La carte OpenPGP

Utilisation avec GnuPG sous GNU/Linux

Damien Goutte-Gattat <dgouttegattat@incenp.org>

2018/06/07

Historique des versions

Version 1.4 2018/06/07
Simplification de la section sur GnuPG (les branches obsolètes ne sont plus traitées). Mise à jour des implémentations de la carte OpenPGP.

Version 1.3 2017/07/14
Mise à jour pour Scute 1.5.0. Exemple d'utilisation de **SCD LEARN**.

Version 1.2 2016/12/25
Mise à jour pour GnuPG 2.1 (maintenant considéré comme la version par défaut de GnuPG).

Version 1.1 2016/02/07
Mise à jour pour les dernières versions de GnuPG et la prise en charge de TLS 1.2 dans la version de développement de Scute.

Version 1.0 2014/12/18
Première version publique.

Résumé

Cet article présente l'application cryptographique « OpenPGP » pour carte à puce et les différents usages, classiques ou plus avancés, qu'il est possible d'en faire avec GnuPG.

Table des matières

1. Présentation de la carte OpenPGP 2
    1.1. Les clefs cryptographiques 2
    1.2. Sécurité de la carte 3
    1.3. Données stockées sur la carte 4
2. Le matériel 5
    2.1. La carte à puce 5
    2.2. Le lecteur de cartes 6
3. Les logiciels 7
    3.1. Les pilotes de lecteurs de carte 7
    3.2. Le *middleware* PCSC-Lite 7
    3.3. GnuPG 8
4. Préparer la carte 10
    4.1. Changer les PIN 10
    4.2. Modifier le contenu de la carte 11
    4.3. Les clefs privées 11
5. Signer ou déchiffrer des messages OpenPGP 16
6. S'authentifier auprès d'un serveur SSH 17
7. Utiliser un certificat X.509 18
    7.1. Créer un certificat X.509 18
    7.2. S'authentifier auprès d'un serveur web 20
    7.3. Signer des messages S/MIME 21
    7.4. Signer des documents OpenDocument 21
    7.5. Utiliser la carte comme *token* PKCS #15 22
8. Miscellanées 23
    8.1. Communiquer avec Scdaemon et la carte 23
    8.2. Ré-initialiser une carte bloquée 25
A. À propos de ce document 25

1. Présentation de la carte OpenPGP

La carte OpenPGP [https://g10code.com/p-card.html] est une application cryptographique pour carte à puce au format ISO 7816 [https://en.wikipedia.org/wiki/ISO/IEC_7816]. Une carte à puce dotée d'une telle application vous permet d'y stocker et de protéger vos clefs cryptographiques privées.

La carte OpenPGP a été imaginée par Achim Pietig avec l'aide des développeurs à l'origine de GnuPG [https://gnupg.org/] et est spécialement conçue pour fonctionner avec ce dernier. Son utilisation est aussi transparente que possible et avoir vos clefs privées sur une carte OpenPGP ne change fondamentalement rien à la manière d'utiliser GnuPG : la seule différence est qu'à chaque fois que GnuPG aura besoin d'accéder à une de vos clefs privées, au lieu de vous demander la phrase de passe protégeant la clef sur votre disque dur, il vous demandera le PIN de la carte.

Outre la signature et le déchiffrement de messages OpenPGP, les autres usages possibles de la carte incluent :

- la signature ou le déchiffrement de messages S/MIME ;
- l'authentification auprès d'un serveur SSH ;
- l'authentification auprès d'un serveur TLS ;
- et probablement d'autres utilisations qui restent à imaginer ou à mettre en œuvre.

Ce document couvre la dernière version de la spécification (version 3.3.1 [https://gnupg.org/ftp/specs/OpenPGP-smart-card-application-3.3.pdf], publiée en août 2017). Les différences importantes avec les versions précédentes seront mentionnées lorsque ce sera nécessaire.

1.1. Les clefs cryptographiques

La carte OpenPGP permet de stocker jusqu'à trois clefs privées, une pour chaque type d'utilisation : une clef de (dé)chiffrement, une clef de signature et une clef d'authentification.

Le rôle majeur de la carte est de *protéger ces clefs privées*. Lorsqu'elles sont stockées sur la carte, il est par conception impossible de les en extraire, elles ne peuvent qu'être utilisées « sur place ».

Ainsi, toutes les opérations cryptographiques nécessitant l'une de ces clefs sont réalisées directement *sur la carte elle-même* : l'ordinateur envoie à la carte les données à déchiffrer,[1] signer ou authentifier, et il reçoit le résultat de l'opération en retour, sans jamais avoir entraperçu les clefs.

Faire réaliser les opérations cryptographiques par la carte ne les rend pas plus rapides, pour au moins trois raisons :

- il est peu probable que le processeur de la carte, même s'il est spécialisé, soit plus véloce que votre CPU ;
- l'échange de données entre l'ordinateur et la carte introduit un délai non négligeable ;
- la carte ne réalise en réalité qu'une partie du travail, le reste restant à la charge de l'ordinateur (précisément, entre autres, pour limiter le volume

[1] La carte n'est jamais utilisée pour *chiffrer*, puisqu'en cryptographie asymétrique, seul le déchiffrement nécessite une clef *privée* — le chiffrement utilise la clef publique du destinataire.

de données à transférer entre les deux appareils) ; par exemple, pour déchiffrer un message OpenPGP, la carte n'est responsable que du déchiffrement de la clef de session symétrique (soit seulement 16 octets pour une clef AES 128 bits) — le déchiffrement du corps du message est fait par l'ordinateur, une fois qu'il a obtenu de la carte la clef de session déchiffrée.

Bref, il ne faut pas compter sur la carte pour *accélérer* les opérations cryptographiques, ce n'est pas son but.

Jusqu'à la version 2.2 incluse, les clefs de la carte ne peuvent être que des clefs RSA, la spécification garantissant une taille minimale pour chaque clef de 1024 bits. La version 3.0 élève la taille minimale garantie à 2048 bits, et ajoute la prise en charge des clefs à base de courbes elliptiques [https://tools.ietf.org/html/rfc6637].

1.2. Sécurité de la carte

La carte OpenPGP est protégée par deux ou (optionnellement) trois codes PIN, désignés `PW1` ou « PIN utilisateur » (*User PIN*, parfois appelé seulement « PIN », par opposition au suivant), `PW3` ou « PIN administrateur » (*Admin PIN*), et `RC` (*Resetting Code*, optionnel).

Les opérations de la carte peuvent être classées en fonction du PIN nécessaire pour les réaliser :

- aucun PIN n'est nécessaire pour *lire* les données de la carte ;
- le PIN utilisateur est demandé pour toute opération exploitant les clefs privées (signature, déchiffrement, authentification), et pour changer le PIN utilisateur lui-même ;
- le PIN administrateur est demandé pour presque toutes les *écritures* sur la carte (modification des informations stockées sur la carte, importation ou génération de clefs privées, changement d'un PIN autre que le PIN utilisateur).

À chaque PIN est associé un compteur d'essai (*Retry Counter*), qui est décrémenté à chaque saisie incorrecte du PIN. Tant que le compteur est supérieur à zéro, il suffit de saisir le PIN correct pour remettre le compteur à trois. Lorsque le compteur tombe à zéro, la vérification du PIN correspondant n'est plus possible, et toutes les opérations dépendantes de cette vérification sont interdites. Dans ce cas, il faut enregistrer un nouveau PIN pour remettre le compteur à trois.

Concrètement, cela signifie ① que trois saisies erronées consécutives du PIN utilisateur entraînent un blocage de la carte (puisque toutes les opérations cryptographiques nécessitent de vérifier le PIN utilisateur), ② que la carte peut être débloquée en changeant le PIN utilisateur, après avoir saisi le PIN administrateur, ③ qu'après trois saisies erronées consécutives du PIN administrateur, la carte est *irréversiblement* bloquée (puisqu'il faudrait changer le PIN administrateur pour remettre son compteur d'essai à trois, or cela nécessite de vérifier le PIN administrateur, ce qui n'est justement pas possible si son compteur est à zéro).

Certaines implémentations prévoient néanmoins la possibilité de ré-initialiser complètement une carte dont les compteurs d'essai des PIN utilisateur et administrateur sont tous les deux à zéro. La ré-initialisation efface toutes les données de la carte, restaure les PIN utilisateur et administrateur par défaut, et remet leur compteur à trois.

Et le *Resetting Code* ? S'il est défini, il peut être utilisé à la place du PIN administrateur pour ré-initialiser le PIN utilisateur et donc débloquer la carte. Peu pertinent pour une utilisation individuelle, le RC est surtout utile dans les situations où la carte est délivrée par une autorité à un utilisateur qui n'est pas supposé en modifier le contenu et à qui on ne veut donc pas révéler le PIN administrateur, mais que l'on veut tout de même autoriser à débloquer sa carte tout seul. (Le RC est donc assimilable au code « PUK » utilisé pour

débloquer une carte SIM.) Par défaut, aucun *Resetting Code* n'est défini et son compteur d'essai est à zéro.

Le PIN utilisateur doit faire au minimum 6 caractères, le PIN administrateur et le *Resetting Code* au minimum 8. La taille maximale n'est pas définie dans la spécification (elle est de 32 caractères pour chaque PIN sur l'implémentation de référence de ZeitControl). Il est possible d'utiliser n'importe quel caractère UTF-8 et non pas seulement des chiffres, mais un code non-numérique ne pourra pas être saisi sur le clavier intégré à certains lecteurs de carte.

Dans la suite de ce document, « PIN » sans précision désignera implicitement le PIN utilisateur. Lorsqu'il sera question du PIN administrateur, il sera toujours désigné explicitement.

1.3. Données stockées sur la carte

Outre les clefs privées, la carte OpenPGP contient un certain nombre de champs (*Data Objects* ou DO, dans le jargon des cartes à puce) pour stocker des données supplémentaires. Ces champs sont réparties en deux grandes catégories.

1.3.1. Les champs à usage défini

Ce sont des champs dont l'usage est explicitement défini dans la spécification de la carte OpenPGP. Toutefois, GnuPG n'utilise pas lui-même la plupart de ces champs — il permet de les consulter et les modifier, mais c'est à l'utilisateur ou à d'autres programmes exploitant la carte de leur trouver une utilisation.

Ces champs sont :

- le nom du détenteur de la carte ;
- son sexe ;
- ses préférences linguistiques : une liste de 1 à 4 codes de langue au format ISO 639-1 ;
- son « identifiant » : la spécification mentionne la possibilité qu'un système d'authentification basée sur la carte OpenPGP utilise ce champ pour savoir sur quel compte connecter l'utilisateur authentifié, mais je ne connais pas de systèmes qui le font ;
- un certificat : typiquement, un certificat X.509 au nom de l'utilisateur, mais n'importe quel autre format est possible — GnuPG traite ce champ comme un simple blob binaire ;
- une URL vers la clef publique de l'utilisateur : GnuPG s'en sert pour rapatrier automatiquement la clef publique en question, ce qui permet d'avoir rapidement un trousseau opérationnel lorsque vous êtes amenés à travailler sur une machine qui n'est pas la vôtre ;
- jusqu'à trois « empreintes de clefs de confiance » (*CA Fingerprints*) : des empreintes SHA-1 de clefs OpenPGP considérées valides et auxquelles il est fait une confiance absolue — là encore, GnuPG ne se ne sert pas de ces champs et je ne connais aucun exemple concret d'utilisation.

1.3.2. Les « champs privés » ou à usage indéfini

En plus des champs ci-dessus, la version 2.0 de la spécification définit 4 champs inutilisés, de 254 octets chacun[2], dont l'usage est entièrement et officiellement laissé à la discrétion de l'utilisateur : les *Private Use DOs* 1 à 4.

[2] À partir de la version 3.0 de la carte, la taille maximale de ces champs n'est plus fixée, mais est laissée à la discrétion des implémentations.

La différence entre ces 4 champs réside dans le PIN demandé pour y accéder en lecture ou en écriture :

Champ	**Lecture**	**Écriture**
Private DO 1	aucun PIN	PIN utilisateur
Private DO 2	aucun PIN	PIN administrateur
Private DO 3	PIN utilisateur	PIN utilisateur
Private DO 4	PIN administrateur	PIN administrateur

(On notera que ces conditions s'écartent quelque peu du principe général énoncé dans la Section 1.2, « Sécurité de la carte », selon lequel la lecture de la carte ne demande pas de PIN et l'écriture demande le PIN administrateur.)

Les champs à usage privé sont une fonctionnalité optionelle de la spécification, ils ne sont pas nécessairement supportés par toutes les implémentations. Lorsqu'ils existent, ils sont vides par défaut, il appartient à chacun d'imaginer l'usage qu'il peut en faire.

2. Le matériel

2.1. La carte à puce

La « carte OpenPGP » est juste une spécification ; tout le monde peut l'implémenter sur le support de son choix.

Actuellement, l'implémentation la plus répandue est réalisée sur la carte BasicCard [http://www.zeitcontrol.de/en/products/basiccard] de ZeitControl, une carte à puce programmable en BASIC. Elle est distribuée par FLOSS-Shop [https://www.floss-shop.de/en/security-privacy/smartcards/] (anciennement *Kernel Concepts*).

Le code source de cette implémentation, écrit par Achim Pietig, est partiellement disponible [http://g10code.com/docs/openpgp-card-v21-free-source.zip]. Le code complet inclut des routines faisant l'objet d'accords de non-divulgation entre Achim Pietig et les fournisseurs de puces, et ne peut de fait être diffusé sous licence libre. La version libre est pleinement fonctionnelle, mais n'offre pas la même résistance aux attaques dirigées contre la carte à puce.

Gnuk [https://www.fsij.org/category/gnuk.html] est une implémentation de la carte OpenPGP pour microcontrôleur STM32F103, écrit par Niibe Yutaka de la *Free Software Initiative of Japan*. Gnuk permet de réaliser un *token USB* — un périphérique qui apparaît, aux yeux de l'ordinateur, comme un lecteur de cartes à puce classique contenant une carte unique insérée en permanence. Le FST-01 [https://www.gniibe.org/FST-01/fst-01.html] est un exemple d'un tel token basé sur Gnuk, également conçu par Niibe Yutaka (qui en fournit également tous les plans, ce qui fait du FST-01 une implémentation complètement libre de la carte OpenPGP, aussi bien côté matériel que logiciel). Un FST-01 pré-chargé avec *NeuG* (un générateur de nombres aléatoires) au lieu de Gnuk peut aussi être obtenu auprès de la Free Software Foundation [https://shop.fsf.org/storage-devices/neug-usb-true-random-number-generator] (il peut être reflashé avec Gnuk en suivant les instructions de Niibe Yutaka [https://www.gniibe.org/FST-01/q_and_a/gnuk_install_over_neug.html]).

Les Nitrokey [https://www.nitrokey.com/] (anciennement *CryptoStick*) sont des tokens USB fournissant, entre autres, les fonctionnalités d'une carte OpenPGP. La Nitrokey Start est basée sur Gnuk, tandis que les Nitrokey Pro et Nitrokey Storage embarquent en leur sein un exemplaire de la carte OpenPGP physique de FLOSS-Shop.

Il existe aussi des implémentations ciblant des Java Cards [https://en.wikipedia.org/wiki/Java_Card] (cartes à puce programmable en Java), comme celle proposée par l'ANSSI [https://github.com/anssi-fr/smartpgp] ou celle de Joeri de Ruiter [https://github.com/jderuiter/javacard-openpgpcard]. Le token YubiKey NEO [https://www.yubico.com/products/yubikey-hardware/yubikey-neo/] contient une version de cette dernière, désactivée par défaut — Yubico fournit les instructions pour l'activer [https://www.yubico.com/2012/12/yubikey-neo-openpgp/]. Une particularité intéressante de la Yubikey NEO est qu'elle est utilisable en mode sans contact, grâce à la technologie NFC [https://en.wikipedia.com/wiki/Near_field_communication]. Cela permet notamment son utilisation sur un téléphone Android avec l'application OpenKeychain [https://www.openkeychain.org/].

À l'heure où ces lignes sont écrites, les implémentations compatibles avec la version 3.0 ou ultérieure de la spécification sont celle d'Achim Pietig sur BasicCard, Gnuk (et les modèles récents de la Nitrokey Start), et l'implémentation de l'ANSSI sur Javacard.

2.2. Le lecteur de cartes

À moins d'opter pour un token USB comme le FST-01, la Nitrokey ou la YubiKey, un lecteur de cartes à puce est évidemment nécessaire pour utiliser une carte OpenPGP.

Il existe une grande variété de lecteurs, voici les principaux points à considérer pour choisir le sien.

Port série, port PCMCIA ou port USB	La plupart des lecteurs aujourd'hui sont des lecteurs USB — et, faute d'expérience de l'auteur avec les autres types de lecteurs, ce sont les seuls dont il sera question dans cet article.
Clavier intégré ?	Un petit nombre de lecteurs contiennent un clavier intégré permettant de sasir le PIN directement sur le lecteur (par exemple le SPR322 de Identive [https://www.identiv.com/products/smart-card-readers/contact/spr332v2/]). L'intérêt en terme de sécurité est que dans ce cas, le PIN ne passe jamais par l'ordinateur et n'est donc pas susceptible d'être enregistré par un éventuel *keylogger* ou autre logiciel malveillant. Attention, le clavier ne comprend la plupart du temps que des chiffres, interdisant la saisie d'un PIN qui ne serait pas entièrement numérique ; certains de ces lecteurs limitent aussi la longueur maximale du PIN, qui peut être plus courte que ce qu'autorise la carte OpenPGP.
Prise en charge des *extended APDU*	Les APDU (*Application Protocol Data Unit*) sont les messages échangés entre la carte à puce et son lecteur. Ils peuvent être *short* (les données transmises sont limitées à 256 octets) ou *extended* (limités à 65536 octets). Pour importer une clef de plus de 1024 bits sur la carte OpenPGP, le lecteur doit prendre en charge les *extended APDU*.

Un autre point important est bien sûr la prise en charge du lecteur sous votre système. La question des pilotes sera abordée dans la section suivante, mais disons tout de suite que sous GNU/Linux, vous vous simplifierez probablement la tâche en choisissant un lecteur conforme à la norme CCID (*Chip/Card Interface Device*). Le pilote ccid [https://ccid.apdu.fr/] prend en charge une grande partie de ces lecteurs, et ses développeurs tiennent à jour des listes des lecteurs pris en charge ou non [https://ccid.apdu.fr/#readers] — et indiquent en plus pour chaque lecteur les problèmes potentiels comme l'absence de support de l'*extended APDU*.

3. Les logiciels

En règle générale, l'utilisation de cartes à puce nécessite une pile logicielle à trois niveaux : un pilote pour le lecteur de cartes, un *middleware* exposant une API standard d'accès aux cartes à puce (permettant aux applications de faire abstraction du matériel et des pilotes), et les applications utilisatrices.

3.1. Les pilotes de lecteurs de carte

Le pilote ccid déjà évoqué ci-dessus prend en charge un grand nombre de lecteurs USB compatibles avec la norme CCID. Cela inclut également les *tokens* USB comme le CryptoStick, les *tokens* basés sur Gnuk ou le YubiKey NEO, qui sont tous compatibles CCID.

Il existe également des pilotes plus spécifiques pour des modèles précis de lecteurs. Pour les utilisateurs de Debian (et dérivés), le paquet virtuel pcsc-ifd-handler [https://packages.debian.org/sid/pcsc-ifd-handler] en fournit commodément une liste.

Des modèles non-pris en charge par les pilotes libres disposent parfois d'un pilote propriétaire fourni par le constructeur. C'est le cas par exemple des lecteurs de la gamme HID Omnikey, dont les pilotes sont disponibles sur www.hidglobal.com/drivers [https://www.hidglobal.com/drivers].

Libre ou non, le pilote d'un lecteur USB doit être installé dans un dossier appelé `nom-du-pilote.bundle`, lui-même situé dans le dossier `/usr/local/pcsc/drivers` (par défaut — ce dossier est configurable à la compilation de PCSC-Lite, vous pouvez consulter la page de manuel de pcscd pour connaître le chemin effectif sur votre système). Vous pouvez bien sûr ignorer cette étape si le pilote est fourni dans les paquets de votre distribution.

3.2. Le *middleware* PCSC-Lite

PCSC-Lite [https://pcsclite.apdu.fr/] est une implémentation libre pour systèmes Unix-like (y compris Mac OS X) de l'API Windows SCard, introduite dans Windows XP/Windows Server 2003 et normalisée par la suite sous le nom de spécification PC/SC [https://www.pcscworkgroup.com/specifications/].

PCSC-Lite comprend deux composants : un démon (pcscd) qui gère le lecteur et communique avec la carte, et une bibliothèque (libpcsclite.so) qui expose l'API PC/SC aux programmes qui lui sont liés.

Le paquet de PCSC-Lite fourni par votre distribution devrait faire le nécessaire pour que le démon pcscd soit automatiquement lancé au démarrage. Si ce n'était pas le cas, consultez la documentation de votre distribution pour savoir comment ajouter le script shell / le *job* Upstart / l'unité Systemd / le truc lanceur de machins approprié à votre système.

Si pcscd tourne sous un compte utilisateur dédié non-privilégié (ce qui est recommandé), il faut s'assurer que ledit compte possède les droits sur le lecteur de carte. Cela peut se faire avec une règle Udev semblable à celle-ci (à ajouter dans `/etc/udev/rules.d/90-local.rules`, ou l'équivalent pour votre distribution) :

```
ACTION=="add", SUBSYSTEM=="usb", ATTR{idVendor}=="04e6", ATTR{idProduct}=="5410", \
OWNER="scard", MODE="600"
```

où *04e6* et *5410* identifient le fournisseur et le modèle de votre lecteur, respectivement (les valeurs données ici correspondent au SCR 3500 de l'auteur, vous pouvez trouver

celles correspondants à votre modèle avec **lsusb -v** par exemple), et *scard* est le nom du compte sous lequel tourne pcscd.

Les lecteurs sur port série ou PCMCIA doivent déjà être présents lorsque pcscd est démarré pour que celui-ci les détecte automatiquement. Si vous connectez un tel lecteur *après* le lancement de pcscd, il faudra utiliser **pcscd --hotplug** pour inciter le démon à rescanner les bus série et PCMCIA. Encore une fois, une règle Udev fera l'affaire, comme dans l'exemple suivant (issu de la distribution du pilote ccid) :

```
SUBSYSTEMS=="pcmcia", DRIVERS=="serial_cs", ACTION=="add", ATTRS{prod_id1}=="Gemplus", \
ATTRS{prod_id2}=="SerialPort", ATTRS{prod_id3}=="GemPC Card", RUN+="/usr/sbin/pcscd --hotplug"
```

Cela ne concerne pas les lecteurs USB, qui sont toujours détectés automatiquement sitôt qu'ils sont connectés.

3.3. GnuPG

GnuPG est bien entendu la principale application utilisatrice de la carte OpenPGP, et c'est la seule qui sera abordée dans cet article. Sachez néanmoins que d'autres applications peuvent exploiter cette carte. En effet, quoique développée spécifiquement pour les besoins de GnuPG, la carte OpenPGP est compatible avec le standard PKCS#15 [https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-15-cryptographic-token-information-format.htm], qui définit une manière commune de stocker et d'accéder à des données cryptographiques sur une carte à puce. En fait, on peut voir la carte OpenPGP comme une carte PKCS#15, avec quelques spécificités propres à OpenPGP en plus.

3.3.1. À propos des versions de GnuPG

Il existe actuellement deux variantes de GnuPG : la branche 1.x, dite « classique », dont la dernière version à l'heure où ces lignes sont écrites est la 1.4.22, et la branche « stable », dont la dernière version est la 2.2.7. Les branches 2.0 (ancienne branche « stable ») et 2.1 (ancienne branche « moderne ») ne sont plus supportées.

Dans le reste de ce document, je supposerai l'utilisation de GnuPG 2.2. Il est possible d'utiliser la carte OpenPGP avec GnuPG 1.x, mais avec cette version ne permet que les opérations « de base » (éditer le contenu de la carte, déchiffrer et signer des messages OpenPGP). Les usages plus avancés qui seront décrits plus loin, et notamment l'utilisation de la clef d'authentification, nécessitent les outils auxiliaires apportés à partir de la branche 2.x, en particulier l'agent GnuPG (gpg-agent).

Par ailleurs, la principale raison d'être de la branche 1.x aujour'hui est de gérer le passé, en continuant à prendre en charge des clefs et des algorithmes obsolètes que les branches 2.x ont laissé de côté (comme les clefs au format OpenPGP V3, qui remontent à l'époque de PGP 2.6 — soit au milieu des années 1990).

Si vous tenez malgré tout à utiliser GnuPG 1.x, sachez que vous pouvez l'utiliser conjointement avec l'agent GnuPG de la branche 2.x. Dans ce cas, certains des « usages avancés » vous seront accessibles, notamment l'authentification SSH. En revanche, si vous tenez à utiliser GnuPG 1.x *seulement* (sans agent), vous ne pourrez pas faire usage de la clef d'authentification (en tout cas pas avec GnuPG).

3.3.2. Comment GnuPG accède au lecteur de carte

Les applications de GnuPG 2.x (**gpg** et **gpgsm**) n'accèdent jamais directement au lecteur de carte, dont ils sont séparés par plusieurs niveaux d'indirection. Elles interagissent

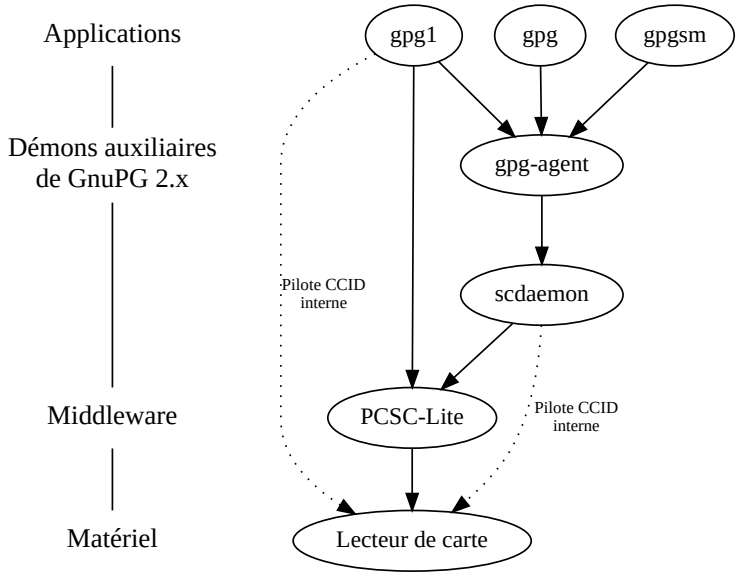
seulement avec l'agent GnuPG, un démon qui gère les clefs secrètes et les phrases de passe pour l'ensemble du système GnuPG 2.x.

L'agent GnuPG lui-même délègue la gestion des cartes à puce à un autre démon, Scdaemon (SmartCard Daemon). C'est lui seul qui s'occupe de détecter le lecteur de carte et de communiquer avec la carte qu'il contient. Il utilise un protocol ad-hoc [https://www.gnupg.org/documentation/manuals/gnupg/Scdaemon-Protocol.html#Scdaemon-Protocol] pour recevoir ses instructions de l'agent GnuPG.

Scdaemon dispose de deux options pour accéder au lecteur de carte. Dans le cas général, il utilise le *middleware* PCSC-Lite que nous avons vu précédemment. Mais il peut aussi utiliser un petit pilote CCID interne, qui lui permet de communiquer directement avec les lecteurs USB compatibles sans passer par un intermédiaire.

GnuPG 1.x, de son côté, *peut* utiliser l'agent GnuPG (ce qu'il fait automatiquement par défaut, s'il détecte qu'un agent est en cours d'exécution), auquel cas tout se passe comme avec GnuPG 2.x. Mais il peut aussi se passer des démons auxiliaires et communiquer seul avec le lecteur de carte.

Figure 1. Architecture de GnuPG pour l'accès aux cartes à puce



3.3.3. Le pilote CCID interne

Comme évoqué à l'instant, GnuPG (toutes versions confondues) contient un petit pilote générique pour les lecteurs compatibles CCID. Ses développeurs décrivent ce pilote comme « un pilote limité […] à utiliser en dernier recours quand rien d'autre ne fonctionne ou que l'on tient à avoir un système minimal pour des raisons de sécurité »[3].

Si votre lecteur de carte est pris en charge par le pilote interne, vous pouvez être tenté de simplifier la pile logicielle en vous passant du *middleware* PCSC-Lite. Gardez toutefois à l'esprit que le pilote interne est spécifique à GnuPG : si vous prévoyez d'utiliser votre lecteur pour d'autres applications (ne serait-ce que pour explorer, par curiosité, le contenu de votre carte bancaire, de votre carte vitale ou de votre carte de transports en commun, par exemple avec Cardpeek [http://pannetrat.com/Cardpeek/]), vous aurez besoin de PCSC-Lite de toute façon.

Pour utiliser PCSC-Lite, il suffit que le démon de PCSC-Lite soit démarré : GnuPG tentera d'accèder au lecteur avec son pilote interne, échouera puisque le lecteur sera déjà

[3] Commentaire en tête du fichier `scd/ccid-driver.c`, dans les sources de GnuPG.

monopolisé par pcscd, et se rabattra automatiquement sur PCSC-Lite (vous pouvez éventuellement désactiver explicitement le pilote interne en ajoutant l'option `disable-ccid` dans le fichier de configuration de scdaemon, `$GNUPGHOME/scdaemon.conf`).

Inversement, pour utiliser le pilote interne, assurez-vous que PCSC-Lite n'est pas installé (ou, *a minima*, que son démon n'est pas démarré), et que votre propre compte utilisateur, sous l'identité duquel tourne GnuPG, a accès au lecteur de carte.

4. Préparer la carte

4.1. Changer les PIN

La carte peut vous être fournie soit avec des PIN par défaut, qui sont alors 123456 pour le PIN utilisateur et 12345678 pour le PIN administrateur, soit avec des PIN personnalisés qui doivent alors vous être communiqués. Dans tous les cas, vous devriez changer ces PIN au plus tôt.

Insérez la carte dans le lecteur et lancez l'éditeur de carte de GnuPG, qui vous accueille en listant le contenu des principaux DO de la carte :

```
$ gpg --card-edit
Application ID ...: D2760001240102000005000012340000
Version ..........: 2.0
Manufacturer .....: ZeitControl
Serial number ....: 00001234
Name of cardholder: [not set]
Language prefs ...: de
Sex ..............: [not set]
URL of public key : [not set]
Login data .......: [not set]
Signature PIN ....: forced
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: [not set]
Encryption key ...: [not set]
Authentication key: [not set]

gpg/card>
```

Lorsque vous voudrez juste afficher le contenu de la carte, vous pourrez obtenir la même sortie avec la commande non-interactive **gpg --card-status**.

Vous êtes alors dans le menu d'édition de carte de GnuPG, signalé par l'invite `gpg/card>`. Lorsque vous arrivez dans ce menu, la plupart des commandes d'édition ne sont pas immédiatement disponibles, il faut les demander explicitement avec la commande **admin** :

```
gpg/card> admin
Admin commands are allowed.
```

Vous pouvez à présent accéder au menu relatif aux différents PIN :

```
gpg/card> passwd

1 - change PIN
2 - unblock PIN
3 - change Admin PIN
```

```
4 - set the Reset Code
Q - Quit

Your selection?
```

Changez successivement le PIN administrateur (option 3 — le PIN administrateur actuel, celui par défaut, vous sera alors demandé), puis le PIN utilisateur (option 1).

Souvenez-vous que que la carte impose une longueur minimale pour chaque PIN. Un PIN utilisateur de moins de 6 caractères sera refusé, de même qu'un PIN administrateur de moins de 8 caractères. Les longueurs maximales, non définies par la spécification, sont quand à elles indiquées par la ligne *Max. PIN lengths* dans la sortie ci-dessus (dans l'ordre PIN utilisateur, *Reset Code*, PIN administrateur).

4.2. Modifier le contenu de la carte

Pendant que vous êtes dans le menu d'édition, vous pouvez remplir les différents champs de données concernant le porteur de la carte (je supposerai par la suite que vous vous appelez *Alice*, par souci de la tradition) :

```
gpg/card> name
Cardholder's surname: Smith
Cardholder's given name: Alice

gpg/card> lang
Language preferences: fr

gpg/card> sex
Sex ((M)ale, (F)emale or space): F

gpg/card> url
URL to retrieve public key: http://example.net/~alice/pgp.asc

gpg/card> login
Login data (account name): alice
```

Vous pouvez aussi écrire dans les champs privés (voir Section 1.3.2, « Les « champs privés » ou à usage indéfini ») avec la commande **privatedo**, même si celle-ci n'apparaît pas dans le menu de l'éditeur de carte. Par exemple, pour écrire dans le *Private DO 1* :

```
gpg/card> privatedo 1
Private DO data: lorem ipsum dolor sit amet
```

Même chose à partir d'un fichier :

```
gpg/card> private do 1 < fichier
```

Quand vous avez modifié ce que vous vouliez, quitter l'éditeur pour revenir à l'invite de commande du shell :

```
gpg/card> quit
```

4.3. Les clefs privées

Avant de pouvoir utiliser la carte pour des opérations cryptographiques, il faut y stocker des clefs privées.

Les clefs peuvent être générées directement sur la carte, auquel cas GnuPG récupère les clefs publiques correspondantes et les importe dans le trousseau public. Attention, dans ce cas la carte contient les *seuls* exemplaires existants des clefs privées, aucune sauvegarde n'est possible puisque, par conception, on ne peut pas extraire les clefs privées de la carte.

L'autre option consiste à générer les clefs comme d'habitude sur son ordinateur, puis à importer les clefs privées sur la carte. Elles sont alors supprimées du trousseau privée, et remplacées par des *stubs* dont la seule fonction est d'indiquer à GnuPG que les vraies clefs privées sont sur une carte à puce.

C'est cette option que nous allons voir à présent — après tout si vous vous êtes procuré une carte OpenPGP, c'est que vous êtes probablement déjà utilisateur de GnuPG et vous avez donc probablement déjà un jeu de clefs.

Dans le reste de ce document, nous utiliserons comme exemple le trousseau suivant :

```
$ gpg --list-keys alice
pub   rsa4096 2016-12-25 [SC] [expires: 2019-12-25]
      DFF9C8A3FE6663F9DD157E16F5C95C96DD4C784D
uid           [ultimate] Alice Smith <alice@example.org>
sub   rsa2048 2016-12-25 [E] [expires: 2017-12-25]
sub   rsa2048 2016-12-25 [S] [expires: 2017-12-25]
```

Nous avons ici une clef maître RSA de 4096 bits et deux sous-clefs RSA de 2048 bits chacune — une de chiffrement (`E`, *Encrypt*) et une de signature (`S`). Seules ces deux sous-clefs sont utilisées quotidiennement, la clef maître ne sert qu'à modifier le jeu de clefs (ajouter ou révoquer des identités ou des sous-clefs) ou pour signer les clefs d'autres utilisateurs.

4.3.1. Créer une sous-clef d'authentification

Nous allons d'abord ajouter une troisième sous-clef RSA de 2048 bits qui servira aux fonctions d'authentification que nous verrons plus loin.

Si vous ne prévoyez pas de faire usage des fonctions d'authentification, vous pouvez passer directement à la section suivante, où nous transférerons toutes les sous-clefs sur la carte. Si vous changez d'avis par la suite, vous pourrez toujours ajouter une sous-clef d'authentification à tout moment.

Lancez l'éditeur de clefs de GnuPG sur votre trousseau (notez l'option `--expert` dans la ligne de commande ci-dessous : elle vous donne accès aux options avancées de création de clefs) :

```
$ gpg --expert --edit-key alice
Secret key is available.

sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
ssb  rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
[ultimate] (1). Alice Smith <alice@example.org>

gpg> addkey

Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
```

```
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
  (10) ECC (sign only)
  (11) ECC (set your own capabilities)
  (12) ECC (encrypt only)
  (13) Existing key
Your selection?
```

Choisissez `(8) RSA (set your own capabilities)`.

```
Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Sign Encrypt

   (S) Toggle the sign capability
   (E) Toggle the encrypt capability
   (A) Toggle the authenticate capability
   (Q) Finished

Your selection?
```

Choisissez successivement `(S)`, `(E)` et `(A)` pour désactiver les fonctions de signature et de chiffrement et activer la fonction d'authentification, puis `(Q)` pour quitter ce sous-menu. Le reste de la procédure est classique pour une génération de sous-clef :

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = keys does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1y
Key expires at Mon 25 Dec 2017 08:24:26 PM CEST
Is this correct? (y/N) y
Really create? (y/N) y

We need to generate a lot of random bytes. It is a good idea to perform
some other actions (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
ssb  rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>

gpg> save
```

4.3.2. Transférer les sous-clefs sur la carte

Nous pouvons à présent transférer les trois sous-clefs sur la carte OpenPGP. Notez qu'il est tout-à-fait possible d'y transférer la clef maître (dans le slot réservé à la clef de signature, puisque la clef maître peut généralement *signer* en plus de pouvoir *certifier*), mais cet usage est généralement déconseillé.

Si vous ne l'avez pas déjà fait après avoir créé vos clefs, vous devriez envisager de sauvegarder votre trousseau privé actuel avant de procéder au transfert. Souvenez-vous, le transfert des clefs privées sur la carte *les supprime du trousseau sur l'ordinateur*. Si vous voulez pouvoir restaurer vos sous-clefs en cas de perte de la carte, vous aurez besoin de cette sauvegarde :

```
$ gpg --armor --output my-private-keys.asc --export-secret-key alice
```

Stockez le fichier résultant dans un endroit sûr, hors-ligne (au même endroit que là où vous avez stocké le certificat de révocation de la clef maître).

Assurez-vous que votre carte est dans le lecteur et relancez l'éditeur de clefs (*pas* l'éditeur de carte — c'est contre-intuitif mais logique : transférer une clef existante sur une carte est une modification du trousseau) :

```
$ gpg --edit-key alice
Secret key is available.

sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
ssb  rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>
```

Sélectionnez la première sous-clef (la clef de chiffrement)…

```
gpg> key 1

sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb* rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
ssb  rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>
```

… et envoyez-là vers la carte à puce :

```
gpg> keytocard
Signature key ....: [none]
Encryption key ...: [none]
Authentication key: [none]

Please select where to store the key:
  (2) Encryption key
Your selection?
```

La première sous-clef n'ayant que la capacité de chiffrer, vous n'avez pas d'autre choix ici que de la stocker dans le slot réservé à la clef de chiffrement.

Saisissez, quand l'agent GnuPG vous le demande, la phrase de passe qui protège actuellement votre clef privée sur votre disque dur.

```
sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb* rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
     card-no: 0005 00001234
ssb  rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>
```

Notez la mention `card no:` qui indique que la sous-clef se trouve sur la carte portant le numéro de série `0005 00001234`.

Déselectionnez la première sous-clef puis répétez la procédure avec la seconde (la clef de signature) :

```
gpg> key 1

sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
     card-no: 0005 00001234
ssb  rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>

gpg> key 2

sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
     card-no: 0005 00001234
ssb* rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>

gpg> keytocard
Signature key ....: [none]
Encryption key ...: 249A 1E8B ADB0 1AF1 B4B0  C03A 06F6 DDA5 0B53 4621
Authentication key: [none]

Please select where to store the key:
  (1) Signature key
  (3) Authentication key
Your selection?
```

Stockez cette clef dans le slot `(1)`, réservé à la clef de signature. Le slot réservé à la clef d’authentification vous est aussi proposé car une clef capable de signer peut aussi servir à authentifier (l’authentification est un cas particulier de signature, où vous signez un défi posé par celui qui vous demande de vous authentifier).

```
sec  rsa4096/F5C95C96DD4C784D
     created: 2016-12-25  expires: 2019-12-25  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/06F6DDA50B534621
     created: 2016-12-25  expires: 2017-12-25  usage: E
     card-no: 0005 00001234
```

```
ssb* rsa2048/345F24439B63479D
     created: 2016-12-25  expires: 2017-12-25  usage: S
     card-no: 0005 00001234
ssb  rsa2048/5C18013DD38A4C90
     created: 2016-12-25  expires: 2017-12-25  usage: A
[ultimate] (1). Alice Smith <alice@example.org>
```

Répétez une dernière fois la procédure pour transférer la sous-clef d'authentification dans le slot prévu à cet effet, puis, quittez l'éditeur de clefs *en enregistrant vos modifications* — sinon les sous-clefs auront bien été transférées sur la carte, mais ne seront pas effacées de votre trousseau privé sur votre disque dur !

```
gpg> save
```

5. Signer ou déchiffrer des messages OpenPGP

Le fait que vos clefs privées soient désormais sur une carte à puce ne change à peu près rien à votre manière d'utiliser GnuPG tous les jours pour signer ou déchiffrer des messages OpenPGP. Et ce, indépendamment du programme que vous utilisez en avant-plan (que ce soit GnuPG directement en ligne de commande, un *frontend* graphique comme GNU Privacy Assistant, un greffon pour client de messagerie comme Enigmail, etc.).

Au moment de signer ou déchiffrer un message, l'agent GnuPG, au lieu de vous demander la phrase de passe protégeant le trousseau privé, vous demandera d'insérer votre carte dans le lecteur si elle n'y est pas déjà, puis vous demandera le PIN. En arrière-plan, le fonctionnement de GnuPG sera quelque peu différent (puisqu'il devra déléguer une partie du travail à la carte au lieu de le faire lui-même), mais ça ne sera pas visible pour vous (à part un éventuel clignotement de la diode témoin d'activité de votre lecteur de carte).

Le seul changement important réside dans le fait qu'une fois votre PIN saisi une première fois, il ne vous sera plus jamais demandé tant que vous ne retirez pas la carte du lecteur. Inutile d'aller jouer avec l'option `--default-cache-ttl` de l'agent GnuPG, ce n'est pas lui qui garde votre PIN en cache : c'est la carte OpenPGP elle-même qui reste dans un état « PIN vérifié » tant que la connexion entre la carte et Scdaemon est maintenue.

Pour contraindre ponctuellement la carte à vous redemander le PIN, il faut provoquer une réinitialisation de la connexion, soit en retirant physiquement la carte du lecteur, soit en envoyant une commande **SCD RESET** à l'agent GnuPG :

```
$ gpg-connect-agent 'SCD RESET' /bye
```

Le mode « PIN forcé » pour les signatures

La carte peut être configurée pour exiger *systématiquement* le PIN avant toute opération utilisant la clef de signature, indépendamment du fait que le PIN a déjà été vérifié ou non. C'est le mode *forced PIN*, dont l'état (enclenché ou non) est visible lorsque GnuPG affiche le contenu de la carte :

```
$ gpg --card-edit
[…]
Signature PIN ....: not forced
[…]
```

Ici, la carte est en mode normal, le PIN ne sera exigé que s'il n'a pas encore été vérifié depuis l'insertion de la carte. La commande **forcesig** permet de basculer entre le mode normal et le mode « PIN forcé » :

```
gpg/card> forcesig

gpg/card> list
[…]
Signature PIN ....: forced
[…]

gpg/card> quit
```

Cette option ne concerne que les opérations de signature. Pour le déchiffrement ou l'authentification, le PIN ne sera jamais exigé s'il a déjà été vérifié une première fois et que la carte n'a pas été réinitialisée entre-temps.

6. S'authentifier auprès d'un serveur SSH

Dans ce que j'appelle les « usages avancés » de la carte OpenPGP, l'authentification SSH est probablement l'un des plus intéressants.

Il faut pour cela remplacer l'agent SSH fourni en standard avec OpenSSH (**ssh-agent**) par l'agent GnuPG. Ajouter simplement l'option **enable-ssh-support** dans le fichier de configuration de l'agent GnuPG `$GNUPGHOME/gpg-agent.conf`. Puis, assurez-vous que l'agent GnuPG est démarré avant toute invocation de **ssh** et que la variable d'environnement `SSH_AUTH_SOCK` est définie et point vers la *socket* spécialement crée à l'intention des clients SSH. Pour cela, ajoutez les deux lignes suivantes dans votre fichier `~/.xprofile` (ou tout autre script exécuté au démarrage de votre session) :

```
gpg-connect-agent /bye
export SSH_AUTH_SOCK=$(gpgconf --list-dirs agent-ssh-socket)
```

L'outil **gpg-connect-agent** est un utilitaire permettant de communiquer avec l'agent GnuPG ; comme les autres composants de GnuPG, il lance automatiquement un nouvel agent s'il n'en détecte pas un déjà en cours d'exécution, de sorte que sa seule invocation suffit à forcer le démarrage de l'agent. L'outil **gpgconf** est ici utilisé pour être sûr de toujours obtenir l'emplacement exact de la *socket* de l'agent, cet emplacement étant susceptible de changer au fil des versions de GnuPG.

Si vous utilisiez précédemment **ssh-agent**, assurez-vous aussi qu'il n'est *plus* démarré au début de votre session.

L'agent GnuPG s'utilise comme l'agent SSH standard, et vous pouvez toujours interagir avec lui avec l'outil classique **ssh-add**. Vos clefs SSH pré-existantes sont toujours utilisables de la même façon qu'auparavant. Par exemple, pour charger votre clef par défaut (`~/.ssh/id_rsa`) dans l'agent, procédez comme d'habitude :

```
$ ssh-add
```

Pour utiliser la clef d'authentification que vous avez générée un peu plus tôt et transférée sur votre carte OpenPGP, vous n'avez rien d'autre à faire que d'insérer la carte dans le lecteur : l'agent GnuPG détectera automatiquement la présence d'une clef d'authentification et la rendra accessible aux clients SSH. Vous pouvez le vérifier en demandant à l'agent de lister les « identités » disponibles :

```
$ ssh-add -l
2048 SHA256:lmV1mszOtiqANZk0Dfp+7EbZYbfTLA/UDMe3Fd+ffG0 cardno:000500001234 (RSA)
```

Il ne vous reste plus qu'à exporter votre clef publique sous une forme utilisable dans un fichier `~/.ssh/authorized_keys`, ce que vous pouvez faire soit avec **ssh-add** encore (seulement quand la carte est dans le lecteur) :

```
$ ssh-add -L
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCsDukLjnTlZXbegDbQdByay1PLbmgsblv
8otFIzXlH66wa6XoRGc/VnKG6IjCGkzv/esaR0Mggc71zUxAGv+u/Yk//1VF0MUM1RDUv41
hMVQbn0CrPfpZ19s2tkqf36IiJZSsjbFc9IjYUvBLu891gO4P+ygSGRtHYs7ECdmkATEx+8
UTzcO8S/p1CPH6WdYj1oGMfza7onaaXdqY98XI2mb607ZvoqcJxfSB4QiyzvrqixB3bwWTL
rSjAXQ/t4nO1IfNoSVMjoSqoezO7JzpyXVPNuNzGSnrjc/5qyMiDU/k+m9AI0ubB1FRUJ+B
fSp9yxCCq679zNQWGR1Ro7J3B cardno:000500001234
```

Soit avec l'option `--export-ssh-key` de **gpg** :

```
$ gpg --export-ssh-key alice
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCsDukLjnTlZXbegDbQdByay1PLbmgsblv
8otFIzXlH66wa6XoRGc/VnKG6IjCGkzv/esaR0Mggc71zUxAGv+u/Yk//1VF0MUM1RDUv41
hMVQbn0CrPfpZ19s2tkqf36IiJZSsjbFc9IjYUvBLu891gO4P+ygSGRtHYs7ECdmkATEx+8
UTzcO8S/p1CPH6WdYj1oGMfza7onaaXdqY98XI2mb607ZvoqcJxfSB4QiyzvrqixB3bwWTL
rSjAXQ/t4nO1IfNoSVMjoSqoezO7JzpyXVPNuNzGSnrjc/5qyMiDU/k+m9AI0ubB1FRUJ+B
fSp9yxCCq679zNQWGR1Ro7J3B openpgp:0xD38A4C90
```

Une fois la clef publique connue du serveur, vous pouvez vous y connecter comme d'habitude, avec n'importe quelle commande SSH (**ssh**, **scp**, etc.) ou n'importe quel programme utilisant SSH en arrière-plan (**git push** par exemple). L'agent GnuPG vous demandera votre PIN si nécessaire ; s'il a déjà été vérifié plus tôt, vous serez automatiquement connecté sans avoir à saisir quoi que ce soit.

Notez que si vous tentez une connexion SSH alors que votre carte n'est pas disponible dans le lecteur, l'agent échouera à trouver la clef. Pour forcer l'agent à vous demander d'insérer votre carte dans ce cas de figure, ajoutez le *keygrip* de votre clef d'authentification ans le fichier `$GNUPGHOME/sshcontrol`.

Un *keygrip* est une empreinte calculée sur une clef « brute », qui permet à l'agent GnuPG d'identifier une clef indépendamment du protocole avec lequel elle est utilisé (OpenPGP, X.509, SSH). Utilisez les `--with-keygrip --list-secret-keys` de GnuPG pour afficher le *keygrip* de chacune de vos clefs.

7. Utiliser un certificat X.509

À partir de chacune des clefs privées de la carte OpenPGP, il est possible de créer un certificat X.509 [https://tools.ietf.org/html/rfc5280]. Un tel certificat permet d'utiliser la carte pour, entre autres usages, s'authentifier auprès d'un serveur TLS, signer ou chiffrer des messages S/MIME, ou encore signer des documents OpenDocument.

7.1. Créer un certificat X.509

GnuPG fournit l'outil **gpgsm** pour créer, manipuler et utiliser des certificats X.509. Mettez votre carte dans le lecteur et appelez **gpgsm** pour créer une nouvelle demande de certificat :

```
$ gpgsm --armor --output certificat.csr --gen-key

Please select what kind of key you want:
   (1) RSA
   (2) Existing key
   (3) Existing key from card
```

```
Your selection? 3

Serial number of the card: D2760001240102000005000012340000
Available keys:
   (1) 6BA62F5EFDB16B8F1D7407E12466166FE90424B8 OPENPGP.1
   (2) 543E2A5037F6C5B8C655255CA76DCF5FB0D9C9C0 OPENPGP.2
   (3) EB6B56BDCA0F3C18340DBAD272B468842519897C OPENPGP.3
Your selection?
```

La clef marquée `OPENPGP.1` est la clef de signature, la clef `OPENPGP.2` est la clef de chiffrement, et la clef `OPENPGP.3` est la clef d'authentification.

Le choix de la clef conditionnera les usages possibles du futur certificat : il sera utilisable seulement pour signer avec la clef `OPENPGP.1`, seulement pour chiffrer avec la clef `OPENPGP.2`, pour signer et s'authentifier avec la clef `OPENPGP.3`. Il n'est pas possible d'utiliser la carte pour produire un certificat utilisable à la fois pour signer/authentifier *et* pour chiffrer.

En pratique, le module Scute (décrit plus bas) ne permet que l'utilisation de la clef d'authentification, c'est donc cette que clef nous choisissons ici.

```
Possible actions for a RSA key:
   (1) sign, encrypt
   (2) sign
   (3) encrypt
Your selection?
```

La clef d'authentification ne permet pas de chiffrer, donc la seule option pertinente ici est la seconde, `(2) sign`. Ce menu ne sert qu'à déterminer la valeur de l'extension *X509v3 Key Usage* de la demande de certificat — même si vous choisissez `(1) sign, encrypt`, ça ne changera rien au fait que la clef sous-jacente ne permet *pas* de chiffrer, même si le certificat proclamera le contraire.

Vous devez ensuite renseigner le sujet du futur certificat :

```
Enter the X.509 subject name: CN=Alice,O=Example Corp,DC=example,DC=net
Enter email addresses (end with an empty line):
> alice@example.net
>
Enter DNS names (optional; end with an empty line):
>
Enter URIs (optional; end with an empty line):
>
Parameters to be used for the certificate request:
    Key-Type: card:OPENPGP.3
    Key-Length: 1024
    Key-Usage: sign
    Name-DN: CN=Alice,O=Example Corp,DC=example,DC=net
    Name-Email: alice@example.net

Really create request? (y/N) y
Now creating certificate request. This may take a while ...
gpgsm: about to sign CSR for key: &EB6B56BDCA0F3C18340DBAD272B468842519897C
gpgsm: certificate request created
Ready.  You should now send this request to your CA.
```

Vous récupérez la demande de certificat dans le fichier `certificat.csr`. Vous pouvez si vous le souhaitez l'examiner avec, par exemple, **openssl req -in *certificat.csr* -text**.

Il vous faut maintenant faire signer cette demande par une autorité de certification. Le choix de l'autorité de certification à laquelle vous adresser dépend grandement de l'usage que vous voulez faire du certificat. Il peut s'agir d'une autorité générique « reconnue » (probablement le meilleur choix pour signer des messages S/MIME à destination

de n'importe qui, puisque tout le monde fait confiance — à tort ou à raison — à ces autorités), d'une autorité spécifique à votre institution ou à votre compagnie (pour signer des messages internes à cette institution ou compagnie), ou même de votre propre autorité (par exemple pour vous authentifier auprès de votre propre serveur TLS).

Une fois en possession de votre certificat signé, importez-le dans le trousseau de **gpgsm** :

```
$ gpgsm --import certificat.crt
```

Vous devriez aussi importer le certificat racine de l'autorité qui l'a signé, ainsi que les éventuels certificats intermédiaires. Finalement, testez votre nouveau certificat en signant un fichier quelconque :

```
$ gpgsm --output quelconque.signed --sign quelconque
gpgsm: signature created
$ gpgsm --verify quelconque.signed
gpgsm: Signature made 2016-12-25 21:11:32 using certificate ID 0xAB4057B0
gpgsm: Good signature from "/CN=Alice/O=Example Corp/DC=example/DC=net"
gpgsm:                 aka "alice@example.net"
```

7.2. S'authentifier auprès d'un serveur web

Je supposerai ici que vous voulez accéder à un serveur web qui exige une authentification du client par certificat, et que vous souhaitez utiliser pour cela le certificat fraîchement obtenu ci-dessus.

Je ne traiterai pas de la mise en place de l'authentification côté serveur, qui sort du cadre de ce document (et qui est de toute façon assez bien documentée ailleurs). Je me contenterai de rappeler que les deux lignes suivantes, dans la configuration d'un serveur Apache httpd 2.4.x, obligent un client à présenter un certificat signé par la clef privé du certificat `/etc/ssl/certs/client_ca.pem` pour être autorisé à accéder au serveur :

```
SSLCACertificateFile    /etc/ssl/certs/client_ca.pem
SSLVerifyClient         require
```

En ces temps où les mots de passe sont de plus en plus mis à mal [http://arstechnica.com/security/2012/08/passwords-under-assault/], l'authentification par certificat (avec ou sans carte à puce) est une méthode de contrôle d'accès très intéressante et probablement trop peu utilisée. Combinée avec l'option `FakeBasicAuth` du module mod_ssl [https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#ssloptions], elle peut remplacer complètement l'authentification par login et mot de passe.

Similairement à ce que nous avons vu plus haut pour SSH, le serveur est indifférent au fait que le certificat que vous allez lui présenter a sa clef privée sur une carte à puce, il ne voit qu'un certificat X.509 comme un autre. C'est du côté du client que les choses intéressantes se passent.

Côté client, donc, vous devez avoir fait signer votre requête de certificat par l'autorité appropriée, et avoir importé le certificat signé dans le trousseau de **gpgsm**. Ensuite, il vous faut installer Scute [https://www.scute.org/], et configurer Firefox pour qu'il charge dynamiquement cette bibliothèque (la procédure complète, captures d'écrans à l'appui, est détaillée dans la documentation de Scute [https://www.scute.org/scute.html/Application-Configuration#Application-Configuration]).

En principe, Scute devrait aussi être utilisable par n'importe quelle application capable d'utiliser un module PKCS #11 [https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-

cryptographic-token-interface-standard.htm] (Chromium par exemple). Mais je n'ai testé que Firefox, LibreOffice et Thunderbird (voir plus bas).

Sitôt la carte OpenPGP insérée dans le lecteur, Scute rendra votre certificat disponible pour Firefox — vous pourrez le vérifier en allant constater sa présence dans le *Certificate Manager* de Firefox. En vous connectant au serveur exigeant un certificat client, Firefox sélectionnera automatiquement ce certificat[4], et vous serez invité à saisir votre PIN (si nécessaire) pour signer une partie de la poignée de main TLS, prouvant ainsi au serveur que vous possédez la clef privée du certificat.

Scute ne fonctionne avec TLS 1.2 qu'à partir de la version 1.5.0, sortie en juillet 2017. Les versions précédentes ne prennent en charge que TLS 1.0 et 1.1.

7.3. Signer des messages S/MIME

S/MIME (RFC 3851 [https://tools.ietf.org/html/rfc3851]) est, avec OpenPGP, l'autre standard proposant la confidentialité et l'authentification « de bout en bout » des messages.

Thunderbird prend en charge nativement S/MIME, mais ne permet pas l'utilisation d'une clef privée sur carte à puce. Comme pour Firefox, c'est la bibliothèque Scute qui va lui apporter cette fonctionnalité.

Comme pour TLS 1.2, Scute ne permet la signature de messages S/MIME que depuis sa version 1.5.0.

Installez Scute sous Thunderbird de la même manière que sous Firefox (dans les préférences, allez dans la section Advanced, onglet Certificates, Security Devices ; dans le « Device Manager », ajoutez un nouveau module PKCS#11 et spécifiez le chemin complet vers la bibliothèque `scute.so`). Une fois la carte dans le lecteur, votre certificat devrait apparaître dans l'onglet Your Certificates du « Certificate Manager ».

Dans les paramètres de votre compte de messagerie, visitez la section « Security » (attention à ne pas confondre avec la section « OpenPGP Security », présente si vous utilisez Enigmail, et qui comme son nom l'indique concerne OpenPGP et non S/MIME) et sélectionnez votre certificat X.509 pour signer vos messages. Cochez éventuellement la case « Digitally sign messages (by default) » si vous souhaitez systématiquement signer vos messages sortant (ce réglage est toujours ponctuellement désactivable au cas par cas.

Votre certificat X.509 est également utilisable à partir de Mutt, à condition que ce dernier utilise la bibliothèque gpgme (GnuPG Made Easy) en lieu et place d'OpenSSL comme *backend* cryptographique (en ajoutant l'option `set crypt_use_gpgme` dans le fichier de configuration de Mutt). Par l'intermédiaire de cette bibliothèque, Mutt déléguera les opérations de signature S/MIME à gpgsm, qui à son tour fera appel à la carte OpenPGP.

7.4. Signer des documents OpenDocument

Si vous avez installé la dernière version de Scute et configuré Firefox pour l'utiliser comme expliqué ci-avant, vous pouvez également utiliser votre certificat X.509 pour apposer une signature électronique à vos documents OpenDocument depuis LibreOffice.

Il suffit pour ça de définir la variable d'environnement `MOZILLA_CERTIFICATE_FOLDER` et de la faire pointer vers le dossier de votre profil Firefox :

[4] À moins que vous ayez plus d'un certificat signé par l'autorité attendue par le serveur, auquel cas il vous demandera de sélectionner vous-même celui qu'il faut utiliser.

```
export MOZILLA_CERTIFICATE_FOLDER=~/.mozilla/firefox/nom_du_profil
```

Dès lors, LibreOffice peut utiliser tous les certificats présents dans le Certificate Manager de Firefox, y compris celui que vous avez généré à partir de votre carte OpenPGP.

Pour signer un document, rien de plus simple : insérez votre carte dans le lecteur, et dans LibreOffice, allez dans File → Digital Signatures). Choisissez Sign Document…, sélectionnez votre certificat, entrez votre PIN si nécessaire.

7.5. Utiliser la carte comme *token* PKCS #15

Dans tous les cas d'utilisation du certificat X.509 présentés ci-dessus, la carte OpenPGP ne contient rien d'autre que la *clef privée* associée au certificat. Le certificat proprement dit est stocké dans le trousseau de GpgSM, où les applications doivent aller le chercher soit en appelant directement **gpgsm**, soit par l'intermédiaire de Scute.

Il est toutefois possible de stocker le certificat sur la carte OpenPGP elle-même, ce qui le rend utilisable indépendamment de GpgSM. En fait, cela transforme *de facto* la carte en un *token* PKCS #15 [https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-15-cryptographic-token-information-format.htm], utilisable avec n'importe quel système ou application compatible avec ce format.

Le certificat doit être au format DER pour pouvoir être importé sur la carte. Si vous l'aviez rangé dans le trousseau de GpgSM, la commande suivante le sortira directement dans ce format :

```
$ gpgsm -o alice.der --export alice@example.net
```

Si votre autorité de certification vous a remis un certificat au format PEM, vous pouvez utiliser OpenSSL (par exemple) pour le convertir en DER :

```
$ openssl x509 -inform PEM -in alice.pem -outform DER -out alice.der
```

Lancez l'éditeur de carte de GnuPG pour procéder au transfert du certificat sur la carte. Notez que la commande **writecert** n'apparaît pas dans l'aide en ligne de l'éditeur.

```
$ gpg --card-edit

Application ID ...: D2760001230102000000500001234000
[…]

gpg/card> admin
Admin commands are allowed

gpg/card> writecert 3 < alice.der

gpg/card> quit
```

Le premier argument de **writecert** est supposé indiquer le slot sur la carte dans lequel enregistrer le certificat. Toutefois, la carte OpenPGP dans sa version 2.x ne possède qu'un seul slot pour certificat, et cet argument doit toujours être 3 (la version 3.x possède trois slots, un pour chacune des clefs).

Votre carte PKCS #15 est désormais utilisable de manière autonome et ne requiert plus la présence des composants de GnuPG sur la machine. Toute application prenant en charge PKCS #15 (par exemple par l'intermédiaire d'OpenSC [https://github.com/OpenSC/OpenSC/wiki]) peut exploiter la carte.

L'utilisation simultanée de la carte avec GnuPG et comme *token* PKCS #15 est impossible. En effet, Scdaemon requiert un accès *exclusif* à la carte, interdisant son utilisation par d'autres programmes comme OpenSC. C'est d'ailleurs la raison d'être de Scute : permettre à des programmes indépendants de GnuPG d'accèder à la carte via une interface standard (PKCS #11) fonctionnant au-dessus de Scdaemon, et laisser ce dernier être le seul à exploiter la carte directement.

8. Miscellanées

8.1. Communiquer avec Scdaemon et la carte

Scdaemon, le démon auxiliaire de GnuPG gérant les cartes à puce, travaille normalement dans l'ombre des autres composants de GnuPG et l'utilisateur n'a jamais affaire à lui.

Il est néanmoins possible de dialoguer avec Scdaemon indépendamment des *frontend* de GnuPG. Cela peut occasionnellement être utile à des fins de débogage, pour d'éventuels usages avancés non prévus par les *frontend*, ou simplement par curiosité, pour comprendre ce qui se passe en arrière-plan.

Comme l'agent GnuPG, Scdaemon écoute sur une *socket* Unix et utilise le protocole Assuan [https://www.gnupg.org/documentation/manuals/assuan/index.html] pour recevoir ses commandes.

On peut utiliser l'outil **gpgconf** et son option `--list-dirs` (déjà évoqué dans la section Section 6, « S'authentifier auprès d'un serveur SSH ») pour obtenir le nom du dossier contenant la *socket* :

```
$ gpgconf --list-dirs socketdir
/run/user/1000/gnupg
```

Ici, la *socket* sera donc accessible à l'adresse `/run/user/1000/gnupg/S.scdaemon`. On pourra alors contacter le démon de la manière suivante :

```
$ echo SERIALNO | socat - unix-connect:/run/user/1000/gnupg/S.scdaemon
OK GNU Privacy Guard's Smartcard server ready
S SERIALNO D2760002401020000050000012340000 0
OK
```

(La commande **SERIALNO** demande à Scdaemon le numéro de la série de la carte OpenPGP actuellement insérée dans le lecteur.)

Mais il est probablement plus simple de passer par l'agent GnuPG (et son utilitaire **gpg-connect-agent**), qui fournit une commande **SCD** pour transmettre des commandes à Scdaemon. Cela permet de contacter Scdaemon sans avoir à se soucier de l'emplacement de sa *socket* :

```
$ gpg-connect-agent 'SCD SERIALNO' /bye
S SERIALNO D2760002401020000050000012340000 0
OK
```

Une fois qu'on sait communiquer avec Scdaemon, on peut obtenir la liste des commandes acceptées avec la commande **HELP**, et l'aide d'une commande particulière avec **HELP *commande*** :

```
$ gpg-connect-agent
```

```
> SCD HELP
[…]
# SERIALNO [<apptype>]
# LEARN [--force] [--keypairinfo]
# READCERT <hexified_certid>|<keyid>
# READKEY <keyid>
# SETDATA [--append] <hexstring>
# PKSIGN [--hash=[rmd160|sha{1,224,256,384,512}|md5]] <hexified_id<
# PKAUTH <hexified_id>
# PKDECRYPT <hexified_id>
# INPUT
# OUTPUT
# GETATTR <name>
# SETATTR <name> <value>
# WRITECERT <hexified_certid>
# WRITEKEY [--force] <keyid>
# GENKEY [--force] [--timestamp=<isodate>] <no>
# RANDOM <nbytes>
# PASSWD [--reset] [--nullpin] <chvno>
# CHECKPIN <idstr>
# LOCK [--wait]
# UNLOCK
# GETINFO <what>
# RESTART
# DISCONNECT
# APDU [--[dump-]attr] [--more] [--exlen[=N]] [hexstring]
# KILLSCD
OK
> SCD HELP READKEY
# READKEY <keyid>
#
# Return the public key for given cert or key ID as a standard
# S-expression.
#
# Note, that this function may even be used on a locked card.
OK
> /bye
```

Pour les plus aventureux, la commande probablement la plus intéressante est **ADPU**, qui permet d'envoyer des commandes APDU brutes à la carte (rapportez-vous à la spécification de la carte OpenPGP [https://g10code.com/docs/openpgp-card-2.0.pdf] pour le détail des commandes APDU). Par exemple, pour lire les « octets historiques » de la carte :

```
$ gpg-connect-agent --hex 'SCD APDU 00CA5F5200' /bye
D[0000]  00 31 C5 73 C0 01 40 05  90 00 90 00            .1.s..@.....
OK
```

Si la communication directe avec Scdaemon n'est normalement jamais nécessaire, il y a un cas de figure où elle peut s'avérer utile : pour découvrir certaines caractéristiques de la carte OpenPGP que GnuPG n'affiche pas par ailleurs.

On utilisera pour ça la commande **LEARN --force**, qui renvoie tout ce que le démon sait de la carte. En particulier, la ligne *EXTCAP* (*Extended Capabilities*) indique les fonctionnalités optionnelles prises en charge par la carte :

```
$ gpg-connect-agent 'SCD LEARN --force' /bye | grep "^S EXTCAP"
S EXTCAP gc=1+ki=1+fc=1+pd=1+mcl3=2048+aac=1+sm=0+si=5+dec=0+bt=0
```

Parmi les valeurs renvoyés, signalons :

- `gc=1`, indiquant que la carte est équipée d'un générateur de nombres aléatoires ;
- `pd=1`, indiquant que la carte prend en charge les *Private DOs* décrit en Section 1.3.2, « Les « champs privés » ou à usage indéfini » ;

- `mcl3=2048`, indiquant la taille maximale, en octets, du certificat qu'il est possible de stocker sur la carte comme expliqué en Section 7.5, « Utiliser la carte comme *token* PKCS #15 » ;
- `si=5`, le *Life Cycle Status Indicator* — une valeur de 5 signifie que la carte peut être ré-initialisée.

8.2. Ré-initialiser une carte bloquée

Comme expliqué dans la Section 1.2, « Sécurité de la carte », après trois saisies erronées consécutives du PIN utilisateur et du PIN administrateur, plus aucun PIN ne peut être vérifié et la carte est définitivement bloquée.

La spécification OpenPGP prévoit toutefois la possibilité de ré-initialiser une carte bloquée. La ré-initialisation efface toutes les données de la carte et la ramène à son état de sortie d'usine.

Cette fonctionnalité est optionnelle. Pour savoir si une carte donnée est ré-initialisable, il faut consulter les « capacités étendues » de la carte comme expliqué à la fin de la section précédente. Si le *Life Cycle Status Indicator* vaut 5, la carte est ré-initialisable.

Pour ré-initialiser une telle carte, il suffit d'utiliser la commande **factory-reset** de l'éditeur de carte de GnuPG :

```
$ gpg --card-edit
Application ID ...: D2760001240102000005000012340000
Version ..........: 2.0
[…]

gpg/card> admin
Admin commands are allowed

gpg/card> factory-reset
gpg: OpenPGP card no. D2760001240102000005000012340000 detected

gpg: Note: This command destroys all keys stored on the card!

Continue? (y/N) y
Really do a factory reset? (enter "yes") yes

gpg/card>
```

La commande sera sans effets si la carte utilisée n'offre pas la fonctionnalité de ré-initialisation.

A. À propos de ce document