File encryption using OpenSSL and GnuPG (GPG/PGP)

Systems and Network Security

---

1 Notes

- Commands preceded with "$" imply that you should execute the command as a general user - not as root.
- Commands preceded with "#" imply that you should be working as root.
- Commands with more specific command lines (e.g. "RTR-GW>" or "mysql>") imply that you are executing commands on remote equipment, or within another program.

2 Exercises

We're going to use PGP to perform encryption. First we need to install the software, then generate a public/private key pair, as we learned about during lectures.

2.1 Install GnuPG (aka PGP/GPG)

```
$ sudo apt-get install gnupg
$ sudo apt-get install rng-tools
```

Answer 'y' if you are told the packages cannot be authenticated. Don't worry about this right now.

Once the software is installed, copy paste the following commands:

```
$ sudo sed -i -e 's|#HRNGDEVICE=/dev/hwrng|HRNGDEVICE=/dev/urandom|'
/etc/default/rng-tools
$ sudo service rng-tools start
```

... this starts a service that will help produce more random numbers. This is needed for the key generation in the next step.

2.2 Generate a public/private key pair

Note: For this lab, you will work as a group, since the sysadm user can only have one keyring in their home directories.

Run the command:

```
$ gpg --gen-key
```

You will see:

```
gpg: directory `/home/sysadm/.gnupg' created
gpg: new configuration file `/home/sysadm/.gnupg/gpg.conf' created
gpg: keyring `/home/sysadm/.gnupg/secring.gpg' created
gpg: keyring `/home/sysadm/.gnupg/pubring.gpg' created
```

... Followed by a menu

```
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?
```

Press '1' and return.

You will then be prompted to pick a key size:

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

Press return to accept the default of 2048

```
Requested keysize is 2048 bits
```

You will then have to decide if the key will expire in time, or remain active until explicitly revoked

```
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)
```

Press '0' and return.

```
Key does not expire at all
Is this correct? (y/N)
```

Answer 'y', then return.

```
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
```

Here, enter your name, email, and a comment about this key:

```
Real name: Bob Bobson       <-- use your name ...
Email address: bob@bob.com  <-- ... and email address here!
Comment: Work address
```

You will be asked to confirm:

```
You selected this USER-ID:
    "Bob Bobson (Work address) <bob@bob.com>"

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

Type 'O' then return.

You are then asked to enter a passphrase to protect your key:

```
You need a Passphrase to protect your secret key.

gpg: gpg-agent is not available in this session
Enter passphrase:
```

Pick a passphrase that is short enough to be typed without too much difficulty, but not too short that it can be guessed. The passphrase will not be shown.

You will be asked to enter the passphrase twice.

```
Repeat passphrase:
```

Once you've entered the passphrase twice, GPG will then proceed to create the keys, and you will see output similar to this (of course the key fingerprints and the name + email will be different)

```
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.
[...]
gpg: /home/sysadm/.gnupg/trustdb.gpg: trustdb created
gpg: key C9FBE546 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/C9FBE546 2013-07-30
      Key fingerprint = 913E E323 6BF2 CA8E 5270  A78E 2E33 1A5B C9FB E546
uid                  Bob Bobson (Work address) <bob@bob.com>
sub   2048R/3BE8FE75 2013-07-30
```

Ok, we know have a public and private key pair.

You can see the files containing these files like this:

```
$ cd
$ ls -l .gnupg
total 32
-rw------- 1 sysadm sysadm 9398 Jul 30 14:07 gpg.conf
-rw------- 1 sysadm sysadm 1198 Jul 30 14:08 pubring.gpg
-rw------- 1 sysadm sysadm 1198 Jul 30 14:08 pubring.gpg~
-rw------- 1 sysadm sysadm  600 Jul 30 14:08 random_seed
-rw------- 1 sysadm sysadm 2575 Jul 30 14:08 secring.gpg
-rw------- 1 sysadm sysadm 1280 Jul 30 14:08 trustdb.gpg
```

Notice the presence of the files "pubring.gpg" and "secring.gpg".

Those files contain, respectively:

- your public key, and, you will see later, the public keys of OTHER people in the class
- your private key, which should NEVER be disclosed

You can also use the following command to list the keys in your keyrings:

```
$ gpg --list-keys
```

This will show you a list of keys. You will notice that you have not only one public/private key pair, but also have a so-called "sub" keys.

```
/home/sysadm/.gnupg/secring.gpg
-------------------------------
sec   2048R/C9FBE546 2013-07-30
uid                  Bob Bobson (Work address) <bob@bob.com>
ssb   2048R/3BE8FE75 2013-07-30
```

One interesting feature of PGP is the ability to *sign* keys. This means that you can ask a third party you trust, and more importantly, who trusts you, to use their *private* key to "sign" *your public* key. This is a way for them to say "I believe this person really is who they say they are, and here's my proof".

- Why is it necessary to sign keys ?
- Can anyone create a key and pretend to be another person ?
- Can you think of a way to make sure that a given key really belongs to the person listed on the key ?
- What do you think are the benefits of signing keys ?

Sometimes, you will find that it is necessary to get rid of "old" keys, and make some new ones. But what if many people have signed your key ?

Not to worry! People who have signed your key actually sign your "Master key".

Master keys are used to sign sub keys. Which means, that you can replace those, and still benefit from the "trust" of those who have signed your master key..

2.3 Encrypt with GPG using public key

It's time to encrypt files with GPG.

If you have deleted the file my-secrets-myname.txt from earlier, and you weren't able to decrypt it from the encryption version, you will need to recreate a new file:

```
$ cd
$ cat > my-secrets-myname.txt <<EOF

My name is "My Name"

My credit card number is 1234-5678-9012-3456

The password for my phone is 42
```

```
EOF
```

Once that is done, let's encrypt the file:

```
gpg -e my-secrets-myname.txt
```

GPG is going to tell you you didn't include a recipient / user ID.

```
You did not specify a user ID. (you may use "-r")

Current recipients:

Enter the user ID.  End with an empty line:
```

Here, you can just write your own email (userid) you chose earlier, for example: bob@bob.com if that is your email:

```
Current recipients:
2048R/3BE8FE75 2013-07-30 "Bob Bobson (Work address) <bob@bob.com>"
```

It then asks you if there are other recipients. Just press RETURN to continue without adding more recipients.

Normally, GPG should finish quietly and leave you back at the shell.

Verify that you now have encrypted files present in your directory:

```
$ ls -l my-secrets-myname.txt*
-rw-rw-r-- 1 sysadm sysadm 102 Jul 30 12:45 my-secrets-myname.txt
-rw-rw-r-- 1 sysadm sysadm 441 Jul 30 14:30 my-secrets-myname.txt.gpg
```

... you may see other files from previous labs, but you should see a ".gpg" file.

Just like in the OpenSSL labs, try and view the contents of the file with more (or less). The file is encoded in binary. You will notice that is is somewhat larger then the file encrypted by OpenSSL.

Let's encrypt with an ASCII encoding - and this time we'll save time and specify the recipient directly with the '-r' flag:

```
$ gpg -a -e -r bob@bob.com my-secrets-myname.txt
```

By the way, do you notice anything ?

Hint: Did you have to specify the passphrase at any point to ENCRYPT ?

Check the contents of the directory again:

```
$ ls -l my-secrets-myname.txt*
-rw-rw-r-- 1 sysadm sysadm 102 Jul 30 12:45 my-secrets-myname.txt
-rw-rw-r-- 1 sysadm sysadm 441 Jul 30 14:30 my-secrets-myname.txt.gpg
-rw-rw-r-- 1 sysadm sysadm 694 Jul 30 14:40 my-secrets-myname.txt.asc
```

You should see an ".asc" file present.

Look at its contents!

Now, you can delete the original .txt file.

```
$ rm my-secrets-myname.txt
```

2.4 Decrypting files

To decrypt a file with GnuPG/PGP, all you have to do is type:

```
$ gpg my-secrets-myname.txt.asc
```

GnuPG/GPG automatically figures out who the file is encrypted for, and checks to see if you are in possession of the private key (you are), and you are prompted for your *passphrase*:

```
You need a passphrase to unlock the secret key for
user: "Bob Bobson (Work address) <bob@bob.com>"
2048-bit RSA key, ID 3BE8FE75, created 2013-07-30 (main key ID C9FBE546)

gpg: gpg-agent is not available in this session
Enter passphrase:
```

If the file original file still exists, then gpg will ask you before it overwrites it:

```
File `my-secrets-myname.txt' exists. Overwrite? (y/N) y
```

If you answer 'y', it will overwrite as indicated.

Look at the contents of the file 'my-secrets-myname.txt' and confirm that they are correctly decrypted!

In the next lab, we will learn to exchange keys and send encrypted files to each other in the class!

- Note: Note that GPG call also do symmetric encryption! You can do what you did in the previous lab, using GPG instead of OpenSSL. There is no real advantage to this - your public/private key pair isn't used: you will be prompted for a passphrase at encryption time, just like with OpenSSL.

Example:

```
$ gpg --output filename.enc -z 0 --symmetric filename.txt
```

Note that by default GPG always compresses the file - to disable encryption, you would use "-z 0" as specified in the example above (for speeding up encryption on very large files, or media like sound, video).