1. YubiKey-Guide

This is a guide to using YubiKey as a smart card for secure encryption, signature and authentication operations.

Keys stored on YubiKey are non-exportable, unlike filesystem-based credentials, while remaining convenient for daily use. YubiKey can be configured to require a physical touch for cryptographic operations, reducing the risk of credential compromise.

To suggest an improvement, send a pull request or open an issue.

Contents

2. Purchase YubiKey

Current YubiKeys except the FIDO-only Security Key Series and Bio Series YubiKeys are compatible with this guide.

Verify YubiKey by visiting yubico.com/genuine. Select *Verify Device* to begin the process. Touch the YubiKey when prompted and allow the site to see the make and model of the device when prompted. This device attestation may help mitigate supply chain attacks.

Several portable storage devices (such as microSD cards) for storing encrypted backups are also recommended.

3. Prepare environment

A dedicated, secure operating environment is recommended to generate cryptographic keys.

The following is a general ranking of environments least to most hospitable to generating materials:

1. Public, shared or other computer owned by someone else
2. Daily-use personal operating system with unrestricted network access
3. Virtualized operating system with limited capabilities (using virt-manager, VirtualBox or VMware, for example)
4. Dedicated and hardened Debian or OpenBSD installation
5. Ephemeral Debian Live or Tails booted without primary storage attached
6. Hardened hardware and firmware (Coreboot, Intel ME removed)
7. Air-gapped system without network capabilities, preferably ARM-based Raspberry Pi or other architecturally diverse equivalent

Debian Live is used in this guide to balance usability and security, with some additional instructions for OpenBSD.

Download the latest image and signature files:

```
curl -fLO "https://cdimage.debian.org/debian-cd/current-live/amd64/iso-hybrid/SHA512SUMS"

curl -fLO "https://cdimage.debian.org/debian-cd/current-live/amd64/iso-hybrid/SHA512SUMS.sign"

curl -fLO "https://cdimage.debian.org/debian-cd/current-live/amd64/iso-hybrid/$(awk '/xfce.iso$/ {print $2}' SHA512SUMS)"
```

Download the Debian signing public key:

```
gpg --keyserver hkps://keyring.debian.org --recv DF9B9C49EAA9298432589D76DA87E80D6294BE9B
```

If the public key cannot be received, use a different keyserver or DNS server:

```
gpg --keyserver hkps://keyserver.ubuntu.com:443 --recv DF9B9C49EAA9298432589D76DA87E80D6294BE9B
```

Verify the signature:

```
gpg --verify SHA512SUMS.sign SHA512SUMS
```

`gpg: Good signature from "Debian CD signing key <debian-cd@lists.debian.org>"` must appear in the output.

Verify the cryptographic hash of the image file matches the one in the signed file:

```
grep $(sha512sum debian-live-*-amd64-xfce.iso) SHA512SUMS
```

See Verifying authenticity of Debian CDs for more information.

Connect a portable storage device and identify the disk label - this guide uses `/dev/sdc` throughout, but this value may differ on your system:

Linux

```
$ sudo dmesg | tail
usb-storage 3-2:1.0: USB Mass Storage device detected
sd 2:0:0:0: [sdc] Attached SCSI removable disk
```

Copy the Debian image to the device:

```
sudo dd if=debian-live-*-amd64-xfce.iso of=/dev/sdc bs=4M status=progress ; sync
```

OpenBSD

```
$ dmesg | tail -n2
sd2 at scsibus4 targ 1 lun 0: <TS-RDF5, SD Transcend, TS3A> SCSI4 0/direct removable serial.0000000000000
sd2: 15193MB, 512 bytes/sector, 31116288 sectors

$ doas dd if=debian-live-*-amd64-xfce.iso of=/dev/rsd2c bs=4m
465+1 records in
465+1 records out
1951432704 bytes transferred in 139.125 secs (14026448 bytes/sec)
```

Power off, remove internal hard drives and all unnecessary devices, such as the wireless card.

4. Install software

Load the operating system and configure networking. Optional hardening steps related to networking can be found below.

> **Note**
>
> If the screen locks on Debian Live, unlock with `user` / `live`

Open terminal and install required software packages.

Debian/Ubuntu

```
sudo apt update

sudo apt -y upgrade

sudo apt -y install \
  wget gnupg2 gnupg-agent dirmngr \
  cryptsetup scdaemon pcscd \
  yubikey-personalization yubikey-manager
```

OpenBSD

```
doas pkg_add gnupg pcsc-tools
```

macOS

Download and install Homebrew and the following packages:

```
brew install \
  gnupg yubikey-personalization ykman pinentry-mac wget
```

> **Note**
>
> An additional Python package dependency may need to be installed to use `ykman` - `pip install yubikey-manager`

NixOS

Build an air-gapped NixOS LiveCD image:

```
ref=$(git ls-remote https://github.com/drduh/Yubikey-Guide refs/heads/master | awk '{print $1}')

nix build --experimental-features "nix-command flakes" \
  github:drduh/YubiKey-Guide/$ref#nixosConfigurations.yubikeyLive.x86_64-linux.config.system.build.isoImage
```

If you have this repository checked out:

Recommended, but optional: update `nixpkgs` and `drduh/config`:

```
nix flake update --commit-lock-file
```

Build the image:

```
nix build --experimental-features "nix-command flakes" .#nixosConfigurations.yubikeyLive.x86_64-
linux.config.system.build.isoImage
```

Copy it to a USB drive:

```
sudo cp -v result/iso/yubikeyLive.iso /dev/sdc ; sync
```

Skip steps to create a temporary working directory and a hardened configuration, as they are already part of the image.

If you want to test your build before copying it into a USB stick, you can try it out on your machine using a tool like QEMU. Please keep in mind that a virtualized environment does not provide the same amount of security as an ephemeral system (see **Prepare environment** above). Here is an example QEMU invocation after placing `yubikeyLive` in `result/iso` using the above `nix build` command:

```
# Launch with 4G memory, 2 CPUs and KVM enabled
qemu-system-x86_64 \
    -enable-kvm \
    -m 4G \
    -smp 2 \
    -drive readonly=on,media=cdrom,format=raw,file=result/iso/yubikeyLive.iso
```

Arch

```
sudo pacman -Syu gnupg pcsclite ccid yubikey-personalization
```

RHEL7

```
sudo yum install -y gnupg2 pinentry-curses pcsc-lite pcsc-lite-libs gnupg2-smime
```

Fedora

```
sudo dnf install wget

wget https://github.com/rpmsphere/noarch/raw/master/r/rpmsphere-release-38-1.noarch.rpm

sudo rpm -Uvh rpmsphere-release*rpm

sudo dnf install \
  gnupg2 dirmngr cryptsetup gnupg2-smime \
  pcsc-tools opensc pcsc-lite secure-delete \
  pgp-tools yubikey-personalization-gui
```

5. Prepare GnuPG

Create a temporary directory which will be cleared on reboot and set it as the GnuPG directory:

```
export GNUPGHOME=$(mktemp -d -t gnupg-$(date +%Y-%m-%d)-XXXXXXXXXX)
```

5.1. Configuration

Import or create a hardened configuration:

```
cd $GNUPGHOME

wget https://raw.githubusercontent.com/drduh/config/master/gpg.conf
```

The options will look similar to:

```
$ grep -ve "^#" $GNUPGHOME/gpg.conf
personal-cipher-preferences AES256 AES192 AES
personal-digest-preferences SHA512 SHA384 SHA256
personal-compress-preferences ZLIB BZIP2 ZIP Uncompressed
default-preference-list SHA512 SHA384 SHA256 AES256 AES192 AES ZLIB BZIP2 ZIP Uncompressed
cert-digest-algo SHA512
s2k-digest-algo SHA512
s2k-cipher-algo AES256
charset utf-8
no-comments
no-emit-version
no-greeting
keyid-format 0xlong
list-options show-uid-validity
verify-options show-uid-validity
with-fingerprint
require-cross-certification
no-symkey-cache
armor
use-agent
throw-keyids
```

> **Note**
>
> Networking can be disabled for the remainder of the setup.

5.2. Identity

When creating an identity with GnuPG, the default options ask for a “Real name”, “Email address” and optional “Comment”.

Depending on how you plan to use GnuPG, set these values respectively:

```
export IDENTITY="YubiKey User <yubikey@example>"
```

Or use any attribute which will uniquely identity the key (this may be incompatible with certain use cases):

```
export IDENTITY="My Cool YubiKey - 2024"
```

5.3. Key

Select the desired algorithm and key size. This guide recommends 4096-bit RSA.

Set the value:

```
export KEY_TYPE=rsa4096
```

5.4. Expiration

Determine the desired Subkey validity duration.

Setting a Subkey expiry forces identity and credential lifecycle management. However, setting an expiry on the Certify key is pointless, because it can just be used to extend itself. Revocation certificates should instead be used to revoke an identity.

This guide recommends a two year expiration for Subkeys to balance security and usability, however longer durations are possible to reduce maintenance frequency.

When Subkeys expire, they may still be used to decrypt with GnuPG and authenticate with SSH, however they can **not** be used to encrypt nor sign new messages.

Subkeys must be renewed or rotated using the Certify key - see Renewing Subkeys.

Set the expiration date to two years:

```
export EXPIRATION=2y
```

Or set the expiration date to a specific date to schedule maintenance:

```
export EXPIRATION=2026-05-01
```

5.5. Passphrase

Generate a passphrase for the Certify key. It will be used infrequently to manage Subkeys and should be very strong. The passphrase is recommended to consist of only uppercase letters and numbers for improved readability. Diceware is another method for creating memorable passphrases.

The following commands will generate a strong passphrase and avoid ambiguous characters:

```
export CERTIFY_PASS=$(LC_ALL=C tr -dc 'A-Z1-9' < /dev/urandom | \
  tr -d "1IOS5U" | fold -w 30 | sed "-es/./ /"{1..26..5} | \
  cut -c2- | tr " " "-" | head -1) ; echo "\n$CERTIFY_PASS\n"
```

Write the passphrase in a secure location, ideally separate from the portable storage device used for key material, or memorize it.

This repository includes a `passphrase.html` template to help with credential transcription. Save the raw file, open it with a browser and print. Use a pen or permanent marker to select a letter or number on each row for each character in the passphrase. `passphrase.csv` can also be printed without a browser:

```
lp -d Printer-Name passphrase.csv
```

6. Create Certify key

The primary key to generate is the Certify key, which is responsible for issuing Subkeys for encryption, signature and authentication operations.

The Certify key should be kept offline at all times and only accessed from a dedicated and secure environment to issue or revoke Subkeys.

Do not set an expiration date on the Certify key.

Generate the Certify key:

```
gpg --batch --passphrase "$CERTIFY_PASS" \
    --quick-generate-key "$IDENTITY" "$KEY_TYPE" cert never
```

Set and view the Certify key identifier and fingerprint for use later:

```
export KEYID=$(gpg -k --with-colons "$IDENTITY" | awk -F: '/^pub:/ { print $5; exit }')

export KEYFP=$(gpg -k --with-colons "$IDENTITY" | awk -F: '/^fpr:/ { print $10; exit }')

printf "\nKey ID: %40s\nKey FP: %40s\n\n" "$KEYID" "$KEYFP"
```

7. Create Subkeys

Use the following command to generate Signature, Encryption and Authentication Subkeys using the previously configured key type, passphrase and expiration:

```
for SUBKEY in sign encrypt auth ; do \
  gpg --batch --pinentry-mode=loopback --passphrase "$CERTIFY_PASS" \
      --quick-add-key "$KEYFP" "$KEY_TYPE" "$SUBKEY" "$EXPIRATION"
done
```

8. Verify keys

List available secret keys:

```
gpg -K
```

The output will display **[C]ertify, [S]ignature, [E]ncryption and [A]uthentication** keys:

```
sec   rsa4096/0xF0F2CFEB04341FB5 2024-01-01 [C]
      Key fingerprint = 4E2C 1FA3 372C BA96 A06A  C34A F0F2 CFEB 0434 1FB5
uid                   [ultimate] YubiKey User <yubikey@example>
ssb   rsa4096/0xB3CD10E502E19637 2024-01-01 [S] [expires: 2026-05-01]
ssb   rsa4096/0x30CBE8C4B085B9F7 2024-01-01 [E] [expires: 2026-05-01]
ssb   rsa4096/0xAD9E24E1B8CB9600 2024-01-01 [A] [expires: 2026-05-01]
```

9. Backup keys

Save a copy of the Certify key, Subkeys and public key:

```
gpg --output $GNUPGHOME/$KEYID-Certify.key \
    --batch --pinentry-mode=loopback --passphrase "$CERTIFY_PASS" \
    --armor --export-secret-keys $KEYID

gpg --output $GNUPGHOME/$KEYID-Subkeys.key \
    --batch --pinentry-mode=loopback --passphrase "$CERTIFY_PASS" \
    --armor --export-secret-subkeys $KEYID

gpg --output $GNUPGHOME/$KEYID-$(date +%F).asc \
    --armor --export $KEYID
```

Create an **encrypted** backup on portable storage to be kept offline in a secure and durable location.

The following process is recommended to be repeated several times on multiple portable storage devices, as they are likely to fail over time. As an additional backup measure, Paperkey can be used to make a physical copy of key materials for improved durability.

Tip The ext2 filesystem without encryption can be mounted on Linux and OpenBSD. Use FAT32 or NTFS filesystem for macOS and Windows compatibility instead.

Linux

Attach a portable storage device and check its label, in this case `/dev/sdc`:

```
$ sudo dmesg | tail
usb-storage 3-2:1.0: USB Mass Storage device detected
sd 2:0:0:0: [sdc] Attached SCSI removable disk

$ sudo fdisk -l /dev/sdc
Disk /dev/sdc: 14.9 GiB, 15931539456 bytes, 31116288 sectors
```

> ⚠ **Warning**
>
> Confirm the destination (`of`) before issuing the following command - it is destructive! This guide uses `/dev/sdc` throughout, but this value may be different on your system.

Zero the header to prepare for encryption:

```
sudo dd if=/dev/zero of=/dev/sdc bs=4M count=1
```

Remove and re-connect the storage device.

Erase and create a new partition table:

```
sudo fdisk /dev/sdc <<EOF
g
w
EOF
```

Create a small (at least 20 Mb is recommended to account for the LUKS header size) partition for storing secret materials:

```
sudo fdisk /dev/sdc <<EOF
n


+20M
w
EOF
```

Use LUKS to encrypt the new partition.

Generate another unique Passphrase (ideally different from the one used for the Certify key) to protect the encrypted volume:

```
export LUKS_PASS=$(LC_ALL=C tr -dc 'A-Z1-9' < /dev/urandom | \
  tr -d "1IOS5U" | fold -w 30 | sed "-es/./ /"{1..26..5} | \
  cut -c2- | tr " " "-" | head -1) ; echo "\n$LUKS_PASS\n"
```

This passphrase will also be used infrequently to access the Certify key and should be very strong.

Write the passphrase down or memorize it.

Format the partition:

```
echo $LUKS_PASS | sudo cryptsetup -q luksFormat /dev/sdc1
```

Mount the partition:

```
echo $LUKS_PASS | sudo cryptsetup -q luksOpen /dev/sdc1 gnupg-secrets
```

Create an ext2 filesystem:

```
sudo mkfs.ext2 /dev/mapper/gnupg-secrets -L gnupg-$(date +%F)
```

Mount the filesystem and copy the temporary GnuPG working directory with key materials:

```
sudo mkdir /mnt/encrypted-storage

sudo mount /dev/mapper/gnupg-secrets /mnt/encrypted-storage

sudo cp -av $GNUPGHOME /mnt/encrypted-storage/
```

Unmount and close the encrypted volume:

```
sudo umount /mnt/encrypted-storage

sudo cryptsetup luksClose gnupg-secrets
```

Repeat the process for any additional storage devices (at least two are recommended).

OpenBSD

Attach a USB disk and determine its label:

```
$ dmesg | grep sd.\ at
sd2 at scsibus5 targ 1 lun 0: <TS-RDF5, SD Transcend, TS37> SCSI4 0/direct removable serial.0000000000000000000
```

Print the existing partitions to make sure it's the right device:

```
doas disklabel -h sd2
```

Initialize the disk by creating an `a` partition with FS type `RAID` and size of 25 Megabytes:

```
$ doas fdisk -giy sd2
Writing MBR at offset 0.
Writing GPT.

$ doas disklabel -E sd2
Label editor (enter '?' for help at any prompt)
sd2> a a
offset: [64]
size: [31101776] 25M
FS type: [4.2BSD] RAID
sd2*> w
sd2> q
No label changes
```

Encrypt with bioctl using a unique Passphrase:

```
$ doas bioctl -c C -l sd2a softraid0
New passphrase:
Re-type passphrase:
softraid0: CRYPTO volume attached as sd3
```

Create an `i` partition on the new crypto volume and the filesystem:

```
$ doas fdisk -giy sd3
Writing MBR at offset 0.
Writing GPT.

$ doas disklabel -E sd3
Label editor (enter '?' for help at any prompt)
sd3> a i
offset: [64]
size: [16001]
FS type: [4.2BSD]
sd3*> w
sd3> q
No label changes.

$ doas newfs sd3i
```

Mount the filesystem and copy the temporary directory with the keyring:

```
doas mkdir /mnt/encrypted-storage

doas mount /dev/sd3i /mnt/encrypted-storage

doas cp -av $GNUPGHOME /mnt/encrypted-storage
```

Unmount and remove the encrypted volume:

```
doas umount /mnt/encrypted-storage

doas bioctl -d sd3
```

See OpenBSD FAQ14 for more information.

10. Export public key

Important Without the public key, it will **not** be possible to use GnuPG to decrypt nor sign messages. However, YubiKey can still be used for SSH authentication.

Connect another portable storage device or create a new partition on the existing one.

Linux

Using the same `/dev/sdc` device as in the previous step, create a small (at least 20 Mb is recommended) partition for storing materials:

```
sudo fdisk /dev/sdc <<EOF
n


+20M
w
EOF
```

Create a filesystem and export the public key:

```
sudo mkfs.ext2 /dev/sdc2

sudo mkdir /mnt/public

sudo mount /dev/sdc2 /mnt/public

gpg --armor --export $KEYID | sudo tee /mnt/public/$KEYID-$(date +%F).asc

sudo chmod 0444 /mnt/public/*.asc
```

Unmount and remove the storage device:

```
sudo umount /mnt/public
```

OpenBSD

```
$ doas disklabel -E sd2
Label editor (enter '?' for help at any prompt)
sd2> a b
offset: [32130]
size: [31069710] 25M
FS type: [swap] 4.2BSD
sd2*> w
sd2> q
No label changes.
```

Create a filesystem and export the public key to it:

```
doas newfs sd2b

doas mkdir /mnt/public

doas mount /dev/sd2b /mnt/public

gpg --armor --export $KEYID | doas tee /mnt/public/$KEYID-$(date +%F).asc
```

Unmount and remove the storage device:

```
doas umount /mnt/public
```

11. Configure YubiKey

Connect YubiKey and confirm its status:

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

If the card is locked, Reset it.

11.1. Change PIN

YubiKey's PGP interface has its own PINs separate from other modules such as PIV:

Name | Default value | Capability ———|———|————————————————- User PIN | `123456` | cryptographic operations (decrypt, sign, authenticate) Admin PIN | `12345678` | reset PIN, change Reset Code, add keys and owner information Reset Code | None | reset PIN (more information)

Determine the desired PIN values. They can be shorter than the Certify key passphrase due to limited brute-forcing opportunities; the User PIN should be convenient enough to remember for every-day use.

The **User PIN** must be at least 6 characters and the **Admin PIN** must be at least 8 characters. A maximum of 127 ASCII characters are allowed. See GnuPG - Managing PINs for more information.

Set PINs manually or generate them, for example a 6 digit User PIN and 8 digit Admin PIN:

```
export ADMIN_PIN=$(LC_ALL=C tr -dc '0-9' < /dev/urandom | fold -w8 | head -1)

export USER_PIN=$(LC_ALL=C tr -dc '0-9' < /dev/urandom | fold -w6 | head -1)

printf "\nAdmin PIN: %12s\nUser PIN: %13s\n\n" "$ADMIN_PIN" "$USER_PIN"
```

Change the Admin PIN:

```
gpg --command-fd=0 --pinentry-mode=loopback --change-pin <<EOF
3
12345678
$ADMIN_PIN
$ADMIN_PIN
q
EOF
```

Change the User PIN:

```
gpg --command-fd=0 --pinentry-mode=loopback --change-pin <<EOF
1
123456
$USER_PIN
$USER_PIN
q
EOF
```

Remove and re-insert YubiKey.

> ⚠ **Warning**
>
> Three incorrect **User PIN** entries will cause it to become blocked and must be unblocked with either the **Admin PIN** or **Reset Code**. Three incorrect **Admin PIN** or **Reset Code** entries will destroy data on YubiKey.

The number of retry attempts can be changed, for example to 5 attempts:

```
ykman openpgp access set-retries 5 5 5 -f -a $ADMIN_PIN
```

11.2. Set attributes

Set the smart card attributes with `gpg --edit-card` and `admin` mode - use `help` to see available options.

Or use predetermined values:

```
gpg --command-fd=0 --pinentry-mode=loopback --edit-card <<EOF
admin
login
$IDENTITY
$ADMIN_PIN
quit
EOF
```

Run `gpg --card-status` to verify results (**Login data** field).

12. Transfer Subkeys

Important Transferring keys to YubiKey is a one-way operation which converts the on-disk key into a stub making it no longer usable to transfer to subsequent YubiKeys. Ensure a backup was made before proceeding.

The Certify key passphrase and Admin PIN are required to transfer keys.

12.1. Signature key

Transfer the first key:

```
gpg --command-fd=0 --pinentry-mode=loopback --edit-key $KEYID <<EOF
key 1
keytocard
1
$CERTIFY_PASS
$ADMIN_PIN
save
EOF
```

12.2. Encryption key

Repeat the process for the second key:

```
gpg --command-fd=0 --pinentry-mode=loopback --edit-key $KEYID <<EOF
key 2
keytocard
2
$CERTIFY_PASS
$ADMIN_PIN
save
EOF
```

12.3. Authentication key

Repeat the process for the third key:

```
gpg --command-fd=0 --pinentry-mode=loopback --edit-key $KEYID <<EOF
key 3
keytocard
3
$CERTIFY_PASS
$ADMIN_PIN
save
EOF
```

13. Verify transfer

Verify Subkeys have been moved to YubiKey with `gpg -K` and look for `ssb>`, for example:

```
sec   rsa4096/0xF0F2CFEB04341FB5 2024-01-01 [C]
      Key fingerprint = 4E2C 1FA3 372C BA96 A06A  C34A F0F2 CFEB 0434 1FB5
uid                   [ultimate] YubiKey User <yubikey@example>
ssb>  rsa4096/0xB3CD10E502E19637 2024-01-01 [S] [expires: 2026-05-01]
ssb>  rsa4096/0x30CBE8C4B085B9F7 2024-01-01 [E] [expires: 2026-05-01]
ssb>  rsa4096/0xAD9E24E1B8CB9600 2024-01-01 [A] [expires: 2026-05-01]
```

The `>` after a tag indicates the key is stored on a smart card.

14. Finish setup

Verify you have done the following:

- [ ] Memorized or wrote down the Certify key (identity) passphrase to a secure and durable location
  - `echo $CERTIFY_PASS` to see it again; `passphrase.html` or `passphrase.csv` to transcribe it
- [ ] Memorized or wrote down passphrase to encrypted volume on portable storage
  - `echo $LUKS_PASS` to see it again; `passphrase.html` or `passphrase.csv` to transcribe it
- [ ] Saved the Certify key and Subkeys to encrypted portable storage, to be kept offline
  - At least two backups are recommended, stored at separate locations
- [ ] Exported a copy of the public key where is can be easily accessed later
  - Separate device or non-encrypted partition was used
- [ ] Memorized or wrote down the User PIN and Admin PIN, which are unique and changed from default values
  - `echo $USER_PIN $ADMIN_PIN` to see them again; `passphrase.html` or `passphrase.csv` to transcribe them
- [ ] Moved Encryption, Signature and Authentication Subkeys to YubiKey
  - `gpg -K` shows `ssb>` for each of the 3 Subkeys

Reboot to clear the ephemeral environment and complete setup.

15. Using YubiKey

Initialize GnuPG:

```
gpg -k
```

Import or create a hardened configuration:

```
cd ~/.gnupg

wget https://raw.githubusercontent.com/drduh/config/master/gpg.conf
```

Set the following option. This avoids the problem where GnuPG will repeatedly prompt for the insertion of an already-inserted YubiKey:

```
touch scdaemon.conf

echo "disable-ccid" >>scdaemon.conf
```

Install the required packages:

Debian/Ubuntu

```
sudo apt update

sudo apt install -y \
  gnupg gnupg-agent gnupg-curl scdaemon pcscd
```

OpenBSD

```
doas pkg_add gnupg pcsc-tools

doas rcctl enable pcscd

doas reboot
```

Mount the non-encrypted volume with the public key:

Debian/Ubuntu

```
sudo mkdir /mnt/public

sudo mount /dev/sdc2 /mnt/public
```

OpenBSD

```
doas mkdir /mnt/public

doas mount /dev/sd3i /mnt/public
```

Import the public key:

```
gpg --import /mnt/public/*.asc
```

Or download the public key from a keyserver:

```
gpg --recv $KEYID
```

Or with the URL on YubiKey, retrieve the public key:

```
gpg/card> fetch

gpg/card> quit
```

Determine the key ID:

```
gpg -k

KEYID=0xF0F2CFEB04341FB5
```

Assign ultimate trust by typing `trust` and selecting option `5` then `quit`:

```
gpg --command-fd=0 --pinentry-mode=loopback --edit-key $KEYID <<EOF
trust
5
y
save
EOF
```

Remove and re-insert YubiKey.

Verify the status with `gpg --card-status` which will list the available Subkeys:

```
Reader ...........: Yubico YubiKey OTP FIDO CCID 00 00
Application ID ...: D2760001240102010006055532110000
Application type .: OpenPGP
Version ..........: 3.4
Manufacturer .....: Yubico
Serial number ....: 05553211
Name of cardholder: YubiKey User
Language prefs ...: en
Salutation .......:
URL of public key : [not set]
Login data .......: yubikey@example
Signature PIN ....: not forced
Key attributes ...: rsa4096 rsa4096 rsa4096
Max. PIN lengths .: 127 127 127
PIN retry counter : 3 3 3
Signature counter : 0
KDF setting ......: on
Signature key ....: CF5A 305B 808B 7A0F 230D  A064 B3CD 10E5 02E1 9637
      created ....: 2024-01-01 12:00:00
Encryption key....: A5FA A005 5BED 4DC9 889D  38BC 30CB E8C4 B085 B9F7
      created ....: 2024-01-01 12:00:00
Authentication key: 570E 1355 6D01 4C04 8B6D  E2A3 AD9E 24E1 B8CB 9600
      created ....: 2024-01-01 12:00:00
General key info..: sub  rsa4096/0xB3CD10E502E19637 2024-01-01 YubiKey User <yubikey@example>
sec#  rsa4096/0xF0F2CFEB04341FB5  created: 2024-01-01  expires: never
ssb>  rsa4096/0xB3CD10E502E19637  created: 2024-01-01  expires: 2026-05-01
                                  card-no: 0006 05553211
ssb>  rsa4096/0x30CBE8C4B085B9F7  created: 2024-01-01  expires: 2026-05-01
                                  card-no: 0006 05553211
ssb>  rsa4096/0xAD9E24E1B8CB9600  created: 2024-01-01  expires: 2026-05-01
                                  card-no: 0006 05553211
```

`sec#` indicates the corresponding key is not available (the Certify key is offline).

YubiKey is now ready for use!

15.1. Encryption

Encrypt a message to yourself (useful for storing credentials or protecting backups):

```
echo "\ntest message string" | \
  gpg --encrypt --armor \
      --recipient $KEYID --output encrypted.txt
```

Decrypt the message - a prompt for the User PIN will appear:

```
gpg --decrypt --armor encrypted.txt
```

To encrypt to multiple recipients/keys, set the preferred key ID last:

```
echo "test message string" | \
  gpg --encrypt --armor \
      --recipient $KEYID_2 --recipient $KEYID_1 --recipient $KEYID \
      --output encrypted.txt
```

Use a shell function to make encrypting files easier:

```
secret () {
  output="${1}".$(date +%s).enc
  gpg --encrypt --armor --output ${output} \
    -r $KEYID "${1}" && echo "${1} -> ${output}"
}

reveal () {
  output=$(echo "${1}" | rev | cut -c16- | rev)
  gpg --decrypt --output ${output} "${1}" && \
    echo "${1} -> ${output}"
}
```

Example output:

```
$ secret document.pdf
document.pdf -> document.pdf.1580000000.enc

$ reveal document.pdf.1580000000.enc
gpg: anonymous recipient; trying secret key 0xF0F2CFEB04341FB5 ...
gpg: okay, we are the anonymous recipient.
gpg: encrypted with RSA key, ID 0x0000000000000000
document.pdf.1580000000.enc -> document.pdf
```

drduh/Purse is a password manager based on GnuPG and YubiKey to securely store and use credentials.

15.2. Signature

Sign a message:

```
echo "test message string" | gpg --armor --clearsign > signed.txt
```

Verify the signature:

```
gpg --verify signed.txt
```

The output will be similar to:

```
gpg: Signature made Mon 01 Jan 2024 12:00:00 PM UTC
gpg:                using RSA key CF5A305B808B7A0F230DA064B3CD10E502E19637
gpg: Good signature from "YubiKey User <yubikey@example>" [ultimate]
Primary key fingerprint: 4E2C 1FA3 372C BA96 A06A  C34A F0F2 CFEB 0434 1FB5
     Subkey fingerprint: CF5A 305B 808B 7A0F 230D  A064 B3CD 10E5 02E1 9637
```

15.3. Configure touch

By default, YubiKey will perform cryptographic operations without requiring any action from the user after the key is unlocked once with the PIN.

To require a touch for each key operation, use YubiKey Manager and the Admin PIN to set key policy.

Encryption:

```
ykman openpgp keys set-touch dec on
```

> **Note**
>
> Versions of YubiKey Manager before 5.1.0 use `enc` instead of `dec` for encryption:

```
ykman openpgp keys set-touch enc on
```

Even older versions of YubiKey Manager use `touch` instead of `set-touch`

Signature:

```
ykman openpgp keys set-touch sig on
```

Authentication:

```
ykman openpgp keys set-touch aut on
```

To view and adjust policy options:

```
ykman openpgp keys set-touch -h
```

`Cached` or `Cached-Fixed` may be desirable for YubiKey use with email clients.

YubiKey will blink when it is waiting for a touch. On Linux, maximbaz/yubikey-touch-detector can be used to indicate YubiKey is waiting for a touch.

15.4. SSH

Import or create a hardened configuration:

```
cd ~/.gnupg

wget https://raw.githubusercontent.com/drduh/config/master/gpg-agent.conf
```

Important The `cache-ttl` options do **not** apply when using YubiKey as a smart card, because the PIN is cached by the smart card itself. To clear the PIN from cache (equivalent to `default-cache-ttl` and `max-cache-ttl`), remove YubiKey, or set `forcesig` when editing the card to be prompted for the PIN each time.

Tip Set `pinentry-program` to `/usr/bin/pinentry-gnome3` for a GUI-based prompt.

macOS

Install pinentry with `brew install pinentry-mac` then edit `gpg-agent.conf` to set the `pinentry-program` path to:

- Apple Silicon Macs: `/opt/homebrew/bin/pinentry-mac`
- Intel Macs: `/usr/local/bin/pinentry-mac`
- MacGPG Suite: `/usr/local/MacGPG2/libexec/pinentry-mac.app/Contents/MacOS/pinentry-mac`

Then run `gpgconf --kill gpg-agent` for the change to take effect.

To use graphical applications on macOS, additional setup is required.

Create `$HOME/Library/LaunchAgents/gnupg.gpg-agent.plist` with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>gnupg.gpg-agent</string>
        <key>RunAtLoad</key>
        <true/>
        <key>KeepAlive</key>
        <false/>
        <key>ProgramArguments</key>
        <array>
            <string>/usr/local/MacGPG2/bin/gpg-connect-agent</string>
            <string>/bye</string>
        </array>
    </dict>
</plist>
```

Load it:

```
launchctl load $HOME/Library/LaunchAgents/gnupg.gpg-agent.plist
```

Create `$HOME/Library/LaunchAgents/gnupg.gpg-agent-symlink.plist` with the following contens:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/ProperyList-1.0/dtd">
<plist version="1.0">
    <dict>
        <key>Label</key>
        <string>gnupg.gpg-agent-symlink</string>
        <key>ProgramArguments</key>
        <array>
            <string>/bin/sh</string>
            <string>-c</string>
            <string>/bin/ln -sf $HOME/.gnupg/S.gpg-agent.ssh $SSH_AUTH_SOCK</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
    </dict>
</plist>
```

Load it:

```
launchctl load $HOME/Library/LaunchAgents/gnupg.gpg-agent-symlink.plist
```

Reboot or to activate changes.

Windows

Windows can already have some virtual smart card readers installed, like the one provided for Windows Hello. To verify YubiKey is the correct one used by scdaemon, add it to its configuration.

Find the YubiKey label using PowerShell:

```
PS C:\WINDOWS\system32> Get-PnpDevice -Class SoftwareDevice | Where-Object {$_.FriendlyName -like "*YubiKey*"} | Select-
Object -ExpandProperty FriendlyName
Yubico YubiKey OTP+FIDO+CCID 0
```

See How to setup Signed Git Commits with a YubiKey NEO and GPG and Keybase on Windows (2018) for more information.

Edit `%APPDATA%/gnupg/scdaemon.conf` to add:

```
reader-port <device name, e.g. Yubico YubiKey OTP+FIDO+CCID 0>
```

Edit `%APPDATA%/gnupg/gpg-agent.conf` to add:

```
enable-ssh-support
enable-putty-support
```

Restart the agent:

```
gpg-connect-agent killagent /bye

gpg-connect-agent /bye
```

Verify YubiKey details:

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

Import the public key and set ultimate trust:

```
gpg --import <path to public key file>
```

Retrieve the public key id:

```
gpg --list-public-keys
```

Export the SSH public key:

```
gpg --export-ssh-key <public key id>
```

Copy the public SSH key to a file - it corresponds to the secret key on YubiKey and can be copied to SSH destination hosts.

Create a shortcut that points to `gpg-connect-agent /bye` and place it in the startup folder `shell:startup` to make sure the agent starts after reboot. Modify the shortcut properties so it starts in a “Minimized” window.

PuTTY can now be used for public-key SSH authentication. When the server asks for public-key verification, PuTTY will forward the request to GnuPG, which will prompt for a PIN to authorize the operation.

WSL

The goal is to configure SSH client inside WSL work together with the Windows agent, such as gpg-agent.exe.

See the WSL agent architecture illustration for an overview.

> **Note**
>
> GnuPG forwarding for cryptographic operations is not supported. See vuori/weasel-pageant for more information.

One way to forward is just `ssh -A` (still need to eval weasel to setup local ssh-agent), and only relies on OpenSSH. In this track, `ForwardAgent` and `AllowAgentForwarding` in ssh/sshd config may be involved. However, when using ssh socket forwarding, do not enable `ForwardAgent` in ssh config. See SSH Agent Forwarding for more information. This requires Ubuntu 16.04 or newer for WSL and Kleopatra.

Download vuori/weasel-pageant.

Add `eval $(/mnt/c/<path of extraction>/weasel-pageant -r -a /tmp/S.weasel-pageant)` to the shell rc file. Use a named socket here so it can be used in the `RemoteForward` directive of `~/.ssh/config`. Source it with `source ~/.bashrc`.

Display the SSH key with `$ ssh-add -l`

Edit `~/.ssh/config` to add the following for each agent forwarding host:

```
RemoteForward <remote SSH socket path> /tmp/S.weasel-pageant
```

> **Note**
>
> The remote SSH socket path can be found with `gpgconf --list-dirs agent-ssh-socket`

Add the following to the shell rc file:

```
export SSH_AUTH_SOCK=$(gpgconf --list-dirs agent-ssh-socket)
```

Add the following to `/etc/ssh/sshd_config`:

```
StreamLocalBindUnlink yes
```

Reload SSH daemon:

```
sudo service sshd reload
```

Remove YubiKey and reboot. Log back into Windows, open a WSL console and enter `ssh-add -l` - no output should appear.

Plug in YubiKey, enter the same command to display the ssh key.

Connect to the remote host and use `ssh-add -l` to confirm forwarding works.

Agent forwarding may be chained through multiple hosts. Follow the same protocol to configure each host.

15.4.1. Replace agents

To launch `gpg-agent` for use by SSH, use the `gpg-connect-agent /bye` or `gpgconf --launch gpg-agent` commands.

Add the following to the shell rc file:

```
export GPG_TTY="$(tty)"
export SSH_AUTH_SOCK="/run/user/$UID/gnupg/S.gpg-agent.ssh"
gpg-connect-agent updatestartuptty /bye > /dev/null
```

On modern systems, `gpgconf --list-dirs agent-ssh-socket` will automatically set `SSH_AUTH_SOCK` to the correct value and is better than hard-coding to `run/user/$UID/gnupg/S.gpg-agent.ssh`, if available:

```
export GPG_TTY="$(tty)"
export SSH_AUTH_SOCK=$(gpgconf --list-dirs agent-ssh-socket)
gpgconf --launch gpg-agent
```

For fish, `config.fish` should look like this (consider putting them into the `is-interactive` block):

```
set -x GPG_TTY (tty)
set -x SSH_AUTH_SOCK (gpgconf --list-dirs agent-ssh-socket)
gpgconf --launch gpg-agent
```

When using `ForwardAgent` for ssh-agent forwarding, `SSH_AUTH_SOCK` only needs to be set on the **local** host, where YubiKey is connected. On the **remote** host, `ssh` will set `SSH_AUTH_SOCK` to something like `/tmp/ssh-mXzCzYT2Np/agent.7541` upon connection. Do **not** set `SSH_AUTH_SOCK` on the remote host - doing so will break SSH Agent Forwarding.

For `S.gpg-agent.ssh` (see SSH Agent Forwarding for more info), `SSH_AUTH_SOCK` should also be set on the **remote**. However, `GPG_TTY` should not be set on the **remote**, explanation specified in that section.

15.4.2. Copy public key

> **Note**
>
> It is **not** necessary to import the GnuPG public key in order to use SSH only.

Copy and paste the output from `ssh-add` to the server's `authorized_keys` file:

```
$ ssh-add -L
ssh-rsa AAAAB4NzaC1yc2EAAAADAQABAAACAz[...]zreOKM+HwpkHzcy9DQcVG2Nw== cardno:000605553211
```

Optional Save the public key for identity file configuration. By default, SSH attempts to use all the identities available via the agent. It's often a good idea to manage exactly which keys SSH will use to connect to a server, for example to separate different roles or to avoid being fingerprinted by untrusted ssh servers. To do this you'll need to use the command line argument `-i [identity_file]` or the `IdentityFile` and `IdentitiesOnly` options in `.ssh/config`.

The argument provided to `IdentityFile` is traditionally the path to the *private* key file (for example `IdentityFile ~/.ssh/id_rsa`). For YubiKey, `IdentityFile` must point to the *public* key file, and `ssh` will select the appropriate private key from those available via ssh-agent. To prevent `ssh` from trying all keys in the agent, use `IdentitiesOnly yes` along with one or more `-i` or `IdentityFile` options for the target host.

To reiterate, with `IdentitiesOnly yes`, `ssh` will not enumerate public keys loaded into `ssh-agent` or `gpg-agent`. This means public-key authentication will not proceed unless explicitly named by `ssh -i [identity_file]` or in `.ssh/config` on a per-host basis.

In the case of YubiKey usage, to extract the public key from the ssh agent:

```
ssh-add -L | grep "cardno:000605553211" > ~/.ssh/id_rsa_yubikey.pub
```

Then explicitly associate this YubiKey-stored key for used with a host, `github.com` for example, as follows:

```
$ cat << EOF >> ~/.ssh/config
Host github.com
    IdentitiesOnly yes
    IdentityFile ~/.ssh/id_rsa_yubikey.pub
EOF
```

Connect with public key authentication:

```
$ ssh git@github.com -vvv
[...]
debug2: key: cardno:000605553211 (0x1234567890),
debug1: Authentications that can continue: publickey
debug3: start over, passed a different list publickey
debug3: preferred gssapi-keyex,gssapi-with-mic,publickey,keyboard-interactive,password
debug3: authmethod_lookup publickey
debug3: remaining preferred: keyboard-interactive,password
debug3: authmethod_is_enabled publickey
debug1: Next authentication method: publickey
debug1: Offering RSA public key: cardno:000605553211
debug3: send_pubkey_test
debug2: we sent a publickey packet, wait for reply
debug1: Server accepts key: pkalg ssh-rsa blen 535
debug2: input_userauth_pk_ok: fp e5:de:a5:74:b1:3e:96:9b:85:46:e7:28:53:b4:82:c3
debug3: sign_and_send_pubkey: RSA e5:de:a5:74:b1:3e:96:9b:85:46:e7:28:53:b4:82:c3
debug1: Authentication succeeded (publickey).
[...]
```

Tip To make multiple connections or securely transfer many files, use the ControlMaster ssh option.

15.4.3. Import SSH keys

If there are existing SSH keys to make available via `gpg-agent`, they will need to be imported. Then, remove the original private keys. When importing the key, `gpg-agent` uses the key filename as the label - this makes it easier to follow where the key originated from. In this example, we're starting with just the YubiKey in place and importing `~/.ssh/id_rsa`:

```
$ ssh-add -l
4096 SHA256:... cardno:00060123456 (RSA)

$ ssh-add ~/.ssh/id_rsa && rm ~/.ssh/id_rsa
```

When invoking `ssh-add`, a prompt for the SSH key passphrase will appear, then the `pinentry` program will prompt and confirm a new passphrase to encrypt the converted key within the GnuPG key store.

The migrated key will be listed in `ssh-add -l`:

```
$ ssh-add -l
4096 SHA256:... cardno:00060123456 (RSA)
2048 SHA256:... /Users/username/.ssh/id_rsa (RSA)
```

To show the keys with MD5 fingerprints, as used by `gpg-connect-agent`'s `KEYINFO` and `DELETE_KEY` commands:

```
$ ssh-add -E md5 -l
4096 MD5:... cardno:00060123456 (RSA)
2048 MD5:... /Users/username/.ssh/id_rsa (RSA)
```

When using the key `pinentry` will be invoked to request the key passphrase. The passphrase will be cached for up to 10 idle minutes between uses, up to a maximum of 2 hours.

15.4.4. SSH agent forwarding

> ⚠ **Warning**
>
> SSH Agent Forwarding can add additional risk - proceed with caution!

There are two methods for ssh-agent forwarding, one is provided by OpenSSH and the other is provided by GnuPG.

The latter one may be more insecure as raw socket is just forwarded (not like `S.gpg-agent.extra` with only limited functionality; if `ForwardAgent` implemented by OpenSSH is just forwarding the raw socket, then they are insecure to the same degree). But for the latter one, one convenience is that one may forward once and use this agent everywhere in the remote. So again, proceed with caution!

For example, tmux does not have environment variables such as `$SSH_AUTH_SOCK` when connecting to remote hosts and attaching an existing session. For each shell, find the socket and `export SSH_AUTH_SOCK=/tmp/ssh-agent-xxx/xxxx.socket`. However, with `S.gpg-agent.ssh` in a fixed place, it can be used as the ssh-agent in shell rc files.

15.4.4.1. Use ssh-agent

You should now be able to use `ssh -A remote` on the *local* host to log into *remote* host, and should then be able to use YubiKey as if it were connected to the remote host. For example, using e.g. `ssh-add -l` on that remote host will show the public key from the YubiKey (`cardno:`). Always use `ForwardAgent yes` only for a single host, never for all servers.

15.4.4.2. Use S.gpg-agent.ssh

First you need to go through GnuPG agent forwarding), know the conditions for gpg-agent forwarding and know the location of `S.gpg-agent.ssh` on both the local and the remote.

You may use the command:

```
$ gpgconf --list-dirs agent-ssh-socket
```

Edit `.ssh/config` to add the remote host:

```
Host
  Hostname remote-host.tld
  StreamLocalBindUnlink yes
  RemoteForward /run/user/1000/gnupg/S.gpg-agent.ssh /run/user/1000/gnupg/S.gpg-agent.ssh
  # RemoteForward [remote socket] [local socket]
  # Note that ForwardAgent is not wanted here!
```

After successfully ssh into the remote host, confirm `/run/user/1000/gnupg/S.gpg-agent.ssh` exists.

Then in the **remote** you can type in command line or configure in the shell rc file with:

```
export SSH_AUTH_SOCK="/run/user/$UID/gnupg/S.gpg-agent.ssh"
```

After sourcing the shell rc file, `ssh-add -l` will return the correct public key.

> **Note**
>
> In this process no gpg-agent in the remote is involved, hence `gpg-agent.conf` in the remote is of no use. Also pinentry is invoked locally.

15.4.4.3. Chained forwarding

If you use `ssh-agent` provided by OpenSSH and want to forward it into a **third** box, you can just `ssh -A third` on the **remote**.

Meanwhile, if you use `S.gpg-agent.ssh`, assume you have gone through the steps above and have `S.gpg-agent.ssh` on the *remote*, and you would like to forward this agent into a *third* box, first you may need to configure `sshd_config` and `SSH_AUTH_SOCK` of *third* in the same way as *remote*, then in the ssh config of *remote*, add the following lines

```
Host third
  Hostname third-host.tld
  StreamLocalBindUnlink yes
  RemoteForward /run/user/1000/gnupg/S.gpg-agent.ssh /run/user/1000/gnupg/S.gpg-agent.ssh
  #RemoteForward [remote socket] [local socket]
  #Note that ForwardAgent is not wanted here!
```

The path must be set according to `gpgconf --list-dirs agent-ssh-socket` on **remote** and **third** hosts.

15.5. GitHub

YubiKey can be used to sign commits and tags, and authenticate SSH to GitHub when configured in Settings.

Configure a signing key:

```
git config --global user.signingkey $KEYID
```

Important The `user.email` option must match the email address associated with the PGP identity.

To sign commits or tags, use the `-S` option.

Windows

Configure authentication:

```
git config --global core.sshcommand "plink -agent"

git config --global gpg.program 'C:\Program Files (x86)\GnuPG\bin\gpg.exe'
```

Then update the repository URL to `git@github.com:USERNAME/repository`

> **Note**
>
> For the error `gpg: signing failed: No secret key` - run `gpg --card-status` with YubiKey plugged in and try the git command again.

15.6. GnuPG agent forwarding

YubiKey can be used sign git commits and decrypt files on remote hosts with GnuPG Agent Forwarding. To ssh through another network, especially to push to/pull from GitHub using ssh, see Remote Machines (SSH Agent forwarding).

`gpg-agent.conf` is not needed on the remote host; after forwarding, remote GnuPG directly communicates with `S.gpg-agent` without starting `gpg-agent` on the remote host.

On the remote host, edit `/etc/ssh/sshd_config` to set `StreamLocalBindUnlink yes`

Optional Without root access on the remote host to edit `/etc/ssh/sshd_config`, socket located at `gpgconf --list-dir agent-socket` on the remote host will need to be removed before forwarding works. See AgentForwarding GNUPG wiki page for more information.

Import the public key on the remote host. On the local host, copy the public keyring to the remote host:

```
scp ~/.gnupg/pubring.kbx remote:~/.gnupg/
```

On modern distributions, such as Fedora 30, there is no need to set `RemoteForward` in `~/.ssh/config`

15.6.1. Legacy distributions

On the local host, run:

```
gpgconf --list-dirs agent-extra-socket
```

This should return a path to agent-extra-socket - `/run/user/1000/gnupg/S.gpg-agent.extra` - though on older Linux distros (and macOS) it may be `/home/<user>/.gnupg/S/gpg-agent.extra`

Find the agent socket on the **remote** host:

```
gpgconf --list-dirs agent-socket
```

This should return a path such as `/run/user/1000/gnupg/S.gpg-agent`

Finally, enable agent forwarding for a given host by adding the following to the local host's `~/.ssh/config` (agent sockets may differ):

```
Host
  Hostname remote-host.tld
  StreamLocalBindUnlink yes
  RemoteForward /run/user/1000/gnupg/S.gpg-agent /run/user/1000/gnupg/S.gpg-agent.extra
  #RemoteForward [remote socket] [local socket]
```

It may be necessary to edit `gpg-agent.conf` on the **local** host to add the following information:

```
pinentry-program /usr/bin/pinentry-gtk-2
extra-socket /run/user/1000/gnupg/S.gpg-agent.extra
```

> **Note**
>
> The pinentry program starts on the **local** host, not remote.

Important Any pinentry program except `pinentry-tty` or `pinentry-curses` may be used. This is because local `gpg-agent` may start headlessly (by systemd without `$GPG_TTY` set locally telling which tty it is on), thus failed to obtain the pin. Errors on the remote may be misleading saying that there is **IO Error**. (Yes, internally there is actually an **IO Error** since it happens when writing to/reading from tty while finding no tty to use, but for end users this is not friendly.)

See Issue 85 for more information and troubleshooting.

15.6.2. Chained GnuPG agent forwarding

Assume you have gone through the steps above and have `S.gpg-agent` on the **remote**, and you would like to forward this agent into a **third** box, first you may need to configure `sshd_config` of **third** in the same way as **remote**, then in the ssh config of **remote**, add the following lines:

```
Host third
  Hostname third-host.tld
  StreamLocalBindUnlink yes
  RemoteForward /run/user/1000/gnupg/S.gpg-agent /run/user/1000/gnupg/S.gpg-agent
  #RemoteForward [remote socket] [local socket]
```

You should change the path according to `gpgconf --list-dirs agent-socket` on **remote** and **third**.

> **Note**
>
> On **local** you have `S.gpg-agent.extra` whereas on **remote** and **third**, you only have `S.gpg-agent`

15.7. Using multiple YubiKeys

When a GnuPG key is added to YubiKey using `keytocard`, the key is deleted from the keyring and a **stub** is added, pointing to the YubiKey. The stub identifies the GnuPG key ID and YubiKey serial number.

When a Subkey is added to an additional YubiKey, the stub is overwritten and will now point to the latest YubiKey. GnuPG will request a specific YubiKey by serial number, as referenced by the stub, and will not recognize another YubiKey with a different serial number.

To scan an additional YubiKey and recreate the correct stub:

```
gpg-connect-agent "scd serialno" "learn --force" /bye
```

Alternatively, use a script to delete the GnuPG shadowed key, where the card serial number is stored (see GnuPG #T2291):

```
cat >> ~/scripts/remove-keygrips.sh <<EOF
#!/usr/bin/env bash
(( $# )) || { echo "Specify a key." >&2; exit 1; }
KEYGRIPS=$(gpg --with-keygrip --list-secret-keys "$@" | awk '/Keygrip/ { print $3 }')
for keygrip in $KEYGRIPS
do
    rm "$HOME/.gnupg/private-keys-v1.d/$keygrip.key" 2> /dev/null
done

gpg --card-status
EOF

chmod +x ~/scripts/remove-keygrips.sh

~/scripts/remove-keygrips.sh $KEYID
```

See discussion in Issues 19 and 112 for more information and troubleshooting steps.

15.8. Email

YubiKey can be used to decrypt and sign emails and attachments using Thunderbird, Enigmail and Mutt. Thunderbird supports OAuth 2 authentication and can be used with Gmail. See this EFF guide for more information. Mutt has OAuth 2 support since version 2.0.

15.8.1. Mailvelope

Mailvelope allows YubiKey to be used with Gmail and others.

Important Mailvelope does not work with the `throw-keyids` option set in `gpg.conf`

On macOS, install gpgme using Homebrew:

```
brew install gpgme
```

To allow Chrome to run gpgme, edit `~/Library/Application\ Support/Google/Chrome/NativeMessagingHosts/gpgmejson.json` to add:

```
{
    "name": "gpgmejson",
    "description": "Integration with GnuPG",
    "path": "/usr/local/bin/gpgme-json",
    "type": "stdio",
    "allowed_origins": [
        "chrome-extension://kajibbejlbohfaggdiogboambcijhkke/"
    ]
}
```

Edit the default path to allow Chrome to find GnuPG:

```
sudo launchctl config user path /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

Finally, install the Mailvelope extension from the Chrome web store.

15.8.2. Mutt

Mutt has both CLI and TUI interfaces - the latter provides powerful functions for processing email. In addition, PGP can be integrated such that cryptographic operations can be done without leaving TUI.

To enable GnuPG support, copy `/usr/share/doc/mutt/samples/gpg.rc`

Edit the file to enable options `pgp_default_key`, `pgp_sign_as` and `pgp_autosign`

`source` the file in `muttrc`

Important `pinentry-tty` set as the pinentry program in `gpg-agent.conf` is reported to cause problems with Mutt TUI, because it uses curses. It is recommended to use `pinentry-curses` or other graphic pinentry program instead.

15.9. Keyserver

Public keys can be uploaded to a public server for discoverability:

```
gpg --send-key $KEYID

gpg --keyserver keys.gnupg.net --send-key $KEYID

gpg --keyserver hkps://keyserver.ubuntu.com:443 --send-key $KEYID
```

Or if uploading to keys.openpgp.org:

```
gpg --send-key $KEYID | curl -T - https://keys.openpgp.org
```

The public key URL can also be added to YubiKey (based on Shaw 2003):

```
URL="hkps://keyserver.ubuntu.com:443/pks/lookup?op=get&search=${KEYID}"
```

Edit YubiKey with `gpg --edit-card` and the Admin PIN:

```
gpg/card> admin

gpg/card> url
URL to retrieve public key: hkps://keyserver.ubuntu.com:443/pks/lookup?op=get&search=0xFF00000000000000

gpg/card> quit
```

16. Updating keys

PGP does not provide forward secrecy, meaning a compromised key may be used to decrypt all past messages. Although keys stored on YubiKey are more difficult to exploit, it is not impossible: the key and PIN could be physically compromised, or a vulnerability may be discovered in firmware or in the random number generator used to create keys, for example. Therefore, it is recommended practice to rotate Subkeys periodically.

When a Subkey expires, it can either be renewed or replaced. Both actions require access to the Certify key.

- Renewing Subkeys by updating expiration indicates continued possession of the Certify key and is more convenient.
- Replacing Subkeys is less convenient but potentially more secure: the new Subkeys will **not** be able to decrypt previous messages, authenticate with SSH, etc. Contacts will need to receive the updated public key and any encrypted secrets

need to be decrypted and re-encrypted to new Subkeys to be usable. This process is functionally equivalent to losing the YubiKey and provisioning a new one.

Neither rotation method is superior and it is up to personal philosophy on identity management and individual threat modeling to decide which one to use, or whether to expire Subkeys at all. Ideally, Subkeys would be ephemeral: used only once for each unique encryption, signature and authentication event, however in practice that is not really practical nor worthwhile with YubiKey. Advanced users may dedicate an air-gapped machine for frequent credential rotation.

To renew or rotate Subkeys, follow the same process as generating keys: boot to a secure environment, install required software and disable networking.

Connect the portable storage device with the Certify key and identify the disk label.

Decrypt and mount the encrypted volume:

```
sudo cryptsetup luksOpen /dev/sdc1 gnupg-secrets

sudo mkdir /mnt/encrypted-storage

sudo mount /dev/mapper/gnupg-secrets /mnt/encrypted-storage
```

Mount the non-encrypted public partition:

```
sudo mkdir /mnt/public

sudo mount /dev/sdc2 /mnt/public
```

Copy the original private key materials to a temporary working directory:

```
export GNUPGHOME=$(mktemp -d -t gnupg-$(date +%Y-%m-%d)-XXXXXXXXXX)

cd $GNUPGHOME

cp -avi /mnt/encrypted-storage/gnupg-*/* $GNUPGHOME
```

Confirm the identity is available, set the key id and fingerprint:

```
gpg -K

export KEYID=$(gpg -k --with-colons "$IDENTITY" | awk -F: '/^pub:/ { print $5; exit }')

export KEYFP=$(gpg -k --with-colons "$IDENTITY" | awk -F: '/^fpr:/ { print $10; exit }')

echo $KEYID $KEYFP
```

Recall the Certify key passphrase and set it, for example:

```
export CERTIFY_PASS=ABCD-0123-IJKL-4567-QRST-UVWX
```

16.1. Renew Subkeys

Determine the updated expiration, for example:

```
export EXPIRATION=2026-09-01

export EXPIRATION=2y
```

Renew the Subkeys:

```
gpg --batch --pinentry-mode=loopback \
  --passphrase "$CERTIFY_PASS" --quick-set-expire "$KEYFP" "$EXPIRATION" \
  $(gpg -K --with-colons | awk -F: '/^fpr:/ { print $10 }' | tail -n "+2" | tr "\n" " ")
```

Export the updated public key:

```
gpg --armor --export $KEYID | sudo tee /mnt/public/$KEYID-$(date +%F).asc
```

Transfer the public key to the destination host and import it:

```
gpg --import /mnt/public/*.asc
```

Alternatively, publish to a public key server and download it:

```
gpg --send-key $KEYID

gpg --recv $KEYID
```

The validity of the GnuPG identity will be extended, allowing it to be used again for encryption and signature operations.

The SSH public key does **not** need to be updated on remote hosts.

16.2. Rotate Subkeys

Follow the original procedure to Create Subkeys.

Previous Subkeys can be deleted from the identity.

Finish by transfering new Subkeys to YubiKey.

Copy the **new** temporary working directory to encrypted storage, which is still mounted:

```
sudo cp -avi $GNUPGHOME /mnt/encrypted-storage
```

Unmount and close the encrypted volume:

```
sudo umount /mnt/encrypted-storage

sudo cryptsetup luksClose gnupg-secrets
```

Export the updated public key:

```
sudo mkdir /mnt/public

sudo mount /dev/sdc2 /mnt/public

gpg --armor --export $KEYID | sudo tee /mnt/public/$KEYID-$(date +%F).asc

sudo umount /mnt/public
```

Remove the storage device and follow the original steps to transfer new Subkeys (`4`, `5` and `6`) to YubiKey, replacing existing ones.

Reboot or securely erase the GnuPG temporary working directory.

17. Reset YubiKey

If PIN attempts are exceeded, the YubiKey is locked and must be Reset and set up again using the encrypted backup.

Copy the following to a file and run `gpg-connect-agent -r $file` to lock and terminate the card. Then re-insert YubiKey to complete reset.

```
/hex
scd serialno
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 e6 00 00
scd apdu 00 44 00 00
/echo Card has been successfully reset.
/bye
```

Or use `ykman` (sometimes in `~/.local/bin/`):

```
$ ykman openpgp reset
WARNING! This will delete all stored OpenPGP keys and data and restore factory settings? [y/N]: y
Resetting OpenPGP data, don't remove your YubiKey...
Success! All data has been cleared and default PINs are set.
PIN:         123456
Reset code:  NOT SET
Admin PIN:   12345678
```

18. Optional hardening

The following steps may improve the security and privacy of YubiKey.

18.1. Improving entropy

Generating cryptographic keys requires high-quality randomness, measured as entropy. Most operating systems use software-based pseudorandom number generators or CPU-based hardware random number generators (HRNG).

Optionally, a device such as OneRNG may be used to increase the speed and possibly the quality of available entropy.

Before creating keys, configure rng-tools:

```
sudo apt -y install at rng-tools python3-gnupg openssl

wget https://github.com/OneRNG/onerng.github.io/raw/master/sw/onerng_3.7-1_all.deb
```

Verify the package:

```
sha256sum onerng_3.7-1_all.deb
```

The value must match:

```
b7cda2fe07dce219a95dfeabeb5ee0f662f64ba1474f6b9dddacc3e8734d8f57
```

Install the package:

```
sudo dpkg -i onerng_3.7-1_all.deb

echo "HRNGDEVICE=/dev/ttyACM0" | sudo tee /etc/default/rng-tools
```

Insert the device and restart rng-tools:

```
sudo atd

sudo service rng-tools restart
```

18.2. Enable KDF

> **Note**
>
> This feature may not be compatible with older GnuPG versions, especially mobile clients. These incompatible clients will not function because the PIN will always be rejected.

This step must be completed before changing PINs or moving keys or an error will occur: `gpg: error for setup KDF: Conditions of use not satisfied`

Key Derived Function (KDF) enables YubiKey to store the hash of PIN, preventing the PIN from being passed as plain text.

Enable KDF using the default Admin PIN of `12345678`:

```
gpg --command-fd=0 --pinentry-mode=loopback --card-edit <<EOF
admin
kdf-setup
12345678
EOF
```

18.3. Network considerations

This section is primarily focused on Debian / Ubuntu based systems, but the same concept applies to any system connected to a network.

Whether you're using a VM, installing on dedicated hardware, or running a Live OS temporarily, start **without** a network connection and disable any unnecessary services listening on all interfaces before connecting to the network.

The reasoning for this is because services like cups or avahi can be listening by default. While this isn't an immediate problem it simply broadens the attack surface. Not everyone will have a dedicated subnet or trusted network equipment they can control, and for the purposes of this guide, these steps treat **any** network as untrusted / hostile.

Disable Listening Services

- Ensures only essential network services are running
- If the service doesn't exist you'll get a "Failed to stop" which is fine
- Only disable `Bluetooth` if you don't need it

```
sudo systemctl stop bluetooth exim4 cups avahi avahi-daemon sshd
```

Firewall

Enable a basic firewall policy of **deny inbound, allow outbound**. Note that Debian does not come with a firewall, simply disabling the services in the previous step is fine. The following options have Ubuntu and similar systems in mind.

On Ubuntu, `ufw` is built in and easy to enable:

```
sudo ufw enable
```

On systems without `ufw`, `nftables` is replacing `iptables`. The nftables wiki has examples for a baseline **deny inbound, allow outbound** policy. The `fw.inet.basic` policy covers both IPv4 and IPv6.

(Remember to download this README and any other resources to another external drive when creating the bootable media, to have this information ready to use offline)

Regardless of which policy you use, write the contents to a file (e.g. `nftables.conf`) and apply the policy with the following comand:

```
sudo nft -f ./nftables.conf
```

Review the System State

`NetworkManager` should be the only listening service on port 68/udp to obtain a DHCP lease (and 58/icmp6 if you have IPv6).

If you want to look at every process's command line arguments you can use `ps axjf`. This prints a process tree which may have a large number of lines but should be easy to read on a live image or fresh install.

```
sudo ss -anp -A inet    # Dump all network state information
ps axjf                 # List all processes in a process tree
ps aux                  # BSD syntax, list all processes but no process tree
```

If you find any additional processes listening on the network that aren't needed, take note and disable them with one of the following:

```
sudo systemctl stop <process-name>                      # Stops services managed by systemctl
sudo pkill -f '<process-name-or-command-line-string>'   # Terminate the process by matching it's command line string
pgrep -f '<process-name-or-command-line-string>'        # Obtain the PID
sudo kill <pid>                                         # Terminate the process via its PID
```

Now connect to a network.

19. Notes

1. YubiKey has two configurations, invoked with either a short or long press. By default, the short-press mode is configured for HID OTP; a brief touch will emit an OTP string starting with `ccccccc`. OTP mode can be swapped to the second configuration via the YubiKey Personalization tool or disabled entirely using YubiKey Manager: `ykman config usb -d OTP`
2. Using YubiKey for GnuPG does not prevent use of other features, such as WebAuthn and OTP.

3. Add additional identities to a Certify key with the `adduid` command during setup, then trust it ultimately with `trust` and `5` to configure for use.
4. To switch between YubiKeys, remove the first YubiKey and restart gpg-agent, ssh-agent and pinentry with `pkill "gpg-agent|ssh-agent|pinentry" ; eval $(gpg-agent --daemon --enable-ssh-support)` then insert the other YubiKey and run `gpg-connect-agent updatestartuptty /bye`
5. To use YubiKey on multiple computers, import the corresponding public keys, then confirm YubiKey is visible with `gpg --card-status`. Trust the imported public keys ultimately with `trust` and `5`, then `gpg --list-secret-keys` will show the correct and trusted key.

20. Troubleshooting

- Use `man gpg` to understand GnuPG options and command-line flags.
- To get more information on potential errors, restart the `gpg-agent` process with debug output to the console with `pkill gpg-agent; gpg-agent --daemon --no-detach -v -v --debug-level advanced --homedir ~/.gnupg`.
- A lot of issues can be fixed by removing and re-inserting YubiKey, or restarting the `gpg-agent` process.
- If you receive the error, `Yubikey core error: no yubikey present` - make sure the YubiKey is inserted correctly. It should blink once when plugged in.
- If you still receive the error, `Yubikey core error: no yubikey present` - you likely need to install newer versions of yubikey-personalize as outlined in Install software.
- If you see `General key info..: [none]` in card status output - import the public key.
- If you receive the error, `gpg: decryption failed: secret key not available` - you likely need to install GnuPG version 2.x. Another possibility is that there is a problem with the PIN, e.g., it is too short or blocked.
- If you receive the error, `Yubikey core error: write error` - YubiKey is likely locked. Install and run yubikey-personalization-gui to unlock it.
- If you receive the error, `Key does not match the card's capability` - you likely need to use 2048-bit RSA key sizes.
- If you receive the error, `sign_and_send_pubkey: signing failed: agent refused operation` - make sure you replaced `ssh-agent` with `gpg-agent` as noted above.
- If you still receive the error, `sign_and_send_pubkey: signing failed: agent refused operation` - run the command `gpg-connect-agent updatestartuptty /bye`
- If you still receive the error, `sign_and_send_pubkey: signing failed: agent refused operation` - edit `~/.gnupg/gpg-agent.conf` to set a valid `pinentry` program path. `gpg: decryption failed: No secret key` could also indicate an invalid `pinentry` path
- If you still receive the error, `sign_and_send_pubkey: signing failed: agent refused operation` - it is a known issue that openssh 8.9p1 and higher has issues with YubiKey. Adding `KexAlgorithms -sntrup761x25519-sha512@openssh.com` to `/etc/ssh/ssh_config` often resolves the issue.
- If you receive the error, `The agent has no identities` from `ssh-add -L`, make sure you have installed and started `scdaemon`
- If you receive the error, `Error connecting to agent: No such file or directory` from `ssh-add -L`, the UNIX file socket that the agent uses for communication with other processes may not be set up correctly. On Debian, try `export SSH_AUTH_SOCK="/run/user/$UID/gnupg/S.gpg-agent.ssh"`. Also see that `gpgconf --list-dirs agent-ssh-socket` is returning single path, to existing `S.gpg-agent.ssh` socket.
- If you receive the error, `Permission denied (publickey)`, increase ssh verbosity with the `-v` flag and verify the public key from the card is being offered: `Offering public key: RSA SHA256:abcdefg... cardno:00060123456`. If it is, verify the correct user the target system - not the user on the local system. Otherwise, be sure `IdentitiesOnly` is not enabled for this host.
- If SSH authentication still fails - add up to 3 `-v` flags to the `ssh` command to increase verbosity.
- If it still fails, it may be useful to stop the background `sshd` daemon process service on the server (e.g. using `sudo systemctl stop sshd`) and instead start it in the foreground with extensive debugging output, using `/usr/sbin/sshd -eddd`. Note that the server will not fork and will only process one connection, therefore has to be re-started after every `ssh` test.
- If you receive the error, `Please insert the card with serial number` see Using Multiple Keys.
- If you receive the error, `There is no assurance this key belongs to the named user` or `encryption failed: Unusable public key` or `No public key` use `gpg --edit-key` to set `trust` to `5 = I trust ultimately`

- If, when you try the above command, you get the error `Need the secret key to do this` - specify trust for the key in `~/.gnupg/gpg.conf` by using the `trust-key [key ID]` directive.
- If, when using a previously provisioned YubiKey on a new computer with `pass`, you see the following error on `pass insert`, you need to adjust the trust associated with the key. See the note above.

```
gpg: 0x0000000000000000: There is no assurance this key belongs to the named user
gpg: [stdin]: encryption failed: Unusable public key
```

- If you receive the error, `gpg: 0x0000000000000000: skipped: Unusable public key`, `signing failed: Unusable secret key`, or `encryption failed: Unusable public key` the Subkey may be expired and can no longer be used to encrypt nor sign messages. It can still be used to decrypt and authenticate, however.
- If the *pinentry* graphical dialog does not show and this error appears: `sign_and_send_pubkey: signing failed: agent refused operation`, install the `dbus-user-session` package and restart for the `dbus` user session to be fully inherited. This is because `pinentry` complains about `No $DBUS_SESSION_BUS_ADDRESS found`, falls back to `curses` but doesn't find the expected `tty`
- If, when you try the above `--card-status` command, you get receive the error, `gpg: selecting card failed: No such device` or `gpg: OpenPGP card not available: No such device`, it's possible that the latest release of pcscd is now requires polkit rules to operate properly. Create the following file to allow users in the `wheel` group to use the card. Be sure to restart pcscd when you're done to allow the new rules to take effect.

```
cat << EOF >  /etc/polkit-1/rules.d/99-pcscd.rules
polkit.addRule(function(action, subject) {
        if (action.id == "org.debian.pcsc-lite.access_card" &&
                subject.isInGroup("wheel")) {
                return polkit.Result.YES;
        }
});
polkit.addRule(function(action, subject) {
        if (action.id == "org.debian.pcsc-lite.access_pcsc" &&
                subject.isInGroup("wheel")) {
                return polkit.Result.YES;
        }
});
EOF
```

- If the public key is lost, follow this guide to recover it from YubiKey.
- Refer to Yubico article Troubleshooting Issues with GPG for additional guidance.

21. Alternative solutions

- `vorburger/ed25519-sk.md` - use YubiKey for SSH without GnuPG
- `smlx/piv-agent` - SSH and GnuPG agent which can be used with PIV devices
- `keytotpm` - use GnuPG with TPM systems

22. Additional resources

- Yubico - PGP
- Yubico - Yubikey Personalization
- A Visual Explanation of GPG Subkeys (2022)
- dhess/nixos-yubikey
- lsasolutions/makegpg
- Trammell Hudson - Yubikey (2020)
- Yubikey forwarding SSH keys (2019)
- GPG Agent Forwarding (2018)
- Stick with security: YubiKey, SSH, GnuPG, macOS (2018)
- PGP and SSH keys on a Yubikey NEO (2015)
- Offline GnuPG Master Key and Subkeys on YubiKey NEO Smartcard (2014)
- Creating the perfect GPG keypair (2013)