DECAP-Distributed Extensible Cloud Authentication Protocol

Andrea Huszti and Norbert Oláh
University of Debrecen
Nijmegen, Netherlands
Contents

1 Security requirements

2 The proposed protocol

3 Security analysis
Entity authentication methods

Scientific literature:
One-factor authentication solutions:
- 2000 M.S. Hwang, L.H. Li: smart card based, ElGamal encryption, impersonation attack
- 2002 Chien, Jan and Tsien: password based, several attacks

Two-factor authentication solutions:
- 2011 Amlan Jyoti Choudhury, Pardeep Kumar and Mangal Sain: two-factor authentication, smart card+password, Out of Band channel (OOB), impersonation attack
- 2014 Nan Chen and Rui Jiang: correcting the impersonation attack, no OOB
Entity authentication methods

In practice:
OpenStack is one of the most popular cloud computing software.
- User+password
- Lightweight Directory Access Protocol (LDAP)
- Kerberos
Centralized structure of authentication

One server authentication vulnerability:

- One target
- Lower attack cost
- Centralized responsibility
- Equipment is cheaper.
Distributed authentication

- Need to attack multiple servers simultaneously
- Increasing the attack cost
- Shared responsibility
- Equipment is more expensive
Security requirements of cloud computing

- Entity authentication
  Lack of strong authentication can lead to unauthorized access to users account on a cloud.

- Data integrity
  Data can be modified only by authorized parties.

- Secrecy, privacy
  Increased number of parties, devices and applications are involved, confidentiality of data should be protected against illegal access.

- Access control
  The data owner needs to make a flexible and scalable access control policy, so that only the authorized users can access.
Design’s goals

Distributed Extensible Cloud Authentication Protocol

- Expand the participants of the server park
- Expansion algorithm for the Merkle tree
- Scalability
- Shared responsibility
- Two-factor authentication
  static password + one-time-password
- MAC key exchange
  providing data origin integrity
- Improving efficiency
Security requirements

Merkle-tree

Y_r

Y_{d1}

Y_1

Y_{d2}

Y_2

Y_3

Y_4

T_{11}

K_{11} = g^{r_{1y_1}}

T_{12}

K_{12} = g^{r_{2y_2}}

T_{21}

K_{21} = g^{r_{3y_3}}

T_{22}

K_{22} = g^{r_{4y_4}}

T_{31}

K_{31} = g^{r_{5y_5}}

T_{32}

K_{32} = g^{r_{6y_6}}

T_{41}

K_{41} = g^{r_{7y_7}}

T_{42}

K_{42} = g^{r_{8y_8}}

Server1

Server2

Server3

Server4
The proposed protocol

Registration

U

$C_i$

AS

ID, PW, X salt, i index

$(y_i, z_i)$ secret key

$(r_i, s_i)$ secret

$(g^{y_i}, g^{z_i})$ public key

$(g^{r_i}, g^{s_i})$

$K_{i_1} = g^{r_i y_i}, K_{i_2} = g^{s_i z_i}$

$T_i = (T_{i_1}, T_{i_2}) = (H(K_{i_1}), H(K_{i_2}))$

Building the Merkle tree

$ID.(g^{r_i}, g^{s_i}).H(PW||X||i)$

$K_i = (g^{r_i y_i}, g^{s_i z_i})$

$Y_i = H(H(K_{i_1})||H(K_{i_2}))$

$Y_i, ID$ $\rightarrow$ $Y_r$, building the Merkle tree

$Y_r, ID$ $\rightarrow$ $<ID, K_i, H(PW||X||i), Y_r>$
Authentication

U

\( \nu \in \{1, \ldots, i\} \) random gen.

\( ID, T_{\nu_1}, Y_{d_1}, Y_{d_2}, H(Y_r || H(PW || X || \nu)) \)

\( H(SK), m \)

\( \leftarrow \)

C_i

\( \nu \in \{1, \ldots, i\} \) random gen.

Checking:

\( T_{\nu_1} \)

\( Y_{d_1}, Y_{d_2} \rightarrow Y_r \) ver.

\( H(Y_r || H(PW || X || \nu)) \) ver.

SK = \( H(Y_r || T_{\nu_2}) \), m rand.

ID, SK, m storage

H(SK) verification

\( ID, MAC(m, SK) \)

\( \rightarrow \)

MAC verification
## Security requirements

<table>
<thead>
<tr>
<th>Security analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronization</td>
</tr>
</tbody>
</table>

### The proposed protocol

<table>
<thead>
<tr>
<th>U</th>
<th>C_v</th>
<th>AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K'_v = K_v * g = g^{r_v y_v + 1}$</td>
<td>$K'_v = K_v * g$</td>
<td></td>
</tr>
<tr>
<td>$K'_v = K_v * g = g^{s_v z_v + 1}$</td>
<td>$K'_v = K_v * g$</td>
<td></td>
</tr>
<tr>
<td>$T'_v = H(K'_v)$</td>
<td>$T'_v = H(K'_v)$</td>
<td></td>
</tr>
<tr>
<td>$T'_v = H(K'_v)$</td>
<td>$T'_v = H(K'_v)$</td>
<td></td>
</tr>
<tr>
<td>$T_v$ path update</td>
<td>$Y_v = H(H(T'_v)</td>
<td></td>
</tr>
<tr>
<td>$&lt;ID, K_v, H(PW</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Security analysis

Requirements:
- Authentication of the User
- Authentication of the Cloud server
- Secrecy of the key
- Key freshness
- Both parties should be assured that the other party knows the new key.

Model:
- Dolev-Yao adversary model: read, modify, delete, and inject messages (record communications, store values, synthesize messages etc.)
Security analysis

In case of an outsider adversary:

- Applied pi-calculus
- Proverif 1.93: Automatic verifier for cryptographic protocols.
- Cryptographic protocols are concurrent programs which interact using public channels.
- An arbitrary number of protocol executions
Main processes

- The main process controls activities between the User and the Server subprocesses
- We execute the User and the Server Processes in parallel infinitely many times

process
new y1: exponent; new y2: exponent; new y3: exponent; new y4: exponent;
new id: bitstring; new idS: bitstring; new x: passx;
let S1 =exp(g, y1) in let S2 =exp(g, y2) in let S3 =exp(g, y3) in let S4 =exp(g, y4) in
out(c,(S1,S2,S3,S4)); (*Public keys*)
new sskU:sskey;
new eskS:skey;
let spkU=spk(sskU) in out(c,spkU);
let epkS=pk(eskS) in out(c,epkS);
((!User(id,idS,S1,S2,S3,S4,sskU,spkU,epkS, x)) |
(!Server(id,idS,y1,y2,y3,y4,eskS,epkS,spkU)))
User process

(*User process*)
let User(id:bitstring, idS:bitstring,

(*Authentication 1*)
let M1=H2((Y7,H_PW)) in
let USK=H2((Y7,TT2)) in
event User_auth_start(USK);
out(c,(id,TT1,Y5,Y6,M1));
in(c,(M2:bitstring,serverm:bitstring));
let E2=H2((USK)) in
if E2=M2 then
event Server_auth_end(USK);
let M3=mac(USK,serverm) in
out(c, (id,M3));

(*Synchronization*)
event User_sync_start;
(*Merkle-tree update*)
let T1_2=exp(T1,S_synch) in
let TT1_2= H(T1_2) in
let Y1_2= H2(TT1_2) in
let Y5_2= H2((Y1_2,Y2)) in
let Y7_2= H2((Y5_2,Y6)) in

(*Authentication 2*)
let M1_2=H2((Y7_2,H_PW)) in
let USK_2=H2((Y7_2,TT4)) in
event User_auth2_start(USK_2);
out(c,(id,TT3,Y5_2,Y6,M1_2));
in(c,(M2_2:bitstring,serverm_2:bitstring));
let CheckSK=H2((USK_2)) in
if CheckSK=M2_2 then
event Server_auth2_end(USK_2);
let M3_2=mac(USK_2,serverm_2) in
out(c, (id,M3_2)).
ProVerif events

Events

**First Authentication**

**Event User Auth Start**

\[ ID, T_{v_1}, Y_{d_1}, Y_{d_2}, H(Y_r||H(PW||X||v)) \]

**Server**

**Event Server Auth Start**

**Event Server Auth end**

**Update Merkle-tree**

**Second Authentication**

**Event User Auth2 Start**

\[ ID, T_{v_1}, Y_{d_1}^*, Y_{d_2}, H(Y_r^*||H(PW||X||v)) \]

**Event Server Auth2 Start**

\[ H(SK^*), m^* \]

**Event User Auth2 End**

**Update Merkle-tree**

\[ ID, \text{MAC}(m^*, SK^*) \]

**Event User Auth2 End**
Security analysis

- Secrecy of the key and the password
  \textit{query attacker:SK.}
  \textit{query attacker:pw.}

- Authentication of the User and the Cloud server
  \textit{query sk:bitstring; inj-event(Server\_auth\_end(sk)) \Rightarrow inj-event(Server\_auth\_start(sk)).}
  \textit{query sk:bitstring; inj-event(User\_auth\_end(sk)) \Rightarrow inj-event(User\_auth\_start(sk)).}
  \textit{query sk:bitstring; inj-event(Server\_auth2\_end(sk)) \Rightarrow inj-event(Server\_auth2\_start(sk)).}
  \textit{query sk:bitstring; inj-event(User\_auth2\_end(sk)) \Rightarrow inj-event(User\_auth2\_start(sk)).}
Key freshness
- Yr changes after authentication - secret
- $T_{v1}$ changes - secret
- v random

Both parties should be assured that the other party knows the new key.
If the following conditions occur, both parties know the new key:

let $E_2 = H_2((USK))$ in if $E_2 = M_2$ then let $M_3 = \text{mac}(USK, serverm)$
let Checkmac = mac(SK, serverm) in if $M_3 = \text{Checkmac}$ then event second(SK).
## Efficiency analysis

<table>
<thead>
<tr>
<th></th>
<th>Hash</th>
<th>Exp</th>
<th>Mult</th>
<th>Inv</th>
<th>Encryption/Decryption</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Choudhury et al (2011)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>10</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>8</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td>18</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td><strong>Nan and Rui (2014)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>4+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Server</td>
<td>4+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td>8+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td><strong>Our</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>3+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>3+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td>6+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
This work was supported by the construction EFOP-3.6.3-VEKOP-16. The project has been supported by the European Union, co-financed by the European Social Fund.
Thank you for your attention!