Generating, exporting, securing your PGP and SSH keys for backups and restoring them

I recently ran into a tiny problem when I forgot to backup my PGP and SSH keys. So, here's a li'l article on generating, exporting, securing your PGP and SSH keys for backups and restoring them from that backup.

PGP (GnuPG)

Generating keys:

When you run `$ gpg --gen-key`, you're walked through the whole process of creating keys.

```
nitin@trusty:~$ gpg --gen-key
gpg (GnuPG) 1.4.16; Copyright (C) 2013 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? 1
```

The default option chooses the RSA algorithm for both encryption and signing which is what I chose.

The next section asks for the key size,

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

The default is `2048`, but I went with `4096`

The next section is for when the key should expire,

```
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) 0
Key does not expire at all
Is this correct? (y/N) y
```

Next section is filling out some personal details,

```
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: Nitin Venkatesh
Email address: xxxxxxx@gmail.com
Comment: nitstorm
You selected this USER-ID:
    "Nitin Venkatesh (nitstorm) <xxxxxxx@gmail.com>"

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

The second to last section is the passphrase generation,

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

Lastly, play around on the computer,

```
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.
```

If this stage's taking too long, you can run something like `ls -laR /` a couple of times in a separate terminal window and that should do the trick.

```
gpg: key 38054D64 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:   3  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 3u
pub   4096R/12345678 2014-08-08
      Key fingerprint = 1234 0047 7F34 7A6E C27E  47F4 F508 62CD A602 4D69
uid                  Nitin Venkatesh (nitstorm) <xxxxxxx@gmail.com>
sub   4096R/87654321 2014-08-08
```

Exporting keys:

If you've forgotten your public keys, you can list the public keys on your system with `$ gpg --list-keys` and the private keys with `$ gpg --list-secret-keys`

```
nitin@trusty:~$ gpg --list-keys
/home/nitin/.gnupg/pubring.gpg
------------------------------
pub   4096R/12345678 2014-08-08
uid                  Nitin Venkatesh (nitstorm) <xxxxxxx@gmail.com>
sub   4096R/87654321 2014-08-08
```

```
nitin@trusty:~$ gpg --list-secret-keys
/home/nitin/.gnupg/secring.gpg
------------------------------
sec   4096R/12345678 2014-08-08
uid                  Nitin Venkatesh (nitstorm) <xxxxxxx@gmail.com>
ssb   4096R/87654321 2014-08-08
```

Here `12345678` is the key id. The lines `pub` and `sub` are the public encryption and signing key ids and the lines `sec` and `ssb` are the private encryption and signing key ids. The first part of these two lines (`4096R` in our example) indicates that we our key-size and algorithm used (key-size of `4096` and `RSA` algorithm in our example).

You can export your public keys with,

```
gpg -ao <filename> --export <key-id>
```

and can export your private keys with,

```
gpg -ao <filename> --export-secret-keys <key-id>
```

I exported my keys as `nitstorm-pgp-public.key` and `nitstorm-pgp-private.key`,

```
nitin@trusty:~$ gpg -ao nitstorm-pgp-public.key --export 12345678
nitin@trusty:~$ gpg -ao nitstorm-pgp-private.key --export-secret-keys 87654321

nitin@trusty:~$ ls *.key
nitstorm-pgp-private.key  nitstorm-pgp-public.key
```

SSH

Generating keys

To generate SSH keys use the following command,

```
$ ssh-keygen -t <algorithm> -b <key-size>
```

I use the same RSA algorithm and the same keysize of 4096 for SSH as well. Hence, the command to generate my SSH keys would be `$ ssh-keygen -t rsa -b 4096`,

nitin@trusty:~$ ssh-keygen -t rsa -b 4096 Generating public/private rsa key pair.

You are prompted as to where to save the file, leave it blank and hit return to use the default value (`/$HOME/.ssh/id_rsa`)

```
Enter file in which to save the key (/home/nitin/.ssh/id_rsa):
```

Next, you are prompted for a passphrase, type it and hit `Return`. You'll have to repeat the passphrase again.

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

In a few moments your key will have been generated and stored in the location you specified in the previous step.

```
Your identification has been saved in /home/nitin/.ssh/id_rsa.
Your public key has been saved in /home/nitin/.ssh/id_rsa.pub.
The key fingerprint is:
b1:12:3f:67:2d:71:1f:d1:28:4a:cb:69:9d:0f:cf:e2 nitin@trusty
The key's randomart image is:
+--[ RSA 4096]----+
|    =E.          |
|    S oo.=       |
|     +.          |
|    .S+          |
|     .oS.        |
|     .=          |
|     o .So       |
|          . o    |
|           .+o   |
+-----------------+
```

And ta-da, your public-key should be at `/$HOME/.ssh/id_rsa.pub` and your private-key at `/$HOME/.ssh/id_rsa`

Backing up your keys

There are a lot of different ways to backup and secure your keys and each one could argue that one way is better than the other. I'm just outlining what I prefer doing. You could do something that suits you best (like using 7-zip to create an encrypted compressed file of a directory holding the keys).

Now, if you remember correctly, our PGP keys are in our `$HOME` directory and the SSH keys are in our `$HOME/.ssh` directory. Let's consolidate them into a single directory for backup. I'm choosing to do this in my `$HOME/Documents` directory under a new folder called `sensitve`

```
nitin@trusty:~$ cd Documents
nitin@trusty:~/Documents$ mkdir sensitive
nitin@trusty:~/Documents$ cp ../nitstorm*.key ../.ssh/id_rsa* sensitive
```

Now, I create a `tar` file of the directory which makes it easier for encryption,

```
nitin@trusty:~$ tar -cvf sensitive.tar sensitive
```

Finally, I encrypt the `tar` archive file with AES-256 and use a system-generated salt.

```
nitin@trusty:~/Documents$ openssl enc -aes256 -in sensitive.tar -out sensitive.tar.enc -salt
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
```

This prompts for a password and verifies the password a second time, and finally you have your AES-256 encrypted `sensitive.tar.enc` ready. This can be backed up as you please.

Restoring from backup:

Decrypting the file is also similar to the above command, except that we use the `-d` switch in addition.

```
nitin@trusty:~/Documents$ openssl enc -aes256 -in sensitive.tar.enc -out sensitive.tar -salt -d
enter aes-256-cbc decryption password:
```

You can untar and access your keys using `$ tar -xvf sensitive.tar` which will create a directory called `sensitive` in your current working directory.

Now copy the SSH keys into `$HOME/.ssh/`.

```
nitin@trusty:~/Documents$ cp sensitive/id_rsa* ../.ssh/
```

To import the PGP public key, use the `$ gpg --import <public-key-file>` and to import the private key, use, `$ gpg --allow-secret-key-import --import <private-key-file>`.

```
nitin@trusty:~/Documents$ gpg --import nitstorm-pgp-public.key
nitin@trusty:~/Documents$ gpg --allow-secret-key-import --import nitstorm-pgp-private.key
```