Chiffrement de donnée avec GnuPG

***

Ayitic – Port au Prince
11 – 16 Août 2014

***

Instructeur: LOISEAU Lucien

Objectifs du TP

Dans ce TP, vous allez comprendre et expérimenter l'outil GnuPG. Cet outil permet de chiffrer, déchiffrer et signer des messages en utilisant les algorithmes de chiffrements symmétrique ou asymétriques.

Des questions vous seront posées durant le TP, prenez le temps d'y répondre sérieusement ! Un certain nombre de commandes seront également introduites. Si une commande n'est pas comprise, il est très important d'avoir le reflexe de consulter le manuel avec la commande suivante :

```
$ man gpg
```

Comme pour la plupart des logiciels sous GNU/Linux, la commande `man` suivi du nom de la commande permet d'avoir beaucoup d'information. Pour fouiller rapidement une page de manuel à la recherche d'un mot clé particulier (par exemple `--sign`) tapez le caractère `/` suivi du texte à chercher puis appuyez sur entrée. Des appuis successifs sur la touche `n` (comme next) permet de sauter d'une occurrence à une autre.

Préparation

Gnu Privacy Guard (GnuPG)

Assurez vous que GnuPG soit bien installé sur votre système. Vous pouvez vérifier cela en tapant dans un terminal :

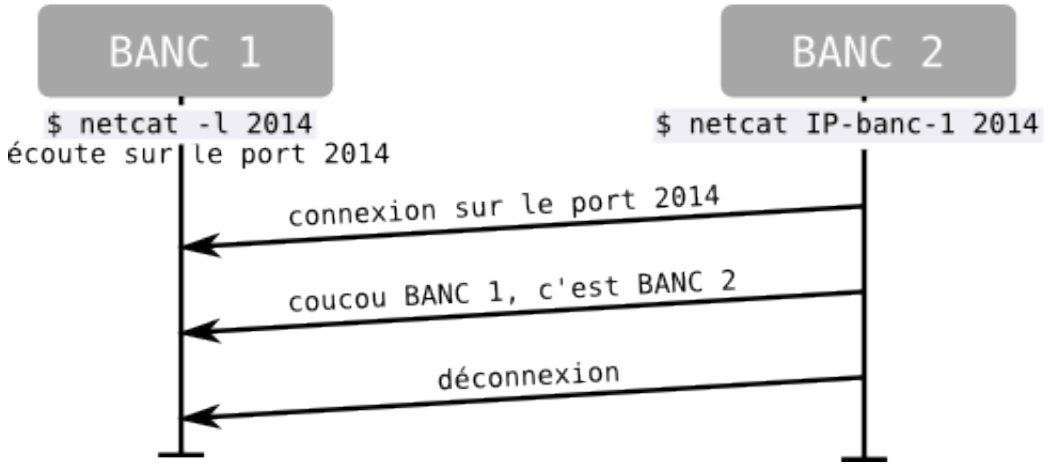
```
$ gpg --version
```

Si du texte s'affiche, alors le logiciel est installé et vous pouvez commencer le TP. Sinon, il est nécessaire de le télécharger et de l'installer, voyez avec l'instructeur si c'est le cas. Vérifiez également que le système est bien à l'heure.

Envoyer / Recevoir des données entre les bancs

Durant ce TP, nous aurons besoin d'envoyer et de recevoir un certain nombre de message entre les bancs. Pour cela, nous utiliserons l'outil `netcat` qui permet de faire cela très simplement. `Netcat` est un utilitaire en ligne de commande qui permet de manipuler des sockets TCP ou UDP. Par exemple, si le *banc 2* veut envoyer un message au *banc 1*, nous procéderons de la façon suivante :

- Le *banc 1* écoute avec `netcat` sur le port `TCP 2014`
- Le *banc 2* se connecte avec `netcat` sur l'`IP` du banc 1 (port `TCP 2014`) et envoie un message



Connaitre son adresse IP :

Tapez la commande `ifconfig eth0` afin de connaître votre adresse IP :

```
$ ifconfig eth0 | grep inet
        inet 192.168.1.43  netmask 255.255.255.0  broadcast 192.168.1.255
```

Écouter sur le port TCP 2014 :

Afin d'écouter sur le port `TCP 2014` pour recevoir un message, tapez la commande suivante :

```
$ netcat -l 2014
```

Cette commande restera en attente d'une connexion. Lorsqu'une connexion sera effectué par un autre banc, il affichera sur la sortie du terminal tout les messages reçu. Cette commande s'arrête lorsque le client se déconnecte.

Vous pouvez également rediriger la sortie de `netcat` vers un fichier si vous vous attendez à recevoir un fichier :

```
$ netcat -l 2014 > fichier
```

Envoyer des données sur le port TCP 2014 :

Pour envoyer un message à un banc, il faut d'abord connaître son adresse `IP`. Si par exemple l'adresse `IP` du banc destination est `192.168.1.43` et que celui ci écoute sur le port `TCP 2014`, alors on peut envoyer des messages de la façon suivante :

```
$ netcat 192.168.1.43 2014
coucou
c'est moi
^C
$
```

(^C = contrôle + C, cela ferme le programme et donc termine la connexion). On peut également envoyer un message en l'envoyant directement sur l'entrée standart de `netcat` :

```
$ echo "coucou" | netcat 192.168.1.43 2014
```

On peut également envoyer un fichier :

```
$ netcat 192.168.1.43 2014 < fichier
```

Pour avoir plus d'information sur le fonctionnement de netcat, qui est un outil très versatil, vous pouvez consulter sa page de manuel :

```
$ man netcat
```

Ajoutez votre adresse IP ainsi que votre nom et numéro de banc au tableau

Exercice

Essayez d'envoyer un message à un autre groupe :

- échangez vous vos adresses `Ips` (ou regardez sur le tableau)
- utilisez `netcat` pour écouter sur le port `TCP 2014` (pour recevoir des messages)
- utilisez `netcat` pour vous connecter à un autre banc et envoyer un message

Répetez l'opération mais en échangeant cette fois des fichiers !

Question 0 : Serait il possible pour un attaquant de voir les données transmises ? Comment ?

Chiffrement Symétrique

1. Motivation

Question 1 : Imaginez des raisons valables d'envoyer des messages secrets: entre personnes se connaissant et entre personnes ne se connaissant pas.

2. Du texte clair au texte chiffré

Un certain nombre d'algorithmes de chiffrement et de hashage sont pris en charge par GnuPG. Pour voir lesquelles sont supportés avec votre version il suffit de lancer gpg avec l'argument `--version` :

```
$ gpg --version
gpg (GnuPG) 2.0.25
libgcrypt 1.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: ~/.gnupg
Algorithmes pris en charge :
Clef publique : RSA, ELG, DSA
Chiffrement : IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256,
              TWOFISH, CAMELLIA128, CAMELLIA192, CAMELLIA256
Hachage : MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression : Non compressé, ZIP, ZLIB, BZIP2
```

Nous allons chiffrer un message avec un algorithme de chiffrement symmétrique. Écrivez quelque chose dans un fichier et enregistrez le. Nous appelerons ce fichier `message.txt`

Pour chiffrer un fichier avec un algorithme à clé symmétrique nous avons besoin de décider deux choses :

- d'un mot de passe à partir duquel sera généré la clé symmétrique
- d'un algorithme de chiffrement (exemple : `AES256`)

La commande suivante permet de chiffrer le fichier :

```
$ gpg --symmetric fichier.txt
```

Le fichier chiffré est enregistré sous le nom de fichier.txt.gpg. On peut choisir le nom du fichier chiffré avec le paramètre `--output`.

Si aucun fichier d'entrée n'est donné en paramètre, gpg va chiffrer ce qui arrive sur l'entrée standard. On peut donc arriver au même résultat avec la commande suivante :

```
$ cat fichier.txt | gpg --symmetric --output fichier.encrypted
```

`cat` est une commande qui affiche sur la sortie standard le contenu d'un fichier, « | » (qu'on appelle `pipe`) permet de connecter la sortie standard sur une entrée standard). On peut directement écrire notre message en ligne de commande sans l'enregistrer préalablement dans un fichier :

```
$ echo "petit message" | gpg --symmetric --output fichier.encrypted
```

Question 2 : Quel algorithme de chiffrement est utilisé par défaut ?

Il est possible de choisir son propre algorithme avec le paramètre `--cipher-algo`. Rechiffrez votre message avec un algorithme de votre choix.

GnuPG peut également adapter sa sortie de sorte que le message chiffré soit une suite de caractère ASCII (on appelle cela le base64) facilement copiable. Pour cela, il suffit d'ajouter le paramètre `--armor` de la façon suivante :

```
$ echo "ceci est un message secret" | gpg --symmetric --cipher-algo AES256 --armor
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.10 (GNU/Linux)

jA0ECQMCF3jLAD0cIq1g0lABBVMPzsy41ZKMfedIhEGSx0TIO76KDhu5Kb72oHM5
O3IisQtZnJqyWee5tj3hRuhmZTUzWFybhuUIYMK3ABZtbUHGErk9BvH9sFKun5P/
iA==
=0LiJ
-----END PGP MESSAGE-----
```

Ce message peut être facilement copié dans un mail, un forum ou même un chat (nous verrons qu'il existe quand même des plugin pour nous faciliter la tâches malgré tout).

2. Du texte chiffré au texte clair

Pour déchiffrer un message chiffré, il suffit d'utiliser le paramètre`--decrypt` (ou `-d`) qui vient remplacer `--symmetric` (ou `-c`). Exemple avec un fichier chiffré nommé secret.txt :

```
$ gpg --decrypt secret.txt
gpg: données chiffrées avec AES256
gpg: chiffré avec 1 phrase de passe
ceci est un message secret
```

Question 3 : Pourquoi n'avons nous pas besoin de préciser l'algorithme de chiffrement utilisé ?

Chiffrez deux messages différents avec deux clés différentes mais avec le même algorithme de chiffrement et avec le mode armor, comparez les deux textes chiffrés.

Question 4 : Est-ce que cela confirme ce que vous pensiez à la question 3 ?

3. Exercice

- Envoyez un message chiffré à un autre groupe de la salle et arrangez vous pour lui faire parvenir le mot de passe. ATTENTION ! N'utilisez pas `netcat` pour partager le secret ! Et gare aux oreilles !
- Déchiffrez les messages que vous avez reçus

Question 5 : Quelle méthode avez vous choisie pour échanger la clé de chiffrement ?

Question 6 : Pourquoi ne faut il pas envoyer le secret à travers le même canal de communication que les données chiffrées ?

4. Chiffrer une clé USB (TrueCrypt)

Chiffrement Asymmétrique

1. Questions préliminaires

Question 7 : Quel est l'avantage du chiffrement à clé publique (asymmétrique) par rapport au chiffrement à secret partagé (symmétrique) ?

Question 8 : Quel est son inconvénient ?

1. Création des paires de clés

Pour chiffrer des messages avec un algorithme à clé publique (chiffrement asymmétrique) il faut d'abord posséder une paire de clé publique/clé privée qui est propre à chaque individu (rien n'empêche d'en avoir plusieurs évidemment). Pour cela on utilise la commande intéractive `gpg --gen-key` qui va construire une identité et la paire de clé correspondante.

```
$ gpg --gen-key
```

Répondez aux questions de la façon suivante :

- Types de clé : `RSA et RSA (par défaut)`
- Taille de la clé : `2048`
- Nom Réel : `votre nom et prénom`
- Adresse e-mail : `nom.prenom@ayitic-securite.com`
- Commentaire : `ce que vous voulez`

Un mot de passe vous sera demandé pour générer la paire de clé publique/privée. Choisissez un mot de passe robuste que vous utiliserez durant tout le TP.

Un bon mot de passe est un mot de passe unique et aléatoire. Si la commande existe, vous pouvez tapez `mkpasswd` pour générer automatiquement un mot de passe.

Ce conseil s'applique également pour n'importe quel mot de passe. Puisqu'il est difficile de mémoriser plusieurs mots de passe aléatoires, vous pouvez enregistrer vos mots de passe dans un fichier texte chiffré ou alors dans un gestionnaire de mots de passe.

Validez, la paire de clé va être générée et cela peut prendre un peu de temps. Vous pouvez accélerer cette étape en lançant une commande qui va augmenter l'entropie du système comme par exemple :

```
$ find /
```

Une fois l'étape terminé vous devriez voir quelque chose comme ceci :

```
pub   2048R/9823F3B3 2014-07-21
 Empreinte de la clef = 5981 6611 DDF0 D302 26BE  F3A0 A71C 2515 9823 F3B3
uid      [  ultime ] loiseau lucien (pas de commentaire) <loiseau.lucien@ayitic-
securite.com>
sub   2048R/ADDD8B69 2014-07-21
```

Vous pouvez lister et éditer les clés présente avec les paramètre `--list-keys` :

```
$ gpg --list-keys
pub   2048R/9823F3B3 2014-07-21
uid      [  ultime ] loiseau lucien (pas de commentaire) <loiseau.lucien@ayitic-
securite.com>
sub   2048R/ADDD8B69 2014-07-21
```

On peut voir sur cet exemple que l'ID de le clé de loiseau.lucien@ayitic-securite.com est `9823F3B3`.
On peut également éditer (modifier) sa clé avec le paramètre `--edit-key` :

```
$ gpg --edit-key loiseau.lucien@ayitic-securite.com
[…]
gpg> ?
```

Question 9 : Quel est l'ID de votre clé ? Ajoutez là au tableau

Si vous perdez votre clé, ou votre mot de passe, ou que l'on vous vole votre ordinateur personnel, il est primordial de prévenir vos contacts de ne plus utiliser votre clé publique. Cela se fait au travers d'un

certificat de révocation que l'on crée de la manière suivante :

```
$ gpg --output revoke.txt --gen-revoke <UID>
```

Question 10 : Générez un certificat de révocation pour votre clé. Comment et où le stockez vous ?

Afin de révoquez une clé, il faut importer le certificat de révocation avec le paramètre `--import` (ne tapez pas cette commande sinon vous revoqueriez votre clé) :

```
$ gpg --import revoke.txt
```

2. Exporter et Importer une clé publique

Pour pouvoir chiffrer un message, il faut se procurer la clé publique du destinataire du message. Inversement, pour recevoir un message chiffré il faut que l'expéditeur se soit procuré votre clé publique. Pour cela on peut utiliser gpg avec le paramètre `--export` pour obtenir sa clé publique. Afin de pouvoir facilement partager cette clé publique, vous pouvez également ajouter l'option `--armor` (ou `-a`) de la façon suivante : (remplacez loiseau.lucien@ayitic-securite.com par votre propre identité) :

```
$ gpg -a --export loiseau.lucien@ayitic-securite.com
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQENBFPM0U8BCADN4vmVCblzBEPvnN9XdDu+ESAcctZaaMjpTMinTNcoObwCayAk
Y6nW+KefW3BW3cOK/uuYy8CcD0fMEWcJ98Z6oWS8DOmqjwiXSZJ8nXYVr3rVE/7D
R9g+qzYKcsqs4e2fN806t91xzePuErmox28j68IFdZVKmnJ6IcP4gRcJTuBxInYL
hLFDcVEc6useV1UOBnsklAfx+NTBoDYFpdvfQxBRUeXszjyUSHRKjmOCQRM5OzNi
DctWcQOdZfjMsm6dR8dVyP5Eb882DDQ/iEUEdL5ZGi8sLrxLyMnvQrpqZ8ne5YHK
hTu8dHLogTspwIHYO27chTaDU4bVyThLxIt7ABEBAAG0SGxvaXNlYXUgbHVjaWVu
IChwYXMgZGUgY29tbWVudGFpcmUpIDxsb2lzZWF1Lmx1Y2llbkBheWl0aWMtc2Vj
dXJpdGUuY29tPokBOQQTAQIAIwUCU8zRTwIbAwcLCQgHAwIBBhUIAgkKCwQWAgMB
Ah4BAheAAAoJEKccJRWYI/OzgLUH/imQiWWeVSceIc+b0Wxa8FvrBerfyn2ONMkg
tSMFmbyxKvyJmuxnYzGVU5PXqjcduTQsTDjen/VTmx7jGrJv3S2XF20hiaIyLQap
ydnlgfJppG6KUvWS9LbZTVgiAUFI0lj9HTw1EiJkIevjqcSzmfUMAGvH3rTmdAdL
/JgXM/bQXN+GKiWWi3PcbAaQYX3h1SVaZfbaLfb1WuRsC280VIo6873+zwI9UwbG
kTh9+ewKfflzbEnKfcBqpx2DMZpOr6qzErRascELz/pwBjZZn5uRpX22hkymdOEs
jiV+2g7NCLmv1qD72dhLs0jkDihIXp4IUfZYuCgnqFzaQcFCpFG5AQ0EU8zRTwEI
AJZxXqXkzbKXY3IpS7oAp3Ux0+97/6Ue4l9ipCc1dm+2iVzLv3PavwYtWD1Yascb
r/lsqsYjcJgKO9F67UCdVsWhG0It2Uk2JCErBTWeSIE5GaLJkmRuBTzpd4ckenGc
KFOM2oqgofH5LvOHVEIPcgBmj+bFnJNhw6kGoBQQyTqpRE3vmly9QEmvB6vCFD0o
Uha/4+T8GughrlePy96IAJLblsNYhPGoDPapkTFSGOP+kgY5qZTfzan2PwYRpRI/
ugxtOzVb6EA+l8WN7QVIs22iqCJhen05vy2C87SGUPjmr/aSnKE1u9m0AzaNS8KT
TzBCyupgB+e+sZDtRl6tClEAEQEAAYkBHwQYAQIACQUCU8zRTwIbDAAKCRCnHCUV
```

```
mCPzs6pQB/9LeTjh1Gp86F9SJyILsPUB4gN1tVnD5CNRsbnfpTAlK56n+FUSDSSf
2dL64lANdarkqY71rrGYNWdk3CEX8U8u8JideYUBM+nS9hHH2Mf4t6aDquH3Wbep
9ejD/myoDI1rgqLyU4lMUMQlwxxHP+xTFRmUhgj99KxvyFY78ThJ7j4W+PZkKZOa
KLeDY3+tfBE+1BRvsu17t/rF4TyZ3rpdblWr9SmZnRT+h35goLgmpRAWIUADRvSh
a/7LaQ0UAv0dz2BKOq/5qObWyfMR+dsJiijAUxXk2WUjVg/BcoiWqIXbw3gCmWYy
qCplMd27gEAnMzdAXcBQXLcbTTYBNjSJ
=Q7c3
-----END PGP PUBLIC KEY BLOCK-----
```

De la même façon, vous pouvez importer une clé publique, c'est à dire l'ajouter à votre trousseau, avec le paramètre `--import` suivi du nom du fichier contenant la clé publique.

Question 11 : Quelle mesure doit on absolument prendre avant d'importer une clé publique ?

Vous pouvez obtenir le fingerprint d'une de vos clé publiques ou d'une clé publique déjà importé de la façon suivante (remplacez loiseau.lucien@ayitic-securite.com par votre propre identité) :

```
$ gpg --fingerprint loiseau.lucien@ayitic-securite.com
pub   2048R/9823F3B3 2014-07-21
 Empreinte de la clef = 5981 6611 DDF0 D302 26BE  F3A0 A71C 2515 9823 F3B3
uid      [  ultime ] loiseau lucien (pas de commentaire) <loiseau.lucien@ayitic-
securite.com>
sub   2048R/ADDD8B69 2014-07-21
```

Exercice : Importez ma clé publique (disponible au dessus) et comparez son fingerprint avec celui au dessus, est-ce le même ?

Question 12 : Pensez vous avoir effectué une vérification suffisante de la clé publique de l'instructeur ?

Levez la main et appelez l'instructeur afin que celui-ci confirme son fingerprint

La vérification d'une clé publique doit impérativement se faire en « vrai » lorsque cela est possible afin de pouvoir vérifier le nom et prénom, le mail et le fingerprint (c'est à dire vérifier que la personne soit bien qui il prétend être).

Les clés peuvent être distribuées directement via des annuaires. Cela permet de simplifier le processus d'exportation et d'importation. Il existe plusieurs sites permettant de stocker les clés publiques comme

`pgp.mit.edu` ou `keys.gnupg.net`. Généralement ces sites synchronisent régulièrement leurs bases de données.

NE TAPEZ PAS LES COMMANDES SUIVANTES,
ELLES SONT DONNÉES À TITRE INFORMATIVES

ous pouvez rechercher des clés sur ces annuaires de la façon suivante :

```
$ gpg --keyserver pgp.mit.edu --search loiseau.lucien
  gpg: recherche de « loiseau.lucien » sur le serveur hkp pgp.mit.edu
(1)   Lucien Loiseau <lucien@teupos.fr>
        2048 bit RSA key BF4F61A8, créé : 2012-03-22
(2)   Loiseau Lucien <loiseau.lucien@gmail.com>
        2048 bit RSA key F2DF8532, créé : 2012-03-21
(3)   Loiseau Lucien <loiseau.lucien@gmail.com>
        1024 bit DSA key 49647148, créé : 2009-01-13
(4)   Loiseau Lucien (clef standart) <lucien.loiseau@etu.univ-nantes.fr>
        1024 bit DSA key 8EB050EA, créé : 2008-02-09
Entrez le ou les nombres, (S)uivant, ou (Q)uitter >
```

Vous pouvez également mettre à jour votre base de clé locale avec le paramètre `--refresh-keys` :

```
$ gpg --keyserver pgp.mit.edu --refresh-keys
```

À l'avenir, vous pourrez publier votre clé publique sur un annuaire. Ne le faites pas lors des exercices et si vous n'êtes pas au courant des risques, et faites le en connaissance de cause ! Mais si vous voulez le faire, faites le avec la commande suivante :

```
$ gpg --keyserver pgp.mit.edu --send-key loiseau.lucien@ayitic-securite.com
```

Question 13 : Quel(s) avantage(s) voyez vous à l'utilisation d'annuaires ?

Question 14 : Quel(s) inconvénient(s) voyez vous à l'utilisation d'annuaires ?

2. Chiffrer un message

Lorsque vous possédez la clé publique d'un destinataire (vous devez normalement avoir importé la clé publique de l'instructeur loiseau.lucien@ayitic-securite.com), vous pouvez chiffrer un message avec le

paramètre `--encrypt` (ou `-e`). Comme pour le chiffrement symmétrique, vous pouvez également ajouter les paramètres `--armor` et/ou `--output`

```
$ echo « message clair » | gpg -a --encrypt -r loiseau.lucien@ayitic-securite.com
-----BEGIN PGP MESSAGE-----
Version: GnuPG v2

hQEMA+0OW4St3YtpAQf/e+/2E/HoE2CkPJzNq2io/a83kI/RhxVnXck6WE+C+0mh
tbmcHj+EVz/AdGQnip5B5hjfGA099SuNE8nVBLwOcmp/fGcWz2z+T9jALU/9FBVu
lRTEXBZne1hqoh/BgQBygjK1FS+DXDw0yOe9jAH4iOs9fpaVwnGqvEfTcz1HjCVO
jS+wwhZuDKPPNHwDseq/qlGyg/jeytPeX4PaqUMI6KzyUAbvdIhpFVIjJ7GRku+r
500jvIJuBoXDJzopsR2th+Nb1pkif8MvxAVSvQ3SNdCRGZEVowfrFwVcCpo+s1RW
IVvF6/dbNq6yt9QR4ocyR9gjqnqtjXyx2ut31TfY2dJJAaKQx6SRG/n9gRJDLQ3k
0lHKYj5kkB94cctvmyQrKkaDBRvseDzs433v3BxHGBLnjEf6Tm/6FthkwQpWMazG
JBR2hTv6d3kAYw==
=AjWc
-----END PGP MESSAGE-----
```

Question 15 : à quoi sert le paramètre `-r` *?*

- **Coordonnez vous avec l'instructeur et envoyez lui un fichier chiffré via `netcat` contant :**
  - **votre message**
  - **votre clé publique**
- **Si cela a fonctionné, l'instructeur vous enverra en retour un message chiffré**

3. Déchiffrer un message

Pour déchiffrer un fichier chiffré, vous pouvez utiliser gpg avec le paramètre `--decrypt` (ou `-d`) :

```
$ gpg --decrypt fichier.encrypted
```

Déchiffrez le message envoyé par l'instructeur et donnez le à l'instructeur pour qu'il le valide

Question 16 : `gpg` *ne vous a pas demandé avec quel clé il fallait le déchiffrer, d'après vous pourquoi ?*

3. Exercice

- Échangez votre clé publique avec un autre groupe de la salle
- Envoyez un (ou plusieurs) message(s) chiffrés
- Déchiffrez le (ou les) message(s) que vous avez reçus

Question 17 : Contrairement au chiffrement symmétrique, pourquoi n'y a-t-il cette fois aucun risque à envoyer votre clé publique par un canal de communication non sécurisé ?

Signature

1. Signer et vérifier des données

GnuPG vous permet également de signer des données (fichiers, clés). Signer des données permet d'assurer à la fois l'intégrité des données mais aussi son authenticité, c'est à dire d'assurer qu'elles ont bien été envoyés par la personne prétendant être à l'origine de ces données.

Signer les données

Comme on l'a vu pendant le cours, la signature est indépendante du chiffrement, on peut transmettre des données en clair mais joindre aux données une signature permettant d'assurer l'intégrité, l'authentification et la non-répudiation. GnuPG propose trois façon de signer des données.

- `--sign` : fournit un bloc (binaire ou ASCII selon `--armor`) contenant les données + la signature . Il est généralement combiné avec le paramètre `--encrypt` pour chiffrer les données en plus de les signer.

- `--clearsign` : fournit un bloc lisible contenant les données + la signature. Il est généralement utilisé pour signer un texte écrit en langage naturel.

- `--detach-sign` : fournit une simple signature sans les données. Il est généralement utilisé pour vérifier qu'un fichier téléchargé soit intègre et authentique.

Nous allons commencer par signer un paquet de donnée. Écrivez quelque chose dans un fichier et nommez le par exemple «`fichier` ». Nous allons d'abord le signer avec le paramètre `--detach-sign` et enregistrez la signature dans `fichier.asc1` (remplacez loiseau.lucien@ayitic-securite.com par votre identite) :

```
$ gpg --local-user loiseau.lucien@ayitic-securite.com --armor \
      --output fichier.asc1 --detach-sign fichier
```

Question 15 : À quoi sert le paramètre `--local-user` ? Est-ce la clé publique ou privée qui est utilisée pour signer les données ?

Si vous n'avez qu'une seule identité sur votre ordinateur, `--local-user` est optionnel car votre identité sera utilisé par défaut. Re-signez maintenant le même fichier avec la même option mais enregistrez la sortie dans `fichier.asc2`.

Question 16 : La signature est-elle la même ? Pourquoi ?

Vérifier les données

Lorsqu'on possède une signature (par exemple `fichier.asc1` ou `fichier.asc2`), on peut la comparer aux données avec le paramètre `--verify`. Comme une signature détachée (`--detach-sign`) n'inclut pas les données, il faut ajouter le fichier à vérifier :

```
$ gpg --verify fichier.asc1 fichier
gpg: Signature faite le lun. 21 juil. 2014 23:41:36 CEST avec la clef RSA
d'identifiant 9823F3B3
gpg: Bonne signature de « loiseau lucien (pas de commentaire)
<loiseau.lucien@ayitic-securite.com> » [ultime]
```

Question 17 : Expliquez comment procède `--verify` pour donner ce résultat

Question 18 :Regardez bien la sortie de `--verify`*, pouvez vous maintenant répondre à la question 16 ? Si non, faites un* `gpg --verify` *avec le* `fichier.asc2` *et répondez à la question.*

Signez le fichier avec les deux autres options `--sign`, `--clearsign` et vérifiez les avec `--verify`. Ces deux options inclut déjà les données dans la signature, il n'est donc pas nécessaire cette fois d'ajouter le fichier.

Exercice :

- Coordonnez-vous avec l'instructeur et envoyez un message chiffré et signé
- L'instructeur vous enverra en retour un message chiffré et signé, donnez lui l'heure de la signature du message afin de valider cet exercice.

2. Signer des clés publiques

Ce système de signature permet également la signature des clés publiques des autres utilisateurs. Si **Alice** signe la clé publique de **Bob**, cela revient à dire publiquement « Moi, **Alice**, certifie que cette clé publique appartient bien à **Bob** ». Ces signatures pouvant être exportées sur un annuaire, cela permet de créer un réseau de confiance.

Attention : La signature d'une clé demande autant de rigueur que d'importer une clé publique ! on ne signe jamais la clé de quelqu'un qu'on a jamais vue ou qu'on ne connaît pas
(comment assurer sinon que c'est bien cette clé qui lui appartient?)

Pour signer une clé il faut utiliser le paramètre `--sign-key` :

```
$ gpg --sign-key <une personne>
```

Signez la clé de l'instructeur. Vous pouvez ensuite vérifier la liste de vos signatures avec le paramètre `--list-sigs` et `--check-sigs` :

```
$ gpg --list-keys
```

Exercice :

- Choisissez un groupe avec qui vous n'avez pas déjà échangé vos clés publiques
- Échangez vos clés publiques en vérifiant bien les informations et les fingerprint
- Signez vos clés publiques si (et seulement si) les informations sont corrects
- Envoyez vous un message chiffré et signé
- Révoquez votre clé et envoyez votre certificat de révocation à tout ceux à qui vous avez partagé votre clé

Conclusion

Durant ce TP, nous avons appris à utiliser l'outil en ligne de commande GnuPG pour chiffrer de façon symmétrique des messages , créer une identité à base de clé publique/clé privée et chiffrer et signer de façon asymmétrique. Nous avons également insisté sur la difficulté à partager un secret dans le cadre du chiffrement symmétrique et à vérifier une identité dans le cadre du chiffrement asymmétrique.

Tout cela peut être fastidieux au quotidien mais heuresement il existe des plugins permettant de simplifier l'envoie et la réception de messages chiffrés. Si vous utilisez Thunderbird pour consulter vos mails, il existe un plugin appelé Enigmail qui permet de faire ce que nous avons vu aujourd'hui de façon graphique et intégré !