Rapport de Projet : Utilisation des YubiKeys pour l'Authentification Sécurisée

CNS M1

Zhipeng WANG

DIALLO Boubacar

Responsible: Pascal PETIT

Paritie 1: Gestion de code TOTP sur yubikey

maquette :

Machine 1 : Le "serveur" Debian où vous configurerez SSH avec TOTP, U2F, et FIDO2, et où la YubiKey sera principalement utilisée pour la configuration initiale. (192.168.5.100)

Machine 2 : Le "client" Debian qui se connectera à Machine 1 via SSH pour tester les authentifications sécurisées.(192.168.5.101)

Dans Machine1

- **Vérifiez la détection :**

ykman info

```
root@debian:~#  ykman info
Device type: YubiKey 5 NFC
Serial number: 12026569
Firmware version: 5.2.4
Form factor: Keychain (USB-A)
Enabled USB interfaces: OTP, FIDO, CCID
NFC transport is enabled.

Applications    USB             NFC
FIDO2           Enabled         Enabled
OTP             Enabled         Enabled
FIDO U2F        Enabled         Enabled
OATH            Enabled         Enabled
YubiHSM Auth    Not available   Not available
OpenPGP         Enabled         Enabled
PIV             Enabled         Enabled
root@debian:~#
```

Nous voyons les protocoles activées et la version du firmware

Réinitialisation de la YubiKey

nouveau

PIN: 123456

PUK: 12345678

Management Key: 010203040506070801020304050607080102030405060708

j'ai installer YubiKey Manager

sudo apt install yubikey-manager -y

et reset tous les modes de sécurité

Sur Machine 1 (Serveur – 192.168.5.100)

j'installer les dépendances :

sudo apt update && sudo apt install libpam-google-authenticator qrencode -y

Générer le secret TOTP avec

google-authenticator

```
Your new secret key is: YEC4MKOID2PNVHO23PBURGUZVE
Enter code from app (-1 to skip): 
```

maintenant nous allons Configurer PAM pour SSH pour spécifie les règles d'authentification pour les connexions SSH.

Sur Machine 1 (serveur) : /etc/pam.d/sshd

nous allons ajouté cette ligne:

auth required pam_google_authenticator.so

auth required :pour Indique que cette étape d'authentification est obligatoire

pam_google_authenticator.so :permet à mon utilisateur de vérifie le code TOTP saisi par rapport à la clé secrète stockée ici c'est root

Nb: Ce fichier est invoqué lors de l'étape keyboard-interactive définie dans sshd_config

1. Sur Machine 1 (serveur) : /etc/ssh/sshd_config

Il définit comment les connexions SSH entrantes sont gérées, quelles méthodes d'authentification sont autorisées, et leurs priorités

nous allons ajouté :

PubkeyAuthentication yes

ChallengeResponseAuthentication yes

AuthenticationMethods publickey,keyboard-interactive

pour Active l'authentification par clé publique et Activé le mécanisme d'authentification interactive, qui est utilisé par le module PAM pour demander le code TOTP et la fin Définir une liste ordonnée de méthodes d'authentification obligatoires. Ici, SSH doit réussir à la fois la vérification de la clé publique (publickey) et une interaction clavier (keyboard-interactive, qui déclenche le TOTP).

Redémarrez SSH pour appliquer les changements

sudo systemctl restart sshd

sur mon serveur ou la yubikey est connetée

ouvir l'application Yubico Authenticator

Ajouter le secret à la Yubikey

Ouvrir Yubico Authenticator.

Branchez la Yubikey.

Cliquez sur Ajouter un compte > Manuellement.

Saisissir :

Issuer : SSH_ServerSSH ici possible par mail mais ici j'utilise directement ma machine serveur .

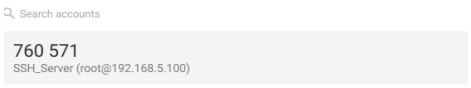
Account Name: root@192.168.5.100 possible de le faire avec

Secret : la clé dans le fichier ~/.google_authenticator

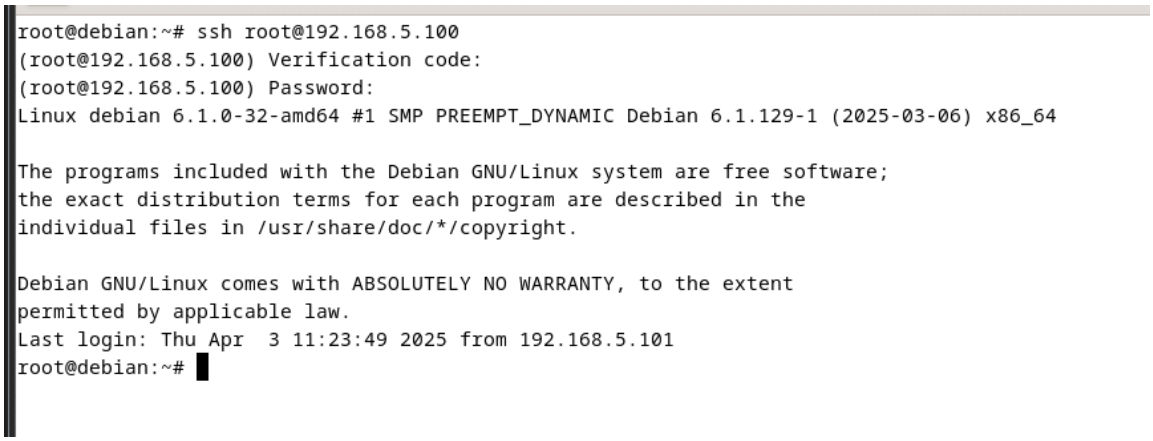
Validez en touchant la Yubikey.

Nous allons tester de se conneter via ssh de la machine test au serveur

Le code d’authentification



Nous allons nous connécter via ssh



Avantages :

Sécurité matérielle : Le secret reste sur la YubiKey (isolé des logiciels malveillants).Indépendance du smartphone : Utilisable sans réseau ou batterie.

Support NFC : Compatible avec des appareils mobiles.

Inconvénients :

Stockage limité (32 identifiants).Coût (nécessite une YubiKey).

Perte de la clé = Perte d'accès (sans backup).

workflow

 # Workflow détaillé :

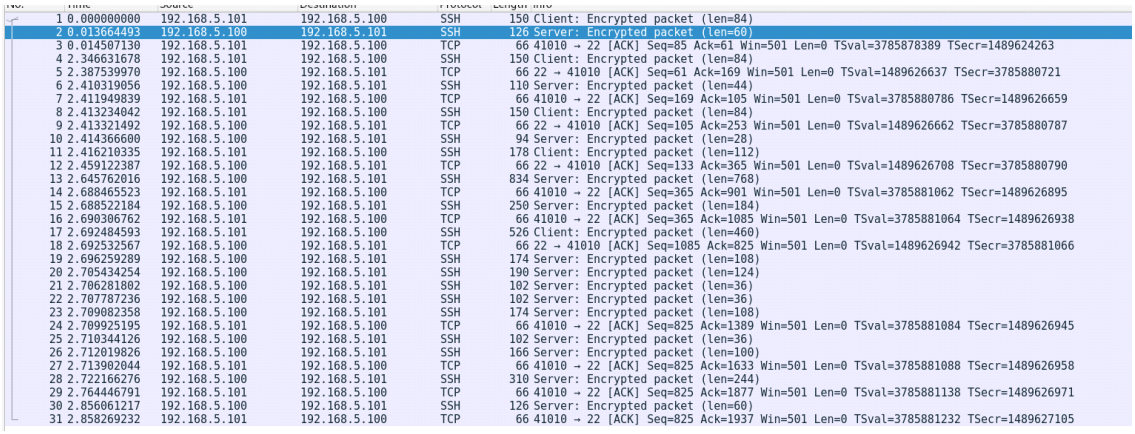
[YubiKey]

Stocke le secret TOTP (via Yubico Authenticator)

Génère un code temporaire

[SSH Server] Reçoit le code TOTP de l'utilisateur

Compae avec son propre calcul TOTP (via PAM)



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.5.101	192.168.5.100	SSH	150	Client: Encrypted packet (len=84)
2	0.013664493	192.168.5.100	192.168.5.101	SSH	126	Server: Encrypted packet (len=60)
3	0.014507130	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=85 Ack=61 Win=501 Len=0 TSval=3785878389 TSecr=1489624263
4	2.346631678	192.168.5.101	192.168.5.100	SSH	150	Client: Encrypted packet (len=84)
5	2.387539970	192.168.5.100	192.168.5.101	TCP	66	22 → 41010 [ACK] Seq=61 Ack=169 Win=501 Len=0 TSval=1489626637 TSecr=3785880721
6	2.410319056	192.168.5.100	192.168.5.101	SSH	110	Server: Encrypted packet (len=44)
7	2.411949839	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=169 Ack=105 Win=501 Len=0 TSval=3785880786 TSecr=1489626659
8	2.413234042	192.168.5.101	192.168.5.100	SSH	150	Client: Encrypted packet (len=84)
9	2.413321492	192.168.5.100	192.168.5.101	TCP	66	22 → 41010 [ACK] Seq=105 Ack=253 Win=501 Len=0 TSval=1489626662 TSecr=3785880787
10	2.414366600	192.168.5.100	192.168.5.101	SSH	94	Server: Encrypted packet (len=28)
11	2.416210335	192.168.5.101	192.168.5.100	SSH	178	Client: Encrypted packet (len=112)
12	2.459122387	192.168.5.100	192.168.5.101	TCP	66	22 → 41010 [ACK] Seq=133 Ack=365 Win=501 Len=0 TSval=1489626708 TSecr=3785880790
13	2.645762016	192.168.5.100	192.168.5.101	SSH	834	Server: Encrypted packet (len=768)
14	2.688465523	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=365 Ack=901 Win=501 Len=0 TSval=3785881062 TSecr=1489626895
15	2.688522184	192.168.5.100	192.168.5.101	SSH	250	Server: Encrypted packet (len=184)
16	2.690306762	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=365 Ack=1085 Win=501 Len=0 TSval=3785881064 TSecr=1489626938
17	2.692484593	192.168.5.101	192.168.5.100	SSH	526	Client: Encrypted packet (len=460)
18	2.692532567	192.168.5.100	192.168.5.101	TCP	66	22 → 41010 [ACK] Seq=1085 Ack=825 Win=501 Len=0 TSval=1489626942 TSecr=3785881066
19	2.696259289	192.168.5.100	192.168.5.101	SSH	174	Server: Encrypted packet (len=108)
20	2.705434254	192.168.5.100	192.168.5.101	SSH	190	Server: Encrypted packet (len=124)
21	2.706281802	192.168.5.100	192.168.5.101	SSH	102	Server: Encrypted packet (len=36)
22	2.707787236	192.168.5.100	192.168.5.101	SSH	102	Server: Encrypted packet (len=36)
23	2.709082358	192.168.5.100	192.168.5.101	SSH	174	Server: Encrypted packet (len=108)
24	2.709925195	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=825 Ack=1389 Win=501 Len=0 TSval=3785881084 TSecr=1489626945
25	2.710344126	192.168.5.100	192.168.5.101	SSH	102	Server: Encrypted packet (len=36)
26	2.712019826	192.168.5.100	192.168.5.101	SSH	166	Server: Encrypted packet (len=100)
27	2.713902044	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=825 Ack=1633 Win=501 Len=0 TSval=3785881088 TSecr=1489626958
28	2.722166276	192.168.5.100	192.168.5.101	SSH	310	Server: Encrypted packet (len=244)
29	2.764446791	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=825 Ack=1877 Win=501 Len=0 TSval=3785881138 TSecr=1489626971
30	2.856061217	192.168.5.100	192.168.5.101	SSH	126	Server: Encrypted packet (len=60)
31	2.858269232	192.168.5.101	192.168.5.100	TCP	66	41010 → 22 [ACK] Seq=825 Ack=1937 Win=501 Len=0 TSval=3785881232 TSecr=1489627105

Connexion TCP (3-way handshake) → (paquets 3, 5, 7)

Échange des clés SSH (clé publique, négociation d'algorithmes) → (paquets 1, 2, 4, 6)

Authentification utilisateur :

Le serveur demande les identifiants.

L'utilisateur envoie son login.

Le serveur déclenche PAM (qui attend un mot de passe ou un TOTP).

L'utilisateur entre un code généré par la YubiKey (TOTP).

Le serveur valide ce code.

Passkey : Authentification FIDO2/U2F avec ProtonPass

avec github nous allons utilisé github pour configurer l'authentification par clée

Créer un compte ProtonMail :

Nous avons Installé ProtonPass : Extension pour Chrome ou Firefox.

Ajoute une Passkey à GitHub :

dans GitHub > Settings > Security > Passkeys.

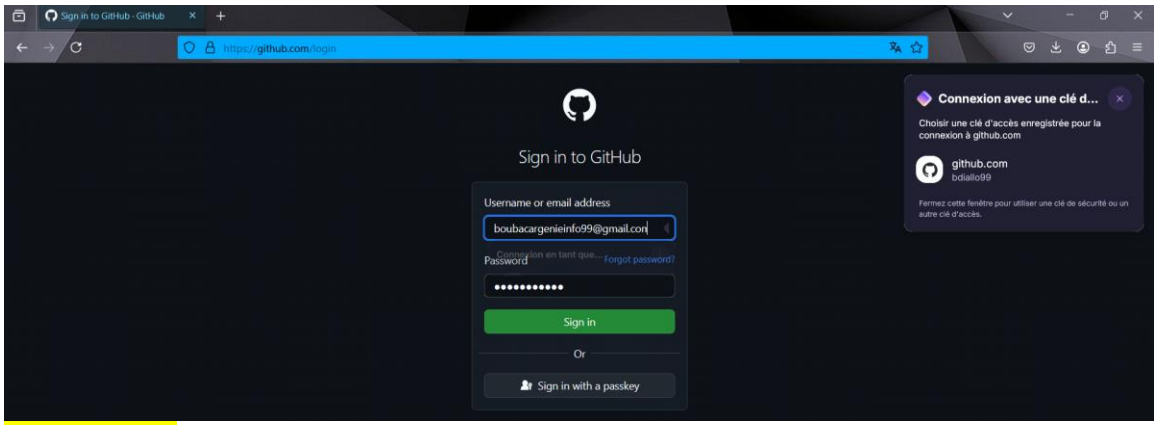
sur Add a Passkey.

ProtonPass détecte la demande et propose de créer une passkey logicielle.

Test :

Déconnectez-vous de GitHub.

je Sélectionne "Sign in with a Passkey" et validez via ProtonPass.



Avantages :

Centralisation : Gestion simplifiée des identifiants.

Génération de mots de passe forts.

Passkeys : Résistance au phishing et suppression des mots de passe.

Inconvénients :

Single Point of Failure : Compromission du compte maître = risque élevé.

Passkeys logicielles : Moins sécurisées qu'une clé matérielle (YubiKey).

Workflow Global (GitHub)

Création de la Passkey :

GitHub demande une passkey → ProtonPass génère une paire de clés (publique/privée).

La clé privée reste chiffrée dans ProtonPass.

La clé publique est envoyée à GitHub.

Authentification :

GitHub envoie un challenge cryptographique.

ProtonPass signe le challenge avec la clé privée.

GitHub vérifie la signature avec la clé publique.

Authentification FIDO2 avec YubiKey

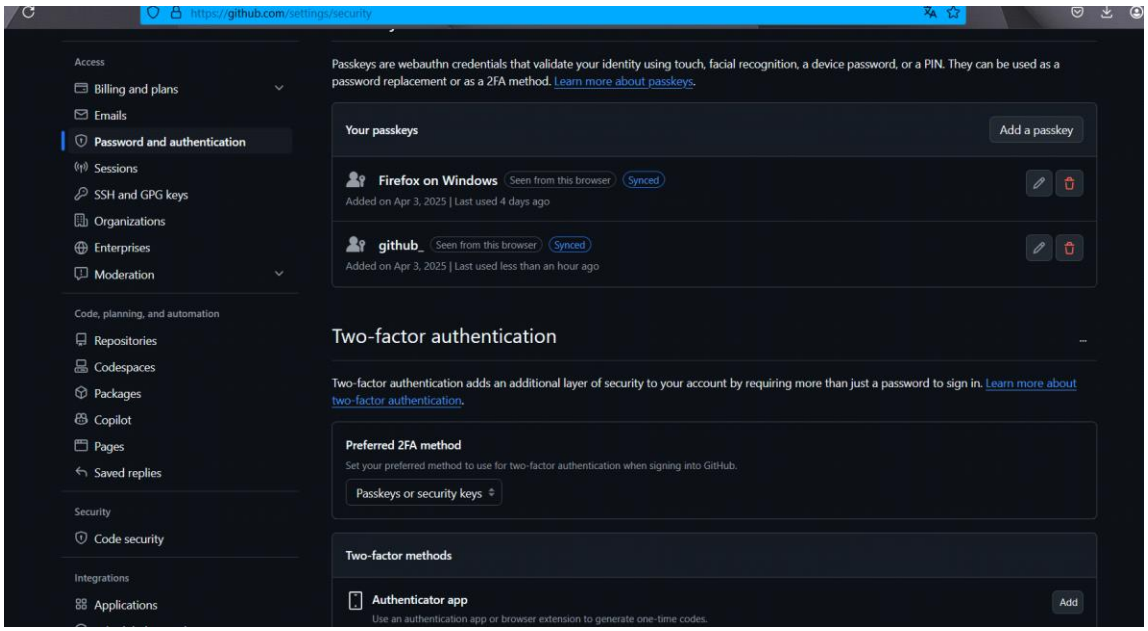
avec github dans paramètre > Security > 2-Step Verification.

j'ai Sélectionné Security Key > Add Key.

j'ai Branché la YubiKey la touche une fois qu'elle demande .

Test :

en choisissant la YubiKey comme méthode.



Workflow Global (avec GitHub)

Enregistrement :

Google génère un challenge → YubiKey crée une paire de clés.

La clé publique est stockée par Google, la clé privée reste sur la YubiKey.

Connexion :

Google envoie un challenge → YubiKey signe avec la clé privée.

Google valide la signature avec la clé publique.

Authentification U2F sous Linux

nous allons installez libpam-u2f

Générez le Fichier de Configuration :

mkdir -p ~/.config/Yubico

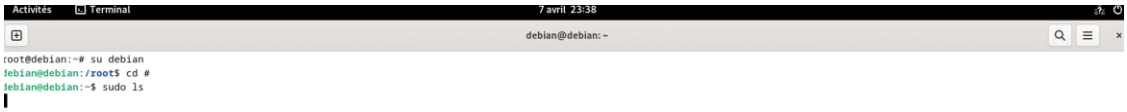
pamu2fcfg -u $USER > ~/.config/Yubico/u2f_keys

Nous allons Branchez la YubiKey et touchez-la quand demandé.

nano /etc/pam.d/sudo

au debut

auth sufficient pam_u2f.so authfile=/home/debian/.config/Yubico/u2f_keys



je valide avec la clée

Workflow U2F avec Linux

Génération d'une Clé par Machine :

La YubiKey génère une paire de clés unique (publique/privée) par machine lors de l'enregistrement.

La clé privée reste sur la YubiKey, la clé publique est stockée dans un fichier (~/.config/Yubico/u2f_keys).

Authentification :

Lors d'un sudo, le système envoie un challenge à la YubiKey.

La YubiKey signe le challenge avec la clé privée correspondant à la machine.

Le système vérifie la signature avec la clé publique enregistrée.

Nombre de Machines/Utilisateurs Supportés

Illimité en théorie :

La YubiKey ne stocke pas les clés privées pour chaque machine. Elle dérive une clé unique par machine à partir d'une clé maître interne.

Authentification FIDO2 et SSH

Nous allons utilisé les deux machines testyubikey et la machine yubikey

dans la machine cliente(TestYubikey) nous allons génerér la paire de clés SSH :

ssh-keygen -t ed25519-sk -O resident -O application=ssh:MyKey

la yubi demande une confirmation par une touch

Configuration de l'authentification U2F:

Installation des paquets nécessaires

On installe les logiciels requis pour gérer l'authentification U2F.

sudo apt-get install libu2f-udev

sudo apt-get install pamu2fcfg

libu2f-udev : Ce paquet contient des règles udev nécessaires pour la gestion des périphériques U2F.

pamu2fcfg : Cet utilitaire nous permet de générer la configuration U2F à enregistrer sur notre système PAM.

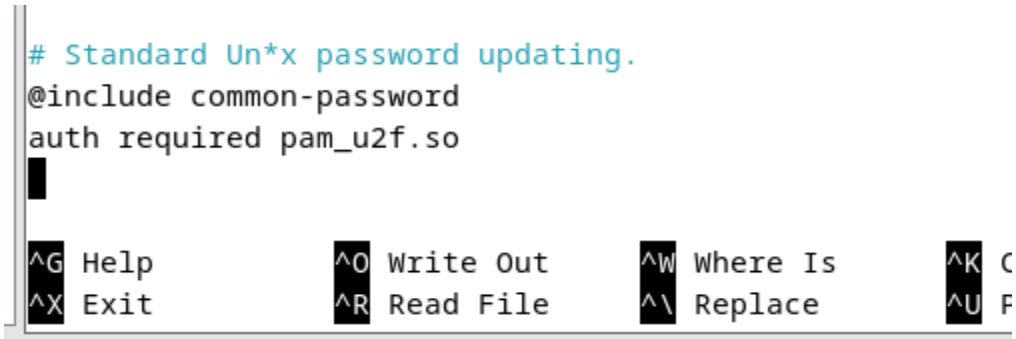
```
lapeng@debian:~$ sudo apt-get install libu2f-udev
[sudo] password for lapeng:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  libu2f-udev
0 upgraded, 1 newly installed, 0 to remove and 80 not upgraded.
Need to get 6,300 B of archives.
After this operation, 14.3 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 libu2f-udev all 1.1.10-3 [6,300 B]
Fetched 6,300 B in 0s (17.0 kB/s)
Selecting previously unselected package libu2f-udev.
(Reading database ... 155848 files and directories currently installed.)
Preparing to unpack .../libu2f-udev_1.1.10-3_all.deb ...
Unpacking libu2f-udev (1.1.10-3) ...
Setting up libu2f-udev (1.1.10-3) ...
lapeng@debian:~$ sudo apt-get install pamu2fcfg
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  pamu2fcfg
0 upgraded, 1 newly installed, 0 to remove and 80 not upgraded.
Need to get 21.4 kB of archives.
After this operation, 66.6 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 pamu2fcfg amd64 1.1.0-1.1+deb12u1 [21.4 kB]
Fetched 21.4 kB in 0s (77.0 kB/s)
Selecting previously unselected package pamu2fcfg.
(Reading database ... 155852 files and directories currently installed.)
Preparing to unpack .../pamu2fcfg_1.1.0-1.1+deb12u1_amd64.deb ...
Unpacking pamu2fcfg (1.1.0-1.1+deb12u1) ...
Setting up pamu2fcfg (1.1.0-1.1+deb12u1) ...
Processing triggers for man-db (2.11.2-2)
```

Configuration du module PAM

Pour intégrer l'authentification U2F dans le processus d'authentification SSH, on modifie la configuration PAM (Pluggable Authentication Modules) du service SSH.

On modifie dans le fichier /etc/pam.d/sshd :

auth required pam_u2f.so(Cette instruction force l'utilisation du module **pam_u2f** lors de l'authentification)



Enregistrement de la YubiKey

Pour que notre YubiKey puisse être utilisée dans le processus d'authentification U2F, on enregistre la clé publique de la YubiKey sur notre machine :

pamu2fcfg > ~/.config/Yubico/u2f_keys

Modification de la configuration SSH

Pour activer le mode challenge–response qui permet l'usage des modules PAM dans SSH, on modifie le fichier /etc/ssh/sshd_config :

ChallengeResponseAuthentication yes

On redemare le service ssh :

sudo systemctl restart sshd

Limite du nombre d'enregistrements U2F avec une YubiKey

Les YubiKeys ont en général une capacité limitée pour stocker des identifiants U2F.

- La plupart des modèles de YubiKey peuvent enregistrer environ **32 identifiants**.
- Cela signifie que vous pouvez configurer jusqu'à 32 appareils ou services avec la même YubiKey pour l'authentification U2F.
- Cette limitation est inhérente au protocole U2F et à la capacité matérielle de la clé.

Cette configuration permet d'utiliser une YubiKey pour renforcer l'authentification SSH sur une machine Linux via le protocole U2F. Les étapes consistent à installer les paquets nécessaires, configurer le module PAM pour SSH, enregistrer la YubiKey pour qu'elle puisse fournir ses informations d'authentification et enfin à adapter la configuration SSH pour accepter les réponses aux challenges. Gardez en mémoire la limite d’enregistrements (environ 32) imposée par le matériel.

Authentification FIDO2 et ssh

Deux types de clés :

ecdsa-sk : Cette clé, basée sur l'algorithme des courbes elliptiques (ECDSA), est spécialement conçue pour les modules de sécurité matériels (HSM). Elle est optimisée pour être utilisée dans des environnements où la sécurité matérielle est essentielle.

ed25519-sk : Cette clé s'appuie sur l'algorithme ed25519. Elle offre un bon compromis entre rapidité et sécurité, ce qui permet d'obtenir des performances élevées tout en garantissant une forte protection cryptographique.

Point commun aux deux : Les deux types de clés sont supportés de manière native par SSH pour la gestion des clés de sécurité. La présence du suffixe « -sk » signifie « Security Key », indiquant qu'il s'agit de clés conçues pour être utilisées avec des dispositifs de sécurité matériels (comme une YubiKey).

Différences entre le Resident mode et le Non-resident (mode partagé)

Resident Mode (mode résident) : Dans ce mode, l'information de la clé privée est stockée directement à l'intérieur de la YubiKey elle-même.

Avantage :

-L'utilisateur peut se connecter à partir de différents appareils en utilisant la même YubiKey, sans avoir à copier la clé privée sur chacun de ces dispositifs.

-Cela offre une grande praticité et une sécurité accrue, car la clé privée reste confinée dans le support matériel et n'est pas exposée en dehors.

Inconvénient :

Ce mode nécessite que la YubiKey supporte la fonctionnalité de stockage résidentiel (c'est-à-dire qu'elle puisse contenir la clé privée).

Non-resident Mode (mode non résident ou mode partagé) : Dans ce mode, la clé privée est générée et stockée sur l'ordinateur local, tandis que la YubiKey est utilisée uniquement pour effectuer la signature lors de l'authentification.

Avantage :

-La YubiKey joue un rôle de signature, ce qui permet d'utiliser du matériel pour renforcer l'authentification.

Inconvénient :

-L'utilisateur doit copier la clé privée sur chaque machine sur laquelle il souhaite se connecter, ce qui peut augmenter le risque de compromission, puisque la clé privée se retrouve sur plusieurs dispositifs.

Donc :

En mode résident, la clé privée reste dans le dispositif (YubiKey), ce qui simplifie la gestion et renforce la sécurité.

En mode non résident, la clé privée est dispersée sur les différents appareils, ce qui peut compliquer la sécurisation de ses copies et accroître les risques en cas de compromission d'un de ces appareils.

Preparation:

Install openssh server qui soutiens le hardware key :

sudo apt install openssh-server

sshd –version

```
lapeng@debian:~$ sudo apt install openssh-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssh-server is already the newest version (1:9.2p1-2+deb12u5).
0 upgraded, 0 newly installed, 0 to remove and 80 not upgraded.
lapeng@debian:~$ sshd --version
```

```
lapeng@debian:~$ /usr/sbin/sshd --version
unknown option -- -
OpenSSH_9.2p1 Debian-2+deb12u5, OpenSSL 3.0.15 3 Sep 2024
usage: sshd [-46DdeiqTtV] [-C connection_spec] [-c host_cert_file]
            [-E log_file] [-f config_file] [-g login_grace_time]
            [-h host_key_file] [-o option] [-p port] [-u len]
```

sudo apt-add-repository ppa:yubico/stable

Cette commande est une étape préparatoire permettant à votre système d'accéder au dernier logiciel stable de Yubico, ce qui est essentiel pour configurer et gérer la prise en charge des clés matérielles avec OpenSSH.

sudo apt install yubikey-manager

Configuration(On choit ed25519-sk) :

```
lapeng@debian:~$ ssh-keygen -t ed25519-sk -f ~/.ssh/id_ed25519_sk
Generating public/private ed25519-sk key pair.
You may need to touch your authenticator to authorize key generation.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match.  Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/lapeng/.ssh/id_ed25519_sk
Your public key has been saved in /home/lapeng/.ssh/id_ed25519_sk.pub
The key fingerprint is:
SHA256:rHAtIxW6c75pQOlzx5X6h5P1vz55lh739DtyUCJavjA lapeng@debian
The key's randomart image is:
+[ED25519-SK 256]-+
|       .         |
|      . .        |
|     ...    .    |
|     oo o  oo . .|
|    o= =.So+ . o |
|     +B.++E ...  |
|      +o. .o+....=|
|       .o  =....BB|
|      .o    o  =BX|
+----[SHA256]-----+
```

On se connecter au serveur (**machine SSH**) et ajouter automatiquement le contenu de votre clé publique au fichier ~/.ssh/authorized_keys de l'utilisateur spécifié :

ssh-copy-id -i ~/.ssh/id_ed25519_sk.pub lapeng@192.168.36.159

```
lapeng@debian:~$ ssh-copy-id -i ~/.ssh/id_ed25519_sk.pub lapeng@192.168.36.159
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/lapeng/.ssh/id_ed25519_sk.pub"
The authenticity of host '192.168.36.159 (192.168.36.159)' can't be established.
ED25519 key fingerprint is SHA256:sSbY48TVgCmzoJQdGD8rdnROOJ9031EhKDCzG3FREh0.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:4: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
lapeng@192.168.36.159's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'lapeng@192.168.36.159'"
and check to make sure that only the key(s) you wanted were added.
```

Sur la Machine YubiKey :

On affiche le contenu de la clé publique avec la commande :

```
lapeng@debian:~$ cat ~/.ssh/id_ed25519_sk.pub
sk-ssh-ed25519@openssh.com AAAAGnNrLXNzaC1lZDI1NTE5QG9wZW5zc2guY29tAAAAIE/B9MYNbkwD2ZTD0EwqXGh7M3rg6nZyX+H2pT7JdfotAAAABHNzaDo= lapeng@debian
```

Sur la Machine SSH (serveur distant) :

On connect au serveur et ouvrez le fichier ~/.ssh/authorized_keys avec un éditeur de texte :

nano ~/.ssh/authorized_keys

Test de la connexion SSH :

depuis la **machine YubiKey (machine locale)**, on teste la connexion vers la **machine SSH** en utilisant la clé privée générée :

ssh -i ~/.ssh/id_ed25519_sk lapeng@192.168.36.159

```
lapeng@debian:~$ cat ~/.ssh/id_ed25519_sk.pub
sk-ssh-ed25519@openssh.com AAAAGnNrLXNzaC1lZDI1NTE5QG9wZW5zc2guY29tAAAAIE/B9MYNbkwD2ZTD0EwqXGh7M3rg6nZyX+H2pT7JdfotAAAABHNzaDo= lapeng@debian
lapeng@debian:~$ ssh -i ~/.ssh/id_ed25519_sk lapeng@192.168.36.159
Linux debian 6.1.0-30-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.124-1 (2025-01-12) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Apr  8 14:48:58 2025 from 192.168.36.145
```