Security Architecture TP2

Davide Caria
Tee Wei Teoh

16 October 2023

1 Getting to GPG

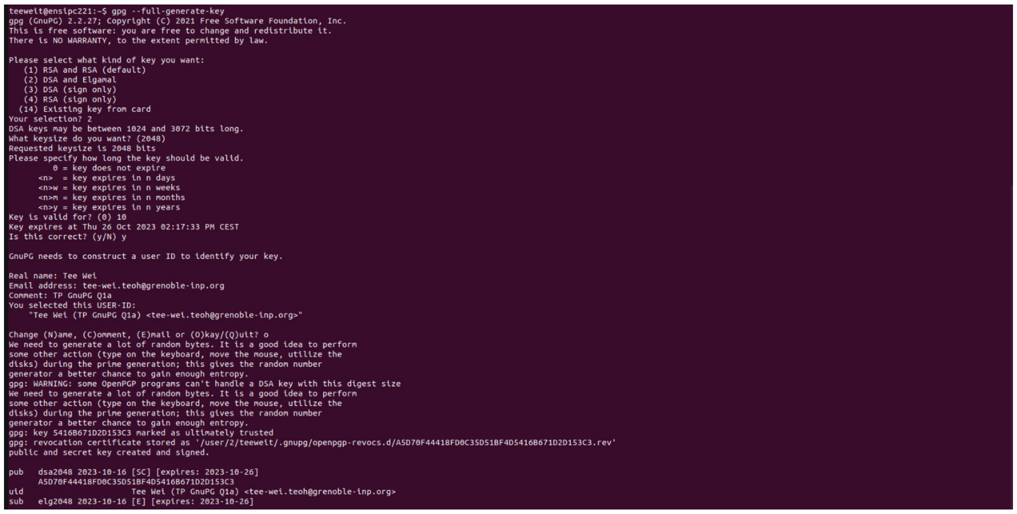
1.a Key creation

1.a.1

The creation of the key has been perfomed using the following command:

```
gpg --full-generate-key
```

since it provided a menu with the list of options required to create a key. The other proposed command only asks for the minimum information that can later be changed. The creation process of the key is detailed here after.



```
teeweit@ensipc221:~$ gpg --full-generate-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 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)
  (14) Existing key from card
Your selection? 2
DSA keys may be between 1024 and 3072 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) 10
Key expires at Thu 26 Oct 2023 02:17:33 PM CEST
Is this correct? (y/N) y

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

Real name: Tee Wei
Email address: tee-wei.teoh@grenoble-inp.org
Comment: TP GnuPG Q1a
You selected this USER-ID:
    "Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
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: WARNING: some OpenPGP programs can't handle a DSA key with this digest size
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 5416B671D2D153C3 marked as ultimately trusted
gpg: revocation certificate stored as '/user/2/teeweit/.gnupg/openpgp-revocs.d/A5D70F44418FD0C35D51BF4D5416B671D2D153C3.rev'
public and secret key created and signed.

pub   dsa2048 2023-10-16 [SC] [expires: 2023-10-26]
      A5D70F44418FD0C35D51BF4D5416B671D2D153C3
uid                      Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
sub   elg2048 2023-10-16 [E] [expires: 2023-10-26]
```

Among the different choices that can be made there is the *type of key* that can be created (choose among RSA, DSA, ElGamal, etc...). Then there is the possibility to choose the *key size* or to leave it has default. Another parameter in the wizard is the *validity period* which can be tuned according to the needs. Then there is the option to add name, surname and email address associated to it.

1.a.2

To revoke the previously created key we can use a revocation certificate that can be created providing the $KEY\ ID$ to the following command:

```
gpg --gen-revoke A5D70F44418FD0C35D51BF4D5416B671D2D153C3
```

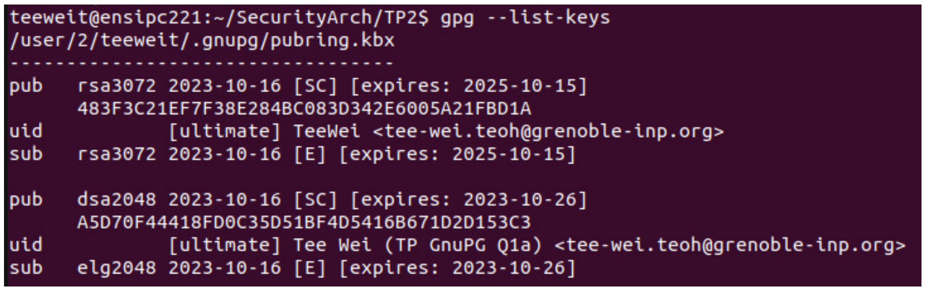
A revocation certificate serves as a means to invalidate a public key and declare it as no longer valid or trustworthy. After having pushed this certificate to the server, the key that it refers to is not valid anymore.

1.a.3

After running the command:

```
gpg --gen-revoke A5D70F44418FD0C35D51BF4D5416B671D2D153C3
```

the list of keys in the key-ring is the following:



```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --list-keys
/user/2/teeweit/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2023-10-16 [SC] [expires: 2025-10-15]
      483F3C21EF7F38E284BC083D342E6005A21FBD1A
uid           [ultimate] TeeWei <tee-wei.teoh@grenoble-inp.org>
sub   rsa3072 2023-10-16 [E] [expires: 2025-10-15]

pub   dsa2048 2023-10-16 [SC] [expires: 2023-10-26]
      A5D70F44418FD0C35D51BF4D5416B671D2D153C3
uid           [ultimate] Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
sub   elg2048 2023-10-16 [E] [expires: 2023-10-26]
```

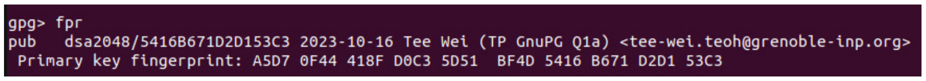
1.a.4

The following points refer to the edition menu (gpg –edit-key)

4.a The *fingerpring* of the key is given by the following command:

```
fpr
```

and in our case it gives the following output:



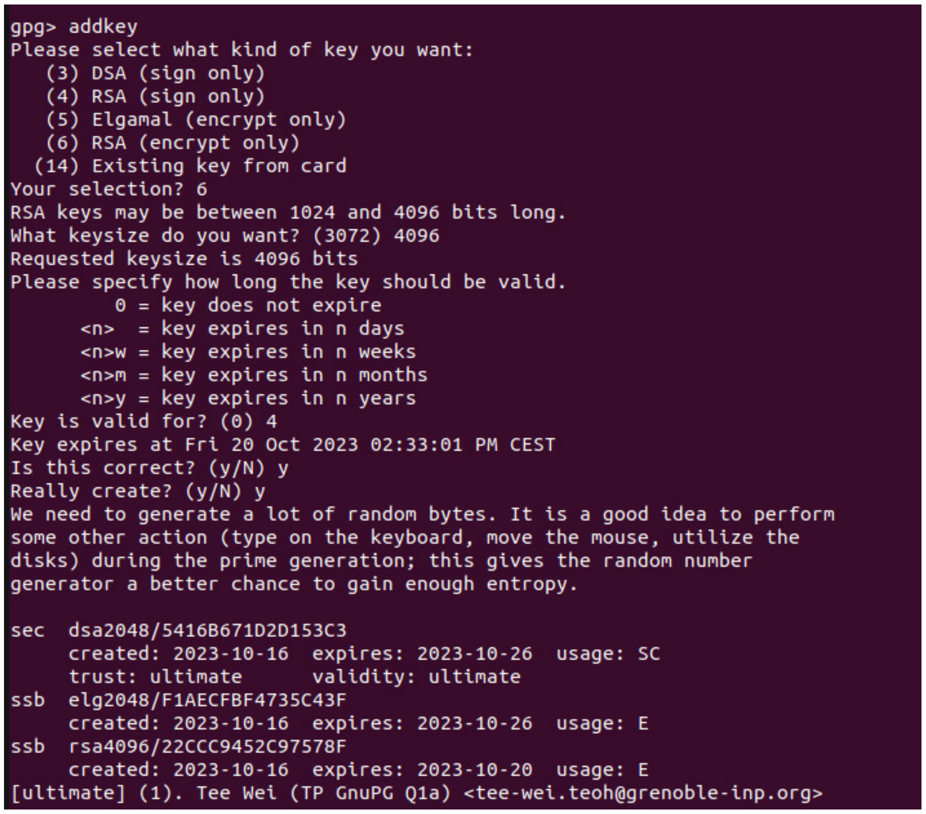
```
gpg> fpr
pub   dsa2048/5416B671D2D153C3 2023-10-16 Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
 Primary key fingerprint: A5D7 0F44 418F D0C3 5D51  BF4D 5416 B671 D2D1 53C3
```

In this context, the fingerprint has to be intepreted as a unique identifier that represents the key and allows for quick and reliable verification of its integrity. It is represend as a sequence of 40 hexadecimal character since it is a cryptograpghic hash generated from the key's public key.

4.b To add an RSA 4096 encryption key we can run the command:

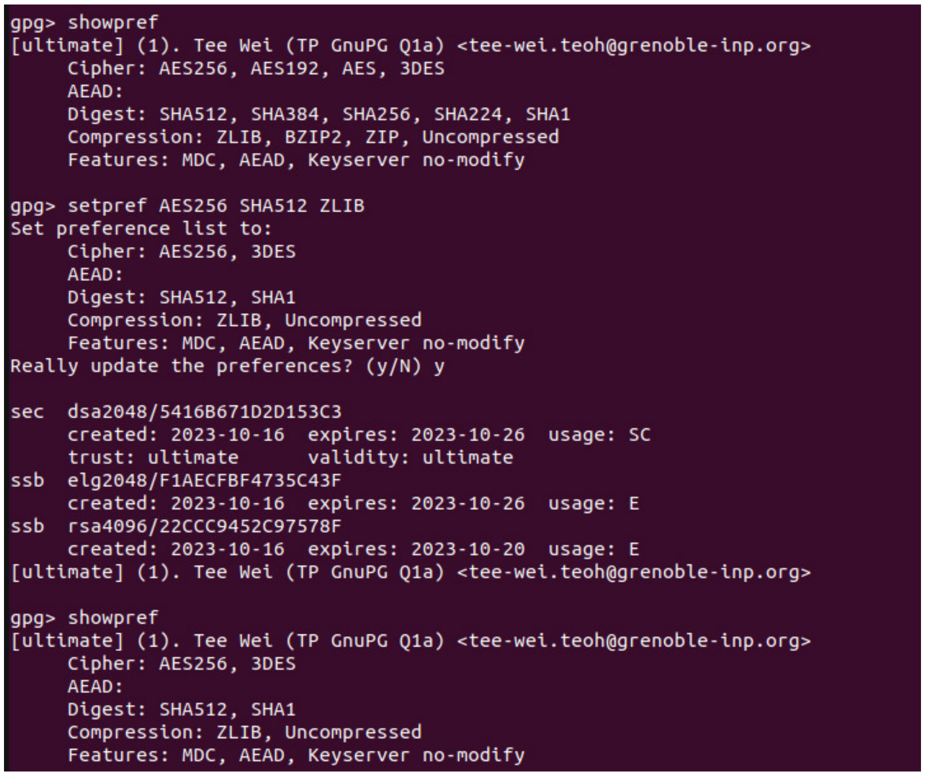
```
addkey
```

and select the proper key to be added. In this case it is *RSA* (*encrypt only*). Then it is mandatory to select the size (4096 in this case) and the validity period (4 days for the exercise). Hereafter it is reported the output of the command:



```
gpg> addkey
Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
  (14) Existing key from card
Your selection? 6
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 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) 4
Key expires at Fri 20 Oct 2023 02:33:01 PM CEST
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.

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  rsa4096/22CCC9452C97578F
     created: 2023-10-16  expires: 2023-10-20  usage: E
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
```

4.c In order to change the required references we have to move into the *key editing mode* and change the parameters according to our needs. To do so we can use the *setpref* sub-command and modify the hash algorithms, compression methods and symmetric encryption algorithms. The following is the series of output that we have got once we run the commands.



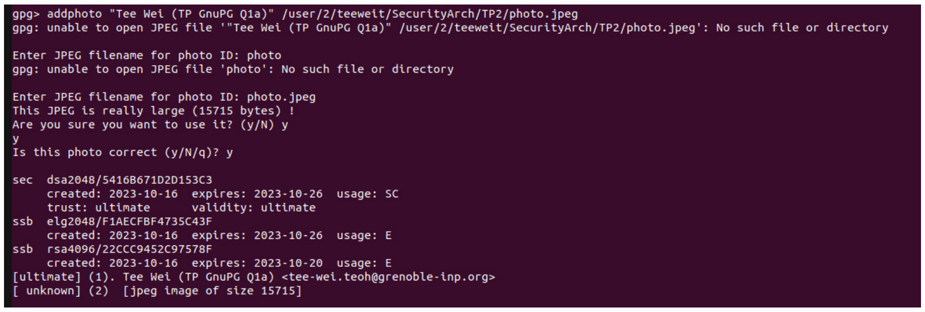
```
gpg> showpref
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
     Cipher: AES256, AES192, AES, 3DES
     AEAD:
     Digest: SHA512, SHA384, SHA256, SHA224, SHA1
     Compression: ZLIB, BZIP2, ZIP, Uncompressed
     Features: MDC, AEAD, Keyserver no-modify

gpg> setpref AES256 SHA512 ZLIB
Set preference list to:
     Cipher: AES256, 3DES
     AEAD:
     Digest: SHA512, SHA1
     Compression: ZLIB, Uncompressed
     Features: MDC, AEAD, Keyserver no-modify
Really update the preferences? (y/N) y

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  rsa4096/22CCC9452C97578F
     created: 2023-10-16  expires: 2023-10-20  usage: E
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>

gpg> showpref
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
     Cipher: AES256, 3DES
     AEAD:
     Digest: SHA512, SHA1
     Compression: ZLIB, Uncompressed
     Features: MDC, AEAD, Keyserver no-modify
```

The first command is *showpref* and it is used to show which are the default preferences for the key.

4.d In order to add an image we chose a random one from internet and added it to the key. The following command can be run:

```
addphoto "Tee Wei (TP GnUPG 01a)" /user /2/teeweit/SecurityArch/TP2/photo. jpeg
```

The following is the command output:

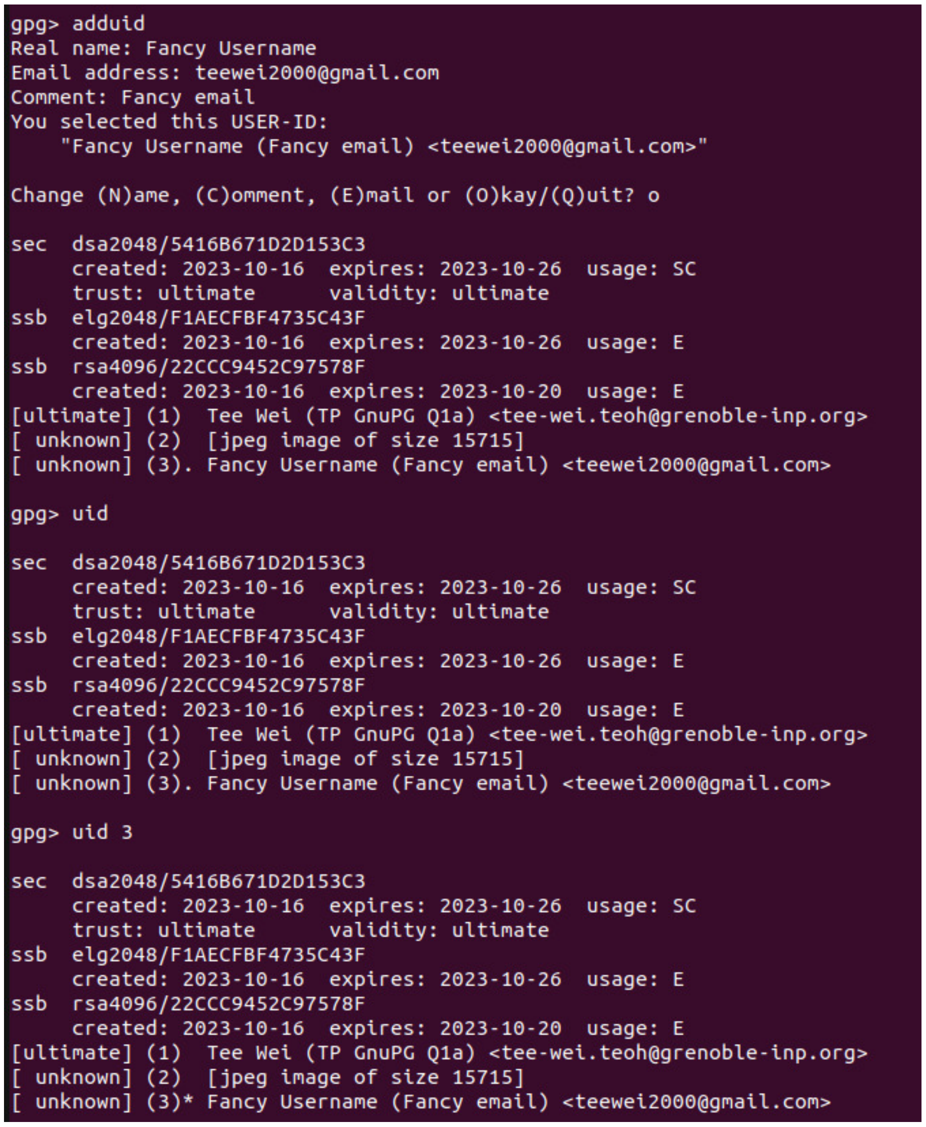


```
gpg> addphoto "Tee Wei (TP GnuPG Q1a)" /user/2/teeweit/SecurityArch/TP2/photo.jpeg
gpg: unable to open JPEG file '"Tee Wei (TP GnuPG Q1a)" /user/2/teeweit/SecurityArch/TP2/photo.jpeg': No such file or directory

Enter JPEG filename for photo ID: photo
gpg: unable to open JPEG file 'photo': No such file or directory

Enter JPEG filename for photo ID: photo.jpeg
This JPEG is really large (15715 bytes) !
Are you sure you want to use it? (y/N) y
y
Is this photo correct (y/N/q)? y

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  rsa4096/22CCC9452C97578F
     created: 2023-10-16  expires: 2023-10-20  usage: E
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
[ unknown] (2)  [jpeg image of size 15715]
```

4.e To add a username and make it the main one we can run the following commands:

```
adduid
uid
uid 3 //for our specific example
```

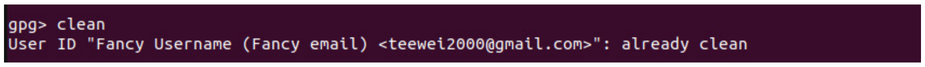
They are used to add a new user id with fake id and fake e-mail address and to make them as main user name.



```
gpg> adduid
Real name: Fancy Username
Email address: teewei2000@gmail.com
Comment: Fancy email
You selected this USER-ID:
    "Fancy Username (Fancy email) <teewei2000@gmail.com>"

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

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  rsa4096/22CCC9452C97578F
     created: 2023-10-16  expires: 2023-10-20  usage: E
[ultimate] (1)  Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
[ unknown] (2)  [jpeg image of size 15715]
[ unknown] (3). Fancy Username (Fancy email) <teewei2000@gmail.com>

gpg> uid

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  rsa4096/22CCC9452C97578F
     created: 2023-10-16  expires: 2023-10-20  usage: E
[ultimate] (1)  Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
[ unknown] (2)  [jpeg image of size 15715]
[ unknown] (3). Fancy Username (Fancy email) <teewei2000@gmail.com>

gpg> uid 3

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  rsa4096/22CCC9452C97578F
     created: 2023-10-16  expires: 2023-10-20  usage: E
[ultimate] (1)  Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
[ unknown] (2)  [jpeg image of size 15715]
[ unknown] (3)* Fancy Username (Fancy email) <teewei2000@gmail.com>
```

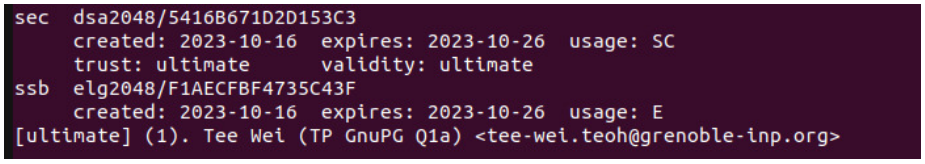
1.a.5

To put back the key in the clean state we can do the following:



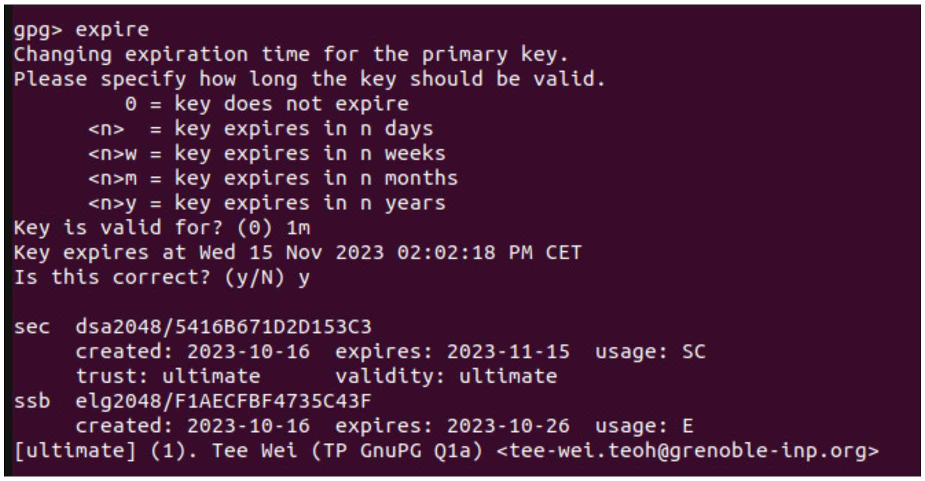
```
gpg> clean
User ID "Fancy Username (Fancy email) <teewei2000@gmail.com>": already clean
```

to see the effect from another point of view we can run the command *list* and see that the image and the fancy name are gone.



```
sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
```

After cleaning the key we can change its expiration with the command *expire* and have the following output:



```
gpg> expire
Changing expiration time for the primary key.
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) 1m
Key expires at Wed 15 Nov 2023 02:02:18 PM CET
Is this correct? (y/N) y

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-11-15  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
```

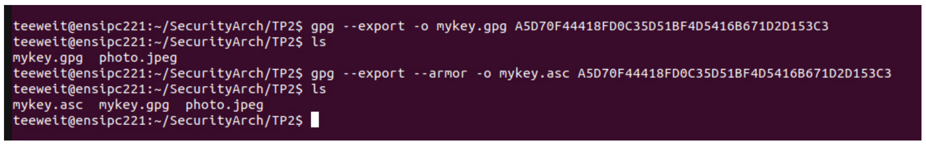
Finally we add a signature sub-key with the *addkey* command (the output is the same as in point 4.b)

1.a.6

To export out key both in binary and text we can use the following:

```
gpg --export -o mykey.gpg <key-id> //for binary format
gpg --export --armor -o mykey.asc <key-id> //for text format
```

we got the following result:



```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --export -o mykey.gpg A5D70F44418FD0C35D51BF4D5416B671D2D153C3
teeweit@ensipc221:~/SecurityArch/TP2$ ls
mykey.gpg  photo.jpeg
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --export --armor -o mykey.asc A5D70F44418FD0C35D51BF4D5416B671D2D153C3
teeweit@ensipc221:~/SecurityArch/TP2$ ls
mykey.asc  mykey.gpg  photo.jpeg
teeweit@ensipc221:~/SecurityArch/TP2$ 
```

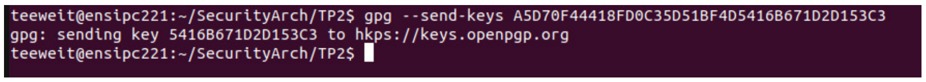
Now the keys are saved as *.asc* and *.gpg* file.

1.a.7

To send the key to a public server we used the command:

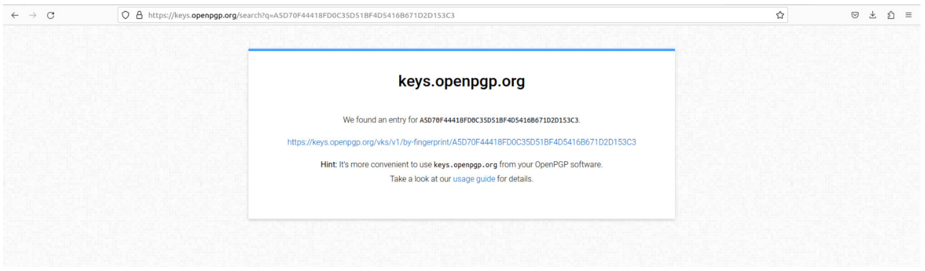
```
gpg --send -keys //add the KeyId
```

and by default the target server was: $hkps://keys.openpgp.org$



```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --send-keys A5D70F44418FD0C35D51BF4D5416B671D2D153C3
gpg: sending key 5416B671D2D153C3 to hkps://keys.openpgp.org
teeweit@ensipc221:~/SecurityArch/TP2$ 
```

To verify that everything went well we checked the existence of the key in the public server:



https://keys.openpgp.org/search?q=A5D70F44418FD0C35D51BF4D5416B671D2D153C3

keys.openpgp.org

We found an entry for A5D70F44418FD0C35D51BF4D5416B671D2D153C3.

https://keys.openpgp.org/vks/v1/by-fingerprint/A5D70F44418FD0C35D51BF4D5416B671D2D153C3

Hint: It's more convenient to use keys.openpgp.org from your OpenPGP software.
Take a look at our usage guide for details.

1.b Signature

1.b.1

Before organizing the *keysigning pary* we defined our signing policy taking inspiration from online formats:

```
#Key Signing Policy for [Tee Wei or tee-wei.teoh@grenoble-inp.org]

## Verification Methods
- I verify identities in person
- For online verification, I use encrypted email exchanges
    with already trusted correspondents.

## Trust Levels
```

```
- I assign "full" trust only to keys of individuals I've met
    and verified.
- I assign "marginal" trust to keys of individuals I've verified
    remotely through trusted channels.
- I assign "never" trust to keys of individuals I've never met.

## Expiry Dates
- I set an expiry date of one month on signatures I make on keys.

## Contact Information
- My email : tee-wei.teoh@grenoble-inp.org
```

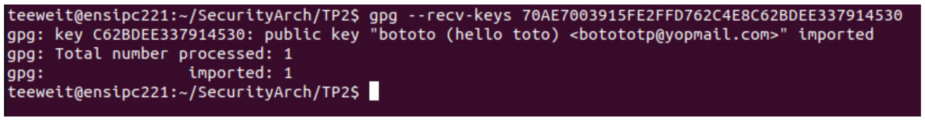
After creating it we can pulish it on our website, in our GPG key's metadata, or on our key server. Also, we must follow the rules stated in our key signing policy when signing others' keys, also making sure others to follow your key signing policy when signing your key. Since we can pretend to be on a *keysigning pary* we could analyze other's key singing policy to improve it.

1.b.2

We got 3 keys from our classmates and for each one of them we performed the following:
a. Got the key through a trusted channel
b. Asked for their ID and student card to verify their identity
c. Signed the key with our personal one
d. Exported the signed key and sent it back to the owner
To perform the previous steps we used the following set of commands:

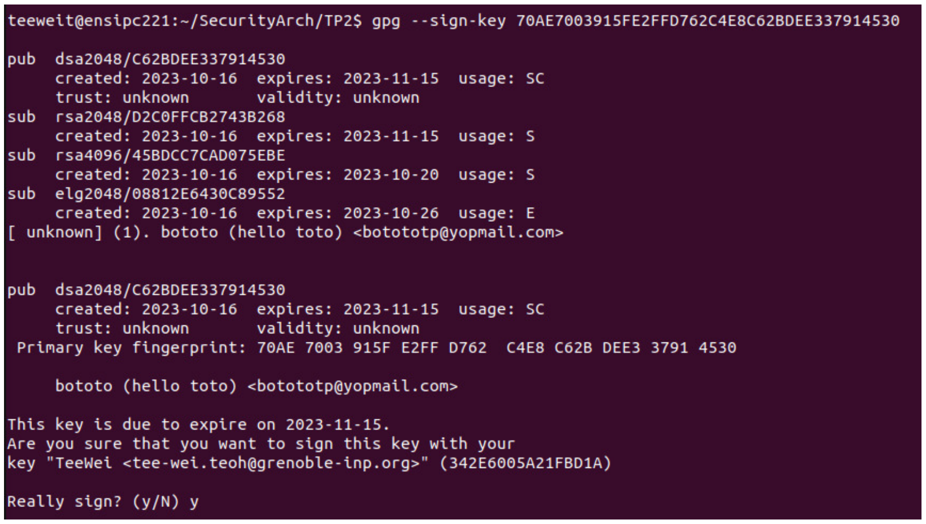
```
gpg --recv-keys KEY-ID //to retrive the key from the repository
gpg --sign-key KEY-ID //to sign the key with our
```

As an example we show only one signing process. First we retrieve the key:



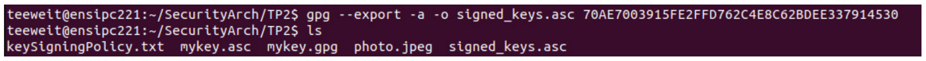
```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --recv-keys 70AE7003915FE2FFD762C4E8C62BDEE337914530
gpg: key C62BDEE337914530: public key "bototo (hello toto) <botototp@yopmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
teeweit@ensipc221:~/SecurityArch/TP2$ 
```

Then we can sign the key:



```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --sign-key 70AE7003915FE2FFD762C4E8C62BDEE337914530

pub  dsa2048/C62BDEE337914530
     created: 2023-10-16  expires: 2023-11-15  usage: SC
     trust: unknown       validity: unknown
sub  rsa2048/D2C0FFCB2743B268
     created: 2023-10-16  expires: 2023-11-15  usage: S
sub  rsa4096/45BDCC7CAD075EBE
     created: 2023-10-16  expires: 2023-10-20  usage: S
sub  elg2048/08812E6430C89552
     created: 2023-10-16  expires: 2023-10-26  usage: E
[ unknown] (1). bototo (hello toto) <bototot p@yopmail.com>


pub  dsa2048/C62BDEE337914530
     created: 2023-10-16  expires: 2023-11-15  usage: SC
     trust: unknown       validity: unknown
 Primary key fingerprint: 70AE 7003 915F E2FF D762  C4E8 C62B DEE3 3791 4530

     bototo (hello toto) <bototot p@yopmail.com>

This key is due to expire on 2023-11-15.
Are you sure that you want to sign this key with your
key "TeeWei <tee-wei.teoh@grenoble-inp.org>" (342E6005A21FBD1A)

Really sign? (y/N) y
```

Finally we can export the key:



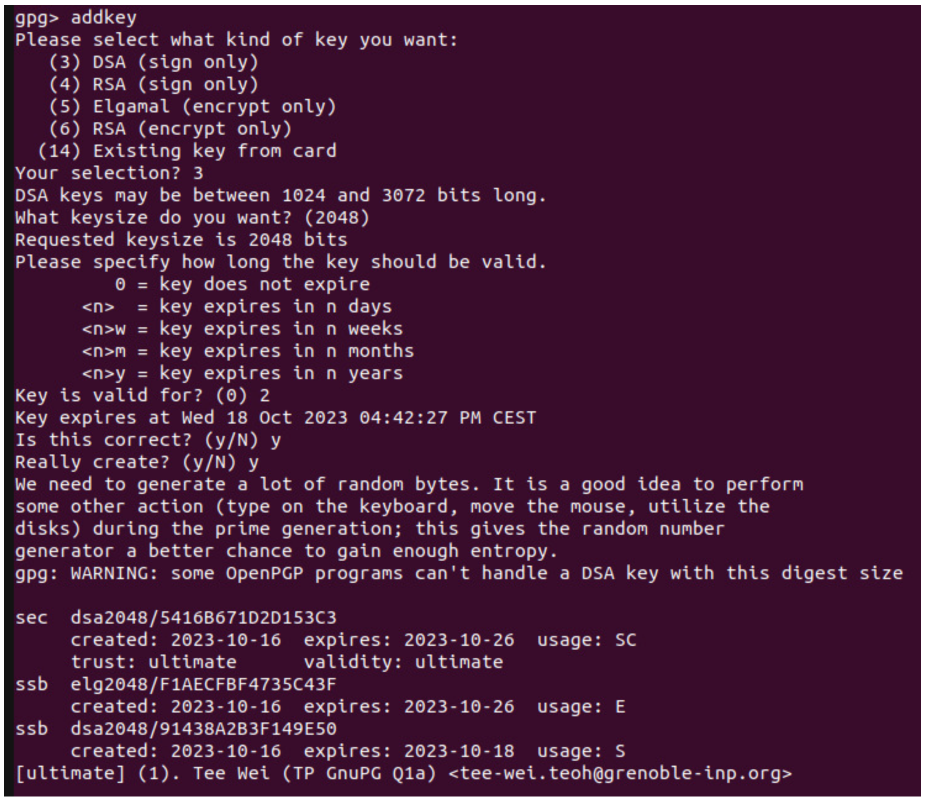
```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --export -a -o signed_keys.asc 70AE7003915FE2FFD762C4E8C62BDEE337914530
teeweit@ensipc221:~/SecurityArch/TP2$ ls
keySigningPolicy.txt  mykey.asc  mykey.gpg  photo.jpeg  signed_keys.asc
```

1.b.3

For this section, we signed a message along with one of our classmates from which we previously retrieved the key. Another student, outside of our group of keys, has received the message and verified it according to what he has found on the public repository.

1.b.4

To add a DSA key we used the previously issued command with different settings (2 days of expiration).



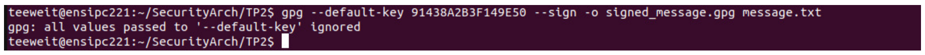
```
gpg> addkey
Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
  (14) Existing key from card
Your selection? 3
DSA keys may be between 1024 and 3072 bits long.
What keysize do you want? (2048)
Requested keysize 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) 2
Key expires at Wed 18 Oct 2023 04:42:27 PM CEST
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.
gpg: WARNING: some OpenPGP programs can't handle a DSA key with this digest size

sec  dsa2048/5416B671D2D153C3
     created: 2023-10-16  expires: 2023-10-26  usage: SC
     trust: ultimate      validity: ultimate
ssb  elg2048/F1AECFBF4735C43F
     created: 2023-10-16  expires: 2023-10-26  usage: E
ssb  dsa2048/91438A2B3F149E50
     created: 2023-10-16  expires: 2023-10-18  usage: S
[ultimate] (1). Tee Wei (TP GnuPG Q1a) <tee-wei.teoh@grenoble-inp.org>
```

1.b.5

To sign a message with the previously defined sub-key we can issue the command:

```
gpg --sing message.txt
```

along with that we have to specify the option $-o$ *singed_message.gpg*. The full command is the following:



```
teeweit@ensipc221:~/SecurityArch/TP2$ gpg --default-key 91438A2B3F149E50 --sign -o signed_message.gpg message.txt
gpg: all values passed to '--default-key' ignored
teeweit@ensipc221:~/SecurityArch/TP2$ 
```

After signing the message we can verify the signature that we have applied with the command:

```
gpg --list-packets signed_message.gpg
```

1.c Encryption

1.c.1

For this purpose, we decided to download a free pdf version of a famous child book and have it signed with the key of another student. We used the public key retrieved from one the previous exchange. The ciphered version is larger that the baseline since we are using RSA encryption for large files. Padding and type of algorithm have to be taken into account to explain why the output file is larger than the input one.

1.c.2

We have attached **Point 5** to the email