GnuPG: Creating a private key and a public key

Confirm that you can run “xclock” and run “hostname” to determine which server your currently on. (I found that I couldn’t get xclock running. It didn’t matter.)

You can also get more info from "https://www.youtube.com/watch?v=QhHkgCFc1xc"

```
cbarros@researchgrid$ xclock
cbarros@researchgrid$ gpg --gen-key
gpg (GnuPG) 2.0.14; Copyright (C) 2009 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

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?
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 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) 3y
Key expires at Fri 29 Mar 2019 11:51:51 AM EDT
Is this correct? (y/N) y

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

Real name: Chris Barros
Email address: cbarros@hbs.edu
Comment:
You selected this USER-ID:
    "Chris Barros <cbarros@hbs.edu>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

can't connect to `/export/home/itg/cbarros/.gnupg/S.gpg-agent': No such file or directory
gpg-agent[18556]: directory `/export/home/itg/cbarros/.gnupg/private-keys-v1.d' created
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.
```

This may take a long time to complete the creation of the key. Let read above. Please wait, the command line will return after 20 min.

```
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
gpg: next trustdb check due at 2019-03-29
pub   2048R/A41EF522 2016-03-29 [expires: 2019-03-29]
      Key fingerprint = B9F4 8159 7002 0171 D8ED  5704 1312 207F A41E F522
uid                  Chris Barros <cbarros@hbs.edu>
sub   2048R/3C211111 2016-03-29 [expires: 2019-03-29]
```

Confirm key

```
cbarros@researchgrid$ gpg --list-keys cbarros@hbs.edu
pub   2048R/A41EF522 2016-03-29 [expires: 2019-03-29]
uid                  Chris Barros <cbarros@hbs.edu>
sub   2048R/3C211111 2016-03-29 [expires: 2019-03-29]
```

Video demonstrating encrypting and decrypting https://www.youtube.com/watch?v=ZSa-d_9O5DA

GNU Privacy Handbook
https://www.gnupg.org/gph/en/manual/x56.html

Exchanging keys

To communicate with others you must exchange public keys. To list the keys on your public keyring use the command-line option --list-keys.

```
alice% gpg --list-keys
/users/alice/.gnupg/pubring.gpg
---------------------------------------
pub  1024D/BB7576AC 1999-06-04 Alice (Judge) <alice@cyb.org>
sub  1024g/78E9A8FA 1999-06-04
```

Exporting a public key

To send your public key to a correspondent you must first export it. The command-line option --export is used to do this. It takes an additional argument identifying the public key to export. As with the --gen-revoke option, either the key ID or any part of the user ID may be used to identify the key to export.

```
alice% gpg --output alice.gpg --export alice@cyb.org
```

The key is exported in a binary format, but this can be inconvenient when the key is to be sent though email or published on a web page. GnuPG therefore supports a command-line option --armor[1] that that causes output to be generated in an ASCII-armored format similar to uuencoded documents. In general, any output from GnuPG, e.g., keys, encrypted documents, and signatures, can be ASCII-armored by adding the --armor option.

```
alice% gpg --armor --export alice@cyb.org
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v0.9.7 (GNU/Linux)
Comment: For info see http://www.gnupg.org

[...]
-----END PGP PUBLIC KEY BLOCK-----
```

Importing a public key

A public key may be added to your public keyring with the --import option.

```
alice% gpg --import blake.gpg
gpg: key 9E98BC16: public key imported
gpg: Total number processed: 1
gpg:               imported: 1
alice% gpg --list-keys
/users/alice/.gnupg/pubring.gpg
---------------------------------------
```

```
pub  1024D/BB7576AC 1999-06-04 Alice (Judge) <alice@cyb.org>
sub  1024g/78E9A8FA 1999-06-04

pub  1024D/9E98BC16 1999-06-04 Blake (Executioner) <blake@cyb.org>
sub  1024g/5C8CBD41 1999-06-04
```

Once a key is imported it should be validated. GnuPG uses a powerful and flexible trust model that does not require you to personally validate each key you import. Some keys may need to be personally validated, however. A key is validated by verifying the key's fingerprint and then signing the key to certify it as a valid key. A key's fingerprint can be quickly viewed with the --fingerprint command-line option, but in order to certify the key you must edit it.

```
alice% gpg --edit-key blake@cyb.org

pub  1024D/9E98BC16  created: 1999-06-04 expires: never      trust: -/q
sub  1024g/5C8CBD41  created: 1999-06-04 expires: never
(1)  Blake (Executioner) <blake@cyb.org>

Command> fpr
pub  1024D/9E98BC16 1999-06-04 Blake (Executioner) <blake@cyb.org>
             Fingerprint: 268F 448F CCD7 AF34 183E  52D8 9BDE 1A08 9E98 BC16
```

A key's fingerprint is verified with the key's owner. This may be done in person or over the phone or through any other means as long as you can guarantee that you are communicating with the key's true owner. If the fingerprint you get is the same as the fingerprint the key's owner gets, then you can be sure that you have a correct copy of the key.

After checking the fingerprint, you may sign the key to validate it. Since key verification is a weak point in public-key cryptography, you should be extremely careful and *always* check a key's fingerprint with the owner before signing the key.

```
Command> sign

pub  1024D/9E98BC16  created: 1999-06-04 expires: never      trust: -/q
             Fingerprint: 268F 448F CCD7 AF34 183E  52D8 9BDE 1A08 9E98 BC16

     Blake (Executioner) <blake@cyb.org>

Are you really sure that you want to sign this key
with your key: "Alice (Judge) <alice@cyb.org>"

Really sign?
```

Once signed you can check the key to list the signatures on it and see the signature that you have added. Every user ID on the key will have one or more self-signatures as well as a signature for each user that has validated the key.

```
Command> check
uid  Blake (Executioner) <blake@cyb.org>
sig!       9E98BC16 1999-06-04   [self-signature]
sig!       BB7576AC 1999-06-04   Alice (Judge) <alice@cyb.org>
```

Encrypting and Decrypting

Encrypting and decrypting documents

A public and private key each have a specific role when encrypting and decrypting documents. A public key may be thought of as an open safe. When a correspondent encrypts a document using a public key, that document is put in the safe, the safe shut, and the combination lock spun several times. The corresponding private key is the combination that can reopen the safe and retrieve the document. In other words, only the person who holds the private key can recover a document encrypted using the associated public key.

The procedure for encrypting and decrypting documents is straightforward with this mental model. If you want to encrypt a message to Alice, you encrypt it using Alice's public key, and she decrypts it with her private key. If Alice wants to send you a message, she encrypts it using your public key, and you decrypt it with your key.

To encrypt a document the option --encrypt is used. You must have the public keys of the intended recipients. The software expects the name of the document to encrypt as input or, if omitted, on standard input. The encrypted result is placed on standard output or as specified using the option --output. The document is compressed for additional security in addition to encrypting it.

```
alice% gpg --output doc.gpg --encrypt --recipient blake@cyb.org doc
```

The --recipient option is used once for each recipient and takes an extra argument specifying the public key to which the document should be encrypted. The encrypted document can only be decrypted by someone with a private key that complements one of the recipients' public keys. In particular, you cannot decrypt a document encrypted by you unless you included your own public key in the recipient list.

To decrypt a message the option --decrypt is used. You need the private key to which the message was encrypted. Similar to the encryption process, the document to decrypt is input, and the decrypted result is output.

```
blake% gpg --output doc --decrypt doc.gpg

You need a passphrase to unlock the secret key for
user: "Blake (Executioner) <blake@cyb.org>"
1024-bit ELG-E key, ID 5C8CBD41, created 1999-06-04 (main key ID 9E98BC16)

Enter passphrase:
```

Documents may also be encrypted without using public-key cryptography. Instead, only a symmetric cipher is used to encrypt the document. The key used to drive the symmetric cipher is derived from a passphrase supplied when the document is encrypted, and for good security, it should not be the same passphrase that you use to protect your private key. Symmetric encryption is useful for securing documents when the passphrase does not need to be communicated to others. A document can be encrypted with a symmetric cipher by using the --symmetric option.

```
alice% gpg --output doc.gpg --symmetric doc
Enter passphrase:
```