Gnuk, NeuG, FST-01

Entre cryptographie et matériel libre

Damien Goutte-Gattat <dgouttegattat@incenp.org>

2018/06/27

Table des matières

Depuis quelques années, la *Free Software Initiative of Japan* (FSIJ [https://www.fsij.org/], association de promotion des logiciels libres au Japon) travaille entre autres sur un ensemble de projets à la croisée des chemins entre cryptographie, informatique embarquée et matériel libre.

Ces projets sont l'œuvre de Niibe Yutaka [https://www.gniibe.org/] (*gniibe* ou *g##*), qui entre autres casquettes est président de la FSIJ, coordinateur des traductions japonaises du projet GNU, développeur Debian, et contributeur à GnuPG (où il travaille notamment mais pas seulement sur la cryptographie à courbes elliptiques et la gestion des cartes à puce — deux aspects qui sont directement liés aux projets dont il va question dans cet article).

Votre serviteur avait déjà très brièvement évoqué l'existence de deux de ces projets (Gnuk et le FST-01) dans un article précédent consacré à la carte OpenPGP [https://incenp.org/dvlpt/docs/carte-openpgp.html] (dont la lecture est recommandée avant de poursuivre), mais sans donner aucun détail, ce que le présent article va corriger.

1. Gnuk, le jeton cryptographique USB

Gnuk [http://www.fsij.org/doc-gnuk/] (*GnuPG toKen*, mais le nom est en réalité dérivé d'une marque de produits pour bébés [https://www.nuk.co.uk/], comme le rappelle le logo du projet [https://www.fsij.org/images/gnuk/gnuk.png]) est une implémentation logicielle d'un « jeton » (*token*) cryptographique USB, ciblant les microcontrôleurs de la famille STM32F103 [http://www.st.com/en/microcontrollers/stm32f103.html?querycriteria=productId=LN1565]. Un tel microcontrôleur relié à un port USB et exécutant Gnuk sera vu par l'ordinateur hôte comme un lecteur de cartes à puce qui contiendrait une carte OpenPGP inamovible.

Gnuk est une implémentation *complètement libre* de la carte OpenPGP, à la différence par exemple de la carte OpenPGP physique de FLOSS-Shop [https://www.floss-shop.de/en/security-privacy/smartcards/13/openpgp-smart-card-v3.3?c=40] (anciennement *Kernel Concepts*) qui n'est que partiellement libre (son auteur n'a pas pu publier le code de certaines routines obtenues sous couvert d'un *Non-Disclosure Agreement*) ou de la

Yubikey 4 [https://www.yubico.com/product/yubikey-4-series/] dont le *firmware* est complètement propriétaire.

1.1. Plate-formes cibles

La plate-forme de prédilection pour exécuter Gnuk est le FST-01, un circuit spécifiquement conçu dans ce but par gniibe et qui sera détaillé dans une section dédiée plus loin.

Gnuk est aussi le moteur de la Nitrokey Start [https://shop.nitrokey.com/shop/product/nitrokey-start-6], qui est probablement le moyen le plus simple de se procurer un jeton Gnuk prêt à l'emploi.

En plus de la Nitrokey Start, la gamme Nitrokey contient deux autres jetons compatibles avec la spécification de la carte OpenPGP. Toutefois, seule la Nitrokey Start est basée sur Gnuk. La Nitrokey Pro et la Nitrokey Storage, qui descendent du projet Crypto-Stick [http://www.privacyfoundation.de/projekte/crypto_stick/crypto_stick_english/], embarquent en leur sein un exemplaire de la carte OpenPGP physique de FLOSS-Shop, et n'ont aucun lien avec Gnuk.

Gnuk peut aussi fonctionner sur plusieurs circuits construits autour d'un contrôleur STM32, parmi lesquels :

- la carte STM32-H103 [https://www.olimex.com/Products/ARM/ST/STM32-H103/] de Olimex ;
- la carte Maple Mini [https://www.leaflabs.com/maple/] de Leaflabs, parfois décrite comme un « Arduino-like dopé » (Leaflabs en a cessé la fabrication mais des clones sont disponibles [http://wiki.stm32duino.com/index.php?title=Maple_Mini] auprès d'autres constructeurs) ;
- la carte « Blue Pill » [http://wiki.stm32duino.com/index.php?title=Blue_Pill], ou « STM32F103C8T6 Minimum System Board », une des moins chères du marché (entre deux et trois dollars la carte auprès de revendeurs chinois) ;
- la carte d'évaluation et de développement NUCLEO-F103RB [http://www.st.com/en/evaluation-tools/nucleo-f103rb.html] de STMicroelectronics (qui est aussi le constructeur des microcontrôleurs STM32) — c'est la carte recommandée par gniibe si vous cherchez avant tout à expérimenter ;
- les cartes STBee et STBee Mini [http://strawberry-linux.com/stbee/] de Strawberry-Linux.

1.2. Fonctionnalités et conformité

Gnuk implémente la dernière version de la spécification de la carte OpenPGP (version 3.3.1 [https://gnupg.org/ftp/specs/OpenPGP-smart-card-application-3.3.pdf], publiée en août 2017), et de manière générale suit d'assez près l'évolution de la spécification (en 2015, Gnuk a été la première implémentation à prendre en charge la version 3.0 de la spécification, quelques semaines seulement après sa publication).

Comme toutes les cartes OpenPGP, Gnuk permet d'utiliser trois clefs privées (une clef de signature, une clef de déchiffrement, une clef d'authentification). Gnuk prend en charge à la fois les clefs RSA et les clefs basées sur les courbes elliptiques (qui sont la principale nouveauté de la carte OpenPGP en version 3.x).

Du côté de RSA, en pratique seules des clefs de 2048 bits sont utilisables. Il est en théorie possible d'utiliser des clefs de 4096 bits, mais les performances du microcontrôleur STM32F103 avec des clefs de cette taille sont rédhibitoires (une opération de signature

avec une clef RSA 4096 peut prendre jusqu'à *dix secondes*) ; il n'est d'ailleurs même pas possible de *générer* une clef RSA de 4096 bits directement sur le jeton, la mémoire disponible n'est tout simplement pas suffisante — si vous tenez absolument à utiliser une clef RSA de 4096 bits malgré tout, il vous faudra la générer sur l'ordinateur hôte puis l'importer sur le jeton.

Notez que *seules* les clefs de 2048 et 4096 bits sont prises en charge ; Gnuk ne permet pas l'utilisation de clefs de taille inférieure (par exemple 1024 bits), intermédiaire (par exemple 3072 bits), ou « exotique » (par exemple 2768 bits).

Du côté de la cryptographie à courbe elliptique, Gnuk prend en charge les courbes suivantes :

- la courbe `secp256r1` (aussi appelée *NIST P-256*, parce que définie dans le standard FIPS 186-4 [https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf] du *National Institute of Standards and Technology*) ;
- la courbe `secp256k1`, notoirement utilisée par Bitcoin [https://en.bitcoin.it/wiki/Secp256k1] ;
- la courbe `Curve25519`, définie dans le RFC 7748 [https://tools.ietf.org/html/rfc7748].

Officiellement, à l'heure actuelle seule la courbe `secp256r1` fait partie du standard OpenPGP (depuis le RFC 6637 [https://tools.ietf.org/html/rfc6637]). Il ne fait guère de doutes que la courbe `Curve25519` sera intégrée à la prochaine révision du standard (elle a été introduite dès le premier brouillon du RFC 4880bis, introduction qui n'a jamais été contestée depuis), on peut donc générer aujourd'hui des clefs utilisant cette courbe sans crainte pour la compatibilité future. En revanche l'avenir de la courbe `secp256k1` dans OpenPGP est incertain ; GnuPG la prend en charge, de même que OpenPGP.js, mais personne n'a jamais proposé de l'ajouter au standard.

Au-delà des types de clefs pris en charge, il y a quelques autres points à noter concernant l'implémentation de la spécification par Gnuk, pêle-mêle :

- Gnuk offre une prise en charge encore considérée « expérimentale » du mécanisme *KDF-DO* introduit dans la version 3.3 de la spécification et visant à assurer une meilleure protection des clefs privées (décrit dans une section suivante).
- Les « champs à usage privé » (*Private-Use DOs*), une partie optionnelle de la spécification, ne sont pas supportés.
- Pour des raisons liées à des détails d'implémentation, le PIN utilisateur ne peut pas être changé si le jeton ne contient aucune clef privée. Lors de la première utilisation d'un nouveau jeton, il faut importer ou générer sur place au moins une clef avant de pouvoir changer le PIN utilisateur installé par défaut par un PIN personnalisé.
- Gnuk propose un mode *admin-less*, dans lequel le PIN utilisateur et le PIN administrateur (`PW1` et `PW3`, dans le jargon de la spécification) sont confondus. Dans ce cas, la taille minimale imposée du PIN utilisateur est de huit caractères (elle est normalement de six caractères en mode « normal »).

1.3. Protection des clefs

L'objectif premier d'un jeton cryptographique comme la carte OpenPGP est de protéger les clefs privées. Comment Gnuk remplit-il cet objectif ?

Les routines cryptographiques utilisées dans Gnuk proviennent de la bibliothèque mbedTLS [https://tls.mbed.org/] (anciennement *PolarSSL*). Le code de ces routines fournit une première ligne de défense contre certaines attaques par canaux cachés (par exemple les *timing attacks*, basées sur le temps passé à effectuer les opérations crytogra-

phiques) ou certaines attaques par faute (par exemple l'attaque de Boneh, Demillo et Lipton [https://eprint.iacr.org/2012/553], où l'injection d'une faute matérielle — *glitch* — lors d'une opération de signature RSA permet de déduire la clef signante). De telles contremesures sont aujourd'hui classiques dans tout logiciel de cryptographie (on les trouve aussi dans GnuPG par exemple).

Contre l'extraction directe des clefs privées, Gnuk se repose *en partie* sur le mécanisme de protection de la mémoire Flash (RDP, *Read Data Protection*) fourni sous une forme ou sous une autre par la plupart des microcontrôleurs, dont le microcontrôleur STM32F103. Une fois le RDP activé, il n'est en principe pas possible de lire le contenu de la mémoire Flash, et la désactivation de la protection entraîne automatiquement son effacement complet. Ce genre de protection n'est toutefois pas à toute épreuve, comme l'ont par exemple montré des chercheurs allemands l'année dernière en contournant le mécanisme RDP [https://www.usenix.org/conference/woot17/workshop-program/presentation/obermaier] d'un microcontrôleur STM32F0, permettant une extraction complète du contenu de la Flash en quelques heures (tester si l'attaque fonctionne sur un modèle STM32F103 est laissé en exercice au lecteur — STMicroelectronics affirme que non [https://community.st.com/thread/46432-hacking-readout-protection-on-stm32#comment-181542]). Et cette protection est probablement inopérante contre les attaques dites *invasives* (où l'attaquant peut aller jusqu'à détruire physiquement le microcontrôleur) — il existe des entreprises chinoises spécialisées dans ce genre d'attaques, y compris contre le STM32F103 [https://lists.gnupg.org/pipermail/gnupg-users/2018-June/060601.html], vraisemblablement à des fins de contrefaçon.

Gnuk implémente une protection supplémentaire en stockant les clefs privées sous forme chiffrée, le PIN utilisateur servant de clef de chiffrement, de sorte qu'un attaquant qui réussirait à extraire les clefs du microcontrôleur devrait *aussi* réussir à obtenir le PIN. Une limitation évidente de cette protection est que le PIN est typiquement choisi dans un espace assez petit (Gnuk autorise des PIN de 127 caractères Unicode, mais la plupart des utilisateurs choisiront sans doute un PIN numérique de 6 à 10 chiffres) et donc possiblement *brute-forceable*.

Pour renforcer le chiffrement des clefs sur le jeton, la version 3.3 de la spécification de la carte OpenPGP a introduit le mécanisme *KDF-DO*. Le principe est qu'avant d'être envoyé au jeton le PIN saisi par l'utilisateur est passé à la moulinette d'une fonction de dérivation de clef (KDF, *Key Derivation Function*), qui transforme un PIN de taille arbitraire en une clef de 32 à 64 octets (selon l'algorithme de condensation utilisée au sein de la fonction de dérivation). Les paramètres de la fonction de dérivation sont eux-mêmes stockés dans un champs dédié du jeton (KDF-DO, *KDF Data Object*). L'objectif est de rendre la recherche exhaustive du PIN trop longue et/ou trop coûteuse, même si le PIN lui-même est choisi dans un espace de taille restreinte.

Gniibe a ajouté le support du mécanisme KDF-DO dans la version 2.2.6 de GnuPG. Pour mettre en œuvre ce mécanisme, il suffit d'appeler la commande **kdf-setup** dans l'éditeur de de cartes de GnuPG, *avant* de stocker une clef sur le jeton :

```
$ gpg --card-edit
Reader ...........: Free Software Initiative of Japan Gnuk (FSIJ-1.2.10) 00 00
Application ID ...: D276000124010200FFFE123456780000
[…]
Signature key ....: [not set]
Encryption key ...: [not set]
Authentication key: [not set]

gpg/card> admin
Admin commands are allowed.

gpg/card> kdf-setup

gpg/card>
```

Une critique parfois formulée à l'encontre de Gnuk est qu'une implémentation de la carte OpenPGP sur un microcontrôleur généraliste ne saurait offrir la même protection qu'une carte à puce spécialisée, et que Gnuk ne serait au mieux qu'une sorte de « carte à puce du pauvre ». On retrouve cette idée dans le tableau comparatif de Nitrokey [https://www.nitrokey.com/#comparison], qui met en avant le fait que seules la Nitrokey Pro et la Nitrokey Storage embarquent une *tamper-resistant smart card* qui « empêche l'extraction des clefs » — sous-entendu, la Nitrokey Start de son côté laisse vos clefs vulnérables.

Pour autant, d'une part Gnuk ne laisse *pas* les clefs dénuées de toute protection comme on vient de le voir, et d'autre part la « résistance aux attaques physiques » (*tamper resistance*) n'est ni l'apanage des cartes à puce, ni absolue. Carte à puce ou pas, il est toujours prudent de considérer qu'un adversaire motivé et équipé *pourra* extraire des données du jeton, si besoin en dépiotant la puce et en allant directement lire la valeur des bits au microscope. Faire reposer votre sécurité sur une vague assurance de *tamper resistance* est simpliste (et dangereux s'il s'avère que votre adversaire *est* motivé et équipé).

Pour ce que ça vaut, mon opinion personnelle est que je préfère un jeton m'offrant une protection peut-être imparfaite mais dont je connais le code et les mécanismes, à une carte à puce m'offrant une protection certainement imparfaite également mais qui dépend d'informations que le constructeur de la puce refuse de dévoiler.

1.4. Utilisation avec GnuPG

Je ne détaillerai pas trop l'*utilisation* de Gnuk, puisqu'un jeton Gnuk se comporte exactement comme une carte OpenPGP (et pour cause, *c'est* une carte OpenPGP), dont l'utilisation a déjà été couverte dans un précédent article [https://incenp.org/dvlpt/docs/carte-openpgp.html]. Quelques points méritent tout de même d'être mentionnés ou rappelés.

Gnuk est compatible avec la norme CCID, ce qui signifie d'une part qu'un jeton Gnuk est reconnu sans heurts par PCSC-Lite, le *middleware* standard de gestion des cartes à puce, et son pilote `ccid` ; et d'autre part qu'un tel jeton est aussi reconnu nativement par le pilote CCID intégré à Scdaemon, le démon auxiliaire de GnuPG chargé de l'interaction avec les cartes à puce.

Vous avez donc le choix entre deux chemins différents pour permettre à GnuPG d'utiliser votre jeton : soit vous utilisez PCSC-Lite, qui accèdera exclusivement au jeton, Scdaemon communiquant alors avec PCSC-Lite ; soit vous n'utilisez pas PCSC-Lite, et Scdaemon accèdera au jeton directement. Dans le premier cas, c'est le compte utilisateur associé au démon PCSC-Lite qui doit avoir les droits d'accès au jeton ; dans le second, c'est votre propre compte utilisateur, sous lequel tournent les démons auxiliaires de GnuPG dont Scdaemon, qui doit avoir ces droits d'accès.

Dans la règle Udev suivante, tout périphérique Gnuk — identifié par le *Vendor ID* de la FSIJ (`234b`), et le *Product ID* de Gnuk (`0000`) — est assigné au compte `scard`, qui chez moi est le compte sous lequel tourne le démon de PCSC-Lite. Adaptez cette règle à votre situation et à votre système.

```
ACTION=="add", SUBSYSTEM=="usb", ATTRS{idVendor}=="234b", ATTRS{idProduct}=="0000", \
  OWNER="scard", GROUP="scard", MODE="660", ENV{ID_SMARTCARD_READER}="1"
```

Si vous utilisez une Nitrokey Start, notez que Nitrokey utilise ses propres identifiants USB, distincts de ceux de la FSIJ : Nitrokey a pour *Vendor ID* `20a0` et la Nitrokey Start a pour *Product ID* `4211`.

Vérifiez ensuite que GnuPG a connaissance de votre jeton :

```
$ gpg --card-status
```

```
Reader ...........: Free Software Initiative of Japan Gnuk (FSIJ-1.2.10) 00 00
Application ID ...: D276000124010200FFFE123456780000
[…]
```

Je vous renvoie à l'article susmentionné pour remplir les différents champs du jeton (nom du titulaire, préférences linguistiques, etc.) et importer des clefs privées. Le PIN administrateur par défaut, qui vous sera demandé avant toute modification, est normalement `12345678`, et le PIN utilisateur est `12356`. Souvenez-vous que vous ne pouvez changer le PIN utilisateur qu'*après* avoir importé au moins une clef privée, et que si voulez bénéficier du mécanisme KDF-DO pour une protection accrue des clefs comme décrit dans la section précédente, vous devez l'activer *avant* d'importer la moindre clef.

2. NeuG, le générateur de nombres aléatoires

NeuG [https://www.gniibe.org/memo/development/gnuk/rng/neug.html] (le nom est dérivé de la prononciation japonaise de *noisy*) est une implémentation d'un générateur de nombres aléatoires pour microcontrôleurs STM32F103, utilisant comme source d'entropie le bruit des convertisseurs analogique/numérique intégrés à ces microcontrôleurs.

NeuG a été initialement développé comme composant de Gnuk, fournissant le générateur de nombres aléatoires pour le jeton OpenPGP (notamment utilisé pour la génération de clefs directement sur le jeton) ; NeuG est une version *stand-alone* du générateur de nombres aléatoires intégré à Gnuk.

Il se présente comme un périphérique USB CDC/ACM (*Communication Device Class/Abstract Control Model*), autrement dit un périphérique USB émulant un port série ; un tel périphérique est typiquement représenté sous GNU/Linux par un fichier `/dev/ttyACM0`. Il suffit de lire depuis ce fichier pour extraire des octets aléatoires du périphérique.

Si vous voulez utiliser le générateur depuis un compte utilisateur normal, assurez-vous d'avoir accès au périphérique en lecture *et* en écriture (l'accès en écriture est nécessaire pour configurer correctement le périphérique comme on le verra ci-dessous), par exemple avec la règle Udev suivante :

```
ACTION=="add", SUBSYSTEM=="tty", \
  ATTRS{idVendor}=="234b", ATTRS{idProduct}=="0001", \
  GROUP="users", MODE="660"
```

Comme on l'a vu plus haut avec Gnuk, `234b` est le *Vendor ID* de la FSIJ ; `0001` est le *Product ID* de NeuG.

Avant de pouvoir lire des octets aléatoires, il faut configurer la *discipline de ligne* du (pseudo-)port série en mode brut (*raw*) :

```
$ stty -F /dev/ttyACM0 -echo raw
```

Enfin, il faut choisir le mode de *conditionnement*, qui détermine le(s) filtre(s) appliqué(s) sur les octets extraits des convertisseurs analogique/numérique. Le choix du mode se fait en réglant les options de parité du port série.

- Aucun filtre : configurez le port série pour utiliser un bit de parité pair (**stty -F /dev/ttyACM0 parenb -parodd**).
- Filtre CRC-32 : configurez le port série pour utiliser un bit de parité impair (**stty -F /dev/ttyACM0 parenb parodd**).

- Filtres CRC-32 et SHA-256 : désactivez le bit de parité (**stty -F /dev/ttyACM0 -parenb**).

Vous pouvez maintenant extraire des octets aléatoires du périphérique. Par exemple, pour créer un fichier de 32 kB avec **dd** :

```
$ dd if=/dev/ttyACM0 of=random_data bs=1024 count=32
```

3. Fraucheky, le conteneur GPL

Fraucheky [http://git.gniibe.org/gitweb/?p=chopstx/fraucheky.git;a=summary] est un projet annexe de Gniibe visant à résoudre un problème bien spécifique se posant à quelqu'un qui voudrait distribuer un périphérique USB dont le microcode est un logiciel libre sous licence GPL (comme par exemple un jeton Gnuk ou NeuG). La GPL, entre autres obligations, demande de transmettre une copie de la licence aux utilisateurs (*give all recipients a copy of this License along with the Program*) ; comment satisfaire cette obligation lorsque le programme en question est exécuté sur un microcontrôleur branché sur un port USB, sans interface utilisateur propre (donc sans menu du genre « à propos de ce logiciel ») ?

Une solution non-technique qui serait sans doute tout-à-fait valable vis-à-vis de la GPL serait de simplement fournir le texte de la licence sur un livret accompagnant le périphérique, mais ce ne serait ni écologique ni amusant.

À la place, Fraucheky, la solution imaginée par Gniibe, est une implémentation minimaliste du protocole *USB Mass Storage*, permettant au périphérique de se faire temporairement passer pour une clef USB. Concrètement, lorsqu'il est connecté à l'ordinateur, le périphérique embarquant Fraucheky apparaît initialement comme une banale clef USB étiquetée FRAUCHEKY, contenant un exemplaire de la licence GPL ainsi qu'un fichier README décrivant le périphérique. C'est seulement lorsque le volume FRAUCHEKY est démontée que Fraucheky fait apparaître la vraie nature du périphérique (NeuG par exemple) et que celui-ci devient utilisable.

Pour éviter à l'utilisateur d'avoir à monter et démonter le volume FRAUCHEKY avant chaque utilisation du périphérique, Fraucheky a un mécanisme de désactivation permanente représenté par un dossier appelé DROPHERE. Toute opération d'écriture dans ce dossier (par exemple un glisser-déposer de n'importe quel fichier, comme le fichier de la GPL commodémment situé juste « à côté ») entraîne la disparition de Fraucheky : dès lors, le périphérique USB apparaîtra systématiquement sous sa véritable identité dès la connexion à l'ordinateur hôte, sans plus nécessiter un cycle de montage/démontage.

4. FST-01

Le FST-01 [https://www.gniibe.org/FST-01/fst-01.html] (Flying Stone Tiny 01) est un circuit imprimé spécifiquement imaginé par Gniibe pour faire tourner Gnuk ou NeuG. Sa conception se veut minimaliste et le circuit n'accueille, autour d'un microcontrôleur STM32F103 (avec ses 128 ko de mémoire Flash et 20 ko de RAM), qu'un terminateur USB, un oscillateur à quartz, un régulateur de tension, et une LED témoin d'activité.

Le FST-01 est un matériel libre dont Gniibe fournit tous les plans [https://git.gniibe.org/gitweb/?p=gnuk/fst-01.git], Vous pouvez donc construire vous-même votre propre FST-01.

Si vous n'êtes pas à ce point adepte du Do-It-Yourself (ou si votre insoleuse est en panne), vous pouvez obtenir un FST-01 déjà tout fait directement auprès de Gniibe, soit via son site personnel [https://www.gniibe.org/category/shop.html] (en japonais uniquement), soit en personne si vous avez l'occasion de le rencontrer lors d'un évènement du monde libre

(DebConf par exemple) — il a généralement quelques exemplaires sur lui pour ces occasions. Un FST-01 pré-chargé avec NeuG et Fraucheky est également en vente auprès de la Free Software Foundation [https://shop.fsf.org/storage-devices/neug-usb-true-random-number-generator].

Figure 1. Le FST-01 de l'auteur, similaire à celui vendu par la FSF

Le FST-01 a aussi été disponible pendant un temps auprès de Seeed Studio [https://www.seeedstudio.com/], mais il a depuis été retiré, même si le produit apparaît toujours [https://wiki.seeedstudio.com/FST-01/] sur leur wiki.

4.1. Charger Gnuk ou NeuG sur un FST-01

Vous pouvez être amenés à devoir flasher votre FST-01 pour plusieurs raisons : vous avez obtenu (ou fabriqué vous-même) un FST-01 « vierge » ; vous avez obtenu un FST-01 pré-chargé avec NeuG (par exemple celui vendu par la FSF) alors que vous avez plutôt besoin de Gnuk, ou inversement ; ou vous souhaitez remplacer le Gnuk ou le NeuG pré-chargé par une version plus récente.

4.1.1. Compiler Gnuk

Dans tous les cas, il faut bien sûr commencer par compiler le programme de votre choix (disons Gnuk, pour cet exemple — la procédure pour NeuG est similaire).

Pour ça, il vous faut une chaîne de compilation ciblant l'architecture `arm-none-eabi`. Votre distribution vous en fournit peut-être une (c'est le cas par exemple sous Debian, il vous suffit d'installer le paquet `gcc-arm-none-eabi` ; pour Slackware, un *SlackBuild* est disponible [https://www.slackbuilds.org/repository/14.2/development/arm-gcc/]), sinon vous pouvez en obtenir une auprès de Arm [https://developer.arm.com/open-source/gnu-toolchain/gnu-rm], pré-compilée ou sous forme de sources.

ARMé de votre compilateur, vous pouvez maintenant télécharger les sources de Gnuk :

```
$ git clone git://git.gniibe.org/gnuk/gnuk.git
[…]
$ cd gnuk/src
$ git submodule update --init
```

Lors de la configuration des sources, choisissez la cible `FST_01` ou `FST_01G` selon votre modèle (le FST-01 est le modèle initial ; le FST-01G [https://www.gniibe.org/memo/development/fst-01/fst-01-revision-g.html] est la seconde génération).

Vous devez aussi choisir le *Vendor ID* et le *Product ID* de votre futur jeton. Pour un usage personnel, vous pouvez utiliser le *Vendor ID* de la FSIJ (`234b`) et le *Product ID* de Gnuk (`0000`). Pour un usage commercial (comme ce que fait Nitrokey avec la Nitrokey Start

par exemple), il vous appartient d'obtenir votre propre *Vendor ID* auprès de l'USB Implementers Forum [http://www.usb.org/].

Je vous suggère d'activer également l'option `--enable-factory-reset`, qui rend possible la ré-initialisation du jeton OpenPGP après blocage (certes, si vous bloquez votre jeton vous pouvez toujours le re-flasher contrairement à une carte OpenPGP physique, mais utiliser la commande **factory-reset** de GnuPG est quand même plus simple).

Selon vos besoins, activez également l'option `--enable-certdo`, qui rend possible le stockage d'un certificat X.509 directement sur le jeton.

```
$ ./configure --target=FST_01G \
  --vidpid=234b:0000 \
  --enable-factory-reset \
  --enable-certdo
Header file is: board-fst-01g.h
Debug option disabled
Configured for bare system (no-DFU)
PIN pad option disabled
CERT.3 Data Object is supported
Card insert/removal by HID device is NOT supported
Life cycle management is supported
$ make
[…]
arm-none-eabi-objcopy -O binary build/gnuk.elf build/gnuk.bin
```

À l'issue de la compilation, vous devez obtenir deux fichiers `build/gnuk.elf` et `build/gnuk.bin`. Le premier est l'exécutable au format ELF généré par le compilateur, le second est une version binaire « brute ». Vous aurez besoin de l'un ou l'autre selon la méthode utilisée pour flasher votre FST-01.

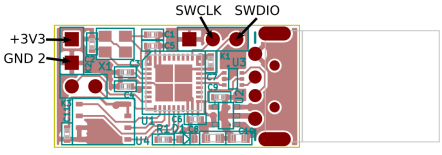
4.1.2. Flashage via SWD

C'est la seule méthode possible si votre FST-01 est « vierge » ou s'il est chargé avec autre chose que Gnuk ou NeuG. Vous aurez besoin d'un débogueur SWD (Gniibe recommande le ST-Link/V2 [http://www.st.com/en/development-tools/st-link-v2.html] ou un ses clones) et éventuellement d'OpenOCD [http://openocd.org/].

Connectez le débogueur ST-Link/V2 au FST-01 comme indiqué sur la Figure 2 :

- Broche 2 du ST-Link/V2 (`MCU VDD`) vers le port `+3V3` du FST-01 (dans le coin en haut à gauche, si le FST-01 est orienté avec le port USB vers la droite) ;
- Broche 4 (`GND`) vers le port `GND 2` (en-dessous du précédent) ;
- Broche 7 (`SWDIO`) vers le port `SWDIO` (dans le coin en haut à droite, juste derrière le connecteur USB) ;
- Broche 9 (`SWCLK`) vers le port `SWCLK` (à gauche du précédent).

Figure 2. Schéma du FST-01 indiquant les ports de connexion au débogueur SWD



Si nécessaire, désactivez la protection de la mémoire Flash du microcontrôleur STM32F103 (cela effacera la mémoire en question) :

```
$ openocd -f interface/stlink-v2.cfg -f target/stm32f1x.cfg \
  -c init \
  -c "reset halt" \
  -c "stm32f1x unlock 0" \
  -c reset -c exit
```

Puis procédez au flashage proprement dit :

```
$ openocd -f interface/stlink-v2.cfg -f target/stm32f1x.cfg \
  -c "program build/gnuk.elf verify reset exit"
```

Et réactivez la protection de la mémoire Flash derrière vous :

```
$ openocd -f interface/stlink-v2.cfg -f target/stm32f1x.cfg \
  -c init \
  -c "reset halt" \
  -c "stm32f1x lock 0" \
  -c reset -c exit
```

OpenOCD peut éventuellement être remplacé par le script `tool/stlinkv2.py` fourni avec les sources de Gnuk et de NeuG, et que Gniibe a écrit à une époque où OpenOCD ne prenait pas encore en charge le débogueur ST-Link/V2. L'équivalent des commandes ci-dessus est dans ce cas :

```
$ ../tool/stlinkv2.py -u
$ ../tool/stlinkv2.py build/gnuk.bin
```

La première commande désactive la protection de la mémoire, la deuxième reprogramme le microcontrôleur (et active à nouveau la protection de la mémoire après). Notez que cette commande attend un fichier binaire (`build/gnuk.bin`), pas un fichier au format ELF.

4.1.3. Flashage via reGNUal

Si votre FST-01 est déjà chargé avec Gnuk ou NeuG, vous pouvez le reflasher sans avoir besoin d'un débogueur. Tant Gnuk que NeuG permettent la mise à jour du logiciel exécuté sur le microcontrôleur directement via le port USB. Cela peut être utilisé soit pour passer à une version plus récente du même programme (par exemple, de Gnuk 1.2.8 à Gnuk 1.2.9) ou pour passer d'un programme à l'autre — le cas probablement le plus typique étant le remplacement de NeuG par Gnuk, sur le FST-01 vendu par la Free Software Foundation.

Cette fonctionnalité est encore considérée comme *expérimentale* et Gniibe conseille de se munir tout de même d'un débogueur SWD « au cas où » — si la mise à jour se passe mal, votre FST-01 peut se retrouver « briqué » et il vous faudra alors vous rabattre sur le flashage via SWD comme décrit dans la section précédente.

Pour ce que ça vaut, j'ai expérimenté cette méthode à plusieurs reprises sans jamais rencontrer de problèmes, mais évidemment *your mileage may vary*…

Pour utiliser cette méthode, après avoir compilé Gnuk comme ci-dessus, compilez également *reGNUal*, le programme de mise à jour. Il est dans le dossier `regnual` dans les sources de Gnuk.

```
$ cd ../regnual
```

```
$ make
[…]
arm-none-eabi-objcopy -Obinary regnual.elf regnual.bin
```

Les choses diffèrent ensuite selon que le FST-01 que vous voulez reflasher contient déja Gnuk ou NeuG.

S'il contient une copie de Gnuk, utilisez le script `tool/upgrade_by_passwd` fourni avec Gnuk (notez que le script doit être appelé avec les droits du superutilisateur) :

```
$ cd ../tool
$ sudo ./upgrade_by_passwd.py ../regnual/regnual.bin ../src/build/gnuk.bin
```

Le PIN administrateur de votre jeton Gnuk vous sera demandé, puis le script chargera sur le microcontrôleur le programme de mise à jour (`regnual.bin`) qui lui-même s'occupera de reprogrammer le microcontrôleur ave le programme final (`gnuk.bin`).

Si votre FST-01 contient déjà NeuG, vous aurez besoin d'utiliser le script `tool/neug_upgrade.py`, fourni avec les sources de NeuG.

```
$ cd ../..
$ git clone git://git.gniibe.org/gnuk/neug.git
$ cd neug/tool
$ sudo ./neug_upgrade.py ../../gnuk/regnual/regnual/bin ../../gnuk/src/build/gnuk.bin
```

5. Émuler Gnuk/NeuG sous GNU/Linux

En plus des cibles matérielles listées plus haut (Section 1.1, « Plate-formes cibles »), Gnuk et NeuG peuvent aussi fonctionner sous GNU/Linux en émulant un périphérique USB. Cela n'apporte bien sûr aucun des bénéfices que fournit l'utilisation d'un vrai périphérique (par exemple, dans le cas de Gnuk, ça ne met pas les clefs privées hors d'atteinte du système hôte ; dans le cas de NeuG, ça ne constitue pas une source physique d'entropie), mais peut être utile à des fins de développement et de tests. Dans le cas de Gnuk, ça peut notamment représenter un moyen simple d'essayer sans frais une pseudo-carte OpenPGP.

L'émulation repose sur le protocole USB/IP [http://usbip.sourceforge.net/], normalement conçu pour permettre le partage de périphériques à travers le réseau mais légèrement détourné dans le cas présent, les programmes Gnuk ou NeuG se faisant passer pour un serveur USB/IP exportant un pseudo-périphérique.

Pour tester ce mode émulation, assurez-vous d'abord que les modules `usbip-core` et `vhci-hcd` sont chargés :

```
# modprobe usbip-core
# modprobe vhci-hcd
```

Puis, assurez-vous que vous disposez du programme client `usbip`. S'il n'est pas fourni par votre distribution, vous le trouverez dans les sources du noyau Linux, dans le répertoire `tools/usb/usbip`.

Puis, rendez-vous dans les sources de Gnuk et compilez le programme en choisissant pour cible `GNU_LINUX` :

```
$ cd gnuk/src
$ ./configure --target=GNU_LINUX
Header file is: board-gnu-linux.h
```

```
Debug option disabled
Configured for bare system (no-DFU)
PIN pad option disabled
CERT.3 Data Object is NOT supported
Card insert/removal by HID device is NOT supported
Life cycle management is NOT supported
$ make
```

À l'issue de la compilation, vous devez obtenir un exécutable `build/gnuk`.

Avant de pouvoir vous en servir, vous devez préparer un fichier qui représentera la mémoire Flash de votre pseudo-périphérique Gnuk. Le script `gnuk/tool/gnuk-emulation-setup` fera ça pour vous :

```
$ ../tool/gnuk-emulation-setup mon-image-flash
```

Lancez l'émulateur en lui passant en paramètres les identifiants de périphérique USB à utiliser ainsi que l'image que vous venez de créer :

```
$ ./build/gnuk --vidpid=234b:0000 mon-image-flash
```

Maintenant que l'émulateur tourne, le client **usbip** doit vous indiquer la présence d'un serveur USB/IP sur l'adresse locale, exportant un seul périphérique :

```
# usbip list --remote=127.0.0.1
Exportable USB devices
======================
 - 127.0.0.1
        1-1: unknown vendor : unknown product (234b:0000)
           : /sys/devices/pci0000:00/0000:00:01.1/usb1/1-1
           : (Defined at Interface level) (00/00/00)
```

« Attachez » ce périphérique pour le rendre disponible sur votre machine :

```
# usbip attach --remote=127.0.0.1 --busid=1-1
```

Et voilà. Vous pouvez maintenant expérimenter avec votre (pseudo-)périphérique comme si vous aviez un vrai périphérique connecté à votre ordinateur. Par exemple, demandez à GnuPG de vous donner les détails du jeton :

```
$ gpg --card-status
Reader ...........: Free Software Initiative of Japan Gnuk (FSIJ-1.2.10-EMULATED) 00 00
Application ID ...: D276000124010200FFFEF1420A7A0000
Version ..........: 2.0
Manufacturer .....: unmanaged S/N range
Serial number ....: F1420A7A
Name of cardholder: [not set]
Language prefs ...: [not set]
Salutation .......:
URL of public key : [not set]
Login data .......: [not set]
Signature PIN ....: forced
Key attributes ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 127 127 127
PIN retry counter : 3 3 3
Signature counter : 0
Signature key ....: [none]
Encryption key ...: [none]
Authentication key: [none]
General key info .: [none]
```

Pour « débrancher » votre pseudo-périphérique :

```
# usbip detach --port=0
```

La procédure pour émuler NeuG au lieu de Gnuk est identique, à celà près que l'émulateur NeuG n'a pas besoin d'un fichier représentant la mémoire Flash du microcontrôleur — NeuG n'a aucun état à stocker dans ladite mémoire, contrairement à Gnuk.

A. À propos de ce document