------------- LR. N_LR. {K_LR. LR}_K_T_LR -------------> T
 2  LR <---{Psi_LR. Psi_T. K_LR. N_LR}_K_T_LR ------------------ T
 3. LS <-- {Psi_LR. Psi_T. K_LR. K_T. DT. Loc.
           {Psi_LR. Psi_T. K_LR. K_T. DT. Loc}_inv(K_T)}_K_LS -- T
 4. LR ------------ {Psi_LR. Psi_T}_K_LS ----------------------> LS
 5. LR <----------- {{Psi_T. DT(Loc)}_inv(K_LS)}_K_LR ---------- LS
```

Problems Considered: 4

- secrecy of `loc, filtered_loc, psi_t, psi_lr, k_lr`
- authentication on `lr_ls_filtered_loc`
- authentication on `lr_t_n_lr`
- weak authentication on `t_lr_lr`

Problem Classification: G1, G2, G3, G12, G20

Attacks Found: None

Further Notes

This version of Geopriv is different from the normal one in the sense that the real identity of the Target should not be known by the Location Server. The Location Server just knows the pseudonyms of the Target and of the Location Recipient.

Further (minor) differences are:

1. In step one, LR sends its key to the LS via the Target. In normal Geopriv, LR sends its public key directly to the Location Server, so the Target (there we use the name MU) does not learn the public key of the LR.

2. We model the Location Sighter as part of the Target and transmit in the third message both the pseudonyms and the location of the Target because the Location Server cannot associate these values since he is not allowed to know the identity of the Target. In normal Geopriv, we model the Location Sighter as part of the Location Server (which is equivalent with assuming that the Location Sighter can send the location of the Target to the LS in a secure way).

3. Passwords for `Psi_T` and `Psi_LR` like `K_Psi` for LR in normal Geopriv are omitted here; they are not really important because already the two secret pseudonyms can be used as passwords. In normal Geopriv, we haven't omitted the password for the `Psi_LR`.

4. The message in the last step is signed by the LS, for the public key of the LR `K_LR` is sent to the T in the first step, so if an intruder plays the role of the Target, then the intruder knows also the public key of the LR and the LR cannot authenticate the LS on the message DT(Loc).

Implicitly, the Target gets authorization by the LS to set up the policies for its location, because it knows its location anyway.

The secrecy fact for `filtered_loc` is given in the role of the Location Server (where the secret actually is produced). To make this possible, LS has LR and T as its parameters, but only for technical reasons to state the goal. LS does not make use of this "knowledge", as it should know only the pseudonyms.

HLPSL Specification

```
role locationRecipient(T, LR, LS        : agent,
                       K_T, K_LS        : public_key,
                       K_T_LR           : symmetric_key,
                       Snd, Rcv         : channel(dy)) played_by LR def=

local State                     : nat,
      Psi_LR, Psi_T,  N_LR      : text,
      K_LR                      : public_key,
      Filtered_Loc              : message

init State := 0

transition

        0. State  = 0 /\ Rcv(start)
       =|> State':= 2 /\ N_LR' := new()
                      /\ K_LR' := new()
                      /\ secret(K_LR' , k_lr, {T, LR, LS})
                      /\ Snd(LR. N_LR'. {K_LR'. LR}_K_T_LR)
                      /\ witness(LR, T, t_lr_lr, LR)

        2. State  = 2 /\ Rcv({Psi_LR'. Psi_T'. K_LR. N_LR}_K_T_LR)
       =|> State':= 8 /\ Snd({Psi_LR'. Psi_T'}_K_LS)

        8. State  = 8 /\ Rcv({{Psi_T. Filtered_Loc'}_inv(K_LS)}_K_LR)
       =|> State':= 10/\ request(LR, LS, lr_ls_filtered_loc, Filtered_Loc')
                      /\ request(LR, T , lr_t_n_lr, N_LR)

end role
```

```
role target(T, LR, LS            : agent,
            K_T, K_LS            : public_key,
            K_T_LR               : symmetric_key,
            Snd, Rcv             : channel(dy)) played_by T def=

local   State                    : nat,
        K_LR                     : public_key,
        Psi_T, Psi_LR, N_LR      : text,
        DT                       : function,
        Loc                      : text

const psi_t, psi_lr, loc, filtered_loc : protocol_id

init State := 1

transition

        1.  State  = 1 /\ Rcv(LR. N_LR'. {K_LR'. LR}_K_T_LR)
        =|> State':= 3 /\ Psi_T'  := new()
                       /\ Psi_LR' := new()
                       /\ secret(Psi_T' , psi_t , {T, LR, LS})
                       /\ secret(Psi_LR', psi_lr, {T, LR, LS})
                       /\ Snd({Psi_LR'. Psi_T'. K_LR'. N_LR'}_K_T_LR)
                       /\ witness (T, LR, lr_t_n_lr, N_LR')
                       /\ wrequest(T, LR, t_lr_lr, LR)

        3.  State = 3
        =|> State':= 5 /\ Loc' := new()
                       /\ DT' := new()
                       /\ Snd({Psi_LR.Psi_T. K_LR.K_T. DT'.Loc'.
                              {Psi_LR.Psi_T. K_LR.K_T. DT'.Loc'}_inv(K_T)}_K_LS)
                       /\ secret(Loc', loc, {T, LS})

end role
```

```
role locationServer(T, LR,      % but LS does not use identity of LR and T
```

```
                    LS           : agent,
                    K_LS, K_T    : public_key,
                    Psi_Table    : (text.text.public_key. function. message) set,
                    Snd, Rcv     : channel(dy)) played_by LS def=

local   State           : nat,
        Psi_LR, Psi_T   : text,
        K_LR            : public_key,
        DT              : function,
        Loc             : text

init   State := 4

transition

        4.  State = 4 /\ Rcv({Psi_LR'.Psi_T'. K_LR'.K_T. DT'.Loc'.
                              {Psi_LR'.Psi_T'. K_LR'.K_T. DT'.Loc'}_inv(K_T)}_K_LS)
       =|> State':= 6 /\ Psi_Table':= cons(Psi_LR'.Psi_T'.K_LR'.DT'.Loc',Psi_Table)

        6. State  = 6 /\ Rcv({Psi_LR. Psi_T}_K_LS)
                      /\ in(Psi_LR'.Psi_T'. K_LR'. DT. Loc', Psi_Table)
       =|> State':= 9 /\ Snd({{Psi_T'. DT(Loc')}_inv(K_LS)}_K_LR')
                      /\ secret(DT(Loc'), filtered_loc, {LR, LS, T})
                      /\ witness(LS, LR, lr_ls_filtered_loc, DT(Loc'))

end role
```

```
role session(T, LR, LS          : agent,
             K_T, K_LS  : public_key,
             K_T_LR             : symmetric_key,
             Psi_Table          : (text.text.public_key.function.message) set
            ) def=

local SLS, ST, SLR, RLS, RT, RLR: channel(dy)

composition

           locationRecipient(T, LR, LS, K_T, K_LS, K_T_LR, SLR, RLR)
        /\ target           (T, LR, LS, K_T, K_LS, K_T_LR, ST, RT)
```

```
        /\ locationServer   (T, LR, LS, K_LS, K_T, Psi_Table, SLS, RLS)

end role
```

```
role environment() def=

local   Psi_Table:  (text.text. public_key.function.message) set
        % shared between all instances of LS

const   lr_ls_si_t , lr_t_n_lr, k_lr,
        ls_lr_psi_lr_t, t_lr_lr,
        lr_ls_filtered_loc      : protocol_id,
        t, lr, ls               : agent,
        k_t, k_ls, k_i          : public_key,
        k_t_lr,  k_t_i, k_i_lr  : symmetric_key

intruder_knowledge = {t, lr, ls, k_t, k_lr, k_ls, k_i, inv(k_i), k_t_i, k_i_lr}

composition

           session(t, lr, ls,  k_t, k_ls, k_t_lr, Psi_Table)
         % repeat session to check for replay attacks
        /\ session(t, lr, ls,  k_t, k_ls, k_t_lr, Psi_Table)
        /\ session(i, lr, ls,  k_i, k_ls, k_i_lr, Psi_Table)
%       /\ session(t, i,  ls,  k_t, k_ls, k_t_i,  Psi_Table)
%       It does not make sense to let the intruder play the role of LR
%       since then the intruder is allowed to know the (secret) location of T.

end role
```

```
goal

        secrecy_of loc, filtered_loc, psi_t, psi_lr, k_lr % addresses G12

        authentication_on lr_ls_filtered_loc             % addresses G2 and G3
        % authentication and integrity of location object
```

```
        % additional authentication goals, not in RFC3639:
        authentication_on lr_t_n_lr                       % addresses G1 and G3,
           % and G20: T authorizes LR to receive the location via LS
        weak_authentication_on t_lr_lr                    % addresses G1

end goal
```

```
environment()
```

6.3 Pervasive access

Definition Reference

Geographic Location Privacy Requirements: Pervasive Scenarios.
Talk by Jorge Cuellar at Pervasive 2002 in Zurich, `http://www.pervasive2002.org/`

Protocol Purpose

authorization for anonymous access (using a pseudonym of the target)
to location services in a spontaneous place through the Location Beacon Server

Model Authors

Lan Liu for Siemens CT IC 3, May 2005

Alice&Bob style

```
 T : Target
 LoSi : Location Sighter
 LBS : Pervasive-Location Beacon Server (or P-LBS)
 1. T    --------------  {P1_T.Loc}_K_LoSi ----------------------> LoSi
 2. LoSi --------------- {P1_T.Loc}_K_LL ------------------------> LBS
 3. T    <---------- {LBS.N_LBS.Loc}_P1_T ------------------------ LBS
 4. T    ------- P1_T.Psi_T.K_T.Cert_Psi_T.N_LBS.
               {P1_T.Psi_T.K_T.Cert_Psi_T.N_LBS}_inv(K_T) ------> LBS
 %5. T   <----------------   {LBS}_K_T  ----------------------------LBS
 Cert_Psi_T = {Psi_T.Dom_T.K_T}_inv(K_CA)
```

Problems Considered: 2

- secrecy of `loc`
- authentication on `lbs_t_n_lbs`

Problem Classification: G1, G3, G12

Attacks Found: None

Further Notes

We model the authorization of the target by the LBS indirectly by checking the certificate of the target which binds the pseudonym of the target with its domain and its public key. If correct, the LBS can communicate further with the target using the public key.

HLPSL Specification

```
role locationSighter(LoSi, T, LBS       : agent,
                     K_LL               : symmetric_key,
                     K_LoSi, K_LBS      : public_key,
                     Snd, Rcv           : channel(dy))
played_by LoSi def=

local   State           : nat,
        P1_T            : public_key,
        Loc             : text

init State := 1

transition

        1.  State  = 1 /\ Rcv({P1_T'.Loc'}_K_LoSi)
        =|> State':= 3 /\ Snd({P1_T'.Loc'}_K_LL)
                     % /\ witness(LoSi, T, t_losi_loc, Loc')

end role
```

```
role target(LoSi, T, LBS        : agent,
            K_T, K_LBS, K_LoSi  : public_key,
            Psi_T               : text,
            Cert_Psi_T          : {text.text.public_key}_inv(public_key),
            Snd, Rcv            : channel(dy))
played_by T def=

local State             : nat,
      N_LBS             : text,
      Loc               : text,
      P1_T              : public_key

const loc               : protocol_id

init State := 0

transition

        0.  State  = 0 /\ Rcv(start)
        =|> State':= 2 /\ P1_T' := new()
                       /\ Loc' := new()
                       /\ Snd({P1_T'.Loc'}_K_LoSi)
                       /\ secret(Loc, loc, {T, LoSi, LBS})

        2.  State  = 2 /\ Rcv({LBS.N_LBS'.Loc}_P1_T)
        =|> State':= 7
                       /\ Snd(P1_T.Psi_T.K_T.Cert_Psi_T.N_LBS'.
                             {P1_T.Psi_T.K_T.Cert_Psi_T.N_LBS'}_inv(K_T))
                       /\ witness(T, LBS, lbs_t_n_lbs, N_LBS')
                     % /\ wrequest(T, LoSi, t_losi_loc, Loc')

        %7. State = 7 /\ Rcv({LBS}_K_T)
        %=|>State':= 9

end role
```

```
role locationBeaconServer(LoSi, T, LBS  : agent,
                          K_LL                 : symmetric_key,
                          K_LBS, K_CA          : public_key,
```

```
                           Domain              : text,
                           Snd, Rcv            : channel(dy))
played_by LBS def=

local State             : nat,
      Loc, Psi_T, N_T   : text,
      P1_T, K_T         : public_key,
      N_LBS             : text

init State := 4

transition

        4.  State  = 4 /\ Rcv({P1_T'.Loc'}_K_LL)
        =|> State':= 6 /\ N_LBS' := new()
                       /\ Snd({LBS.N_LBS'.Loc'}_P1_T')

        6.  State  = 6
        /\ Rcv(P1_T.Psi_T'.K_T'.{Psi_T'.Domain.K_T'}_inv(K_CA).N_LBS.
              {P1_T.Psi_T'.K_T'.{Psi_T'.Domain.K_T'}_inv(K_CA).N_LBS}_inv(K_T'))
        =|> State':= 8 /\ request(LBS, T, lbs_t_n_lbs, N_LBS)
                     % /\ Snd({LBS}_K_T')

end role
```

```
role session (LoSi, T, LBS             : agent,
              Psi_T                    : text,
              K_T, K_LBS, K_CA, K_LoSi : public_key,
              K_LL                     : symmetric_key,
              Domain                   : text,
              Cert_Psi_T               : {text.text.public_key}_inv(public_key))
def=

local SLBS, ST, SLoSi, RLBS, RT, RLoSi: channel(dy)

composition

           locationSighter(LoSi, T, LBS, K_LL, K_LoSi, K_LBS, SLoSi, RLoSi)
        /\ target(LoSi, T, LBS, K_T, K_LBS, K_LoSi, Psi_T, Cert_Psi_T, ST, RT)
```

```
        /\ locationBeaconServer(LoSi, T, LBS, K_LL, K_LBS,K_CA,Domain,SLBS,RLBS)

end role
```

```
role environment() def=

const   lbs_t_n_lbs, t_losi_loc          : protocol_id,
        t, lbs, losi                     : agent,
        k_t, k_i, k_lbs, k_ca, k_losi    : public_key,
        psi_t, psi_i                     : text,
        dom, dom_i                       : text,
        k_ll                             : symmetric_key

intruder_knowledge = {losi, t, lbs, k_t, k_lbs, k_ca, k_losi,
                      k_i, inv(k_i), psi_i, {psi_i.dom_i.k_i}_inv(k_ca)}

composition

           session(losi, t, lbs, psi_t, k_t, k_lbs, k_ca, k_losi, k_ll,
                                             dom, {psi_t.dom.k_t}_inv(k_ca))
         % repeat session to check for replay attacks
        /\ session(losi, t, lbs, psi_t, k_t, k_lbs, k_ca, k_losi, k_ll,
                                             dom, {psi_t.dom.k_t}_inv(k_ca))
        /\ session(losi, i, lbs, psi_i, k_i, k_lbs, k_ca, k_losi, k_ll,
                                             dom, {psi_i.dom_i.k_i}_inv(k_ca))
% Since the intruder has no certificate of the domain that the LBS knows, the
% LBS does not authorise the intruder and the third sesssion is not executable,
% which is not a failure here.

end role
```

```
goal

        secrecy_of loc                % addresses G12
        authentication_on lbs_t_n_lbs % addresses G1 and G3

      % weak_authentication_on t_losi_loc
```

```
      % it is not important in this protocol to authenticate the LS
      % That's also the reason why we have no session here
      % where the intruder impersonates the location sighter.

end goal
```

```
environment()
```

7 Simple

Protocol Purpose

The Session Initiation Protocol for Instant Messaging and Presence

Definition Reference

- http://www.ietf.org/internet-drafts/draft-ietf-simple-presence-rules-02.txt
- http://www.ietf.org/internet-drafts/draft-ietf-sip-identity-05.txt
- RFC2617 - HTTP Authentication: Basic and Digest Access Authentication
- RFC3261 - SIP: Session Initiation Protocol
- RFC3325 - Private Extensions to the SIP for Asserted Identity

Model Authors

Judson Santiago, LORIA Nancy, June 2005

Alice&Bob style

```
WR --> PS: SUBSCRIBE
PS --> WR: Challenge.realm (a nonce and id of the user domain)
WR --> PS: Hash(Username.Challenge.Password)
PS --> WR: PresenceInfo

WR = Watcher
PS = Presence Server
```

Model Limitations

The protocol has two more agents, besides WR and PS, that do not appear in the specification, namely PT (Presentity) and RM (Rule Maker). The Presentity is the agent about whom the watcher wants to obtain informations. The Rule Maker is the agent that will provide a policy document to the Presence Server, stating the watchers that can obtain the presence information and under what conditions. Both agents were abstracted, the policy document is considered to be already known by the Presence Server because the document is transported using security mechanisms that prevents eavesdropping and interception of the message. We also assume that all watchers want to gather informations about the same Presentity and that they have the rights to do so.

The presence server has two ways to obtain the identity of the watcher, either the watcher authenticates himself in a local proxy, that can assert the watcher identity, and the proxy forwards the subscribe message to the presence server, or the watcher authenticates directly with the presence server. In the former case the local proxy will add a `P-Asserted-identity` field to the message that is forwarded to the presence server. This field can then be used to get the watcher identity and decide if the presence information should or not be granted to the watcher. The latter case, the one that is specified here, is a simpler view of the protocol that consider the presence server can authenticate the watcher identity.

The current specification uses common digest authentication and that implies no replay attack protection.

Problems Considered: 3

- secrecy of `presenceinfo`
- weak authentication on `wr_ps_presenceinfo`
- weak authentication on `ps_wr_user`

Problem Classification: G1 G2 G12

Attacks Found: None

Further Notes

The main concern with the PresenceInfo is its confidentialidy and that its receiver, the Watcher, is authenticated. Here we also analyse the agreement on (and even the freshness of) the PresenceInfo. `wr_ps_info` checks these properties, and the authentication of PS happens as a by-product.

The use of transport or network layer hop-by-hop security mechanisms, such as TLS or IPSec with apropriate cipher suites, should be used to prevent eavesdropping and interception of the final message containing the presence info. Here the message is encrypted with a symmetric key scheme.

A further simplification made to the protocol was the use of the Watcher name as the user name. In the SIP protocol a user can choose a username, via the `P-prefered-identity` field, under which he wants to be authenticated, but that feature do not add any extra security concerns.

HLPSL Specification

```
role watcher (WR, PS   : agent,
              Password : text,
              K        : symmetric_key,
              Hash     : function,
              Realm    : text,
              Snd, Rcv : channel(dy)) played_by WR def=

  local State        : nat,
        Challenge,
        PresenceInfo : text

  init State := 0

  transition

   1.    State  = 0 /\ Rcv(start) =|>
         State':= 1 /\ Snd(subscribe)

   2.    State  = 1 /\ Rcv(Challenge'.Realm) =|>
         State':= 2 /\ Snd(Hash(WR.Challenge'.Password))
                    /\ witness(WR,PS,ps_wr_user,WR.Password)

   3.    State  = 2 /\ Rcv({WR.PresenceInfo'}_K) =|>
         State':= 3 /\ wrequest(WR,PS,wr_ps_presenceinfo,PresenceInfo')

end role
```

```
role pserver (PS       : agent,
              UserMap  : (agent.text.symmetric_key) set,
              Hash     : function,
              Realm    : text,
              Snd, Rcv : channel(dy)) played_by PS def=

  local WR           : agent,
        State        : nat,
        Challenge,
```

```
        Password,
        PresenceInfo : text,
        K            : symmetric_key

  init State := 0

  transition

   1.    State  = 0 /\ Rcv(subscribe) =|>
         State':= 1 /\ Challenge' := new()
                    /\ Snd(Challenge'.Realm)

   2.    State  = 1 /\ Rcv(Hash(WR'.Challenge.Password'))
                    /\ in (WR'.Password'.K', UserMap) =|>
         State':= 2 /\ PresenceInfo' := new()
                    /\ Snd({WR'.PresenceInfo'}_K')
                    /\ secret(PresenceInfo',presenceinfo,{WR',PS})
                    /\ witness(PS,WR',wr_ps_presenceinfo,PresenceInfo')
                    /\ wrequest(PS,WR',ps_wr_user,WR'.Password')

end role
```

```
role session (PS       : agent,
              WR       : agent,
              K        : symmetric_key,
              Password : text,
              Realm    : text,
              H        : function,
              UserMap  : (agent.text.symmetric_key) set,
              Snd,Rcv  : channel (dy)) def=

  composition
    watcher(WR,PS,Password,K,H,Realm,Snd,Rcv) /\
    pserver(PS,UserMap,H,Realm,Snd,Rcv)

end role
```

```
role environment () def=

 local UserMap: (agent.text.symmetric_key) set,
       Snd, Rcv : channel (dy)

 const wr1,wr2,ps,i : agent,
       k1,k2,ki     : symmetric_key,
       h            : function,
       subscribe    : message,
       pass1,pass2,
       passi,domain : text,
       presenceinfo,
       wr_ps_presenceinfo,
       ps_wr_user   : protocol_id

 init
       UserMap := {(wr1.pass1.k1),(wr2.pass2.k2),(i.passi.ki)}

 intruder_knowledge = {wr1,wr2,ps,i,ki,passi,h,subscribe}

 composition

          session(ps,wr1,k1,pass1,domain,h,UserMap,Snd,Rcv)
       /\ session(ps,wr1,k1,pass1,domain,h,UserMap,Snd,Rcv)
       /\ session(ps,wr2,k2,pass2,domain,h,UserMap,Snd,Rcv)
       /\ session(ps,i  ,ki,passi,domain,h,UserMap,Snd,Rcv)

end role
```

```
goal

   % Confidentiality (G12)
   secrecy_of presenceinfo

   % Message authentication (G2)
   weak_authentication_on wr_ps_presenceinfo

   % Entity authentication (G1)
   weak_authentication_on ps_wr_user
```

```
end goal
```

```
environment()
```

8 SPKM-LIPKEY

8.1 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

[Eis00, Ada96]

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: 6

- authentication on `k`
- authentication on `ktrgtint`
- secrecy of `sec_i_Log, sec_i_Pwd,`

Problem Classification: G1, G2, G3, G7, G10

Attacks Found: None

Further Notes

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,Nb        : text,
        Rnumber1     : text,
        X            : message,
        Keycompleted : message,
        W            : nat,
        K            : text.text

  const sec_i_Log, sec_i_Pwd: protocol_id

  init  State := 0

  transition

  1.  State  = 0 /\ RCV(start) =|>
      State':= 1 /\ Na' := new()
                 /\ Rnumber1' := new()
                 /\ 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,sec_i_Log,{S})
                 /\ secret(Pwd_A_S,  sec_i_Pwd,{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,Nb         : text,
        Rnumber2      : text,
        Y             : message,
        Keycompleted : message,
        W             : nat,
        K             : text.text

  const sec_t_Log, sec_t_Pwd: protocol_id

  init  State := 0

  transition

  1. State  = 0 /\ RCV(A.S.Na'.Y'.{A.S.Na'.Y'}_inv(Ka)) =|>
     State':= 1 /\ Nb' := new()
                /\ Rnumber2' := new()
                /\ 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),sec_t_Log,{A})
                /\ secret(Pwd(A,S),  sec_t_Pwd,{A})
```

```
               /\ witness(S,A,ktrgtint,Keycompleted')

  2. 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)
def=

  local   SndI, RcvI,
          SndT, RcvT : channel (dy)
  composition

     initiator(A,S,G,H,Ka,Ks,Login(A,S),Pwd(A,S),SndI,RcvI)
  /\ target(   A,S,G,H,Ka,Ks,Login,Pwd,SndT,RcvT)

end role
```

```
role environment()
def=

  const a,s,i,b: agent,
        ka, ki, kb, ks: public_key,
        login, pwd : function,
        h: function,
        g: nat,
        k,ktrgtint: protocol_id

  intruder_knowledge = {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)
        /\  session(b,s,login,pwd,kb,ks,h,g)
        /\  session(i,s,login,pwd,ki,ks,h,g)

end role
```

```
goal

  %Target authenticates Initiator on k
  authentication_on k % addresses G1, G2, G3
  %Initiator authenticates Target on ktrgtint
  authentication_on ktrgtint % addresses G1, G2, G3

  %secrecy_of Login, Pwd
  secrecy_of sec_i_Log, sec_i_Pwd, % adresses G7, G10
             sec_t_Log, sec_t_Pwd  % adresses G7, G10

end goal
```

```
environment()
```

8.2 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

[Eis00, Ada96]

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: 5

- authentication on `k`
- secrecy of `sec_i_Log, sec_i_Pwd,`

Problem Classification: G1, G2, G3, G7, G10

Attacks Found: None

Further Notes

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,Nb        : text,
        Rnumber1     : text,
        X            : message,
        Keycompleted : message,
        W            : nat,
        K            : text.text

  const sec_i_Log, sec_i_Pwd : protocol_id

  init  State := 0

  transition

  1.  State  = 0 /\ RCV(start) =|>
      State':= 1 /\ Na' := new()
                 /\ Rnumber1' := new()
                 /\ 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,sec_i_Log,{S})
                 /\ secret(Pwd_A_S,sec_i_Pwd,{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,Nb        : text,
        Rnumber2     : text,
        Y            : message,
        Keycompleted : message,
        W            : nat,
        K            : text.text

  const sec_t_Log, sec_t_Pwd : protocol_id

  init  State := 0

  transition

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

```
  local SndI,RcvI : channel (dy),
        SndT,RcvT : channel (dy)

  composition
     initiator(A,S,G,H,Ka,Ks,Login(A,S),Pwd(A,S),SndI,RcvI) /\
     target(   A,S,G,H,Ka,Ks,Login,Pwd,SndT,RcvT)

end role
```

```
role environment()
def=

  const a,s,i,b: agent,
        ka, ki, kb, ks: public_key,
        login, pwd : function,
        h: function,
        g: nat,
        k: protocol_id

  intruder_knowledge = {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)
        /\  session(b,s,login,pwd,kb,ks,h,g)
        /\  session(i,s,login,pwd,ki,ks,h,g)

end role
```

```
goal

  %Target authenticates Initiator on k
  authentication_on k % addresses G1, G2 and G3

  %secrecy_of Login, Pwd
  secrecy_of sec_i_Log, sec_i_Pwd, % adresses G7 and G10
```

```
          sec_t_Log, sec_t_Pwd  % adresses G7 and G10

end goal
```

```
environment()
```

9 (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 `sec_kab1, sec_kab2`
- authentication on `nb`
- authentication 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, Nb : text

  const sec_kab1 : protocol_id

  init  State := 0

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

   2. State   = 1 /\ Rcv(Nb') =|>
      State' := 2 /\ Na' := new() /\ Snd(Na'.H(Kab.Na'.Nb'.A))
                  /\ witness(A,B,na,Na')
                  /\ secret(Kab,sec_kab1,{A,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,
        Na, Nb : text

  const sec_kab2 : protocol_id

  init  State := 0

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

   2. State   = 1 /\ Rcv(Na'.H(Kab.Na'.Nb.A)) =|>
      State' := 2 /\ Snd(H(Kab.Na'))
                  /\ request(B,A,na,Na')
                  /\ secret(Kab,sec_kab2,{A,B})

end role
```

```
role session(A,B: agent,
             Kab: symmetric_key,
             H: function)
def=

  local SA, SB, RA, RB: channel (dy)

  composition
```

```
           chap_Init(A, B, Kab, H, SA, RA)
       /\  chap_Resp(B, A, Kab, H, SB, RB)
end role
```

```
role environment()
def=

  const a, b          : agent,
        kab, kai, kbi : symmetric_key,
        h             : function,
        na, nb        : protocol_id

  intruder_knowledge = {a, b, h, kai, kbi }

  composition
        session(a,b,kab,h) /\
        session(a,i,kai,h) /\
        session(b,i,kbi,h)

end role
```

```
goal

 %secrecy of the shared key
 secrecy_of sec_kab1, sec_kab2

 %CHAP_Init authenticates CHAP_Resp on nb
 authentication_on nb
 %CHAP_Resp authenticates CHAP_Init on na
 authentication_on na

end goal

environment()
```

10 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

- authentication 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

  const
    timestamp : protocol_id

  init
    State := 10

  transition

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

  intruder_knowledge = {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
  authentication_on timestamp
end goal
```

```
environment()
```

11 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 `sec_SK`
- authentication 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

  const  sec_SK : protocol_id

  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,sec_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,
        Auth  : message

  init  State := 0

  transition
   1. State = 0   /\ RCV(A')
      =|>
      State' := 1 /\ Ns' := new()
                  /\ T' := new()
                  /\ 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)
```

```
def=

  local SK: message,
        SNDA, SNDS, RCVA, RCVS: channel (dy)

  init SK = K(A.S)

  composition
       client(A,S,SK,F,SNDA,RCVA)
    /\ server(S,K,F,SNDS,RCVS)

end role
```

```
role environment()
def=

 const a, s : agent,
       k, f : function,
       auth : protocol_id

 intruder_knowledge = {a,s,i,f}

 composition
      session(a,s,k,f)
   /\ session(i,s,k,f)
   /\ session(a,s,k,f)

end role
```

```
goal

  %secrecy_of SK
  secrecy_of sec_SK

  %Server authenticates Client on auth
  authentication_on auth
```

```
end goal
```

```
environment()
```

12 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 key words MAY, MUST etc. 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.

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 `sec_k`
- authentication on `sig`

Problem Classification: G1, G2, G3, G12

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,
        Sig   : message

  const delayedAuthReq : protocol_id,
        succ           : function,   % Successor function
        sec_k          : protocol_id

  init  State := 0

  transition

   1. State = 0
      /\ Rcv(start)
```

```
      =|>
      State' := 1
      /\ Time1' := new()
      /\ 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,sec_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

  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)
def=

  local SA, RA, SB, RB : channel (dy)

  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=

 const a, b       : agent,
       k1, k2, k3 : text,
       h, keyid   : function,
       sig        : protocol_id

 intruder_knowledge = {a,b,k2,i,delayedAuthReq,
                       keyid,h,succ,
                       k3}

 composition
        session(a,b,h,keyid,k1)
    /\  session(a,i,h,keyid,k2)
    /\  session(i,b,h,keyid,k3)
```

```
end role
```

```
goal
  secrecy_of sec_k % addresses G12

  %DHCP_Delayed_Client authenticates DHCP_Delayed_Server on sig
  authentication_on sig % addresses G1, G2, G3
end goal
```

```
environment()
```

13 TSIG

Protocol Purpose

This protocol allows for transaction level authentication using shared secrets and one way hashing. It can be used to authenticate dynamic updates as coming from an approved client, or to authenticate responses as coming from an approved recursive name server.

Definition Reference

[VGEW00]

Model Authors

- Yohan Boichut LIFC-INRIA Besancon
- David Gümbel, Universität Tübingen (Germany), May 2004

Alice&Bob style

```
1. C -> S: TAG1.M1.{H(TAG1.M1).N1}_K
2. S -> C: TAG2.M1.M2.{H(TAG2.M1.M2).N2}_K
```

Model Limitations

Since this protocol can be used to secure any transactions, we assume here that the constant `M1` represents a request of a client and `M2` is a response corresponding to the request. The variables `N1` and `N2` are two timestamps represented here by two nonces.

Problems Considered: 2

- weak authentication on `server_client_k_ab`
- weak authentication on `client_server_k_ba`

Problem Classification: G1, G2

Attacks Found: None

Further Notes

Client is the initiator. He sends a DNS request whose integrity is ensured by {H(M1).N1}_K where K is a shared secret. Server sends a DNS response whose integrity and freshness are ensured as well, by {H(M1.M2).N2}_K.

HLPSL Specification

```
role client (A, S     : agent,
             K        : symmetric_key,
             H        : function,
             M1       : text,
             Tag1,Tag2 :text,
             SND, RCV : channel(dy))
  played_by A def=

  local  State : nat,
         N1, N2, M2 : text
  init State:=0

  transition
    step1. State=0
           /\  RCV(start)
          =|>
           State':=1
           /\ N1':=new()
           /\ SND(Tag1.M1.{H(Tag1.M1).N1'}_K)
           /\ witness(A,S,server_client_k_ab,Tag1.M1.{H(Tag1.M1).N1'}_K)

    step2. State=1
           /\ RCV(Tag2.M1.M2'.{H(Tag2.M1.M2').N2'}_K)
          =|>
           State':=2
           /\ wrequest(A,S,client_server_k_ba,Tag2.M1.M2'.{H(Tag2.M1.M2').N2'}_K)
```

```
end role
```

```
role server(S         : agent,
            A         : agent,
            K         : symmetric_key,
            H         : function,
            M2        : text,
            Tag1,Tag2: text,
            SND, RCV : channel(dy))
  played_by S def=

  local State : nat,
        N1,M1,N2 : text
  init State:=0

  transition
    step1. State=0
           /\ RCV(Tag1.M1'.{H(Tag1.M1').N1'}_K)
          =|>
           State':=1
           /\ N2':=new()
           /\ SND(Tag2.M1'.M2.{H(Tag2.M1'.M2).N2'}_K)
           /\ witness(S,A,client_server_k_ba,Tag2.M1'.M2.{H(Tag2.M1'.M2).N2'}_K)
           /\ wrequest(S,A,server_client_k_ab,Tag1.M1'.{H(Tag1.M1').N1'}_K)
end role
```

```
role session(A,S          : agent,
             K            : symmetric_key,
             M1,M2        : text,
             H            : function,
             Tag1,Tag2    : text,
             Se,Re,Sf,Rf : channel(dy)) def=

  const server_client_k_ab, client_server_k_ba,

  composition
     client(A,S,K,H,M1,Tag1,Tag2,Se,Re)
```

```
  /\ server(S,A,K,H,M2,Tag1,Tag2,Sf,Rf)

end role
```

```
role environment() def=

  local Ra,Rs,Sa,Ss,Si,Ri : channel(dy)

  const a,s,i          : agent,
        kia,kis,kas        : symmetric_key,
        m1,m2,mi1,mi2,tag1,tag2 : text,
        h              : function

  intruder_knowledge = {i,a,s,h,kia,kis,mi1}

  composition
            session(a,s,kas,m1,m2,h,tag1,tag2,Sa,Ra,Ss,Rs)
        /\  session(a,s,kas,m1,m2,h,tag1,tag2,Sa,Ra,Ss,Rs)
        /\  session(i,s,kis,m1,m2,h,tag1,tag2,Si,Ri,Ss,Rs)
        /\  session(a,i,kia,m1,m2,h,tag1,tag2,Si,Ri,Ss,Rs)

end role
```

```
goal
  weak_authentication_on server_client_k_ab % addresses G1,G2
  weak_authentication_on client_server_k_ba % addresses G1,G2
end goal

environment()
```

14 EAP: Extensible Authentication Protocol

14.1 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

```
The protocol exchanges messages between a peer P, e.g. an UMTS subscriber
identity module, and an authentication (EAP-)server S. Before the protocol
starts, S has obtained an authentication vector
   AV = (AT_RAND,AT_AUTN,AT_RES,IK,CK)
from the home environment (HE) of the peer P.
For constructing AV, HE/S and P share the following data/functions:
   SK          : symmetric key   (long term secret)
   SQN         : sequence number (unique to a session)
   F1, F2      : authentication functions
   F3, F4, F5  : key generation functions
The AV-components are computed by
   AT_RAND     : a unique random number
   CK          : F3(SK,AT_RAND)
   IK          : F4(SK,AT_RAND)
   AK          : F5(SK,AT_RAND)
   MAC         : F1(SK,SQN,AT_RAND)
   AT_AUTN     : {SQN}_AK . MAC
   AT_RES      : F2(SK,AT_RAND)
```

```
S -> P: request_id
P -> S: respond_id.NAI
        % NAI is Network Address Identifier.
        % S uses authentication vector AV = (AT_RAND,AT_AUTN,AT_RES,IK,CK)
        % S computes message authentication code for next message:
        %    AT_MAC = HMAC(PRF_SHA1(NAI,IK,CK),AT_RAND.AT_AUTN)
S -> P: AT_RAND.AT_AUTN.AT_MAC
        % P checks validity of AT_MAC,AT_AUTN and SQN
        % P computes AT_RES = F2(SK,AT_RAND), IK, CK
        % P computes message authentication code for next message:
        %    AT_MAC = HMAC(PRF_SHA1(NAI,IK,CK),AT_RES)
P -> S: AT_RES.AT_MAC
        % S checks validity of AT_MAC,AT_RES
S -> P: success
        % S,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 server S combines the (logically) different roles of the home environment HE, the network access server NAS and the EAP server.
- The modeller has to take care that each session gets a unique sequence number SQN
- No synchronization of SQN (in case the peer decides SQN is invalid).
- No resumption of a previous session.

Problems Considered: 3

- secrecy of `sec_ck1, sec_ck2, sec_ik1, sec_ik2`
- authentication on `at_rand`
- authentication 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.

HLPSL Specification

```
role peer (
    P,S                : agent,
    F1,F2,F3,F4,F5     : function,
    PRF_SHA1, HMAC     : function,
    SK                 : symmetric_key,
    SQN                : text,
    SND,RCV            : channel (dy))
played_by P def=

  local
    AT_RAND            : text,
    NAI                : text,
    AT_MAC1, AT_MAC2   : message,
    AT_RES, AT_AUTN    : message,
    IK, CK             : message,
    State              : nat

  const
    request_id,
```

```
    respond_id,
    success      : text,
    sec_ck1, sec_ik1,
    at_rand,
    at_rand2     : protocol_id

  init
    State := 0

  transition

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

  2. State   = 2
       /\ RCV(AT_RAND'.AT_AUTN'.AT_MAC1')
       /\ AT_AUTN'  = {SQN}_F5(SK.AT_RAND').F1(SK.SQN.AT_RAND')
       /\ CK'       = F3(SK,AT_RAND')
       /\ IK'       = F4(SK,AT_RAND')
       /\ AT_MAC1'  = HMAC(PRF_SHA1(NAI,IK',CK'),AT_RAND'.AT_AUTN')
     =|>
     State' := 4
       /\ AT_RES'  := F2(SK.AT_RAND')
       /\ AT_MAC2' := HMAC(PRF_SHA1(NAI,IK',CK'),AT_RES')
       /\ SND(AT_RES'.AT_MAC2')
       /\ request(P,S,at_rand,AT_RAND')
       /\ witness(P,S,at_rand2,AT_RAND')
       /\ secret(CK',sec_ck1,{S,P})
       /\ secret(IK',sec_ik1,{S,P})

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

end role
```

```
role server (
    P,S                 : agent,
    F1,F2,F3,F4,F5      : function,
    PRF_SHA1, HMAC      : function,
    SK                  : symmetric_key,
    SQN                 : text,
    SND,RCV             : channel (dy))
played_by S def=

  local
    AT_RAND             : text,
    NAI                 : text,
    AT_MAC1, AT_MAC2 : message,
    AT_RES, AT_AUTN  : message,
    IK, CK              : message,
    State               : nat

  const
    request_id,
    respond_id,
    success      : text,
    sec_ck2, sec_ik2,
    at_rand,
    at_rand2     : protocol_id

  init
    State := 1

  transition

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

  2. State   = 3
       /\ RCV(respond_id.NAI')
     =|>
     State' := 5
       /\ AT_RAND' := new()
```

```
      /\ AT_AUTN' := {SQN}_F5(SK.AT_RAND').F1(SK.SQN.AT_RAND')
      /\ CK'      := F3(SK,AT_RAND')
      /\ IK'      := F4(SK,AT_RAND')
      /\ AT_MAC1' := HMAC(PRF_SHA1(NAI',IK',CK'),AT_RAND'.AT_AUTN')
      /\ SND(AT_RAND'.AT_AUTN'.AT_MAC1')
      /\ witness(S,P,at_rand,AT_RAND')
      /\ secret(CK',sec_ck2,{S,P})
      /\ secret(IK',sec_ik2,{S,P})

  3. State   = 5
      /\ RCV(AT_RES'.AT_MAC2')
      /\ AT_RES'  = F2(SK.AT_RAND)
      /\ AT_MAC2' = HMAC(PRF_SHA1(NAI,IK,CK),AT_RES')
     =|>
     State' := 7
      /\ SND(success)
      /\ request(S,P,at_rand2,AT_RAND)

end role
```

```
role session(
    P,S                : agent,
    F1,F2,F3,F4,F5     : function,
    PRF_SHA1, HMAC     : function,
    SK                 : symmetric_key,
    SQN                : text)
def=

local
    SNDP, RCVP, SNDS, RCVS : channel (dy)

const
    at_rand,at_rand2       : protocol_id

  composition
     peer(  P,S,F1,F2,F3,F4,F5,PRF_SHA1,HMAC,SK,SQN,SNDP,RCVP)
  /\ server(P,S,F1,F2,F3,F4,F5,PRF_SHA1,HMAC,SK,SQN,SNDS,RCVS)

end role
```

```
role environment() def=

  const
    p,s                     : agent,
    kps,kis                 : symmetric_key, % !!one per user   !!
    sqnp1, sqnp2, sqni      : text,          % !!one per session!!
    f1,f2,f3,f4,f5          : function,
    prf_sha1, hmac          : function

  intruder_knowledge = {p,s,i,f1,f2,f3,f4,f5,prf_sha1,hmac
                       }

  composition
     session(p,s,f1,f2,f3,f4,f5,prf_sha1,hmac,kps,sqnp1)
  /\ session(p,s,f1,f2,f3,f4,f5,prf_sha1,hmac,kps,sqnp2)
% /\ session(i,s,f1,f2,f3,f4,f5,prf_sha1,hmac,kis,sqni)

end role
```

```
goal

  %secrecy_of CK, IK
  secrecy_of sec_ck1, sec_ck2, sec_ik1, sec_ik2

  %Peer authenticates Server on at_rand
  authentication_on at_rand
  %Server authenticates Peer on at_rand2
  authentication_on at_rand2

end goal
```

```
environment()
```

14.2 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.

Note: the original draft has expired. The new version is EAP-PSK.

Definition Reference

- `http://www.ietf.cnri.reston.va.us/internet-drafts/draft-bersani-eap-psk-06.txt` (new version EAP-PSK)
- `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: request_id
P -> S: respond_id.P
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

- authentication on `sd`
- authentication on `na`

- authentication on `bind`
- authentication on `np`
- secrecy of `sec_na, sec_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)
 - Bind: addressing information (address_of_peer,address_of_server)
```

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,
    Na,Sd        : text,          % Sd (=SessionID)
```

```
    State        : nat,
    EMK          : message

  const
    request_id,
    respond_id    : text,
    sec_np,
    na,np,sd,bind : protocol_id

  init
    State := 0

  transition

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

 1. State   = 1 /\ RCV(S.Sd') =|>
    State' := 2 /\ Np'   := new()
                /\ Bind' := new()
                /\ SND(Sd'.P.
                       {Np'}_KEK.Bind'.
                       MAC(KCK.S.Sd'.P.{Np'}_KEK.Bind'))
                /\ secret(Np',sec_np,{P,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')

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,
    State           : nat,
    EMK             : message

  const
    request_id,
    respond_id     : text,
    sec_na,
    na,np,sd,bind : protocol_id

  init
    State := 0

  transition

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

 1. State   = 1 /\ RCV(respond_id.P) =|>
    State' := 3 /\ Sd' := new()
                /\ 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 /\ Na' := new()
                /\ 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',sec_na,{P,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)
def=

  local Speer,Rpeer,Sserver,Rserver : channel (dy)

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

end role
```

```
role environment()
def=

  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

  intruder_knowledge = {s,p,mac,prf}

  composition

     session(s,p,mac,prf,kek,kck,kdk)
  /\ session(s,p,mac,prf,kek,kck,kdk)

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
  authentication_on sd
  %Peer authenticates Server on na
  authentication_on na
  %Server authenticates Peer on bind
  authentication_on bind
  %Server authenticates Peer on np
  authentication_on np

  %secrecy_of Na, Np
  secrecy_of sec_na, sec_np

end goal
```

```
environment()
```

14.3 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
- http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-17.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
S     : server (Network Access Server NAS + Authentication Server AS)

SAi1  : cryptographic algorithms (from S)
DHi   : S's Diffie-Hellman exponent (nonce of S)
KEi   : exp(g,DHi), i.e. S's Diffie-Hellman value
Ni    : nonce of S

SAr1  : P's selected cryptographic algorithms, here SAr1 = SAi1
DHr   : P's Diffie-Hellman exponent (nonce of P)
KEr   : exp(g,DHr), i.e. P's Diffie-Hellman value
Nr    : nonce of P

SK    : PRF(Ni.Nr.exp(KEr,DHi))
        Note: SK == PRF(Ni.Nr.exp(KEi,DHr))

AUTHi : SAi1.KEi.Ni.Nr
AUTHr : SAi1.KEr.Nr.Ni

P <- S: request_id
P -> S: respond_id.ID
P <- S: SAi1, KEi, Ni
P -> S: SAr1, KEr, Nr
P <- S: {S, {AUTHi}_inv(Ks)}_SK
P -> S: {P, {AUTHr}_inv(Kp)}_SK
P <- S: success
```

Model Limitations

- The server S combines the (logically) different roles of the network access server NAS and the authentication server AS.

Problems Considered: 3

- secrecy of `sec_sk1, sec_sk2`
- authentication on `nr`
- authentication on `ni`

Attacks Found: None

HLPSL Specification

```
role peer(P,S      : agent,
          G        : text,
          PRF      : function,
          Kp, Ks   : public_key,
          SND, RCV : channel (dy))
played_by P def=

  local
    Ni, SAi1 : text,
    KEi      : message,
    Nr, DHr  : text,
    SK       : message,
    State    : nat

  const
    request_id : text,
    respond_id : text,
    success    : text,
    sec_sk1,
    ni, nr     : protocol_id

  init
    State := 0

  transition
```

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

  2. State   = 2
       /\ RCV(SAi1'.KEi'.Ni')
     =|>
     State' := 4
       /\ DHr' := new()
       /\ Nr'  := new()
       /\ SND(SAi1'.exp(G,DHr').Nr')
       /\ SK' := PRF(Ni'.Nr'.exp(KEi',DHr'))
       /\ secret(SK',sec_sk1,{S,P})
       /\ witness(P,S,nr,Nr')

  % As opposed to IKEv2, in EAP-IKEv2 there is no negotiation of a
  % CHILD_SA => no second SA payload and no traffic selector payload
  4. State   = 4
       /\ RCV({S.{SAi1.KEi.Ni.Nr}_inv(Ks)}_SK)
     =|>
     State' := 6
       /\ SND({P.{SAi1.exp(G,DHr).Nr.Ni}_inv(Kp)}_SK)
       /\ request(P,S,ni,Ni)

end role
```

```
role server(
          P,S      : agent,
          G        : text,
          PRF      : function,
          Kp, Ks   : public_key,
          SND, RCV : channel (dy))
played_by S def=

  local
    Nr            : text,
```

```
  SAi1, DHi, Ni : text,
  KEr           : message,
  SK            : message,
  X,Z           : message,
  State         : nat

const
  request_id : text,
  respond_id : text,
  success    : text,
  sec_sk2,
  ni, nr     : protocol_id

init
  State := 0

transition

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

1. State   = 1
     /\ RCV(respond_id.P)
   =|>
   State' := 2
     /\ SAi1' := new()
     /\ DHi'  := new()
     /\ Ni'   := new()
     /\ SND(SAi1'.exp(G,DHi').Ni')
     /\ witness(S,P,ni,Ni')

% 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   = 2
     /\ RCV(SAi1.KEr'.Nr')
   =|>
   State' := 3
     /\ SK' := PRF(Ni.Nr'.exp(KEr', DHi))
```

```
       /\ SND({S.{SAi1.exp(G,DHi).Ni.Nr'}_inv(Ks)}_SK')
       /\ secret(SK',sec_sk2,{S,P})

  3. State   = 3
       /\ RCV({P.{SAi1.KEr'.Nr.Ni}_inv(Kp)}_SK)
     =|>
     State' := 4
       /\ SND(success)
       /\ request(S,P,nr,Nr)

end role
```

```
role session(
       P, S   : agent,
       G      : text,
       PRF    : function,
       Kp, Ks : public_key)
def=

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

  composition
       peer(  P,S, G,PRF, Kp, Ks, S1, R1)
    /\ server(P,S, G,PRF, Kp, Ks, S2, R2)

end role
```

```
role environment() def=

  const
    p, s        : agent,
    g           : text,
    f           : function,
    kp, ks      : public_key

  intruder_knowledge = {p,s,f}
```

```
  composition
     session(p,s,g,f,kp,ks)
  /\ session(p,s,g,f,kp,ks)

end role
```

```
goal

   %secrecy_of SK
   secrecy_of sec_sk1, sec_sk2

   %Server authenticates Peer on nr
   authentication_on nr
   %Peer authenticates Server on ni
   authentication_on ni

end goal
```

```
environment()
```

14.4 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

```
S -> P: request_id
        % P gets his UserId = IMSI/TMSI
        % here, we use UserID = P
P -> S: respond_id.UserID
        % S sends a list of supported versions
        % and P must agree of one of them
        % here, we assume only one version
S -> P: request_sim_start.Version
        % P generates a nonce Np
P -> S: respond_sim_start.Version.Np
        % S uses an authentication triplet (Rand.SRES.Kc) which it has
        % previously obtained from some authentication center AuC
        % in the home environment HE of P. Here we have
        %    SRES = A3(Kps,Rand)
        %    Kc   = A8(Kps,Rand)
        % where Kps is some long term secret symmetric key shared by P and
        % AuC/S, and A3,A8 are some known one-way functions.
        % S computes a message authentication code MAC1 via
        %    MK   = SHA1(P,Kc,Np,Version)
        %    Mac1 = MAC1(MK,Rand.Np)
        % MK is a master key which will be used for generating keys for
        % encryption, authentication and data-integrity.
S -> P: request_sim_challenge.Rand.Mac1
        % P checks validity of Mac1
        % P computes SRES = A3(Kps,Rand) and MAC2(MK,SRES)
P -> S: respond_sim_challenge.Mac2
        % S checks Mac2 and thus authenticates P.
S -> P: request_success
```

Model Limitations

- The server S combines the (logically) different roles of the home environment HE, the network access server NAS and the EAP server.

- No resumption of a previous session.

Problems Considered: 3

- secrecy of `sec_mk1, sec_mk2`
- authentication on `mac1`
- authentication 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, S                  : agent,
           Kps                   : symmetric_key,
           SHA1,A3,A8,MAC1,MAC2 : function,
           SND, RCV              : channel (dy))
played_by P 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         % random number
```

```
  const   sec_mk1, 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

  transition

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

  2. State    = 2 /\ RCV(request_sim_start.Ver')
     =|>
     State' := 4 /\ Np' := new()
                 /\ SND(respond_sim_start.Ver'.Np')

  4. State    = 4 /\ RCV(request_sim_challenge.Rand'.Mac1')
                 /\ Kc'    = A8(Kps,Rand')
                 /\ MK'    = SHA1(P,Kc',Np,Ver)
                 /\ Mac1'  = MAC1(MK',Rand'.Np)
     =|>
     State' := 6 /\ SRES' := A3(Kps,Rand')
                 /\ Mac2' := MAC2(MK',SRES')
                 /\ SND(respond_sim_challenge.Mac2')
                 /\ request(P,S,mac1,Mac1')
                 /\ witness(P,S,mac2,Mac2')
                 /\ secret(MK',sec_mk1,{S,P})

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

end role
```

```
role server (P, S                  : agent,
             Kps                   : symmetric_key,
```

```
            SHA1,A3,A8,MAC1,MAC2 : function,
            SND, RCV             : channel (dy))
played_by S 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          % random number

  const  sec_mk2, 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

  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 /\ Rand' := new()
                 /\ Kc'   := A8(Kps,Rand')
                 /\ MK'   := SHA1(P,Kc',Np',Ver)
                 /\ Mac1' := MAC1(MK',Rand'.Np')
                 /\ SND(request_sim_challenge.Rand'.Mac1')
                 /\ witness(S,P,mac1,Mac1')

  7. State   = 7 /\ RCV(respond_sim_challenge.Mac2')
                 /\ Mac2'  = MAC2(MK,SRES')
```

```
                 /\ SRES'  = A3(Kps,Rand)
     =|>
     State' := 9 /\ SND(request_success)
                 /\ secret(MK, sec_mk2, {S,P})
                 /\ request(S,P,mac2,Mac2')

end role
```

```
role session(P, S                 : agent,
             Kps                  : symmetric_key,
             SHA1,A3,A8,MAC1,MAC2 : function)
def=

   local
     SNDP, RCVP, SNDS, RCVS : channel (dy)

  composition
           peer(  P,S,Kps,SHA1,A3,A8,MAC1,MAC2,SNDP,RCVP)
        /\ server(P,S,Kps,SHA1,A3,A8,MAC1,MAC2,SNDS,RCVS)

end role
```

```
role environment() def=

   const
     p, s               : agent,
     kps, kpi, kis      : symmetric_key,
     sha1,a3,a8,mc1,mc2 : function

   intruder_knowledge = {p, s, sha1, a3, a8, mc1, mc2,
                         kpi,
                         kis
                        }

   composition
        session(p,s,kps, sha1,a3,a8,mc1,mc2)
    /\  session(p,i,kpi, sha1,a3,a8,mc1,mc2)
```

```
    /\  session(i,s,kis, sha1,a3,a8,mc1,mc2)

end role
```

```
goal
        %secrecy_of MK
        secrecy_of sec_mk1, sec_mk2

        %Peer authenticates Server on mac1
        authentication_on mac1
        %Server authenticates Peer on mac2
        authentication_on mac2

end goal
```

```
environment()
```

14.5 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         : {P.Kp}_inv(Kca)
Server_certificate         : {S.Ks}_inv(Kca)
Server_key_exchange        : <not needed for public key encryption>
Client_key_exchange        : {PMS}_Ks with pre-master-secret PMS (nonce of P)
Client_certificate_verify  : {H(Np.Ns.S.PMS)}_inv(Kp)
Change_cipher_spec         : text
Server_hello_done          : text
Finished                   : encrypted hash of all previous messages with
                             master secret PRF(PMS,Np,Ns)

S -> P: request_id
P -> S: respond_id.UserId
S -> P: start_tls
P -> S: Client_hello
S -> P: Server_hello,
        Server_certificate,
        Server_key_exchange,
        Server_certificate_request,   % only if authentication of P required
        Server_hello_done
P -> S: Client_certificate,           % only if authentication of P required
        Client_key_exchange,
        Client_certificate_verify,    % only if authentication of P required
        Change_cipher_spec,
        Finished
S -> P: Change_cipher_spec,
        Finished
```

Model Limitations

- The server S combines the (logically) different roles of the network access server NAS and the EAP server.
- no modelling of session-resumption
- only public key encryption in TLS

Problems Considered: 3

- secrecy of `sec_clientK, sec_serverK`
- authentication on `nps1`
- authentication 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,
         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 sec_clientK,
        sec_serverK,
        nps1, nps2 : protocol_id,
        sid0       : text,  % session id = 0
        request_id : text,
        respond_id : text,
        start_tls  : text

  init State := 0

  transition

  0. State  = 0 /\ RCV_S(request_id) =|>
     State':= 2 /\ SND_S(respond_id.P)

  2. State  = 2 /\ RCV_S(start_tls) =|>
     State':= 4 /\ Np'   := new()
                /\ Csus' := new()
                /\ 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
```

```
      /\ PMS'      := new()
      /\ 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(Np.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
      /\ PMS'      := new()
      /\ 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,sec_clientK,{P,S})
                /\ secret(ServerK,sec_serverK,{P,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,
         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
        request_id : text,
        respond_id : text,
        start_tls  : text

  init State := 1

  transition

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

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

  % with client authentication
  51. State  = 5 /\ RCV_P(TNo'.sid0.Np'.Csus')     % client hello
     =|>
      State':= 7
       /\ Ns'   := new()
       /\ SeID' := new()
       /\ 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(Np.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)
def=

  local SP, SS, RP, RS : channel (dy)

  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=

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

   intruder_knowledge = {p,s, h,prf,keygen, kp,ks,kca,ki,inv(ki),
                         {i.ki}_inv(kca)
                        }

   composition
        session(p,s,kp,ks,kca,h,prf,keygen)
%   /\  session(p,i,kp,ki,kca,h,prf,keygen)
    /\  session(i,s,ki,ks,kca,h,prf,keygen)

end role

------------------------------------------------------------------------

goal
```

```
        %secrecy_of ClientK, ServerK
        secrecy_of sec_clientK, sec_serverK

        %Peer authenticates Server on nps1
        authentication_on nps1
        %Server authenticates Peer on nps2
        authentication_on nps2

end goal
```

```
environment()
```

14.6 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

```
The protocol involves 4 (logically) different agents: the client (here
called peer P in acc. to previous EAP-modelling), the access point NAS,
the TTLS server and the AAA/H server (in user's home domain). Here, we
join the last 3 agents into the role of server S.

P <- S : request_id
P -> S : respond_id.UserId

1st phase: TLS
P <- S : start_ttls
P -> S : Version.SessionID.Np.CipherSuite   % client_hello
P <- S : Version.SessionID.Ns.Cipher        % server_hello
         {S.Ks}_inv(Kca)                    % certificate
         Ske                                % server_key_exchange, not needed
                                            % for public key encryption
         Shd                                % server_hello_done  (text)
P -> S : {PMS}_Ks                           % client_key_exchange
         Ccs                                % change_cipher_spec (text)
         {Finished}_ClientK                 % finished
P <- S : Ccs                                % change_cipher_spec (text)
         {Finished}_ServerK                 % finished

2nd phase: using tunneling to authenticate peer
P -> S : {UserName,
          CHAP_challenge,
          CHAP_Password
         }_ClientK
P <- S : success

with
CipherSuite:    set of cipher suites supplied by P (for EAP-TLS)
Cipher:         cipher suite selected by S (from CipherSuite)
Np:             nonce created by P
```

```
Ns:             nonce created by S
Ks:             public key of S
Kca:            public key of certification authority
PMS:            pre-master-secret created by P (nonce)
MS:             master-secret ( =PRF(PMS,Np,Ns) )
Finished:       hash(MS,<all previous messages>)
ClientK:        session key for client =KeyGen(P.Np.Ns.MS)
ServerK:        session key for server =KeyGen(S.Np.Ns.MS)
CHAP_challenge: Tranc(CHAP_PRF(M.Txt.Np.Ns).1.16)
ChapId:         Tranc(CHAP_PRF(M.Txt.Np.Ns).17.17)
CHAPRs:         Chap response
CHAP_Password:  ChapId + ChapRs
```

Model Limitations

- The server S combines the (logically) different roles of the access point NAS, TTLS server and the AAA/H server (in user's home domain).
- No authentication of client in TLS.
- Only public key encryption in TLS.
- Selection of cipher suites only abstractly modelled.

Problems Considered: 3

- secrecy of `sec_clientK, sec_serverK, sec_uname`
- authentication on `ns`
- authentication on `np`

Attacks Found: None

Further Notes

- The role of the NAS is redundant since it mostly only forwards received messages. There are only two situations where the NAS deviates from forwarding messages: it takes part in the negotiation of a ciphersuite (which is only abstractly modelled anyway) and finally receives keying material for deriving keys to be used at some later time (which does not concern the security aspects of this protocol).

Note: 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, the NAS role is redundant.

HLPSL Specification

```
role peer(P, S                              : agent,
          Kca                               : public_key,
          H, PRF, CHAP_PRF, Tranc, KeyGen : function,
          SND, RCV                          : channel (dy))
played_by P def=

  local
    UserId        : text,          % should not reveal user
    Version       : text,          % version of TLS protocol, presently v1.0
    SeID          : text,          % session id
    Np            : text,          % nonce from client
    Ns            : text,          % nonce from server
    CipherSuite   : text,          % TLS ciphersuites supplied by the peer
    Cipher        : text,          % TLS ciphersuite selected by server

    Ks            : public_key,    % from server

    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

  ChapRs       : text,          % CHAP response

  State        : nat

const
  request_id   : text,
  respond_id   : text,
  start_ttls   : text,
  success      : text,
  sec_clientK,
  sec_serverK,
  sec_uname,
  np, ns       : protocol_id

init State := 0

transition

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

1. State   = 1
     /\ RCV(start_ttls)
   =|>
   State' := 2
     /\ Np' := new()
     /\ SND(Version'.SeID'.Np'.CipherSuite') % client_hello
     /\ witness(P,S,np,Np')

2. State   = 2
     /\ RCV(Version.SeID'.Ns'.Cipher'.       % server_hello
            {S.Ks'}_inv(Kca).                % server_certificate
            Shd')                            % server_hello_done
   =|>
   State' := 3
     /\ PMS' := new()
```

```
      /\ MS'  := PRF(PMS'.Np.Ns')             % master secret
      /\ Finished' := H(MS'.P.S.Np.Cipher'.SeID)
      /\ ClientK'  := KeyGen(P.Np.Ns'.MS')
      /\ ServerK'  := KeyGen(S.Np.Ns'.MS')
      /\ SND({PMS'}_Ks'.                      % client_key_exchange
             Ccs'.                            % client_change_cipher_spec
             {Finished'}_ClientK')            % finished
      /\ secret(ClientK',sec_clientK,{P,S})
      /\ secret(ServerK',sec_serverK,{P,S})

  3. State   = 3
      /\ RCV(Ccs.{Finished}_ServerK)
    =|>
    State' := 4
      /\ Txt' := new()
      /\ SND({UName'.
             Tranc(CHAP_PRF(MS.Txt'.Np.Ns).1.16).
             Tranc(CHAP_PRF(MS.Txt'.Np.Ns).17.17).
             ChapRs'
            }_ClientK)
      /\ secret(UName',sec_uname,{P,S})
      /\ request(P,S,ns,Ns)

  4. State   = 4
      /\ RCV(success)
    =|>
    State' := 5

end role
```

```
role server (P, S                            : agent,
             Ks, Kca                         : public_key,
             H, PRF, CHAP_PRF, Tranc, KeyGen : function,
             SND, RCV                        : 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
  Np            : text,          % nonce from client
  Ns            : text,          % nonce from server
  CipherSuite   : text,          % TLS ciphersuites supplied by the peer
  Cipher        : text,          % TLS ciphersuite selected by server

  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

  ChapRs        : text,          % CHAP response

  State         : nat

const
  request_id    : text,
  respond_id    : text,
  start_ttls    : text,
  success       : text,
  np, ns        : protocol_id

init State := 0

transition

0. State   = 0
     /\ RCV(start)
   =|>
   State' := 1
     /\ SND(request_id)
```

```
  1. State   = 1
       /\ RCV(respond_id.UserId')
     =|>
     State' := 2
       /\ SND(start_ttls)

  2. State   = 2
       /\ RCV(Version'.SeID'.Np'.CipherSuite')  % client_hello
     =|>
     State' := 3
       /\ Ns' := new()
       /\ SND(Version'.SeID'.Ns'.Cipher'.       % server_hello
              {S.Ks}_inv(Kca).                  % server_certificate
              Shd')                             % server_hello_done
       /\ witness(S,P,ns,Ns')

  3. State    = 3
       /\ RCV({PMS'}_Ks.                        % client_key_exchange
              Ccs'.                             % client_change_cipher_spec
              {Finished'}_ClientK')             % finished
       /\ MS' = PRF(PMS'.Np.Ns)                 % master secret
       /\ Finished' = H(MS'.P.S.Np.Cipher'.SeID)
       /\ ClientK'  = KeyGen(P.Np.Ns.MS')
     =|>
     State' := 4
       /\ ServerK'  := KeyGen(S.Np.Ns.MS')
       /\ SND(Ccs'.                             % server_change_cipher_spec
              {Finished'}_ServerK')             % finished

  4. State  = 4
       /\ RCV({UName'.
               Tranc(CHAP_PRF(MS.Txt'.Np.Ns).1.16).
               Tranc(CHAP_PRF(MS.Txt'.Np.Ns).17.17).
               ChapRs'
              }_ClientK)
     =|>
     State':= 5
       /\ SND(success)
       /\ request(S,P,np,Np)

end role
```

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

  local
    SNDP, RCVP, SNDS, RCVS  : channel (dy)

  composition
           peer(  P,S,   Kca,H,PRF,CHAP_PRF,Tranc,KeyGen,SNDP,RCVP)
        /\ server(P,S,Ks,Kca,H,PRF,CHAP_PRF,Tranc,KeyGen,SNDS,RCVS)

end role
```

```
role environment() def=

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

   intruder_knowledge = {p,s, ks,kca,
                         h,prf,chapprf,tranc,keygen,
                         kca
                        }

   composition
        session(p,s,ks,kca,h,prf,chapprf,tranc,keygen)
%    /\ session(p,s,ks,kca,h,prf,chapprf,tranc,keygen)
     /\ session(i,s,ks,kca,h,prf,chapprf,tranc,keygen)

end role
```

```
goal
```

```
        %secrecy_of ClientK, ServerK, UName
        secrecy_of sec_clientK, sec_serverK, sec_uname

        %Peer authenticates Server on ns
        authentication_on ns
        %Server authenticates Peer on np
        authentication_on np

end goal
```

```
environment()
```

14.7 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 `sec_clientK, sec_serverK`
- authentication on `np_ns`
- authentication 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,
        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,
        State: nat

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

  %owns SND_S
  init State := 0

  transition

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

  2. State  = 2 /\ RCV_S(start_peap) =|>
```

```
    State':= 4 /\ Np' := new()
               /\ SND_S(Np'.SeID'.Csu')

  3. State  = 4 /\ RCV_S(Ns'.SeID'.Csu'.{S.Ks'}_inv(Kca)) =|>
     State':= 6 /\ PMS' := new()
               /\ 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'))

  4. State  = 6 /\ RCV_S(Ccs.{Finished}_ServerK) =|>
     State':= 8 /\ SND_S({P}_ClientK)
               /\ secret(ClientK,sec_clientK,{P,S})
               /\ secret(ServerK,sec_serverK,{P,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 (P, S                 : agent,
             H1, H2, PRF, KeyGen : function,
             Pw                  : symmetric_key,
```

```
             Kca, Ks              : public_key,
             SND_P, RCV_P         : channel (dy))
played_by S def=

  local Ns: text,
        Np, SeID, Csu, PMS: text,
        Ccs: text,
        M, Finished, ClientK, ServerK: message,
        State: nat

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

  %owns SND_P
  init State = 1

  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 /\ Ns' := new()
                /\ SND_P(Ns'.SeID'.Csu'.{S.Ks}_inv(Kca))
                /\ witness(S,P,np_ns,Np'.Ns')

  4. State  = 7 /\ RCV_P({PMS'}_Ks.Ccs') =|>
     State':= 9 /\ 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))

  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)
def=

  local S_SP,R_SP,S_PS,R_PS : channel (dy)

  composition
           peer(  P,S,H1,H2,PRF,KeyGen,Pw,Kca,   S_SP,R_SP)
        /\ server(P,S,H1,H2,PRF,KeyGen,Pw,Kca,Ks,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

   intruder_knowledge = {p,s, h1,h2,prf,keygen,
                         kca,ks,ki,inv(ki),
                         kpi,kis}
   composition
```

```
        session(p,s,kps,ks,kca,h1,h2,prf,keygen)
    /\  session(p,i,kpi,ki,kca,h1,h2,prf,keygen)
    /\  session(i,s,kis,ks,kca,h1,h2,prf,keygen)

end role
```

```
goal
        %secrecy_of ClientK, ServerK
        secrecy_of sec_clientK, sec_serverK

        %Peer authenticates Server on np_ns
        authentication_on np_ns
        %Server authenticates Peer on ns
        authentication_on ns
end goal
```

```
environment()
```

15 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

- authentication 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

  intruder_knowledge = {c1,s1,c2,s2,md4,success,
                        passwd2,seed2
                       }

  composition

    session(c1,s1,md4,passwd1,success,seed1)
```

```
 /\ session(i, s1,md4,passwd2,success,seed2)

end role
```

```
goal

  %Server authenticates Client on m
  authentication_on m

end goal
```

```
environment()
```

16 EKE: Encrypted Key Exchange

16.1 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 `sec_k1, sec_k2`
- authentication on `nb`
- authentication on `na`

Problem Classification: G2 G12

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,
        Na,Nb,K : text

  const sec_k1 : protocol_id

  init  State := 0

  transition
```

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

   2. State = 1
      /\ Rcv({{K'}_Ea}_Kab)
      =|>
      State' := 2
      /\ Na' := new()
      /\ Snd({Na'}_K')
      /\ secret(K',sec_k1,{A,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,
        Na,Nb,K  : text,
        Ea       : public_key

  const sec_k2 : protocol_id

  init  State := 0
```

```
  transition

   1. State = 0 /\ Rcv({Ea'}_Kab)
      =|>
      State' := 1
      /\ K' := new()
      /\ Snd({{K'}_Ea'}_Kab)
      /\ secret(K',sec_k2,{A,B})

   2. State = 1 /\ Rcv({Na'}_K)
      =|>
      State' := 2
      /\ Nb' := new()
      /\ 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)
def=

  local SA, RA, SB, RB: channel (dy)

  composition
     eke_Init(A,B,Kab,SA,RA)
  /\ eke_Resp(B,A,Kab,SB,RB)

end role
```

```
role environment()
```

```
def=

  const a, b   : agent,
        kab    : symmetric_key,
        na, nb : protocol_id

  intruder_knowledge={a,b}

  composition
      session(a,b,kab)
   /\ session(b,a,kab)

end role
```

```
goal

 % Confidentiality (G12)
 secrecy_of sec_k1, sec_k2

 % Message authentication (G2)
 % EKE_Init authenticates EKE_Resp on nb
 authentication_on nb

 % Message authentication (G2)
 % EKE_Resp authenticates EKE_Init on na
 authentication_on na

end goal
```

```
environment()
```

16.2 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 `sec_i_MK_A, sec_r_MK_B`
- authentication on `mk_a`
- authentication on `mk_b`

Problem Classification: G2 G12

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,
        GY        : message,
        MK_A,MK_B : message

  const two : text,
        sec_i_MK_A : protocol_id

  init  State := 0

  transition

   1. State = 0  /\ Rcv(start) =|>
      State':= 1 /\ X' := new()
                 /\ 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'% Message authentication (G2)
                 /\ Snd(H(H(MK_A').two))
                 /\ secret(MK_A',sec_i_MK_A,{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=% Message authentication (G2)

  local State     : nat,
        Y         : text,
        GX        : message,
        MK_A,MK_B : message

  const one : text,
        sec_r_MK_B : protocol_id

  init  State := 0

  transition

   1. State = 0  /\ Rcv(A.{GX'}_Kab) =|>
      State':= 1 /\ Y' := new()
                 /\ MK_B' := A.B.GX'.exp(G,Y').exp(GX',Y')
                 /\ MK_A' := MK_B'
                 /\ Snd({exp(G,Y')}_Kab.H(H(MK_B').one))
                 /\ secret(MK_B',sec_r_MK_B,{A,B})% Message authentication (G2)
                 /\ witness(B,A,mk_a,MK_A')

   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) def=

   local   SA,RA,SB,RB: channel(dy)

   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,
        a,b,c       : agent,
        kab,kai,kib : symmetric_key,
        g           : text,
        h           : function

  intruder_knowledge  = {a,b,c,kai,kib}

  composition

        session(a,b,g,h,kab) /\
        session(a,i,g,h,kai) /\
        session(i,b,g,h,kib)

end role
```

```
goal

  % Confidentiality (G12)
  % secrecy_of MK
```

```
  secrecy_of sec_i_MK_A, sec_r_MK_B

  % Message authentication (G2)
  % Eke2_Init authenticates Eke2_Resp on mk_a
  authentication_on mk_a

  % Message authentication (G2)
  % Eke2_Resp authenticates Eke2_Init on mk_b
  authentication_on mk_b

end goal
```

```
environment()
```

16.3 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 `sec_i_Ca,sec_i_Cb`,
- authentication on `cb`
- authentication on `ca`

Problem Classification: G2 G12

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,
         Cb   : text,
         X,K  : message

  const  sec_i_Ca, sec_i_Cb : protocol_id
```

```
  init   State := 0

  transition

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

   2. State  = 1 /\ Rcv(X') =|>
      State':= 2 /\ Ca' := new()
                 /\ K' := exp(X',Na)
                 /\ Snd({Ca'}_exp(X',Na))
                 /\ secret(Ca',sec_i_Ca,{A,B})
                 /\ witness(A,B,ca,Ca')

   3. State  = 2 /\ Rcv({Cb'.Ca}_K ) =|>
      State':= 3 /\ Snd({Cb'}_K)
                 /\ secret(Cb',sec_i_Cb,{A,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,
        Ca   : text,
        Y,K  : message

  const sec_r_Ca, sec_r_Cb : protocol_id

  init  State := 0

  transition
```

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

   2. State  = 1 /\ Rcv({Ca'}_K) =|>
      State':= 2 /\ Cb' := new()
                 /\ Snd({Cb'.Ca'}_K)
                 /\ secret(Ca',sec_r_Ca,{A,B})
                 /\ secret(Cb',sec_r_Cb,{A,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)
def=

   local SA,RA,SB,RB: channel (dy)

   composition

       speke_Init(A,B,Kab,SA,RA)
    /\ speke_Resp(A,B,Kab,SB,RB)

end role
```

```
role environment()
def=

  const a, b           : agent,
        kab, kai, kbi  : symmetric_key,
        ca, cb         : protocol_id

  intruder_knowledge = {a, b, kai, kbi}
```

```
  composition
        session(a,b,kab)
    /\  session(a,i,kai)
    /\  session(i,b,kbi)

end role
```

```
goal

   % Confidentiality (G12)
   %secrecy_of Ca, Cb
   secrecy_of sec_i_Ca,sec_i_Cb,
              sec_r_Ca,sec_r_Cb

   % Message Authentication (G2)
   % SPEKE_Init authenticates SPEKE_Resp on cb
   authentication_on cb

   % Message Authentication (G2)
   % SPEKE_Resp authenticates SPEKE_Init on ca
   authentication_on ca

end goal
```

```
environment()
```

17 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 `sec_i_K, sec_r_K`
- authentication on `k2`
- authentication on `k1`

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,
        Salt  : message,
        DHY, V, K, M : message

  const sec_i_K : protocol_id

  init  State := 0

  transition

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

  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,k1,K')
                /\ secret(K',sec_i_K,{A,B})

  3. State = 2  /\ Rcv(H(exp(G,Na).M.K)) =|>
     State':= 3
                /\ request(A,B,k2,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,
        M, K, DHX, V: message

  const sec_r_K : protocol_id

  init  State := 0

  transition

  1. State = 0  /\ Rcv(A.DHX') =|>
     State':= 1 /\ Nb' := new()
                /\ Snd(Salt.{exp(G,Nb')}_(exp(G,H(Salt.H(A.Password)))))
                /\ V' := exp(G,H(Salt.H(A.Password)))
                /\ K' := H( V'.exp(DHX',Nb') )
                /\ M' := H(H(G).H(A).Salt.DHX'.{exp(G,Nb')}_V'.K')
                /\ witness(B,A,k2,K')
                /\ secret(K',sec_r_K,{A,B})

  2. State = 1  /\ Rcv(M) =|>
     State':= 3 /\ Snd(H(DHX.M.K))
                /\ request(B,A,k1,K)
end role
```

```
role session(A,B: agent,
             Password: symmetric_key,
             Salt: message,
             H: function,
             G: text)
def=
```

```
   local SA,RA,SB,RB: channel (dy)

   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=

  const k1,k2 : protocol_id,
        a,b,i: agent,
        kab,kai,kbi: symmetric_key,
        s_ab,s_ai,s_bi: message,
        h:function,
        g:text

  intruder_knowledge = {i, kai, kbi, s_ai, s_bi}
  composition
           session(a,b,kab,s_ab,h,g)
        /\ session(a,i,kai,s_ai,h,g)
        /\ session(b,i,kbi,s_bi,h,g)

end role
```

```
goal

    % confidentiality (G12)
    secrecy_of sec_i_K, sec_r_K

    % Entity Authentication (G1)
    % Message Authentication (G2)
    % Replay Protection (G3) --- forgotten in d6.1
    authentication_on k2
    authentication_on k1
```

```
end goal
```

```
environment()
```

18 IKEv2: Internet Key Exchange, version 2

18.1 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 `sec_a_SK, sec_b_SK`
- authentication on `sk1`
- authentication 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,
        Nr: text,
        KEr: message, %% more specific: exp(text,text)
        SK: message,
        State: nat

  const sec_a_SK : protocol_id

  init  State := 0

  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) =|>
     State':= 2 /\ SA1' := new()
                /\ DHX' := new()
                /\ Ni' := new()
                /\ SND_B( SA1'.exp(G,DHX').Ni' )

  %% 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') =|>
     State':= 4 /\ SA2' := new()
                /\ SK' := F(Ni.Nr'.SA1.exp(KEr',DHX))
                /\ SND_B( {A.{SA1.exp(G,DHX).Ni.Nr'}_(inv(Ka)).SA2'}_SK' )
                /\ 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,sec_a_SK,{A,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,
        SK, KEi: message,
        State: nat

  const sec_b_SK : protocol_id
```

```
  init  State := 1

  transition

  1. State = 1  /\ RCV_A( SA1'.KEi'.Ni' ) =|>
     State':= 3 /\ DHY' := new()
                /\ Nr' := new()
                /\ SND_A(SA1'.exp(G,DHY').Nr')
                /\ SK' := F(Ni'.Nr'.SA1'.exp(KEi',DHY'))
                /\ 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 ) =|>
     State':= 9 /\ SND_A( {B.{SA1.exp(G,DHY).Nr.Ni}_(inv(Kb)).SA2'}_SK )
                /\ secret(SK,sec_b_SK,{A,B})
                /\ request(B,A,sk2,SK)

end role
```

```
role session(A, B: agent,
             Ka, Kb: public_key,
             G: text, F: function)
def=

  local SA, RA, SB, RB: channel (dy)

  composition
          alice(A,B,G,F,Ka,Kb,SA,RA)
       /\ bob(B,A,G,F,Kb,Ka,SB,RB)

end role
```

```
role environment()
def=

  const sk1,sk2    : protocol_id,
        a, b       : agent,
        ka, kb, ki : public_key,
```

```
        g:text, f        : function

  intruder_knowledge = {g,f,a,b,ka,kb,i,ki,inv(ki)
                       }

  composition

        session(a,b,ka,kb,g,f)
     /\ session(a,i,ka,ki,g,f)
     /\ session(i,b,ki,kb,g,f)

end role
```

```
goal

  %secrecy_of SK
  secrecy_of sec_a_SK, sec_b_SK

  %Alice authenticates Bob on sk1
  authentication_on sk1
  %Bob authenticates Alice on sk2
  authentication_on sk2

end goal
```

```
environment()
```

18.2 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 `sec_a_SK, sec_b_SK`
- authentication on `sk1`
- authentication 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,
        Nr: text,
        KEr: message, %% more specifically: exp(text,text)
        SK: message,
        State: nat,
        MA: text,
        MB: text,
        AUTH_B: message

  const sec_a_SK : protocol_id

  init  State := 0

  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) =|>
     State':= 2 /\ SA1' := new()
                /\ DHX' := new()
                /\ Ni' := new()
                /\ SND_B( SA1'.exp(G,DHX').Ni' )

  %% 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') =|>
     State':= 4 /\ SA2' := new()
                /\ SK' := F(Ni.Nr'.SA1.exp(KEr',DHX))
                /\ SND_B( {A.{SA1.exp(G,DHX).Ni.Nr'}_(inv(Ka)).SA2'}_SK' )

  3. State = 4  /\ RCV_B({B.{SA1.KEr.Nr.Ni}_(inv(Kb)).SA2}_SK) =|>
     State':= 6 /\ MA' := new()
```

```
                /\ SND_B({MA'.zero}_SK)
                /\ AUTH_B' := {SA1.KEr.Nr.Ni}_(inv(Kb))
                /\ secret(SK,sec_a_SK,{A,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,
        SK, KEi: message,
        State: nat,
        MA: text,
        MB: text,
        AUTH_A: message

  const sec_b_SK : protocol_id

  init  State := 1

  transition

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

  2. State = 3  /\ RCV_A( {A.{SA1.KEi.Ni.Nr}_(inv(Ka)).SA2'}_SK ) =|>
```

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

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

end role
```

```
role session(A, B: agent,
             Ka, Kb: public_key,
             G: text, F: function)
def=

  local SA, RA, SB, RB: channel (dy)

  composition

     alice(A,B,G,F,Ka,Kb,SA,RA)
  /\ bob(B,A,G,F,Kb,Ka,SB,RB)

end role
```

```
role environment()
def=

  const sk1, sk2    : protocol_id,
        a, b        : agent,
        ka, kb, ki  : public_key,
        g           : text,
        f           : function,
        zero, one   : text

  intruder_knowledge = {g,f,a,b,ka,kb,i,ki,inv(ki),zero,one
```

```
                       }

  composition

        session(a,b,ka,kb,g,f)
     /\ session(a,i,ka,ki,g,f)
     /\ session(i,b,ki,kb,g,f)

end role
```

```
goal

  %secrecy_of SK
  secrecy_of sec_a_SK, sec_b_SK

  %Alice authenticates Bob on sk1
  authentication_on sk1
  %Bob authenticates Alice on sk2
  authentication_on sk2

end goal
```

```
environment()
```

18.3 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 `sec_a_SK, sec_b_SK`
- authentication on `sk1`
- authentication 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,
        Nr: text,
        KEr: message, %% more spefic: exp(text,text)
        SK: message,
        State: nat,
        AUTH_B: message

  const sec_a_SK : protocol_id
```

```
  init  State := 0

  transition

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

  %% 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') =|>
     State':= 4 /\ SA2' := new()
                /\ SK' := F(Ni.Nr'.SA1.exp(KEr',DHX))
                /\ SND_B( {A.F(PSK.SA1.exp(G,DHX).Ni.Nr').SA2'}_SK' )
                /\ 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,sec_a_SK,{A,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,
```

```
        SK, KEi: message,
        State: nat,
        AUTH_A: message

  const sec_b_SK : protocol_id

  init  State := 1

  transition

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

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

end role
```

```
role session(A, B: agent,
             PSK: symmetric_key,
             G: text, F: function)
def=

  local SA, RA, SB, RB: channel (dy)

  composition

           alice(A,B,G,F,PSK,SA,RA)
        /\ bob(B,A,G,F,PSK,SB,RB)

end role
```

```
role environment()
def=

  const sk1, sk2      : protocol_id,
        a, b          : agent,
        kab, kai, kbi : symmetric_key,
        g             : text,
        f          : function

  intruder_knowledge = {g,f,a,b,i,kai,kbi
                       }

  composition

        session(a,b,kab,g,f)
     /\ session(a,i,kai,g,f)
     /\ session(i,b,kbi,g,f)

end role
```

```
goal
  %secrecy_of SK
  secrecy_of sec_a_SK, sec_b_SK

  %Alice authenticates Bob on sk1
  authentication_on sk1
  %Bob authenticates Alice on sk2
  authentication_on sk2
end goal
```

```
environment()
```

18.4 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 `sec_a_SK, sec_b_SK`
- authentication on `sk1`
- authentication on `sk2`

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,
        Nr: text,
        KEr: message, %% more specifically: exp(text,text)
        SK: message,
        State: nat,
        MA: text,
        MB: text,
        AUTH_B: message

  const sec_a_SK : protocol_id

  init  State := 0

  transition

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

  %% 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') =|>
     State' = 4 /\ SA2' := new()
                /\ SK' := F(Ni.Nr'.SA1.exp(KEr',DHX))
                /\ SND_B( {A.F(PSK.SA1.exp(G,DHX).Ni.Nr').SA2'}_SK' )
```

```
  3. State = 4  /\ RCV_B({B.F(PSK.SA1.KEr.Ni.Nr).SA2}_SK) =|>
     State':= 6 /\ MA' := new()
                /\ SND_B({MA'.zero}_SK)
                /\ AUTH_B' := F(PSK.SA1.KEr.Ni.Nr)
                /\ witness(A,B,sk1,SK)

  4. State = 6  /\ RCV_B({MB'.one}_SK) =|>
     State':= 8 /\ secret(SK,sec_a_SK,{A,B})
                /\ request(A,B,sk2,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,
        SK, KEi: message,
        State: nat,
        MA: text,
        MB: text,
        AUTH_A: message

  const sec_b_SK : protocol_id

  init  State := 1

  transition

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

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

  3. State = 5  /\ RCV_A({MA'.zero}_SK) =|>
     State':= 7 /\ MB' := new()
                /\ SND_A({MB'.one}_SK)
                /\ secret(SK,sec_b_SK,{A,B})
                /\ request(B,A,sk1,SK)

end role
```

```
role session(A, B: agent,
             PSK: symmetric_key,
             G: text, F: function)
def=

  local SA, RA, SB, RB: channel (dy)

  composition

     alice(A,B,G,F,PSK,SA,RA)
  /\ bob(B,A,G,F,PSK,SB,RB)

end role
```

```
role environment()
def=

  const sk1, sk2      : protocol_id,
        a, b          : agent,
        kab, kai, kbi : symmetric_key,
        g :text, f          : function,
        zero, one     : text
```

```
  intruder_knowledge = {g,f,a,b,i,kai,kbi,zero,one
                       }

  composition

        session(a,b,kab,g,f)
     /\ session(a,i,kai,g,f)
     /\ session(i,b,kbi,g,f)

end role
```

```
goal

  %secrecy_of SK
  secrecy_of sec_a_SK, sec_b_SK

  %Alice authenticates Bob on sk
  authentication_on sk1
  %Bob authenticates Alice on sk
  authentication_on sk2

end goal
```

```
environment()
```

18.5 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 `sec_a_CSK,sec_b_CSK`
- authentication on `nr`
- authentication 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,
        Nr: text,
        KEr: message, % more specifically: exp(text,text)
        CSK: message, % CHILD_SA to be established.
        State: nat,
        MA,MB: text

  const sec_a_CSK : protocol_id

  init  State := 0

  transition

  1. State = 0  /\ RCV_B(start) =|>
     State':= 2 /\ SA' := new()
                /\ Ni' := new()
```

```
                /\ DHX' := new()
                /\ SND_B( {SA'.Ni'.exp(G,DHX')}_SK )
                /\ witness(A,B,ni,Ni')

  2. State = 2  /\ RCV_B({SA.Nr'.KEr'}_SK) =|>
     State':= 4 /\ MA' := new()
                /\ 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,sec_a_CSK,{A,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,
        KEi, CSK: message,
        State: nat,
        MA,MB: text

  const sec_b_CSK : protocol_id

  init  State := 1

  transition

  1. State = 1  /\ RCV_A( {SA'.Ni'.KEi'}_SK ) =|>
     State':= 3 /\ Nr' := new()
                /\ DHY' := new()
                /\ 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 /\ MB' := new()
                /\ SND_A( {MB'.one}_CSK )
                /\ request(B,A,ni,Ni)
                /\ secret(CSK,sec_b_CSK,{A,B})

end role
```

```
role session(A, B: agent,
             SK: symmetric_key,
             G: text, F: function)
def=

  local SAC, RA, SB, RB: channel (dy)

  composition
           alice(A,B,G,F,SK,SAC,RA)
        /\ bob(B,A,G,F,SK,SB,RB)
end role
```

```
role environment()
def=

  const ni,nr         : protocol_id,
        a, b          : agent,
        kab, kai, kbi : symmetric_key,
        g:text, f          : function,
        zero, one     : text

  intruder_knowledge = {g,f,a,b,i,kai,kbi,zero,one
                       }

  composition
```

```
        session(a,b,kab,g,f)
     /\ session(a,i,kai,g,f)
     /\ session(i,b,kbi,g,f)

end role
```

```
goal
        %secrecy_of CSK
        secrecy_of sec_a_CSK,sec_b_CSK

        %Alice authenticates Bob on nr
        authentication_on nr
        %Bob authenticates Alice on ni
        authentication_on ni
end goal
```

```
environment()
```

18.6 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

```
 A is the client
```

```
B is the server and AAA server
 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 `sec_SK, sec_EMK`
- authentication on `ker_nr_sid__nonces`
- authentication on `kei_ni_binding_noncep`

Attacks Found: None

Further Notes

- For simplicity, the server and the AAA server are merged.
- 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,
    Nr                              : text,
    SK                              : message,
    KEr                             : message,
    SID_                            : text,
    State                           : nat,
    Binding,NonceP                  : text,
    NonceS                          : text,
    EMK                             : message,
    KEr_Nr_SID__NonceS,
    KEi_Ni_Binding_NonceP           : message

  const
    sec_SK, sec_EMK  : protocol_id

  init
    State := 0

  transition

 1. State  = 0  /\ RCV(start) =|>
    State':= 2  /\ Ni' := new()
                /\ SAi1' := new()
                /\ DHX' := new()
                /\ 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  /\ Binding' := new()
                /\ NonceP' := new()
                /\ 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, sec_SK, {A,B})
                /\ secret(EMK,sec_EMK,{A,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,
    KEi                               : message,
    SK,EMK                            : message,
    NonceP,Binding                    : text,
    State                             : nat,
    KEr_Nr_SID__NonceS,
    KEi_Ni_Binding_NonceP             : message

  init
    State := 1

  transition

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

 2. State  = 3  /\ RCV({A}_PRFP1(SK)) =|>
    State':= 5  /\ SID_' := new()
                /\ 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  /\ NonceS' := new()
                /\ 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

  intruder_knowledge = {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
  secrecy_of sec_SK, sec_EMK

  %Alice authenticates Bob on ker_nr_sid__nonces
  authentication_on ker_nr_sid__nonces
  %Bob authenticates Alice on kei_ni_binding_noncep
  authentication_on kei_ni_binding_noncep

end goal
```

```
environment()
```

19 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 `sec_c_Kcs, sec_s_Kcs`
- authentication 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=

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

      const  kcs       : protocol_id,
             sec_c_Kcs : protocol_id

      init State := 0

      transition

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

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

     const  kcs       : protocol_id,
            sec_s_Kcs : protocol_id

     init State := 11

     transition

     s1.    State  = 11 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
            State':= 12 /\ SND(NAS_ID'.Access_accept)
                        /\ secret(Kcs,sec_s_KCS,{C,S})

     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,
           kcs               : protocol_id

    intruder_knowledge = {c1,s1,md5,kisk,kcik,
                          succs, fails,
                          acc_acp, acc_rej
                         }

    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
  secrecy_of sec_c_Kcs, sec_s_Kcs

  %Server authenticates Client on kcs
  authentication_on kcs

end goal
```

```
environment()
```

20 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 `sec_c_Kcs, sec_s_Kcs`
- authentication 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=

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

            const  kcs       : protocol_id,
                   sec_c_Kcs : protocol_id

            init State := 0

            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,sec_c_Kcs,{C,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=

            local  State              : nat,
                   NAS_ID , NAS_Port  : text,
                   Chall_message      : text,
                   Client_chall_reply : {text}_symmetric_key % ??? message

            const kcs       : protocol_id

            init State := 11

            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=

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

            const  kcs        : protocol_id,
                   sec_s_Kcs  : protocol_id

            init State := 21

    transition

     s5.    State  = 21 /\ RCV(NAS_ID'.NAS_Port'.Md5(Kcs)) =|>
            State':= 26 /\ SND(NAS_ID'.Chall_Message')
                        /\ secret(Kcs,sec_s_KCS,{C,S})
     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,
       kcsk , kisk, kcik : symmetric_key,
       md5               : function,
       succs, fails      : text,
       acc_acp, acc_rej  : text,
       eap_succ,
       eap_start,
       eap_req_id        : text

     intruder_knowledge = {c1,a1,s1, md5, kisk,kcik,
                           succs, fails,
```

```
                           acc_acp, acc_rej,
                           eap_succ, eap_start,
                           eap_req_id
                          }

     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
  secrecy_of sec_c_Kcs, sec_s_Kcs

  %Server authenticates Client on kcs
  authentication_on kcs

end goal
```

```
environment()
```

21 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-03.txt [MNJH05]

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_dh`
- authentication on `initiator_responder_hash_dh`

Problem Classification: G1, G3, G9, G10

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,         % Initiator's Diffie Hellman Value
        SPI_I   : text,         % Initiator's Security Parameter Index Value
        LSI_I   : text,         % Initiator's Local Scope Index Value
        SPI_R   : text,         % 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
        hash_dh   : protocol_id

        init State := 0

%       knowledge(J) = {J,R,Hash,Soln,HI_I,HI_R,G,inv(HI_I)}

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 /\ X':=new() /\ SPI_I':=new()
                  /\ 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))
```

```
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,initiator_responder_hash_dh,R2)
                  /\ secret(Hash(exp(EGY,X)),hash_dh,{J,R})

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,           % Responder's Diffie Hellman parameter
        SPI_R   :text,           % Responder's Security Parameter Index Value
        LSI_R   :text,           % Responder's Local Scope Index Value
        SPI_I   :text,           % Initiator's Security Parameter Index Value
        LSI_I   :text,           % Initiator's Local Scope Index Value
        Puzzle  :text,           % 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 /\ Y':=new() /\ Puzzle':=new() /\
                      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)) /\ SPI_R':=new()
                   /\ SND(Hash(SPI_R'.LSI_R.Hash(R2')).
                              {SPI_R'.LSI_R.Hash(R2')}_inv(HI_R))
                   /\ witness(R,J,initiator_responder_hash_dh,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=

        local SND,RCV :channel(dy)

        const
          j,r        : agent,        % Initiator and Responder
          hash_      : function,     % Hash Function
          soln_      : function,     % Solution
          hi_j,hi_r:public_key,  % Public key of the Initiator,Responder
          hi_i       :public_key,    % Public key of the intruder
          g          : nat,          % Diffie Hellman's public G value
          r2         : protocol_id % Protocol ID

        intruder_knowledge = {j,r,hash_,soln_,hi_j,hi_r,g,hi_i,inv(hi_i)}
        % in the first session, intruder should not solve puzzles.

        composition

        session(j,r,SND,RCV,hash_,soln_,hi_j,hi_r,g)
% /\    session(i,r,SND,RCV,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(j,i,SND,RCV,hash_,soln_,hi_j,hi_i,g)

end role
```

```
goal

        secrecy_of hash_dh                                  % addresses G9 and G10
        authentication_on initiator_responder_hash_dh % addresses G1 and G3

end goal
```

```
environment()
```

22 PBK: Purpose Built Keys Framework

22.1 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

- authentication 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,
    Nonce      : text

  init State := 0

  transition

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

```
    State':= 2 /\ Msg' := new()
               /\ 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,
    Msg        : text

  init State := 1

  transition

 1. State  = 1 /\ RCV({Msg'}_inv(PK_A).Hash(PK_A)) =|>
    State':= 5 /\ Nonce' := new()
               /\ SND(Nonce')

 3. State  = 5 /\ RCV({Nonce}_inv(PK_A)) =|>
    State':= 7 /\ request(A,A,msg,Msg)

end role
```

```
role session(A,B  : agent,
             Hash : function,
             PK_A : public_key)
```

```
def=

  local SNDA,RCVA,SNDB,RCVB  : channel (dy)

  composition

    alice(A,B,SNDA,RCVA,Hash,PK_A)
 /\ bob(B,A,SNDB,RCVB,Hash,PK_A)

end role
```

```
role environment()
def=

  const
    a,b            : agent,
    f              : function,
    msg            : protocol_id,
    pk_a,pk_b,pk_i : public_key

  intruder_knowledge = {a,b,f,pk_a,pk_b,pk_i,inv(pk_i)}

  composition

     session(a,b,f,pk_a)
  /\ session(b,a,f,pk_b)
  /\ session(i,b,f,pk_i)
  /\ session(a,i,f,pk_a)

end role
```

```
goal

  % Sender Invariance (G16)
  authentication_on msg

end goal
```

```
environment()
```

22.2 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

- authentication 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,
    Nonce       : text
```

```
  init  State := 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State':= 2 /\ Msg' := new()
               /\ 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,
    Msg        : text

  init State := 1

  transition

 1. State  = 1 /\ RCV(B.{Tag1.Msg'}_inv(PK_A).Hash(PK_A)) =|>
    State':= 5 /\ Nonce' := new()
               /\ 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,
             Hash       : function,
             PK_A       : public_key,
             Tag1,Tag2  : text)
def=

  local SNDA,RCVA,SNDB,RCVB  : channel (dy)

  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=

  const
    a,b             : agent,
    f               : function,
    msg             : protocol_id,
    pk_a,pk_b,pk_i  : public_key,
    tag1,tag2       : text

  intruder_knowledge = {a,b,f,pk_a,pk_b,pk_i,inv(pk_i)}

  composition
    session(a,b,f,pk_a,tag1,tag2)
 /\ session(a,b,f,pk_a,tag1,tag2)

end role
```

```
goal
```

```
  % Sender Invariance (G16)
  authentication_on msg

end goal
```

```
environment()
```

22.3 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

- authentication 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,
    Nonce      : text

  init State := 0

  transition

 1. State  = 0 /\ RCV(start) =|>
    State':= 2 /\ Msg' := new()
               /\ 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,
    Msg         : text

  init State := 1

  transition

 1. State  = 1 /\ RCV(B.{Tag1.Msg'}_inv(PK_A).Hash(PK_A)) =|>
    State':= 5 /\ Nonce' := new()
               /\ 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,
             Hash       : function,
             PK_A       : public_key,
             Tag1,Tag2        : text)
def=

  local SND,RCV,SNDA,RCVA  : channel (dy)

  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=

  const
    a,b             : agent,
    f               : function,
    msg             : protocol_id,
    pk_a,pk_b,pk_i : public_key,
    tag1,tag2       : text

  intruder_knowledge = {a,b,f,pk_a,pk_b,pk_i,inv(pk_i)}

  composition
     session(a,b,f,pk_a,tag1,tag2)
  /\ session(b,a,f,pk_b,tag1,tag2)
  /\ session(i,b,f,pk_i,tag1,tag2)
  /\ session(a,i,f,pk_a,tag1,tag2)

end role
```

```
goal

  % Sender Invariance (G16)
  authentication_on msg

end goal
```

```
environment()
```

23 Kerberos Network Authentication Service (V5)

23.1 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 `sec_a_K_CG`,
- weak authentication on `k_cg`
- weak authentication on `k_cg`
- weak authentication on `k_cs`
- weak authentication on `k_cs`
- weak authentication on `t2a`
- weak authentication on `t2a`
- weak authentication on `t1`

Problem Classification: G1, G2, G7, G10

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,
        N1, Lifetime_1  : text,
        Tstart, Texpire : text

  const k_cg : protocol_id,
        sec_a_K_CG : protocol_id

  init  St := 0

  transition

   1. St = 0  /\ Rcv(C.G.Lifetime_1'.N1') =|>
      St':= 1 /\ Tstart' := new()
              /\ Texpire' := new()
              /\ K_CG' := new()
              /\ 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')
              /\ witness(A,G,k_cg,K_CG')
              /\ secret(K_CG',sec_a_K_CG,{A,C,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,
        Lifetime_2, Tstart, Texpire, T, N2 : text,
        Tstart2, Texpire2                : text

  const t1,k_cs : protocol_id,
        sec_g_K_CG, sec_g_K_CS : protocol_id

  init  St := 0

  transition

   1. St = 0  /\
       Rcv(S.Lifetime_2'.N2'.{C.G.K_CG'.Tstart'.Texpire'}_K_AG.{C.T'}_K_CG') =|>
      St':= 1 /\ K_CS' := new()
              /\ Tstart2' := new()
              /\ Texpire2' := new()
              /\ 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')
              /\ witness(G,C,k_cs,K_CS')
              /\ secret(K_CG',sec_g_K_CG,{A,C,G})
              /\ secret(K_CS',sec_g_K_CS,{G,C,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,
        sec_s_K_CS : protocol_id

  init  St := 0

  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',sec_s_K_CS,{G,C,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,
      Tstart, Texpire, Tstart2, Texpire2    : text,
      Ticket_1, Ticket_2 : {agent.agent.symmetric_key.text.text}_symmetric_key,
      N1, N2  : text

  const t1, k_cg, k_cs, t2a, t2b : protocol_id,
        sec_c_K_CG, sec_c_K_CS : protocol_id,
        cLifetime_1, cLifetime_2: text
```

```
  init  St := 0

  transition

   1. St = 0  /\ Rcv(start) =|>
      St':= 1 /\ N1' := new()
              /\ Snd(C.G.cLifetime_1.N1')

   2. St = 1  /\ Rcv(C.Ticket_1'.{G.K_CG'.Tstart'.Texpire'.N1}_K_CA) =|>
      St':= 2 /\ N2' := new()
              /\ T' := new()
              /\ 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',sec_c_K_CG,{A,C,G})

   3. St = 2  /\ Rcv(C.Ticket_2'.{S.K_CS'.Tstart2'.Texpire2'.N2}_K_CG)  =|>
      St':= 3 /\ T2' := new()
              /\ Snd(Ticket_2'.{C.T2'}_K_CS')
              /\ witness(C,S,t2b,T2')
              /\ wrequest(C,G,k_cs,K_CS')
              /\ secret(K_CS',sec_c_K_CS,{G,C,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

  intruder_knowledge = {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
  secrecy_of sec_a_K_CG,
             sec_g_K_CG, sec_g_K_CS,
             sec_s_K_CS,
             sec_c_K_CG, sec_c_K_CS % addresses G10

  %Kerberos_C weakly authenticates Kerberos_A on k_cg
  weak_authentication_on k_cg % addresses G1 and G7
  %Kerberos_G weakly authenticates Kerberos_A on k_cg
  weak_authentication_on k_cg % addresses G1 and G7

  %Kerberos_C weakly authenticates Kerberos_G on k_cs
  weak_authentication_on k_cs % addresses G1 and G7
  %Kerberos_S weakly authenticates Kerberos_G on k_cs
  weak_authentication_on k_cs % addresses G1 and G7
```

```
  %Kerberos_C weakly authenticates Kerberos_S on t2a
  weak_authentication_on t2a % addresses G1 and G2
  %Kerberos_S weakly authenticates Kerberos_C on t2b
  weak_authentication_on t2a % addresses G1 and G2

  %Kerberos_G weakly authenticates Kerberos_C on t1
  weak_authentication_on t1 % addresses G1 and G2

end goal
```

```
environment()
```

23.2 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 `sec_k_Kcg`,
- authentication on `n1`
- authentication on `n2`
- authentication on `t2a`
- authentication on `t2b`
- authentication on `t1`

Problem Classification: G1, G2, G3, G7, G10

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,
        T1start : text,
        T1expire: text

  const sec_k_Kcg : protocol_id

  init  State := 11

  transition
    1. State  = 11 /\ RCV(U'.G.N1') =|>
       State' = 12 /\ Kcg' := new()
                   /\ T1start' := new()
                   /\ T1expire' := new()
                   /\ SND(U'.{U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                         {G.Kcg'.T1start'.T1expire'.N1'}_Kca)
                   /\ witness(A,C,n1,Kcg'.N1')
                   /\ secret(Kcg',sec_k_Kcg,{A,C,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,
        T1start, T1expire : text,
        T2start, T2expire : text,
        T1      : text
```

```
  const sec_t_Kcg, sec_t_Kcs : protocol_id

  init  State := 21

  transition
    1. State  = 21 /\ RCV( S.N2'.
                           {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                           {C.T1'}_Kcg')
                           /\ not(in(T1',L))
                            =|>

       State' = 22 /\ Kcs' := new()
                   /\ T2start' := new()
                   /\ T2expire' := new()
                   /\ 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,Kcs'.N2')
                   /\ secret(Kcg',sec_t_Kcg,{A,C,G})
                   /\ secret(Kcs',sec_t_Kcs,{G,C,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
```

```
  const  sec_s_Kcs : protocol_id

  init   State := 31

  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',sec_s_Kcs,{G,C,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,
         N1,N2   : text

  const sec_c_Kcg, sec_c_Kcs : protocol_id

  init   State := 1

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

```
        State' = 2 /\ N1' := new()
                   /\ SND(U.G.N1')

     2. State  = 2 /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Kca) =|>
        State' = 3 /\ N2' := new()
                   /\ T1' := new()
                   /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                   /\ witness(C,G,t1,T1')
                   /\ request(C,A,n1,Kcg'.N1)
                   /\ secret(Kcg',sec_c_Kcg,{A,C,G})

     3. State  = 3 /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
        State' = 4 /\ T2' := new()
                   /\ SND(Tcs'.{C.T2'}_Kcs')
                   /\ witness(C,S,t2b,T2')
                   /\ request(C,G,n2,Kcs'.N2)
                   /\ secret(Kcs',sec_c_Kcs,{G,C,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 = {}

  intruder_knowledge = {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
  secrecy_of sec_k_Kcg,
             sec_t_Kcg, sec_t_Kcs,
             sec_s_Kcs,
             sec_c_Kcg, sec_c_Kcs % addresses G10

  %Client authenticates KeyDistributionCentre on n1
```

```
  authentication_on n1 % addresses G1, G3, and G7
  %Client authenticates TicketGrantingServer on n2
  authentication_on n2 % addresses G1, G3, and G7
  %Client authenticates Server on t2a
  authentication_on t2a % addresses G1, G2, and G3
  %Server authenticates Client on t2b
  authentication_on t2b % addresses G1, G2, and G3
  %TicketGrantingServer weakly authenticates Client on t1
  authentication_on t1 % addresses G1, G2, and G3

end goal
```

```
environment()
```

23.3 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 `sec_c_KC_TGSlocal,sec_c_KC_TGSremote,sec_c_KC_Sremote,sec_c_T3,`
- authentication on `n1`
- authentication on `n1r`
- authentication on `n2`
- authentication on `t2a`
- authentication on `t2b`
- weak authentication on `t1`
- weak authentication on `t1r`

Problem Classification: G1, G2, G3, G7, G10

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,
        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

  const sec_c_KC_TGSlocal,
        sec_c_KC_TGSremote,
        sec_c_KC_Sremote,
        sec_c_T3    : protocol_id

  init  State := 0

  transition
```

```
step1.
  State = 0  /\ RCV(start)
  =|>
  State':= 1 /\ N1' := new()
             /\ SND(C.TGSlocal.N1')

step2.
  State = 1 /\ RCV(C.Ticket1'.
                 {TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'.N1}_KC_ASlocal)
  =|>
  State':= 2 /\ N2' := new()
             /\ T1' := new()
             /\ SND(TGSremote.N2'.Ticket1'.{C.T1'}_KC_TGSlocal')
             /\ witness(C,TGSlocal,t1,T1')
             /\ request(C,ASlocal,n1,KC_TGSlocal'.N1)
             /\ secret(KC_TGSlocal',sec_c_KC_TGSlocal,{ASlocal,C,TGSlocal})

step3.
  State = 2  /\ RCV(C.Ticket2b'.
             {TGSremote.KC_TGSremote'.Tstart2b'.Texpire2'.N2}_KC_TGSlocal)
  =|>
  State':= 3 /\ N3' := new()
             /\ T2B' := new()
             /\ SND( Sremote.N3'.Ticket2b'.{C.T2B'}_KC_TGSremote')
             /\ witness(C,TGSremote,t1r,T2B')
             /\ request(C,TGSlocal,n1r,KC_TGSremote'.N2)
             /\ secret(KC_TGSremote',sec_c_KC_TGSremote,{TGSlocal,C,TGSremote})

step4.
  State = 3  /\ RCV(C.Ticket3'.
                 {Sremote.KC_Sremote'.Tstart3'.Texpire3'.N3}_KC_TGSremote )
  =|>
  State':= 4 /\ T3' := new()
             /\ SND (Ticket3'.{C.T3'}_KC_Sremote')
             /\ witness(C,Sremote,t2b,T3')
             /\ request(C,TGSremote,n2,KC_Sremote'.N3)
             /\ secret(KC_Sremote',sec_c_KC_Sremote,{TGSremote,C,Sremote})
             /\ secret(T3',sec_c_T3,{C,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,
        KC_TGSlocal       : symmetric_key

  const sec_a_KC_TGSlocal : protocol_id

  init  State := 6

  transition

  step1.
    State = 6  /\ RCV( C.TGSlocal.N1') =|>
    State':= 7 /\ Tstart1' := new()
               /\ Texpire1' := new()
               /\ KC_TGSlocal' := new()
               /\ SND(C.
             {C.TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'}_KASlocal_TGSlocal.
             {TGSlocal.KC_TGSlocal'.Tstart1'.Texpire1'.N1'}_KC_ASlocal)
               /\ witness(ASlocal,C,n1,KC_TGSlocal'.N1')

               /\ secret(KC_TGSlocal',sec_a_KC_TGSlocal,{ASlocal,C,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,
        KC_TGSlocal        : symmetric_key,
        KC_TGSremote       : symmetric_key,
        T1                 : text

  const sec_tl_KC_TGSlocal,
        sec_tl_KC_TGSremote : protocol_id

  init  State := 8

  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 /\ Tstart2b' := new()
              /\ Texpire2' := new()
              /\ KC_TGSremote' := new()
              /\ 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,KC_TGSremote'.N2')
              /\ secret(KC_TGSlocal',sec_tl_KC_TGSlocal, {ASlocal,C,TGSlocal})
              /\ secret(KC_TGSremote',sec_tl_KC_TGSremote, {TGSlocal,C,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,
        KC_TGSremote,
        KC_Sremote         : symmetric_key,
        T2B                : text

  const sec_tr_KC_Sremote,
        sec_tr_KC_TGSremote : protocol_id

  init  State := 10

  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 /\ Tstart3' := new()
                /\ Texpire3' := new()
                /\ 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,KC_Sremote'.N3')
                /\ secret(KC_Sremote',sec_tr_KC_Sremote,{TGSremote,C,Sremote})
                /\ secret(KC_TGSremote',sec_tr_KC_TGSremote,{TGSlocal,C,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

  const sec_s_KC_Sremote,
        sec_s_T3          : protocol_id

  init  State := 12

  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',sec_s_KC_Sremote,{TGSremote,C,Sremote})
                /\ secret(T3',sec_s_T3,{C,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 = {}

  intruder_knowledge = {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
  secrecy_of sec_c_KC_TGSlocal,sec_c_KC_TGSremote,sec_c_KC_Sremote,sec_c_T3,
             sec_a_KC_TGSlocal,
             sec_tl_KC_TGSlocal,sec_tl_KC_TGSremote,
             sec_tr_KC_Sremote,sec_tr_KC_TGSremote,
             sec_s_KC_Sremote,sec_s_T3 % addresses G10

  %Client authenticates ASlocalRole   on n1
  authentication_on n1 % addresses G1, G3, and G7
  %Client authenticates TGSlocalRole  on n1r
  authentication_on n1r % addresses G1, G3, and G7
  %Client authenticates TGSremoteRole on n2
  authentication_on n2 % addresses G1, G3, and G7
  %Client authenticates SremoteRole   on t2a
  authentication_on t2a % addresses G1, G2, and G3
  %SremoteRole   authenticates Client   on t2b
  authentication_on t2b % addresses G1, G2, and G3
  %TGSlocalRole  weakly authenticates Client  on t1
  weak_authentication_on t1 % addresses G1 and G2
  %TGSremoteRole weakly authenticates Client on t1r
```

```
  weak_authentication_on t1r % addresses G1 and G2

end goal
```

```
environment()
```

23.4 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 `sec_a_Kcg`,
- authentication on `n1`
- authentication on `n2`
- authentication on `t2a`

- authentication on `t2b`
- authentication on `t1`

Problem Classification: G1, G2, G3, G7, G10

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,
    T1start  : text,
    T1expire : text

  const sec_a_Kcg : protocol_id

  init
    State := 11
```

```
  transition

 1. State  = 11  /\ RCV(U'.G.N1') =|>
    State' := 12 /\ Kcg' := new()
                 /\ T1start' := new()
                 /\ T1expire' := new()
                 /\ SND(U'.{U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                       {G.Kcg'.T1start'.T1expire'.N1'}_Kca       )
                 /\ witness(A,C,n1,Kcg'.N1')
                 /\ secret(Kcg',sec_a_Kcg,{A,C,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,
    T1start  : text,
    T2start  : text,
    T1expire : text,
    T2expire : text,
    T1       : text,
    IP_ADDR  : text,
    Forwardable_or_not  : protocol_id

  const  forwardable,
         sec_t_Kcg,
         sec_t_Kcs   : protocol_id

  init   State := 21
```

```
  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
      /\ Kcs' := new()
      /\ T2start' := new()
      /\ T2expire' := new()
      /\ 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,Kcs'.N2')
      /\ secret(Kcg',sec_t_Kcg,{A,C,G})
      /\ secret(Kcs',sec_t_Kcs,{G,C,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

  const sec_s_Kcs : protocol_id

  init  State := 31

  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',sec_s_Kcs,{G,C,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,T2   : text,
  IP_ADDR : text,
  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

const forwardable,
      un_forwardable : protocol_id,
      sec_c_Kcg1,
      sec_c_Kcg2,
      sec_c_Kcs      : protocol_id

init  State := 1

transition

 1. State  = 1  /\ RCV(start)  =|>
   State' := 2 /\ N1' := new()
               /\ SND(U.G.N1')

21. State  = 2  /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Kca) =|>
   State' := 3 /\ N2' := new()
               /\ T1' := new()
               /\ IP_ADDR' := new()
               /\ SND(IP_ADDR'.S.N2'.Tcg'.{C.T1'}_Kcg'.forwardable)
               /\ witness(C,G,t1,T1')
               /\ request(C,A,n1,Kcg'.N1)
               /\ secret(Kcg',sec_c_Kcg1,{A,C,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,Kcg'.N1)
                /\ secret(Kcg',sec_c_Kcg2,{A,C,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,Kcs'.N2)
                /\ secret(Kcs',sec_c_Kcs,{G,C,S})

 4. State  = 4  /\ RCV(U.Tcs2'.{S.Kcs'.T2start.T2expire.N2}_Kcg) =|>
    State' := 5 /\ T2' := new()
                /\ 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 = {}

  intruder_knowledge = {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
  secrecy_of sec_a_Kcg,
             sec_t_Kcg,sec_t_Kcs,
             sec_s_Kcs,
             sec_c_Kcg1,sec_c_Kcg2,sec_c_Kcs % addresses G10

  %Client authenticates AuthenticationServer on n1
  authentication_on n1 % addresses G1, G3, and G7
  %Client authenticates TicketGrantingServer on n2
  authentication_on n2 % addresses G1, G3, and G7
  %Client authenticates Server on t2a
  authentication_on t2a % addresses G1, G2, and G3
  %Server authenticates Client on t2b
```

```
  authentication_on t2b % addresses G1, G2, and G3
  %TicketGrantingServer authenticates Client on t1
  authentication_on t1 % addresses G1, G2, and G3

end goal
```

```
environment()
```

23.5 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 `sec_a_Kcg`,
- authentication on `n1`
- authentication on `n2`
- authentication on `t2a`
- authentication on `t2b`
- authentication on `t1`
- authentication on `t0`

Problem Classification: G1, G2, G3, G7, G10

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,
        T1start  : text,
        T1expire : text,
        Ktemp    : symmetric_key

  const sec_a_Kcg : protocol_id

  init  State := 11

  transition
    1. State  = 11  /\ RCV(U'.G.N1'.
                              {Kca.T0'.N1'.Hash(U'.G.N1')}_inv(Kca))
                    /\ not(in(T0',L)) =|>
       State' := 12 /\ Kcg' := new()
                    /\ T1start' := new()
                    /\ T1expire' := new()
                    /\ Ktemp' := new()
                    /\ 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,Kcg'.N1')
                     /\ wrequest(A,C,t0,T0')
                     /\ secret(Kcg',sec_a_Kcg,{A,C,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,
        T1start,T1expire : text,
        T2start,T2expire : text,
        T1      : text

  const sec_t_Kcg, sec_t_Kcs : protocol_id

  init  State := 21

  transition
    1. State  = 21  /\ RCV( S.N2'.
                           {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                           {C.T1'}_Kcg')
                           /\ not(in(T1',L)) =|>
       State' := 22 /\ Kcs' := new()
                    /\ T2start' := new()
                    /\ T2expire' := new()
                    /\ 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,Kcs'.N2')
                      /\ secret(Kcg',sec_t_Kcg,{A,C,G})
                      /\ secret(Kcs',sec_t_Kcs,{G,C,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

  const sec_s_Kcs : protocol_id

  init  State := 31

  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',sec_s_Kcs,{G,C,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,T1,T2 : text,
        Ktemp    : symmetric_key,
        N1, N2   : text

  const sec_c_Kcs,sec_c_Kcg : protocol_id

  init  State := 1

  transition
    1. State  = 1  /\ RCV(start) =|>
       State' := 2 /\ T0' := new()
                   /\ N1' := new()
                   /\ 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 /\ T1' := new()
                   /\ N2' := new()
                   /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                   /\ witness(C,G,t1,T1')
                   /\ request(C,A,n1,Kcg'.N1)
                   /\ secret(Kcg',sec_c_Kcg,{A,C,G})
```

```
      3. State  = 3  /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
         State' := 4 /\ T2' := new()
                     /\ SND(Tcs'.{C.T2'}_Kcs')
                     /\ witness(C,S,t2b,T2')
                     /\ request(C,G,n2,Kcs'.N2)
                     /\ secret(Kcs',sec_c_Kcs,{G,C,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_gi,
        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 = {}

  intruder_knowledge = {a,g,c,s,pka,hash_,k_gi,u1,u2,
                        kia,inv(kia)}

  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
  secrecy_of sec_a_Kcg,
             sec_t_Kcg, sec_t_Kcs,
             sec_s_Kcs,
             sec_c_Kcs,sec_c_Kcg % addresses G10

  %Client authenticates AuthenticationServer on n1
  authentication_on n1 % addresses G1, G3, and G7
  %Client authenticates TicketGrantingServer on n2
  authentication_on n2 % addresses G1, G3, and G7
  %Client authenticates Server on t2a
  authentication_on t2a % addresses G1, G2, and G3
  %Server authenticates Client on t2b
  authentication_on t2b % addresses G1, G2, and G3
  %TicketGrantingServer authenticates Client on t1
```

```
  authentication_on t1 % addresses G1, G2, and G3
  %AuthenticationServer authenticates Client on t0
  authentication_on t0 % addresses G1, G2, and G3

end goal
```

```
environment()
```

23.6 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 `sec_a_Kcg`,
- authentication on `n1`
- authentication on `n2`
- authentication on `t2b`
- authentication on `t2a`
- authentication on `t1`
- authentication on `t0`

Problem Classification: G1, G2, G3, G7, G10

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,
        T1start  : text,
        T1expire : text

  const sec_a_Kcg : protocol_id

  init  State := 11

  transition
    1. State  = 11 /\ RCV(U'.G.N1'.{C.T0'}_Kca)
                   /\ not(in(T0',L)) =|>
       State':= 12 /\ Kcg' := new()
                   /\ T1start' := new()
                   /\ T1expire' := new()
                   /\ SND(U'.
                          {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                          {G.Kcg'.T1start'.T1expire'.N1'}_Kca)
                   /\ L' := cons(T0',L)
                   /\ witness(A,C,n1,Kcg'.N1')
                   /\ wrequest(A,C,t0,T0')
                   /\ secret(Kcg',sec_a_Kcg,{A,C,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,
        T1start,T1expire  : text,
        T2start, T2expire : text,
        T1        : text

  const sec_t_Kcg, sec_t_Kcs : protocol_id

  init  State := 21

  transition
    1. State  = 21 /\ RCV(S.N2'.
                          {U'.C.G.Kcg'.T1start'.T1expire'}_Kag.
                          {C.T1'}_Kcg')
                   /\ not(in(T1',L))
       =|>
       State':= 22 /\ Kcs' := new()
                   /\ T2start' := new()
                   /\ T2expire' := new()
                   /\ 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,Kcs'.N2')
                   /\ secret(Kcg',sec_t_Kcg,{A,C,G})
                   /\ secret(Kcs',sec_t_Kcs,{G,C,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

  const sec_s_Kcs : protocol_id

  init  State := 31

  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',sec_s_Kcs,{G,C,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,T1,T2 : text,
```

```
        N1,N2    : text

    const sec_c_Kcg, sec_c_Kcs : protocol_id

    init  State := 1

    transition
      1. State  = 1 /\ RCV( start )  =|>
         State':= 2 /\ N1' := new()
                    /\ T0' := new()
                    /\ 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 /\ N2' := new()
                    /\ T1' := new()
                    /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                    /\ witness(C,G,t1,T1')
                    /\ request(C,A,n1,Kcg'.N1)
                    /\ secret(Kcg',sec_c_Kcg,{A,C,G})

      3. State  = 3 /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
         State':= 4 /\ T2' := new()
                    /\ SND(Tcs'.{C.T2'}_Kcs')
                    /\ witness(C,S,t2a,T2')
                    /\ request(C,G,n2,Kcs'.N2)
                    /\ secret(Kcs',sec_c_Kcs,{G,C,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,
        kgi,
        kca,kgs,kag             : symmetric_key,
        kia                     : symmetric_key,
        u3,
        u1,u2                   : agent,
        t0,t1,t2a,t2b,n1,n2     : protocol_id

  init LS = {} /\ LG = {} /\ LA = {}

  intruder_knowledge = {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

end role
```

```
goal

  %secrecy_of Kcg,Kcs
  secrecy_of sec_a_Kcg,
             sec_t_Kcg, sec_t_Kcs,
             sec_s_Kcs,
             sec_c_Kcg, sec_c_Kcs % addresses G10

  %Client authenticates AuthenticationServer on n1
  authentication_on n1 % addresses G1, G3, and G7
  %Client authenticates TicketGrantingServer on n2
  authentication_on n2 % addresses G1, G3, and G7
  %Client authenticates Server on t2b
  authentication_on t2b % addresses G1, G2, and G3
  %Server authenticates Client on t2a
  authentication_on t2a % addresses G1, G2, and G3
  %TicketGrantingServer authenticates Client on t1
  authentication_on t1 % addresses G1, G2, and G3
  %AuthenticationServer authenticates Client on t0
  authentication_on t0 % addresses G1, G2, and G3

end goal
```

```
environment()
```

24 TESLA: Timed Efficient Stream Loss-tolerant Authentication

Protocol Purpose

Secure source authentication mechanism for multicast or broadcast data streams

Definition Reference

- `http://www.ietf.org/rfc/rfc4082.txt` [PSC+04]
- `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

- authentication on `sender_dummy_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 `sender_dummy_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
```

```
  const k_N: symmetric_key

  init State = 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 /\ M' := new() /\ SND(M'.hash_(K',M').F(K'))
/\ K_prev' = K'
                /\ Time' = tick(Time)
                /\ witness(S,S,sender_dummy_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

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,sender_dummy_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

  intruder_knowledge = {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

  authentication_on sender_dummy_msgstream

end goal
```

```
environment()
```

25 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-24.txt [YKS+05]

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 `sec_K, sec_KCS, sec_KSC %%`
- authentication 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,
        NS:       text,
        X:        text,
        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,
        sec_K,
        sec_KCS,
        sec_KSC,
        sec_secretC : protocol_id

  init   State := 1

  transition

   1. State   = 1 /\ RCV(start) =|>
      State':= 3 /\ NC' := new()
                 /\ SND(NC')
```

```
   3. State  = 3 /\ RCV(NS') =|>
      State':= 5 /\ X' := new()
                 /\ 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',  sec_K,  {C,S})
                 /\ secret(KCS',sec_KCS,{C,S})
                 /\ secret(KSC',sec_KSC,{C,S})
                 /\ SND({secretC}_KCS')
                 %/\ secret(secretC,sec_secretC,{C,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,
        NC:       text,
        Y:        text,
        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

  transition
   2. State  = 2 /\ RCV(NC') =|>
      State':= 6 /\ NS' := new()
                 /\ SND(NS')

   6. State  = 6 /\ RCV(EGX') =|>
      State':= 8 /\ Y' := new()
                 /\ 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

  intruder_knowledge = {c,s,hash_,host_key,g,letter_c,letter_d,
                         inv(host_key(i))}

  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
  secrecy_of sec_K, sec_KCS, sec_KSC

  %Client authenticates Server on k
  authentication_on k

end goal
```

```
environment()
```

26 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

- authentication 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,
    NonceC   : text,
    Time     : text

  init
    State := 0

  transition

 1. State   = 0 /\ RCV(start) =|>
    State' := 2 /\ Data' := new()
                /\ Nonce' := new()
                /\ 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

  init
    State := 1

  transition

 1. State   = 1 /\ RCV(HashedData'.NonceC') =|>
    State' := 3 /\ Time' := new()
                /\ 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

  intruder_knowledge = {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
  authentication_on authdata

end goal
```

```
environment()
```

27 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 `sec_clientk,sec_serverk`
- authentication on `na_nb1`
- authentication 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,
         Nb: text,
         State: nat,
         Finished, ClientK, ServerK: message,
         Kb: public_key,
         M: message

   const sec_clientk, sec_serverk : protocol_id

   init  State := 0
```

```
transition

1.  State = 0
    /\ RCV(start)
    =|>
    State' := 2
    /\ Na' := new()
    /\ Pa' := new()
    /\ Sid' := new()
    /\ 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
    /\ PMS' := new()
    /\ 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,sec_clientk,{A,B})
    /\ secret(ServerK,sec_serverk,{A,B})
```

```
end role
```

```
role bob(A, B : agent,
         H, PRF, KeyGen: function,
         Kb, Ks: public_key,
         SND, RCV: channel (dy))
played_by B
def=

   local Na, Nb, Sid, Pa, PMS: text,
         State: nat,
         Ka: public_key

   init  State := 1

   transition

   1.  State = 1
       /\ RCV(A.Na'.Sid'.Pa')
       =|>
       State' := 3
       /\ Nb' := new()
       /\ 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)
def=

   local  SA, SB, RA, RB: channel (dy)

   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=

   const na_nb1, na_nb2 : protocol_id,
         h, prf, keygen : function,
         a, b           : agent,
         ka, kb, ki, ks : public_key

   intruder_knowledge = { a, b, ka, kb, ks, ki, inv(ki),
                          {i.ki}_(inv(ks)) }

   composition
        session(a,b,ka,kb,ks,h,prf,keygen)
    /\  session(a,i,ka,ki,ks,h,prf,keygen)
    /\  session(i,b,ki,kb,ks,h,prf,keygen)

end role
```

```
goal
```

```
        secrecy_of sec_clientk,sec_serverk
        %Alice authenticates Bob on na_nb1
        authentication_on na_nb1
        %Bob authenticates Alice on na_nb2
        authentication_on na_nb2

end goal
```

```
environment()
```

Part III

e-Business

28 ASW Fair Exchange Protocol

28.1 original

Protocol Purpose

The ASW protocol, presented by Asokan, Shoup, and Waidner in [ASW98], is an optimistic fair exchange protocol for contract signing intended to enable two parties to commit themselves to a previously agreed upon contractual text. A trusted third party (T3P) is involved *only* if dispute resolution is required (hence the term *optimistic*, which differentiates this approach from others in which an online trusted party is involved in every exchange). In resolving disputes, the T3P issues either a *replacement contract* asserting that he recognises the contract in question as valid, or an *abort token* asserting that he has never issued, and will never issue, a replacement contract. An important requirement of the protocol is that the intruder cannot block messages between an honest agent and the T3P forever.

Definition Reference

[HDM04, ASW98]

Model Authors

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

Alice&Bob style

In ASW, the parties are generally called the originator `O`, the responder `R`, and the trusted third party `T`. Their respective public keys are labelled `Vo`, `Vr`, and `Vt`. We denote with `Text1` the contractual text that `O` and `R` wish to sign. Finally, `No` and `Nr` are the respective nonces of `O` and `R`. The constant `aborted` is used to indicate that the originator wishes to abort the attached contract. The `Aborted` and `Resolved` sets are maintained by the trusted server to keep track of what contracts have been aborted and for which contracts replacements have been issued.

```
The Exchange Subprotocol
1. O -> R : me1 = {Vo.Vr.T.Text1.h(No)}_inv(Vo)
2. R -> O : me2 = {me1.h(Nr)}_inv(Vr)
3. O -> R : No
```

```
4. R -> O : Nr
The Abort Subprotocol
1. O -> T: ma1 = {aborted.me1}_inv(Vo)
2. T -> O: ma2 = if Resolved(me1) then {me1.me2}_inv(Vt)
                 else {aborted.ma1}_inv(Vt); Aborted(ma1) := true
Resolve Subprotocol
1. O -> T: mr1 = me1.me2
2. T -> O: mr2 = if Aborted(me1) then {aborted.me1}_inv(Vt)
                 else {me1.me2}_inv(Vt) ; Resolved(me1) := true
```

Model Limitations

Issues abstracted from:

- In order to avoid that the model becomes infinite merely because the trusted server must always listen for new requests, we limit the number of requests that `T` can answer.

Problems Considered: 3

- authentication on `no`
- authentication on `nr`
- secrecy of `no_secret`

Attacks Found: None

Further Notes

This specification reflects the protocol in its original form and led to the discovery of the attack presented in Section 5 of [HDM04]. In that paper, the authors show how the fair exchange security goal of ASW can be reduced, via a meta-reasoning step, to a secrecy goal. In particular, they show that this goal is achieved for the originator, if he is assured that, whenever he aborts a contract exchange and receives an abort token, then the actual contract remains secret. In this specification, we declare the originator's nonce (or "secret committment") to be secret, as it is required for any valid standard contract. The security goals required to detect the attack are not included in this variant, as they are quite complex. See the "abort token attack" variant.

HLPSL Specification

```
role orig(O, R, T    : agent,
          Text       : text,
          Vo, Vr, Vt : public_key) played_by O def=

 local S      : nat,
       No, Nr : text,
          SND, RCV   : channel (dy)

 init S := 0

 transition

 % Exchange subprotocol
 1. S = 0 /\ RCV(start)
    =|>
    S' := 1 /\ No' := new()
    /\ SND({Vo.Vr.T.Text.h(No')}_inv(Vo))
    /\ witness(O,R,no,No'.Text)

 2. S = 1 /\ RCV({{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr')}_inv(Vr))
    =|>
    S' := 2 /\ SND(No)

 3. S = 2 /\ RCV(Nr)
    =|>
    S' := 3 /\ request(O,R,nr,Nr.Text)

 % Abort subprotocol
 4. S = 1 /\ RCV(timeout)
    =|>
    S' := 5
    /\ SND({aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo))
    /\ secret(No,no_secret,{O})

 5. S = 5
    /\ RCV({ aborted.
            {aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo)}_inv(Vt))
    =|>
    S' := 6
```

```
 6. S = 5
    /\ RCV({ {Vo.Vr.T.Text.h(No)}_inv(Vo).
            {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr')}_inv(Vr)}_inv(Vt))
    =|>
    S' := 7

 % Resolve subprotocol
 7. S = 2 /\ RCV(resolve)
    =|>
    S' := 8
    /\ SND( {Vo.Vr.T.Text.h(No)}_inv(Vo).
           {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr))

 8. S = 8 /\ RCV({ aborted.
                  {aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo)}_inv(Vt))
    =|>
    S' := 9

 9. S = 8 /\ RCV({ {Vo.Vr.T.Text.h(No)}_inv(Vo).
                  {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr)}_inv(Vt))
    =|>
    S' := 10 /\ request(O,R,nr,Nr.Text)

end role
```

```
role resp(O, R, T    : agent,
          Text       : text,
          Vo, Vr, Vt : public_key) played_by R def=

 local S      : nat,
       No, Nr : text,
          SND, RCV   : channel (dy)

 init S := 0

 transition

 % Exchange subprotocol
```

```
 1. S = 0 /\ RCV({Vo.Vr.T.Text.h(No')}_inv(Vo))
    =|>
    S' := 1 /\ Nr' := new()
    /\ SND({{Vo.Vr.T.Text.h(No')}_inv(Vo).h(Nr')}_inv(Vr))
    /\ witness(R,O,nr,Nr'.Text)

 2. S = 1 /\ RCV(No)
    =|>
    S' := 2 /\ SND(Nr)
    /\ request(R,O,no,No.Text)

 % Resolve subprotocol
 3. S = 1 /\ RCV(resolve)
    =|>
    S' := 3
    /\ SND( {Vo.Vr.T.Text.h(No)}_inv(Vo).
           {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr))

 8. S = 3 /\ RCV({ aborted.
                  {aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo)}_inv(Vt))
    =|>
    S' := 4

 9. S = 3 /\ RCV({ {Vo.Vr.T.Text.h(No)}_inv(Vo).
                  {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr)}_inv(Vt))
    =|>
    S' := 5
    /\ request(R,O,no,No.Text)

end role
```

```
role server(T        : agent,
            Vt       : public_key,
            AList    : (message.message) set,
            RList    : (message.message) set) played_by T def=

 local S        : nat,
       Vo, Vr   : public_key,
       Text     : text,
```

```
       No, Nr   : text,
       Count, X : message,
            SND, RCV : channel (dy)

init S := 0
     %% The Count variable specifies how many requests
     %% the trusted server can answer.  One request is
     %% possible per application of "succ"
     /\ Count := succ(t)

transition

% Respond to an abort request
1. S = 0 /\ RCV({aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo'))
   /\ in(( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
          {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')), RList)
   /\ Count = succ(X')
   =|>
   S' := 0
   /\ SND({ {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
           {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')}_inv(Vt))
   /\ Count' := X'

2. S = 0
   /\ RCV({aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo'))
   /\ not(in(( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
              {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')), RList))
   /\ Count = succ(X')
   =|>
   S' := 0
   /\ SND({ aborted.
           {aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')}_inv(Vt))
   /\ AList' := cons(( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
                      {aborted.
                       {Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')), AList)
   /\ Count' := X'

% Respond to a resolve request
3. S = 0 /\ RCV({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
               {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr'))
   /\ in(( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
```

```
           {aborted.
            {Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')), AList)
    /\ Count = succ(X')
    =|>
    S' := 0
    /\ SND({ aborted.
            {aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')}_inv(Vt))
    /\ Count' := X'

 4. S = 0 /\ RCV( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
                 {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr'))
    /\ not(in(( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
               {aborted.
                {Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')), AList))
    /\ Count = succ(X')
    =|>
    S' := 0
    /\ SND({ {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
            {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')}_inv(Vt))
    /\ RList' := cons(( {Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
                       {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')), RList)
    /\ Count' := X'

end role
```

```
role session(O, R, T    : agent,
             Vo, Vr, Vt : public_key,
             Text       : text) def=

 composition
  orig(O,R,T,Text,Vo,Vr,Vt) /\
  resp(O,R,T,Text,Vo,Vr,Vt)
end role
```

```
role environment() def=

  local AList, RList   : (message.message) set
```

```
  const succ                              : function,
        no, nr, no_secret                 : protocol_id,
        o, r, t, ref                      : agent,
        vo, vr, vt, ki                    : public_key,
        aborted, timeout, resolve, text1 : text,
        h                                 : function

  init AList = {}
    /\ RList = {}

 intruder_knowledge = {aborted, timeout, resolve, text1,
                       o, r, t, vo, vr, vt, ki, inv(ki), h }

 composition
  session(o,r,t,vo,vr,vt,text1)  /\
%  session(i,r,t,ki,vr,vt,text1)  /\
%  session(i,r,t,ki,vr,vt,text1) /\
  server(t,vt,AList,RList)

end role
```

```
goal
  % Entity authentication (G1)
  % Message authentication (G2)
  % Replay protection (G3)
  % Accountability (G17)
  % Proof of origin (G18)
  % Proof of delivery (G19)
  authentication_on no
  authentication_on nr

  % Expressing fair exchange via observer (not described in D6.1)
  secrecy_of no_secret
end goal

environment()
```

28.2 abort token attack

Protocol Purpose

The ASW protocol, presented by Asokan, Shoup, and Waidner in [ASW98], is an optimistic fair exchange protocol for contract signing intended to enable two parties to commit themselves to a previously agreed upon contractual text. A trusted third party (T3P) is involved *only* if dispute resolution is required (hence the term *optimistic*, which differentiates this approach from others in which an online trusted party is involved in every exchange). In resolving disputes, the T3P issues either a *replacement contract* asserting that he recognises the contract in question as valid, or an *abort token* asserting that he has never issued, and will never issue, a replacement contract. An important requirement of the protocol is that the intruder cannot block messages between an honest agent and the T3P forever.

Definition Reference

[HDM04, ASW98]

Model Authors

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

Alice&Bob style

In ASW, the parties are generally called the originator `O`, the responder `R`, and the trusted third party `T`. Their respective public keys are labelled `Vo`, `Vr`, and `Vt`. We denote with `Text1` the contractual text that `O` and `R` wish to sign. Finally, `No` and `Nr` are the respective nonces of `O` and `R`. The constant `aborted` is used to indicate that the originator wishes to abort the attached contract. The `Aborted` and `Resolved` sets are maintained by the trusted server to keep track of what contracts have been aborted and for which contracts replacements have been issued.

```
The Exchange Subprotocol
1. O -> R : me1 = {Vo.Vr.T.Text1.h(No)}_inv(Vo)
2. R -> O : me2 = {me1.h(Nr)}_inv(Vr)
3. O -> R : No
4. R -> O : Nr
The Abort Subprotocol
1. O -> T: ma1 = {aborted.me1}_inv(Vo)
```

```
2. T -> O: ma2 = if Resolved(me1) then {me1.me2}_inv(Vt)
           else {aborted.ma1}_inv(Vt); Aborted(ma1) := true
Resolve Subprotocol
1. O -> T: mr1 = me1.me2
2. T -> O: mr2 = if Aborted(me1) then {aborted.me1}_inv(Vt)
           else {me1.me2}_inv(Vt) ; Resolved(me1) := true
```

Model Limitations

Issues abstracted from:

- In order to avoid that the model becomes infinite merely because the trusted server must always listen for new requests, we limit the number of requests that T can answer.

Problems Considered: 3

- authentication on no
- authentication on nr
- secrecy of no_secret
- secrecy of secret_ref

Attacks Found:

```
i -> (r,4): {ki.vr.t.text1.h(x78)}_inv(ki)
(r,4) -> i: {{ki.vr.t.text1.h(x78)}_inv(ki).h(Nr(1))}_inv(vr)
i -> (r,7): {ki.vr.t.text1.h(x78)}_inv(ki)
(r,7) -> i: {{ki.vr.t.text1.h(x78)}_inv(ki).h(Nr(2))}_inv(vr)
i -> (t,7): {aborted.{ki.vr.t.text1.h(x78)}_inv(ki)}_inv(ki)
(t,7) -> i: {aborted.{aborted.{ki.vr.t.text1.h(x78)}_inv(ki)}_inv(ki)}_inv(vt)
i -> (r,7): x78
(r,7) -> i: Nr(2)
i -> (r,4): x78
(r,4) -> i: Nr(1)
i -> (ref,7): {aborted.{aborted.{ki.vr.t.text1.h(x78)}_inv(ki)}_inv(ki)}_inv(vt).
           {{ki.vr.t.text1.h(x78)}_inv(ki).h(Nr(1))}_inv(vr).Nr(1).
           {{ki.vr.t.text1.h(x78)}_inv(ki).h(Nr(2))}_inv(vr).Nr(2)
```

The attack is described in the following section.

Further Notes

This specification reflects the protocol in its original form and led to the discovery of the attack presented in Section 5 of [HDM04]. In that paper, the authors show how the fair exchange security goal of ASW can be reduced, via a meta-reasoning step, to a secrecy goal. In particular, they show that this goal is achieved for the originator, if he is assured that, whenever he aborts a contract exchange and receives an abort token, then the actual contract remains secret. In this specification, we declare the originator's nonce (or "secret committment") to be secret, as it is required for any valid contract.

A second security goal relating to the responder, described in detail in that paper, is quite complicated. To specify it directly in HLPSL would require a very complex temporal formula. We therefore instead define a "monitor" role called "referee" which, if the intruder violates this goal, raises a trivial secrecy error in order to flag an attack.

The reason this is required is as follows. In ASW, there are three important contract-related pieces of information. Firstly, one could have the standard contract, as exchanged by two agents. Secondly, the originator can timeout and request that the contract be aborted; he will receive an abort token from the T3P. Finally, the T3P might also issue a so-called replacement contract to either party. Now, if an intruder has exchanged a standard contract with an honest responder `R` without the involvement of the T3P, then he can always request an abort token, and it will be issued. So our security goal must be stronger than simply precluding the intruder from obtaining both a contract and an abort token. Now, note that the `me1` message is the basis of the abort token but it contains no information about `R`'s nonce. That means that the intruder could get both a standard contract and an abort token and could then initiate a third session with `R`, using the same contractual text and the same nonce. `R` will respond, but I ignores him. When `R` contacts the T3P, he will get an abort token, although the intruder already has a valid contract for this `me1`. Note that `R` in fact possesses this contact too, but he associates it with a different session of the protocol. For this reason, the referee checks if the intruder can provide the following things:

- A standard contract with `R`: `me1.me2.Ni.Nr`
- An abort token on `me1`: `{abort.{abort.me1}_inv(Ki)}_inv(Vt)`
- A second half-contract related to the same `me1`: `me2' = {me1.h(Nr')}_inv(Vr)` where `me2'` is different from `me2`.

If the intruder can provide all of these, then that indicates that `R` cannot obtain a replacement contract from the T3P even though the intruder has a valid contract.

HLPSL Specification

```
role orig(O, R, T: agent,
          Text: text,
          Vo, Vr, Vt: public_key) played_by O def=

 local S: nat,
       No, Nr: text,
         SND, RCV: channel (dy)
 init S := 0

 transition

 % Exchange subprotocol
 1. S = 0 /\ RCV(start)
    =|>
    S' := 1 /\ No' := new() /\
    SND({Vo.Vr.T.Text.h(No')}_inv(Vo))
    /\ witness(O,R,no,No'.Text)

 2. S = 1 /\ RCV({{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr')}_inv(Vr))
    =|>
    S' := 2 /\ SND(No)

 3. S = 2 /\ RCV(Nr)
    =|>
    S' := 3 /\ request(O,R,nr,Nr.Text)

 % Abort subprotocol
 4. S = 1 /\ RCV(timeout)
    =|>
    S' := 5 /\ SND({aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo))
   /\ secret(No,no_secret,{O})

 5. S = 5 /\ RCV({aborted.{aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo)}_inv(Vt))
    =|>
    S' := 6

 6. S = 5 /\ RCV({{Vo.Vr.T.Text.h(No)}_inv(Vo).
                 {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr')}_inv(Vr)}_inv(Vt))
    =|>
```

```
   S' := 7

 % Resolve subprotocol
 7. S = 2 /\ RCV(resolve)
   =|>
   S' := 8 /\ SND({Vo.Vr.T.Text.h(No)}_inv(Vo).
                  {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr))

 8. S = 8 /\ RCV({aborted.{aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo)}_inv(Vt))
   =|>
   S' := 9

 9. S = 8 /\ RCV({{Vo.Vr.T.Text.h(No)}_inv(Vo).
                 {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr)}_inv(Vt))
   =|>
   S' := 10 /\ request(O,R,nr,Nr.Text)

end role
```

```
role resp(O, R, T: agent,
          Text: text,
          Vo, Vr, Vt: public_key) played_by R def=

 local S: nat,
        No,Nr: text,
          SND, RCV: channel (dy)
 init S := 0

 transition

 % Exchange subprotocol
 1. S = 0 /\ RCV({Vo.Vr.T.Text.h(No')}_inv(Vo))
   =|>
   S' := 1 /\ Nr' := new()
   /\ SND({{Vo.Vr.T.Text.h(No')}_inv(Vo).h(Nr')}_inv(Vr))
   /\ witness(R,O,nr,Nr'.Text)

 2. S = 1 /\ RCV(No)
   =|>
```

```
    S' := 2 /\ SND(Nr)
   /\ request(R,O,no,No.Text)

 % Resolve subprotocol
 3. S = 1 /\ RCV(resolve)
    =|>
    S' := 3 /\ SND({Vo.Vr.T.Text.h(No)}_inv(Vo).
                   {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr))

 8. S = 3 /\ RCV({aborted.{aborted.{Vo.Vr.T.Text.h(No)}_inv(Vo)}_inv(Vo)}_inv(Vt))
    =|>
    S' := 4

 9. S = 3 /\ RCV({{Vo.Vr.T.Text.h(No)}_inv(Vo).
                 {{Vo.Vr.T.Text.h(No)}_inv(Vo).h(Nr)}_inv(Vr)}_inv(Vt))
    =|>
    S' := 5
   /\ request(R,O,no,No.Text)

end role
```

```
role server(T: agent,
            Vt: public_key,
            AList: (message.message) set,
            RList: (message.message) set) played_by T def=

 local S: nat, Vo, Vr: public_key,
       Text: text,
       No, Nr: text,
       Count, X: message,
           SND, RCV: channel (dy)

 init S := 0 /\
      %% The Count variable specifies how many requests
      %% the trusted server can answer.  One request is
      %% possible per application of "succ"
      Count := succ(t)

 transition
```

```
 % Respond to an abort request
 1. S = 0 /\ RCV({aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo'))
    /\ in(({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
            {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')), RList)
    /\ Count = succ(X')
    =|>
    S' := 0
 /\ SND({{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
         {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')}_inv(Vt))
 /\ Count' := X'

 2. S = 0
 /\ RCV({aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo'))
 /\ not(in(({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
             {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')), RList))
 /\ Count = succ(X')
    =|>
    S' := 0
 /\ SND({aborted.
         {aborted.
           {Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')}_inv(Vt))
 /\ AList' := cons(({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
                    {aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')), AList)
 /\ Count' := X'

% Respond to a resolve request
3. S = 0 /\ RCV({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
               {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr'))
/\ in(({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
        {aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')), AList)
/\ Count = succ(X')
   =|>
   S' := 0 /\ SND({aborted.
                   {aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')}_inv(Vt))
           /\ Count' := X'

4. S = 0 /\ RCV({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
                {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr'))
/\ not(in(({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
           {aborted.{Vo'.Vr'.T.Text'.h(No')}_inv(Vo')}_inv(Vo')), AList))
```

```
/\ Count = succ(X')
    =|>
    S' := 0
 /\ SND({{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
        {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')}_inv(Vt))
 /\ RList' := cons(({Vo'.Vr'.T.Text'.h(No')}_inv(Vo').
                     {{Vo'.Vr'.T.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr')), RList)
 /\ Count' := X'

end role
```

```
role referee(R: agent, Ki, Vt: public_key,
            HonestAgents: public_key set) played_by R def=
 local State : nat, Me2: message,
        Vo, Vr: public_key,
        T : agent,
        Text : text,
        No, Nr, Nr2: text,
            SND, RCV: channel (dy)
 init State := 0

 transition

 %% The referee checks for the security condition described above.
 %% If it arises, he declared his own name R to be secret.
 %% This raises an attack since R is already in the intruder's
 %% initial knowledge.
 1. State = 0 /\ in(Vr', HonestAgents) /\
    RCV({aborted.
         {aborted.
          {Vo'.Vr'.T'.Text'.h(No')}_inv(Vo')}_inv(Vo')}_inv(Vt).
        {{Vo'.Vr'.T'.Text'.h(No')}_inv(Vo').h(Nr')}_inv(Vr').Nr'.
        {{Vo'.Vr'.T'.Text'.h(No')}_inv(Vo').h(Nr2')}_inv(Vr').Nr2') /\
    Nr' /= Nr2'
    =|>
    secret(R,secret_ref,{T'}) /\ State' := 1
end role
```

```
role session(O,R,T: agent,
             Vo,Vr,Vt: public_key,
             Text: text) def=

 composition
  orig(O,R,T,Text,Vo,Vr,Vt) /\
  resp(O,R,T,Text,Vo,Vr,Vt)
end role
```

```
role environment() def=

  local AList, RList: (message.message) set

  const succ: function,
        no, nr, no_secret, secret_ref: protocol_id,
        o, r, t, ref: agent,
        vo, vr, vt, ki: public_key,
        aborted, timeout, resolve, text1: text,
        h: function

  init AList = {}
    /\ RList = {}

 intruder_knowledge = {aborted, timeout, resolve,h,
                       o,r,t,ref,vo,vr,vt,ki,inv(ki),text1 }

 composition
  session(i,r,t,ki,vr,vt,text1) /\
  session(i,r,t,ki,vr,vt,text1) /\
  server(t,vt,AList,RList)/\
  referee(ref,ki,vt,{vo,vr})

end role
```

```
goal
  % Entity authentication (G1)
```

```
  % Message authentication (G2)
  % Replay protection (G3)
  % Accountability (G17)
  % Proof of origin (G18)
  % Proof of delivery (G19)
  authentication_on no
  authentication_on nr

  % Expressing fair exchange via observer (not described in D6.1)
  secrecy_of no_secret
  secrecy_of secret_ref
end goal

environment()
```

29 FairZG

Protocol Purpose

Fair Zhou Gollmann Non-Repudiation

Definition Reference

`http://citeseer.ist.psu.edu/62704.html`

Model Authors

Judson Santiago, LORIA Nancy, 2005

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

- weak authentication on `alice_bob_nrr`

- weak authentication on `bob_alice_nro`
- weak authentication on `bob_alice_sub`
- weak authentication on `alice_server_con`
- weak authentication on `bob_server_con`

Problem Classification: G1 G2 G18 G19

Attacks Found: None

HLPSL Specification

```
role alice ( A, B, S    : agent,
             Ka, Kb, Ks : public_key,
             Hash       : function,
             Snd, Rcv   : channel(dy)) played_by A def=

  local State : nat,
        M     : text,
        K     : symmetric_key,
        C     : {text}_symmetric_key,
        L,
        NRO,
        NRR,
        SubK,
        ConK  : message

  init   State = 0

  transition

   1.    State=0
      /\ Rcv(start)
      =|>
         State':=1
```

```
      /\ M':=new()
      /\ K':=new()
      /\ C':={M'}_K'
      /\ L':=Hash(M'.K')
      /\ NRO':={fNRO.B.L'.C'}_inv(Ka)
      /\ Snd(fNRO.B.L'.C'.NRO')

      % Non-repudiation of Origin
      /\ witness(A,B,bob_alice_nro,NRO')

   2.    State=1
      /\ Rcv(fNRR.A.L.NRR')
      /\ NRR' = {fNRR.A.L.C}_inv(Kb)
      =|>
         State':=2
      /\ SubK':={fSUB.B.L.K}_inv(Ka)
      /\ Snd(fSUB.B.L.K.SubK')

      % Non-repudiation of Submission
      /\ witness(A,B,bob_alice_sub,K)

   3.    State=2
      --|>
         State':=3
         /\ Snd(fREQ.A.B.L)

   4.    State=3
      /\ Rcv(fCON.A.B.L.K.ConK')
      /\ ConK'={fCON.A.B.L.K}_inv(Ks)
      =|>
         State':=4
      % Non-repudiation of Delivery
      /\ wrequest(A,S,alice_server_con,ConK')
      % Non-repudiation of Receipt
      /\ wrequest (A,B,alice_bob_nrr,NRR)

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,
      NRO,
      NRR,
      ConK: message

init State = 0

transition

 1.    State=0
    /\ Rcv(fNRO.B.L'.C'.NRO')
    /\ NRO'={fNRO.B.L'.C'}_inv(Ka)
    =|>
       State':=1
    /\ NRR':={fNRR.A.L'.C'}_inv(Kb)
    /\ Snd(fNRR.A.L'.NRR')
    % Non-repudiation of Receipt
    /\ witness (B,A,alice_bob_nrr,NRR')

 2.    State=1
    --|>
       State':=2
    /\ Snd(fREQ.A.B.L)

 3.    State=2
    /\ Rcv(fCON.A.B.L.K'.ConK')
    /\ ConK'={fCON.A.B.L.K'}_inv(Ks)
    /\ C = {M'}_K'
    =|>
       State':=3
    % Non-repudiation of Origin
    /\ wrequest(B,A,bob_alice_nro,NRO)
    % Non-repudiation of Delivery
    /\ wrequest(B,S,bob_server_con,ConK')
```

```
      % Non-repudiation of Origin/Submission
      /\ wrequest(B,A,bob_alice_sub,K')

end role
```

```
role server (S, A, B  : agent,
             Ks, Ka   : public_key,
             Snd, Rcv : channel (dy)) played_by S def=

  local State: nat,
        K    : symmetric_key,
        L,
        SubK,
        ConK : message

  init State = 0

  transition

   1.    State=0
      /\ Rcv(fSUB.B.L'.K'.SubK')
      /\ SubK'={fSUB.B.L'.K'}_inv(Ka)
      =|>
      State':=1

   2.    State=1
      /\ Rcv(fREQ.A.B.L)
      =|>
         State':=2
      /\ ConK':={fCON.A.B.L.K}_inv(Ks)
      /\ Snd(fCON.A.B.L.K.ConK')
      % Non-repudiation of Delivery
      /\ witness (S,A,alice_server_con,ConK')
      /\ witness (S,B,bob_server_con,ConK')

end role
```

```
role session(A,B,S: agent,
             Ka,Kb,Ks: public_key,
             H: function,
             Snd,Rcv: channel (dy)) def=

  composition
    alice(A,B,S,Ka,Kb,Ks,H,Snd,Rcv) /\
    bob(B,A,S,Kb,Ka,Ks,Snd,Rcv) /\
    server(S,A,B,Ks,Ka,Snd,Rcv)
end role
```

```
role environment() def=

 local Snd,Rcv: channel (dy)

 const a,b,s,i: agent,
       ka,kb,ks,ki: public_key,
       h: function,
       alice_bob_nrr,
       bob_alice_nro,
       bob_alice_sub,
       alice_server_con,
       bob_server_con,
       fREQ,fNRO,fNRR,fSUB,fCON: protocol_id

 intruder_knowledge = {a,b,s,ka,kb,ks,ki,i,inv(ki),fNRO,fNRR,fSUB,fCON}

 composition

       session(a,b,s,ka,kb,ks,h,Snd,Rcv)
    /\ session(a,b,s,ka,kb,ks,h,Snd,Rcv)

% The non-repudiation property can not be described like authentification
% problems when the intruder plays a role in the protocol. The intruder
% does not fire the witnesses on behalf of the honest agents and there
% will be always an false attack.
%
%   /\ session(a,i,s,ka,ki,ks,h,Snd,Rcv)
%   /\ session(i,b,s,ki,kb,ks,h,Snd,Rcv)
```

```
%   /\ session(a,b,i,ka,kb,ki,h,Snd,Rcv)

end role
```

```
goal

   % All authentications together provide
   % entity authentication (G1) and
   % message authentication (G2)

   % alice_bob_nrr and alice_server_con provide
   % proof of delivery (G19)

   % bob_alice_nro, bob_alice_sub and bob_server_con
   % provide proof of origin (G18)

   weak_authentication_on alice_bob_nrr

   weak_authentication_on bob_alice_nro

   weak_authentication_on bob_alice_sub

   weak_authentication_on alice_server_con

   weak_authentication_on bob_server_con

end goal
```

```
environment()
```

30 SET Purchase Request, and Payment Authorization

30.1 Original

Protocol Purpose

The Secure Electronic Transactions (SET) Protocol Suite is designed to allow for a secure e-commerce. The key feature is to hide the customer's credit card details from the merchant, and the customer's purchase details from the bank. Rather, by the construction of the protocol, both merchant and bank see only what they need to see in order to complete the transaction. Following [BMP01] we focus here on the main part of the protocol, purchase request and payment authorization, leaving out the initial registration protocols and assuming already registered participants. Note that we do allow dishonest participants.

Definition Reference

[BMP01], [MV77]

Model Authors

Sebastian Mödersheim, ETH Zürich

Alice&Bob style

The protocol involves three parties: Cardholder `C`, Merchant `M`, and Payment Gateway `P`. The cryptographic constructions of this protocol are quite complex and for readability we thus use the following macros:

- `Sign_A(Msg)=Msg.{h(Msg)}_inv(SignK(A))`
- `Encrypt_B(Msg1,K,Msg2)={Msg2}_K.{Msg1.K}EncK(B)`. Note that we explicitly give a symmetric key that is used in the encryption and that is transmitted in a digital envelope together with `Msg1` that is most precious.
- `SignCert(M)={A.SignK(A)}_inv(SignK(CA))`
- `EncCert(A)={A.EncK(A)}_inv(SignK(CA))`
- `DualSig_A(M1,M2) = Sign_A(h(M1).h(M2))`

Further, for the communicated data, we use the following abbreviations (in accordance with the business specification of SET and the model of [BMP01]):

- `P_Init_Req=LID_M.Chall_C`
  The purchase initiate request, which consists of an identifier and a challenge chosen by the cardholder, both modelled as nonces;

- `P_Init_Resp=LID_M.Chall_C.XID.Chall_M`
  The response to an initiate request, containing an identifier and a challenge from the merchant, modelled as nonces.

- `AI`
  Account Information, the details of the credit card of the cardholder. This is the most precious secret of the protocol.

- `OI=XID.Chall_C.h(OrderDesc.PurchAmt).Chall_M`
  Order Information, which contains a hash-value of the order description and the purchase amount. These data are negotiated out-of-band before the protocol and both cardholder and merchant initially share this information.

- `PI=LID_M.XID.h(OrderDesc.PurchAmt).PurchAmt.M.h(XID.AI)`
  Payment Information, containing a hash of the account information

- `AuthReq=LID_M.XID.h(OI).h(OrderDesc.PurchAmt).DualSign_C(PI,OI)`

- `Auth_Res=LID_M.XID.PurchAmt`

- `Purch_Res=LID_M.XID.Chall_C.h(PurchAmt)`

Purchase Request Protocol:

```
% Purchase Initiate Request
1. C->M: P_Init_Req
% Purchase Initiate Response
2. M->C: Sign_M(P_Init_Resp).SignCert(M).EncCert(P)
% Purchase Request
3. C->M: OI.DualSig_C(OI,PI).
         Encrypt_P(AI,K1,DualSig_C(OI,PI).PI).
         SignCert(C)
% Purchase Response
4. M->C: Sign_M(Purch_Res).SignCert(M)
```

Payment Authorization Protocol:

```
% Payment Authorization Request
1. M->P: Encrypt_P(_,K2,Sign_M(AuthReq)).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI).
         SignCert(C).SignCert(M).EncCert(M)
% Payment Authorization Response
2. P->M: Encrypt_M(K3,Sign_P(Auth_Res)).
         Encrypt_P(AI,K4,Sign_P(Cap_Token)).
         SignCert(P)
```

Following [BMP01], we simplify this into one protocol, omitting certificates (we assume that all agents initially have each other's public-keys) and remove the sub-message that in the payment authorization response step which the payment gateway encrypts to itself:

```
% Purchase Initiate Request
1. C->M: P_Init_Req
% Purchase Initiate Response
2. M->C: Sign_M(P_Init_Resp)
% Purchase Request
3. C->M: OI.DualSig_C(OI,PI).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI)
% Payment Authorization Request
4. M->P: Encrypt_P(_,K2,Sign_M(AuthReq)).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI)
% Payment Authorization Response
5. P->M: Encrypt_M(K3,SignK_P(Auth_Res))
% Purchase Response
6. M->C: Sign_M(Purch_Res)
```

We consider the following goals: the parties shall authenticate each other on (the hash of) the order and payment information. Moreover, the order information shall remain secret between cardholder and merchant, and the payment information, in particular the credit card details, shall remain secret between cardholder and payment gateway.

Model Limitations

We have abstracted from the following details:

- The shopping process itself, i.e. selection of goods and computing the price to pay.
- The registration of the participants.

- The public-key infrastructure and verification of certificates (assuming everybody already has each other's public key).
- Omitting a special PAN and PANSecret (assuming AI data is sufficient for payment gateway).

Problems Considered: 4

- authentication on `deal`
- weak authentication on `deal`
- secrecy of `order`
- secrecy of `payment`

Attacks Found:

The first attack is that a dishonest payment gateway p can forward a payment authorization request to any other payment gateway p'. In a nutshell, the attack trace has the form

```
...
m     -> p:  Encrypt_p (_,K2,Sign_M(AuthReq)).
             Encrypt_p (AI,K1,DualSig_C(OI,PI).PI)
p(m)  -> p': Encrypt_p'(_,K2,Sign_M(AuthReq)).
             Encrypt_p'(AI,K1,DualSig_C(OI,PI).PI)
...
```

This is due to the fact that the part of the message signed by the cardholder (as well as the one signed by the merchant) does not contain the name of the desired payment gateway. Rather, the payment gateway is only determined by the public-key encryption for the desired gateway. Though only a dishonest payment gateway can "forward" the payment requests, this may lead to the situation that two payment gateways charge the account of the cardholder and both posses messages that seem to prove that the cardholder authorized the transaction. The vulnerability is limited by the fact that the payment gateway actually chosen by the cardholder must be dishonest in the first place. In our opinion, one should definitely include the name of the payment gateway also in the dual signature to prevent this situation.

We also found a second, albeit quite artificial, attack. For the simplicity of the presentation, we do not display here the complete attack trace. Let c be an honest cardholder, m and m' be an honest and a dishonest merchant, and p and p' be an honest and a dishonest payment gateway. (All dishonest parties cooperate and are thus merged into one intruder.) Consider a session between c, m and p', and a session between c, m', and p, which all run according to the protocol. Let *lid*, *xid*, *orderdesc*, *purchamt* be the data of the session between c, m', and p. Let finally *ai* be the account information of c. Then the intruder (i.e. m' and p' together) can construct the message

```
lid.xid.h(orderdesc.purchamt).purchamt.m'.h(xid.ai_c)
```

which is the payment information that only the honest participants c and p are supposed to see. It is thus possible that the secrecy of the payment information between an honest cardholder and an honest payment gateway is violated, if a dishonest merchant and a dishonest payment gateway cooperate. However, the relevance of this attack is questionable. It is standard to check protocols under the assumption of dishonest players, and it is clear that in such sessions secrecy guarantees, for instance, are void (as the intruder knows the secrets). The question is rather whether such a session can also compromise the security goals of other sessions (as it is the case for instance in the well-known attack against the Needham-Schroeder Public Key Protocol). For the SET protocol, however, it is clear that, once an honest cardholder runs a session with a dishonest payment gateway, the account-information of the cardholder is compromised in *all* sessions; it is thus not surprizing that the payment information from a session with an honest payment gateway can also be reconstructed in such a case. Note that this attack is not possible without a dishonest merchant, i.e. even though a dishonest payment gateway knows the account details, it cannot obtain order information of sessions with an honest merchant.

Further Notes

- There is nothing in the messages that ensures freshness for the payment gateway. However it is unreasonable to assume a payment gateway would not log the payment requests it has received. So we can assume that it won't accept a second time a message with the same identifiers `LID_M` and `XID`, and thus check only for weak authentication from the gateways point-of-view.

- The cardholder cannot be sure, upon receiving the final purchase response message from the merchant, that the payment gateway has actually seen the transaction. This is not very surprizing as this message can be sent by the merchant without first contacting the payment gateway (the merchant then looses the guarantee that he will receive the money).

HLPSL Specification

```
role cardholder(C,M,P: agent,
                AI : text,
                PurchAmt : nat,
                OrderDesc : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) played_by C def=

 local S : nat,
       LID_M, Chall_C : text (fresh),
       XID, Chall_M : text,
       OI,PI,DualSig : message,
       K1 : symmetric_key (fresh),
       SND, RCV: channel (dy)

 init S := 0

 transition

 % =|> Purchase Initiate Request
 1. S = 0 /\
    RCV(start)
    =|>
    S' := 1 /\
    LID_M' := new() /\
    Chall_C' := new() /\
    SND(LID_M'.Chall_C')

 % Purchase Initiate Response =|> Purchase Request
 2. S = 1 /\
    RCV(LID_M.Chall_C.XID'.Chall_M'.
        {h(LID_M.Chall_C.XID'.Chall_M')}_inv(SignK_M))
    =|>
    S' := 2 /\
    OI' := XID'.Chall_C.h(OrderDesc.PurchAmt).Chall_M' /\
    PI' := LID_M.XID'.h(OrderDesc.PurchAmt).PurchAmt.M.h(XID'.AI) /\
    DualSig' := h(OI').h(PI').{h(h(OI').h(PI'))}_inv(SignK_C) /\
    K1' := new() /\
```

```
    SND(OI'.DualSig'.
        {DualSig'.PI'}_K1'.{AI.K1'}_EncK_P) /\
    witness(C,M,deal,OI'.h(PI')) /\
    witness(C,P,deal,OI'.PI') /\
    secret(OrderDesc,order,{C,M}) /\
    secret(PurchAmt,order,{C,M,P}) /\
    secret(PI',payment,{C,P})

 % Purchase Response =|>
 3. S = 2 /\
    RCV(LID_M.XID.Chall_C.h(PurchAmt).
        {h(LID_M.XID.Chall_C.h(PurchAmt))}_inv(SignK_M))
    =|>
    S' := 3 /\
    request(C,M,deal,OI.h(PI))
    % /\ request(C,P,deal,OI.PI) %% cannot be done; see notes, item 2

end role
```

```
role merchant  (C,M,P: agent,
                PurchAmt : nat,
                OrderDesc : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) played_by M def=

 local S : nat,
       LID_M, Chall_C : text ,
       XID, Chall_M : text (fresh),
       OI,HPI,DualSig,Paymentpart,AuthReq : message,
       K2 : symmetric_key (fresh),
       K3 : symmetric_key,
       SND, RCV: channel (dy)

 init S := 0

 transition
```

```
 % Purchase Initiate Request =|> Purchase Initiate Response
 1. S = 0 /\
    RCV(LID_M'.Chall_C')
    =|>
    S' := 1 /\
    XID' := new() /\
    Chall_M' := new() /\
    SND(LID_M'.Chall_C'.XID'.Chall_M'.
        {h(LID_M'.Chall_C'.XID'.Chall_M')}_inv(SignK_M))

 % Purchase Request =|> Payment Authorization Request
 2. S = 1 /\
    RCV(XID.Chall_C.h(OrderDesc.PurchAmt).Chall_M.
        h(OI').HPI'.{h(h(OI').HPI')}_inv(SignK_C).
        Paymentpart') /\
    OI' = XID.Chall_C.h(OrderDesc.PurchAmt).Chall_M
    =|>
    S' := 2 /\
    DualSig' := h(OI').HPI'.{h(h(OI').HPI')}_inv(SignK_C) /\
    K2' := new() /\
    AuthReq' := LID_M.XID.h(OI').h(OrderDesc.PurchAmt).
                DualSig' /\
    SND({AuthReq'.{h(AuthReq')}_inv(SignK_M)}_K2'.{K2'}_EncK_P.
        Paymentpart') /\
    witness(M,C,deal,OI'.HPI') /\
    witness(M,P,deal,OI'.HPI')

 % Payment Authorization Response =|> Purchase Response
 3. S = 2 /\
    RCV({LID_M.XID.PurchAmt.
         {h(LID_M.XID.PurchAmt)}_inv(SignK_P)}_K3'.{K3'}_EncK_M)
    =|>
    S' := 3 /\
    SND(LID_M.XID.Chall_C.h(PurchAmt).
        {h(LID_M.XID.Chall_C.h(PurchAmt))}_inv(SignK_M)) /\
    request(M,C,deal,OI.HPI) /\
    request(M,P,deal,OI.HPI)

end role
```

```
role paymentgateway(C,M,P: agent,
                AI : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) played_by P def=

 local S : nat,
       XID, Chall_M, LID_M, Chall_C : text ,
       AuthReq,Paymentpart,OI,PI,DualSig : message,
       K1,K2 : symmetric_key,
       K3 : symmetric_key (fresh),
       PurchAmt : nat,
       OrderDesc : text,
       SND, RCV: channel (dy)

 init S := 0

 transition

 % Payment Authorization Request =|> Payment Authorization Response

 1. S = 0 /\
    RCV({AuthReq'.{h(AuthReq')}_inv(SignK_M)}_K2'.{K2'}_EncK_P.
        {DualSig'.PI'}_K1'.{AI.K1'}_EncK_P
       ) /\
    AuthReq' = LID_M'.XID'.h(OI').h(OrderDesc'.PurchAmt').DualSig' /\
    OI' = XID'.Chall_C'.h(OrderDesc'.PurchAmt').Chall_M' /\
    DualSig' = h(OI').h(PI').{h(h(OI').h(PI'))}_inv(SignK_C) /\
    PI' = LID_M'.XID'.h(OrderDesc'.PurchAmt').PurchAmt'.M.h(XID'.AI)
    =|>
    S' := 1 /\
    K3' := new() /\
    SND({LID_M'.XID'.PurchAmt'.
         {h(LID_M'.XID'.PurchAmt')}_inv(SignK_P)}_K3'.{K3'}_EncK_M) /\
    wrequest(P,C,deal,OI'.PI') /\
    wrequest(P,M,deal,OI'.h(PI')) /\
    witness(P,C,deal,OI'.PI') /\
    witness(P,M,deal,OI'.h(PI'))
```

```
end role
```

```
role session(C,M,P: agent,
                AI : text,
                PurchAmt : nat,
                OrderDesc : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) def=

% local SI, RI, SR, RR: channel(dy)

 composition
  cardholder(C,M,P,AI,PurchAmt,OrderDesc,
             EncK_C,SignK_C,EncK_M,SignK_M,EncK_P,SignK_P) /\
  merchant  (C,M,P,   PurchAmt,OrderDesc,
             EncK_C,SignK_C,EncK_M,SignK_M,EncK_P,SignK_P) /\
  paymentgateway(C,M,P,AI,
             EncK_C,SignK_C,EncK_M,SignK_M,EncK_P,SignK_P)
end role
```

```
role environment() def=

  local AList, RList: (message.message) set,
        S2, R2, S3, R3: channel (dy)

  const h: function,
        deal,order,payment : protocol_id,
        c,m,p: agent,
        enc_c,sign_c,enc_m,sign_m,enc_p,sign_p,enc_i,sign_i: public_key,
        ai_c,ai_i,od1,od2,od3,od4,od5: text,
        pa1,pa2,pa3,pa4,pa5 : nat

 intruder_knowledge = {c,m,p,enc_c,sign_c,enc_m,sign_m,enc_p,sign_p,
         enc_i,sign_i,inv(enc_i),inv(sign_i),ai_i,pa3,od3,pa4,od4,h }
```

```
 composition
  session(c,m,p,ai_c,pa2,od2,enc_c,sign_c,enc_m,sign_m,enc_p,sign_p) /\
%  session(i,m,p,ai_i,pa3,od3,enc_i,sign_i,enc_m,sign_m,enc_p,sign_p) /\
  session(c,i,p,ai_c,pa4,od4,enc_c,sign_c,enc_i,sign_i,enc_p,sign_p) /\
  session(c,m,i,ai_c,pa5,od5,enc_c,sign_c,enc_m,sign_m,enc_i,sign_i)

end role
```

```
goal

  % Entity authentication (G1)
  % Message authentication (G2)
  % Replay protection (G3)
  % Accountability (G17)
  % Proof of Origin (G18)
  % Proof of Delivery (G19)
  authentication_on deal
  weak_authentication_on deal

  % ID protection (Eavesdr.) (G13)
  % Conifidentiality (G12) --- Missing in table of D6.1
  secrecy_of order
  secrecy_of payment
end goal

environment()
```

30.2 Instantiation with only honest payment gateways

Protocol Purpose

The Secure Electronic Transactions (SET) Protocol Suite is designed to allow for a secure e-commerce. The key feature is to hide the customer's credit card details from the merchant, and the customer's purchase details from the bank. Rather, by the construction of the protocol, both merchant and bank see only what they need to see in order to complete the transaction. Following [BMP01] we focus here on the main part of the protocol, purchase request and payment authorization, leaving out the initial registration protocols and assume already registered participants. Note that we do allow dishonest participants.

Definition Reference

[BMP01], [MV77]

Model Authors

Sebastian Mödersheim, ETH Zürich

Alice&Bob style

The protocol involves three parties: Cardholder `C`, Merchant `M`, and Payment Gateway `P`. The cryptographic constructions of this protocol are quite complex and for readability we thus use the following macros:

- `Sign_A(Msg)=Msg.{h(Msg)}_inv(SignK(A))`
- `Encrypt_B(Msg1,K,Msg2)={Msg2}_K.{Msg1.K}EncK(B)`. Note that we explicitly give a symmetric key that is used in the encryption and that is transmitted in a digital envelope together with `Msg1` that is most precious.
- `SignCert(A)={A.SignK(A)}_inv(SignK(CA))`
- `EncCert(A)={A.EncK(A)}_inv(SignK(CA))`
- `DualSig_A(M1,M2) = Sign_A(h(M1).h(M2))`

Further, for the communicated data, we use the following abbreviations (in accordance with the business specification of SET and the model of [BMP01]):

- `P_Init_Req=LID_M.Chall_C`
  The purchase initiate request, which consists of an identifier and a challenge chosen by the cardholder, both modelled as nonces;
- `P_Init_Resp=LID_M.Chall_C.XID.Chall_M`
  The response to an initiate request, containing an identifier and a challenge from the merchant, modelled as nonces.
- `AI`
  Account Information, the details of the credit card of the cardholder. This is the most precious secret of the protocol.
- `OI=XID.Chall_C.h(OrderDesc.PurchAmt).Chall_M`
  Order Information, which contains a hash-value of the order description and the purchase amount. These data are negotiated out-of-band before the protocol and both cardholder and merchant initially share this information.

- `PI=LID_M.XID.h(OrderDesc.PurchAmt).PurchAmt.M.h(XID.AI)`
  Payment Information, containing a hash of the account information
- `AuthReq=LID_M.XID.h(OI).h(OrderDesc.PurchAmt).DualSign_C(PI,OI)`
- `Auth_Res=LID_M.XID.PurchAmt`
- `Purch_Res=LID_M.XID.Chall_C.h(PurchAmt)`

Purchase Request Protocol:

```
% Purchase Initiate Request
1. C->M: P_Init_Req
% Purchase Initiate Response
2. M->C: Sign_M(P_Init_Resp).SignCert(M).EncCert(P)
% Purchase Request
3. C->M: OI.DualSig_C(OI,PI).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI).
          SignCert(C)
% Purchase Response
4. M->C: Sign_M(Purch_Res).SignCert(M)
```

Payment Authorization Protocol:

```
% Payment Authorization Request
1. M->P: Encrypt_P(_,K2,Sign_M(AuthReq)).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI).
         SignCert(C).SignCert(M).EncCert(M)
% Payment Authorization Response
2. P->M: Encrypt_M(K3,Sign_P(Auth_Res)).
         Encrypt_P(AI,K4,Sign_P(Cap_Token)).
         SignCert(P)
```

Following [BMP01], we simplify this into one protocol, omitting certificates (we assume that all agents initially have each other's public-keys) and remove the sub-message that in the payment authorization response step which the payment gateway encrypts to itself:

```
% Purchase Initiate Request
1. C->M: P_Init_Req
% Purchase Initiate Response
2. M->C: Sign_M(P_Init_Resp)
```

```
% Purchase Request
3. C->M: OI.DualSig_C(OI,PI).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI)
% Payment Authorization Request
4. M->P: Encrypt_P(_,K2,Sign_M(AuthReq)).
          Encrypt_P(AI,K1,DualSig_C(OI,PI).PI)
% Payment Authorization Response
5. P->M: Encrypt_M(K3,SignK_P(Auth_Res))
% Purchase Response
6. M->C: Sign_M(Purch_Res)
```

We consider the following goals: the parties shall authenticate each other on (the hash of) of the order and payment information. Moreover, the order information shall remain secret between cardholder and merchant, and the payment information, in particular the credit card details, shall remain secret between cardholder and payment gateway.

Model Limitations

We have abstracted from the following details:

- The shopping process itself, i.e. selection of goods and computing the price to pay
- The registration of the participants
- The public-key infrastructure and verification of certificates (assuming everybody already has each other's public key)
- Omitting a special PAN and PANSecret (assuming AI data is sufficient for payment gateway).

Problems Considered: 4

- authentication on `deal`
- weak authentication on `deal`
- secrecy of `order`
- secrecy of `payment`

Attacks Found: None

Further Notes

In this variant the payment gateway role is played only by honest participants to avoid the attacks of found in the case with dishonest payment gateways.

- There is nothing in the messages that ensures freshness for the payment gateway. However it is unreasonable to assume a payment gateway would not log the payment requests it has received. So we can assume that it won't accept a second time a message with the same identifiers `LID_M` and `XID`, and thus check only for weak authentication from the gateways point-of-view.
- The cardholder cannot be sure, upon receiving the final purchase response message from the merchant, that the payment gateway has actually seen the transaction. This is not very surprizing as this message can be sent by the merchant without first contacting the payment gateway (the merchant then looses the guarantee that the he will receive the money).

HLPSL Specification

```
role cardholder(C,M,P: agent,
                AI : text,
                PurchAmt : nat,
                OrderDesc : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) played_by C def=

 local S : nat,
       LID_M, Chall_C : text (fresh),
       XID, Chall_M : text,
       OI,PI,DualSig : message,
       K1 : symmetric_key (fresh),
       SND, RCV: channel (dy)

 init S := 0
```

```
transition

% =|> Purchase Initiate Request
1. S = 0 /\
   RCV(start)
   =|>
   S' := 1 /\
   LID_M' := new() /\
   Chall_C' := new() /\
   SND(LID_M'.Chall_C')

% Purchase Initiate Response =|> Purchase Request
2. S = 1 /\
   RCV(LID_M.Chall_C.XID'.Chall_M'.
       {h(LID_M.Chall_C.XID'.Chall_M')}_inv(SignK_M))
   =|>
   S' := 2 /\
   OI' := XID'.Chall_C.h(OrderDesc.PurchAmt).Chall_M' /\
   PI' := LID_M.XID'.h(OrderDesc.PurchAmt).PurchAmt.M.h(XID'.AI) /\
   DualSig' := h(OI').h(PI').{h(h(OI').h(PI'))}_inv(SignK_C) /\
   K1' := new() /\
   SND(OI'.DualSig'.
       {DualSig'.PI'}_K1'.{AI.K1'}_EncK_P) /\
   witness(C,M,deal,OI'.h(PI')) /\
   witness(C,P,deal,OI'.PI') /\
   secret(OrderDesc,order,{C,M}) /\
   secret(PurchAmt,order,{C,M,P}) /\
   secret(PI',payment,{C,P}) /\
   secret(AI,payment,{C,P})

% Purchase Response =|>
3. S = 2 /\
   RCV(LID_M.XID.Chall_C.h(PurchAmt).
       {h(LID_M.XID.Chall_C.h(PurchAmt))}_inv(SignK_M))
   =|>
   S' := 3 /\
   request(C,M,deal,OI.h(PI))
   % /\ request(C,P,deal,OI.PI) %% cannot be done; see notes, item 2

end role
```

```
role merchant  (C,M,P: agent,
                PurchAmt : nat,
                OrderDesc : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) played_by M def=

 local S : nat,
       LID_M, Chall_C : text ,
       XID, Chall_M : text (fresh),
       OI,HPI,DualSig,Paymentpart,AuthReq : message,
       K2 : symmetric_key (fresh),
       K3 : symmetric_key,
       SND, RCV: channel (dy)

 init S := 0

 transition

 % Purchase Initiate Request =|> Purchase Initiate Response
 1. S = 0 /\
    RCV(LID_M'.Chall_C')
    =|>
    S' := 1 /\
    XID' := new() /\
    Chall_M' := new() /\
    SND(LID_M'.Chall_C'.XID'.Chall_M'.
        {h(LID_M'.Chall_C'.XID'.Chall_M')}_inv(SignK_M))

 % Purchase Request =|> Payment Authorization Request
 2. S = 1 /\
    RCV(XID.Chall_C.h(OrderDesc.PurchAmt).Chall_M.
        h(OI').HPI'.{h(h(OI').HPI')}_inv(SignK_C).
        Paymentpart') /\
    OI' = XID.Chall_C.h(OrderDesc.PurchAmt).Chall_M
    =|>
    S' := 2 /\
    DualSig' := h(OI').HPI'.{h(h(OI').HPI')}_inv(SignK_C) /\
```

```
    K2' := new() /\
    AuthReq' := LID_M.XID.h(OI').h(OrderDesc.PurchAmt).
                DualSig' /\
    SND({AuthReq'.{h(AuthReq')}_inv(SignK_M)}_K2'.{K2'}_EncK_P.
        Paymentpart') /\
    witness(M,C,deal,OI'.HPI') /\
    witness(M,P,deal,OI'.HPI')

 % Payment Authorization Response =|> Purchase Response
 3. S = 2 /\
    RCV({LID_M.XID.PurchAmt.
         {h(LID_M.XID.PurchAmt)}_inv(SignK_P)}_K3'.{K3'}_EncK_M)
    =|>
    S' := 3 /\
    SND(LID_M.XID.Chall_C.h(PurchAmt).
        {h(LID_M.XID.Chall_C.h(PurchAmt))}_inv(SignK_M)) /\
    request(M,C,deal,OI.HPI) /\
    request(M,P,deal,OI.HPI)

end role
```

```
role paymentgateway(C,M,P: agent,
                AI : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) played_by P def=

 local S : nat,
       XID, Chall_M, LID_M, Chall_C : text ,
       AuthReq,Paymentpart,OI,PI,DualSig : message,
       K1,K2 : symmetric_key,
       K3 : symmetric_key (fresh),
       PurchAmt : nat,
       OrderDesc : text,
       SND, RCV: channel (dy)

 init S := 0
```

```
 transition

 % Payment Authorization Request =|> Payment Authorization Response

 1. S = 0 /\
    RCV({AuthReq'.{h(AuthReq')}_inv(SignK_M)}_K2'.{K2'}_EncK_P.
        {DualSig'.PI'}_K1'.{AI.K1'}_EncK_P
       ) /\
    AuthReq' = LID_M'.XID'.h(OI').h(OrderDesc'.PurchAmt').DualSig' /\
    OI' = XID'.Chall_C'.h(OrderDesc'.PurchAmt').Chall_M' /\
    DualSig' = h(OI').h(PI').{h(h(OI').h(PI'))}_inv(SignK_C) /\
    PI' = LID_M'.XID'.h(OrderDesc'.PurchAmt').PurchAmt'.M.h(XID'.AI)
    =|>
    S' := 1 /\
    K3' := new() /\
    SND({LID_M'.XID'.PurchAmt'.
         {h(LID_M'.XID'.PurchAmt')}_inv(SignK_P)}_K3'.{K3'}_EncK_M) /\
    wrequest(P,C,deal,OI'.PI') /\
    wrequest(P,M,deal,OI'.h(PI')) /\
    witness(P,C,deal,OI'.PI') /\
    witness(P,M,deal,OI'.h(PI'))

end role
```

```
role session(C,M,P: agent,
                AI : text,
                PurchAmt : nat,
                OrderDesc : text,
                EncK_C, SignK_C,
                EncK_M, SignK_M,
                EncK_P, SignK_P : public_key
               ) def=

% local SI, RI, SR, RR: channel(dy)

 composition
  cardholder(C,M,P,AI,PurchAmt,OrderDesc,
             EncK_C,SignK_C,EncK_M,SignK_M,EncK_P,SignK_P) /\
  merchant  (C,M,P,   PurchAmt,OrderDesc,
```

```
             EncK_C,SignK_C,EncK_M,SignK_M,EncK_P,SignK_P) /\
  paymentgateway(C,M,P,AI,
             EncK_C,SignK_C,EncK_M,SignK_M,EncK_P,SignK_P)
end role
```

```
role environment() def=

  local AList, RList: (message.message) set,
        S2, R2, S3, R3: channel (dy)

  const h: function,
        deal,order,payment : protocol_id,
        c,m,p: agent,
        enc_c,sign_c,enc_m,sign_m,enc_p,sign_p,enc_i,sign_i: public_key,
        ai_c,ai_i,od1,od2,od3,od4,od5: text,
        pa1,pa2,pa3,pa4,pa5 : nat

 intruder_knowledge = {c,m,p,enc_c,sign_c,enc_m,sign_m,enc_p,sign_p,
        enc_i,sign_i,inv(enc_i),inv(sign_i),ai_i,pa3,od3,pa4,od4,h }

 composition
  session(c,m,p,ai_c,pa2,od2,enc_c,sign_c,enc_m,sign_m,enc_p,sign_p) /\
  session(i,m,p,ai_i,pa3,od3,enc_i,sign_i,enc_m,sign_m,enc_p,sign_p)
  % /\ session(c,i,p,ai_c,pa4,od4,enc_c,sign_c,enc_i,sign_i,enc_p,sign_p)

end role
```

```
goal

  % Entity authentication (G1)
  % Message authentication (G2)
  % Replay protection (G3)
  % Accountability (G17)
  % Proof of Origin (G18)
  % Proof of Delivery (G19)
  authentication_on deal
  weak_authentication_on deal
```

```
  % ID protection (Eavesdr.) (G13)
  % Conifidentiality (G12) --- Missing in table of D6.1
  secrecy_of order
  secrecy_of payment
end goal

environment()
```

Part IV
Non IETF Protocols

31 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 `sseq1,sseq2`
- weak authentication on `r1`
- weak authentication on `r2`

CLASSIFICATON: G2 G12

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

  const r1,r2,sseq1 : protocol_id,
        add         : function

  init  State := 1

  transition

    1.  State   = 1 /\ Rec(M)
        =|>
        State' := 2 /\ R' := new()
                    /\ Snd(R'.{Seq}_F5(K_M.R').F1(K_M.Seq.R'))
                    /\ secret(Seq,sseq1,{S,M})
                    /\ 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

  const
        r1,r2,sseq2 : protocol_id

  init  State := 1

  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,sseq2,{M,S})
                  /\ 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,
       k_as, k_is, kai        : symmetric_key,
       f1, f2, f5             : function,
       seq_as, seq_is, seq_ai : text

 intruder_knowledge={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

  % Confidentiality (G12)
  secrecy_of sseq1,sseq2

  % Message Authentication (G2)
```

```
  % Mobile weakly authenticates Server on r1  % the nonce R
  weak_authentication_on r1

  % Message Authentication (G2)
  % Server weakly authenticates Mobile on r2  % the nonce R
  weak_authentication_on r2

end goal
```

```
environment()
```

32 ISO1 Public Key Unilateral Authentication Protocol

32.1 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

- authentication on `na`

Problem Classification: G1, G2

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

 init  State := 0

 transition

   1. State = 0
      /\ Rcv(start)
      =|>
      State' := 1
      /\ Na' := new()
      /\ Snd(Pka.A.{Pka.A}_inv(Pks).Na'.B.ctext.{Na'.B.ctext}_inv(Pka))
      /\ 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

  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

  intruder_knowledge={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
   authentication_on na % addressess G1 and G2

end goal
```

```
environment()
```

32.2 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

- authentication on `ra`

Problem Classification: G1, G2

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,
         Ra, Text2 : text

  init State := 0

  transition
```

```
   1. State = 0
      /\ Rec(start)
      =|>
      State' := 1
      /\ Rb' := new()
      /\ Snd(Rb'.ctext1)

   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,
         Rb, Text1  : text

  init State := 0

  transition

   1. State = 0
      /\ Rec(Rb'.Text1')
      =|>
      State' := 2
      /\ Ra' := new()
      /\ Snd(Pka.A.{Pka.A}_inv(Pks).Ra'.Rb'.B.ctext2.
                    {Ra'.Rb'.B.Text1'}_inv(Pka))
      /\ 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,
         pkb,pks,pki   : public_key

  intruder_knowledge={i,a,b,pks,pki,inv(pki),ctext1,ctext2,
                      {pki.i}_inv(pks)}

  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
```

```
   authentication_on ra % addressess G1 and G2

end goal
```

```
environment()
```

32.3 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

- weak authentication on `nb`
- weak authentication on `na`

Problem Classification: G1, G2

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,
         Nb, Text3, Text4  : text,
         Pkb               : public_key

  init State := 0
```

```
  transition

   1. State = 0
      /\ Rcv(start)
      =|>
      State' := 1
      /\ Na' := new()
      /\ Snd(Pka.A.{Pka.A}_inv(Pks).Na'.B.ctext2.{Na'.B.ctext1}_inv(Pka))
      /\ 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,
          Na,Text1,Text2  : text,
          Pka             : public_key

   init State := 0

   transition

   1. State = 0
      /\ Rcv(Pka'.A.{Pka'.A}_inv(Pks).Na'.B.Text2'.{Na'.B.Text1'}_inv(Pka'))
      =|>
      State' := 1
      /\ Nb' := new()
```

```
      /\ Snd(Pkb.B.{Pkb.B}_inv(Pks).Nb'.A.ctext4.{Nb'.A.ctext3}_inv(Pkb))
      /\ 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

  intruder_knowledge={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
   weak_authentication_on nb % addressess G1 and G2

   %ISO3_Resp weakly authenticates ISO3_Init on na
   weak_authentication_on na % addressess G1 and G2

end goal
```

```
environment()
```

32.4 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

- authentication on `nb`
- authentication on `na`

Problem Classification: G1, G2

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,
         Na,Text2,Text3: text

  const ctext1,ctext4,ctext5: text

  init State := 0

  transition

   1. State = 0
      /\ Rec(start)
      =|>
      State' := 1
```

```
      /\ Nb' := new()
      /\ Snd(Nb'.ctext1)
      /\ 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'))
      =|>
      State' := 2
      /\ Snd(Pkb.B.{Pkb.B}_inv(Pks).Nb.Na'.A.ctext5.{Nb.Na'.A.ctext4}_inv(Pkb))
      /\ 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,
         Nb,Text1,Text4,Text5: text

  const ctext2,ctext3: text

  init State := 0

  transition

   1. State = 0
      /\ Rec(Nb'.Text1')
      =|>
      State' := 1
      /\ Na' := new()
      /\ Snd(Pka.A.{Pka.A}_inv(Pks).
             Na'.Nb'.B.ctext3.{Na'.Nb'.B.ctext2}_inv(Pka))
      /\ 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

 intruder_knowledge={a,b,pki,inv(pki),pks,
                     ctext1,ctext4,ctext5,{pki.i}_inv(pks),
                     ctext2,ctext3,{pki.i}_inv(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
   authentication_on nb % addressess G1 and G2

   %ISO4_Init authenticates ISO4_Resp on na
   authentication_on na % addressess G1 and G2

end goal
```

```
environment()
```

33 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

- authentication 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,
        M,SSM: message

  init State = 0

  transition
    1.      State = 0
            /\ RCV(start)
        =|>
            State' := 1
            /\ M0' := new()
            /\ M'  := H(M0')                        % using   hashed message
```

```
            %/\ M' := MO'                          % 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

  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

  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)
def=

  local
        CS, SC : channel (dy)

  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=

  const c,bc,s     : agent,
        kbc,ks,ki  : public_key,
        h          : function

  intruder_knowledge = {c,bc,s, h, kbc,ks,ki,inv(ki)}

  composition
     session(c,bc,s,kbc,ks,h)
  /\ session(c,bc,s,kbc,ks,h)
  /\ session(c,i, s,ki, ks,h)

end role
```

```
goal
  %Consumer weakly authenticates Server on m
  authentication_on m
end goal
```

```
environment()
```

34 LPD: Low-Powered Devices

34.1 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 secx
- weak authentication on x

Problem Classification: G1, G2, G12

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

  transition

   1. State = 0
      /\ Rcv(start)
      =|>
      State' = 1
      /\ Snd(B.PKb)

   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
```

```
  const secx    : protocol_id

  init   State := 0

  accept State = 1

  transition

   1. State = 0
      /\ Rcv(B.PKb')
      =|>
      State' := 1
      /\ X'   := new()
      /\ Snd({X'}_PKb'.{M.SCm}_X')
      /\ witness(M,B,x,X')
      /\ secret(X',secx,{B,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

 intruder_knowledge = {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 secx % addresses G12

  % Authentication: base station authenticates mobile
  %MSR_Base weakly authenticates MSR_Mobile on x
  weak_authentication_on x % addresses G1, G2

end goal
```

```
environment()
```

34.2 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 `secx`
- weak authentication on `x`

Problem Classification: G1, G2, G12

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,
        X       : symmetric_key,
        Nb      : text,
        Package : message

  const  x : protocol_id

  init   State := 0

  accept State = 2

  transition
```

```
   1. State = 0
      /\ Rcv(start)
      =|>
      State' := 1
      /\ Nb' := new()
      /\ Snd(B.Nb'.PKb.{B.PKb}_inv(PKs))

   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,
        Nb     : text,
        Cert   : message

  const secx   : protocol_id

  init   State := 0

  accept State = 1

  transition

   1. State = 0
      /\ Rcv(B.Nb'.PKb'.Cert')
```

```
      /\ Cert' = {B.PKb'}_inv(PKs)
      =|>
      State'=1
      /\ X' := new()
      /\ Snd({X'}_PKb'.{Nb'.M.SCm}_X')
      /\ secret(X',secx,{B,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)

  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

  intruder_knowledge = {b,m,scm2,scm3,i,ki,ks,inv(ki),
                         m,{i.ki}_inv(ks)
                        }

  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 secx % addresses G12

  % Authentication: base station authenticates mobile
  %IMSR_Base weakly authenticates IMSR_Mobile on x
  weak_authentication_on x % addresses G1, G2

end goal
```

```
environment()
```

35 SHARE

SHARE enables two principals to obtain a shared key, assuming that initially each knows the public key of the other.

Protocol Purpose

Key establishment protocol

Definition Reference

Martin Abadi, Two Facets of Authentication
Technical Report, Digital Systems Research Centre,
March 18, 1998

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: {Na}_Kb
2. B -> A: {Nb}_Ka
3. A -> B: {zero,Msg}_(Na,Nb)
4. B -> A: {one,Msg}_(Na,Nb)
```

Problems Considered: 3

- secrecy of `nanb`
- weak authentication on `k1`
- weak authentication on `k2`

Attacks Found:

The responder B believes to talk with the initiator A, while it is talking to the intruder. The attack trace looks like:

```
i       -> (a,6)   :  start
(a,6)   -> i       :  {na(a,6)}ki
i       -> (b,4)   :  {na(a,6)}kb
(b,4)   -> i       :  {nb(b,4)}ka
i       -> (a,6)   :  {nb(b,4)}ka
(a,6)   -> i       :  {zero,msg(a,6)}(na(a,6),nb(b,4))
i       -> (b,4)   :  {zero,msg(a,6)}(na(a,6),nb(b,4))
(b,4)   -> i       :  {one,msg(a,6)}(na(a,6),nb(b,4))
```

Further Notes

Such a protocol exploits the compound types feature by simply imposing that the variable `K` can be only instantiated with a pair of nonces.

HLPSL Specification

```
role share_Init ( A, B     : agent,
                  Ka, Kb   : public_key,
                  Snd, Rcv : channel(dy)) played_by A def=

  local  State   : nat,
         Na, Msg : text,
         Nb      : text,
         K       : text.text

  init   State := 0
  accept State = 3

  transition

   1. State  = 0 /\ Rcv(start) =|>
      State':= 1 /\ Na':= new()
                 /\ Snd({Na'}_Kb)

   2. State  = 1 /\ Rcv({Nb'}_Ka) =|>
      State':= 2 /\ Msg':= new()
```

```
                 /\ Snd({zero.Msg'}_(Na.Nb'))
                 /\ K':= Na.Nb'
                 /\ secret(Na.Nb',nanb,{A,B})
                 /\ witness(A,B,k2,Na.Nb')

   3. State  = 2 /\ Rcv({one.Msg}_K) =|>
      State':= 3 /\ wrequest(A,B,k1,K)

end role
```

```
role share_Resp (B, A     : agent,
                 Kb, Ka   : public_key,
                 Snd, Rcv : channel (dy)) played_by B def=

  local State    : nat,
        Nb       : text,
        Msg, Na  : text,
        K        : text.text

  init   State := 0
  accept State = 2

  transition

   1. State  = 0 /\ Rcv({Na'}_Kb) =|>
      State':= 1 /\ Nb':= new()
                 /\ Snd({Nb'}_Ka)
                 /\ K':= Na'.Nb'
                 /\ witness(B,A,k1,Na'.Nb')
                 /\ secret(Na'.Nb',nanb,{A,B})

   2. State  = 1 /\ Rcv({zero.Msg'}_K) =|>
      State':= 2 /\ Snd({one.Msg'}_K)
                 /\ wrequest(B,A,k2,K)

end role
```

```
role session(A, B          : agent,
             Ka, Kb        : public_key) def=

  local  SA, RA, SB, RB : channel (dy)

  composition
    share_Init(A,B,Ka,Kb,SA,RA) /\
    share_Resp(B,A,Kb,Ka,SB,RB)

end role
```

```
role environment() def=

 const zero, one    : text,
       a, b, i      : agent,
       ka, kb, ki   : public_key,
       k1, k2, nanb : protocol_id

 intruder_knowledge = {a,b,ka,kb,ki,i,inv(ki),zero,one}

 composition

        session(a,b,ka,kb)
    /\  session(a,i,ka,ki)

end role
```

```
goal

    secrecy_of nanb

    weak_authentication_on k1

    weak_authentication_on k2

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

[Ada96] C. Adams. RFC 2025: The Simple Public-Key GSS-API Mechanism (SPKM), October 1996. Status: Proposed Standard.

[ASW98] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[AVI03] AVISPA. Deliverable 6.1: List of selected problems. Available at `http://www.avispa-project.org`, 2003.

[BLMTM04] Julien Bournelle, Maryline Laurent-Maknavicius, Hannes Tschofenig, and Yacine El Mghazli. Handover-aware access control mechanism: Ctp for pana. In *ECUMN*, pages 430–439, 2004.

[BM98] Colin Boyd and Anish Mathuria. Key establishment protocols for secure mobile communications: A selective survey. *Lecture Notes in Computer Science*, 1438:344ff, 1998.

[BMP01] Giampaolo Bella, Fabio Massacci, and Lawrence C. Paulson. Verifying the SET Purchase Protocols. Technical Report 524, University of Cambridge, November 2001. URL: `http://www.cl.cam.ac.uk/Research/Reports/TR524-lcp-purchase.pdf`.

[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.

[CMM+04] J. Cuellar, J. Morris, D. Mulligan, J. Peterson, and J.Polk. RFC 3693: Geopriv requirements, 2004. `http://www.faqs.org/rfcs/rfc3693.html`.

[DA99] T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0, January 1999. Status: Proposed Standard.

[dB03] Sven Van den Bosch. NSLP for Quality-of-Service signaling, October 2003. Work in Progress.

[Eis00] M. Eisler. RFC 2847: LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM, June 2000. Status: Proposed Standard.

[HDM04] Paul Hankes Drielsma and Sebastian Mödersheim. The ASW protocol revisited: A unified view. In *Proceedings of the IJCAR04 Workshop ARSPA*, 2004. To appear in ENTCS, available at `http://www.avispa-project.org`.

[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.

[MNJH05] Robert Moskowitz, Pekka Nikander, Petri Jokela, and T. Henderson. Host Identity Protocol, June 2005. Work in Progress.

[MV77] Mastercard and VISA. SET Secure Electronic Transaction Specification, May 1977.

[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.

[PSC+04] Adrian Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe. Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction, September 2004. `http://www.ietf.org/rfc/rfc4082.txt`, Informational.

[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.

[VGEW00] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington. RFC 2845: Secret Key Transaction Authentication for DNS (TSIG), May 2000. Status: Proposed Standard.

[Wu00] T. Wu. RFC 2945: The SRP Authentication and Key Exchange System, September 2000. Status: Proposed Standard.

[YKS+05] Tatu Ylonen, Tero Kivinen, Markku Juhani Saarinen, Timo Rinne, and Sami Lehtinen. SSH Transport Layer Protocol, March 2005. Work in Progress.

[Zor00] G. Zorn. RFC 2759: Microsoft PPP CHAP Extensions, Version 2, January 2000. Status: Informational.