Experiencing Technology
Tinned-Software Blog

Create GnuPG key with sub-keys to sign, encrypt, authenticate

Posted on 2017-07-10 by Gerhard

In order to use a GnuPG key on a smartcard or Yubikey, a GnuPG key needs to be created. This post will show you how to create a GnuPG key with sub-keys for signing, encryption and authentication. The authentication key can be used later on to authenticate via ssh as well.

Configure GnuPG

Before the key can be generated, first you need to configure GnuPG. The following settings are suggested before creating the key. The settings contain the documentation from the official GnuPG documentation. Add these settings to the "gpg.conf" file located in the GnuPG home directory. This is either the "~/.gnupg/" or the directory specified in the "–homedir" parameter.


```
########################################################################
# GnuPG Options

# (OpenPGP-Configuration-Options)
# Assume that command line arguments are given as UTF8 strings.
utf8-strings

# (OpenPGP-Protocol-Options)
# Set the list of personal digest/cipher/compression preferences. This allows
# the user to safely override the algorithm chosen by the recipient key
# preferences, as GPG will only select an algorithm that is usable by all
# recipients.
personal-digest-preferences SHA512 SHA384 SHA256 SHA224
personal-cipher-preferences AES256 AES192 AES CAST5 CAMELLIA192 BLOWFISH TWOFISH CAMELLIA1
personal-compress-preferences ZLIB BZIP2 ZIP

# Set the list of default preferences to string. This preference list is used
# for new keys and becomes the default for "setpref" in the edit menu.
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP

# (OpenPGP-Esoteric-Options)
# Use name as the message digest algorithm used when signing a key. Running the
# program with the command --version yields a list of supported algorithms. Be
# aware that if you choose an algorithm that GnuPG supports but other OpenPGP
# implementations do not, then some users will not be able to use the key
# signatures you make, or quite possibly your entire key.
#
# SHA-1 is the only algorithm specified for OpenPGP V4. By changing the
# cert-digest-algo, the OpenPGP V4 specification is not met but with even
# GnuPG 1.4.10 (release 2009) supporting SHA-2 algorithm, this should be safe.
# Source: https://tools.ietf.org/html/rfc4880#section-12.2
```

```
cert-digest-algo SHA512
digest-algo SHA256

# Selects how passphrases for symmetric encryption are mangled. 3 (the default)
# iterates the whole process a number of times (see --s2k-count).
s2k-mode 3

# (OpenPGP-Protocol-Options)
# Use name as the cipher algorithm for symmetric encryption with a passphrase
# if --personal-cipher-preferences and --cipher-algo are not given. The
# default is AES-128.
s2k-cipher-algo AES256

# (OpenPGP-Protocol-Options)
# Use name as the digest algorithm used to mangle the passphrases for symmetric
# encryption. The default is SHA-1.
s2k-digest-algo SHA512

# (OpenPGP-Protocol-Options)
# Specify how many times the passphrases mangling for symmetric encryption is
# repeated. This value may range between 1024 and 65011712 inclusive. The
# default is inquired from gpg-agent. Note that not all values in the
# 1024-65011712 range are legal and if an illegal value is selected, GnuPG will
# round up to the nearest legal value. This option is only meaningful if
# --s2k-mode is set to the default of 3.
s2k-count 1015808

################################################################################
# GnuPG View Options

# Select how to display key IDs. "long" is the more accurate (but less
# convenient) 16-character key ID. Add an "0x" to include an "0x" at the
# beginning of the key ID.
keyid-format 0xlong

# List all keys with their fingerprints. This is the same output as --list-keys
# but with the additional output of a line with the fingerprint. If this
# command is given twice, the fingerprints of all secondary keys are listed too.
with-fingerprint
with-fingerprint
```

The “cert-digest-algo” and “digest-algo” also contain a personal explanation why these settings where chosen even if they are supposed to break the OpenPGP specification.

The last settings in the above example are options influencing the way the keys are shown in the output of GnuPG. It is widely suggested to use the long keyid format to identify the keys. The keyid is a short representation of the fingerprint. The long format takes the last 16 (instead of 8 in the short format) characters from the fingerprint. To be really sure to have the correct key, the fingerprint is the information to use instead of any shortened version of it.

Generate a master-key

With the configuration in place, the master key can be created. The master key will only have the capability “Certify” and is only needed when the key is modified.

When OpenPGP 2.x is used, the program itself is called “gpg2” in many distributions but also the option to generate the key has changed to “–full-gen-key”. If the option “–gen-key” is used with gpg2, many settings described in this example can not be selected.

```
$ gpg --homedir ./gnupg-test --expert --gen-key
gpg (GnuPG) 1.4.20; Copyright (C) 2015 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)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
Your selection? 8

Possible actions for a RSA key: Sign Certify Encrypt Authenticate
Current allowed actions: Sign Certify Encrypt

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection? S

Possible actions for a RSA key: Sign Certify Encrypt Authenticate
Current allowed actions: Certify Encrypt

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection? E

Possible actions for a RSA key: Sign Certify Encrypt Authenticate
Current allowed actions: Certify

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection? Q
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096
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) 2y
Key expires at Fri 30 Nov 2018 10:44:14 PM CET
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@example.com
Comment:
You selected this USER-ID:
    "Alice <alice@example.com>"

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

Enter passphrase: YourPassword
Repeat passphrase: YourPassword

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: key 0xD93D03C13478D580 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
gpg: next trustdb check due at 2018-11-30
pub   4096R/0xD93D03C13478D580 2016-11-30 [expires: 2018-11-30]
      Key fingerprint = F8C8 1342 2A7F 7A3A 9027  E158 D93D 03C1 3478 D580
uid                          Alice <alice@example.com>
```

The above will generate a 4096 bit key which will be used as a master key. The parameter "–homedir ./gnupg-test/" defines the directory used for the keyring to be generated. This is the directory where the previously created gpg.conf file should be located.

I suggest to use a removable storage like a USB-Stick to store the master-key on. The master key is not used in every day operation and should be stored in a safe place.

Generating the random data needed for the key generation might take a long time. To speed up the process, use the rngd(8) to feed random data into the random number pool of the kernel. This can be done by installing the **rng-tools** package.

```
# Debian / Ubuntu
$ sudo apt-get install rng-tools

# RedHat / CentOS
$ yum install rng-tools
```

To verify the generated keys, execute the following command.

```
$ gpg --homedir ./gnupg-test -K
./gnupg-test/secring.gpg
------------------------
sec   4096R/0xD93D03C13478D580 2016-11-30 [expires: 2018-11-30]
      Key fingerprint = F8C8 1342 2A7F 7A3A 9027  E158 D93D 03C1 3478 D580
uid                          Alice <alice@example.com>
```

According to the view options in the gpg.conf configuration file, the output shows the long keyid format as well as the fingerprint of each key or subkey.

Set key preferences

To ensure that only strong algorithms are used, set the preferences for the key using the "setpref" command.

```
$ gpg --homedir ./gnupg-test --expert --edit-key 0xD93D03C13478D580
gpg (GnuPG) 1.4.20; Copyright (C) 2015 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.

Secret key is available.

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
[ultimate] (1). Alice <alice@example.com>

gpg> setpref SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP Uncompresse
```

```
Set preference list to:
     Cipher: AES256, AES192, AES, CAST5, 3DES
     Digest: SHA512, SHA384, SHA256, SHA224, SHA1
     Compression: ZLIB, BZIP2, ZIP, Uncompressed
     Features: MDC, Keyserver no-modify
Really update the preferences? (y/N) Y

You need a passphrase to unlock the secret key for
user: "Alice <alice@example.com>"
4096-bit RSA key, ID 0xD93D03C13478D580, created 2016-11-30

Enter passphrase: YourPassword

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
[ultimate] (1). Alice <alice@example.com>

gpg> save
```

Generate the sub-keys

Before starting to generate the different sub-keys, verify the maximum size of keys the smartcard can store. The Yubikey NEO can store keys up to 2048 bits while the Yubikey 4 can store keys up to 4096 bits. Smartcards usually support different sizes as well like 2048, 3072 or 4096 bits.

To add a subkey, the master-key needs to be opened for editing. The following command will open the key specified (in the following example via key ID) for editing. To be able to create all the different key types, the "–expert" option is used.

```
$ gpg --homedir ./gnupg-test --expert --edit-key 0xD93D03C13478D580
gpg (GnuPG) 1.4.20; Copyright (C) 2015 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.

Secret key is available.

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
[ultimate] (1). Alice <alice@example.com>

gpg>
```

Signing sub-key

With the key opened for editing, the sub-key can be added to it. To start the guided process of creating a sub-key the command is "addkey".

After the passphrase is entered, the type of sub-key must be entered. For a signing key, the "(4) RSA (sign only)" is used. The key size should match the size fitting on the smartcard or Yubikey.

While GnuPG version 1 will ask for the passphrase at the beginning of the "addkey" procedure, version 2 will ask at the end of the creation process for an individual passphrase for the new subkey as well as for the passphrase of the master key.

```
gpg> addkey
Key is protected.

You need a passphrase to unlock the secret key for
```

```
user: "Alice <alice@example.com>"
4096-bit RSA key, ID 0xD93D03C13478D580, created 2016-11-30

Enter passphrase: YourPassword

Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
Your selection? 4
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 3072
Requested keysize is 3072 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) 2y
Key expires at Fri 30 Nov 2018 10:44:21 PM CET
Is this correct? (y/N) Y
Really create? (y/N) Y
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.
.............+++++
+++++

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
[ultimate] (1). Alice <alice@example.com>

gpg>
```

The output above now shows an additional sub-key for signing (“usage: S”). If there is no additional sub-key to be created, the process can be ended by the command “save” to store the modifications to the key.

```
gpg> save
```

Encryption sub-key

An encryption key can now be created in the same way as the signing key just by selecting the “RSA (encrypt only)” key type.

```
gpg> addkey
Key is protected.

You need a passphrase to unlock the secret key for
user: "Alice <alice@example.com>"
4096-bit RSA key, ID 0xD93D03C13478D580, created 2016-11-30

Enter passphrase: YourPassword

Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
```

```
Your selection? 6
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 3072
Requested keysize is 3072 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) 2y
Key expires at Fri 30 Nov 2018 10:44:23 PM CET
Is this correct? (y/N) Y
Really create? (y/N) Y
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.
.......+++++
.....................+++++

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
sub  3072R/0x76737ABEB92745D7  created: 2016-11-30  expires: 2018-11-30  usage: E
[ultimate] (1). Alice <alice@example.com>

gpg>
```

Authentication sub-key

When the GnuPG key should be used for authentication, an additional authentication subkey needs to be created. Such a sub-key can be used to authenticate when connecting via ssh.

To create such a authentication sub-key, the type “(8) RSA (set your own capabilities)” needs to be selected.

```
gpg> addkey
Key is protected.

You need a passphrase to unlock the secret key for
user: "Alice <alice@example.com>"
4096-bit RSA key, ID 0xD93D03C13478D580, created 2016-11-30

Enter passphrase: YourPassword

Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
Your selection? 8

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Sign Encrypt

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection?
```

The type (8) allows to set the capability manually. The list above shows the available capabilities. The default assigned capabilities as shown are “Sign” and “Encrypt”.

Disable the default capabilities by entering the related letter followed by ENTER one capability after the other. The use the toggle “A” to enable authentication capability and proceed with “Q”.

```
Your selection? S

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Encrypt

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection? E

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions:

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection? A

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Authenticate

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection? Q
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 3072
Requested keysize is 3072 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) 2y
Key expires at Fri 30 Nov 2018 10:44:26 PM CET
Is this correct? (y/N) Y
Really create? (y/N) Y
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.
..+++++
.....+++++

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
sub  3072R/0x76737ABEB92745D7  created: 2016-11-30  expires: 2018-11-30  usage: E
sub  3072R/0xE379FB0D81B6925D  created: 2016-11-30  expires: 2018-11-30  usage: A
[ultimate] (1). Alice <alice@example.com>

gpg> save
```

As the last output shows, the master key has 4096 bit and the 3 subkeys have a different sizes to fit later on to the smartcard or Yubikey.

Add identities

If necessary, more identities can be added to the GnuPG key. To do so, the key needs to be opened again in edit mode. The “adduid” command is used then to add an additional identity.

```
$ gpg --homedir ./gnupg-test --expert --edit-key 0xD93D03C13478D580
gpg (GnuPG) 1.4.20; Copyright (C) 2015 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.

Secret key is available.

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
sub  3072R/0x76737ABEB92745D7  created: 2016-11-30  expires: 2018-11-30  usage: E
sub  3072R/0xE379FB0D81B6925D  created: 2016-11-30  expires: 2018-11-30  usage: A
[ultimate] (1). Alice <alice@example.com>

gpg> adduid
Real name: Alice
Email address: alice@example.org
Comment:
You selected this USER-ID:
    "Alice <alice@example.org>"

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

You need a passphrase to unlock the secret key for
user: "Alice <alice@example.com>"
4096-bit RSA key, ID 0xD93D03C13478D580, created 2016-11-30

Enter passphrase: YourPassword

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
sub  3072R/0x76737ABEB92745D7  created: 2016-11-30  expires: 2018-11-30  usage: E
sub  3072R/0xE379FB0D81B6925D  created: 2016-11-30  expires: 2018-11-30  usage: A
[ultimate] (1)  Alice <alice@example.com>
[ unknown] (2). Alice <alice@example.org>

gpg> uid 1

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
sub  3072R/0x76737ABEB92745D7  created: 2016-11-30  expires: 2018-11-30  usage: E
sub  3072R/0xE379FB0D81B6925D  created: 2016-11-30  expires: 2018-11-30  usage: A
[ultimate] (2)* Alice <alice@example.com>
[ultimate] (1). Alice <alice@example.org>

gpg> primary

You need a passphrase to unlock the secret key for
user: "Alice <alice@example.com>""
4096-bit RSA key, ID 0xD93D03C13478D580, created 2016-11-30

Enter passphrase: YourPassword

pub  4096R/0xD93D03C13478D580  created: 2016-11-30  expires: 2018-11-30  usage: C
                               trust: ultimate      validity: ultimate
sub  3072R/0x1ED73636975EC6DE  created: 2016-11-30  expires: 2018-11-30  usage: S
sub  3072R/0x76737ABEB92745D7  created: 2016-11-30  expires: 2018-11-30  usage: E
sub  3072R/0xE379FB0D81B6925D  created: 2016-11-30  expires: 2018-11-30  usage: A
[ultimate] (2)* Alice <alice@example.com>
[ultimate] (1)  Alice <alice@example.org>

gpg> save
```

The last output after adding the identity shows the key with a trust of “unknown” which will change to unlimited after saving the new identity with the “save” command.

The “.” at the identity marks the primary identity for that key. As shown above, after adding the second identity, the added identity is selected as primary identity. As of this, the primary identity has been manually defined by selecting the primary identity using the “uid” command followed by the “primary” command to set that identity as primary.

Export the public and secret keys as backup

To backup the keys, export them into a file. Exporting the keys is done in two steps, the private keys and the secret keys are exported separately.

```
$ gpg --homedir ./gnupg-test --export-secret-keys --armor --output secret-keys.gpg 0xD93D0
$ gpg --homedir ./gnupg-test --export --armor --output public-keys.gpg 0xD93D03C13478D580
```

With the first command, all secret keys (master + subkeys) are exported into one file. The second command will export all public keys (master + subkeys) into another file. Those files can be used to backup the created keys.

To export only one particular subkey, the subkey ID can be specified with an “!” exclamation mark at the end of the key ID instructs gpg to only export this particular subkey(s).

```
$ gpg --homedir ./gnupg-test --export-secret-subkeys --armor --output secret-subkey_sign.g
```

The above command will export only the signing subkeys secret key. The gpg man page describes the exclamation mark at the key ID like this.

"When using gpg an exclamation mark (!) may be appended to force using the specified primary or secondary key and not to try and calculate which primary or secondary key to use."

Daily use keyring

For everyday use, you want to “remove” the master secret key from the keyring. This is sometimes refered to as a laptop keyring. Such a keyring can be prepared in multiple ways.

Option 1: Removing the secret master key

By exporting only the secret subkeys, deleting all the secret keys of that key from the keyring (which includes not only the master key but also the subkeys) and then reimporting only the secret subkeys.

```
$ gpg --homedir ./gnupg-test --export-secret-subkeys --armor --output secret-subkeys.gpg 0
```

The above command will export all the secret subkeys of the given key ID and stores it in the given output file.

```
$ gpg --homedir ./gnupg-test --delete-secret-keys 0xD93D03C13478D580
gpg (GnuPG) 1.4.20; Copyright (C) 2015 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.


sec  4096R/0xD93D03C13478D580 2016-11-30 Alice <alice@example.org>

Delete this key from the keyring? (y/N) Y
This is a secret key! - really delete? (y/N) Y
```

Version 2 of GnuPG will ask for every secret key/subkey to be deleted. At this point, the delete operation for the master key can be accepted but the deletion of the subkeys can be denied. This will result in a error message for the delete operation which indicates that the subkeys where not deleted. This allows to delete the secret master key but keep the subkeys and therefore does not require the reimport of the secret subkeys.

After the above command is executed, all the secret keys are removed for the created keys. With the following command the exported secret subkeys are reimported back to the keyring.

```
$ gpg --homedir ./gnupg-test --import ./gnupg-backup/secret-subkeys.gpg
gpg: key 0xD93D03C13478D580: secret key imported
gpg: key 0xD93D03C13478D580: "Alice <alice@example.org>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
gpg:       secret keys read: 1
gpg:   secret keys imported: 1
```

Option 2: Build from backup

If the keyring you used to create is not the keyring you intend to use for every day use, the backup files created earlier can be used to create the daily use keyring. This is the prefered method if you have created the key not on the computer you are using for everyday tasks or you have created the key on a USB drive or similar.

```
$ gpg --homedir ./gnupg-test --export-secret-subkeys --armor --output secret-subkeys.gpg 0
```

As with the first option, an export with only the subkeys has to be created with the command above. Additionally to the following commands, I suggest to also copy the **gpg.conf** used in the keyring to create the key to the daily-use-keyring.

```
$ gpg --homedir ~/.gnupg --import gnupg-backup/public-keys.gpg
gpg: key 0xD93D03C13478D580: public key "Alice <alice@example.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

The above command can be used to import the public key into the keyring you use for everyday use. With the command below, the secret subkeys, without the master secret key, can be imported into the daily use keyring. The “–homedir ~/.gnupg” can be ommited if the keyring is at the default location like in this example.

```
$ gpg --homedir ~/.gnupg --import gnupg-backup/secret-subkeys.gpg
gpg: key 0xD93D03C13478D580: secret key imported
gpg: key 0xD93D03C13478D580: "Alice <alice@example.org>" 1 new signature
```

```
gpg: Total number processed: 1
gpg:         new signatures: 1
gpg:       secret keys read: 1
gpg:   secret keys imported: 1
```

Check the daily use keyring

To verify that only the subkeys secret keys have been imported back into the keyring, execute the following command. This command will list all secret keys. The master key is marked with a hash character "#" indicating that the secret key is missing – as expected.

```
$ gpg --homedir ./gnupg-test -K
./gnupg-test/secring.gpg
------------------------
sec#  4096R/0xD93D03C13478D580 2016-11-30 [expires: 2018-11-30]
      Key fingerprint = F8C8 1342 2A7F 7A3A 9027  E158 D93D 03C1 3478 D580
uid                          Alice <alice@example.com>
uid                          Alice <alice@example.org>
ssb   3072R/0x1ED73636975EC6DE 2016-11-30
      Key fingerprint = 292D 3E78 6B2E DBEA 1D10  02C8 1ED7 3636 975E C6DE
ssb   3072R/0x76737ABEB92745D7 2016-11-30
      Key fingerprint = 0C33 42E5 670A B099 8ED7  3E87 7673 7ABE B927 45D7
ssb   3072R/0xE379FB0D81B6925D 2016-11-30
      Key fingerprint = 7357 2158 947D BAFF A89F  4911 E379 FB0D 81B6 925D
```

Finally keep a backup of the secret keys, in particular the secret master key, and remove any other temporary export file of the secret subkeys not needed any more. Consider using the shred(1) utility to securely delete those files.

Verify the created GnuPG key

As always, verifying the result is important. Thankfully, there is a tool to assist with that. The hkt (hopenpgp-tools) provide exactly that. The utility does not come pre-installed but can be installed directly from the repository of Ubuntu / LinuxMint. So far it seems hopenpgp-tools is not available as rpm for RHEL or CentOS.

```
# Debian / Ubuntu
$ sudo apt-get install hopenpgp-tools

# RedHat / CentOS
# ... not available.
```

The hopenpgp-tools can be used to check the key for hopenpgp-tools setting. The command below uses the "hkt", which is part of hopenpgp-tools, to extract the public key from the keyring. As well, a hopenpgp-tools utility called "hokey" performs the check of the key settings. The green sections show where the best practice is fulfilled. If a setting would not be according to the best practice, the utility would mark it red.

```
$ hkt export-pubkeys --keyring gnupg-test/pubring.gpg 0xD93D03C13478D580 | hokey lint
hokey (hopenpgp-tools) 0.17
Copyright (C) 2012-2015  Clint Adams
hokey comes with ABSOLUTELY NO WARRANTY. This is free software, and you are welcome to red
hkt (hopenpgp-tools) 0.17
Copyright (C) 2012-2015  Clint Adams
hkt comes with ABSOLUTELY NO WARRANTY. This is free software, and you are welcome to redis

Key has potential validity: good
Key has fingerprint: F8C8 1342 2A7F 7A3A 9027  E158 D93D 03C1 3478 D580
```

```
Checking to see if key is OpenPGPv4: V4
Checking to see if key is RSA or DSA (>= 2048-bit): RSA 4096
Checking user-ID- and user-attribute-related items:
  Alice <alice@example.com>:
    Self-sig hash algorithms: [SHA-512]
    Preferred hash algorithms: [SHA-512, SHA-384, SHA-256, SHA-224]
    Key expiration times: [1y11m29d81000s = Fri Nov 30 21:44:14 UTC 2018]
    Key usage flags: [[certify-keys]]
  Alice <alice@example.org>:
    Self-sig hash algorithms: [SHA-512]
    Preferred hash algorithms: [SHA-512, SHA-384, SHA-256, SHA-224]
    Key expiration times: [1y11m29d81000s = Fri Nov 30 21:44:14 UTC 2018]
    Key usage flags: [[certify-keys]]
```

Another way of checking the generated GnuPG key is pgpdump. This utility will not interpret the key information and mark any information considered not good practice but will show you the raw key information. On the other hand, way more details about the key itself are revealed.

The pgpdump utility is also not preinstalled but can be installed directly from the repositories of the distribution. While the “hopenpgp-tools” utility is not available as rpm package, pgpdump is available as rpm for RedHat and CentOS.

```
# Debian / Ubuntu
$ sudo apt install pgpdump

# RedHat / CentOS
$ yum install pgpdump
```

The pgpdump utility takes a secret key export as produced from the backup and dumps all of its information in a human readable format. The below command shows grouped information for the master key and the subkeys.

The first block contains the details about the secret master key. The highlighted area in the first block shows the s2k settings defined in the gpg.conf.

```
$ gpg --homedir ./gnupg-test --export-secret-keys --armor 0xD93D03C13478D580 | pgpdump
Old: Secret Key Packet(tag 5)(1862 bytes)
   Ver 4 - new
   Public key creation time - Wed Nov 30 22:44:14 CET 2016
   Pub alg - RSA Encrypt or Sign(pub 1)
   RSA n(4096 bits) - ...
   RSA e(17 bits) - ...
   Sym alg - AES with 256-bit key(sym 9)
   Iterated and salted string-to-key(s2k 3):
      Hash alg - SHA512(hash 10)
      Salt - 12 43 4f 59 74 01 a2 ff
      Count - 1015808(coded count 159)
   IV - 2f d5 54 c1 1f f0 a6 87 a3 af 58 ef 69 47 ec 4f
   Encrypted RSA d
   Encrypted RSA p
   Encrypted RSA q
   Encrypted RSA u
   Encrypted SHA1 hash
```

The above shows as well that this key is a 4096 bits RSA key. The first block is followed the the identity and its signature. This shows clearly the self signature of the identities from the master key. In the signature the hash algorithm SHA512 as well as the preferred algorithms as defined in the gpg.conf and the key itself can be found.

```
Old: User ID Packet(tag 13)(25 bytes)
   User ID - Alice <alice@example.com>
Old: Signature Packet(tag 2)(573 bytes)
   Ver 4 - new
   Sig type - Positive certification of a User ID and Public Key packet(0x13).
   Pub alg - RSA Encrypt or Sign(pub 1)
   Hash alg - SHA512(hash 10)
   Hashed Sub: signature creation time(sub 2)(4 bytes)
      Time - Wed Nov 30 22:44:14 CET 2016
   Hashed Sub: key flags(sub 27)(1 bytes)
      Flag - This key may be used to certify other keys
   Hashed Sub: key expiration time(sub 9)(4 bytes)
      Time - Fri Nov 30 22:44:14 CET 2018
   Hashed Sub: preferred symmetric algorithms(sub 11)(4 bytes)
      Sym alg - AES with 256-bit key(sym 9)
      Sym alg - AES with 192-bit key(sym 8)
      Sym alg - AES with 128-bit key(sym 7)
      Sym alg - CAST5(sym 3)
   Hashed Sub: preferred hash algorithms(sub 21)(4 bytes)
      Hash alg - SHA512(hash 10)
      Hash alg - SHA384(hash 9)
      Hash alg - SHA256(hash 8)
      Hash alg - SHA224(hash 11)
   Hashed Sub: preferred compression algorithms(sub 22)(4 bytes)
      Comp alg - ZLIB <RFC1950>(comp 2)
      Comp alg - BZip2(comp 3)
      Comp alg - ZIP <RFC1951>(comp 1)
      Comp alg - Uncompressed(comp 0)
   Hashed Sub: features(sub 30)(1 bytes)
      Flag - Modification detection (packets 18 and 19)
   Hashed Sub: key server preferences(sub 23)(1 bytes)
      Flag - No-modify
   Sub: issuer key ID(sub 16)(8 bytes)
      Key ID - 0xD93D03C13478D580
   Hash left 2 bytes - b4 17
   RSA m^d mod n(4096 bits) - ...
      -> PKCS-1
Old: User ID Packet(tag 13)(25 bytes)
   User ID - Alice <alice@example.org>
Old: Signature Packet(tag 2)(573 bytes)
   Ver 4 - new
   Sig type - Positive certification of a User ID and Public Key packet(0x13).
   Pub alg - RSA Encrypt or Sign(pub 1)
   Hash alg - SHA512(hash 10)
   Hashed Sub: signature creation time(sub 2)(4 bytes)
      Time - Wed Nov 30 22:44:28 CET 2016
   Hashed Sub: key flags(sub 27)(1 bytes)
      Flag - This key may be used to certify other keys
   Hashed Sub: key expiration time(sub 9)(4 bytes)
      Time - Fri Nov 30 22:44:14 CET 2018
   Hashed Sub: preferred symmetric algorithms(sub 11)(4 bytes)
      Sym alg - AES with 256-bit key(sym 9)
      Sym alg - AES with 192-bit key(sym 8)
      Sym alg - AES with 128-bit key(sym 7)
      Sym alg - CAST5(sym 3)
   Hashed Sub: preferred hash algorithms(sub 21)(4 bytes)
      Hash alg - SHA512(hash 10)
      Hash alg - SHA384(hash 9)
      Hash alg - SHA256(hash 8)
      Hash alg - SHA224(hash 11)
   Hashed Sub: preferred compression algorithms(sub 22)(4 bytes)
      Comp alg - ZLIB <RFC1950>(comp 2)
      Comp alg - BZip2(comp 3)
      Comp alg - ZIP <RFC1951>(comp 1)
      Comp alg - Uncompressed(comp 0)
   Hashed Sub: features(sub 30)(1 bytes)
      Flag - Modification detection (packets 18 and 19)
   Hashed Sub: key server preferences(sub 23)(1 bytes)
      Flag - No-modify
   Sub: issuer key ID(sub 16)(8 bytes)
      Key ID - 0xD93D03C13478D580
   Hash left 2 bytes - e5 1a
```

```
    RSA m^d mod n(4096 bits) - ...
       -> PKCS-1
```

Following the master key and the identities, the secret keys can be found. The first is the signing key with a size of 3072 bits. The signing subkey has two signatures, one from the master key and one signature from itself (self signature). The signature itself can be identified by the “issuer key ID”.

```
Old: Secret Subkey Packet(tag 7)(1414 bytes)
   Ver 4 - new
   Public key creation time - Wed Nov 30 22:44:21 CET 2016
   Pub alg - RSA Encrypt or Sign(pub 1)
   RSA n(3072 bits) - ...
   RSA e(17 bits) - ...
   Sym alg - AES with 256-bit key(sym 9)
   Iterated and salted string-to-key(s2k 3):
      Hash alg - SHA512(hash 10)
      Salt - 0f 7d 79 55 7c 4e 99 71
      Count - 1015808(coded count 159)
   IV - f6 1f ba 7c d4 40 28 19 72 c5 12 0f 4f c1 a3 65
   Encrypted RSA d
   Encrypted RSA p
   Encrypted RSA q
   Encrypted RSA u
   Encrypted SHA1 hash
Old: Signature Packet(tag 2)(964 bytes)
   Ver 4 - new
   Sig type - Subkey Binding Signature(0x18).
   Pub alg - RSA Encrypt or Sign(pub 1)
   Hash alg - SHA512(hash 10)
   Hashed Sub: signature creation time(sub 2)(4 bytes)
      Time - Wed Nov 30 22:44:21 CET 2016
   Hashed Sub: key flags(sub 27)(1 bytes)
      Flag - This key may be used to sign data
   Hashed Sub: key expiration time(sub 9)(4 bytes)
      Time - Fri Nov 30 22:44:21 CET 2018
   Sub: issuer key ID(sub 16)(8 bytes)
      Key ID - 0xD93D03C13478D580
   Sub: embedded signature(sub 32)(412 bytes)
   Ver 4 - new
   Sig type - Primary Key Binding Signature(0x19).
   Pub alg - RSA Encrypt or Sign(pub 1)
   Hash alg - SHA512(hash 10)
   Hashed Sub: signature creation time(sub 2)(4 bytes)
      Time - Wed Nov 30 22:44:21 CET 2016
   Sub: issuer key ID(sub 16)(8 bytes)
      Key ID - 0x1ED73636975EC6DE
   Hash left 2 bytes - 2c da
   RSA m^d mod n(3071 bits) - ...
      -> PKCS-1
   Hash left 2 bytes - c0 7e
   RSA m^d mod n(4095 bits) - ...
      -> PKCS-1
```

The encryption key as well as the authentication key only show one signature from the master key. As their capabilities do not include signing, they are not signed by them self.

```
Old: Secret Subkey Packet(tag 7)(1414 bytes)
   Ver 4 - new
   Public key creation time - Wed Nov 30 22:44:23 CET 2016
   Pub alg - RSA Encrypt or Sign(pub 1)
   RSA n(3072 bits) - ...
   RSA e(17 bits) - ...
   Sym alg - AES with 256-bit key(sym 9)
   Iterated and salted string-to-key(s2k 3):
      Hash alg - SHA512(hash 10)
      Salt - 5c 54 ae 09 9d d2 01 c6
```

```
      Count - 1015808(coded count 159)
   IV - 6d 44 b1 68 d4 c5 94 41 c0 23 ea 92 bc 50 ce 68
   Encrypted RSA d
   Encrypted RSA p
   Encrypted RSA q
   Encrypted RSA u
   Encrypted SHA1 hash
Old: Signature Packet(tag 2)(549 bytes)
   Ver 4 - new
   Sig type - Subkey Binding Signature(0x18).
   Pub alg - RSA Encrypt or Sign(pub 1)
   Hash alg - SHA512(hash 10)
   Hashed Sub: signature creation time(sub 2)(4 bytes)
      Time - Wed Nov 30 22:44:23 CET 2016
   Hashed Sub: key flags(sub 27)(1 bytes)
      Flag - This key may be used to encrypt communications
      Flag - This key may be used to encrypt storage
   Hashed Sub: key expiration time(sub 9)(4 bytes)
      Time - Fri Nov 30 22:44:23 CET 2018
   Sub: issuer key ID(sub 16)(8 bytes)
      Key ID - 0xD93D03C13478D580
   Hash left 2 bytes - fb 43
   RSA m^d mod n(4096 bits) - ...
      -> PKCS-1
Old: Secret Subkey Packet(tag 7)(1414 bytes)
   Ver 4 - new
   Public key creation time - Wed Nov 30 22:44:26 CET 2016
   Pub alg - RSA Encrypt or Sign(pub 1)
   RSA n(3072 bits) - ...
   RSA e(17 bits) - ...
   Sym alg - AES with 256-bit key(sym 9)
   Iterated and salted string-to-key(s2k 3):
      Hash alg - SHA512(hash 10)
      Salt - a5 7a f7 44 d2 b7 3c 2d
      Count - 1015808(coded count 159)
   IV - 25 3c aa 37 c9 8e 16 eb 12 b6 7d 04 36 a3 db 92
   Encrypted RSA d
   Encrypted RSA p
   Encrypted RSA q
   Encrypted RSA u
   Encrypted SHA1 hash
Old: Signature Packet(tag 2)(549 bytes)
   Ver 4 - new
   Sig type - Subkey Binding Signature(0x18).
   Pub alg - RSA Encrypt or Sign(pub 1)
   Hash alg - SHA512(hash 10)
   Hashed Sub: signature creation time(sub 2)(4 bytes)
      Time - Wed Nov 30 22:44:26 CET 2016
   Hashed Sub: key flags(sub 27)(1 bytes)
      Flag - This key may be used for authentication
   Hashed Sub: key expiration time(sub 9)(4 bytes)
      Time - Fri Nov 30 22:44:26 CET 2018
   Sub: issuer key ID(sub 16)(8 bytes)
      Key ID - 0xD93D03C13478D580
   Hash left 2 bytes - 9d 8b
   RSA m^d mod n(4095 bits) - ...
      -> PKCS-1
```

Even if it is not as comfortable to use gpgdump as it is to verify the key with hopenpgp-tools, there are more details that might be of interest. Additionally, for RedHat / CentOS and other rpm based distributions it offers a good alternative to the hopenpgp-tools.

Read more of my posts on my blog at https://blog.tinned-software.net/.

Related Posts:

1. **Restart hanging gpg-agent automatically using swatch on MacOSX**
2. **Create a luks encrypted partition on Linux Mint**
3. **Automount a luks encrypted volume on system start**
4. **apt – install the package that contains a specific file**
5. **Increase the size of a LUKS encrypted partition**
6. **SSH passwordless login with SSH-key**
7. **Set up apache and the log path with SELinux**
8. **Generate public SSH key from private SSH key**

This entry was posted in Encryption, Security and tagged authentication, encryption, gnupg, gpg, pgp, signing, subkeys. Bookmark the permalink.

Experiencing Technology