(in)sécurité des e-mails et utilisation de PGP

AliENS*

14 octobre 2021

Durant cette initiation, nous allons voir comment protéger nos messages à l'aide de l'outil `GnuPG`.

Table des matières

1 Introduction à la cryptographie

La *cryptographie* désigne l'ensemble des méthodes mises en œuvre pour protéger nos communications. Elle ne cherche pas uniquement à garantir que personne en dehors du destinataire ne puisse lire le message, mais cherche aussi à assurer l'intégrité de ceux-ci ainsi qu'une assurance sur l'identité de l'expéditeur par des méthodes que nous verrons au cours de cet atelier.

*Contact: `fabrice.mouhartem@ens-lyon.org`.

1.1 Pourquoi envoyer un mail n'offre aucune garantie sur la confidentialité ?

Envoyer un mail revient à envoyer une carte postale : même si le protocole est chiffré (c'est-à-dire qu'un membre du réseau ne peut pas lire votre message même s'il le voyait passer, pour reprendre l'analogie, comme les cartes voyagent dans le camion de la poste, un individu quelconque n'est pas en mesure d'en lire le contenu), le serveur de mail (autrement dit la poste, qui participe activement au transit) est en mesure de lire le message.

Notre but ici est de remplacer cela par une enveloppe scellée et une missive signée.

1.2 Comment se protéger ?

Heureusement pour nous, la question a déjà été étudiée auparavant et des solutions existent pour protéger nos données (pas seulement nos e-mails donc).

Pour cela nous avons différents outils :

— **Le chiffrement :** pour garantir que seul le destinataire sera en mesure de lire l'email en question ;
— **La signature :** elle permet d'assurer deux choses, d'une part que c'est bien nous qui avions envoyé le message, et d'autre part que le message n'a pas été modifié depuis l'opération de signature.

1.2.1 La cryptographie asymétrique : principes

La cryptographie à clé publique ou cryptographie asymétrique se caractérise par la présence de 2 clés différentes : une *privée* et une *publique*.

Comme leur nom le laisse à penser, la clé **privée** doit être précieusement gardée et la clé **publique** distribuée à ceux qui veulent, par exemple, vous envoyer un e-mail ou vérifier que le mail qu'ils ont reçu est bien de vous.

On touche là un point sensible : *comment garantir que la clé publique dont on dispose est bien celle de son interlocuteur et non d'un parti malicieux ?*

Une solution possible est de passer par un *tiers de confiance*, potentiellement un serveur de clés à qui on demandera « quelle est la clé publique de X ? » et il répondra « la voilà ».

Une autre solution est de proposer sa clé publique sur son site web ou les réseaux sociaux.

Mais comme nous sommes ~~paranoïaques~~ des crypto-anarchistes, ces solutions ne nous conviennent pas. En effet cela implique de faire confiance à un tiers, ou d'être sûr que les moyens de communications de notre interlocuteur sont sûres (qu'il n'ait pas été piraté par exemple).

C'est pourquoi nous allons simplement échanger nos clés en mains propres en étant vraiment sûr que notre interlocuteur est bien celui qu'on croit.

1.2.2 Pourquoi ne pas utiliser une clé commune ?

Une autre solution aurait été celle du monde symétrique, qui est de partager une clé commune.

Un problème de cette solution est qu'il faut assurer un échange sûr de la clé en question entre deux partis qui veulent communiquer. Un autre problème est que l'on doit retenir la clé partagée de chacun de nos interlocuteurs, et si on veut communiquer entre 8 personnes, il y a $\frac{8\times7}{2} = 28$ clés à générer, contre 8 paires dans le cas asymétrique.

En revanche les opérations sont en général plus rapides dans les protocoles symétriques, c'est pourquoi ils sont parfois utilisés en couplage avec de la cryptographie asymétrique pour produire un moyen sûr d'échanger les clés.

2 Mise en pratique : `PGP`

2.1 Qu'est-ce que PGP ?

`PGP` est un ensemble de standards qui définissent comment doivent être implanté les protocoles que nous avons vu précédemment, initialement prévu pour la sécurité des e-mails.

Comme il serait douloureux de les coder nous même, nous allons utiliser un outil préexistant pour gérer tout cela : `GnuPG`.

2.2 Premiers pas

2.2.1 Configurer GnuPG

Comme une grande partie des utilitaires sous *linux* (`git` par exemple), `GnuPG` se configure à l'aide d'un fichier de configuration.

Ici le fichier de configuration local se situe dans votre répertoire personnel dans le dossier `.gnupg/`, s'il n'existe pas encore, créez-le et modifiez le fichier `.gnupg/gpg.conf` :

```
### Commandes générales ###
keyserver hkp://hkps.pool.sks-keyservers.net   # un serveur de clés publiques
no-greeting             # cacher un message de copyright
armor                   # exporte de préférence en base64 et non en binaire
keyid-format 0xlong     # format long pour les clés, limite le risque de collision
with-fingerprint        # même idée: afficher l'empreinte complète
### Configuration de Crypto ###
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP Uncompressed
personal-cipher-preferences AES256 AES192 AES CAST5
personal-digest-preferences SHA512 SHA384 SHA256 SHA224
personal-compress-preferences ZLIB BZIP2 ZIP Uncompressed
keyserver-options no-honor-keyserver-url
```

Quelques explications. `armor` permet d'utiliser l'encodage *binaire-vers-texte* `base64`, qui utilise la décomposition en base 64 du message binaire pour l'encoder en utilisant 64 caractères `ascii` (donc sans accents ou autre). La commande `base64` permet aussi d'effectuer cette opération (et dans le sens inverse avec l'option `-d`).

Remarque. Il est recommandé d'utiliser un protocole chiffré pour communiquer avec le serveur de clés. Je vous laisse chercher comment faire pour le serveur `sks`.

Note. c'est bugué dans GnuPG 2.1.

Vous avez là un fichier de configuration très minimal, plus d'informations sur comment configurer `GnuPG` sont disponibles en section 3.

2.2.2 Créer sa paire clé publique/clé privée

Maintenant les choses sérieuses, créer ses clés et les gérer.
Pour cela il vous suffit de demander gentiment à `GnuPG` de le faire pour vous :

```
gpg --full-gen-key
```

Et de répondre aux questions qu'il vous pose. Pour le choix de la taille de clé, je vous laisse vous référer à la section 3.3. Il ne vous reste plus qu'à attendre que votre ordinateur génère assez d'entropie pour générer un mot de passe *sûr*; ce qui peut prendre plusieurs minutes.

Attention

la clé privée n'est qu'un fichier sur votre ordinateur, si quelqu'un la récupère, alors il pourra déchiffrer vos mails et signer à votre place. Pour pallier à ça, il est fortement recommandé de protéger sa clé par un mot de passe pour **limiter** les risques.

Note. le chiffrement de sa clé privée est obligatoire depuis GnuPG 2.1.

Remarque. Il est une bonne pratique de mettre une date d'expiration à sa clé, ainsi si on en perd l'accès, elle perd son pouvoir signant au bout d'un certain temps. Il est possible de modifier ce paramètre *a posteriori*

À l'issue de cette commande, vous serez en possession de votre paire de clé, que vous pouvez examiner via la commande :

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

Pour modifier votre clé, vous pouvez simplement utiliser la commande :

```
gpg --edit-key votre-identifiant-de-clé
```

Vous entrez ainsi dans un prompt qui servira à éditer votre clé. Pour rattacher une autre adresse e-mail par exemple, vous pouvez taper `adduid`.

2.2.3 Signer un message

Maintenant que vous avez une paire de clé toute fraîche rattachée à l'adresse de votre choix, vous pouvez commencer à l'utiliser pour signer vos messages (*rappel :* cela signifie garantir l'intégrité du message et de vous authentifier).

Pour générer votre message suivi de sa signature, il suffit de passer l'argument `--clearsign`.

```
gpg --clearsign <mail.txt
```

Ainsi en utilisant les *pipes linux*[1], vous pouvez simplement construire la signature de votre message et l'envoyer tel quel.

2.3 Interagir avec ses voisins

Maintenant qu'on a vu comment se servir de sa clé privée, nous allons voir comment utiliser les clés publiques des autres.

2.3.1 Échanger sa clé avec son voisin

Avant de pouvoir commencer à utiliser la clé publique de son voisin, il faut pouvoir échanger sa clé avec lui. Nous allons voir deux méthodes pour arriver à ce but.

En lui donnant sa clé publique directement pour exporter sa clé publique, il vous suffit d'écrire `gpg --export votre_nom` et de récupérer le résultat, par exemple dans le fichier `pubkey.asc` et le transmettre à votre voisin.

> **Attention**
>
> `gpg --export` exportera **tout votre trousseau de clé**, en particulier l'ensemble de votre réseau de confiance. C'est une erreur assez commune.

1. Pour plus de renseignements, voir `http://git.aliens-lyon.fr/initiations/linux`

Maintenant que vous avez chacun la clé de votre voisin, vous pouvez la rajouter à votre trousseau *via* la commande `gpg --import pubkey.asc` (sous réserve bien entendu que le fichier s'appelle `pubkey.asc`).

En passant par un serveur de clé plus haut dans le fichier de configuration, nous avions rajouté la ligne `keyserver hkp://pool.sks-keyservers.net`, elle permet de spécifier un serveur de clé pour rechercher les clés de ses camarades ou rajouter la sienne. La marche à suivre est quasiment la même : `gpg --send-key votre_nom` pour envoyer votre clé.

Afin de récupérer la clé de votre voisin, demandez-lui son nom, et cherchez sa clé publique : `gpg --search-keys son_nom`, et vérifiez l'identifiant de sa signature en le lui demandant (ce qui a la forme `0x0123456789ABCDEF0` avec notre configuration, dit identifiant long, par opposition à `0x9ABCDEF0` qui est appelé identifiant court, et qui peut être sujet à des *attaques par collision*).

Remarque. À la place des noms, nous aurions pu utiliser l'identifiant de clé.

Remarque. Si vous n'avez pas la ligne `keyserver xxx` dans votre fichier de configuration, il vous faudra spécifier le serveur de clé à utiliser à chaque fois (*via* l'argument `--keyserver xxx`)

> **Attention**
>
> L'ordre des arguments importe : si vous utilisez l'argument `--keyserver xxx`, alors mettez-le en premier argument, sinon les commandes précédentes n'auront pas accès à cette information.

2.3.2 Chiffrer mutuellement vos messages

Voilà, la dernière étape et la dernière chose qu'il nous manquait : « mais comment diantre chiffrer des messages ? »

Maintenant que vous avez votre trousseau de clé fraîchement garni, il ne vous reste plus qu'à choisir un destinataire parmi ceux-ci et de lui envoyer un message chiffré.

Comme pour la signature vous écrivez votre message dans un fichier et vous tapez la commande :

```
gpg --recipient son_nom --encrypt mail.txt
```

Si vous voulez chiffrer un mail pour plusieurs personnes, il suffit de multiplier de nombre de fois l'argument `--recipient` autant que nécessaire :

```
gpg --recipient destinataire1 --recipient destinataire2 --encrypt mail.txt
```

Il ne vous reste plus qu'à envoyer le contenu de mail.txt.asc à son (ses) destinataire(s).

Si vous voulez signer et chiffrer votre mail, il existe un alias pour cela :

```
gpg -se -r destinataire mail.txt
```

Remarque. Notez les tirets simples.

Pour choisir le fichier d'arrivée, l'argument est `--output`.

Remarque. Pour ponctuellement chiffrer un fichier, vous pouvez vous utiliser comme destinataire. Pour économiser de l'espace, il est aussi possible d'ignorer l'encodage binaire-vers-texte dont nous avons parlé précédemment avec l'argument `--no-armor`. Pour un chiffrement global du disque dur d'autres solutions existent.

2.3.3 Déchiffrer et vérifier les messages

Bon, voilà, on est désormais en possession d'un joli message chiffré/signé, mais on est incapable de le lire ou de le vérifier.

Pour demander à GnuPG de déchiffrer un message, il nous suffit d'écrire :

```
gpg --decrypt mail.txt.asc
```

De même pour vérifier si une signature est correcte, la commande est

```
gpg --verify mail.txt.asc
```

Dans le cadre d'un envoi de fichier, il est aussi possible de signer un fichier unique en dehors de ce fichier, c'est par exemple utilisé pour garantir que celui qui fournit un *paquet* sous linux est bien celui que l'on croit. Pour cela :

```
gpg --detach-sign mon_projet.tgz
```

Et pour vérifier la procédure est la même en spécifiant le fichier `.asc` à GnuPG, la seule contrainte est de laisser la signature avec le même nom que le fichier signé dans le même dossier. Ou alors spécifier le fichier à vérifier :

```
gpg --verify signature.asc mon_projet.tgz
```

2.3.4 Garantir sa sécurité : signer la clé de son voisin

On arrive presque à la fin de cet atelier, il ne nous reste plus qu'à dire au monde « j'ai vu Machin et je garantis que cette clé est bien à lui ! ». Pour faire ça, la commande est aussi intuitive que les autres :

```
gpg --sign-key son_nom
```

Maintenant il ne vous reste plus qu'à renvoyer la clé au serveur ou à son propriétaire (pour qu'il ait une version à jour de sa clé) qui n'aura plus qu'à la mettre à jour *via* `gpg --refresh-keys` pour la mise à jour des clés du serveur ou `gpg --import sa_cle`.

Nous pouvons maintenant signer mutuellement nos clés dans la joie et la bonne humeur.

En effet, on a vu en section 2.3.1 qu'il existait deux manières d'échanger ses clés avec son voisin : soit directement, ce qui permet d'avoir une garantie *absolue* de l'identité de son interlocuteur…mais qui est une méthode coûteuse, en effet si je veux envoyer un e-mail à mon correspondant canadien, je ne souhaite pas nécessairement aller au Canada pour être sûr que c'est lui ; ou alors en passant par un serveur de clés, mais on a vu que c'était simple de se rajouter à un serveur de clés, et on a *a priori* aucune garanties sur l'identité. C'est là qu'intervient *le réseau de confiance* : si un certain nombre de personnes (dont certaines que vous connaissez) assurent l'identité de quelqu'un, alors on peut, avec un certain degré de confiance, supposer que c'est bel et bien lui, sans le rencontrer directement.

2.4 Simplifier tout ça

On a donc vu comment utiliser GnuPG pour effectuer les actions de base de la cryptographie. Mais pour un usage quotidien cela peut s'avérer un peu lourd.

On pourrait utiliser les *arguments courts* (`-v` pour vérifier, `-s` pour signer, `-d` pour déchiffrer…) au prix de la lisibilité, mais il existe des solutions alternatives : certains clients mails supportent GnuPG nativement, comme `claws-mail` ou `mutt`.

Sous `Thunderbird`, il existe le plugin `Enigmail`. Une fois que votre trousseau est bien configuré, il n'y a plus grand-chose à faire.

Quant à la webmail, vous êtes obligés de passer par les commandes que nous avons vu. Cependant vous pouvez simplifier la procédure en utilisant la puissance des *pipes linux* par exemple via un gestionnaire de presse-papier (comme `xsel`) pour copier le texte et juste taper :

```
xsel | gpg -se -r destinataire | xsel
```

pour récupérer le mail signé venant du presse-papier dans son presse-papier.

2.4.1 Configurer enigmail

À l'installation d'enigmail, vous avez un assistant de configuration. Laissez les options par défaut pour le premier et second écran. Les troisième et quatrième écrans suivant vos préférences. Puis vous arrivez sur le cinquième : cliquez sur « Détails » et désactivez le troisième et le cinquième items. Et à l'écran suivant sélectionnez votre clé et c'est presque bon.

3 Pour aller plus loin

3.1 Révoquer votre clé

En cas de problèmes, il peut toujours être bon d'être en mesure de dire que votre clé n'est plus fiable.

Pour cela il vous faut générer une « anti-clé », autrement dit un certificat de révocation à garder précieusement, et de manière à y avoir accès même si vous perdez l'accès de vos données (cas typique où on voudrait révoquer c'est un vol d'ordinateur par exemple). Pour l'avoir il suffit de demander gentiment à gpg :

```
gpg --gen-revoke votre_nom
```

Note. c'est fait par défaut depuis GnuPG 2.1.

Au moment du drame, il ne vous reste plus qu'à importer le certificat de révocation et l'envoyer au serveur de clé comme vu précédemment.

> **Attention**
>
> Même si votre clé périme, **il ne faut pas la supprimer de vos fichiers**. Sans quoi vous ne serez plus en mesure de déchiffrer vos anciens messages.

3.2 Signature dans le message

Jusqu'à maintenant nous avons utilisé `PGP` sous sa forme *en clair*, ce qui signifie que si l'interlocuteur n'a pas un client supportant `PGP`, il verra la signature en plus de son message. Il y a moyen d'intégrer la signature au mail (visible ainsi uniquement dans le mail brut) et ainsi que cela soit transparent dans le cadre d'un client ne supportant pas la vérification de signatures.

Ceci peut être fait par les clients supportant `PGP` (par exemple `claws-mail`, ou `Thunderbird` + `Enigmail`) contre quelques réglages : « utiliser PGP/mime ».

3.3 Choisir la taille de sa clé

À la création de votre clé, il vous est demandé de choisir votre type de clé : `RSA/RSA` ou `El Gamal/DSA` puis la taille de la clé.

La principale différence entre les deux protocoles est l'assertion de sécurité sous-jacente : *RSA* (ou *factorisation* pour une hypothèse plus forte) pour … `RSA`, et *DDH* (ou *logarithme discret* pour une hypothèse plus forte) pour `El Gamal/DSA`. Les meilleures attaques connues contre le logarithme discret et la factorisation reposant sur le même algorithme (le *crible algébrique*), il n'y a pas trop de différences sur le choix de l'algorithme en soi.

La question se pose en revanche sur la taille de la clé à choisir. On suppose aujourd'hui que 1024 bits pour une clé asymétrique (en dehors des protocoles reposant sur des courbes elliptiques) est atteignable si on le veut vraiment (c'est-à-dire par la **NSA**), la recommandation 2048 bits est donc d'usage aujourd'hui.

Certains argumentent que les clés de 4096 bits sont un peu de la surenchère inutile et que si dans un futur proche on arrivait à casser les clés de 2048 bits, c'est que la percée est suffisamment importante pour qu'on ne soit plus à l'abri avec des clés de 4096 bits. Ainsi cela ne justifierait pas le surcoût calculatoire des clés de 4096 bits.

Au final cela dépend vraiment de l'usage qui en sera fait, et de la sensibilité des données en questions, cela est entièrement à votre convenance.

Pour avoir une idée des recommandations : `http://www.keylength.com/en/compare/`.

3.4 Documentation

Le *manuel* de gpg peut être une source d'inspiration si vous savez ce que vous souhaitez faire.

Sinon le Manuel Utilisateur de GnuPG peut aussi servir à trouver des idées :
`https://www.gnupg.org/gph/en/manual.html`

Les sources de ce fichier sont disponibles sur le dépot `git` d'AliENS :
`https://gitlab.aliens-lyon.fr/Initiations/Crypto`