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

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

    %secrecy_of K
    secrecy_of sec_i_K, sec_r_K
```

```
    %SRP_Init authenticates SRP_Resp on k
    authentication_on k2
    %SRP_Resp authenticates SRP_Init on k
    authentication_on k1

end goal
```

```
environment()
```

References

[Wu00] T. Wu. RFC 2945: The SRP Authentication and Key Exchange System, September 2000. Status: Proposed Standard.