De la gestion des clefs OpenPGP

Damien Goutte-Gattat <dgouttegattat@incenp.org>

2015/05/17

Résumé

Cet article aborde plusieurs questions relatives à la bonne gestion des clefs OpenPGP.

Table des matières

1. Faut-il changer les algorithmes préférés ?

Chaque certificat OpenPGP contient, entre autres choses, une sélection d'algorithmes préférés permettant au titulaire du certificat d'annoncer les algorithmes à utiliser pour communiquer avec lui. Plusieurs [https://help.riseup.net/en/security/message-security/openpgp/best-practices#stated-digest-algorithm-preferences-must-include-at-least-one-member-of-the-sha-2-family-at-a-higher-priority-than-both-md5-and-sha1] documents [https://alexcabal.com/creating-the-perfect-gpg-keypair/#strengthening-hash-preferences] sur Internet [http://www.bortzmeyer.org/nouvelle-cle-pgp.html] suggèrent que la sélection par défaut de GnuPG est bancale et doit être corrigée par l'utilisateur. Par exemple, une critique qui revient souvent est que l'algorithme de condensation préféré par défaut serait SHA-1.

Ce n'est plus vrai depuis… GnuPG 2.0.13 [http://git.gnupg.org/cgi-bin/gitweb.cgi?p=gnupg.git;a=commitdiff;h=e50cac1d848d332c4dbf49d5f705d3cbbf074ba1], il y a bientôt six ans.

En fait, si on regarde les préférences par défaut sur une clef fraîchement générée :

```
$ gpg2 --edit-key alice
[…]

gpg> showpref
[ultimate] (1). Alice <alice@example.org>
     Cipher: AES256, AES192, AES, 3DES
     Digest: SHA256, SHA384, SHA512, SHA224, SHA1
     Compression: ZLIB, BZIP2, ZIP, Uncompressed
     Features: MDC, Keyserver no-modify
```

on constate que ces préférences sont plutôt « saines » et ne nécessitent pas, à mon avis, que l'on recommande systématiquement aux utilisateurs de les changer (pour chipoter, je ferais bien passer AES128 _avant_ AES256 et AES192, mais c'est un détail). La présence de 3DES et SHA1 peut surprendre (à quoi bon les utiliser si AES et la famille SHA2 sont disponibles ?), mais ces algorithmes sont les seuls imposés par le standard OpenPGP[1] et ils garantissent l'interopérabilité avec d'autres implémentations du standard (pas la peine d'essayer de les supprimer de la liste des préférences, GnuPG les rajoutera implicitement).

Donc, à moins que vous n'ayez des opinions bien arrêtées sur ce que valent les différents algorithmes cryptographiques, faites confiance à la sélection par défaut de GnuPG.

Détail amusant, même Bruce Schneier n'a pas jugé utile de changer les préférences par défaut de GnuPG pour sa clef [https://www.schneier.com/contact.html], même pas pour y insérer ses propres algorithmes *Blowfish* (qui fut en compétition avec Rijndael pour devenir AES) ou *Twofish*.

Attention néanmoins, si vous avez générée votre clef il y a longtemps, sa sélection d'algorithmes préférés est peut-être effectivement dépassée. Pour la mettre automatiquement à jour, utilisez la commande `setpref` sans arguments dans l'éditeur de clef de GnuPG.

2. Clef primaire et sous-clefs

Lors de la création d'une nouvelle clef, le comportement par défaut de GnuPG est de créer en réalité *deux* clefs : une clef primaire de certification et de signature, et une sous-clef de chiffrement.

L'utilisation de sous-clefs se justifie d'abord pour une simple raison technique : à l'exception notable du très flexible RSA, la plupart des algorithmes à clef publiques utilisés par OpenPGP permettent *soit* de signer (comme DSA ou ECDSA) *soit* de chiffrer (comme ElGamal ou ECDH), mais ils ne permettent pas de faire les deux. Cela impose donc d'utiliser des clefs distinctes pour les opérations de signature/certification et de chiffrement.

Mais un autre avantage des sous-clefs est qu'elles peuvent être remplacées à tout moment indépendamment de la clef primaire dont elles dépendent.

La clef primaire est la clef qui est associée à vos identités, c'est celle que tout le monde connaît et surtout c'est celle que tout le monde *signe*. Révoquer cette clef et la remplacer par une autre revient à perdre toutes les signatures péniblement accumulées au fil du temps, et à disparaître virtuellement de la toile de confiance. Il faut alors faire signer sa nouvelle clef pour se ré-introduire dans la toile — quelque chose de fastidieux que l'on souhaite avoir à faire le moins souvent possible.

À l'inverse, une sous-clef peut être révoquée et remplacée à tout moment, cela n'affecte aucunement la validité et la réputation de la clef primaire, et c'est en plus transparent pour les utilisateurs qui de toute façon ne manipulent jamais directement les sous-clefs : si votre interlocuteur veut vous envoyer un message chiffré, il n'a besoin que de sélectionner votre clef primaire, c'est son implémentation d'OpenPGP qui se chargera de trouver la sous-clef de chiffrement.

En bref, le principe des sous-clefs permet de dissocier la clef qui établit votre identité et vous représente au sein de la toile de confiance, des clefs que vous utilisez concrètement.

Donc, ne vous contentez pas de la sous-clef de chiffrement générée par défaut par GnuPG, et générez en plus une seconde sous-clef pour les opérations de signature. Utilisez la clef primaire exclusivement pour signer des *clefs*.

[1] La première version du standard [http://tools.ietf.org/html/rfc2440], en 1998, demandait que MD5 soit aussi pris en charge aux côtés de SHA1, mais la version suivante [http://tools.ietf.org/html/rfc4880] en 2007 a déclaré cet algorithme obsolète. SHA1 pourrait subir le même sort si une troisième version devait voir le jour.

2.1. Sous-clefs de compatibilité

Il n'y a, en principe, pas de raison d'avoir plus d'une sous-clef de chiffrement et une sous-clef de signature.

Une raison plausible et intéressante, toutefois, est d'assurer une certaine compatibilité avec des implémentations plus anciennes d'OpenPGP.

Imaginons la situation suivante : Alice utilise GnuPG 2.1, et elle aimerait pouvoir utiliser les algorithmes à base de courbes elliptiques (ECC, *Elliptic Curve Cryptography*) que ce dernier prend en charge. Malheureusement, si Bob utilise aussi GnuPG 2.1, Charlie, lui, utilise encore GnuPG 2.0, qui ne connaît pas les courbes elliptiques. Si Alice génère des clefs ECC, elle se coupe de Charlie ; si elle génère des clefs RSA, elle ne se coupe de personne, mais personne ne profite de l'introduction d'ECC dans GnuPG — du coup, pas grand'monde n'est motivé pour passer à GnuPG 2.1, ce qui freine d'autant plus le déploiement des clefs ECC dont tout le monde dit pourtant qu'elles sont l'avenir parce que RSA, c'est has-been.[2]

Les sous-clefs offrent une solution possible à ce blocage. Si Alice a une clef primaire RSA, elle peut avoir une sous-clef de chiffrement RSA *et* une sous-clef de chiffrement ECDH. Quand Charlie voudra chiffrer un message à destination d'Alice, son GnuPG 2.0 sélectionnera automatiquement la sous-clef RSA en ignorant la sous-clef ECDH qu'il ne sait pas utiliser. Le GnuPG 2.1 de Bob, en revanche, pourra utiliser la sous-clef ECDH (petite subtilité : il faudra que la sous-clef ECDH ait été générée *après* la sous-clef RSA, puisque GnuPG sélectionne automatiquement la sous-clef la plus récente lorsqu'il en trouve plusieurs utilisables). Ainsi, Alice ne se coupe de personne, et peut utiliser les algorithmes les plus récents avec ceux qui les acceptent.

Cette solution est aussi applicable aux opérations de signature. Dans ce cas, comme le signataire ne sait pas forcément *qui* vérifiera la signature, la compatibilité maximale peut être assurée en signant systématiquement à la fois avec une sous-clef RSA ou DSA *et* avec une sous-clef ECDSA.

À terme, lorsque GnuPG 2.1 (ou d'autres implémentations d'OpenPGP prenant en charge les courbes elliptiques, comme Google End-to-end) sera suffisamment répandu, les sous-clefs RSA pourront éventuellement être révoquées.

J'ai expérimenté cette notion de « sous-clefs de compatibilité » à l'occasion de l'écriture de ce journal, mais je ne l'utilise pas en pratique. Je n'ai pas de clefs ECC, donc le problème de compatibilité avec GnuPG 1.4 et 2.0 ne se pose pas à moi.

3. Faut-il faire expirer les clefs ?

Par défaut, les nouvelles clefs OpenPGP créées par GnuPG 2.1 n'ont pas de date d'expiration et sont donc valides *ad vitam aeternam* jusqu'à ce qu'elles soient éventuellement révoquées. Est-ce une bonne idée ? Faudrait-il spécifier une date d'expiration, et le cas échéant, quelle devrait être la durée initiale de validité ?

Je n'ai pas de réponse absolue à ces questions, mais il faut noter deux choses pour commencer. D'une part, GnuPG est assez psychorigide vis-à-vis de l'expiration et refusera catégoriquement de vous laisser utiliser une clef expirée (ce comportement est non-débrayable, ce qui est sujet à débats [https://lists.gnupg.org/pipermail/gnupg-users/2014-September/050850.html] dans la communauté). D'autre part, la date d'expiration n'est pas gravée dans le marbre : tant que vous avez accès à la clef primaire privée, vous pouvez à tout moment changer la date d'expiration à votre guise (y compris ajouter une

[2] Pour ce que ça vaut, je ne partage absolument pas cette opinion. Je suis même plutôt sceptique vis-à-vis des algorithmes ECC et surtout des courbes dont les paramètres ont été choisis sur des critères non-spécifiés.

date d'expiration à une clef qui était initialement éternellement valide, ou inversement repousser la date d'expiration initiale aux calendes grecques). Cela se fait en re-signant votre propre clef.

Cela étant dit, quel est l'intérêt de faire expirer une clef ?

Sur une clef primaire, qui ne peut être révoquée que par la clef privée correspondante,[3] l'intérêt se manifeste dans le cas où vous perdez d'une façon ou d'une autre l'usage de votre clef privée, et que n'avez pas (ou plus) sous la main le certificat de révocation qui avait été généré en même temps (par exemple, une défaillance de votre disque dur vous fait perdre tout votre répertoire `~/.gnupg`, et quant à vos sauvegardes… « euh, quelles sauvegardes ? »). L'expiration permettra dans ce cas d'éviter de laisser en circulation une clef éternellement valide mais en pratique inutilisable.

Notez que l'expiration n'a aucun intérêt en cas de *vol* de votre clef privée, puisque votre attaquant n'aura qu'à utiliser la clef volée pour re-signer votre clef publique et repousser la date d'expiration.

Sur une sous-clef, l'intérêt est à mon sens plus réduit. Si vous perdez l'usage d'une sous-clef, vous pourrez toujours révoquer la clef publique correspondante (et donc signaler à vos interlocuteurs de ne plus l'utiliser) avec votre clef *primaire*. En fait, même si vous ne révoquez pas la clef inutilisable, le simple fait de générer une nouvelle sous-clef devrait suffire pour que vos interlocuteurs cessent d'utiliser l'ancienne, puisqu'en présence de plusieurs sous-clefs GnuPG utilisera systématiquement la plus récente. Faire expirer ou révoquer les anciennes clefs permet juste d'exprimer *explicitement* votre volonté de ne plus voir ces clefs utilisées.

Je suggérerai personnellement (sans aller néanmoins jusqu'à prétendre que c'est une bonne pratique) de faire expirer votre clef primaire tous les deux ou trois ans. Ça vous oblige à la re-signer périodiquement, ce qui d'une part vous donne l'occasion de vérifier qu'elle est bien à jour (par exemple, elle ne contient pas une identité que vous n'utilisez plus depuis des années, et que vous aviez complètement oublié), et d'autre part ré-affirme auprès des autres utilisateurs que cette clef est toujours d'actualité et que vous continuez à vous en servir.

4. Où et comment stocker ses clefs privées ?

À part les attaques cryptanalytiques, le principal risque auquel sont exposées les clefs privées est celui de l'*exfiltration*, où un attaquant obtient une copie de vos clefs privées. Le stockage des clefs doit tenir compte de ce risque.

GnuPG stocke les clefs privées dans le dossier `~/.gnupg/private-keys-v1.d`, à raison d'un fichier par clef, ou dans le fichier `~/.gnupg/secring.gpg` (GnuPG 1.4/2.0). La restriction de l'accès à ces fichiers est du ressort du système d'exploitation, GnuPG n'a pas grand'chose à voir là-dedans — tout ce qu'il fait, lors de la première utilisation, est de s'assurer que le dossier `$GNUPGHOME` a les permissions appropriées, soit 0700.

Les clefs sont chiffrées par une clef AES de 128 bits dérivée de votre phrase de passe par *n* itérations d'une fonction de condensation, *n* étant déterminée empiriquement pour que la dérivation prenne environ 100 millisecondes — ceci afin de rendre la recherche exhaustive de la phrase de passe trop coûteuse. Par ailleurs, une clef n'est déchiffrée que juste avant d'être utilisée, et la clef déchiffrée est supprimée de la mémoire immédiatement après utilisation.

Un attaquant souhaitant s'emparer de vos clefs a donc deux obstacles à surmonter : il doit accéder aux fichiers contenant les clefs (par effraction physique ou, plus probablement,

[3] C'est un petit mensonge : il est en réalité possible de désigner des « révocateurs », des tiers de confiance que vous autorisez à révoquer votre clef à votre place dans l'éventualité où vous perdriez l'usage de votre clef primaire privée.

par piratage informatique), et déchiffrer celles-ci (soit en cassant la clef AES, soit en trouvant la phrase de passe).

Toutefois, dans un scénario où l'attaquant a infiltré votre machine pour exfiltrer les fichiers contenant les clefs privées, il n'est pas difficile d'imaginer qu'il a aussi installé un *keylogger* pour récupérer en même temps votre phrase de passe.

L'exposition de vos clefs à ce dernier scénario peut être réduite en les stockant *hors-ligne*, au lieu de les laisser sur votre machine.

4.1. Stockage hors-ligne de la clef primaire

Si vous avez une sous-clef de chiffrement et une sous-clef de signature comme recommandée plus haut, vous n'avez plus besoin de la clef primaire que pour modifier votre propre clef (ajouter/révoquer une identité ou une sous-clef, changer les préférences) et signer les clefs d'autres utilisateurs. Du coup, vous ne perdrez pas grand'chose en confort d'utilisation à conserver la clef primaire privée exclusivement hors ligne, hors de portée de toute attaque informatique.

Si la procédure pour cela était relativement « tordue » avec GnuPG 1.4/2.0 (il n'était pas possible de supprimer uniquement la clef primaire du trousseau privée, de sorte qu'il fallait, en gros : ① exporter les sous-clefs uniquement, ② supprimer la clef primaire, ce qui supprimait *aussi* les sous-clefs rattachées, puis ③ ré-importer uniquement les sous-clefs — et encore je ne parle pas de la procédure à suivre pour _utiliser_ la clef hors-ligne…), c'est devenu beaucoup plus simple avec GnuPG 2.1.

Trouvez le *keygrip* de votre clef primaire :

```
$ gpg2 --with-keygrip -K
/home/alice/.gnupg/pubring.kbx
------------------------------
sec   rsa4096/5B491A54 2014-12-31 [SC] [expires: 2017-12-30]
      Keygrip = D4DF0C35D3E22FA6AC37DA2E54FB03F73616A3CB
uid               [ultimate] Alice <alice@example.org>
[…]
```

La clef privée 5B491A54 se trouve dans le fichier `~/.gnupg/private-keys-v1.d/D4DF0C35D3E22FA6AC37DA2E54FB03F73616A3CB.key`. Sortez simplement ce fichier de son répertoire et stockez-le sur le support de votre choix. Quand vous avez besoin d'utiliser la clef primaire, remettez temporairement le fichier en place. C'est tout.

Le « support de votre choix » peut être une clef USB, une carte SD, une disquette [http://fr.wikipedia.org/wiki/Disquette], un CD-R [http://fr.wikipedia.org/wiki/CD-R]…

Où conserver le support ? Là où on ne vous le volera pas, mais aussi et surtout là où vous ne le *perdrez* pas (égarer le support est un risque probablement plus grand que le risque de vol, à mon avis). Donc, *pas* caché quelque part dans un coin improbable (sous l'oreiller, dans une brique creuse de la cheminée ou sous une latte de parquet), mais plutôt proprement rangé dans un tiroir, en fait au même endroit que là où vous rangez déjà vos papiers et autres trucs importants.

La FSF Europe recommande une grotte gardée par des orques [http://wiki.fsfe.org/Card_howtos/Card_with_subkeys_using_backups#Keep_that_medium_in_a_save_place], ce qui est certainement efficace, mais si vous n'avez ni grottes ni orques dans votre région, un coffre-fort gardé par des banquiers peut aussi faire l'affaire.

Une option intéressante est d'utiliser une méthode de secret réparti [http://fr.wikipedia.org/wiki/Secret_r%C3%A9parti] pour partager la clef privée en *n* fragments, dont *m* sont nécessaires pour reconstituer la clef complète. La clef peut ainsi répartie sur plu-

sieurs supports sans que la perte de l'un d'entre eux ne fasse perdre toute la clef, et sans que le vol de l'un d'entre eux ne compromette toute la clef non plus.

Chez moi, j'utilise libgfshare [http://www.digital-scurf.org/software/libgfshare], une implémentation de la méthode de secret réparti d'Adi Shamir, pour mettre en œuvre cette dernière option.

4.2. Stockage hors-ligne des sous-clefs

En principe, les sous-clefs peuvent être gardées hors-ligne de la même façon que la clef primaire. Toutefois, les sous-clefs sont utilisées beaucoup plus fréquemment que la clef primaire et devoir ressortir le support et rapatrier les clefs dans le dossier `~/.gnupg/private-keys-v1.d` pour chaque opération de signature ou de chiffrement deviendrait rapidement pénible.

Si vous tenez à garder les sous-clefs hors-ligne, la meilleure solution est de les stocker sur un *token* à partir duquel elles peuvent être directement utilisées *sans* devoir les rapatrier sur l'ordinateur. C'est ce que permet la carte OpenPGP, sur laquelle je ne m'étendrai pas ici puisque j'en ai déjà largement parlé dans un autre article [http://www.incenp.org/dvlpt/docs/carte-openpgp.html].

4.3. Les sauvegardes

Une question voisine du stockage des clefs privées est celle de leur *sauvegarde*. Avoir (au moins) une copie de sauvegarde de vos clefs privées est crucial : plein de choses imprévues peuvent arriver à vos clefs ou au disque dur sur lequel elles se trouvent. En fait, la perte *accidentelle* est probablement plus probable qu'une compromission par un attaquant…

Il est préférable de ne *pas* sauvegarder les clefs privées au même endroit que le reste de vos données, à moins que vous ne stockiez toutes vos sauvegardes sur des supports eux-mêmes chiffrés et/ou auxquels vous seul avez accès. (Si votre support de sauvegarde est chiffré, attention au problème d'œuf et de poule : ne sauvegardez pas la clef permettant d'ouvrir le support de sauvegarde sur ce même support !)

Outre les copies de sauvegardes classiques, je recommande également une sauvegarde au format *papier*. L'outil paperkey [http://www.jabberwocky.com/software/paperkey/] exporte les informations essentielles de votre trousseau privé dans un format aisément imprimable sur une bonne vieille feuille de papier, dont la capacité de rétention d'informations sur de *longues* périodes n'est plus à démontrer. Conservez ensuite cette feuille avec le reste de vos documents importants.

5. Quelles sont les autres données à protéger ?

À part les clefs privées, quels sont les autres fichiers sensibles ?

Le(s) certificat(s) de révocation	Quiconque s'en empare peut inconditionnellement révoquer votre clef (par exemple pour tenter de faire accepter ensuite une fausse clef à vos correspondants, *avant* que nous n'ayez eu le temps d'en générer une nouvelle de votre côté). Il est souvent recommandé de le stocker hors-ligne, au même endroit que là où vous mettez votre clef primaire.
L'état du pool d'entropie	(Stocké dans `~/.gnupg/random_seed`). La connaissance de cet état pourrait permettre à un attaquant de déduire les prochaines clefs de session. En cas de

	doute, il vaut mieux supprimer purement et simplement ce fichier, GnuPG reconstituera un pool avec de l'entropie en provenance du système.
La base de confiance	(Stockée dans `/.gnupg/trustdb.gpg`). Modérément sensible. La fuite de cette base peut permettre de tisser un graphe plus précis de vos relations : savoir que vous avez signé la clef d'Alice (ce que tout le monde peut savoir en regardant sa clef publique) dit seulement que vous connaissez Alice, mais savoir que vous lui faites complètement confiance témoigne que vous la connaissez *bien* (ou que vous prenez la gestion de votre toile de confiance à la légère). Cette base de confiance peut toujours être reconstituée manuellement si besoin (mais c'est fastidieux si votre trousseau public est bien rempli).

6. Clef privée compromise, quelles conséquences ?

Il n'est pas inutile, je pense, de passer rapidement en revue les conséquences d'une compromission d'une clef privée. Admettons donc, pour les besoins du raisonnement, que Mallory a d'une façon ou d'une autre mis la main sur une des clefs privées d'Alice, à son insu. Que peut-il en faire ?

6.1. Sous-clef de chiffrement compromise

C'est probablement le pire cas de figure pour la confidentialité des communications. Mallory peut déchiffrer tous les messages à l'intention d'Alice, le risque de détection pour lui est quasiment nul : tout ce qu'il a à faire est d'intercepter les messages destinés à Alice tout en les laissant arriver intact jusqu'à elle. Seule une erreur de Mallory, comme une exploitation imprudente des renseignements qu'il obtient de cette manière, pourra conduire Alice à soupçonner que ses messages sont lus.

6.2. Sous-clef de signature compromise

Mallory peut signer des messages au nom d'Alice. C'est potentiellement dommageable (imaginons par exemple qu'Alice signe les *releases* de son logiciel libre : Mallory peut faire accepter à Bob une version proprement troué par ses soins), mais avec un risque de détection assez élevé. Mallory doit absolument éviter que les fausses signatures ne parviennent jusqu'à Alice, faute de quoi celle-ci réalisera immédiatement que quelque chose ne va pas. C'est néanmoins envisageable si Mallory a un contrôle suffisant du réseau (et en cryptographie, on suppose toujours que l'attaquant contrôle *totalement* le réseau).

6.3. Clef primaire compromise

Il est un peu plus difficile d'appréhender ce que peut faire un attaquant ayant mis la main sur la clef primaire. Avant tout, il faut noter que Mallory *ne peut pas* s'en servir pour obtenir les sous-clefs privées : il n'y a aucun lien mathématique entre une clef primaire et une sous-clef et la connaissance de la clef primaire ne permet pas de tirer la moindre information sur les sous-clefs. Partant, Mallory ne peut pas déchiffrer en l'état les communications d'Alice.

En revanche, Mallory peut

Signer des messages au nom d'Alice. Même si Alice a une sous-clef de signature, la clef primaire reste utilisable pour signer.

Signer des clefs au nom d'Alice. À la différence de la sous-clef de signature, la clef primaire peut signer non seulement des messages mais aussi les *clefs* d'autres utilisateurs. Mallory peut ainsi faire accepter de fausses clefs à ceux qui font confiance en la signature d'Alice.

Jouer à l'homme du milieu. Mallory génère une nouvelle sous-clef de chiffrement et publie une nouvelle version de la clef publique d'Alice avec cette sous-clef rattachée. Les correspondants d'Alice récupèrent cette sous-clef en rafraîchissant leur trousseau public (contrairement à une attaque MITM « classique », Mallory n'a ici aucune difficulté à faire accepter sa sous-clef de chiffrement, puisqu'elle est attachée à la vraie clef publique d'Alice que ses correspondants connaissent), et se mettent à chiffrer leurs messages destinés à Alice avec la sous-clef créée par Mallory. Celui-ci intercepte les messages, les déchiffre, les *re-chiffre* avec la sous-clef de chiffrement originale d'Alice, et les fait suivre à Alice.

Toutes ces actions ont en commun de présenter un risque de détection élevé. C'est particulièrement vrai pour la dernière, où Mallory doit s'assurer non seulement que sa nouvelle sous-clef de chiffrement n'est vue que par les correspondants d'Alice et surtout pas par Alice elle-même, mais aussi qu'il intercepte bien *tous* les messages destinés à Alice — si celle-ci voit arriver ne serait-ce qu'un seul message à son intention mais chiffré avec une sous-clef qu'elle ne connaît pas, elle réalisera immédiatement l'interférence.

Modifier les préférences de la clef publique. Plus sournoisement, Mallory peut publier une nouvelle version de la clef publique d'Alice dans laquelle il a altéré la sélection d'algorithmes préférés, afin de mettre en tête (ou uniquement) un algorithme plus « facile » à casser (3DES par exemple). Contrairement à ce qui précède, cette attaque, même si elle est détectable, peut tout de même facilement inaperçue. Elle suppose néanmoins que Mallory soit capable de casser au moins un des algorithmes symétriques utilisés par OpenPGP, ce qui n'est pas gagné (même 3DES, *a priori* le plus faible des algorithmes disponibles, résiste encore très bien aux attaques cryptanalytiques).

A. À propos de ce document