Bien démarrer avec GnuPG

Damien Goutte-Gattat <dgouttegattat@incenp.org>

2020/05/21

Table des matières

1. Installation de GnuPG

Si vous utilisez GNU/Linux, GnuPG est dans les dépôts de toutes les distributions, et il est très souvent installé par défaut.

Il peut être utile néanmoins de vérifier que la version installée est bien issue de la dernière branche stable (2.2.x), et non de la branche 1.4.x (qui n'est maintenue que pour la compatibilité avec les versions de PGP datant des années 1990) ou des branches 2.0.x/2.1.x (qui sont obsolètes).

Il est possible de faire cohabiter GnuPG 1.4 et GnuPG 2.2 sur le même système ; dans ce cas, assurez-vous que la version que vous utilisez en temps normal est bien la 2.2.

En 2020, il semble que sur la plupart des distributions GNU/Linux, demander l'installation d'un paquet `gnupg` installe bien GnuPG 2.2, le binaire correspondant étant disponible sous le nom `gpg`. Il y a quelques exceptions, comme par exemple Fedora, où `gnupg` installe GnuPG 1.4 — il faut demander l'installation de `gnupg2` pour avoir GnuPG 2.2, le binaire correspondant étant appelé `gpg2`.

Dans le reste de cet article, je supposerai que `gpg` est le binaire de GnuPG 2.2 ; remplacez `gpg` par `gpg2` si besoin en fonction de votre distribution.

Sous Windows, Gpg4Win [https://gpg4win.org/download.html] est la distribution GnuPG de référence. La version 3.1.11, à l'heure où ces lignes sont écrites, fournit GnuPG 2.2.17.

Sous Mac OS X, GPG Suite [https://gpgtools.org/] est une distribution fournissant GnuPG 2.2.17. GnuPG est aussi disponible via MacPorts [https://www.macports.org/], sous le nom `gnupg2`.

2. Générer sa clef

Étape incontournable, la génération de la clef est malheureusement l'objet d'un volume considérable de désinformation. C'est l'étape où la plupart des « tutos » consacrés à GnuPG se fourvoient, et noient l'aspirant utilisateur sous une foule de « conseils » mal avisés, inutiles voire dangereux.

Alors, une bonne fois pour toutes : générer une clef, ça se fait en une seule étape, une seule commande :

```
$ gpg --gen-key
GnuPG doit construire une identité pour identifier la clef.

Nom réel : Alice
Adresse électronique : alice@example.org
Vous avez sélectionné cette identité :
    "Alice <alice@example.org>"

Changer le (N)om, l'(A)dresse électronique ou (O)ui/(Q)uitter ? o
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
gpg: clef 54B4CC7749CAE7C3 marquée de confiance ultime
gpg: revocation certificate stored as ↪
'/home/alice/.gnupg/openpgp-revocs.d/7685DC4214D727BB011BD6B754B4CC7749CAE7C3.rev'
les clefs publique et secrète ont été créées et signées.

pub   rsa2048 2020-05-13 [SC] [expires: 2022-05-13]
      7685DC4214D727BB011BD6B754B4CC7749CAE7C3
uid                      Alice <alice@example.org>
sub   rsa2048 2020-05-13 [E]
```

Oui, c'est tout. Ça n'a pas besoin d'être plus compliqué que ça. Non, il n'est pas nécessaire d'ajouter préalablement trente-cinq lignes dans le fichier de configuration de GnuPG pour changer les réglages par défaut. *La configuration par défaut de GnuPG est saine !* Les réglages par défaut ont été choisis comme tels pour de bonnes raisons. Vous ne devriez y toucher que si vous savez exactement *pourquoi* ils ne conviendraient pas à votre cas d'utilisation — pas parce qu'un *crypto-nerd* vous dit de le faire dans son billet de blog « générer une clef parfaite en dix-sept étapes ».

À quoi ressemble une clef générée en suivant les réglages par défaut ? C'est une clef principale RSA de 2048 bits, destinée aux opérations de signature, et une sous-clef de chiffrement similaire. Elle a une durée de validité de deux ans à compter de sa création, et est associée aux préférences d'algorithmes suivantes :

- pour le chiffrement : AES256, puis AES192, AES128, 3DES ;
- pour la condensation : SHA2-512, puis SHA2-384, SHA2-256, SHA2-224, SHA-1 ;
- pour la compression : ZLIB, puis BZIP2, ZIP, pas de compression.

Une fois encore, ces préférences par défaut (qui ne datent pas d'hier — la plupart de ces réglages datent de 2009/2010) sont saines et n'ont pas besoin d'être modifiées.

Éventuellement, si l'idée d'une clef RSA de 2048 bits vous fait tiquer (ça ne devrait pas) et si vous êtes sûrs que vos correspondants utilisent tous des implémentations suffisamment

modernes d'OpenPGP, vous pouvez opter pour une clef utilisant l'algorithme par défaut des prochaines versions de GnuPG[1] :

```
$ gpg --quick-gen-key 'Robert <bob@example.org>' future-default
Sur le point de créer une clef pour :
    "Robert <bob@example.org>"

Faut-il continuer ? (O/n) o
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
gpg: clef AC44CDC5733527A9 marquée de confiance ultime
gpg: revocation certificate stored as ↪
'/home/bob/.gnupg/openpgp-revocs.d/D7D0521F44673693DFFEB13FAC44CDC5733527A9.rev'
les clefs publique et secrète ont été créées et signées.

pub   ed25519 2020-05-13 [SC] [expires: 2022-05-13]
      D7D0521F44673693DFFEB13FAC44CDC5733527A9
uid                      Robert <bob@example.org>
sub   cv25519 2020-05-13 [E]
```

Vous obtenez alors une clef principale de signature de type `ed25519` et une sous-clef de chiffrement de type `cv25519` ; comme leur nom le laisse supposer, ces clefs sont basées sur la courbe elliptique dite « 25519 » (RFC 7748 [https://tools.ietf.org/html/rfc7748]).

La courbe 25519 ne fait pas encore partie du standard OpenPGP, qui officiellement ne supporte pour l'instant que les courbes P-256, P-384, et P-521 du NIST (RFC 6637 [https://tools.ietf.org/html/rfc6637]). Elle a néanmoins été ajoutée dès les premiers brouillons du RFC « 4880bis » (la prochaine version du standard) en 2016 et peut être utilisée dès maintenant sans crainte pour la compatibilité future.

En pratique, la courbe 25519 est supportée par la plupart des implémentations d'OpenPGP — à ma connaissance, au moins GnuPG (≥ 2.1), OpenPGP.js, Sequoia-PGP, et RNP. Déterminer si elle est supportée par « Broadcom Encryption Desktop » (héritier de Symantec PGP), en dénichant des informations techniques au milieu des arguments commerciaux de Broadcom, est laissé en exercice au lecteur.

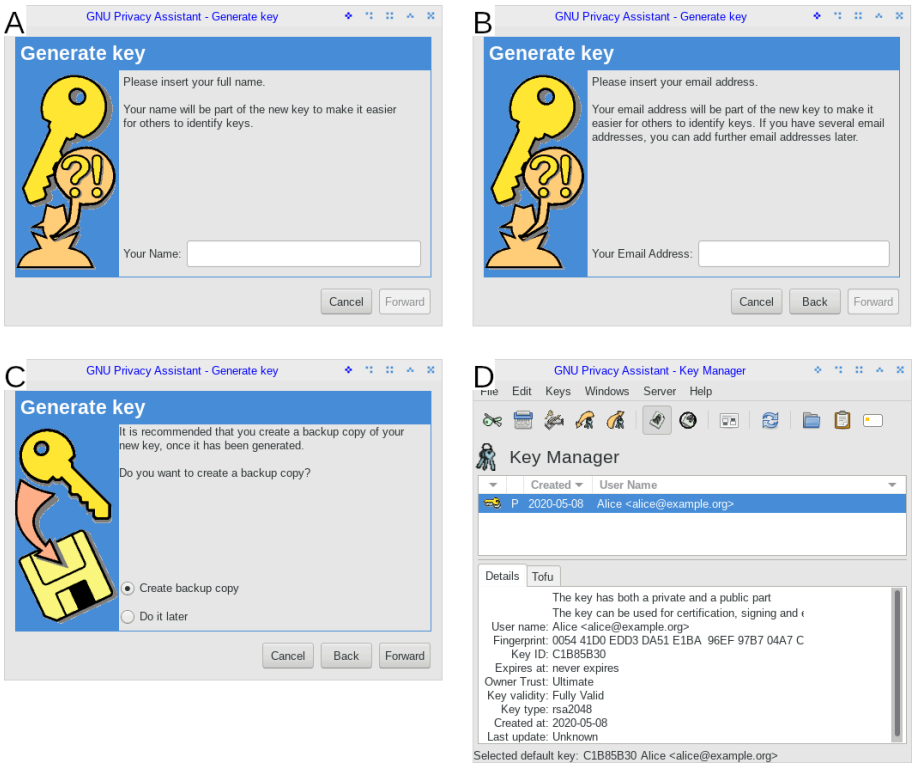
Concrètement, vous ne devriez pas rencontrer de problèmes si vous lchoisissez d'utiliser l'option `future-default`, à moins que certains de vos correspondants n'utilisent toujours GnuPG ≤ 2.0 — auquel cas essayez de les convaincre de se mettre à jour, ce sera beaucoup plus productif que de suivre un tutoriel vous recommandant de générer une clef RSA de 8192 bits.

Bien entendu, si la ligne de commande vous rebute, il est parfaitement possible de s'en passer. La manière exacte de générer une clef peut varier légèrement d'un frontal GnuPG à un autre (GPA, Seahorse, KGpg, etc.), mais les grandes lignes restent les mêmes et sont illustrées dans la Figure 1.

Dans le reste de cet article, je privilégierai la ligne de commande aux frontaux graphiques. Nul élitisme de ma part, c'est simplement que l'interface en ligne de commande est la même partout alors que chaque frontal a sa propre interface et qu'il n'est pas réaliste de les présenter tous — je reparlerai parfois de GPA parce que c'est le frontal que j'utilise, mais pour les autres, je vous renvoie à leur documentation.

[1] Rien n'est encore décidé du côté des développeurs de GnuPG concernant l'entrée en vigueur du nouveau choix d'algorithme par défaut ; le plus probable est que cela arrivera dans la prochaine branche 2.3.

Figure 1. Créer une clef avec GNU Privacy Assistant



Les quatre étapes pour créer une clef avec GNU Privacy Assistant (GPA), le frontal officiel de GnuPG: saisissez votre nom (A), votre adresse e-mail (B), choisissez si vous voulez une copie de sauvegarde de votre future clef (C), observez votre clef nouvellement créée (D).

3. Que faire après avoir créé sa clef ?

3.1. Sauvegarder les clefs

Sauf contraintes très particulières, vos clefs *doivent* être sauvegardées ; n'en gardez qu'une seule copie sur la machine où elles ont été créées et sont utilisées, c'est courir le risque de les perdre (et avec elles, les données dont elles dépendent) le jour où le disque dur rend l'âme ou la machine est perdue ou volée.

S'il est possible d'adopter pour les clefs publiques une stratégie de sauvegarde torvaldienne (*i.e.* les « mettre sur un serveur FTP et laisser le reste du monde en faire des mirroirs »), ce n'est évidemment pas le cas des clefs privées…

Vous pourriez être tenté de faire une simple archive du dossier `.gnupg`, qui contient tous les fichiers manipulés par GnuPG (dont les trousseaux). Toutefois ce n'est pas nécessairement une bonne idée : le contenu exact de ce dossier est considéré comme un détail d'implémentation interne à GnuPG, qui à ce titre est susceptible de changer au fil des versions (par exemple, la manière de stocker les clefs sur le disque a radicalement changé entre GnuPG 2.0 et GnuPG 2.1). Il est préférable de n'utiliser que l'interface publique de GnuPG, qui générera un fichier au format standard OpenPGP, dont il est garanti qu'il sera lisible par n'importe quelle version future de GnuPG (ou par une autre implémentation conforme du standard, indépendante de GnuPG).

Comme entr'aperçu en Figure 1, si vous avez crée vos clefs avec GPA, vous avez pu choisir de créer automatiquement une copie de sauvegarde dès le début. Sinon, utilisez la commande suivante pour créer une telle copie :

```
$ gpg -o backup.gpg --export-secret-keys alice@example.org
```

Contrairement à ce que le nom de la commande `--export-secret-keys` peut laisser supposer, le fichier *`backup.gpg`* ne contient pas que la partie secrète des clefs, mais aussi tous les éléments qui composent la clef publique. Ce fichier est donc suffisant à lui seul pour restaurer l'intégralité de vos clefs.

Plus tard, lorsque vous aurez commencé à utiliser GnuPG pour échanger avec vos correspondants, vous aurez à sauvegarder deux éléments supplémentaires : les clefs publiques de vos correspondants et la confiance que vous leur accordez. Vous pouvez utiliser pour ça les deux commandes suivantes :

```
$ gpg -o public-keys.gpg --export
$ gpg --export-ownertrust > trust.txt
```

Ce que vous faites ensuite de votre copie de sauvegarde est de votre ressort. Notez qu'elle contient vos clefs privées sous leur forme protégée par votre phrase de passe (sauf si vous avez choisi de ne pas avoir de phrase de passe lors de la création de la clef), donc pour peu que ladite phrase de passe soit assez robuste quiconque mettrait la main sur votre sauvegarde ne serait pas pour autant en mesure d'utiliser vos clefs.

Ah, et cela peut sembler évident, mais : ne stockez pas la copie de sauvegarde de votre clef privée sur un support chiffré avec la clef publique correspondante, qui nécessiterait la clef privée pour y accéder !

Une option de sauvegarde que j'apprécie particulièrement et que je recommande est celle de la sauvegarde sur papier. L'outil Paperkey [http://www.jabberwocky.com/software/paperkey/] est spécialement conçu pour ça : donnez-lui votre copie de sauvegarde et il en fera une version imprimable :

```
$ paperkey --secret-key backup.gpg | lpr
```

Avoir des sauvegardes, c'est bien. Savoir les utiliser le jour où on en a besoin, c'est mieux. Les trois commandes suivantes restaurent successivement votre propre clef (parties publiques et privées), les clefs publiques de vos correspondants, et les informations de confiance :

```
$ gpg --import backup.gpg
$ gpg --import public-keys.gpg
$ gpg --import-ownertrust < trust.txt
```

Si vous devez restaurer votre clef à partir de la sauvegarde sur papier générée par **paperkey**, numérisez le papier en question, passez-le à l'OCR pour obtenir un fichier texte (appelé *`frompaper.txt`* dans la commande ci-dessous), puis utilisez **paperkey** à nouveau pour reconstituer le fichier *`backup.gpg`* :

```
$ paperkey --pubring public-keys.gpg --secrets frompaper.txt --output backup.gpg
```

Notez l'utilisation du fichier *`public-keys.gpg`*, dans lequel **paperkey** vient trouver les parties publiques de votre clef (qui sont absentes de la version imprimable).

3.2. Mettre à l'abri le certificat de révocation

Un certificat de révocation vous permet de signaler à vos correspondants de ne plus utiliser votre clef publique, dans l'éventualité où vous ne seriez plus en mesure d'utiliser la clef privée correspondante — typiquement, soit parce que vous avez perdu la clef elle-

même (je vous avais bien dit de faire une copie de sauvegarde…), soit parce que vous avez oublié la phrase de passe qui la protège.

De nombreux tutoriels vous enjoignent à créer un tel certificat immédiatement après avoir créé votre clef. Toutefois, c'est inutile. En effet, GnuPG l'a déjà fait pour vous, comme vous l'avez peut-être remarqué dans les sorties console plus haut :

```
gpg: revocation certificate stored as ↪
'/home/alice/.gnupg/openpgp-revocs.d/7685DC4214D727BB011BD6B754B4CC7749CAE7C3.rev'
```

Vous le trouverez donc le dossier `.gnupg/openpgp-revocs.d`, dans un fichier nommé d'après l'empreinte de votre clef.

C'est probablement une bonne idée de stocker ce certificat ailleurs que sur la machine où vous avez déjà votre clef, puisque vous ne voulez pas perdre ce certificat en même temps que la clef elle-même. Attention, où que vous décidiez de le stocker, gardez à l'esprit que quiconque met la main dessus peut unilatéralement révoquer votre clef, sans avoir en sa possession la clef privée et sans la connaissance de la phrase de passe (c'est tout l'objet de ce certificat que de ne pas avoir besoin de la clef privée).

N'utilisez ce certificat de révocation *que* dans le cas où vous perdez l'usage de votre clef. Si vous en avez toujours l'usage et que vous souhaitez la révoquer pour une toute autre raison (par exemple, simplement parce que vous estimez qu'elle a fait son temps et que vous souhaitez la remplacer par une nouvelle, ou si vous craignez qu'elle n'ait été compromise), générez un certificat de révocation *ad hoc* au moment où vous en avez besoin :

```
$ gpg -o revcert.asc --generate-revocation alice@example.org

sec  rsa2048/54B4CC7749CAE7C3 2020-05-13 Alice <alice@example.org>

Faut-il créer un certificat de révocation pour cette clef ? (o/N) o
Choisissez la cause de la révocation :
  0 = Aucune cause indiquée
  1 = La clef a été compromise
  2 = La clef a été remplacée
  3 = La clef n'est plus utilisée
  Q = Annuler
(Vous devriez sûrement sélectionner 1 ici)
Quelle est votre décision ? 2
Entrez une description facultative, en terminant par une ligne vide :
>
Cause de révocation : La clef a été remplacée
(Aucune description donnée)
Est-ce d'accord? (o/N) o
Sortie forcée avec armure ASCII.
Certificat de révocation créé.

Veuillez le déplacer sur un support que vous pouvez cacher ; toute personne
accédant à ce certificat peut l'utiliser pour rendre votre clef inutilisable.
Imprimer ce certificat et le stocker ailleurs est une bonne idée, au cas où le
support devienne illisible. Attention tout de même : le système d'impression
utilisé pourrait stocker ces données et les rendre accessible à d'autres.
```

L'avantage d'un certificat de révocation *ad hoc*, par rapport à un certificat « générique » généré préventivement à la création de la clef, est double. D'une part, comme illustré dans la sortie ci-dessus, il permet de spécifier la raison motivant la révocation ; d'autre part, dans les cas où la clef est révoquée pour une raison autre qu'une compromission (choix 2 ou 3 ci-dessus : clef remplacée ou plus utilisée), les signatures émises par la clef antérieurement à la révocation resteront valables, alors que dans le cas d'une clef révoquée sans raison explicitement spécifiée, *toutes* les signatures jamais émises par la clef sont remises en cause (comme dans le cas où la clef a été compromise).

Pour utiliser un certificat de révocation, importez-le simplement dans votre trousseau avec **gpg --import**. Attention, importer un certificat de révocation ne demande pas de confirmation et est irréversible.

4. Diffuser la clef publique

Maintenant que vous avez une clef, il vous faut mettre la partie publique à disposition de vos correspondants.

4.1. Les serveurs de clefs SKS

Pendant longtemps, la méthode « classique » pour diffuser une clef publique a consisté à l'envoyer sur un des serveurs de clefs disponibles un peu partout sur Internet. Ces serveurs font typiquement partie d'un réseau au sein duquel les différentes instances se synchronisent régulièrement. Non seulement cela permet à l'utilisateur de ne pas se soucier du serveur auquel il s'adresse, mais cela fournit aussi une certaine résilience aussi bien face aux incidents (un des serveurs du réseau devient subitement inaccessible) que contre certaines interférences de gens mal intentionnés.

Malheureusement, aujourd'hui la survie à long terme du réseau des serveurs de clefs est incertaine. Il y a plusieurs raisons à cela, qui sortent du cadre de cet article, mais on peut citer pêle-mêle : quasiment pas de développeurs motivés pour travailler sur le logiciel serveur de référence (SKS-Keyserver [https://bitbucket.org/skskeyserver/sks-keyserver/wiki/Home]), un réseau entièrement dépendant de la bonne volonté des administrateurs (tous les nœuds sont gérés bénévolement), des désaccords dans la communauté sur les fonctionnalités que doit ou ne doit pas offrir un serveur de clefs, un principe de fonctionnement qui rend les serveurs vulnérables aux empoisonnements… C'est notamment ce dernier point qui a en 2019 porté un coup probablement fatal au réseau, avec une attaque par empoisonnement visant quelques membres influents de la communauté.[2]

Quelles que soient les raisons, le fait est qu'il n'y a plus à l'heure où j'écris ces lignes qu'à peine une vingtaine de serveurs dans le pool principal sks-keyservers.net [https://sks-keyservers.net/status/], contre facilement plus d'une centaine habituellement. Le réseau est toujours utilisable *pour l'instant*, mais il faut se préparer à ne plus pouvoir compter dessus dans un futur plus ou moins proche.

Pour l'instant donc, GnuPG est toujours configuré pour utiliser le pool sks-keyservers.net par défaut, alors tant que le pool est vivant et si vous acceptez une certaine incertitude sur la disponibilité, vous n'avez rien de particulier à faire. Pour envoyer votre clef sur un des serveurs du réseau, faites simplement :

```
$ gpg --send-keys 7685DC4214D727BB011BD6B754B4CC7749CAE7C3
```

où `7685DC4214D727BB011BD6B754B4CC7749CAE7C3` est l'empreinte de votre clef, telle qu'affichée par la commande **gpg -k *alice@example.org***.

4.2. Le serveur keys.openpgp.org

Un nouveau serveur de clefs a récemment vu le jour, en partie en réaction aux déboires du réseau SKS: keys.openpgp.org [https://keys.openpgp.org/]. Il utilise non pas SKS-Keyserver, mais Hagrid [https://gitlab.com/hagrid-keyserver/hagrid] (le « guardien des clefs » de Poudlard), un serveur de clefs basé sur Sequoia-PGP (une bibliothèque OpenPGP en Rust).

Bien qu'il puisse être considéré comme un remplaçant ou un successeur du réseau SKS, plusieurs différences importantes sont à noter avant de décider de l'utiliser.

[2] L'identité et les motivations de l'attaquant sont pour autant que je sache inconnues à ce jour, mais la nature ciblée de l'attaque pointe vers certains utilisateurs mécontents vis-à-vis du principe de fonctionnement des serveurs SKS et qui voulaient ainsi démontrer avec force que le réseau était vulnérable. Si tel est le cas, c'est stupide à plus d'un titre. Ils n'ont rien démontré que la communauté ne sache pas déjà (la vulnérabilité du réseau aux empoisonnements était bien connue) ; ils ont surtout réussi à saper les quelques bonnes volontés qui restaient parmi les développeurs et administrateurs SKS ; leur action est équivalente à « je vais mettre le feu à cet immeuble d'habitation, ça prouvera à tout le monde que le promoteur a utilisé un revêtement inflammable — je ferai ça de nuit quand les occupants dormiront, la démonstration sera plus convaincante ». S'ils me lisent : vous êtes des connards.

Il s'agit d'*un* serveur, non d'un pool. Même si n'importe qui peut monter son propre serveur Hagrid (tout comme avec SKS), il n'y a aucune synchronisation possible entre serveurs (contrairement à SKS, pour lequel c'est même une fonctionnalité majeure). Même si les développeurs affirment qu'une certaine décentralisation est prévue à l'avenir, ils préviennent que quelle que soit la forme que prendra cette décentralisation il ne sera pas question d'une « fédération ouverte » similaire à SKS (ce qui personnellement me fait m'interroger sur ce que peut être une fédération « fermée » — d'autant que contrairement à ce qu'ils semblent suggérer, n'entrait déjà pas dans le pool SKS qui veut). Le serveur `keys.openpgp.org` est donc un *single point of failure*, et un *single point of attack*.

Hagrid vérifie les adresses e-mail lorsqu'une clef est déposé sur le serveur, afin que seul le titulaire d'une adresse ne puisse déposer une clef associée à cette adresse. Le but étant à la fois de permettre à chacun de garder le contrôle sur ce qui est publié par le serveur, et d'offrir la garantie qu'une clef trouvée sur le serveur appartient bien à qui elle prétend appartenir (deux garanties jamais offertes par les serveurs classiques, par conception). C'est sympathique, mais cela implique que le serveur est de fait analogue à une autorité de certification à laquelle vous devez faire confiance, une idée qui personnellement ne me plaît guère.[3]

Dernier point et pas des moindres, Hagrid viole délibérément le standard OpenPGP dans certaines situations (notamment pour la diffusion des certificats de révocation), en diffusant des paquets de clef publique associés à aucune identité — ce que le standard ne permet pas. Cela conduit les implémentations conformes à refuser certains paquets en provenance d'un serveur Hagrid. Ce n'est pas un « bug de GnuPG » comme le laisse entendre la FAQ de `keys.openpgp.org`.

Cela étant dit, si vous souhaitez utiliser ce serveur pour y chercher des clefs, il vous suffit d'ajouter la ligne suivante dans le fichier `~/.gnupg/dirmngr.conf` :

```
keyserver hkps://keys.openpgp.org
```

et de relancer le démon réseau de GnuPG, dirmngr :

```
$ gpgconf --reload dirmngr
```

Pour *déposer* une clef en revanche, la commande **--send-keys** de GnuPG ne suffit pas, puisqu'elle ne permet pas à Hagrid de vérifier l'adresse (ce n'est pas prévu par le protocole HKP, utilisé par GnuPG derrière cette commande). À la place, il vous faut exporter la clef :

```
$ gpg -o alice.pub --export alice@example.org
```

puis télécharger le fichier *`alice.pub`* sur keys.openpgp.org/upload [https://keys.openpgp.org/upload], et suivre les instructions d'Hagrid pour procéder à la vérification d'adresse.

Si vous utilisez le greffon Enigmail pour Thunderbird, il utilise déjà `keys.openpgp.org` par défaut et implémente l'API spécifique de Hagrid (en remplacement du protocole HKP), lui permettant de procéder à la vérification d'adresse en arrière-plan sans intervention de l'utilisateur.

4.3. Autres méthodes de distribution

D'autres méthodes existent pour assurer la diffusion des clefs publiques, notamment la publication dans le DNS (DANE, RFC 7929 [https://tools.ietf.org/html/rfc7929]) et dans un *Web Key Directory* (WKD [https://www.ietf.org/id/draft-koch-openpgp-webkey-service-09.txt]). Pour ces deux méthodes je vous renvoie à un précédent article [https://in-

[3] L'intégrité des développeurs d'Hagrid et administrateurs de `keys.openpgp.org` n'est pas en cause. Ce sont des membres reconnus de la communauté OpenPGP, il ne fait aucun doute qu'ils sont dignes de confiance. C'est l'idée même de devoir faire confiance à une entité centralisée qui me dérange, indépendamment des personnes qui sont derrière.

cenp.org/dvlpt/docs/publication-clefs-openpgp/] ; néanmoins la mise en œuvre de DANE et de WKD dépend de celui qui contrôle le domaine de votre adresse e-mail (votre fournisseur d'accès à Internet ou de messagerie, votre employeur, votre université…) — à moins que vous ne disposiez de votre propre domaine, décider d'utiliser DANE ou WKD n'est pas vraiment de votre ressort.

Au-delà des méthodes formalisées de distribution (serveurs de clefs décentralisés ou non, DANE, WKD), vous pouvez (devez ?) aussi distribuer votre clef par tous les moyens possibles et imaginables. La poster sur un forum, sur votre blog, dans votre profil Facebook (il y a un champ dédié à cet usage), etc.

Là où il n'est pas pratique de publier la clef proprement dite faute de place, n'hésitez pas à publier *a minima* son empreinte : sur votre profil Twitter ou Mastodon (ou n'importe quel autre type de réseau dit « social »), sur vos cartes de visite, en signature de vos messages sur les forums où vous intervenez, etc.

Pour obtenir l'empreinte d'une clef (dont la vôtre), demandez simplement à GnuPG d'afficher ladite clef :

```
$ gpg -k alice@incenp.org
pub   rsa2048 2020-05-13 [SC]
      7685DC4214D727BB011BD6B754B4CC7749CAE7C3
uid           [ ultime ] Alice <alice@example.org>
sub   rsa2048 2020-05-13 [E]
```

5. Chiffrer, signer des fichiers

Cette section passe rapidement en revue la manière d'utiliser GnuPG sur des fichiers locaux, sans communication avec l'extérieur.

5.1. Chiffrer un fichier

Pour chiffrer un fichier à votre propre intention, la commande de base est la suivante :

```
$ gpg -r alice@example.org -e lorem.txt
```

L'option `-r` *`UID`* (ou `--recipient` *`UID`*) indique la clef publique avec laquelle chiffrer le fichier. Elle attend en paramètre l'identité (UID, *User ID*) associée à la clef que vous voulez utiliser. Ici, il s'agit de la vôtre (en supposant, comme dans tout le reste de cet article, que vous êtes Alice). Notez que vous n'avez pas nécessairement besoin de spécifier l'identité en entier, dès lors qu'il n'y a aucune ambiguïté : si votre trousseau de clefs publiques ne contient qu'une seule clef dont l'identité associée contient la chaîne « alice », alors `-r` *`alice`* sera suffisant (dans le cas contraire, si plusieurs clefs peuvent correspondre, GnuPG sélectionnera arbitrairement la première qu'il trouve dans le trousseau, qui ne sera peut-être pas celle que vous vouliez).

Si vous prévoyez de chiffrer des fichiers à votre intention assez fréquemment, vous pouvez envisager d'ajouter l'option `default-recipient-self` dans le fichier de configuration de GnuPG (`~/.gnupg/gpg.conf`) ; elle conduira GnuPG à sélectionner votre propre clef publique par défaut si vous ne spécifiez aucune clef explicitement. Avec cette option en place, la commande ci-dessus devient simplement :

```
$ gpg -e lorem.txt
```

Le paramètre `-e` (ou `--encrypt`) est la commande de chiffrement. Elle attend simplement le nom du fichier à chiffrer, dans le cas présent *`lorem.txt`*. Cette commande produira un fichier *`lorem.txt.gpg`*, contenant la version chiffrée du fichier précédent.

Quelle que soit l'opération que vous demandez à GnuPG (chiffrer, signer, déchiffrer), vous pouvez toujours utiliser l'option `-o` (ou `--output`) pour spécifier explicitement le nom du fichier de sortie.

5.2. Signer un fichier

Il y a trois commandes différentes pour effectuer une signature, qui correspondent aux trois types de signature possibles.

- La signature « standard » (faute d'un meilleur nom pour la désigner). Demandée avec la commande `-s` (ou `--sign`), elle produit un fichier contenant à la fois le document qui a été signé (enveloppé dans un paquet OpenPGP) *et* la signature correspondante.
- La signature en clair (*cleartext signature*), avec la commande `--clear-sign`. Elle produit aussi un fichier contenant à la fois le document original et sa signature, mais ici le document original n'est pas enveloppé dans un paquet OpenPGP et reste sous une forme textuelle lisible. Ce type de signature n'est utile que si le document original à signer est lui-même de type texte, une signature en clair sur un document binaire n'a aucun sens.
- La signature détachée, avec la commande `-b` (ou `--detach-sign`). Elle produit un fichier ne contenant *que* la signature, sans le document original ; lors de la vérification, le document original doit être fourni à GnuPG en même temps que la signature détachée. Ce type de signature est typiquement utilisé pour signer les archives de code source, l'avantage dans ce cas de figure étant que ceux qui ne sont pas intéressés par la vérification de la signature peuvent utiliser l'archive directement, sans avoir à l'extraire de son enveloppe OpenPGP comme dans le cas d'une signature « normale ».

Il est possible de signer *et* chiffrer en même temps en combinant les commandes `-s` et `-e`. Dans ce cas, le fichier produit contient à la fois le document chiffré et la signature correspondante. Ce n'est possible que pour les signatures « standard », combiner chiffrement et signature en clair ou détachée n'a pas de sens.

Toutes les commandes de signature utilisent par défaut la première clef trouvée dans le trousseau de clefs secrètes. La plupart des utilisateurs n'ont normalement qu'une seule clef secrète (compte non tenu des sous-clefs), donc ce comportement convient la plupart du temps. Si néanmoins vous avez plusieurs clefs secrètes, vous pouvez ajouter l'option `default-key` *`UID1`* dans le fichier de configuration de GnuPG pour toujours signer avec la clef associée à l'identité *`UID1`*, ou utiliser l'option `-u` *`UID2`* sur la ligne de commande pour ponctuellement signer avec la clef associée à l'identité *`UID2`*.

Les commandes de signatures produisent par défaut un fichier portant le même nom que le fichier à signer plus une extension dépendant du type de signature : `.gpg` pour une signature normale, `.asc` pour une signature en clair, `.sig` pour une signature détachée. Ces extensions sont purement conventionnelles et n'ont aucune signification pour GnuPG, qui identifie les fichiers qu'il manipule sur la base du type de paquets OpenPGP qu'ils contiennent et non sur leur extension.

5.3. Déchiffrer, vérifier un fichier

La commande `-d` (ou `--decrypt`) déchiffre un fichier :

```
$ gpg -d lorem.txt.gpg
gpg: chiffré avec une clef RSA de 2048 bits, identifiant 45EDD81BCE62E9BD, créée le 2020-05-13
      "Alice <alice@example.org>
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus
[...]
```

Notez l'asymmétrie entre les commandes de chiffrement (`-e`) et de déchiffrement (`-d`) : la première produit un fichier, tandis que la seconde écrit par défaut sur la sortie standard.

Si le fichier *`lorem.txt.gpg`* était un fichier chiffré et signé, GnuPG vérifiera la signature en même temps qu'il déchiffrera, et affichera le résultat à la fin de l'opération :

```
$ gpg -d lorem.txt.gpg
gpg: chiffré avec une clef RSA de 2048 bits, identifiant 45EDD81BCE62E9BD, créée le 2020-05-13
```

```
      "Alice <alice@example.org>
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus
[...]
gpg: Signature faite le mar. 19 mai 2020 23:23:21 BST
gpg:                avec la clef RSA 7685DC4214D727BB011BD6B754B4CC7749CAE7C3
gpg: Bonne signature de « Alice <alice@example.org> » [ultime]
```

La commande `-d`, malgré son nom, s'utilise aussi sur un fichier signé uniquement, pour vérifier la signature et extraire le document signé de son enveloppe OpenPGP et le rendre ainsi utilisable.

On utilisera la commande `--verify` pour vérifier une signature en clair…

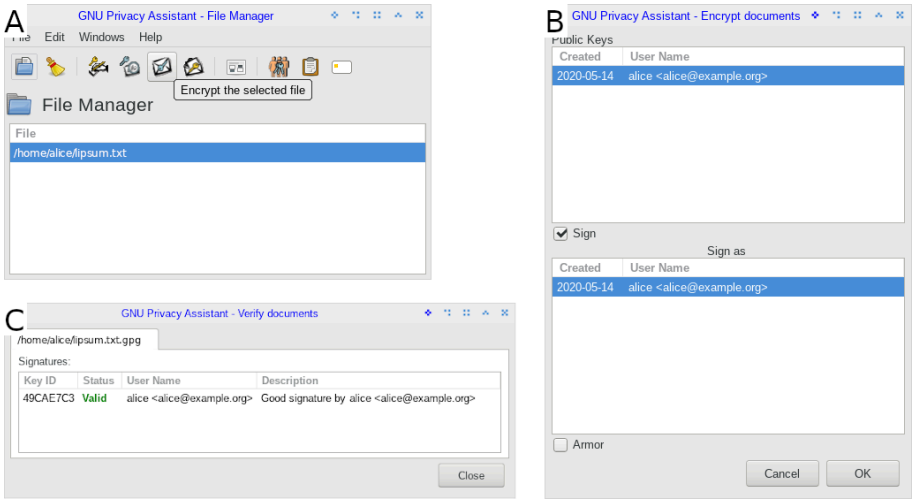
```
$ gpg --verify lorem.txt.asc
gpg: Signature faite le mar. 19 mai 2020 23:34:22 BST
gpg:                avec la clef RSA 7685DC4214D727BB011BD6B754B4CC7749CAE7C3
gpg: Bonne signature de « Alice <alice@example.org> » [ultime]
```

… ainsi que pour vérifier une signature détachée. Dans ce cas, GnuPG attend en premier argument le fichier de signature proprement dit, et en second argument le document original (en l'absence de cet argument GnuPG cherchera un fichier portant le même nom que le fichier de signature moins l'extension `.sig`, mais il est préférable de spécifier le fichier explicitement) :

```
$ gpg --verify lorem.txt.sig lorem.txt
gpg: Signature faite le mar. 19 mai 2020 23:41:52 BST
gpg:                avec la clef RSA 7685DC4214D727BB011BD6B754B4CC7749CAE7C3
gpg: Bonne signature de « Alice <alice@example.org> » [ultime]
```

Il est aussi possible d'utiliser la commande `--verify` sur une signature « standard », mais dans ce cas, GnuPG ne fera *que* vérifier la signature, sans extraire le document signé contrairement à ce que fait la commande `-d`.

Figure 2. Opérations sur des fichiers avec GNU Privacy Assistant



Exemples d'opérations sur des fichiers avec GPA. Pour chiffrer et un fichier, ouvrez-le dans le *File Manager* (A) et cliquez sur le bouton correspondant dans la barre d'outils. Dans la fenêtre de dialogue qui s'ouvre (B), sélectionnez la clef publique avec laquelle chiffrer le document, et éventuellement la clef privée avec laquelle le signer. En (C), un exemple du résultat d'une vérification de signature.

6. Chiffrer, signer des e-mails

Pour finir, il est temps d'utiliser GnuPG pour ce pour quoi il est en principe conçu (même si en vrai vous l'utilisez bien pour ce que vous voulez…), à savoir chiffrer et signer des e-mails.

Si vous n'avez pas d'amis, vous pouvez toujours envoyer un mail à *Edward*, un *bot* mis en place par la Free Software Foundation pour permettre aux gens d'essayer le chiffrement des e-mails.

6.1. Obtenir la clef d'Edward

La première étape pour établir une communication chiffrée, que ce soit avec Edward ou n'importe qui d'autre, est d'obtenir la clef publique de votre correspondant.

Cherchez donc la clef d'Edward sur les serveurs de clefs, et importez-là :

```
$ gpg --search-keys edward-fr@fsf.org
gpg: data source: http://pgp.surf.nl:11371
(1)     Edward, el simpático robot GnuPG <edawrd-es@fsf.org>
        Edward the GPG Bot <edward@fsf.org>
        Edward, the GPG Bot <edward-en@fsf.org>
        Edward, le gentil robot de GnuPG <edward-fr@fsf.org>
        […]
        2048 bit RSA key 9FF2194CC09A61E8, créé : 2014-06-29
Keys 1-1 of 1 for "edward-fr@fsf.org". ↪
Entre le ou les nombres, (S)uivant, ou (Q)uitter > 1
gpg: clef 9FF2194CC09A61E8 : clef publique « Edward, el simpático robot GnuPG <edward-es@fsf.org> »
gpg: Quantité totale traitée : 1
gpg:               importées : 1
```

Si vous avez choisi, dans la section relative aux serveurs de clefs, d'utiliser le nouveau serveur `keys.openpgp.org`, la clef d'Edward n'y est malheureusement pas disponible. Dans ce cas, vous pouvez utiliser l'option `--keyserver` sur la ligne de commande pour ponctuellement ignorer le serveur que vous avez configuré dans `~/.gnupg/dirmngr.conf` :

```
$ gpg --keyserver hkp://pool.sks-keyservers.net --search-keys edward-fr@fsf.org
```

En cas de problème avec le réseau SKS, une copie de la clef est incluse ci-dessous :

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBFOwfzoBCADpwK6sGC3EUzgD7IW1X5ZDR1nC5/rcXacAJLarPpvQBEz4pwjT
joAzATM7F9RwIzJ3hJTZHiYaQY4cfiGlKSnrd8GPC8A4QkxXIaQ0hLpcsBSbtZJp
o2iOzL2fRHmW2ZlnSHXPKbDwx5p0NcdQfjL9i2Yo31aLIO/Chhn5uyvIznOjaCSC
/O6x2C4m81Lu+B4UTDpl8y6ChtphUxyFGd7RXDXmkYQrxVqJbXKuSVmNMhM09myG
7iQ1l0YLOcCxa3IXDqQkte49BhMGB9wl4eDTE86HEzRjMtdhFpbTOW/+1N4XkOUV
y42HzGtmSAttojpIp00foNlWn1sn7JZJ18ojABEBAAG0NEVkd2FyZCwgbGUgZ2Vu
dGlsIHJvYm90IGRlIEdudVBHIDxlZHdhcmQtZnJAZnNmLm9yZz6JATgEEwECACIF
AlOxd9YCGwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEJ/yGUzAmmHorAwI
AI/THdk1Lj0IoYxGzOxGq1j2l1iRa2JcKNdsj0PzSpDHjtycCqJZrjBWAMJRymBt
WHeS6KLw992cEbmZ7wh0ObFXSicDTBOTAu3xwjNIRATAlH3f5nPNMnyULiNUbQin
naU7zOr/Iq88onb3FrjqhGETdxGXBm6RgoWGX7vdzdgY99CnC5bTt3TCu+9ddzVJ
NxS3yycwPl4IlRaSyQHQIMMRVs3p0e/8cqtZDBzsMOLPtoRGFoYBo3/pZWwF7kFX
n6u/z7UzuvW+COYtU2wIdSVkdmpn9jWl4+7UgKOXprDcQmrdGiCeErUxPbpWikza
VvVQkruAG9skMjuSk0Cmmai5AQ0EU7B/OgEIAMJfFthcYpgykvEHCBVm6vpMof1E
xuQ1bxNI1KVs2GTXF2sn9dXa6RvM6dz9xferglaZnsG+j7ACVaEHsgqe/E0VjhIS
NP1sJgH4dtyoL06dWp6Bs8SdI1Jasm+h55cXgYagahNpub1TUxjpsu8ZsyM/5cRp
B2HCmCXIsTYPEwIQu77XMpNo+mRi8oguOJ44ZMIYrvzivrJh1GnCbimSFfj7eMoF
1SHwl+e+k8reDqnoIWp514NGo9LLlwGIG0TQcg9S/tIchibmMZOV+xSS+MFxpMvm
eWCrgVJdK2paJ4d+8ZaxvkRDEtfGbmTOr7dFfA8i1heIPcbw4ejZEHGKWesAEQEA
AYkBHwQYAQIACQUCU7B/OgIbDAAKCRCf8hlMwJph6O+7CADBAYe5gTjFsA+vwVNt
gwrYXQv/w1XIndFUsOO3T7NjfTVETd652kIU4zFJRf3ebXbxz3E+1f1qPuVD8WJ9
5Roeyl8nsEoxr+iB6+/FqRIbHMnC0qqYRjVYvtD5ezgNpqGy/3dJmrhOuj7JHKIm
aB6tALq6JWb5URDHU5tCHPCyBJQhuMGBZzzyAexmBSk2CiKLX9DyX36ZO2+vlQK4
X0FW1M4qrC1gEB7sEpP9xTsST7MZr9USevwRcbRd/GvPFpTnI6JWazAmnhoRyOEA
ld6ORPNW1EUPBsIhfazP3v5SG5NXDAjYMHH/MbX872FhoBWerfHpi1yyZPHSkkXI
UAaY
=/0NJ
-----END PGP PUBLIC KEY BLOCK-----
```

Copiez-là dans un fichier puis importez-là manuellement :

```
$ gpg --import edward.asc
```

Une fois la clef d'Edward dans votre trousseau public, il faut la signer pour la marquer comme valide (je vous renvoie à un précédent article [https://incenp.org/dvlpt/docs/confiance-openpgp/index.html] pour plus de détails sur la notion de validité, et comment la validité d'une clef est affectée par les signatures). Pour ça, lancez l'éditeur de clefs sur la clef d'Edward :

```
$ gpg --edit-key edward-fr@fsf.org

pub  rsa2048/9FF2194CC09A61E8
     créé : 2014-06-29  expire : jamais      utilisation : SC
     confiance : inconnu       validité : inconnu
sub  rsa2048/469DDF6D9014D2D6
     créé : 2014-06-29  expire : jamais      utilisation : E
[ inconnue] (1). Edward, el simpático robot GnuPG <edward-es@fsf.org>
[ inconnue] (2)  Edward the GPG Bot <edward@fsf.org>
[ inconnue] (3)  Edward, the GPG Bot <edward-en@fsf.org>
[ inconnue] (4)  Edward, le gentil robot de GnuPG <edward-fr@fsf.org>
[…]
```

Affichez l'empreinte complète de la clef et vérifiez qu'elle correspond à l'empreinte dans la sortie ci-dessous :

```
gpg> fpr
pub   rsa2048/469DDF6D9014D2D6 2014-06-29 Edward, el simpático robotGnuPG <edward-es@fsf.org>
 Empreinte clef princip. : F357 AA1A 5B1F A42C FD9F  E52A 9FF2 194C C09A 61E8
```

Si l'empreinte correspond, vous pouvez procéder à la signature :

```
gpg> sign
Voulez-vous vraiment signer toutes les identités ? (o/N) o

pub  rsa2048/9FF2194CC09A61E8
     créé : 2014-06-29  expire : jamais      utilisation : SC
     confiance : inconnu       validité : inconnu
Empreinte clef princip. : F357 AA1A 5B1F A42C FD9F  E52A 9FF2 194C C09A 61E8

     Edward, el simpático robot GnuPG <edwardes@fsf.org>
     Edward the GPG Bot <edward@fsf.org>
     Edward, the GPG Bot <edward-en@fsf.org>
     Edward, le gentil robot de GnuPG <edward-fr@fsf.org>
     […]

Voulez-vous vraiment signer cette clef avec votre
clef « Alice <alice@example.org> » (54B4CC7749CAE7C3)

Voulez-vous vraiment signer ? (o/N) o

gpg> save
```

6.2. Thunderbird et Enigmail

Si vous utilisez le client e-mail Thunderbird [https://www.thunderbird.net/], vous devez (pour l'instant) installer le greffon Enigmail [https://enigmail.net/] pour y ajouter la prise en charge d'OpenPGP, Thunderbird ne supportant nativement que S/MIME. Cherchez Enigmail depuis le gestionnaire de greffons de Thunderbird, puis installez-le.

Enigmail ne fonctionnera plus à partir de Thunderbird 78, dont la sortie est prévue d'ici la fin de l'année 2020. À la place, la nouvelle version de Thunderbird prendra nativement en charge OpenPGP [https://blog.thunderbird.net/2019/10/thunderbird-enigmail-and-openpgp/] sans qu'un greffon ne soit nécessaire.

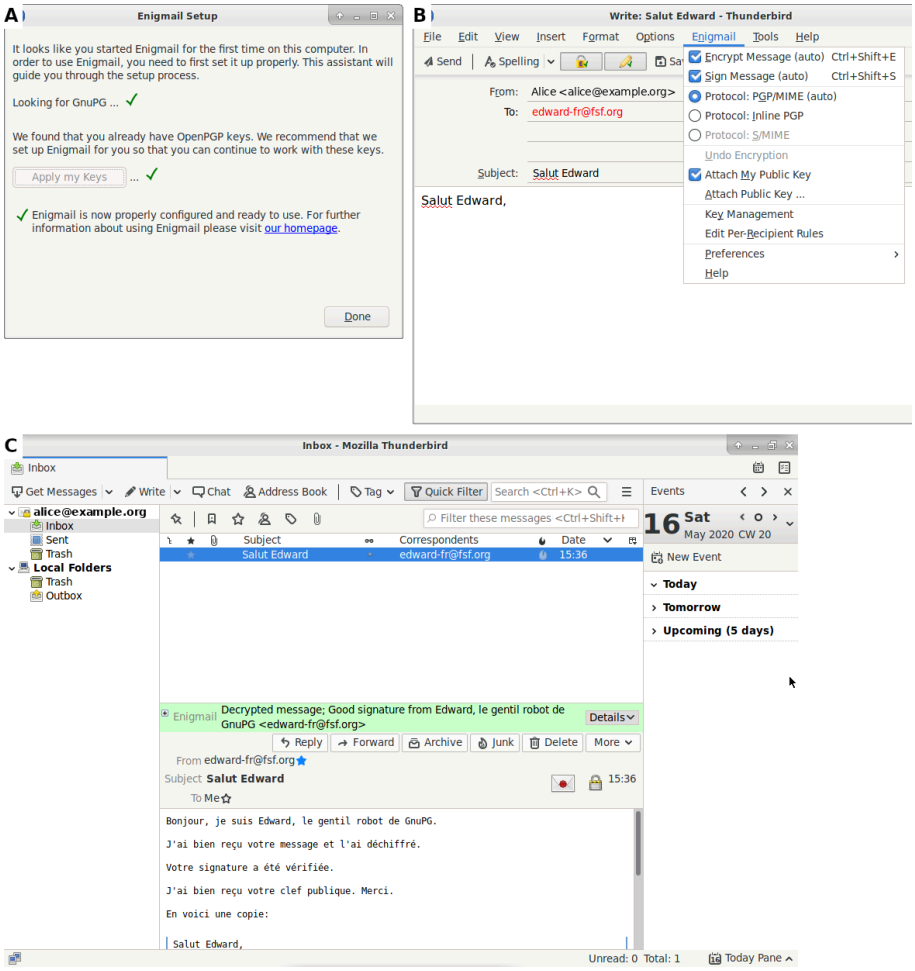
L'étendue de cette prise en charge native reste encore à voir, mais elle sera probablement, au moins dans l'immédiat, plus limitée [https://wiki.mozilla.org/Thunderbird:OpenPGP:2020] que ce qui est actuellement offert par Enigmail.

Une chose semble déjà sûre, Thunderbird n'utilisera pas les trousseaux de GnuPG (pas plus le trousseau public que le trousseau privé). Il sera possible d'importer les clefs de GnuPG vers Thunderbird, mais une fois cela fait, les clefs importées seront gérées par Thunderbird indépendamment de GnuPG — toute modification faite dans GnuPG sera invisible depuis Thunderbird et inversement.

À titre personnel, je trouve que c'est une très mauvaise idée. Et je suis globalement assez dubitatif de l'approche consistant à jeter à la poubelle une implémentation pleinement fonctionnelle pour après coup se rendre que compte que « oh ben zut, il y avait des fonctionnalités utiles en fait, bon comment on fait pour les ré-implémenter *from scratch* maintenant ? », comme par exemple avec le support des cartes OpenPGP [https://mail.mozilla.org/pipermail/tb-planning/2019-December/007288.html].

L'avenir dira si cette orientation aura permis d'attirer de nouveaux utilisateurs ou si elle aura surtout fait fuir les utilisateurs déjà existants. /rant

Figure 3. Utilisation d'Enigmail dans Thunderbird



(A) Sitôt installé, Enigmail détecte votre installation de GnuPG et vos clefs pré-existantes. (B) Envoi d'un e-mail chiffré et signé à Edward, auquel on joint une copie de sa propre clef publique. (C) Réception et déchiffrement de la réponse automatique d'Edward.

Une fois Enigmail installé, il devrait automatiquement détecter GnuPG et vous proposer de se configurer pour utiliser la clef que vous avez déjà ; acceptez en cliquant sur *Apply my keys* (Figure 3A).

Si vous installez Enigmail sans avoir préalablement généré vous-même votre clef, Enigmail en générera silencieusement une pour vous mais se configurera automatiquement pour utiliser *p≡p* plutôt que GnuPG. Sans rentrer dans les détails, p≡p [https://www.pep.security/] est une solution de chiffrement opportuniste, basée entre autres sur OpenPGP mais où, en gros, l'utilisateur ne contrôle plus rien… L'utilisation de p≡p a parfois été appelée *Junior mode*, mais ce terme ne semble plus apparaître dans la documentation ou l'interface de Enigmail.

Vous pouvez maintenant envoyer un mail à `edward-fr@fsf.org` (Figure 3B). Par défaut le message sera signé, et comme la clef publique d'Edward est déjà dans votre trousseau, Enigmail devrait reconnaître son adresse et aussi activer le chiffrement. Si vous voulez qu'Edward puisse chiffrer également sa réponse à votre intention, pensez à joindre votre propre clef publique au message, en utilisant l'option appropriée dans le menu d'Enigmail.

Lors de l'envoi, il se peut qu'Enigmail vous propose une option appelé *protected headers*, qui vise à chiffrer certains en-têtes du message (notamment l'objet) et non seulement le corps du message. Je déconseille personnellement cette option qui repose sur une proposition de standard [https://www.ietf.org/id/draft-autocrypt-lamps-protected-headers-02.txt] à mon sens beaucoup trop complexe pour le bénéfice qu'elle apporte, et dont le support par un grand nombre de clients de messagerie est incertain. Gardez plutôt à l'esprit que les en-têtes ne sont *pas* chiffrés,[4] et abstenez-vous de divlguer trop d'informations dans les objets de vos messages.

Après quelques minutes, vous devriez recevoir une réponse automatique d'Edward. Vous aurez ainsi confirmation que tout s'est bien passé (Figure 3C). Félicitations, vous venez de procéder à votre premier échange d'e-mails chiffrés.

6.3. (Neo)Mutt

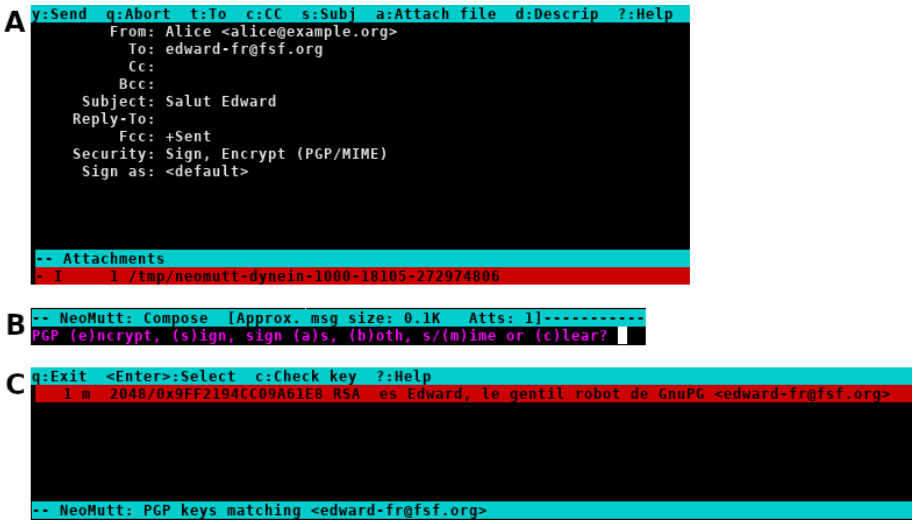
Si vous utilisez le client Mutt [http://mutt.org/] ou le *fork* Neomutt [https://neomutt.org/], tous deux prennent nativement en charge OpenPGP et ne nécessitent qu'un minimum de configuration. Ajoutez simplement les deux lignes suivantes à votre fichier `mutrc` :

```
set crypt_use_gpgme = yes
set pgp_default_key = 0x7685DC4214D727BB011BD6B754B4CC7749CAE7C3
```

La première ligne configure (Neo)Mutt pour utiliser la bibliothèque GpgME pour interagir avec GnuPG (ce qui est la manière recommandée par les développeurs de GnuPG), au lieu d'appeler le binaire `gpg` directement (ce qui est toujours possible mais *error-prone*). La seconde ligne indique la clef à utiliser par défaut ; cette clef sera utilisée pour signer vos messages, et pour en chiffrer une copie à votre intention (afin que vous puissiez toujours déchiffrer les messages envoyés à des tiers). Remplacez la valeur indiquée par l'empreinte de votre propre clef.

[4] Du moins, pas chiffrés de bout en bout par OpenPGP. Si le message est acheminé à travers des connexions SMTP chiffrées par TLS (ce qui est de plus en plus courant aujourd'hui), les en-têtes sont chiffrés au même titre que le reste du message, en mode point-à-point.

Figure 4. Utilisation de Neomutt



(A) Préparation d'un message chiffré et signé dans Neomutt. (B) Le menu PGP permettant de sélectionner si un message doit être chiffré, signé, signé avec une clef autre que celle configurée par défaut, chiffré *et* signé. (C) Le menu de sélection de la clef publique du destinataire.

Une fois (Neo)Mutt configuré, envoyez donc un mail à Edward. Préparez le message de la manière habituelle jusqu'à arriver à la fenêtre d'envoi (Figure 4A). Par défaut le message sera seulement signé, appuyez sur la touche p pour appeler le menu PGP (Figure 4B) puis sur la touche b comme indiqué pour demander que le message soit chiffré et signé.

Attachez au message une copie de votre clef publique en tapant Esc+k et en sélectionnant votre clef dans le menu correspondant. Au moment d'envoyer le message, vous serez invité à sélectionner la clef publique avec laquelle chiffrer le message (Figure 4C), parmi les clefs associées à une adresse correspondant à celle du destinataire du message. Sélectionnez la clef d'Edward (qui logiquement devrait être la seule, vous ne devriez pas avoir plus d'une clef associée à l'adresse `edward-fr@fsf.org`), puis envoyez.

Attendez quelques minutes de recevoir la réponse d'Edward et vérifiez que tout s'est bien passé.

6.4. Autres clients

Il ne serait pas réaliste de vouloir couvrir tous les clients e-mail existants et je n'essayerai même pas.

Beaucoup de clients libres ont un support natif pour OpenPGP : GNOME Evolution, KMail, Claws-Mail… Ils n'opposent pas de difficultés majeures.

Sous Windows, la distribution Gpg4Win déjà mentionnée dans la première section est fournie avec GpgOL, un greffon pour Microsoft Outlook.

Sous Mac OS X, GPG Suite fournit un greffon pour Apple Mail appelé *GPG Mail*. Attention, ce greffon est libre (comme le reste de GPG Suite) mais, depuis peu, payant.

Sous Android, OpenKeychain [https://www.openkeychain.org/] apporte la prise en charge d'OpenPGP à plusieurs applications, dont le client de messagerie K-9 Mail [https://k9mail.github.io/].

À titre personnel, il est hors de question que j'accepte de stocker une quelconque clef privée sur un téléphone Android. Je ne recommande l'utilisation

d’OpenKeychain que couplée à un jeton cryptographique comme la Yubibey 5 NFC [https://www.yubico.com/product/yubikey-5-nfc].

A. À propos de ce document