Gnu Privacy Guard (gpg)

Serveur Linux CentOS

Table des matières

1 Introduction

Les échanges d'information ne sont pas sûrs. En particulier, le courrier électronique peut être lu par des tierces personnes, et on n'est jamais sûr de la provenance réelle d'un émail.

Il est possible d'utiliser des procédés cryptographiques. Il existe des méthodes à clés asymétriques, permettant soit de signer (authentifier) un texte, soit de le chiffrer (le rendre illisible au monde entier).

GnuPG est une des implémentations utilisant un tel système. Il a l'énorme avantage en comparaison avec son équivalent propriétaire PGP d'être un logiciel libre et de reposer sur la norme OpenPGP. De plus, le créateur de PGP, Philip Zimmermann, a rejoint récemment le groupe OpenPGP.

Cette documentation va présenter l'utilisation de GnuPG, ainsi que toute la philosophie qui tourne autour du système de chiffrement à clé publique.

2 Configuration

2.1 Création des comptes utilisateurs

Les deux personnes qui s'échangent des messages sont Romeo et Juliet. On va créer les deux comptes utilisateurs :

```
[root@localhost ~]# useradd romeo
```

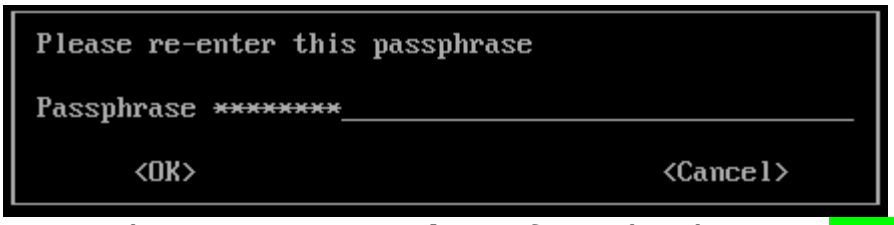
```
[root@localhost ~]# useradd juliet
```

2.2 Création des clés

Il faut créer une paire de clés pour chaque utilisateur. Les seront utilisées pour chiffrer/déchiffrer et signer/vérifier la signature des messages.

Création de la clé de Romeo

```
[root@localhost ~]# chmod o+rw $(tty)
[root@localhost ~]# su - romeo
[romeo@localhost ~]$ gpg --gen-key
gpg (GnuPG) 2.0.22; 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?
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)
Key does not expire at all
Is this correct? (y/N) y
GnuPG needs to construct a user ID to identify your
key.
Real name: Romeo Romeo
Email address: romeo@hotmail.com
Comment: Romeo
You selected this USER-ID:
    "Romeo Romeo (Romeo) <romeo@hotmail.com>"
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 disks) during the
prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key AFCF8C34 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
pub   2048R/AFCF8C34 2019-06-11
Key fingerprint = FF1E 2767 12F9 956F A591  694A 1903
139A AFCF 8C34
uid Romeo Romeo (Romeo) <romeo@hotmail.com>
sub   2048R/EA17EB64 2019-06-11
```

Ouvrir une nouvelle session et lancer la commande suivante pour générer de l'activité sur le serveur :

```
[root@localhost ~]# dd if=/dev/sda2 of=/dev/null
```

Création de la clé de Juliet

```
[root@localhost ~]# chmod o+rw $(tty)
[root@localhost ~]# su - juliet
[juliet@localhost ~]$ gpg --gen-key
```

- Deux clés ont été générées. L'une d'elle est la clé publique, que l'on va pouvoir donner à tout le monde. L'autre est la clé secrète, qu'on doit absolument être le seul à conserver. Ces clés sont enregistrées dans le répertoire par défaut qui est le répertoire ***.gnupg*** dans le répertoire personnel de l'utilisateur.

  Lorsqu'on regarde à l'intérieur de ce répertoire, on remarque en effet qu'il y a en particulier deux fichiers, *pubring.gpg* et *secring.gpg*.

- Les clés publique et secrète sont étroitement liées. L'une sert à défaire ce que l'autre a fait :

  Lorsqu'on chiffre un message avec la clé publique, n'importe qui avec la clé secrète peut déchiffrer ce résumé pour vérifier s'il correspond bien au message. Vu que personne ne doit connaître la clé secrète, si le message est valide, c'est qu'il a été signé par la bonne personne.

- Si quelqu'un d'autre a accès à notre clé secrète, il pourra envoyer des messages en se faisant passer pour nous, ou bien il pourra lire des messages chiffrés à notre intention. C'est à ce moment qu'intervient le concept de mot de passe : pour améliorer encore plus la sécurité, les programmes demanderont le mot de passe pour pouvoir utiliser la clé secrète.

2.3 Exporter une clé

Une fois que la création de la clé est complétée, on peut la diffuser. L'usage veut que les clés publiques soient envoyées sur des serveurs de clés publique. Ainsi, lorsqu'on reçoit un message signé d'une personne dont on n'a pas la clé dans notre porte-clés, il est possible d'aller chercher la clé publique sur un serveur de clés afin de vérifier l'authenticité du message. De la même façon, si on veut envoyer un message chiffré à une personne, on peut récupérer sa clé publique afin de l'utiliser pour chiffrer le message.

La commande suivante permet d'exporter la clé dans le fichier **cle_pub_romeo**:

Romeo :

```
[romeo@localhost ~]$ gpg -o cle_pub_romeo --export
```

Transférer la clé à **juliet** :

```
[root@localhost ~]# cp /home/romeo/cle_pub_romeo   /home/juliet
```

Juliet :

```
[juliet@localhost ~]$ gpg -o cle_pub_juliet --export juliet
```

Transférer la clé à **romeo** :

```
[root@localhost ~]# cp /home/juliet/cle_pub_juliet   /home/romeo
```

2.4 Importer une clé

La personne qui reçoit le fichier peut l'importer dans son porte-clés avec la commande suivante:

Romeo :

```
[romeo@localhost ~]$ gpg --import cle_pub_juliet
gpg: key ECE7A52A: public key "Juliet (Juliet Juliet)
<juliet@hotmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

```
[romeo@localhost ~]$ gpg --list-keys
/home/romeo/.gnupg/pubring.gpg
------------------------------
pub   1024D/F3AC83A8 2019-06-11
uid                  Romeo (Romeo Romeo)
<romeo@hotmail.com>
sub   2048g/A614A959 2019-06-11

pub   1024D/ECE7A52A 2019-06-11
uid                  Juliet (Juliet Juliet)
<juliet@hotmail.com>
sub   2048g/B183A845 2019-06-11
```

Juliet :

```
[juliet@localhost ~]$ gpg --import cle_pub_romeo
gpg: key F3AC83A8: public key "Romeo (Romeo Romeo)
<romeo@hotmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

```
[juliet@localhost ~]$ gpg --list-keys
/home/juliet/.gnupg/pubring.gpg
-------------------------------
pub   1024D/ECE7A52A 2019-06-11
uid                  Juliet (Juliet Juliet)
<juliet@hotmail.com>
sub   2048g/B183A845 2019-06-11

pub   1024D/F3AC83A8 2019-06-11
uid                  Romeo (Romeo Romeo)
<romeo@hotmail.com>
sub   2048g/A614A959 2019-06-11
```

2.5 Encrypter un message

```
[romeo@localhost ~]$ echo 'message secret pour Juliet'
> message.txt
```

```
[romeo@localhost ~]$ gpg -r juliet -e message.txt
gpg: B183A845: There is no assurance this key belongs
to the named user

pub  2048g/B183A845 2019-06-11 Juliet (Juliet Juliet)
<juliet@hotmail.com>
 Primary key fingerprint: 4092 7689 B1E6 6428 BEA3
D842 C3CD ECE7 A52A
     Subkey fingerprint: 71F3 CAF2 7E87 F9C0 A5F2
7E92 5183 0 B183 A845

It is NOT certain that the key belongs to the person
named
in the user ID.  If you *really* know what you are
doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

Le fichier **message.txt.gpg** est créé.

```
[root@localhost ~]# cp /home/romeo/message.txt.gpg    /home/juliet
```

2.6 Décrypter un message

```
[juliet@localhost ~]$ gpg -d message.txt.gpg
You need a passphrase to unlock the secret key for
user: "Juliet (Juliet Juliet) <juliet@hotmail.com>"
2048-bit ELG-E key, ID B183A845, created 2019-06-11
(main key ID ECE7A52A)
Enter passphrase: xxxxxx
gpg: encrypted with 2048-bit ELG-E key, ID B183A845,
created 2019-06-11
      "Juliet (Juliet Juliet) <juliet@hotmail.com>"
message secret pour Juliet
```

Pour rediriger le message dans un fichier :

```
[juliet@localhost ~]$ gpg -d message.txt.gpg >
message.txt
You need a passphrase to unlock the secret key for
user: "Juliet (Juliet Juliet) <juliet@hotmail.com>"
2048-bit ELG-E key, ID B183A845, created 2019-06-11
(main key ID ECE7A52A)
Enter passphrase: xxxxxx
gpg: encrypted with 2048-bit ELG-E key, ID B183A845,
created 2019-06-11
      "Juliet (Juliet Juliet) <juliet@hotmail.com>"
```

```
[juliet@localhost ~]$ cat message.txt
message secret pour Juliet
```

2.7 Signature

Pour signer un message :

Romeo :

```
[romeo@localhost ~]$ gpg --clearsign --armor message.txt
You need a passphrase to unlock the secret key for
user: "Romeo (Romeo Romeo) <romeo@hotmail.com>"
1024-bit DSA key, ID F3AC83A8, created 2019-06-11
Enter passphrase:  xxxxxx
```

```
[romeo@localhost ~]$ cat message.txt.asc
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
message secret pour Juliet
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.5 (GNU/Linux)
iD8DBQFE/m5N1GhfY/Osg6gRApQKO3s3yC9AjJ0jGfZL0ZDVSgQTQCeL4Lq
1CKncgns03TDJ/B5Ays6HYI==0+TD
-----END PGP SIGNATURE-----
```

Le fichier **message.txt.asc** est créé.

```
[root@localhost ~]# cp /home/romeo/message.txt.asc  /home/juliet
```

2.8 Vérifier une signature

La commande suivante pour vérifier l'authenticité de la provenance d'un message:

```
[juliet@localhost ~]$ rm message.txt

[juliet@localhost ~]$ gpg message.txt.asc
gpg: Signature made Tue 11 Jun 2019 02:44:29 AM EDT using DSA key
ID F3AC83A8
gpg: Good signature from "Romeo (Romeo Romeo) <romeo@hotmail.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to
the owner.
Primary key fingerprint: 8473 C008 69C5 3341 09D6  01AF D468 5F63
F3AC 83A8
```

2.9 Signer et encrypter un message

```
[romeo@localhost ~]$ gpg --sign --armor --encrypt message.txt

You need a passphrase to unlock the secret key for
user: "Romeo (Romeo Romeo) <romeo@hotmail.com>"
1024-bit DSA key, ID F3AC83A8, created 2019-06-11

Enter passphrase: xxxxxxx

You did not specify a user ID. (you may use "-r")

Current recipients:

Enter the user ID.  End with an empty line: juliet
gpg: B183A845: There is no assurance this key belongs to the named
user

pub  2048g/B183A845 2019-06-11 Juliet (Juliet Juliet)
<juliet@hotmail.com>
 Primary key fingerprint: 4092 7689 B1E6 6428 BEA3  D842  EBCD ECE7
A52A
      Subkey fingerprint: 71F3 CAF2 7E87 F9C0 A5F2  7E92 5183 0806
B183 A845

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y

Current recipients:
2048g/B183A845 2019-06-11 "Juliet (Juliet Juliet)
<juliet@hotmail.com>"

Enter the user ID.  End with an empty line:
```

En plus du mot de passe, il est demandé l'ID pour lequel le message sera chiffré.

Le fichier **message.txt.asc** est créé.

2.10 Certificats de révocation

Il est préférable de créer des certificats de révocation. En effet, il se peut que notre clé ne soit plus sûre, perdue, ou tout simplement périmée. Afin de pouvoir dire au reste du monde que cette clé est compromise, on peut utiliser un certificat qu'on aura créé auparavant (quand on n'avait pas encore perdu la clé par exemple) et que l'on publiera, afin que les personnes soient au courant.

Pour émettre un certificat de révocation, on utilise la commande suivante:

```
[romeo@localhost ~]$ gpg --gen-revoke romeo
sec 1024D/F3AC83A8 2019-06-11 Romeo (Romeo Romeo)
<romeo@hotmail.com>
Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 1
Enter an optional description; end it with an empty line:
> cle perdue
>
Reason for revocation: Key has been compromised
cle perdue
Is this okay? (y/N) y
You need a passphrase to unlock the secret key for
user: "Romeo (Romeo Romeo) <romeo@hotmail.com>"
1024-bit DSA key, ID F3AC83A8, created 2019-06-11
Enter passphrase: xxxxxxxx
ASCII armored output forced.
Revocation certificate created.
Please move it to a medium which you can hide away; if
Mallory gets access to this certificate he can use it to make
your key unusable.
It is smart to print this certificate and store it away, just
in case your media become unreadable.  But have some caution:
The print system of your machine might store the data and
make it available to others!
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.5 (GNU/Linux)
Comment: A revocation certificate should follow
iFMEIBECABMFAkUU0hkMHQJjbGUgcGVyZHVlAAoJENRoX2PzrIOoHxcAoL7zk
RMPpPdMBHoDOOApHeJMzBtgAJwIzVoOmN6emCD1KPcDpWdyls3jLg===vhGS
-----END PGP PUBLIC KEY BLOCK-----
```

Il est conseillé de générer un certificat de révocation.

2.11 Gestion du porte-clés

Après la création du porte-clés, il est possible d'importer les clés publiques d'autres personnes dans notre porte-clés publiques (pubring). Afin de lister les clés qui sont dans notre porte-clés (keyring), il suffit de taper la commande suivante:

```
[romeo@localhost ~]$ gpg --list-keys
/home/romeo/.gnupg/pubring.gpg
------------------------------
pub   1024D/F3AC83A8 2019-06-11
uid   Romeo (Romeo Romeo) <romeo@hotmail.com>
sub   2048g/A614A959 2019-06-11

pub   1024D/ECE7A52A 2019-06-11
uid   Juliet (Juliet Juliet) <juliet@hotmail.com>
sub   2048g/B183A845 2019-06-11
```

Notre propre clé publique est déjà dans le porte-clés. Elle est séparée en deux parties : la première pour valider les signatures que l'on émet, et la deuxième pour que d'autres personnes puissent nous chiffrer des messages (que l'on déchiffrera donc avec notre clé secrète).

Il est possible d'ajouter une adresse de messagerie à la clé. On va éditer notre clé et ajouter un uid:

```
[romeo@localhost ~]$ gpg --edit-key romeo
.
pub  1024D/F3AC83A8  created: 2019-06-11  expires: never
     usage:SC  trust:ultimate  validity:ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
     usage: E
[ultimate] (1). Romeo (Romeo Romeo) <romeo@hotmail.com>
```

À ce stade, il est possible de taper la commande help pour avoir la liste des commandes disponibles.

```
Command> help
quit         quit this menu
save         save and quit
help         show this help
fpr          show key fingerprint
list         list key and user IDs
uid          select user ID N
key          select subkey N
check        check signatures
sign        sign selected user IDs [* see below for related commands]
lsign        sign selected user IDs locally
tsign        sign selected user IDs with a trust signature
nrsign       sign selected user IDs with a non-revocable signature
adduid       add a user ID
addphoto     add a photo ID
deluid       delete selected user IDs
addkey       add a subkey
addcardkey   add a key to a smartcard
keytocard    move a key to a smartcard
bkuptocard   move a backup key to a smartcard
delkey       delete selected subkeys
addrevoker   add a revocation key
delsig       delete signatures from the selected user IDs
expire     change the expiration date for the key or selected subkeys
primary      flag the selected user ID as primary
toggle       toggle between the secret and public key listings
pref         list preferences (expert)
showpref     list preferences (verbose)
setpref      set preference list for the selected user IDs
keyserver   set the preferred keyserver URL for the selected user IDs
notation     set a notation for the selected user IDs
passwd       change the passphrase
trust        change the ownertrust
revsig       revoke signatures on the selected user IDs
revuid       revoke selected user IDs
revkey       revoke key or selected subkeys
enable       enable key
disable      disable key
showphoto    show selected photo IDs
clean     compact unusable user IDs and remove unusable signatures
from key
minimize   compact unusable user IDs and remove all signatures from key
* The `sign' command may be prefixed with an `l' for local signatures (lsign),
  a `t' for trust signatures (tsign), an `nr' for non-revocable signatures
  (nrsign), or any combination thereof (ltsign, tnrsign, etc.).

Grâce à cette commande on peut ajouter un uid de la même façon qu'on avait créé
l'utilisateur initial:
```

```
Command> adduid
Real name: Hakim Benameurlaine
Email address: hakimb@yahoo.com
Comment: hb
You selected this USER-ID:
    "Hakim Benameurlaine (hb) <hakimb@yahoo.com>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O

You need a passphrase to unlock the secret key for
user: "Romeo (Romeo Romeo) <romeo@hotmail.com>"
1024-bit DSA key, ID F3AC83A8, created 2019-06-11
Enter passphrase: xxxxxxxx

            pub  1024D/F3AC83A8  created: 2019-06-11  expires:
never       usage: SC
                     trust: ultimate      validity: ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
usage: E
[ultimate] (1)  Romeo (Romeo Romeo) <romeo@hotmail.com>
[ unknown] (2). Hakim Benameurlaine (hb) hakimb@yahoo.com
```

```
Command> list
pub  1024D/F3AC83A8  created: 2019-06-11  expires: never
usage: SC
                     trust: ultimate      validity: ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
usage: E
[ultimate] (1)  Romeo (Romeo Romeo) <romeo@hotmail.com>
[ unknown] (2). Hakim Benameurlaine (hb) <hakimb@yahoo.com>
```

Pour sauvegarder:

```
Commande> save
```

Après les changements sauvés, on remarque qu'une nouvelle entrée est apparue dans notre porte-clés:

```
[romeo@localhost ~]$ gpg --list-keys
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
/home/romeo/.gnupg/pubring.gpg
------------------------------
pub   1024D/F3AC83A8 2019-06-11
uid                  Hakim Benameurlaine (hb) <hakimb@yahoo.com>
uid                  Romeo (Romeo Romeo) <romeo@hotmail.com>
sub   2048g/A614A959 2019-06-11

pub   1024D/ECE7A52A 2019-06-11
uid                  Juliet (Juliet Juliet) <juliet@hotmail.com>
sub   2048g/B183A845 2019-06-11
```

3 Les réseaux de confiance ("web of trust")

Maintenant que l'on a parlé de la partie technique concernant l'utilisation au jour le jour de GnuPG, il reste un problème majeur : l'authenticité des clés.

3.1 Empreintes

Imaginons qu'une clé que l'on vient de télécharger à partir d'un serveur de clés ne soit pas réellement la clé de la personne à laquelle on pense. Si on envoie un message chiffré en utilisant cette clé, une tierce personne aura accès à ce message, et pourra réémettre le message avec la clé réelle du destinataire, ce qui fait qu'une personne aura lu le message sans qu'aucune des deux extrémités ne soit au courant. De même, si un message vient d'une personne dont on ne possède pas la clé avec certitude, le fait de la télécharger à partir d'un serveur ne garantit pas son authenticité.

Il va falloir authentifier toutes ces clés. Pour l'instant, les différentes clés qu'on a pu récupérer par l'intermédiaire des serveurs de clés sont inutilisables car non sûres.

Nous allons donc utiliser des mécanismes de signature des clés et de contre-signature.

Dans un premier temps, il est possible d'afficher l'empreinte (fingerprint) des différentes clés de son porte-clés, et en particulier la sienne :

```
[romeo@localhost ~]$ gpg --fingerprint
/home/romeo/.gnupg/pubring.gpg
------------------------------
pub   1024D/F3AC83A8 2019-06-11
      Key fingerprint = 8473 C008 69C5 3341 09D6  01AF D468 5F63
F3AC 83A8
uid   Romeo (Romeo Romeo) <romeo@hotmail.com>
sub   2048g/A614A959 2019-06-11

pub   1024D/ECE7A52A 2019-06-11
    Key fingerprint = 4092 7689 B1E6 6428 BEA3  D842 C32B EBCD ECE7
A52A
uid                  Juliet (Juliet Juliet) <juliet@hotmail.com>
sub   2048g/B183A845 2019-06-11
```

```
[juliet@localhost ~]$ gpg --fingerprint
/home/juliet/.gnupg/pubring.gpg
------------------------------
pub   1024D/ECE7A52A 2019-06-11
Key fingerprint = 4092 7689 B1E6 6428 BEA3  D842 C32B EBCD ECE7
A52A
uid   Juliet (Juliet Juliet) <juliet@hotmail.com>
sub   2048g/B183A845 2019-06-11

pub   1024D/F3AC83A8 2019-06-11
      Key fingerprint = 8473 C008 69C5 3341 09D6  01AF D468 5F63
F3AC 83A8
uid                  Romeo (Romeo Romeo) <romeo@hotmail.com>
sub   2048g/A614A959 2019-06-11
```

On peut alors noter sur un support facilement transportable (typiquement, une carte de visite) le fingerprint correspondant à sa propre clé publique.

Lorsqu'on va rencontrer une personne avec qui on sera susceptible d'avoir des échanges utilisant GnuPG ou PGP, on s'échange ses fingerprint mutuels, dans notre cas en s'échangeant les cartes de visite. Pour que l'échange se fasse dans les règles de l'art, l'échange doit comporter une vérification de l'identité réelle de la personne grâce aux papiers d'identité.

Une fois qu'on a récupéré le fingerprint de la personne, on peut télécharger la clé publique à partir d'un serveur (petite astuce : les 8 derniers caractères du fingerprint correspondent à l'ID de la clé), et relancer la commande gpg --fingerprint pour afficher le fingerprint correspondant à la clé que l'on vient de télécharger. Il ne reste plus qu'à comparer le fingerprint obtenu et celui marqué sur la carte de visite, et si les deux concordent, alors on est sûr à 100% que la clé publique qu'on a maintenant dans notre porte-clés correspond à la personne avec qui on a fait physiquement l'échange des cartes et la vérification des papiers d'identité. En bref, on a la bonne clé publique.

3.2 Signature

On pourrait s'arrêter là, mais cela limiterait fortement les possibilités d'action (on ne pourrait utiliser GnuPG qu'avec des personnes qu'on aurait rencontrées physiquement). C'est là qu'entrent en jeu deux mécanismes : les contre-signatures et les réseaux de confiance.

On va donc contre-signer la clé publique que l'on vient de télécharger. La clé publique de notre porte-clés va donc être légèrement modifiée car on va y ajouter un petit morceau de signature propre à nous, généré par notre clé secrète, attestant qu'on est d'accord sur le fait que cette clé publique appartient bien à son propriétaire.

Il suffit d'utiliser la commande:

```
[romeo@localhost ~]$ gpg --sign-key romeo

pub  1024D/F3AC83A8  created: 2019-06-11  expires: never
usage: SC
                     trust: ultimate      validity: ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
usage: E
[ultimate] (1). Hakim Benameurlaine (hb) <hakimb@yahoo.com>
[ultimate] (2)  Romeo (Romeo Romeo) <romeo@hotmail.com>

Really sign all user IDs? (y/N) y
"Hakim Benameurlaine (hb) <hakimb@yahoo.com>" was already signed by
key F3AC83A8
"Romeo (Romeo Romeo) <romeo@hotmail.com>" was already signed by key
F3AC83A8
Nothing to sign with key F3AC83A8

Key not changed so no update needed.
```

```
[romeo@localhost ~]$ gpg --sign-key juliet
pub  1024D/ECE7A52A  created: 2019-06-11  expires: never  usage: SC
                     trust: unknown       validity: unknown
sub  2048g/B183A845  created: 2019-06-11  expires: never   usage: E
[ unknown] (1). Juliet (Juliet Juliet) <juliet@hotmail.com>
pub  1024D/ECE7A52A  created: 2019-06-11  expires: never  usage: SC
                     trust: unknown       validity: unknown
 Primary key fingerprint: 4092 7689 B1E6 6428 BEA3  D842 C32B EBCD
ECE7 A52A
     Juliet (Juliet Juliet) <juliet@hotmail.com>
Are you sure that you want to sign this key with your
key "Hakim Benameurlaine (hb) <hakimb@yahoo.com>" (F3AC83A8)
Really sign? (y/N) y
You need a passphrase to unlock the secret key for
user: "Hakim Benameurlaine (hb) <hakimb@yahoo.com>"
1024-bit DSA key, ID F3AC83A8, created 2019-06-11
Enter passphrase: xxxxxxx
```

On vient donc de signer (contre-signer en fait) la clé publique de notre interlocuteur. On va maintenant dire au reste du monde qu'on a signé la clé de l'autre personne.

On peut soit envoyer à son propriétaire la clé signée : on exporte la clé et on l'envoie:

```
[romeo@localhost ~]$ gpg --export --armor juliet
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.5 (GNU/Linux)

yYvWG17nVImaNUpTmjOYpxPyjMLN3qWVGLwIOp4fedAScoEs/9iYQu4
Epgd7lR7nHVNxsX1Jb5TXsJFIJAK7s2EdWclUFSnmWDJnaWKElOmCfb
oVApwOlWnAekGg8Hls6ttRCKXUSunQLbVbdu0tIP0BT/xKCBPfv8P1i
EkEGBECAAkFAkT+ascCGwwACgkQwyvrzeznpSo8LgCfYndWaamvvJ6t
Tzgksc1MLqEUYfsAoJ35fIK0oGNHgfS7kBWv/b7up3TD=GY96
-----END PGP PUBLIC KEY BLOCK-----
```

ou on l'envoie directement sur un serveur de clés :

```
$   gpg --keyserver pgp.mit.edu --send-key   juliet
```

L'usage fait que généralement, on fait les deux : on envoie la clé à un serveur de clés et on prévient l'autre qu'on a signé la clé. Lorsqu'on envoie la clé au serveur de clés ou lorsqu'on réimporte une clé qui a été signée depuis, cette dernière n'est pas écrasée, mais les signatures sont extraites et ajoutées le cas échéant. Les clés qu'on a récupérées ont donc peut-être été signées par d'autres personnes. Pour le voir, il suffit de taper la commande :

```
[romeo@localhost ~]$ gpg --list-sigs romeo
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:  1  signed:  1  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid:  1  signed:  0  trust: 1-, 0q, 0n, 0m, 0f, 0u
pub   1024D/F3AC83A8 2019-06-11
uid                  Hakim Benameurlaine (hb) <hakimb@yahoo.com>
sig 3   F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
uid                  Romeo (Romeo Romeo) <romeo@hotmail.com>
sig 3   F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
sub   2048g/A614A959 2019-06-11
sig      F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
```

On remarque qu'on a sa propre clé auto-signée, et que la clé qu'on a récupérée a été signée par plusieurs personnes qu'on ne connaît pas (on n'a pas leur clé publique dans notre porte-clés).

On peut donc savoir qui sont les personnes qui ont signé les clés de notre porte-clés. Mais avec l'option --list-sigs, on sait juste que la clé a été signée par un certain nombre de personnes, sans le vérifier. On peut demander à GnuPG de vérifier les différentes signatures qu'il possède de chaque clé avec les clés qu'il a dans son porte-clés:

```
[romeo@localhost ~]$ gpg --check-sigs romeo
pub   1024D/F3AC83A8 2019-06-11
uid   Hakim Benameurlaine (hb) <hakimb@yahoo.com>
sig!3   F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
uid     Romeo (Romeo Romeo) <romeo@hotmail.com>
sig!3   F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
sub   2048g/A614A959 2019-06-11
sig!     F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
```

- Les ? Correspondant à des signatures qui n'ont pas été vérifiées (on ne possède pas la clé publique pour vérifier).
- Les ! indiquent que la signature correspond à la clé publique que l'on a sur notre porte-clés.

Il est possible d'attribuer plusieurs niveaux de confiance à un utilisateur. Pour se faire il faut éditer la clé et utiliser la commande trust :

```
[romeo@localhost ~]$ gpg --edit-key romeo

pub  1024D/F3AC83A8  created: 2019-06-11  expires: never
usage: SC      trust: ultimate      validity: ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
usage: E
[ultimate] (1). Hakim Benameurlaine (hb) <hakimb@yahoo.com>
[ultimate] (2)  Romeo (Romeo Romeo) <romeo@hotmail.com>
Command> trust
pub  1024D/F3AC83A8  created: 2019-06-11  expires: never
usage: SC
                     trust: ultimate      validity: ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
usage: E
[ultimate] (1). Hakim Benameurlaine (hb) <hakimb@yahoo.com>
[ultimate] (2)  Romeo (Romeo Romeo) <romeo@hotmail.com>
Please decide how far you trust this user to correctly verify other
users' keys
(by looking at passports, checking fingerprints from different
sources, etc.)
  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu
Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y
pub  1024D/F3AC83A8  created: 2019-06-11  expires: never
usage: SC
                     trust: ultimate      validity: ultimate
sub  2048g/A614A959  created: 2019-06-11  expires: never
usage: E
[ultimate] (1). Hakim Benameurlaine (hb) <hakimb@yahoo.com>
[ultimate] (2)  Romeo (Romeo Romeo) <romeo@hotmail.com>
Command> save
Key not changed so no update needed.
```

```
[romeo@localhost ~]$ gpg --check-sigs romeo
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:  1  signed:  1  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid:  1  signed:  0  trust: 1-, 0q, 0n, 0m, 0f, 0u
pub   1024D/F3AC83A8 2019-06-11
uid                  Hakim Benameurlaine (hb) <hakimb@yahoo.com>
sig!3   F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
uid                  Romeo (Romeo Romeo) <romeo@hotmail.com>
sig!3   F3AC83A8 2019-06-11  Hakim Benameurlaine (hb)
<hakimb@yahoo.com>
sub   2048g/A614A959 2019-06-11
```