This is a guide to using YubiKey as a SmartCard for storing GPG encryption, signing and authentication keys, which can also be used for SSH. Many of the principles in this document are applicable to other smart card devices.

Keys stored on YubiKey are non-exportable (as opposed to file-based keys that are stored on disk) and are convenient for everyday use. Instead of having to remember and enter passphrases to unlock SSH/GPG keys, YubiKey needs only a physical touch after being unlocked with a PIN code. All signing and encryption operations happen on the card, rather than in OS memory.

New! drduh/Purse is a password manager which uses GPG and YubiKey.

If you have a comment or suggestion, please open an issue on GitHub.

- Purchase YubiKey
- Live image
- Required software
- Entropy
- Creating keys
- Master key
- Subkeys
- Signing
- Encryption
- Authentication
- Verify keys
- Export keys
- Backup keys
- Linux
- OpenBSD
- Configure YubiKey
- Configure Smartcard
- Change PIN
- Set information
- Transfer keys
- Signing
- Encryption
- Authentication
- Verify card

- Export public key
- Cleanup
- Using keys
- Import public key
- Trust master key
- Insert YubiKey
- Encryption
- Decryption
- Signing
- Verifying signature
- SSH
- Create configuration
- Replace agents
- Copy public key
- (Optional) Save public key for identity file configuration
- Connect with public key authentication
- Touch to authenticate
- Import SSH keys
- Remote Machines (agent forwarding)
- GitHub
- OpenBSD
- Windows
  - WSL
  - Prerequisites
  - WSL configuration
  - Remote host configuration
  - Final test
- Troubleshooting
- Notes
- Links

Purchase YubiKey

All YubiKeys except the blue "security key" model are compatible with this guide. NEO models are limited to 2048-bit RSA keys. See Compare YubiKeys.

Consider purchasing a pair of YubiKeys, programming both, and storing one in a safe secondary location, in case of loss or damage to the first key.

Live image

It is recommended to generate cryptographic keys and configure YubiKey from a secure environment to minimize exposure. One way to do that is by downloading and booting to a Debian Live or Tails image loaded from a USB drive into memory.

Download the latest image and verify its integrity:

~~~
```console $ curl -LfO https://cdimage.debian.org/debian-cd/current-live/amd64/iso-hybrid/debian-live-9.7.0-amd64-xfce.iso $ curl -LfO https://cdimage.debian.org/debian-cd/current-live/amd64/iso-hybrid/SHA512SUMS $ curl -LfO https://cdimage.debian.org/debian-cd/current-live/amd64/iso-hybrid/SHA512SUMS.sign

$ gpg --verify SHA512SUMS.sign SHA512SUMS [...] gpg: Good signature from "Debian CD signing key debian-cd@lists.debian.org" [unknown] [...]

$ grep $(sha512sum debian-live-9.6.0-amd64-xfce.iso) SHA512SUMS
e35dd65fe1b078f71fcf04fa749a05bfefe4aa11a9e80f116ceec0566d65636a4ac84a
debian-live-9.6.0-amd64-xfce.iso ```
~~~

Mount a USB disk and copy the image over to it:

```
console $ sudo dd if=debian-live-9.6.0-amd64-xfce.iso of=/dev/sdc
bs=4M && sync
```

Shut down the computer and disconnect any hard drives and unnecessary peripherals.

Plug in the USB disk and boot to the live image. Configure networking to continue. If the screen locks, unlock with user/live.

Required software

Install several packages required for the following steps:

Debian/Ubuntu

```console $ sudo apt-get update

$ sudo apt-get install -y \ curl gnupg2 gnupg-agent \ cryptsetup scdaemon pcscd \ yubikey-personalization \ dirmngr \ secure-delete \ hopenpgp-tools ```

### Arch Linux

```
console $ sudo pacman -Syu gnupg2 pcsclite ccid yubikey-personalization hopenpgp-tools
```

RHEL7

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

OpenBSD

```
console $ doas pkg_add gnupg pcsc-tools
```

macOS

Download and install Homebrew and the following Brew packages:

```
console brew install gnupg yubikey-personalization hopenpgp-tools ykman pinentry-mac
```

Windows

Download and install Gpg4Win and PuTTY.

Note You may also need more recent versions of yubikey-personalization and yubico-c.
```

# Entropy

Generating keys will require a lot of randomness. To check the available bits of entropy available on Linux:

```
console $ cat /proc/sys/kernel/random/entropy_avail 849
```

Optional A hardware random number generator like OneRNG will increase the speed of entropy generation and possibly its quality. To install and configure OneRNG:

~~~
```console $ sudo apt-get install -y rng-tools at python-gnupg openssl

$ curl -LfO https://github.com/OneRNG/onerng.github.io/raw/master/sw/onerng_3.6-1_all.deb

$ sha256sum onerng_3.6-1_all.deb a9ccf7b04ee317dbfc91518542301e2d60ebe205d38e80563f29aac7cd845ccb

$ sudo dpkg -i onerng_3.6-1_all.deb

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

$ sudo service rng-tools restart ```
~~~

If the service fails to start, kick off `atd` and try again:

```
console $ sudo atd ; sudo service rng-tools restart
```

Plug in the OneRNG and empty `/dev/random` - the light on the device should dim briefly. Verify the available entropy pool is re-seeded.

~~~
```console $ cat /dev/random >/dev/null [Control-C]

$ cat /proc/sys/kernel/random/entropy_avail 3049 ```
~~~

An entropy pool value greater than 3000 is sufficient.

Creating keys

Create a temporary directory which will be deleted on reboot:

```
console $ export GNUPGHOME=$(mktemp -d) ; echo $GNUPGHOME
/tmp/tmp.aaiTTovYgo
```

Create a hardened configuration for GPG with the following options or by downloading drduh/config/gpg.conf:

```console $ curl -o $GNUPGHOME/gpg.conf https://raw.githubusercontent.com/drduh/config/master/gpg.conf

$ cat $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 fixed-list-mode no-comments no-emit-version keyid-format 0xlong list-options show-uid-validity verify-options show-uid-validity with-fingerprint require-cross-certification no-symkey-cache throw-keyids use-agent ```

Disable networking for the remainder of the setup.
```

Master key

The first key to generate is the master key. It will be used for certification only - to issue subkeys that are used for encryption, signing and authentication. This master key should be kept offline at all times and only accessed to revoke or issue new subkeys.

You'll be prompted to enter and verify a passphrase - keep it handy as you'll need it throughout. To generate a strong passphrase which could be written down in a hidden or secure place; or memorized:

```
console $ gpg --gen-random -a 0 24 ydOmByxmDe63u7gqx2XI9eDgpvJwibNH
```

Generate a new key with GPG, selecting `(8) RSA (set your own capabilities)`, `Certify`-only and `4096` bit keysize. Do not set the key to expire - see Note #3.

```console $ gpg --expert --full-generate-key

Please select what kind of key you want: (1) RSA and RSA (default) (2) DSA and Elgamal (3) DSA (sign only) (4) RSA (sign only) (7) DSA (set your own capabilities) (8) RSA (set your own capabilities) (9) ECC and ECC (10) ECC (sign only) (11) ECC (set your own capabilities) (13) Existing key Your selection? 8

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

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

Your selection? E

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

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

Your selection? S

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

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

Your selection? Q RSA keys may be between 1024 and 4096 bits long. What keysize do you want? (2048) 4096 Requested keysize is 4096 bits Please specify how long the key should be valid. 0 = key does not expire = key expires in n days w = key expires in n weeks m = key expires in n months y = key expires in n years Key is valid for? (0) 0 Key does not expire at all Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Dr Duh Email address: doc@duh.to Comment: [Optional - leave blank] You selected this USER-ID: "Dr Duh doc@duh.to"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy. gpg: /tmp.FLZC0xcM/trustdb.gpg: trustdb created gpg: key 0xFF3E7D88647EBCDB marked as ultimately trusted gpg: directory '/tmp.FLZC0xcM/openpgp-revocs.d' created gpg: revocation certificate stored as '/tmp.FLZC0xcM/openpgp-revocs.d/011CE16BD45B27A55BA8776DFF3E7D88647EBCDB.rev' public and secret key created and signed.

Note that this key cannot be used for encryption. You may want to use the command "--edit-key" to generate a subkey for this purpose. pub rsa4096/0xFF3E7D88647EBCDB 2017-10-09 [C] Key fingerprint = 011C E16B D45B 27A5 5BA8 776D FF3E 7D88 647E BCDB uid Dr Duh doc@duh.to ```

As of GPG version 2.1, a revocation certificate is automatically generated at this time.

Export the key ID as a variable (`KEYID`) for use later:

```
console $ export KEYID=0xFF3E7D88647EBCDB
```

Subkeys

Edit the Master key to add subkeys:

```console $ gpg --expert --edit-key $KEYID

Secret key is available.

sec rsa4096/0xEA5DE91459B80592 created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate [ultimate] (1). Dr Duh doc@duh.to ```

Use 4096-bit keysize - or 2048-bit on NEO.

Use a 1 year expiration - it can always be renewed using the offline Master certification key.
```

Signing

Create a signing key by selecting `(4) RSA (sign only)`:

```console gpg> addkey Key is protected.

You need a passphrase to unlock the secret key for user: "Dr Duh doc@duh.to" 4096-bit RSA key, ID 0xFF3E7D88647EBCDB, created 2016-05-24

Please select what kind of key you want: (3) DSA (sign only) (4) RSA (sign only) (5) Elgamal (encrypt only) (6) RSA (encrypt only) (7) DSA (set your own capabilities) (8) RSA (set your own capabilities) Your selection? 4 RSA keys may be between 1024 and 4096 bits long. What keysize do you want? (2048) 4096 Requested keysize is 4096 bits Please specify how long the key should be valid. 0 = key does not expire = key expires in n days w = key expires in n weeks m = key expires in n months y = key expires in n years Key is valid for? (0) 1y Key expires at Mon 10 Sep 2018 00:00:00 PM UTC Is this correct? (y/N) y Really create? (y/N) y We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate ssb rsa4096/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S [ultimate] (1). Dr Duh doc@duh.to ```
```

Encryption

Next, create an encryption key by selecting `(6) RSA (encrypt only)`:

```console gpg> addkey Please select what kind of key you want: (3) DSA (sign only) (4) RSA (sign only) (5) Elgamal (encrypt only) (6) RSA (encrypt only) (7) DSA (set your own capabilities) (8) RSA (set your own capabilities) (10) ECC (sign only) (11) ECC (set your own capabilities) (12) ECC (encrypt only) (13) Existing key Your selection? 6 RSA keys may be between 1024 and 4096 bits long. What keysize do you want? (2048) 4096 Requested keysize is 4096 bits Please specify how long the key should be valid. 0 = key does not expire = key expires in n days w = key expires in n weeks m = key expires in n months y = key expires in n years Key is valid for? (0) 1y Key expires at Mon 10 Sep 2018 00:00:00 PM UTC Is this correct? (y/N) y Really create? (y/N) y We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate ssb rsa4096/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S ssb rsa4096/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E [ultimate] (1). Dr Duh doc@duh.to ```
```

Authentication

Finally, create an authentication key.

GPG doesn't provide an authenticate-only key type, so select `(8) RSA (set your own capabilities)` and toggle the required capabilities until the only allowed action is `Authenticate`:

```console gpg> addkey Please select what kind of key you want: (3) DSA (sign only) (4) RSA (sign only) (5) Elgamal (encrypt only) (6) RSA (encrypt only) (7) DSA (set your own capabilities) (8) RSA (set your own capabilities) (10) ECC (sign only) (11) ECC (set your own capabilities) (12) ECC (encrypt only) (13) Existing key Your selection? 8

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

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

Your selection? S

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

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

Your selection? E

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

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

Your selection? A

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

actions: Authenticate

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

Your selection? Q RSA keys may be between 1024 and 4096 bits long. What keysize do you want? (2048) 4096 Requested keysize is 4096 bits Please specify how long the key should be valid. 0 = key does not expire = key expires in n days w = key expires in n weeks m = key expires in n months y = key expires in n years Key is valid for? (0) 1y Key expires at Mon 10 Sep 2018 00:00:00 PM UTC Is this correct? (y/N) y Really create? (y/N) y We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate ssb rsa4096/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S ssb rsa4096/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E ssb rsa4096/0x3F29127E79649A3D created: 2017-10-09 expires: 2018-10-09 usage: A [ultimate] (1). Dr Duh doc@duh.to

gpg> save ```

Verify keys

List the generated secret keys and verify the output:

```console $ gpg --list-secret-keys /tmp.FLZC0xcM/pubring.kbx

---

sec rsa4096/0xFF3E7D88647EBCDB 2017-10-09 [C] Key fingerprint = 011C E16B D45B 27A5 5BA8 776D FF3E 7D88 647E BCDB uid Dr Duh doc@duh.to ssb rsa4096/0xBECFA3C1AE191D15 2017-10-09 [S] [expires: 2018-10-09] ssb rsa4096/0x5912A795E90DD2CF 2017-10-09 [E] [expires: 2018-10-09] ssb rsa4096/0x3F29127E79649A3D 2017-10-09 [A] [expires: 2018-10-09] ```

**Optional** Add any additional identities or email addresses now using the `adduid` command.

To verify with OpenPGP key checks, use the automated key best practice checker:

```
console $ gpg --export $KEYID | hokey lint
```

The output will display any problems with your key in red text. If everything is green, your key passes each of the tests. If it is red, your key has failed one of the tests.

> hokey may warn (orange text) about cross certification for the authentication key. GPG's Signing Subkey Cross-Certification documentation has more detail on cross certification, and gpg v2.2.1 notes "subkey does not sign and so does not need to be cross-certified". hokey may also indicate a problem (red text) with `Key expiration times: []` on the primary key (see Note #3 about not setting an expiry for the primary key).
```

# Export keys

The Master and subkeys will be encrypted with your passphrase when exported.

Save a copy of your keys:

```console
$ gpg --armor --export-secret-keys $KEYID > $GNUPGHOME/mastersub.key

$ gpg --armor --export-secret-subkeys $KEYID > $GNUPGHOME/sub.key
```

On Windows, note that using any extension other than `.gpg` or attempting IO redirection to a file will garble the secret key, making it impossible to import it again at a later date:

```console
$ gpg --armor --export-secret-keys $KEYID -o \path\to\dir\mastersub.gpg

$ gpg --armor --export-secret-subkeys $KEYID -o \path\to\dir\sub.gpg
```

Backup keys

Once GPG keys are moved to YubiKey, they cannot be extracted again!

Make sure you have made an **encrypted** backup before proceeding. An encrypted USB drive or container can be made using VeraCrypt.

Also consider using a paper copy of the keys as an additional backup measure.

Linux

Attach a USB disk and check its label:

```
console $ sudo dmesg | tail scsi8 : usb-storage 2-1:1.0 usbcore:
registered new interface driver usb-storage scsi 8:0:0:0: USB 0: 0
ANSI: 6 sd 8:0:0:0: Attached scsi generic sg4 type 0 sd 8:0:0:0:
[sde] 62980096 512-byte logical blocks: (32.2 GB/30.0 GiB) sd
8:0:0:0: [sde] Write Protect is off sd 8:0:0:0: [sde] Mode Sense:
43 00 00 00 sd 8:0:0:0: [sde] Attached SCSI removable disk
```

Check the size to make sure it's the right device:

```
console $ sudo fdisk -l /dev/sde Disk /dev/sde: 30 GiB, 32245809152
bytes, 62980096 sectors /dev/sde1 2048 62980095 62978048 30G 6
FAT16
```

Erase and create a new partition table:

```console $ sudo fdisk /dev/sde Welcome to fdisk (util-linux 2.25.2).

Command (m for help): o Created a new DOS disklabel with disk identifier 0xeac7ee35.

Command (m for help): w The partition table has been altered. Calling ioctl() to re-read partition table. Syncing disks. ```

Remove and reinsert the USB drive, then create a new partition, selecting defaults:

```console $ sudo fdisk /dev/sde Welcome to fdisk (util-linux 2.25.2).

Command (m for help): n Partition type p primary (0 primary, 0 extended, 4 free) e extended (container for logical partitions) Select (default p): p Partition number (1-4, default 1): 1 First sector (2048-62980095, default 2048): Last sector, +sectors or +size{K,M,G,T,P} (2048-62980095, default 62980095):

Created a new partition 1 of type 'Linux' and of size 30 GiB.

Command (m for help): w The partition table has been altered. Calling ioctl() to re-read partition table. Syncing disks. ```

Use LUKS to encrypt the new partition:

~~~
```console $ sudo cryptsetup luksFormat /dev/sde1
~~~

# WARNING!

This will overwrite data on /dev/sde1 irrevocably.

Are you sure? (Type uppercase yes): YES Enter passphrase: Verify passphrase: ```

Mount the partition:

```
console $ sudo cryptsetup luksOpen /dev/sde1 usb Enter passphrase
for /dev/sde1:
```

Create a filesystem:

```console $ sudo mkfs.ext4 /dev/mapper/usb -L usb mke2fs 1.43.4 (31-Jan-2017) Creating filesystem with 7871744 4k blocks and 1970416 inodes Superblock backups stored on blocks: 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208, 4096000

Allocating group tables: done Writing inode tables: done Creating journal (32768 blocks): done Writing superblocks and filesystem accounting information: done ```

Mount the filesystem and copy the temporary GNUPG directory:

```console $ sudo mkdir /mnt/encrypted-usb

$ sudo mount /dev/mapper/usb /mnt/encrypted-usb

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

Keep the backup mounted if you plan on setting up two or more keys as `keytocard` **will delete the local copy** on save.

Otherwise, unmount and disconnected the encrypted USB disk:

```console $ sudo umount /mnt

$ sudo cryptsetup luksClose usb ```
```

OpenBSD

Attach a USB disk and determine its label:

```
console $ dmesg | grep sd.\ at sd2 at scsibus5 targ 1 lun 0:
<Samsung, Flash Drive DUO, 1100> SCSI4 0/direct removable
serial.50010000000000000001
```

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

```
console $ doas disklabel -h sd2
```

Initialize the disk by creating an `a` partition with FS type `RAID`:

```console $ doas fdisk -iy sd2 Writing MBR at offset 0.

$ doas disklabel -E sd2 Label editor (enter '?' for help at any prompt)

> a a offset: [64] size: [62653436] FS type: [4.2BSD] RAID w q No label changes.

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

Make an `i` partition, then make and mount the filesystem:

```console $ doas fdisk -iy sd3 Writing MBR at offset 0.

$ doas disklabel -E sd3 Label editor (enter '?' for help at any prompt)

> a i offset: [64] size: [62637371] FS type: [4.2BSD] w q No label changes.

$ doas newfs sd3i /dev/rsd3i: 30584.6MB in 62637344 sectors of 512 bytes 152 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each super-block backups (for fsck -b #) at: 32, 414688, 829344, 1244000, 1658656, 2073312, 2487968, 2902624, 3317280, 3731936, 4146592, 4561248, 4975904, [...] ```

Mount the filesystem and copy the temporary GNUPG directory:

```console $ doas mkdir /mnt/encrypted-usb
```

$ doas mount /dev/sd3i /mnt/encrypted-usb

$ doas cp -avi $GNUPGHOME /mnt/encrypted-usb ```

Keep the backup mounted if you plan on setting up two or more keys as `keytocard` **will delete the local copy** on save.

Otherwise, unmount and disconnected the encrypted USB disk:

```console $ doas umount /mnt/encrypted-usb

$ doas bioctl -d sd3 ```

See OpenBSD FAQ#14 for more information.
```

Configure YubiKey

Note YubiKey NEO shipped after November 2015 have all modes enabled; so this step may be skipped. Older versions of the YubiKey NEO may need to be reconfigured as a composite USB device (HID + CCID) which allows OTPs to be emitted while in use as a SmartCard. Plug in YubiKey and configure it with the `ykpersonalize` utility:

```console
$ sudo ykpersonalize -m82 Firmware version 4.3.7 Touch level 527 Program sequence 1

The USB mode will be set to: 0x82

Commit? (y/n) [n]: y
```

The -m option is the mode command. To see the different modes, enter `ykpersonalize -help`. Mode 82 (in hex) enables the YubiKey NEO as a composite USB device (HID + CCID). Once you have changed the mode, you need to re-boot the YubiKey, so remove and re-insert it. On YubiKey NEO with firmware version 3.3 or higher, you can enable composite USB device with `-m86` instead of `-m82`.

Windows Use the YubiKey NEO Manager to enable CCID functionality.

Configure Smartcard

Use GPG to configure YubiKey as a smartcard:

```
console $ gpg --card-edit Reader ...........: Yubico Yubikey 4 OTP
U2F CCID Application ID ...: D2760001240102010006055532110000
Version ..........: 2.1 Manufacturer .....: Yubico Serial number
....: 05553211 Name of cardholder: [not set] Language prefs ...:
[not set] Sex ..............: unspecified URL of public key : [not
set] Login data .......: [not set] Signature PIN ....: not forced
Key attributes ...: rsa2048 rsa2048 rsa2048 Max. PIN lengths .: 127
127 127 PIN retry counter : 3 0 3 Signature counter : 0 Signature
key ....: [none] Encryption key....: [none] Authentication key:
[none] General key info..: [none]
```

Change PIN

The default PIN is `123456` and default Admin PIN (PUK) is `12345678`. CCID-mode PINs can be up to 127 ASCII characters long.

The Admin PIN is required for some card operations and to unblock a PIN that has been entered incorrectly more than three times. See the GnuPG documentation on [Managing PINs]() for details.

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

gpg/card> passwd gpg: OpenPGP card no. D2760001240102010006055532110000 detected

1 - change PIN 2 - unblock PIN 3 - change Admin PIN 4 - set the Reset Code Q - quit

Your selection? 3 PIN changed.

1 - change PIN 2 - unblock PIN 3 - change Admin PIN 4 - set the Reset Code Q - quit

Your selection? 1 PIN changed.

1 - change PIN 2 - unblock PIN 3 - change Admin PIN 4 - set the Reset Code Q - quit

Your selection? q
```

Set information

Some fields are optional.

```console gpg/card> name Cardholder's surname: Duh Cardholder's given name: Dr

gpg/card> lang Language preferences: en

gpg/card> login Login data (account name): doc@duh.to

gpg/card> [Press Enter]

Application ID ...: D2760001240102010006055532110000 Version ..........: 2.1 Manufacturer .....: unknown Serial number ....: 05553211 Name of cardholder: Dr Duh Language prefs ...: en Sex ..............: unspecified URL of public key : [not set] Login data .......: doc@duh.to Private DO 4 .....: [not set] Signature PIN ....: not forced Key attributes ...: 2048R 2048R 2048R Max. PIN lengths .: 127 127 127 PIN retry counter : 3 0 3 Signature counter : 0 Signature key ....: [none] Encryption key....: [none] Authentication key: [none] General key info..: [none]

gpg/card> quit ```
```

Transfer keys

Important Transferring keys to YubiKey using `keytocard` is a destructive, one-way operation only. Make sure you've made a backup before proceeding: `keytocard` converts the local, on-disk key into a stub, which means the on-disk copy is no longer usable to transfer to subsequent security key devices or mint additional keys.

Previous GPG versions required the `toggle` command before selecting keys. The currently selected key(s) are indicated with an `*`. When moving keys only one key should be selected at a time.

~~~
```console $ gpg --edit-key $KEYID

Secret key is available.

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate ssb rsa4096/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S ssb rsa4096/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E ssb rsa4096/0x3F29127E79649A3D created: 2017-10-09 expires: 2018-10-09 usage: A [ultimate] (1). Dr Duh doc@duh.to ```
~~~

Signing

Select and move the signature key. You will be prompted for the key passphrase and Admin PIN.

```console
gpg> key 1

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate ssb* rsa4096/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S ssb rsa4096/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E ssb rsa4096/0x3F29127E79649A3D created: 2017-10-09 expires: 2018-10-09 usage: A [ultimate] (1). Dr Duh doc@duh.to

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

You need a passphrase to unlock the secret key for user: "Dr Duh doc@duh.to" 4096-bit RSA key, ID 0xBECFA3C1AE191D15, created 2016-05-24
```

Encryption

Type `key 1` again to de-select and `key 2` to select the next key:

```console gpg> key 1

gpg> key 2

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage: C trust: ultimate validity: ultimate ssb rsa4096/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S ssb* rsa4096/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E ssb rsa4096/0x3F29127E79649A3D created: 2017-10-09 expires: 2018-10-09 usage: A [ultimate] (1). Dr Duh doc@duh.to

gpg> keytocard Please select where to store the key: (2) Encryption key Your selection? 2

[...] ```
```

Authentication

Type `key 2` again to deselect and `key 3` to select the last key:

~~~
```console gpg> key 2

gpg> key 3

sec rsa4096/0xFF3E7D88647EBCDB created: 2017-10-09 expires: never usage:
C trust: ultimate validity: ultimate ssb rsa4096/0xBECFA3C1AE191D15
created: 2017-10-09 expires: 2018-10-09 usage: S ssb
rsa4096/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09
usage: E ssb* rsa4096/0x3F29127E79649A3D created: 2017-10-09 expires:
2018-10-09 usage: A [ultimate] (1). Dr Duh doc@duh.to

gpg> keytocard Please select where to store the key: (3) Authentication key
Your selection? 3

gpg> save ```
~~~

Verify card

Verify the subkeys have moved to YubiKey as indicated by `ssb>`:

```console $ gpg --list-secret-keys /tmp.FLZC0xcM/pubring.kbx

---

sec rsa4096/0xFF3E7D88647EBCDB 2017-10-09 [C] Key fingerprint = 011C E16B D45B 27A5 5BA8 776D FF3E 7D88 647E BCDB uid Dr Duh doc@duh.to ssb> rsa4096/0xBECFA3C1AE191D15 2017-10-09 [S] [expires: 2018-10-09] ssb> rsa4096/0x5912A795E90DD2CF 2017-10-09 [E] [expires: 2018-10-09] ssb> rsa4096/0x3F29127E79649A3D 2017-10-09 [A] [expires: 2018-10-09] ```
```

Export public key

Mount another USB disk to copy the *public* key, or save it somewhere where it can be easily accessed later.

Important Without importing the *public* key, you will not be able to use GPG to encrypt, decrypt, nor sign messages. However, you will still be able to use YubiKey for SSH authentication.

```
console $ gpg --armor --export $KEYID > /mnt/public-usb-key/pubkey.txt
```

Windows:

```
console $ gpg --armor --export $KEYID -o \path\to\dir\pubkey.gpg
```

Optional Upload the public key to a public keyserver:

```console $ gpg --send-key $KEYID

$ gpg --keyserver pgp.mit.edu --send-key $KEYID

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

After some time, the public key will to propagate to other servers.
```

Cleanup

Ensure you have:

- Saved the Encryption, Signing and Authentication subkeys to YubiKey.
- Saved the YubiKey PINs which you changed from defaults.
- Saved the password to the Master key.
- Saved a copy of the Master key, subkeys and revocation certificates on an encrypted volume stored offline.
- Saved the password to that encrypted volume in a separate location.
- Saved a copy of the public key somewhere easily accessible later.

Reboot or securely delete `$GNUPGHOME` and remove the secret keys from the GPG keyring:

```console $ sudo srm -r $GNUPGHOME || sudo rm -rf $GNUPGHOME

$ gpg --delete-secret-key $KEYID ```

**Important** Make sure you have securely erased all generated keys and revocation certificates if a Live image was not used!
```

Using keys

You can reboot back into the Live image to test YubiKey.

Install required programs:

```console
$ sudo apt-get update

$ sudo apt-get install -y \ curl gnupg2 gnupg-agent \ cryptsetup scdaemon pcscd
```

Download drduh/config/gpg.conf:

```console
$ mkdir ~/.gnupg ; curl -o ~/.gnupg/gpg.conf https://raw.githubusercontent.com/drduh/config/master/gpg.conf

$ chmod 600 ~/.gnupg/gpg.conf
```

Import public key

To import the public key from a file on an encrypted USB disk:

```console
$ sudo cryptsetup luksOpen /dev/sdd1 usb Enter passphrase for /dev/sdd1:

$ sudo mount /dev/mapper/usb /mnt

$ gpg --import /mnt/pubkey.txt gpg: key 0xFF3E7D88647EBCDB: public key "Dr Duh doc@duh.to" imported gpg: Total number processed: 1 gpg: imported: 1
```

To download the public key from a keyserver:

```
console $ gpg --recv $KEYID gpg: requesting key 0xFF3E7D88647EBCDB from hkps server hkps.pool.sks-keyservers.net [...] gpg: key 0xFF3E7D88647EBCDB: public key "Dr Duh <doc@duh.to>" imported gpg: Total number processed: 1 gpg: imported: 1
```

If you get the error `gpgkeys: HTTP fetch error 1: unsupported protocol` - this means you need to install a special version of curl which supports GPG:

```
console $ sudo apt-get install -y gnupg-curl
```

Trust master key

Edit the Master key to assign it ultimate trust by selecting `trust` then option `5`:

```console $ gpg --edit-key $KEYID

Secret key is available.

gpg> trust pub 4096R/0xFF3E7D88647EBCDB created: 2016-05-24 expires: never usage: C trust: unknown validity: unknown sub 4096R/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S sub 4096R/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E sub 4096R/0x3F29127E79649A3D created: 2017-10-09 expires: 2018-10-09 usage: A [ unknown] (1). Dr Duh doc@duh.to

Please decide how far you trust this user to correctly verify other users' keys (by looking at passports, checking fingerprints from different sources, etc.)

1 = I don't know or won't say 2 = I do NOT trust 3 = I trust marginally 4 = I trust fully 5 = I trust ultimately m = back to the main menu

Your decision? 5 Do you really want to set this key to ultimate trust? (y/N) y

pub 4096R/0xFF3E7D88647EBCDB created: 2016-05-24 expires: never usage: C trust: ultimate validity: unknown sub 4096R/0xBECFA3C1AE191D15 created: 2017-10-09 expires: 2018-10-09 usage: S sub 4096R/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09 usage: E sub 4096R/0x3F29127E79649A3D created: 2017-10-09 expires: 2018-10-09 usage: A [ unknown] (1). Dr Duh doc@duh.to

gpg> save ```
```

Insert YubiKey

Re-connect YubiKey and check the status:

```
console $ gpg --card-status Application ID ...:
D2760001240102010006055532110000 Version ..........: 2.1
Manufacturer .....: Yubico Serial number ....: 05553211 Name of
cardholder: Dr Duh Language prefs ...: en Sex ..............:
unspecified URL of public key : [not set] Login data .......:
doc@duh.to Signature PIN ....: not forced Key attributes ...: 4096R
4096R 4096R Max. PIN lengths .: 127 127 127 PIN retry counter : 3 3
3 Signature counter : 0 Signature key ....: 07AA 7735 E502 C5EB
E09E B8B0 BECF A3C1 AE19 1D15 created ....: 2016-05-24 23:22:01
Encryption key....: 6F26 6F46 845B BEB8 BDF3 7E9B 5912 A795 E90D
D2CF created ....: 2016-05-24 23:29:03 Authentication key: 82BE
7837 6A3F 2E7B E556 5E35 3F29 127E 7964 9A3D created ....: 2016-05-
24 23:36:40 General key info..: pub 4096R/0xBECFA3C1AE191D15 2016-
05-24 Dr Duh <doc@duh.to> sec# 4096R/0xFF3E7D88647EBCDB created:
2016-05-24 expires: never ssb> 4096R/0xBECFA3C1AE191D15 created:
2017-10-09 expires: 2018-10-09 card-no: 0006 05553211 ssb>
4096R/0x5912A795E90DD2CF created: 2017-10-09 expires: 2018-10-09
card-no: 0006 05553211 ssb> 4096R/0x3F29127E79649A3D created: 2017-
10-09 expires: 2018-10-09 card-no: 0006 05553211
```

`sec#` indicates master key is not available (as it should be stored encrypted offline).

Note If you see `General key info..: [none]` in the output instead - go back and import the public key using the previous step.

Encryption

```console
$ echo "test message string" | gpg --encrypt --armor --recipient $KEYID -----BEGIN PGP MESSAGE-----

hQIMA1kSp5XpDdLPAQ/+JyYfLaUS/+llEzQaKDb5mWhG4HlUgD99dNJUX
PSSt3I8Ac0ctwyMnenZvBEbHMqdRnfZJsj5pHidKcAZrhgs+he+B1tdZ/KPa8in
NIGqd8W1OraVSFmPEdC1kQ5he6R/WCDH1NNel9+fvLtQDCBQaFae/s3yX
HKCJLyHK8K9hDvgFmXOY8j1qTknBvDbmYdcCKVE1ejgpUCi3Watusobp
6DN8bXyfxLPYm1PTLfW7v4kwddktB8eVioV8A45lndJZvliSqDwxhrwyE5V
NmqzBkCaOHQFr0ofL91xgwpCI5kM2ukIR5SxUO4hvzlHn58QVL9GfAyCH
Q9eiR0joo9TjTwR8XomVhRJShrrcPeGgu3YmIak4u7OndyBFpu2E79RQ0ehpl
tSECB6mNd/gt0Wy3y15ccaFI4CVP6jrMN6q3YhXqNC7GgI/OWkVZIAgUFY
tQ3z3wlbvFFngeFy5IlhsPduK8T9XgPnOtgQxHaepKz0h3m2lJegmp4YZ4CbS9
kcBTUjys5Vin1SLuqL4PhErzmlAZgVzG2PANsnHYPe2hwN4NlFtOND1wgB
1pqz1I0O+jmyId+jVlAK076c2AwdkVbokKUcIT/OcTc0nwHjOUttJGmkUHlbt/
iAFNniSfzf6fwAFHgsvWiRJMa3keolPiqoUdh0tBIiI1zxOMaiTL7C9BFdpnvz
Krj0pDc7AlF4spWhm58WgAW20P8PGcVQcN6mSTG8jKbXVSP3bvgPXkpG
pLORcRPbauusBqovgaBWU/i3pMYrbhZ+LQbVEaJlvblWu6xe8HhS/jo=
=pzkv -----END PGP MESSAGE-----
```

To encrypt to multiple recipients (or to multiple keys):

```
console $ echo "test message string" | gpg --encrypt --armor --
recipient $KEYID_0 --recipient $KEYID_1 --recipient $KEYID_2 -----
BEGIN PGP MESSAGE----- [...]
```

Decryption

```console $ gpg --decrypt --armor -----BEGIN PGP MESSAGE-----

hQIMA1kSp5XpDdLPAQ/+JyYfLaUS/+llEzQaKDb5mWhG4HlUgD99dNJUX
PSSt3I8Ac0ctwyMnenZvBEbHMqdRnfZJsj5pHidKcAZrhgs+he+B1tdZ/KPa8in
NIGqd8W1OraVSFmPEdC1kQ5he6R/WCDH1NNel9+fvLtQDCBQaFae/s3yX
HKCJLyHK8K9hDvgFmXOY8j1qTknBvDbmYdcCKVE1ejgpUCi3WatusobpW
6DN8bXyfxLPYm1PTLfW7v4kwddktB8eVioV8A45lndJZvliSqDwxhrwyE5V
NmqzBkCaOHQFr0ofL91xgwpCI5kM2ukIR5SxUO4hvzlHn58QVL9GfAyCH
Q9eiR0joo9TjTwR8XomVhRJShrrcPeGgu3YmIak4u7OndyBFpu2E79RQ0ehpl
tSECB6mNd/gt0Wy3y15ccaFI4CVP6jrMN6q3YhXqNC7GgI/OWkVZIAgUFY
tQ3z3wlbvFFngeFy5IlhsPduK8T9XgPnOtgQxHaepKz0h3m2lJegmp4YZ4CbS9
kcBTUjys5Vin1SLuqL4PhErzmlAZgVzG2PANsnHYPe2hwN4NlFtOND1wgB
1pqz1I0O+jmyId+jVlAK076c2AwdkVbokKUcIT/OcTc0nwHjOUttJGmkUHlbt/
iAFNniSfzf6fwAFHgsvWiRJMa3keolPiqoUdh0tBIiI1zxOMaiTL7C9BFdpnvzY
Krj0pDc7AlF4spWhm58WgAW20P8PGcVQcN6mSTG8jKbXVSP3bvgPXkpG
pLORcRPbauusBqovgaBWU/i3pMYrbhZ+LQbVEaJlvblWu6xe8HhS/jo=
=pzkv -----END PGP MESSAGE----- gpg: encrypted with 4096-bit RSA key, ID
0x5912A795E90DD2CF, created 2016-05-24 "Dr Duh doc@duh.to"

[Press Control-D]

test message string ```
```

Signing

~~~
```console $ echo "test message string" | gpg --armor --clearsign --default-key
0xBECFA3C1AE191D15 -----BEGIN PGP SIGNED MESSAGE----- Hash:
SHA512

test message string -----BEGIN PGP SIGNATURE-----

iQIcBAEBCgAGBQJXRPo8AAoJEL7Po8GuGR0Vh8wP/jYXTR8SAZIZSMV
k6JxB0rF928WDYPihjo/d0Jd+XpoV1g+oipDRjP78xqR9H/CJZlE10IPQbNaom
+3RGxA3Zr085cVFoixI8rxYOSu0Vs2cAzAbJHNcOcD7vXxTHcX4T8kfKoF9
XTJ42eEjpO0fX76tFX2/Uzxl43ES0dO7Y82ho7xcnaYwakVUEcWfUpfDAroL
wCZGr8Z64QDQzxQ9L45Zc61wMx9JEIWD4Bnagllfe0YrEwTJfYG8uhDDN
j1PBHn5d556aX6DHUH05kq3wszvQ4W40RctLgAA3l1VnEKebhBKjLZA/Ee
QM7MFUV1X/pi2zlyoZSnHkVl8b5Q7RU5ZtRpq9fdkDDepeiUo5PNBUMJER
ri8DtavkwTNWBRLnVR2gHBmVQNN7ZDOkHcfyqR4I9chx6TMpfcxk0zAT
FVPKySifuXpunn+0MwdZl5XkhHGdpdYQz4/LAZUGhrA9JTnFtc4cl4JrTzufF
c3JJumMsyGvw9OQKQHF8gHme4PBu/4P31LpfX9wzPOTpJaI31Sg5kdJLTo9
uvkmJS7ETjLQZOsRyAEn7gcEKZQGPQcNAgfEgQPoepS/KvvI68u+JMJm4n
fEkp501u8kAZkWauhiL+ =+ylJ -----END PGP SIGNATURE----- ```
~~~

Verifying signature

```console $ gpg gpg: Go ahead and type your message ... -----BEGIN PGP SIGNED MESSAGE----- Hash: SHA512

test message string -----BEGIN PGP SIGNATURE-----

```
iQIcBAEBCgAGBQJXRPo8AAoJEL7Po8GuGR0Vh8wP/jYXTR8SAZIZSMV
+3RGxA3Zr085cVFoixI8rxYOSu0Vs2cAzAbJHNcOcD7vXxTHcX4T8kfKoF9
XTJ42eEjpO0fX76tFX2/Uzxl43ES0dO7Y82ho7xcnaYwakVUEcWfUpfDAroL
wCZGr8Z64QDQzxQ9L45Zc61wMx9JEIWD4BnagllfeOYrEwTJfYG8uhDDN
j1PBHn5d556aX6DHUH05kq3wszvQ4W40RctLgAA3l1VnEKebhBKjLZA/Ee
QM7MFUV1X/pi2zlyoZSnHkVl8b5Q7RU5ZtRpq9fdkDDepeiUo5PNBUMJER
ri8DtavkwTNWBRLnVR2gHBmVQNN7ZDOkHcfyqR4I9chx6TMpfcxk0zATA
FVPKySifuXpunn+0MwdZl5XkhHGdpdYQz4/LAZUGhrA9JTnFtc4cl4JrTzufF
c3JJumMsyGvw9OQKQHF8gHme4PBu/4P31LpfX9wzPOTpJaI31Sg5kdJLTo9
uvkmJS7ETjLQZOsRyAEn7gcEKZQGPQcNAgfEgQPoepS/KvvI68u+JMJm4n
fEkp501u8kAZkWauhiL+ =+ylJ -----END PGP SIGNATURE-----
```

[Press Control-D]

gpg: Signature made Wed 25 May 2016 00:00:00 AM UTC gpg: using RSA key 0xBECFA3C1AE191D15 gpg: Good signature from "Dr Duh doc@duh.to" [ultimate] Primary key fingerprint: 011C E16B D45B 27A5 5BA8 776D FF3E 7D88 647E BCDB Subkey fingerprint: 07AA 7735 E502 C5EB E09E B8B0 BECF A3C1 AE19 1D15 ```
```

# SSH

gpg-agent supports the OpenSSH ssh-agent protocol (`enable-ssh-support`), as well as Putty's Pageant on Windows (`enable-putty-support`). This means it can be used instead of the traditional ssh-agent / pageant. There are some differences from ssh-agent, notably that gpg-agent does not *cache* keys rather it converts, encrypts and stores them - persistently - as GPG keys and then makes them available to ssh clients. Any existing ssh private keys that you'd like to keep in `gpg-agent` should be deleted after they've been imported to the GPG agent.

When importing the key to `gpg-agent`, you'll be prompted for a passphrase to protect that key within GPG's key store - you may want to use the same passphrase as the original's ssh version. GPG can both cache passphrases for a determined period (ref. `gpg-agent`'s various `cache-ttl` options), and since version 2.1 can store and fetch passphrases via the macOS keychain. Note than when removing the old private key after importing to `gpg-agent`, keep the `.pub` key file around for use in specifying ssh identities (e.g. `ssh -i /path/to/identity.pub`).

Probably the biggest thing missing from `gpg-agent`'s ssh agent support is being able to remove keys. `ssh-add -d/-D` have no effect. Instead, you need to use the `gpg-connect-agent` utility to lookup a key's keygrip, match that with the desired ssh key fingerprint (as an MD5) and then delete that keygrip. The gnupg-users mailing list has more information.

# Create configuration

Create a hardened configuration for gpg-agent by downloading drduh/config/gpg-agent.conf:

```console $ curl -o ~/.gnupg/gpg-agent.conf https://raw.githubusercontent.com/drduh/config/master/gpg-agent.conf

$ cat ~/.gnupg/gpg-agent.conf enable-ssh-support pinentry-program /usr/bin/pinentry-curses default-cache-ttl 60 max-cache-ttl 120 ```

Alternatively, you may want to use `/usr/bin/pinentry-gnome3` to use a GUI manager. On macOS, use `brew install pinentry-mac` and adjust the program path to suit.
```

Replace agents

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

Add these to the shell `rc` file:

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

On some systems, you may need to use the following instead:

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

Copy public key

Note It is *not* necessary to import the corresponding GPG public key in order to use SSH.

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

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

(Optional) Save 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 the YubiKey - indeed, in general for keys stored in an ssh agent - `IdentityFile` should point to the *public* key file, `ssh` will select the appropriate private key from those available via the ssh agent. To prevent `ssh` from trying all keys in the agent use the `IdentitiesOnly yes` option along with one or more `-i` or `IdentityFile` options for the target host.

To reiterate, with `IdentitiesOnly yes`, `ssh` will not automatically enumerate public keys loaded into `ssh-agent` or `gpg-agent`. This means `publickey` 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:

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

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

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

Connect with public key authentication

```
console $ 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). [...]
```

Note To make multiple connections or securely transfer many files, consider using the ControlMaster ssh option. Also see drduh/config/ssh_config.

Touch to authenticate

Note This is not possible on YubiKey NEO.

By default, YubiKey will perform key operations without requiring a touch from the user. To require a touch for every SSH authentication, use the YubiKey Manager and Admin PIN:

```
console $ ykman openpgp touch aut on
```

To require a touch for signing and encryption operations:

```console $ ykman openpgp touch sig on

$ ykman openpgp touch enc on ```

The YubiKey will blink when it's waiting for touch.
```

Import SSH keys

If there are existing SSH keys that you wish to make available via `gpg-agent`, you'll need to import them. You should then remove the original private keys. When importing the key, `gpg-agent` uses the key's filename as the key's label; this makes it easier to follow where the key originated from. In this example, we're starting with just the YubiKey's key in place and importing `~/.ssh/id_rsa`:

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

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

When invoking `ssh-add`, it will prompt for the SSH key's passphrase if present, then the `pinentry` program will prompt and confirm for a new passphrase to use to encrypt the converted key within the GPG key store.

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

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

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

```
console $ 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's passphrase. The passphrase will be cached for up to 10 minutes idle time between uses, to a maximum of 2 hours.

Remote Machines (agent forwarding)

If you want to use YubiKey to sign a git commit on a remote machine, or ssh through another layer, then this is possible using "Agent Forwarding". This section should help you setup GPG and SSH agent forwarding.

To do this, you need to already have shell access to the remote machine, and the YubiKey setup on the host machine.

- First, on the local machine, run:

```
console $ 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`.

- Next, find the agent socket on the **remote** machine:

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

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

- On the remote machine, edit the file `/etc/ssh/sshd_config`, so that option `StreamLocalBindUnlink` is set to `StreamLocalBindUnlink yes`

- **Optional** If you do not have root access to the remote machine to edit `/etc/ssh/sshd_config`, you will need to remove the socket on the remote machine before forwarding works. For example, `rm /run/user/1000/gnupg/S.gpg-agent`. Further information can be found on the AgentForwarding GNUPG wiki page.

- Import public keys to the remote machine. This can be done by fetching from a keyserver. On the local machine, copy the public keyring to the remote machine:

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

- Finally, to enable agent forwarding for a given machine, add the following to the local machine's ssh config file `~/.ssh/config` (your agent sockets may be different):

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

You should then be able to use YubiKey as if it were connected to the remote machine.

If you're still having problems, it may be necessary to edit `gpg-agent.conf` file on both the remote and local machines to add the following information:

```
enable-ssh-support pinentry-program /usr/bin/pinentry-curses extra-
socket /run/user/1000/gnupg/S.gpg-agent.extra
```

GitHub

You can use YubiKey to sign GitHub commits and tags. It can also be used for GitHub SSH authentication, allowing you to push, pull, and commit without a password.

Login to GitHub and upload SSH and PGP public keys in Settings.

To configure a signing key:

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

Make sure the user.email option matches the email address associated with the PGP identity.

Now, to sign commits or tags simply use the `-S` option. GPG will automatically query YubiKey and prompt you for a PIN.

To authenticate:

Windows Run the following command:

```
> git config --global core.sshcommand 'plink -agent'
```

You can then change the repository url to `git@github.com:USERNAME/repository` and any authenticated commands will be authorized by YubiKey.

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

OpenBSD

`doas pkg_add pcsc-tools` and enable with `doas rcctl enable pcscd`, then reboot in order to recognize YubiKey.

Windows

Windows can already have some virtual smartcard readers installed, like the one provided for Windows Hello. To ensure your YubiKey is the correct one used by scdaemon, you should add it to its configuration. You will need your device's full name. To find out what is your device's full name, plug your YubiKey, open the Device Manager, select "View->Show hidden devices". Go to the Software Devices list, you should see something like `Yubico YubiKey OTP+FIDO+CCID 0`. The name slightly differs according to the model. Thanks to Scott Hanselman for sharing this information.

- Create or edit %APPDATA%/gnupg/scdaemon.conf, add `reader-port <your yubikey device's full name>`.
- In %APPDATA%/gnupg/gpg-agent.conf, add:

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

- Open a command console, restart the agent:

\`\`\`

  gpg-connect-agent killagent /bye gpg-connect-agent /bye \`\`\`

- Enter `> gpg --card-status` to see YubiKey details.
- Import the public key: `> gpg --import <path to public key file>`
- Trust it: Trust master key
- Retrieve the public key id: `> gpg --list-public-keys`
- Export the SSH key from GPG: `> gpg --export-ssh-key <public key id>`

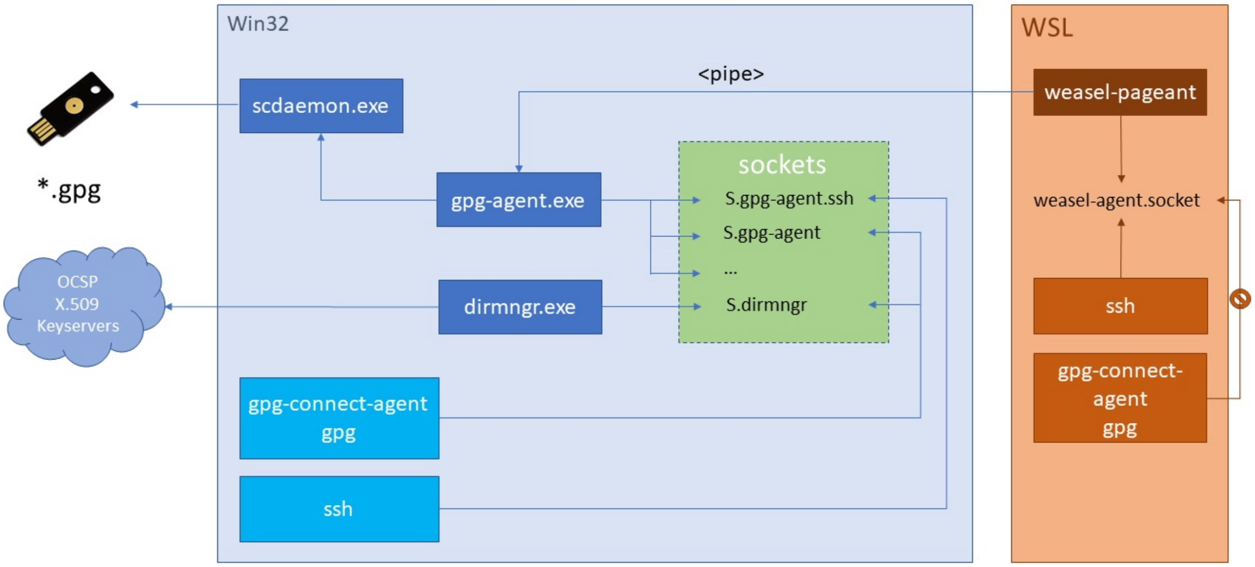
Copy this key to a file for later use. It represents the public SSH key corresponding to the secret key on the YubiKey. You can upload this key to any server you wish to SSH into.

- 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 a system shutdown. Modify the shortcut properties so it starts in a "Minimized" window, to avoid unnecessary noise at startup.

Now you can use PuTTY for public key SSH authentication. When the server asks for public key verification, PuTTY will forward the request to GPG, which will prompt you for a PIN and authorize the login using YubiKey.

WSL

The goal here is to make the SSH client inside WSL work together with the Windows agent you are using (gpg-agent.exe in our case). Here is what we are going to achieve:



Note this works only for SSH agent forwarding. Real GPG forwarding (encryption/decryption) is actually not supported. See the weasel-pageant readme for further information.

Prerequisites

- Ubuntu >16.04 for WSL
- Kleopatra
- Windows configuration

WSL configuration

- Download or clone weasel-pageant.
- Add `eval $(/mnt/c/<path of extraction>/weasel-pageant -r -a /tmp/S.weasel-pageant)` to shell rc file. Use a named socket here so it can be used in the RemoteForward directive of the .ssh/config file.
- Source it with `source ~/.bashrc`.
- Display the SSH key with `$ ssh-add -l`.
- Edit `~/.ssh/config` - for each host you want to use agent forwarding, add:

```
ForwardAgent yes RemoteForward <remote ssh socket path>
/tmp/S.weasel-pageant
```

Note The remote ssh socket path can be found by executing `$ gpgconf --list-dirs agent-ssh-socket` on the host.

Remote host configuration

Add the following to the shell rc file:

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

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

```
AllowAgentForwarding yes StreamLocalBindUnlink yes
```

And reload the SSH daemon (e.g., `sudo service sshd reload`).

Final test

- Unplug YubiKey, disconnect or reboot.
- Log back in to Windows, open a WSL console and enter `ssh-add -l` - you should see nothing.
- Plug in YubiKey, enter the same command to display the ssh key.
- Log in to the remote host, you should have the pinentry dialog asking for the YubiKey pin.
- On the remote host, type `ssh-add -l` - if you see the ssh key, that means forwarding works!

Note Agent forwarding may be chained through multiple hosts - just follow the same protocol to configure each host.

Troubleshooting

- If you don't understand some option - read `man gpg`.

- If you encounter problems connecting to YubiKey with GPG - try unplugging and re-inserting YubiKey, and restarting the `gpg-agent` process.

- If you receive the error, `gpg: decryption failed: secret key not available` - you likely need to install GnuPG version 2.x.

- 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 Required software.

- 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` - check `~/.gnupg/gpg-agent.conf` to make sure the path to `pinentry` is correct.

- 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"`

- If you receive the error, `Permission denied (publickey)`, increase ssh verbosity with the `-v` flag and ensure the public key from the card is being offered: `Offering public key: RSA SHA256:abcdefg... cardno:00060123456`. If it is, ensure you are connecting as the right user on the target system, rather than as the user on the local system. Otherwise, be sure `IdentitiesOnly` is not enabled for this host.

- If SSH authentication stil fails - add up to 3 `-v` flags to increase verbosity.

- If you totally screw up, you can reset the card.

Notes

1. YubiKey has two configurations: one invoked with a short press, and the other with a long press. By default, the short-press mode is configured for HID OTP - a brief touch will emit an OTP string starting with `cccccccc`. If you rarely use the OTP mode, you can swap it to the second configuration via the YubiKey Personalization tool. If you *never* use OTP, you can disable it entirely using the YubiKey Manager application (note, this not the similarly named YubiKey NEO Manager).
2. Programming YubiKey for GPG keys still lets you use its two configurations - OTP and static password modes, for example.
3. Setting an expiry essentially forces you to manage your subkeys and announces to the rest of the world that you are doing so. Setting an expiry on a primary key is ineffective for protecting the key from loss - whoever has the primary key can simply extend its expiry period. Revocation certificates are better suited for this purpose. It may be appropriate for your use case to set expiry dates on subkeys.
4. To switch between two or more identities on different keys - unplug the first key and restart gpg-agent, ssh-agent and pinentry with `pkill gpg-agent ; pkill ssh-agent ; pkill pinentry ; eval $(gpg-agent --daemon --enable-ssh-support)`, then plug in the other key and run `gpg-connect-agent updatestartuptty /bye` - then it should be ready for use.

Links

- https://alexcabal.com/creating-the-perfect-gpg-keypair/
- https://blog.habets.se/2013/02/GPG-and-SSH-with-Yubikey-NEO
- https://blog.josefsson.org/2014/06/23/offline-gnupg-master-key-and-subkeys-on-yubikey-neo-smartcard/
- https://developers.yubico.com/PGP/Card_edit.html
- https://developers.yubico.com/PIV/Introduction/Admin_access.html
- https://developers.yubico.com/yubico-piv-tool/YubiKey_PIV_introduction.html
- https://developers.yubico.com/yubikey-personalization/
- https://developers.yubico.com/yubikey-piv-manager/PIN_and_Management_Key.html
- https://evilmartians.com/chronicles/stick-with-security-yubikey-ssh-gnupg-macos
- https://gist.github.com/ageis/14adc308087859e199912b4c79c4aaa4
- https://github.com/herlo/ssh-gpg-smartcard-config
- https://github.com/tomlowenthal/documentation/blob/master/gpg/smartcard-keygen.md
- https://help.riseup.net/en/security/message-security/openpgp/best-practices
- https://jclement.ca/articles/2015/gpg-smartcard/
- https://rnorth.org/gpg-and-ssh-with-yubikey-for-mac
- https://trmm.net/Yubikey
- https://www.bootc.net/archives/2013/06/09/my-perfect-gnupg-ssh-agent-setup/
- https://www.esev.com/blog/post/2015-01-pgp-ssh-key-on-yubikey-neo/
- https://www.hanselman.com/blog/HowToSetupSignedGitCommitsWithAYu
- https://www.void.gr/kargig/blog/2013/12/02/creating-a-new-gpg-key-with-subkeys/
- https://mlohr.com/gpg-agent-forwarding/