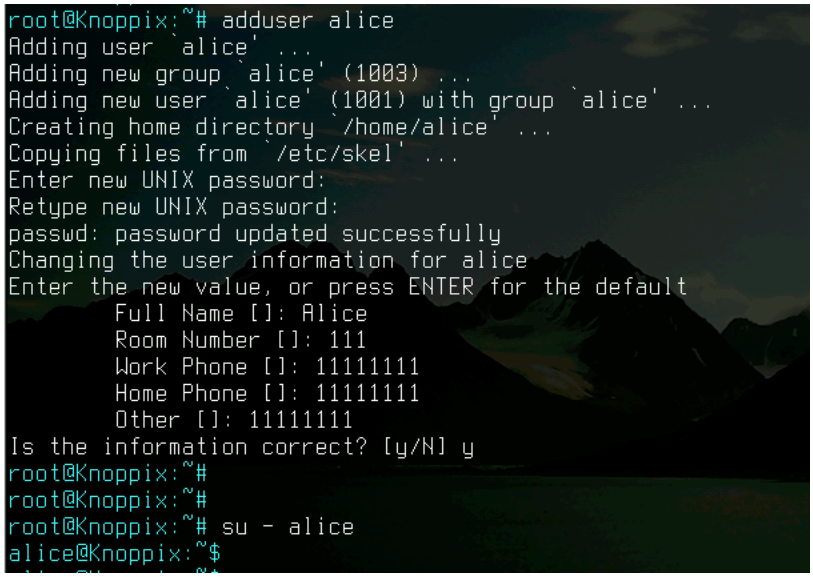
GnuPG is a tool for secure communication. This Lab exercise is a quick-start guide that covers the core functionality of GnuPG. This includes keypair creation, exchanging and verifying keys, encrypting and decrypting documents, and authenticating documents with digital signatures. GnuPG uses public-key cryptography so that users may communicate securely. In a public-key system, each user has a pair of keys consisting of a *private key* and a *public key*. A user's private key is kept secret; it need never be revealed. The public key may be given to anyone with whom the user wants to communicate. GnuPG uses a somewhat more sophisticated scheme in which a user has a primary keypair and then zero or more additional subordinate keypairs. The primary and subordinate keypairs are bundled to facilitate key management and the bundle can often be considered simply as one keypair.

Create two users (1)Alice and (2)Blake users and generate gpg-keys for these two users:

Key generation process:

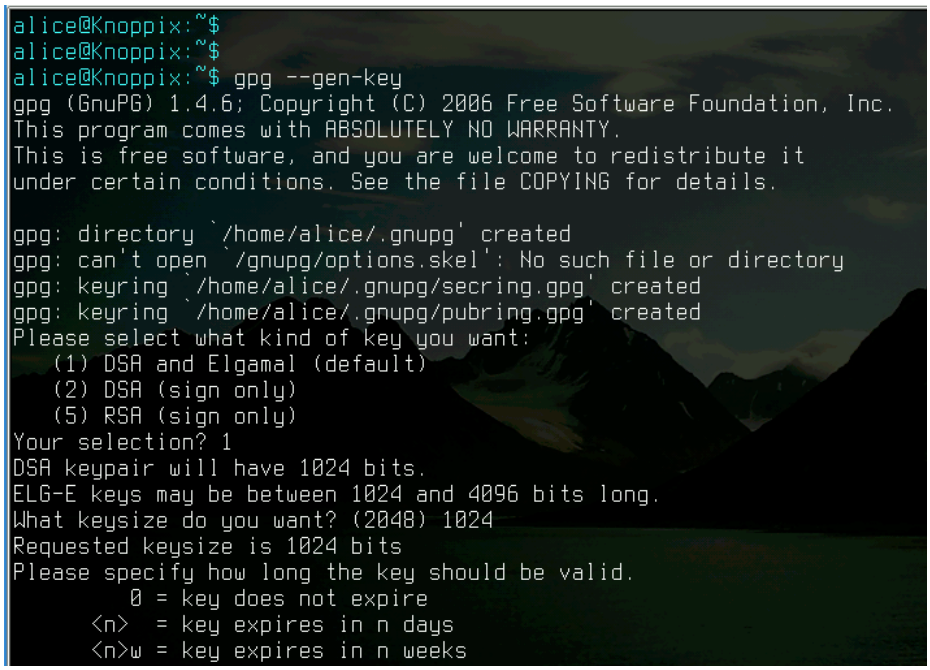
1. To start using GnuPG you need to first run the key generation process. List the command-line and parameters required.

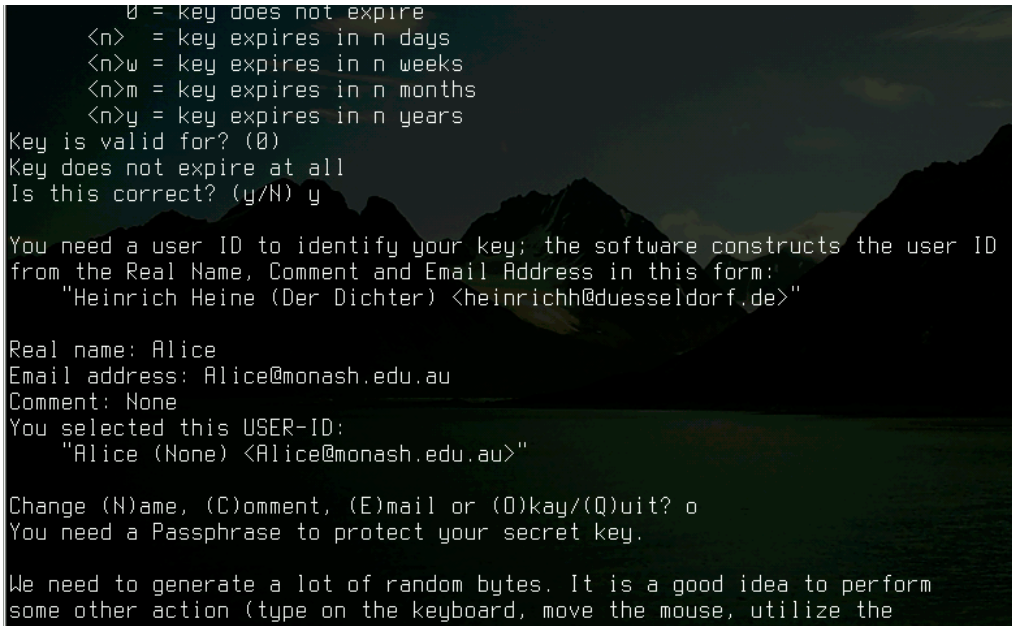
   - knoppix@Knoppix:~$ su – root
   - root@Knoppix:~$ adduser alice
   - root@Knoppix:~$ su – alice
   - alice@Knoppix:~$

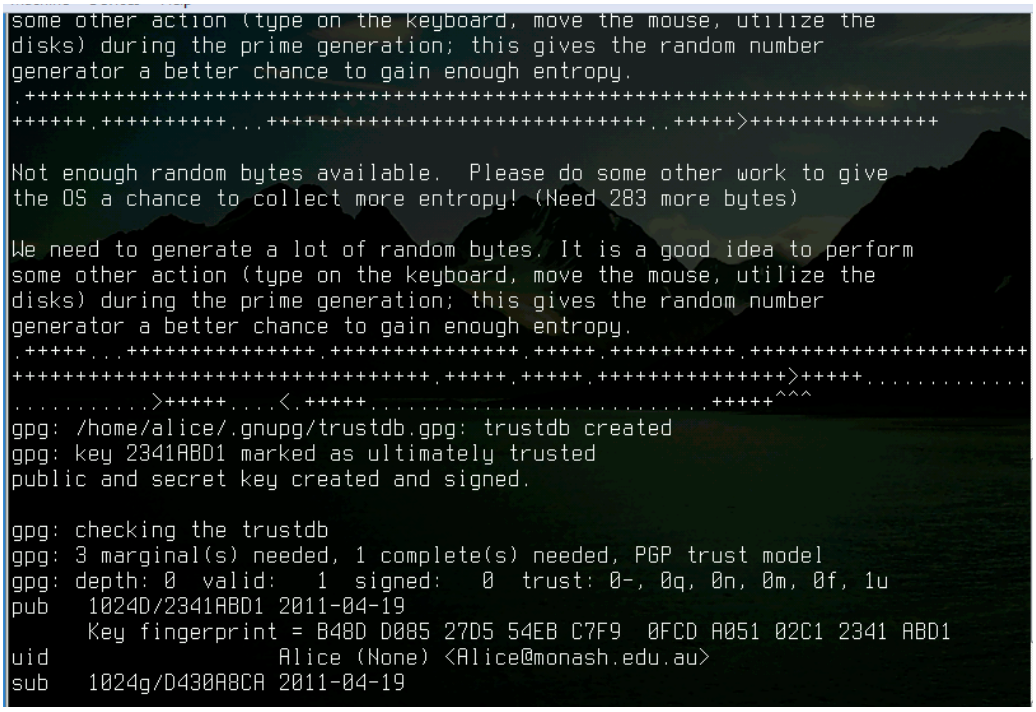


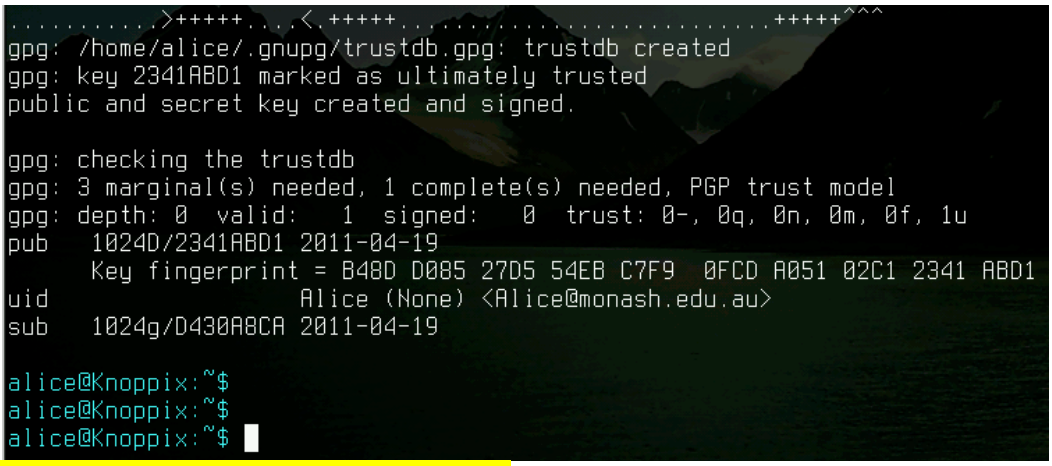
The command-line option --gen-key is used to create a new primary keypair.

- alice@Knoppix:~$ gpg --gen-key

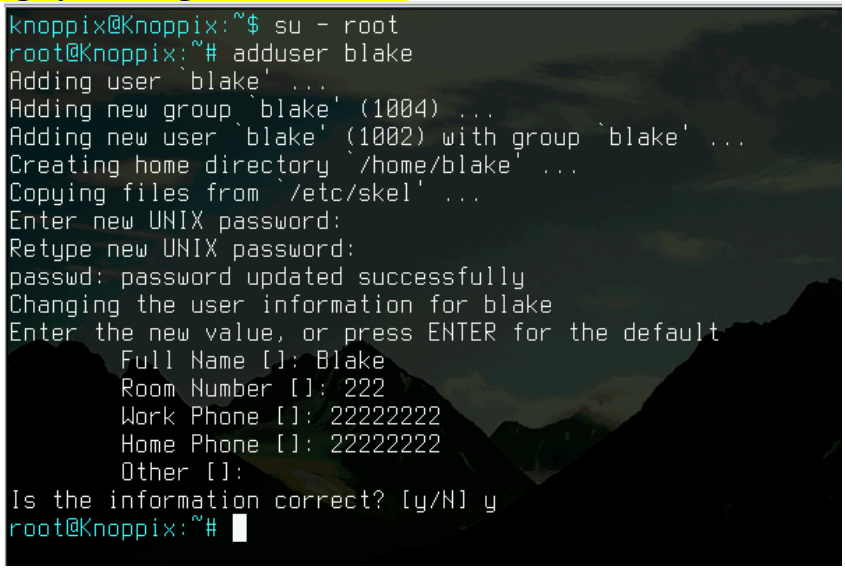

alice@Knoppix:~$
alice@Knoppix:~$
alice@Knoppix:~$ gpg --gen-key
gpg (GnuPG) 1.4.6; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
gpg: directory `/home/alice/.gnupg' created
gpg: can't open `/gnupg/options.skel': No such file or directory
gpg: keyring `/home/alice/.gnupg/secring.gpg' created
gpg: keyring `/home/alice/.gnupg/pubring.gpg' created
Please select what kind of key you want:
(1) DSA and Elgamal (default)
(2) DSA (sign only)
(5) RSA (sign only)
Your selection? 1
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 1024
Requested keysize is 1024 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


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)
Key does not expire at all
Is this correct? (y/N) y
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>"
Real name: Alice
Email address: Alice@monash.edu.au
Comment: None
You selected this USER-ID:
"Alice (None) <Alice@monash.edu.au>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
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



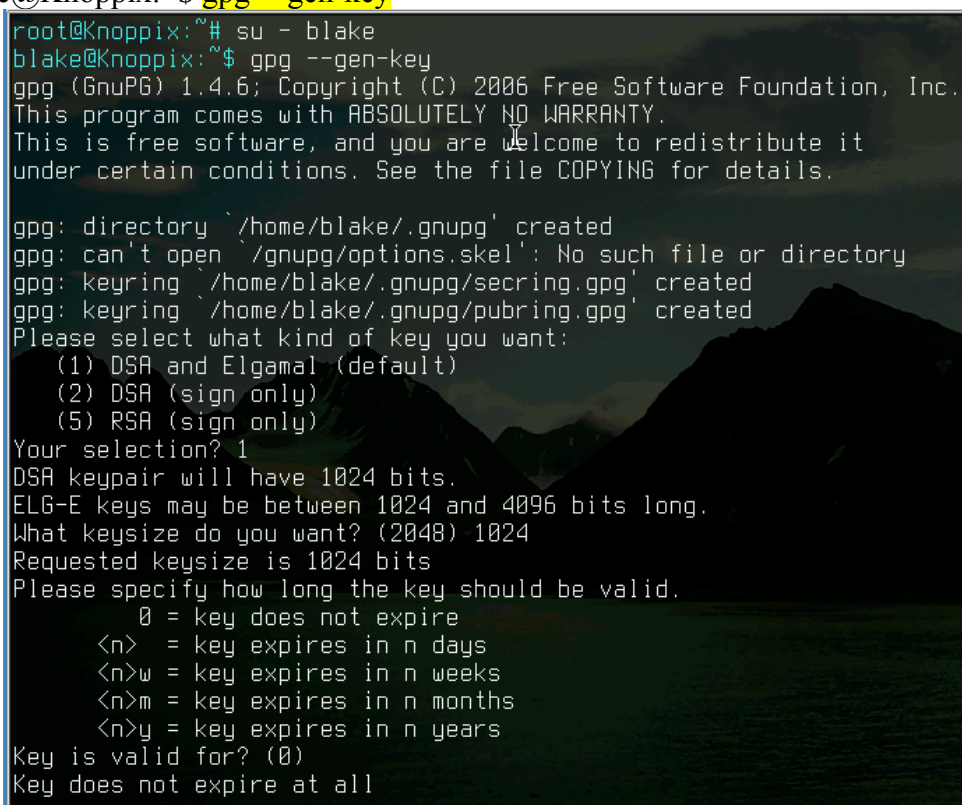


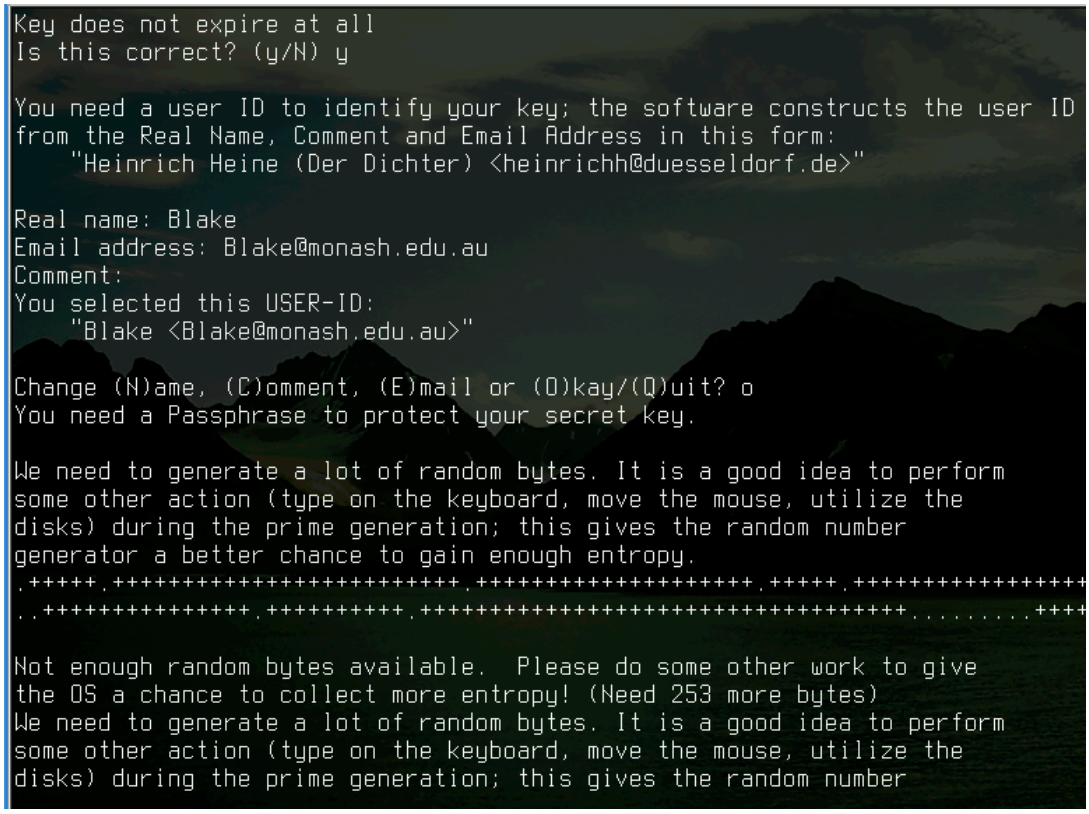
Do the same thing by creating the user Blake:

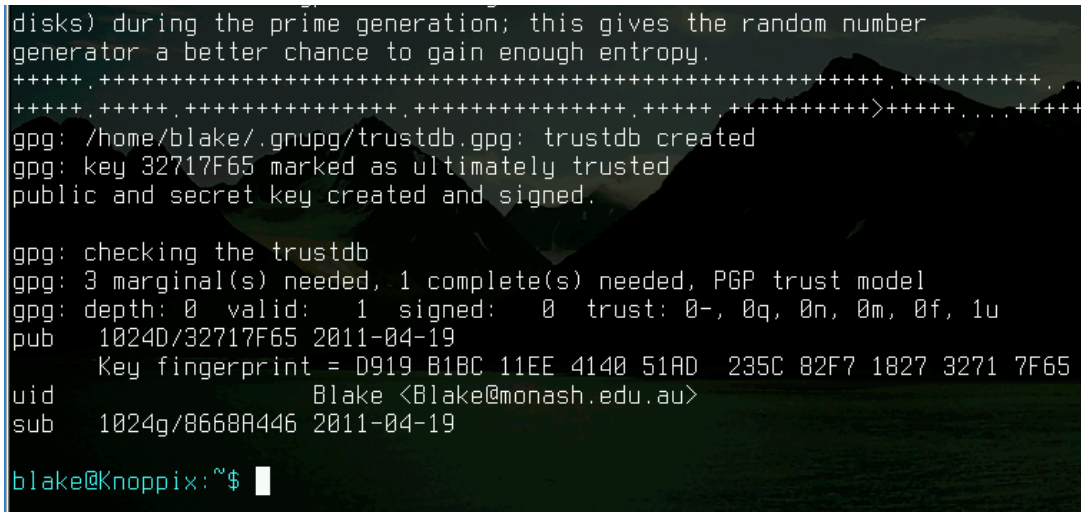


The command-line option --gen-key is used to create a new primary keypair.

- blake@Knoppix:~$ gpg --gen-key



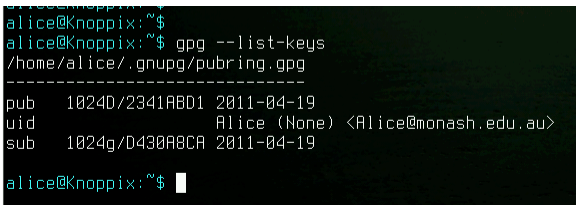




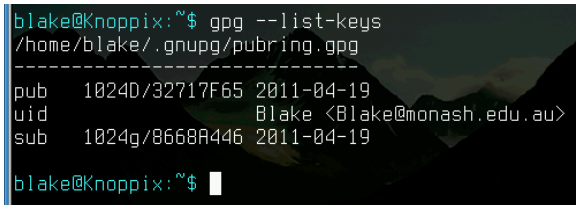
Key Exchanging process between the two users Alice and Blake:

2. To communicate with others you must exchange the public keys. What is the command-line used to list the keys on your public keyring ?

alice@Knoppix:~$ gpg --list-keys

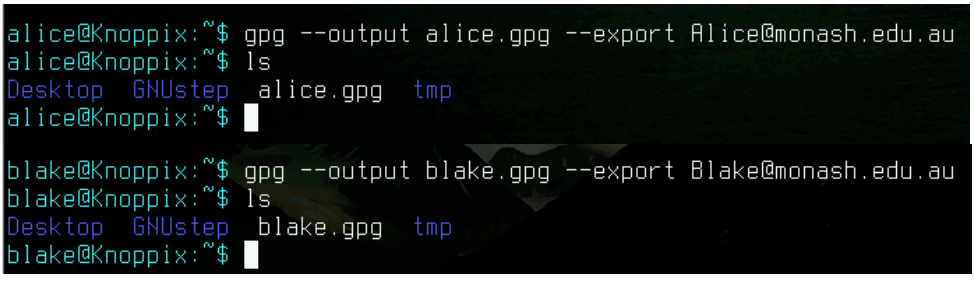


blake@Knoppix:~$ gpg --list-keys



3. **Exporting a public key**. To send your public key to others you must first export it. List the command-line to export your public key. (You can distribute your public key using email or pen drive).

alice@Knoppix:~$gpg - -output alice.gpg - -export Alice@monash.edu.au
blake@Knoppix:~$gpg - -output blake.gpg - -export Blake@monash.edu.au



4. **Importing a public key**. A public key of others may be added to your public keyring . List the command-line to import a public key to your public keyring.
   - A public key may be added to your public keyring with the **--import** option.

alice@Knoppix:~$gpg - -import /home/blake/blake.gpg
blake@Knoppix:~$gpg - -import /home/alice/alice.gpg

```
alice@Knoppix:~$
alice@Knoppix:~$ gpg --import /home/blake/blake.gpg
gpg: key 32717F65: public key "Blake <Blake@monash.edu.au>" imported
gpg: Total number processed: 1
gpg:               imported: 1
alice@Knoppix:~$
```

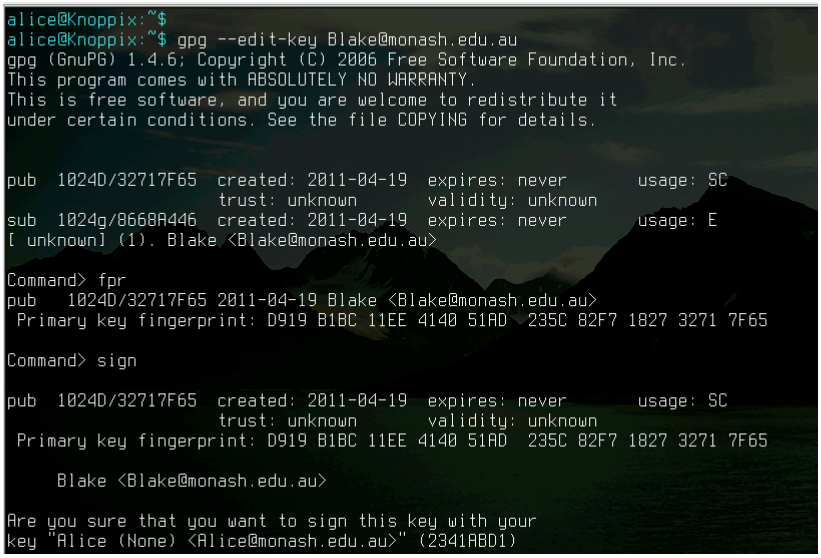
```
blake@Knoppix:~$
blake@Knoppix:~$ gpg --import /home/alice/alice.gpg
gpg: key 2341ABD1: public key "Alice (None) <Alice@monash.edu.au>" imported
gpg: Total number processed: 1
gpg:               imported: 1
blake@Knoppix:~$
```

5. **Validating imported keys**. Once a key is imported it should be validated. How to validate a public key?

alice@Knoppix:~$gpg - -edit-key Blake@monash.edu.au
blake@Knoppix:~$gpg - -edit-key Alice@monash.edu.au

Command > fpr
Command> sign
Command> check

```
alice@Knoppix:~$
alice@Knoppix:~$ gpg --edit-key Blake@monash.edu.au
gpg (GnuPG) 1.4.6; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.


pub  1024D/32717F65  created: 2011-04-19  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  1024g/8668A446  created: 2011-04-19  expires: never       usage: E
[ unknown] (1). Blake <Blake@monash.edu.au>

Command> fpr
pub   1024D/32717F65 2011-04-19 Blake <Blake@monash.edu.au>
 Primary key fingerprint: D919 B1BC 11EE 4140 51AD  235C 82F7 1827 3271 7F65

Command> sign

pub  1024D/32717F65  created: 2011-04-19  expires: never       usage: SC
                     trust: unknown       validity: unknown
 Primary key fingerprint: D919 B1BC 11EE 4140 51AD  235C 82F7 1827 3271 7F65

     Blake <Blake@monash.edu.au>

Are you sure that you want to sign this key with your
key "Alice (None) <Alice@monash.edu.au>" (2341ABD1)
```



You need to run the command on both the user sides and it will prompt with command to check finger print, signing and checking the imported keys..

Command> fpr

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.

Command>sign

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>check

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.

6. Encrypting and decrypting files

At ALICE user end:-
alice@Knoppix:~$vi file.txt (using vi editor we will create a sample text file called file.txt)

- **Encrypting file file.txt to file.gpg**

alice@Knoppix:~$gpg - -output file.gpg - -encrypt - -recipient Blake@monash.edu.au file.txt

Now instead of transferring the file via email or file transfer we will use the root users to copy the encrypted file from alice home directory to blake's home directory.

root@Knoppix:~$cp /home/alice/file.gpg /home/blake/file.gpg

- **Transferring the file file.gpg from Alice to Blake's home directory**

At BLAKE user end:-
blake@Knoppix:~$gpg - -output file.txt - - decrypt file.gpg file.gpg

- **Decrypting file file.gpg to file.txt**

7. Compression/Uncompressing with signing and verifying the documents.

List the command-line to compress and sign the document using digital signature.
At Alice user end:-
alice@Knoppix:~$gpg - -output file.sig –sign file.txt

You need a passphrase to unlock the private key for
user: "Alice (Judge) <alice@monash.edu.au>"
1024-bit DSA key, ID BB7576AC, created 1999-06-04
Enter passphrase:
The document is compressed before being signed, and the output is in binary format.

Uncompressing and verify the signed document received by Blake user.

Given a compressed signed file, you can either check the signature or check the signature and recover the original document.
List the command-line to check the signature and to verify the signature and extract the file.

At BLAKE user end:-
blake@Knoppix:~$gpg - -output file.txt - - decrypt file.sig

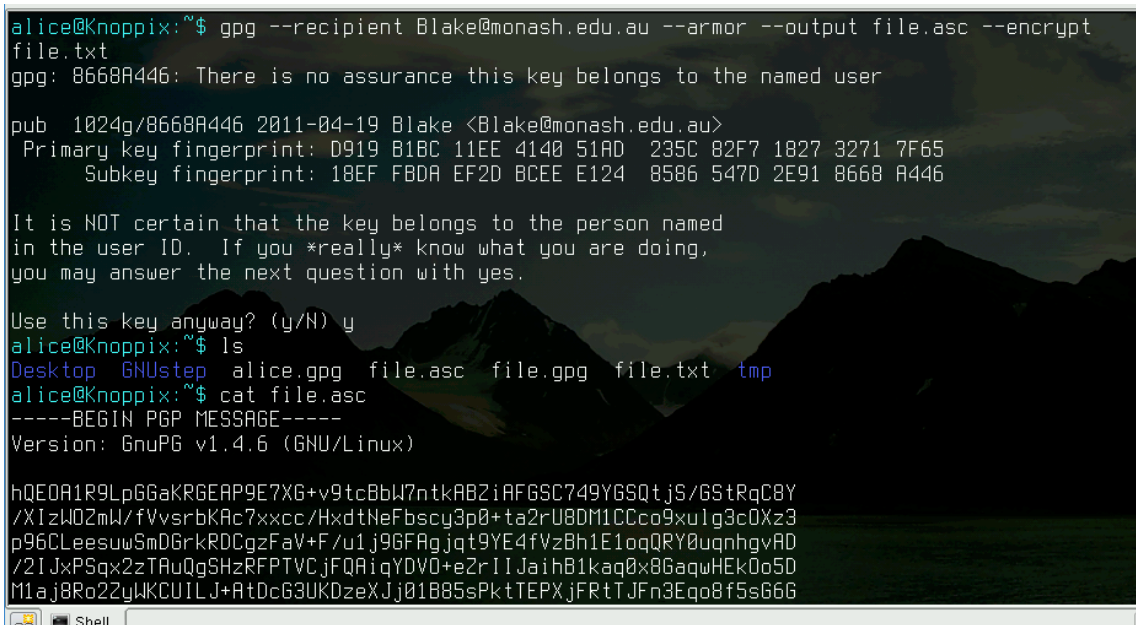
gpg: Signature made Fri Jun 4 12:02:38 1999 CDT using DSA key ID BB7576AC
gpg: Good signature from "Alice (Judge) <alice@monash.edu.au>"

8. Binary to text encoding using ASCII armor referred as Radix-64 in GPG.

A binary-to-text encoding is encoding of binary data into plain text. More precisely, it is an encoding of binary data into a sequence of ASCII-printable characters. These encodings are necessary for transmission of data when the channel or the protocol only allows ASCII-printable characters, such as e-mail. GPG documentation uses the term ASCII armor for binary-to-text encoding when referring to Radix-64 (R64). GnuPG supports a command-line option –armor that causes output to be generated in an ASCII-armored format similar to uuencoded documents. In general, any output from GnuPG, e.g., emails, keys, encrypted documents, and signatures, can be ASCII-armored by adding the --armor option.

At Alice user end:-
alice@Knoppix:~$gpg - -r Blake@monash.edu.au - -armor - -output file.asc - -encrypt file.txt

alice@Knoppix:~$cat file.asc
To see the ASCII content of the file we just armored.



In this example GPG encrypts my text file file.txt and produces an ASCII Armored file named file.asc. When using the --encrypt command, you may receive a warning from GPG about the "trust" in a key's owner:

****

Reference:
The GNU Privacy Handbook, Copyright © 1999 by The Free Software Foundation,
http://www.gnupg.org/gph/en/manual.html

****