

Basics of SSH authentication using public/private key authentication

This scenario will cover public/private key authentication between a Windows client and a Linux server.

Create User Account on Linux System

```
useradd -d /home/example.org example.org  
passwd example.org
```

```
chown -R 506.506 /home/example.org/  
cd /home/example.org/
```

```
mkdir data  
mkdir .ssh
```

```
chown -R 506.506 data .ssh  
chmod 700 data .ssh
```

Run PuTTY Key Generator on Windows Client

```
Key: SSH2 DSA KEY  
Bits: 1024
```

```
Generate Key Pair  
Key Comment: administrator@example.org
```

Paste public key into OpenSSH ~/.ssh/authorized_keys file

```
Save Public Key as d:\.ssh\public.ppk  
Save Private Key as d:\.ssh\private.ppk
```

Example of SSH public/private key authentication in Windows Script (BACKUP.BAT)

```
for /f "tokens=2,3,4 delims=/ " %i In ('Date /t') Do Set date=%k%i%j  
set date=%date:~2,6%
```

```
del d:\bf_%date%.tar.gz  
tar.exe -czf d:\bf_%date%.tar.gz e:\var\www\www.example.org\  
pscp.exe -i d:\.ssh\private.ppk d:\bf_%date%.tar.gz example.org@ftp.example.org:data/
```

Basics of SSH authentication using public/private key authentication

This scenario will cover public/private key authentication between a Linux client and a Linux server.

Create Storage User Account

```
-----
server:      useradd -d /home/example.com/ example.com
server:      passwd example.com
server:      # append "exit" line to end of new user profile
server:      vi ~/.bash_profile # for example.com
server:      mkdir ~/data/      # for example.com
server:      chmod 700 ~/data/    # for example.com
server:      chown 0.0 ~/.bash*  # for example.com
```

Configure PublicKey Exchange

```
-----
client:      su root
client:      cd
client:      mkdir .ssh
client:      chmod 700 .ssh
client:      cd .ssh
client:      ssh-keygen -d -C root@example.com
client:      # save as id_dsa w/o passphrase

server:      useradd -d /home/example.com example.com
server:      passwd example.com
server:      su example.com
server:      cd
server:      mkdir data
server:      mkdir .ssh
server:      chmod 700 .ssh
server:      cd .ssh

client:      scp ~/.ssh/id_dsa.pub example.com@ftp.example.net:~/.ssh/
client:      # this SHOULD require password on remote server

server:      cat id_dsa.pub >> authorized_keys
server:      chmod 644 authorized_keys
server:      rm id_dsa.pub

client:      echo "test123" > testfile
client:      scp testfile example.com@ftp.example.net:data/
client:      # this SHOULD NOT require password on remote server

server:      cd
server:      cat data/testfile
server:      rm data/testfile

client:      rm testfile
```

Basics of creating your own SSL certificates

Creating a test certificate

The following instructions are from <http://www.apache-ssl.org/#FAQ>.

```
openssl req -new -out server.csr
```

This creates a certificate signing request and a private key. When asked for "Common Name (eg, your websites domain name)", give the exact domain name of your web server (e.g. www.my-server.dom). The certificate belongs to this server name and browsers complain if the name doesn't match.

```
openssl rsa -in privkey.pem -out server.key
```

This removes the passphrase from the private key. You MUST understand what this means; server.key should be only readable by the apache server and the administrator. You should delete the .rnd file because it contains the entropy information for creating the key and could be used for cryptographic attacks against your private key.

```
openssl x509 -in server.csr -out server.crt -req -signkey server.key -days 365
```

This creates a self-signed certificate that you can use until you get a "real" one from a certificate authority. (Which is optional; if you know your users, you can tell them to install the certificate into their browsers.) Note that this certificate expires after one year, you can increase -days 365 if you don't want this.

```
openssl x509 -in server.crt -out server.der.crt -outform DER
```

If you have users with MS Internet Explorer 4.0+ and want them to be able to install the certificate into their certificate storage (by downloading and opening it), you need to create a DER-encoded version of the certificate:

Create an /etc/httpd/conf/ssl directory and move server.key and server.crt into it. For Linux create two directories: ssl.key and ssl.crt. Move server.crt into ssl.crt and move server.key into ssl.key.

Configuring Apache and mod_ssl

```
<VirtualHost www.my-server.dom:443>
    SSLEngine On
    SSLCertificateFile conf/ssl/server.cert
    SSLCertificateKeyFile conf/ssl/server.key
</VirtualHost>
```

Don't forget to call apache with -D SSL if the IfDefine directive is active in the config file! In other words, either start Apache from the command line with -D SSL or comment out the IfDefine start/end tags in ssl.conf.

NOTE: When using SSL with multiple Virtual Hosts, you must use an ip-based configuration. This is because SSL requires you to configure a specific port (443), whereas name-based specifies all ports (*). You might the following error if you try to mix name-based virtual hosts with SSL.

```
[error] VirtualHost _default_:443 -- mixing * ports and non-* ports with a
NameVirtualHost address is not supported, proceeding with undefined results
```

SSL HOWTO SOURCE - <http://raibledesigns.com/tomcat/ssl-howto.html>