**Project 3: PGP and User Authentication**

[1 Introduction and General Information 1](#_Toc463451735)

[2 PGP 1](#_Toc463451736)

[2.1 Install GnuPG（1 point） 1](#_Toc463451737)

[2.2 Generate a PGP Key Pair 2](#_Toc463451738)

[2.3 Sign and Verify Documents 4](#_Toc463451739)

[2.4 En/DecryptDocuments 4](#_Toc463451740)

[3 Password Cracking with John the Ripper 5](#_Toc463451741)

[3.1 Install John the Ripper 5](#_Toc463451742)

[3.2 Create additional user accounts on FC VM 6](#_Toc463451743)

[3.3 Crack UNIX-like system password files 6](#_Toc463451744)

[3.4 Crack MD5 hash values 7](#_Toc463451745)

[4 Authentication via Public Key 7](#_Toc463451746)

[4.1 Disable the SELinux（2 points） 7](#_Toc463451747)

[4.2 CreateRSA key pairs 8](#_Toc463451748)

[4.3 Place the public key on server 8](#_Toc463451749)

[4.4 Test authentication 9](#_Toc463451750)

[4.5 Use Digital Signature Algorithm (DSA)（10 points） 10](#_Toc463451751)

# Introduction and General Information

**Goal**: This lab will introduce you to security issues involving password cracking. In addition, we will learn how to setup remote authentication without exchanging passwords.

# PGP

## Install GnuPG（1 point）

As the root user, type the following command on your Linux VM (say Fedora server):

*yum –y install gnupg*

You can also use the new version:

*yum –y install gnupg2*

## Generate a PGP Key Pair

Ref: <http://zacharyvoase.com/2009/08/20/openpgp/>.

Type the following key pair to generate a PGP key pair: （3 points）

*gpg2 --gen-key*

You’ll see:

Please select what kind of key you want:

(1) RSA and RSA (default)

(2) DSA and Elgamal

(3) DSA (sign only)

(4) RSA (sign only)

Your selection?

Hit enter, since the defaults work fine. Then:

RSA keys may be between 1024 and 4096 bits long.

What keysize do you want? (2048)

Whilst longer keys are more secure, they can also take a long time to generate, and \*cryption will also be slower. This might not matter if you and all your contacts are running relatively recent and powerful hardware, but once OpenPGP support for embedded and mobile devices becomes more available, it will present a serious issue. For this reason, you may want to go for the default. After deciding, you’ll be presented with this screen:

Requested keysize is 4096 bits

Please specify how long the key should be valid.

0 = key does not expire

<n> = key expires in n days

<n>w = key expires in n weeks

<n>m = key expires in n months

<n>y = key expires in n years

Key is valid for? (0)

Hit enter again. This should be fine: if somehow your key becomes compromised you’ll be able to revoke it using the revocation certificate (more on that later). It’ll also ask you for confirmation; just hit y then enter again. The generator will then ask you to answer a series of questions:

GnuPG needs to construct a user ID to identify your key.

Real name: <Type your full name, hit enter>

Email address: <Type your email address, hit enter>

Comment: <Optional: type a comment or your homepage URL, hit enter>

You selected this USER-ID:

"Real Name (http://example.com) <name@example.com>"

After that it will ask you to confirm:

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?

Just type O (that’s the capital letter, not the numeral) and then enter. At this point a dialog box will appear asking for a passphrase. This is necessary to make sure that even if someone does gain access to your secret key, they might not be able to use it. The normal rules for strong passwords apply.

At this point, you’ll see something like this:

We need to generate a lot of random bytes. It is a good idea to perform

some other action (type on the keyboard, move the mouse, utilize the

disks) during the prime generation; this gives the random number

generator a better chance to gain enough entropy.

Do what it says. It’s generating your keys, but needs a lot of random numbers to do so.

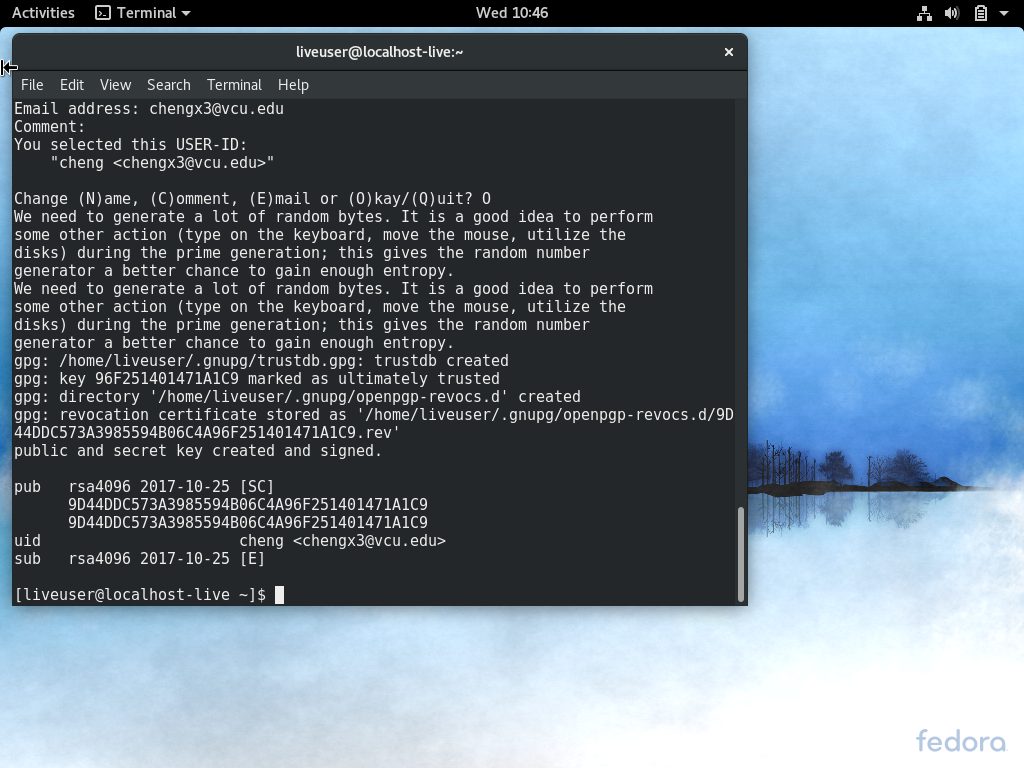
After this is done, you should see something along the lines of:

A screenshot of a computer

Description automatically generated



The 8-character Key ID on the line beginning with pub after 4096R is the ID of your public master key (the one used for identity and signing), the one on the line beginning with sub is that of your public subkey (the one used for \*cryption), and your primary user ID is on the line beginning with uid. You can change the uid via command: *gpg2 --edit-key KEYID*.

If you use the new version, you will see:



You also can use the command *gpg2 --list-keys* to view the PGP keys.

### Which folder contains the newly generated key pair? （2 points）

## Sign and Verify Documents

After all that setup, we can start signing and verifying documents. Signing a document to prove its authenticity is very simple with GPG2. You only need to provide your private key’s passphrase. Assume you have a file called document.txt. To sign a document, just run: （2 points）

gpg2 --sign document.txt

You can also pass in --armor, if you need plaintext output. This will create a file called document.txt.gpg or document.txt.asc, depending on whether or not you armor the output. To verify a signature, just run: （2 points）

gpg2 --verify document.txt.gpg

For this to work, document.txt has to be in the same directory as its signature.

If the signed document is from another person, you also must have imported the public key of the person who signed it (since the public key is used in verification).

## En/DecryptDocuments

Next, we could encrypt and decrypt documents. First, let’s extract the public key. You can use the following command to do so:

*gpg2 --export -a KEYID > pgp\_pub\_key.asc*（2 points）

Next, you need to transfer the extracted public key to another Linux VM (say Fedora Client) using scp. (To use scp, you must install openssh and start sshd.service.)

### What is option to extract the private key? （2 points）

Take a look at the manual for the gpg2 command (by typing man gpg2 at the console) for more information.

On your Client, you can import the public key using the following command:

*gpg2 --import pgp\_pub\_key.asc*（2 points）

If you have a file called document.txt on your client that you’d like to encrypt, just run: （1 point）

gpg2 --encrypt document.txt

You’ll be prompted to enter the user IDs to whom you’re sending the encrypted document: （2 point）

Enter the user ID. End with an empty line: <Type User ID here>

If you have only one key in your keyring for a person called “John Smith”, you can just type John and it’ll get the correct key for you. At each stage the command will print all the recipients you’ve specified so far. To finish, just hit enter on a prompt without typing anything.

GPG2 will encrypt the document and save the encrypted ciphertext as document.txt.gpg. This file will, however, be in a binary format. You can transfer the encrypted file to the server, where the private key is located.

Go back to FC server, to decrypt the transferred document, you can just run: （2 points）

gpg2 --decrypt document.txt.gpg

This will prompt you for your private key’s passphrase, and save the decrypted document indocument.txt.

### Did the decrypted document match the original document on FC client? （2 points）

### How to register your PGP keys? （2 points）

### What is option to delete the PGP keys on your local machine? （2 points）

# Password Cracking with John the Ripper

## Install John the Ripper

On your Fedora based VM, you will be installing and using the software “John the Ripper”, which can be found at <http://www.openwall.com/john/>, to crack the passwords stored on a Linux system. This is a great tool to use within your network to check the strength and uniqueness of your organization’s passwords.

\*\*\*NOTE: As always, remember that you must have permission (written) from the organization to conduct this password analysis.

Installation: as the root user, type:

# yum -y install john（1 point）

## Create additional user accounts on FC VM

Create the following user accounts on your Fedora virtual machine using command line. The command to add a user is *useradd*. For example, you can useradd User1 to add User1.

The command to set a user’s password is *passwd*. For example, you can passwd User1 to set its password. （1 point）

[Username] [Password]

User1 Hello

User2 123

User3 Flower

User4 Dragon

User5 Hellodragon

User6 123Hello

User7 i!h@d@M$!wbr

After creating users and their passwords, please check which file stores the user names and passwords.

### Where are the usernames and encrypted passwords stored? （2 points）

Tip: There are numbers in preceding the hash values. These numbers have different meanings. For example:   
$1$ == md5  
$5$ == sha256  
$6$ == sha512

### What are the differences between */etc/passwd* and*/etc/shadow*?Check the properties of these two files using the command *ls -l filename*. （2 points）

### Why are these two files setup this way? （2 points）

## Crack UNIX-like system password files

First, locate the password list of John the Ripper. Type the following commands:

updatedb

locate password.lst

Run John the Ripper on the shadow password file you just created.（2 points）

john -wordlist=PATH\_TO /password.lst /etc/shadow

To display the cracked password（2 points）

john --show /etc/shadow

Some passwords should be cracked quickly and others taking much longer. The more complicated passwords might take hours or days to crack. Pressing the spacebar will show the passwords being tried. Hit it several times to see the way the password combinations are tried. Stop the program after some time using Ctrl-C.

### Write down how many passwords has been cracked and what passwords they were. （2 points）

To look at the password file that comes with John, type vi /usr/share/john/password.lst

### Do you see any passwords that were just created on the FC machine? （2 points）

<http://www.openwall.com/john/doc/EXAMPLES.shtml>

## Crack MD5 hash values

We can tell the system to create users’ passwords in MD5 format. On another Fedora VM, type the following commands. （2 point）

authconfig --test | grep hashing

authconfig --passalgo=md5 --update

authconfig --test | grep hashing

Repeat the steps in section 3.2 and 3.3. （6 points）

### 

# Authentication via Public Key

Passwords are known to be a weak authentication method because an unwanted person or application can hijack the password. It is always risky if you need to store and exchange passwords. There are secure protocols that allow remote logins that bypasses the conventional password authentication.

SSH allows you to prove your identity to a remote host by using encrypted public keys instead of a password. It is more secure because the encrypted keys are never transmitted over the network. In this lab, we will use public key scheme to setup SSH authentication to avoid passwords.

## Disable the SELinux（2 points）

If you have done this previously, please go to the next section. As the root user, type:

vi /etc/selinux/config

Change enforcing to disabled.

Reboot your system.

## Create RSA key pairs

A RSA key pair must be generated on the client system. The public portion of this key pair will reside on the servers being connected to, while the private portion needs to remain on a secure local area of the client system. The key generation can be done with the ssh-keygen utility.

### Create a 2048 bit RSA key on your client VM as a regular user (e.g., student):（5 points）

ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key (/home/user/.ssh/id\_rsa):

Created directory '/home/user/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Note that if you want to use your key for password-less authentication you should not supply a passphrase. If you do the key is then encrypted with a password and you will need to supply that specific password in order for the authentication to work.

You will see:

Your identification has been saved in /home/user/.ssh/id\_rsa.

Your public key has been saved in /home/user/.ssh/id\_rsa.pub.

The key fingerprint is:

c7:2a:ac:0f:24:ad:ee:93:fe:e2:45:04:9c:c3:b6:7e user@client\_hostname



You can view the public key by:（2 points）

cat /home/user/.ssh/id\_rsa.pub | more

You can view the prive key by:（2 points）

cat /home/user/.ssh/id\_rsa | more

## Place the public key on server

Transfer the /home/user/.ssh/id\_rsa.pub file to the server you want to connect to. You can use scp to copy.（2 points）  
scp /home/user/.ssh/id\_rsa.pub user@server:

Login to the server as the user you want to use the public key for.

Concatenate the contents of id\_rsa.pub to /home/user/.ssh/authorized\_keys（3 points）

mkdir ~/.ssh  
cat /home/user/id\_rsa.pub >> /home/user/.ssh/authorized\_keys

chmod 600 /home/user/.ssh/authorized\_keys

chmod 700 /home/user/.ssh/

Remove the public key file: （1 point）  
rm /home/user/id\_rsa.pub

Start the sshd service on the serevr:（1 point）

service sshd restart

## Test authentication

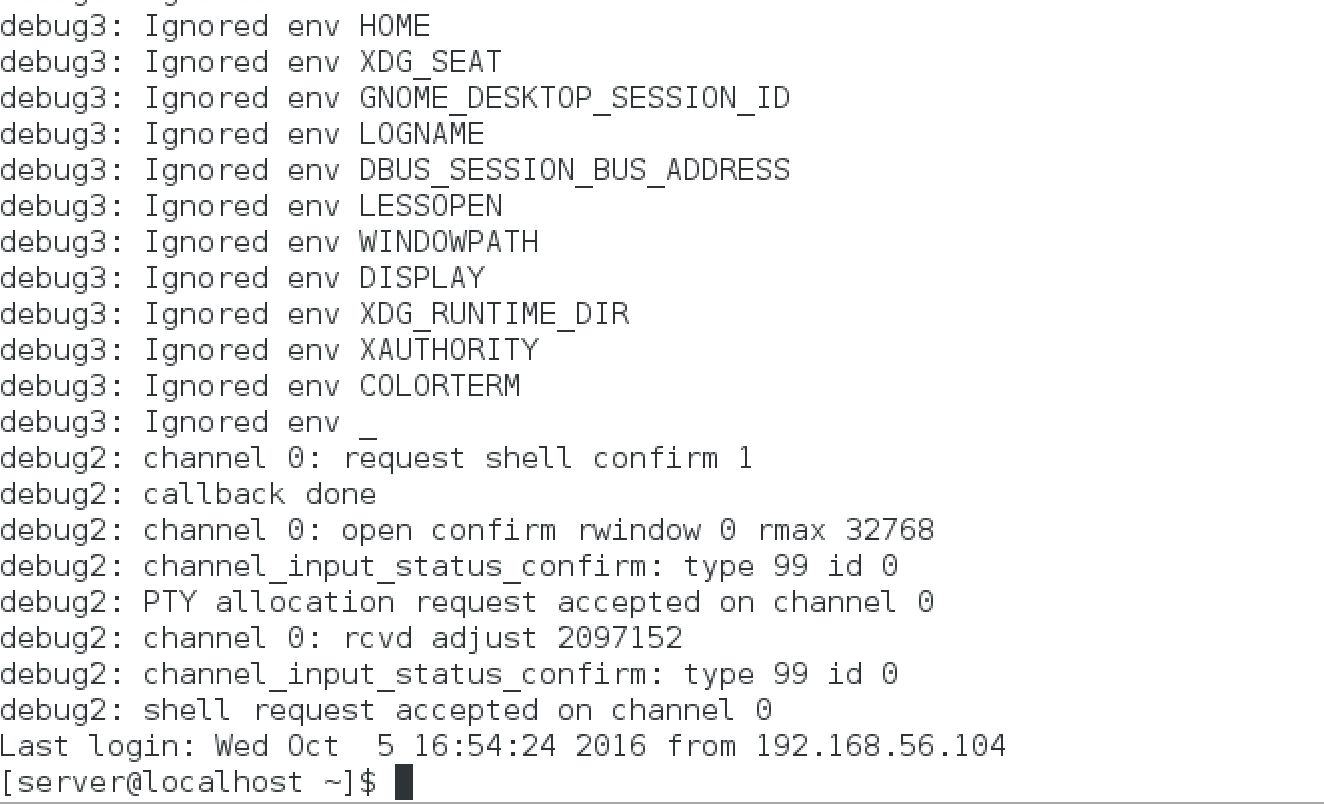
Login to the client where you created the RSA key pairs. Try to connect to the server where you copied the public key to using the ssh command.（2 points）

ssh user@server

If you are prompted for a password verifies that the steps have been followed correctly. Otherwise, congratulations! You have successfully setup SSH public key authentication.

You can see more information about what is happening by typing:（2 points）

ssh –vvv user@server



In addition, you can place the user’s public key into another user’s (e.g.,Instructor or root) .ssh directory. Test what will happen when you try to log in as another user. For example,（2 points）

ssh Instructor@server

### What are the pros and cons of this type of authentication approach?（2 points）

## Use Digital Signature Algorithm (DSA)（10 points）

In general, DSA is faster for key generation and signing, whereas RSA is quicker at verification. Yet, the speed difference in authentication is so small (such as ssh'ing into a system with a regular size password, or a few more if you use MD5) that you will not notice a difference.DSA performance is more important for SSL enabled web applications.

Repeat procedures listed 4.1 to 4.4. When typing the commands, replace the key word **rsa** with **dsa**.

