Table des matières

Les Clés Sécurité Multi-Protocoles

Linux

Installation

Ubuntu

```
sudo add-apt-repository ppa:yubico/stable && sudo apt-get update
```

A Installer

Programme	Commande d'installation
YubiKey Manager (CLI)	sudo apt install yubikey-manager
YubiKey Personalization Tool	sudo apt install yubikey-personalization-gui
libpam-yubico	sudo apt install libpam-yubico
libpam-u2f	sudo apt install libpam-u2f

Debian

Il ne sera pas nécessaire d'ajouter le Repo !

```
apt-get install pamu2fcfg libpam-u2f libpam-yubico
```

Paramétrer

Clé Initial

NE PAS ÊTRE SUDO POUR PARAMÉTRER VOTRE CLÉ

Voici les commande à éxecuter pour associer une Clé a un compte utilisateur:

Dans un terminal, être positionné dans votre profil utilisateur.

1. `mkdir -p ~/.config/Yubico`

2. `pamu2fcfg > ~/.config/Yubico/u2f_keys`

3. Saisir votre code PIN de clé
4. Ensuite quand la clé Clignote, toucher la pour faire l'association.

Clé de Secour

Toujours avec un terminal se repositionner dans votre profil (ou l'endroit ou est le fichier de configuration).

1. Brancher votre Clé

2. `pamu2fcfg -n >> ~/.config/Yubico/u2f_keys`

3. Saisir votre code PIN de clé
4. Quand la clé Clignote, toucher la pour faire l'association.

Avec SUDO

En lieux et place de la saisie du password pour sudo, il faudra seulement valider avec la Clé

Sudo & Session Clé Only

Si vous souhaitez juste le valider avec votre clé votre ouverture de session.
Modifier alors ce fichier:

```
sudo nano /etc/pam.d/common-auth
```

En tout début de fichier insérer:

```
auth       sufficient   pam_u2f.so
```

Dès que vous appellerez la commande “SUDO” dans un terminal, il vous suffira seulement d'appuyer sur votre clé pour valider celle-ci.

Session Key + Passwd

Il vous faudra saisir votre password et avoir votre Clé pour vous authentifier!!
Pour le systèmes après Ubuntu 17.10:

```
sudo nano /etc/pam.d/gdm-password
```

Pour les systèmes avant Ubuntu 17.10:

```
sudo nano /etc/pam.d/lightdm
```

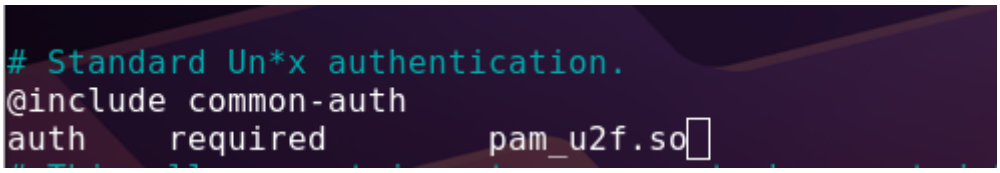
Après *@include common-auth* ajouter :

```
auth       required   pam_u2f.so
```

Session Clé Only

```
sudo nano /etc/pam.d/login
```

Après *@include common-auth* ajouter :




```
auth       required   pam_u2f.so
```

Verrouiller Session

Pour vous permettre de verrouiller votre session dès lors que la clé est retirée.

```
sudo nano /usr/local/bin/lockscreen.sh
```

Vous pouvez décider d'un endroit différent si vous le souhaiter.

Script:

```
#!/bin/sh

# Ce script, quand il est appelé, verrouille le session si la yubikey est
```

```
absente.

sleep 2

if ! ykman info >> /dev/null 2>&1
then
loginctl lock-sessions
fi
```

Rendre le script exécutable:

```
sudo chmod +x /usr/local/bin/lockscreen.sh
```

Ajouter Entrée UDEV

```
sudo nano /etc/udev/rules.d/20-yubikey.rules
```

Ajouter:

```
ACTION=="remove", ENV{SUBSYSTEM}=="usb", ENV{PRODUCT}=="1050/407/526",
RUN+="/usr/local/bin/lockscreen.sh"
```

Monitorer UDEV

```
# Brancher la Yubikey
# monitor les actions systèmes
udevadm monitor --environment --udev

# Débrancher la yubikey, puis CTRL + C pour stopper le monitoring
```

Retour:

```
ID_VENDOR=Yubico
ID_VENDOR_ID=1050
ID_MODEL_ID=0407
ID_REVISION=0526
Code
```

Il ne vous restera plus qu'a modifier la variable ENV{PRODUCT}==“xxxx/xxx/xxx” en conséquence.
ENV{PRODUCT}==“xxxx/xxx/xxx” ID_VENDOR_ID, ID_MODEL_ID et ID_REVISION vous obtiendrez 1050/407/526
Recharger la configuration UDEV:

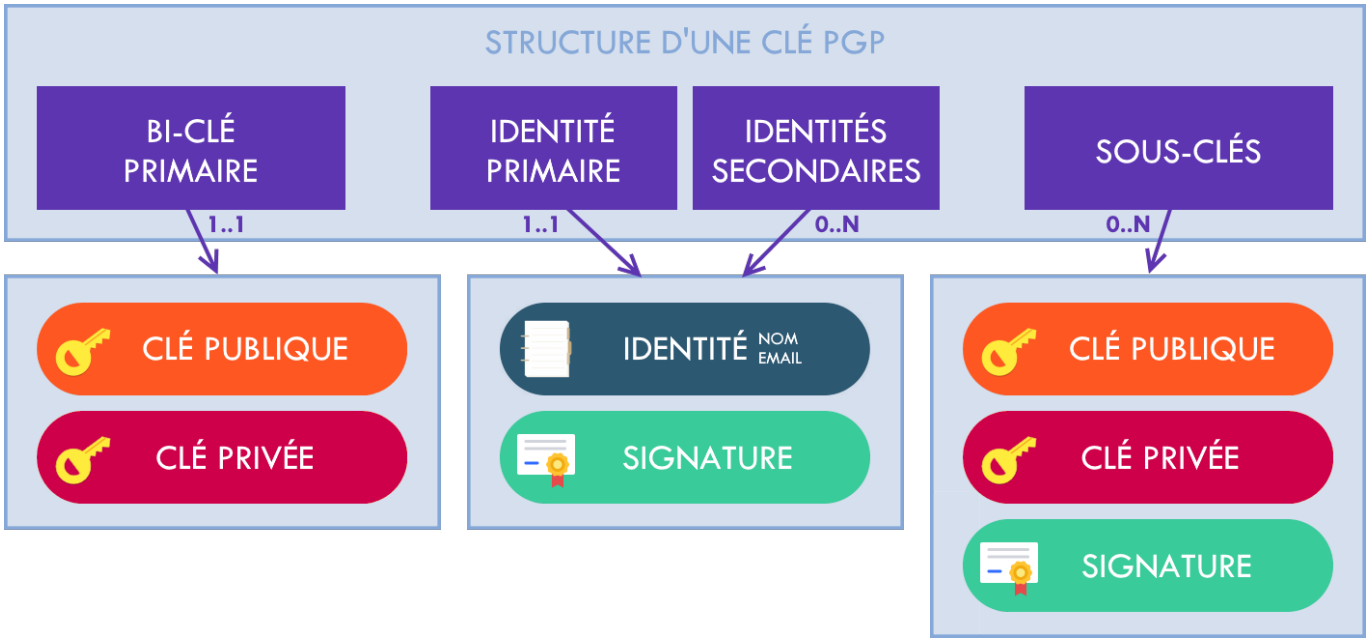
```
sudo udevadm control --reload-rules
```

GPG & Yubikey

Structure d'une clé PGP

Une clé PGP n'est pas simplement une bi-clé publique/privée comme le sont les clés SSH, elles contiennent également des métadonnées et éventuellement d'autres clés. De manière générale, une clé PGP est constituée de quatre éléments :

- Une clé primaire (ou clé maître) : il s'agit d'une bi-clé utilisée pour signer les informations contenues dans la clé PGP. L'ID de cette clé primaire (une empreinte de 20 octets, soit 40 caractères, défini dans la RFC 4880-12.2) est utilisée pour référencer la clé PGP.
- Une identité primaire : c'est elle qui définit le nom et l'adresse email du détenteur de la clé PGP.
- Une ou plusieurs identités secondaires optionnelles : Si généralement le nom est le même pour toutes les identités (je ne vous prive pas de vous inventer des surnoms), il est commun d'associer plusieurs adresses email à la clé PGP.
- Éventuellement des sous-clés : il s'agit là aussi de bi-clé utilisées pour signer, chiffrer ou s'authentifier. Ces clés sont certifiées par la clé primaire.



On peut se rendre compte qu'une clé PGP est en réalité une véritable structure pouvant contenir une ou plusieurs bi-clés ainsi que des métadonnées concernant le propriétaire de la clé. Initialement, lorsque Zimmermann décrit PGP dans la RCF 1991, une clé PGP ne comporte qu'une seule bi-clé. Lorsque la notion de sous-clés a été ajouté, le terme de "clé PGP" au singulier est resté.

Les sous-clés

On distingue quatre actions pour les clés PGP : la certification, la signature, le chiffrement et l'authentification. Seule la clé primaire possède le pouvoir de certification. Les sous-clés sont des bi-clés signée par la clé primaire. La clé publique permet de vérifier une signature et chiffrer un document, tandis que la clé privée permet de signer ou déchiffrer un document, mais aussi à s'authentifier.

- La sous-clé de chiffrement : Elle est automatiquement créée par défaut et signée par la clé primaire. Elle permet de chiffrer les documents, les mails, etc.
- La sous-clé de signature : Elle permet de signer les documents, les mails, etc. Par défaut, cette

fonction est remplie par la clé primaire.
- La sous-clé d'authentification : Elle est utilisée pour s'authentifier auprès de services externes tels que SSH. Par défaut, cette fonction est remplie par la clé primaire.

Chaque sous-clé a sa propre date d'expiration (optionnelle) et peut être révoquées indépendamment des autres. Il peut être alors judicieux de n'attribuer qu'un seul rôle à chaque clé, une clé primaire utilisée pour signer les sous-clés et les identités, et trois sous-clés utilisées respectivement pour signer, chiffrer et s'authentifier.

Les cartes à puces

Pour terminer cette (longue) introduction, parlons rapidement des cartes à puces (ou smart-card), telle que la Yubikey. Ce sont simplement des supports physiques pour stocker des clés privées et réalisée des opérations cryptographiques directement sur la carte.

Les cartes à puce sont conçues de telle sorte qu'une fois les clés privées ont été importées sur le périphérique, elles ne peuvent plus être extraites. Ainsi, même en cas de compromission de la machine, un attaquant ne pourra pas obtenir les clés privées.

Configurer la Yubikey

Maintenant que les bases sont posées, il est temps de générer une clé PGP et de configurer notre Yubikey pour pouvoir signer et chiffrer des documents, mais aussi utiliser SSH avec une sous-clé stockée sur la Yubikey.

Créer une paire de Clé

le paramètre expert permetd'avoir toutes les options.

```
gpg --expert --full-generate-key
```

Choisir l'option (**8) RSA (set your own capabilities)**, désactiver tous les rôles sauf **Certify** option **S** puis **Q** et générer une clé de 4096 bits.

Ensuite vérifier que la clé a bien été créée :

```
gpg --list-keys
pub   rsa4096 2022-04-18 [C] [expire : 2027-04-17]
      B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid           [  ultime ] Sylvain Key (Ma clé pour certifier) <mon-
mail@me.com>
gpg --list-secret-keys
sec#  rsa4096 2022-04-18 [C] [expire : 2027-04-17]
      B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid           [  ultime ] Sylvain Key (Ma clé pour certifier) <mon-
mail@me.com>
```

On a donc une **clé publique** et une **clé secrète** (privée) dont le rôle est de certifier **[C]** les autres informations de la clé PGP. On a également une identité **uid**.

L'étape suivante est de générer nos trois sous-clés :

```
gpg --expert --edit-key B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
gpg> addkey
```

GnuPG dispose d'une interface de commande pour l'édition des clés. Pour ajouter une clé, saisir addkey, puis choisir l'option **(8) RSA (set your own capabilities)**, désactiver tous les rôles **sauf Sign** Oprion **S** puis **Q** et générer une clé de 4096 bits.

Libre à vous de générer des clés ECC en sélectionnant l'option **(11) ECC (set your own capabilities)**.

Répéter l'opération pour **générer les deux autres clés** avec les rôles **Encrypt** et **Authenticate**, puis enregistrer les modifications :

```
gpg> quit
Save changes? (y/N) y
```

Créer un certificat de révocation

Pourquoi ?

Qu'est ce qu'un certificat de révocation ?

Si vous oubliez votre mot de passe, ou si votre clé privée est compromise ou perdue, ce certificat de révocation peut être publié pour notifier aux autres que votre clé publique ne doit plus être utilisée. On peut toujours se servir d'une clé publique révoquée pour vérifier des signatures que vous avez faites par le passé, mais on ne peut s'en servir pour chiffrer de nouveaux messages à votre attention. Cela n'affecte pas non plus votre capacité à déchiffrer les messages qui vous ont été adressés précédemment, si vous avez toujours accès à votre clé privée.
Source

Avoir une clef GPG, c'est bien. Avoir créer le certificat de révocation associé à cette clef, c'est mieux. Cela se fait de façon simple, en ligne de commande. On tape la commande :

```
gpg --output revoke.asc --gen-revoke mykey
```

mykey = id de clé ou mail utilisé pour identifié la clé.
Suivez les instructions.

Comment l'utiliser

Dès lors où la clef est compromise (vol du PC ou autre) et même si l'on a une copie de la clef privée sur une autre machine, dès lors qu'il y a une copie de la clef privée qui se ballade dans la nature, il faut révoquer sa clef (et en créer une autre par la suite). Pour celà, on utilise à nouveau gpg en ligne

de commande et lui donnant comme fichier le fameux certificat de révocation.

```
gpg --import revoke.asc
```

Et ensuite :
export vers le serveur de clef avec la commande suivante :

```
gpg --keyserver subkeys.pgp.net --send KEYNAME
```

Et lors de la prochaine synchronisation avec les serveurs de clefs, les personnes qui utilisaient la clef publique seront informées qu'elle n'est plus à utiliser.

Il faut donc penser à régulièrement mettre à jour ses clefs locales par synchronisation avec les serveurs.

Contrôlé les Clé

```
gpg --list-key
/home/sylvain/.gnupg/pubring.kbx
---------------------------------
pub   rsa4096 2022-04-18 [C] [expire : 2027-04-17]
      B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid           [  ultime ] Sylvain Key (Ma clé pour certifier) <mon-
mail@me.com>
sub   rsa4096 2022-04-18 [S] [expire : 2027-04-17]
sub   rsa4096 2022-04-18 [A] [expire : 2027-04-17]
sub   rsa4096 2022-04-18 [E] [expire : 2027-04-17]

gpg --list-secret-key
/home/sylvain/.gnupg/pubring.kbx
---------------------------------
sec   rsa4096 2022-04-18 [C] [expire : 2027-04-17]
      B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid           [  ultime ] Sylvain Key (Ma clé pour certifier) <mon-
mail@me.com>
ssb  rsa4096 2022-04-18 [S] [expire : 2027-04-17]
ssb  rsa4096 2022-04-18 [A] [expire : 2027-04-17]
ssb  rsa4096 2022-04-18 [E] [expire : 2027-04-17]
```

On retrouve donc nos **sous-clés** privées **(ssb)** et publiques **(sub)** pour signer **[S]**, chiffrer **[E]** et s'authentifier **[A]**.

Exporter les Clés

Nous n'avons plus besoin à présent de la clé privée primaire qui a été utilisée pour signer

l'identité et les sous-clés. Mais on va tout de même éviter de la supprimer : on ne sait jamais, elle pourrait nous servir ... On va donc exporter toutes nos clés sur une clé USB vierge dédiée au stockage de notre clé PGP.

```
gpg --armor --output /mnt/usb/secret-keys.txt --export-secret-key
B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Vous pouvez choisir une autre destination pour pour l'export de vos clés

Après l'importation, nous aurons récupéré les sous-clés mais pas la clé primaire.

```
gpg --armor --output ./secret-subkeys.txt --export-secret-subkeys
B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
gpg --delete-secret-key B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
gpg --import ./secret-subkeys.txt
rm ./secret-subkeys.txt
gpg --list-secret-keys
/home/sylvain/.gnupg/pubring.kbx
--------------------------------
sec#  rsa4096 2022-04-18 [C] [expire : 2027-04-17]
      B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid          [  ultime ] Sylvain Key (Ma clé pour certifier) <mon-
mail@me.com>
ssb>  rsa4096 2022-04-18 [S] [expire : 2027-04-17]
ssb>  rsa4096 2022-04-18 [A] [expire : 2027-04-17]
ssb>  rsa4096 2022-04-18 [E] [expire : 2027-04-17]
```

On observe que la clé **sec**rète est à présent suivie d'un **#**. Cela indique que la clé privée n'est pas stockée sur le poste. Nous sommes prêts pour importer les sous-clés sur la Yubikey.

Configurer la Yubikey

Assurez vous d'avoir les packages pcscd et scdaemon d'installé.

Après avoir branché la Yubikey, vérifions que la communication se fait !

```
 gpg-connect-agent --hex "scd apdu 00 f1 00 00" /bye
D[0000]  04 03 04 90 00                                        .....
OK
```

Personnaliser

```
gpg --card-edit
```

Le plus important ici est de modifier les codes **PIN** par défaut. Et oui, ça ne sert pas à grand-chose de protéger ses clés privées sur une Yubikey si tout le monde peut les utiliser (à condition de détenir physiquement votre Yubikey).

Dans l'interface d'édition de la Yubikey, tapez **admin** afin d'activer les commandes d'administration. Ensuite, tapez **passwd** pour **changer le code PIN et le code PIN admin**. Par défaut, le PIN est 123456 et le PIN admin est 12345678
Sauvegardez vos modifications et quitter l'interface d'édition de la Yubikey.

Importer nos clés sur la Yubikey

importer les trois sous-clés sur la Yubikey. Pour cela, il nous faut la console d'édition de notre clé PGP.

```
gpg --edit-key B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
gpg (GnuPG) 2.2.20; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret subkeys are available.

pub  rsa4096/03C8XXXXXXXXXXX
     créé : 2022-04-18  expire : 2027-04-17  utilisation : C
     confiance : ultime        validité : ultime
ssb  rsa4096/F96CXXXXXXXXXXX
     créé : 2022-04-18  expire : 2027-04-17  utilisation : S
ssb  rsa4096/57C19AC550FE79E2
     créé : 2022-04-18  expire : 2027-04-17  utilisation : A
ssb  rsa4096/9F51XXXXXXXXXXX
     créé : 2022-04-18  expire : 2027-04-17  utilisation : E
[  ultime ] (1). Sylvain Key (Ma clé pour certifier) <mon-mail@me.com>

gpg>
```

Ensuite, nous sélectionnons les clés une à une pour les importer sur la Yubikey.

```
gpg> key 1      # Sélection de la première sous-clé à transférer
gpg> keytocard  # Déplacement de la clé sur la carte à puce
Please select where to store the key: 1 (Signing)

gpg> key 1      # Désélection de la première sous-clé
gpg> key 2      # Sélection de la seconde sous-clé à transférer
gpg> keytocard  # Déplacement de la clé sur la carte à puce
Please select where to store the key: 2 (Encryption)

gpg> key 2      # Désélection de la seconde sous-clé
gpg> key 3      # Sélection de la troisième sous-clé à transférer
```

```
gpg> keytocard  # Déplacement de la clé sur la carte à puce
Please select where to store the key: 3 (Authentication)

gpg> quit
Save changes? (y/N) y
```

Maintenant que les sous-clés ont été déplacées sur la Yubikey, si nous listons les clés, nous verrons qu'elles ne sont plus stockées localement mais pointent vers la carte à puce (Le suffixe **>** signifie "pointeur vers une clé") :

```
gpg --list-secret-keys
/home/sylvain/.gnupg/pubring.kbx
--------------------------------
sec#  rsa4096 2022-04-18 [C] [expire : 2027-04-17]
      B327DD2BF8XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid          [  ultime ] Sylvain Key (Ma clé pour certifier) <mon-
mail@me.com>
ssb>  rsa4096 2022-04-18 [S] [expire : 2027-04-17]
ssb>  rsa4096 2022-04-18 [A] [expire : 2027-04-17]
ssb>  rsa4096 2022-04-18 [E] [expire : 2027-04-17]
```

Contrôle

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

Vous devriez voir votre clé primaire et vos sous clès, avec leur dates d'expirations et slots peuplés.

Exporter sa Clé Pub

```
gpg --export --armor mon@email.com > ma_cle_publique_gpg.asc
```

PKCS & Yubikey

Utiliser la Yubikey avec SSH:
Charger le module PAM pour l'utilsation de sa clé RSA avec l'agent SSH:

```
ssh-add -s /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

Décharger le module de l'agent:

```
ssh-add -e /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

Visauliser sa Clé ssh-rsa:

```
ssh-add -L
```

Ligne de Commande

Avec GPG

```
gpg --edit-card
```

Permet d'éditer la clé.

```
gpg/carte> help
quit            quitter ce menu
admin           afficher les commandes d'administration
help            afficher cette aide
list            afficher toutes les données disponibles
fetch           récupérer la clef indiquée dans l'URL de la carte
passwd          menu pour modifier ou déverrouiller le code personnel
verify          vérifier le code personnel et afficher toutes les données
unblock         débloquer le code personnel en utilisant un code de
réinitialisation
```

```
gpg/carte> admin
Les commandes d'administration sont permises
```

Permet d'afficher toutes les commandes disponible du mode admin.

```
gpg/carte> help
quit            quitter ce menu
admin           afficher les commandes d'administration
help            afficher cette aide
list            afficher toutes les données disponibles
name            modifier le nom du détenteur de la carte
url             modifier l'URL pour récupérer la clef
fetch           récupérer la clef indiquée dans l'URL de la carte
login           modifier l'identifiant de connexion
lang            modifier les préférences de langue
salutation      change card holder's salutation
cafpr           modifier une empreinte d'autorité de certification
forcesig        inverser le paramètre obligeant à entrer le code personnel
pour les
signatures
generate        générer de nouvelles clefs
passwd          menu pour modifier ou déverrouiller le code personnel
verify          vérifier le code personnel et afficher toutes les données
unblock         débloquer le code personnel en utilisant un code de
```

```
réinitialisation
factory-reset  destroy all keys and data
kdf-setup      setup KDF for PIN authentication
key-attr       change the key attribute
```

Pour sortir proprement:

```
quit
```

Avec Ykman

```
ykman --help
Usage: ykman [OPTIONS] COMMAND [ARGS]...

  Configure your YubiKey via the command line.

  Examples:

    List connected YubiKeys, only output serial number:
    $ ykman list --serials

    Show information about YubiKey with serial number 0123456:
    $ ykman --device 0123456 info

Options:
  -d, --device SERIAL             Specify which YubiKey to interact with by
serial number.
  -r, --reader NAME               Use an external smart card reader.
Conflicts with --device and
                                  list.
  -l, --log-level [DEBUG|INFO|WARNING|ERROR|CRITICAL]
                                  Enable logging at given verbosity level.
  --log-file FILE                 Write logs to the given FILE instead of
standard error; ignored
                                  unless --log-level is also set.
  --diagnose                      Show diagnostics information useful for
troubleshooting.
  -v, --version                   Show version information about the app
  --full-help                     Show --help, including hidden commands,
and exit.
  -h, --help                      Show this message and exit.

Commands:
  info     Show general information.
  list     List connected YubiKeys.
  config   Enable or disable applications.
  fido     Manage the FIDO applications.
  oath     Manage the OATH application.
  openpgp  Manage the OpenPGP application.
```

```
  otp      Manage the YubiOTP application.
  piv      Manage the PIV application.
```

RSA vs ECC

Quelles sont les différences entre RSA et ECC ?

Le facteur de différenciation clé entre l'ECC et RSA est la taille de la clé comparé à la force de chiffrement. Comme le montre le graphique ci-dessus, l'ECC peut fournir la même force de chiffrement qu'un système basé sur l'algorithme RSA, mais avec des clés beaucoup plus courtes. Par exemple, une clé ECC de 256 bits équivaut à des clés RSA de 3072 bits, qui sont alors 50% plus longues que les clés de 2048 bits couramment utilisées à l'heure actuelle.

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

Les derniers algorithmes symétriques en date plus sécurisés qui sont utilisés pour TLS (par ex. AES) font usage des clés de 128 bits. Il est donc tout à fait logique que les clés asymétriques fournissent au moins ce niveau de sécurité.

Les clés plus courtes font de l'ECC une option très attrayante pour les périphériques dont la capacité de stockage ou la puissance de traitement est limitée, ce qui devient de plus en plus courant à l'ère de l'Internet des Objets.

En ce qui concerne les cas d'utilisation de serveur Web plus classiques, les clés plus courtes peuvent se transcrire en de plus rapides négociations SSL (pouvant ainsi mener à une accélération de la vitesse de chargement des pages web) et une sécurité renforcée.

ECC (en anglais Elliptic curve cryptography) ou la cryptographie sur les courbes elliptiques.

Doc technique Yubikey / Yubico Guide Système ykman CLI Command

---