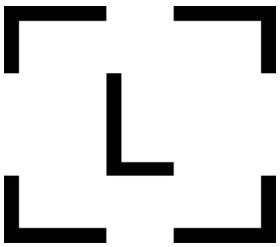
OpenPGP Card Application

User Guide

Ledger SAS



https://github.com/LedgerHQ/app-openpgp

March 14, 2024

Contents

1 License

2 Introduction

GnuPG application for Ledger devices.

This application implements "The OpenPGP card" specification revision 3.3. This specification is available in doc directory at [G10CODE].

The application supports:

- RSA with key up to 3072 bits
- ECDSA with secp256R1 and secp256K1
- EDDSA with Ed25519 curve
- ECDH with secp256R1, secp256K1 and curve25519 curves

This release has known missing parts (see also [GPGADD]):

- Seed mode ON/OFF via apdu

3 How to install GPG Application

3.1 System Configuration

You need to install CCID. For Linux and MAC, the Ledger CCID interface is not supported by default by `pcscd` and must be manually added.

3.1.1 Linux

You have to add your devices to `/etc/libccid_Info.plist`

3.1.2 MAC

1. First it is necessary to disable SIP, that forbid editing files in `/usr/`.
2. You have to add your devices to `/usr/libexec/SmartCardServices/drivers/ifd-ccid.bundle/Contents/Info.p`
3. Enable SIP

Note: See https://developer.apple.com/library/content/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html

3.1.3 Windows

TODO...

3.1.4 Manual update of CCID

In case the devices ids are not set or not correct, please update the `Info.plist` file manually. Remember there are 3 important nodes in this xml file, and the lines must be coherent between those nodes:

- <key>ifdVendorID</key>
- <key>ifdProductID</key>
- <key>ifdFriendlyName</key>

Thus, you must ensure (or add):

- For Nanos:
  - ifdVendorID: 0x2C97

  - ifdProductID: 0x1009
  - ifdFriendlyName: Ledger Nano S
- For Nanox:
  - ifdVendorID: 0x2C97
  - ifdProductID: 0x4009
  - ifdFriendlyName: Ledger Nano X
- For Nanos+:
  - ifdVendorID: 0x2C97
  - ifdProductID: 0x5009
  - ifdFriendlyName: Ledger Nano S Plus
- for Stax:
  - ifdVendorID: 0x2C97
  - ifdProductID: 0x6009
  - ifdFriendlyName: Ledger Stax

Notes:

- The 3 entry nodes must be added for each device. It can be easier to add new ones at the end of each list.
- A file 0001-plist.patch is provided in this directory.

4 OpenPGP Card application explained

4.1 Menu Overview

The full menu layout is:

Select Slot
Choose:
Slot 1 #+
Slot 2
Slot 3
Set Default slot
Settings
Key Template
Choose Key...
Signature
Decryption
Authentication
Choose Type...
RSA 2048
RSA 3072
SECP 256R1
ED25519
Set Template
Seed mode *ON/OFF*
PIN mode
Choose:
On Screen
Confirm only #+
Trust
Set as Default
UIF mode
UIF for Signature *ON/OFF*
UIF for Decryption *ON/OFF*
UIF for Authentication *ON/OFF*
Reset
About
OpenPGP Card

(c) Ledger SAS
Spec 3.3.1
App 1.5.4

Emphasis entries are not selectable and just provide information.
A "#" after the entry label means default value on reset.
A "+" after the entry label means current value.

4.2 Device Info

The *Device Info* provides current user and slot information. The format is:

```
<User: **name** / Serial: **s** / Slot: **n** >
```

with:

- **name** is the one provided to `gpg --card-edit`. See [GPGSC].
- **s** is the 32 bits card serial number. Note that the last three bits always encode the current slot value.
- **n** is the current slot, see below.

4.3 Select Slot

For Nanos, this menu is only available on *XL* version. It is available on all other devices.

A Slot is a set of 3 key pairs *Signature, Decryption, Authentication* as defined by gnupg specification.

Usually a GPG card application only manages a single set. Ledger version enhances this and allows you to manage 3 key sets.

The *Select Slot* menu allows you to select the slot you want to play with, and to set the default slot when the application start.

To change the current slot, display the slot you want and select it

To change the default slot, first select it, and then select the *Set Default* entry.

4.4 Settings

4.4.1 Key Template

A key template is defined by the OpenGPG card application specification. It describes the key to be generated with the `generate` command in `gpg --card-edit`

To set up a new ECC template you have three choices:

- The `gpg --edit-card` interactive setup (recommended)
- The `gpg-connect-agent` tool
- The device menu.

gpg --card-edit

This method suppose you have a recent GnuPG tool and that you correctly configured it. See the dedicated section for that.

In a terminal launch:

```
$ gpg --card-edit
gpg/card> admin
Admin commands are allowed

gpg/card> key-attr
Changing card key attribute for: Signature key
Please select what kind of key you want:
  (1) RSA
  (2) ECC
```

```
Your selection? 2
Please select which elliptic curve you want:
  (1) Curve 25519
  (4) NIST P-384
Your selection? 1
The card will now be re-configured to generate a key of type: ed25519
Note: There is no guarantee that the card supports the requested size.
    If the key generation does not succeed, please check the
    documentation of your card to see what sizes are allowed.
Changing card key attribute for: Encryption key
Please select what kind of key you want:
  (1) RSA
  (2) ECC
Your selection? 2
Please select which elliptic curve you want:
  (1) Curve 25519
  (4) NIST P-384
Your selection? 1
The card will now be re-configured to generate a key of type: cv25519
Changing card key attribute for: Authentication key
Please select what kind of key you want:
  (1) RSA
  (2) ECC
Your selection? 2
Please select which elliptic curve you want:
  (1) Curve 25519
  (4) NIST P-384
Your selection? 1
The card will now be re-configured to generate a key of type: ed25519
```

To show the current template use the `gpg --card-status` command.

gpg-connect-agent

This method suppose you have correctly configured your GnuPG tool. See the dedicated section for that.

In a terminal launch:

```
gpg-connect-agent "SCD SETATTR KEY-ATTR --force 1 <tag> <curvename>" /bye
gpg-connect-agent "SCD SETATTR KEY-ATTR --force 2 18    <curvename>" /bye
gpg-connect-agent "SCD SETATTR KEY-ATTR --force 3 <tag> <curvename>" /bye
```

This 3 commands fix, in that order, the template for Signature, Decryption, Authentication keys.

Supported curve name are:

- secp256k1 with tag 19
- secp256r1 with tag 19
- nistp256 with tag 19
- cv25519 (only for key 2)
- ed25519 with tag 22 (only for key 1 and 3)

To show the current template use the `gpg --card-status` command.

Device menu

First under *Choose Key* menu, select the one of 3 keys for which you want to modify the template. Then under "Choose Type", select the desired key template. Finally select "Set Template" entry to set it.

To show the current template use the `gpg --card-status` command.

4.4.2 Seed mode

When generating new keys on the device, those keys can be generated randomly or in a deterministic way. The deterministic way is specified in [GPGADD]. The current mode is displayed in the first sub menu. To activate the seed mode select *ON*, to deactivate the seed mode select *OFF*.

When the application starts, the seed mode is always set to *ON*

4.4.3 PIN mode

Some operations require the user to enter his PIN code. The default PIN values are:

- user: `123456`
- admin: `12345678`

The PIN entry can be done using 3 methods, named *On Screen*, *Confirm only*, *Trust*.

After each mode a + or # symbol may appear to tell which mode is the current one and which one is the default when the application starts. The default mode can be changed by first selecting the desired mode and then selecting the *Set default* menu.

Note: *Trust* can not be set as default mode.

Before you can change the PIN mode, you need to verify the PIN on the client. To do this, run `gpg --card-edit`, then `admin` and finally `verify` on you PC. You will then be asked to enter the current PIN. After doing so, you can change the PIN mode on your device.

On Screen

The PIN is entered on the device screen. For entering the PIN choose the next digit by using the left or right button. When the digit you expect is displayed select it by pressing both buttons at the same time.

Once all digits are selected, validate the PIN by selecting the **'V'** (Validate) letter

If you want to change the previous digit select the **'C'** (Cancel) letter.

Finally if you want to abort the PIN entry, select the **'A'** (Abort) letter.

Confirm only

The user is requested, on the device screen, to confirm the PIN validation. The PIN value is not required, the user just has to push the *REJECT* or *OK* button on the device.

This is the default mode after application installation.

Trust

Act as if the PIN is always validated. This is a dangerous mode which should only be used in a highly secure environment.

4.4.4 UIF mode

By activating UIF mode for either signature, decryption or authentication, a user validation will be ask by the device each time the related operation is performed.

To activate or deactivate the UIF, select the operation to protect and press both button. When activated, a '+' symbol appears after the operation name.

4.4.5 Reset

Selecting the menu will erase all OpenPGP Card Application data and will reset the application in its '*just installed*' state.

5 OpenPGP Card application usage

5.1 GPG

The OpenGPG Card application need at least version 2.1.19 for full support. A version prior to 2.1.19 will fail when using ECC.

You should test with a test key and make a backup of your keyring before starting, except if your are sure about what you do.

5.1.1 Configuration

In order to use a Ledger device with gpg it is needed to explicitly setup the reader and the delegated PIN support. Edit the file `~/.gnupg/scdaemon.conf` and add the following lines:

```
reader-port "Ledger Token"
allow-admin
enable-pinpad-varlen
```

Note: `enable-pinpad-varlen` option is mandatory, else `gpg` could request the PIN on the *host*, which is not supported by Ledger App.

You can check `the reader-port` value by running the command line `pcsc_scan`:

```
$ pcsc_scan
Using reader plug'n play mechanism
Scanning present readers...
0: Ledger Nano S Plus [Nano S Plus] (0001) 00 00
1: Alcor Micro AU9540 01 00

Thu Jan 11 10:58:25 2024
 Reader 0: Ledger Nano S Plus [Nano S Plus] (0001) 00 00
  Event number: 0
  Card state: Card inserted, Exclusive Mode,
  ATR: 3B 00

ATR: 3B 00
+ TS = 3B --> Direct Convention
+ T0 = 00, Y(1): 0000, K: 0 (historical bytes)
 Reader 1: Alcor Micro AU9540 01 00
  Event number: 0
  Card state: Card removed,
```

5.1.2 Get/Set basic information

The `gpg --card-status` command provides default card information. Just after installation it should look like this:

```
$ gpg --card-status
Reader ...........: Ledger Nano S Plus [Nano S Plus] (0001) 01 00
Application ID ...: D2760001240103002C97AFB114290000
Version ..........: 3.3
Manufacturer .....: unknown
Serial number ....: AFB11429
```

```
Name of cardholder: [not set]
Language prefs ...: [not set]
Salutation .......:
URL of public key: [not set]
Login data .......: [not set]
Signature PIN ....: not forced
Key attributes ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 12 12 12
PIN retry counter: 3 0 3
Signature counter: 0
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]
```

You can set the user information with the `gpg --card-edit` subcommands. For examples:

```
$ gpg --card-edit
gpg/card> admin
Admin commands are allowed

gpg/card> name
Cardholder's surname: Doe
Cardholder's given name: John

gpg/card> salutation
salutation ((M)ale, (F)emale or space): M

gpg/card> list

Reader ...........: Ledger Nano S Plus [Nano S Plus] (0001) 01 00
Application ID ...: D2760001240103002C97AFB114290000
Version ..........: 3.3
Manufacturer .....: unknown
Serial number ....: AFB11429
Name of cardholder: John Doe
Language prefs ...: [not set]
Salutation .......: Mr.
URL of public key: [not set]
Login data .......: [not set]
Signature PIN ....: not forced
Key attributes ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 12 12 12
PIN retry counter: 3 0 3
Signature counter: 0
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]
```

Notes:

- Modifying the user information will prompt you to enter `User PIN`.
- Setting user information is not required for using gpg client.

5.1.3 Generate new key pair

For generating a new key pair follow those steps:

- Select the desired slot
- Setup the desired key template for this slot

- Generate the new key set

Step 1

Starting from main menu:

- Select *Select slot* menu
- Scroll to desired slot
- Select it
- Optionally set it as default by selecting *Set Default* menu
- Select *Back* to return to main menu.

Step 2

The default template for each three keys (*signature*, *decryption*, *authentication*) is `RSA 2048`. If you want another kind of key you have to set the template before generating keys.

WARNING: Changing the current template of a key automatically erases the associated one.

Starting from main menu:

- Select *Settings*
- Select *Key template*
- Select *Choose Key...* (a)
- Scroll and select which key you want to set the new template for
- Select *Choose type...*
- Scroll and select among the supported key types and sizes
- Select *Set template*
- Repeat this process from (a) if you want to modify another key template
- Select *Back* to return to main.

Step 3

Once the template has been set, it's possible to generate new key pairs with `gpg`.

WARNING: gpg will generate the 3 key pairs and will overwrite any key already present in the selected slot.

Here after is a detailed log of key generation of ECC keys, assuming the key templates are `NIST P256`.

Edit Card

```
$ gpg --edit-card
Reader ...........: Ledger Nano S Plus [Nano S Plus] (0001) 01 00
Application ID ...: D2760001240103002C97AFB1142B0000
Version ..........: 3.3
Manufacturer .....: unknown
Serial number ....: AFB1142B
Name of cardholder: John Doe
Language prefs ...: [not set]
Salutation .......: Mr.
URL of public key: [not set]
Login data .......: [not set]
Signature PIN ....: not forced
Key attributes ...: nistp256 nistp256 nistp256
Max. PIN lengths .: 12 12 12
PIN retry counter: 3 0 3
Signature counter: 0
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]
```

Switch to admin mode:

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

Request new key generation without backup

```
gpg/card> generate
Make off-card backup of encryption key? (Y/n) n
```

Unlock user level “81“

```
Please unlock the card

Number: 2C97 AFB1142B
Holder: John Doe

Use the reader's pinpad for input.
 OK
Press any key to continue.
```

Set key validity

```
Please specify how long the key should be valid.
      0 = key does not expire
   <n> = key expires in n days
  <n>w = key expires in n weeks
  <n>m = key expires in n months
  <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y
```

Set user ID

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

Real name: John Doe
Email address: john.doe@foo.com
Comment:
You selected this USER-ID:
  "John Doe <john.doe@foo.com>"

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

Unlock admin level “83“

```
Please enter the Admin PIN

Number: 2C97 AFB1142B
Holder: John Doe

Use the reader's pinpad for input.
 OK
Press any key to continue.
```

Unlock user level “81“

```
Please unlock the card

Number: 2C97 AFB1142B
Holder: John Doe
Counter: 0

Use the reader's pinpad for input.
```

```
 OK
Press any key to continue.
```

Final confirmation

```
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
gpg: key DF3FA4A33EF00E47 marked as ultimately trusted
gpg: revocation certificate stored as 'xxxx/openpgp-revocs.d/89F772243C9A3E583CB59AB5DF3FA4A33E
public and secret key created and signed.
```

Get information after key generation

```
gpg/card> list

Reader ...........: Ledger Nano S Plus [Nano S Plus] (0001) 01 00
Application ID ...: D2760001240103002C97AFB1142B0000
Version ..........: 3.3
Manufacturer .....: unknown
Serial number ....: AFB1142B
Name of cardholder: John Doe
Language prefs ...: [not set]
Salutation .......: Mr.
URL of public key: [not set]
Login data .......: [not set]
Signature PIN ....: not forced
Key attributes ...: nistp256 nistp256 nistp256
Max. PIN lengths .: 12 12 12
PIN retry counter: 3 0 3
Signature counter: 12
Signature key ....: F844 38BB CA87 F9A7 6830  F002 F8A4 A353 3CBF CAA5
    created ....: 2017-08-22 15:59:36
Encryption key....: B1D3 C9F2 C3C5 87CA 36A7  F02E E137 28E9 13B8 77E1
    created ....: 2017-08-22 15:59:36
Authentication key: F87D EF02 9C38 C43D 41F0  6872 2345 A677 CE9D 8223
    created ....: 2017-08-22 15:59:36
General key info..: pub  nistp256/F8A4A3533CBFCAA5 2017-08-22 John Doe
<john.doe@foo.com>
sec>  nistp256/F8A4A3533CBFCAA5  created: 2017-08-22  expires: never
                     card-no: 2C97 AFB1142B
ssb>  nistp256/2345A677CE9D8223  created: 2017-08-22  expires: never
                     card-no: 2C97 AFB1142B
ssb>  nistp256/E13728E913B877E1  created: 2017-08-22  expires: never
                     card-no: 2C97 AFB1142B
```

At this point it's possible to check that the key has been generated on card with the following command:

```
$ gpg --list-secret-keys john.doe@foo.com
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u

sec>  nistp256 2017-08-22 [SC]
    F84438BBCA87F9A76830F002F8A4A3533CBFCAA5
    Card serial no. = 2C97 AFB1142B
uid           [ultimate] John Doe <john.doe@foo.com>
ssb>  nistp256 2017-08-22 [A]
ssb>  nistp256 2017-08-22 [E]
```

5.1.4 Moving an existing key pair

This section shows how to move an existing key onto the Ledger device.

The key to transfer here is a RSA 4096 bits key:

```
$ gpg --list-secret-keys "RSA 4096"
sec   rsa4096 2017-04-26 [SC]
    FB6C6C75FB016635872ED3E49B93CB47F954FB53
uid           [ultimate] RSA 4096
ssb   rsa4096 2017-04-26 [E]
```

In case of transfer it is not necessary to previously set the template. It will be automatically changed. When generating a new key, the 3 keys (*signature*, *decryption*, *authentication*) are automatically generated. When transferring existing ones, it is possible to choose which one will be moved.

Edit Key

```
$ gpg --edit-key "RSA 4096"
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

sec  rsa4096/9B93CB47F954FB53
   created: 2017-04-26  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb  rsa4096/49EE12B0F5CBDF26
   created: 2017-04-26  expires: never       usage: E
[ultimate] (1). RSA 4096
```

Select the key to move, here the encryption one.

```
gpg> key 1

sec  rsa4096/9B93CB47F954FB53
  created: 2017-04-26  expires: never       usage: SC
  trust: ultimate      validity: ultimate
ssb* rsa4096/49EE12B0F5CBDF26
   created: 2017-04-26  expires: never       usage: E
[ultimate] (1). RSA 4096
```

Move

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

Unlock admin level “83“

```
Please enter the Admin PIN

Number: 2C97 1D49B409
Holder:

Use the reader's pinpad for input.
 OK
Press any key to continue.
```

Unlock admin level “83“ (maybe twice....)

```
Please enter the Admin PIN

Number: 2C97 1D49B409
Holder:
```

```
Use the reader's pinpad for input.
 OK
Press any key to continue.

sec  rsa4096/9B93CB47F954FB53
   created: 2017-04-26  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb* rsa4096/49EE12B0F5CBDF26
   created: 2017-04-26  expires: never       usage: E
[ultimate] (1). RSA 4096

gpg> save
gpg> quit
```

check

```
$ gpg --edit-key "RSA 4096"
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

sec  rsa4096/9B93CB47F954FB53
   created: 2017-04-26  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb  rsa4096/49EE12B0F5CBDF26
   created: 2017-04-26  expires: never       usage: E
  card-no: 2C97 7BB895B9
[ultimate] (1). RSA 4096
```

The encryption key is now associated with a card.

5.1.5 Decrypting and Signing

Decrypting and Signing will act exactly the same way as if keys were not on the card. The only difference is gpg will request the PIN code instead of the passphrase.

5.2 SSH

5.2.1 Overview

In order to use gpg for SSH authentication, an "authentication" is needed. There are two solutions for that, either generate one on the device or add an authentication sub-key to your existing master gpg key.

Once done, it is necessary to configure ssh to point to the right key and delegate the authentication to *gpg-ssh-agent* instead of *ssh-agent*.

5.2.2 Generate new key on device

The important thing to keep in mind here is there is no way to tell gpg to only generate the authentication key. So generating this key will also generate the two other under a new identity and will erase existing keys on the current slot on the device.

Nevertheless, if you want to use a different identity for ssh login, you can use another slot on the device. See OpenPGP Card application explained and Generate new key pair.

5.2.3 Add sub-key

Edit pgp key set

```
$ gpg --expert --edit-key "john.doe@foo.com"
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
```

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

sec  rsa2048/831415DA94A9A15C
   created: 2017-08-25  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb  rsa2048/8E95F2999EEC38C4
   created: 2017-08-25  expires: never       usage: E
[ultimate] (1). John Doe <john.doe@foo.com>
```

Add sub key

```
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
 (14) Existing key from card
Your selection? 8
```

Toggle sign/encrypt OFF, Toggle authentication ON

```
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
```

Set key options

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
       0 = key does not expire
    <n>  = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
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  rsa2048/831415DA94A9A15C
   created: 2017-08-25  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb  rsa2048/8E95F2999EEC38C4
   created: 2017-08-25  expires: never       usage: E
ssb  rsa2048/C20B90E12F68F035
   created: 2017-08-28  expires: never       usage: A
[ultimate] (1). John Doe <john.doe@foo.com>
```

Select the key and move it

```
gpg> key 2

sec  rsa2048/831415DA94A9A15C
   created: 2017-08-25  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb  rsa2048/8E95F2999EEC38C4
  created: 2017-08-25  expires: never       usage: E
ssb* rsa2048/C20B90E12F68F035
   created: 2017-08-28  expires: never       usage: A
[ultimate] (1). John Doe <john.doe@foo.com>

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

sec  rsa2048/831415DA94A9A15C
```

```
   created: 2017-08-25  expires: never       usage: SC
   trust: ultimate      validity: ultimate
ssb  rsa2048/8E95F2999EEC38C4
   created: 2017-08-25  expires: never       usage: E
ssb* rsa2048/C20B90E12F68F035
   created: 2017-08-28  expires: never       usage: A
[ultimate] (1). John Doe <john.doe@foo.com>

gpg> save
```

5.2.4 Configure SSH and GPG

First, tell `gpg-agent` to enable `ssh-auth` feature by adding the following line to your `.gpg-agent.conf`:

```
enable-ssh-support
```

Starting with `gpg` is necessary to add some configuration options to make the *pinentry* work properly. Add the following line to `~/.bashrc` file:

```
export SSH_AUTH_SOCK=`gpgconf --list-dirs agent-ssh-socket`
export GPG_TTY=`tty`
gpgconf --launch gpg-agent
```

It may be also necessary to setup the loopback pinentry options.

Add the following line to your `~/.gnupg/gpg-agent.conf`:

```
allow-loopback-pinentry
```

And add the following line to your `~/.gnupg/gpg.conf`:

```
pinentry-mode loopback
```

Then export your authentication public key. First execute the command: `gpg -k --with-subkey-fingerpr --with-keygrip john.doe@foo.com`.

```
pub   rsa2048 2017-08-25 [SC]
    7886147C4C2E5CE2A4B1546C831415DA94A9A15C
    Keygrip = DE2B63C13AB92EBD2D05C1021A9DAA2D40ECB564
uid           [ultimate] John Doe <john.doe@foo.com>
sub   rsa2048 2017-08-25 [E]
    789E56872A0D9A5AC8AF9C2F8E95F2999EEC38C4
    Keygrip = 9D7C2EF8D84E3B31371A09DFD9A4B3EF72AB4ACE
sub   rsa2048 2017-08-28 [A]
    2D0E4FFFAA448AA2770C7F02C20B90E12F68F035
    Keygrip = 6D60CB58D9D66EE09804E7FE460E865A91F5E41A
```

Add the `keygrip` of the authentication key, the one identified by `[A]`, to `.gnupg/sshcontrol` file:

```
$ echo 6D60CB58D9D66EE09804E7FE460E865A91F5E41A > .gnupg/sshcontrol
```

Export your authentication key, identifier by its fingerprint, in a SSH compliant format.

```
$ gpg --export-ssh-key 2D0E4FFFAA448AA2770C7F02C20B90E12F68F035
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDCIARKh0IZTHld+I6oA8nwrgnCUQE8f
7X3pmI4ZwryT52fKhpcsQJsd3krodXrM//LiK8+m2ZRMneJ9iGlqqE7SCyZkNBj1GUm9s
rK3Q5eoR6nU0s+sq17b/FAtQWHBJTqqaOtyA33hFj5twUtWZ6rokX9cNZrD1ne8kRVHDe
3uEBsaY5PR1Tuko/GwywLyZu0SwfEobl/RPjL7P8rUSc7DTHpQMw8fjJFb4BNvIHAlaVC
5FwZwkuogygaJdN/44MayHFmOZmzx9CAgYgLpTzen35+CcyhlqCqi+HjNlnHL2DDWd4iR
d3Y6pY8LjS3xQkECc3Bhedptp17D+H9AVJt openpgp:0x2F68F035
```

Finally copy the above export (`ssh-rsa AAAAB...Jt openpgp:0x2F68F035`) into the `~/.ssh/authorized_keys` file on your remote server.

Now, if everything is correctly setup and running, an `ssh-add -l` should show your key:

```
$ ssh-add -l
2048 SHA256:sLCzsoi5GAG2kJkG6hSp8gTLPxSvo/zNtsks2kQ7vTU cardno:2C979421A9E1
(RSA)
2048 SHA256:sLCzsoi5GAG2kJkG6hSp8gTLPxSvo/zNtsks2kQ7vTU (none) (RSA)
```

And you should be able to ssh to your remote server with your gpg key!

5.3 Backup and Restore

5.3.1 Introduction

The OpenPGP card specification does not provide any mechanism for backuping you key. Thus if you generate your keys on device and loose it, you definitively loose you private key.

In order to avoid such extreme panic situation, a backup/restore mechanism is provided. At any time you can backup a snapshot of your device data, including your private keys. All public data are retrieve in clear form. The private key are stored encrypted with a key derived from your seed, i.e. from your 24 BIP words.

The backup/restore tool is located in `pytools` directory.

See Tools later in this document for the tools details and usage.

Note: The keys backup will work *only* if the SEED Mode is enabled!

5.3.2 Restore without backup

If you have seeded key but do not have done a backup and still have your keyring, there is a solution to restore at least the key and their related information: serial and fingerprints. All other information such as name, url, ... shall be set manually with `gpg --card-edit`.

Step 1: Retrieve information

Run the command `gpg --edit-key john.doe@foo.com`.

```
$ gpg --edit-key john.doe@foo.com
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

sec  ed25519/8451AAF7D43D1095
   created: 2018-10-10  expires: never       usage: SC
   card-no: 2C97 FD6C11BE
   trust: ultimate      validity: ultimate
ssb  ed25519/C5A8FB078520ABBB
   created: 2018-10-10  expires: never       usage: A
   card-no: 2C97 FD6C11BE
ssb  cv25519/0953D871FC4B9EA4
   created: 2018-10-10  expires: never       usage: E
   card-no: 2C97 FD6C11BE
[ultimate] (1). John Doe <john.doe@foo.com>
```

The *usage* field tells you each key purpose:

- **S** for signature,
- **C** for Certification (subkey signature),
- **A** for authentication,
- **E** for encryption.

The *card-no* field provides you with the serial number of the card on which the key are stored. You should have 3 or less keys with the same serial. These are the keys we want to restore.

For each key you also have the key template (*rsa2048*, *rsa3072*, *ed2559*, *cv25519*) followed by the short fingerprint, e.g. `ed25519/8451AAF7D43D1095`

Please note the serial and the 3 key template names: `FD6C11BE` , `ed25519:cv25519:ed25519`. Take care of the order: `SC:E:A`.

To get the full fingerprint of each key, run (yes twice `--fingerprint`):

```
$ gpg --fingerprint --fingerprint John
pub   ed25519 2018-10-10 [SC]
    2C68 8345 BDDA 0EDF B24D  B4FB 8451 AAF7 D43D 1095
uid           [ultimate] John Doe <john.doe@foo.com>
sub   ed25519 2018-10-10 [A]
    CEC5 9AE6 A766 14BC 3C6D  37D9 C5A8 FB07 8520 ABBB
sub   cv25519 2018-10-10 [E]
    DF15 7BD4 AC3B D1EE 9910  99C8 0953 D871 FC4B 9EA4
```

Assemble the 3 full fingerprint, corresponding to the one identified previously, in the the following order `SC:E:A`:

`2C688345BDDA0EDFB24DB4FB8451AAF7D43D1095:DF157BD4AC3BD1EE991099C80953D871FC4B9EA4:CEC59AE6A76614BC3C6D37D9C5A8FB078520ABBB`.

Note: If you only have one single key to restore you can omit the others. For example, to only restore the authentication key: `::CEC59AE6A76614BC3C6D37D9C5A8FB078520ABBB`

Step 2: Restore

Plug your device and run the OpenPGP application.

Finally run the following command:

```
python3 -m gpgcard.gpgcli --pinpad --set-template ed255519:cv25519:ed255519
--set-fingerprints
 '2C688345BDDA0EDFB24DB4FB8451AAF7D43D1095:DF157BD4AC3BD1EE991099C80953D871FC4B9EA4:CEC59AE6A76
 --set-serial 'FD6C11BE' --seed-key
```

5.3.3 Restore lost Keyring

In case the local keyring files are lost, follow the recovery process hereafter. Usually under `~/.gnupg/`, the keyring files contain the Public keys and associated metadata.

Step 1: Retrieve key metadata

Check that your device is connected and recognised, and print out the *keygrips* and *creation timestamps* of your keys:

```
$ gpg --card-status --with-keygrip

Reader ...........: Ledger Nano S Plus [Nano S Plus] (0001) 00 00
Application ID ...: D2760001240103032C97E1A67CBF0000
Application type .: OpenPGP
Version ..........: 3.3
Manufacturer .....: unknown
Serial number ....: E1A67CBF
Name of cardholder: [not set]
Language prefs ...: [not set]
Salutation .......:
URL of public key : [not set]
Login data .......: [not set]
Signature PIN ....: not forced
Key attributes ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 12 12 12
PIN retry counter : 3 0 3
Signature counter : 4
```

```
Signature key ....: FE93 6FEC 13BE BDAA A0C6  3E72 05DC 472D A6F6 A13B
    created ....: 2024-01-18 10:08:41
    keygrip ....: 348411953EBC6DE6416D40A7048F5C5795A956A2
Encryption key....: CD29 B086 FE23 3DAD 3D51  B713 7E6F 425E 7A90 EE9E
    created ....: 2024-01-18 10:08:41
    keygrip ....: 1066E2EC6FB7F21738C010D62676CA64FDD5001F
Authentication key: 218F 67FB 8577 1DF1 60C1  CFE0 4A6F EB8C 0F76 76FD
    created ....: 2024-01-18 10:08:41
    keygrip ....: 73921B6FC73851E61AE9A0196003BE9516B916A0
General key info..:
```

Step 2: Import the Master key

First, import your master **Signature key** from the device.

Because GPG key IDs are based in part on their creation time, we need to set a fake system time to match the *created* time for the Signature key shown above.

Convert the creation date format like so by removing punctuation, adding a "T" between the date and time, and adding an exclamation mark to the end:

`2024-01-18 10:08:41` becomes `20240118T100841!`

Add that to your GPG arguments like so to start importing the key. When the menu pops up, pick the *Existing key from card* option. Then, pick the key which has the **cert,sign** right enabled **(1)**, and follow through the prompts to create your user ID.

```
$ gpg --faked-system-time "20240118T100841!" --full-generate-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: WARNING: running with faked system time: 2024-01-18 10:08:41
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)
 (14) Existing key from card
Your selection? 14
Serial number of the card: D2760001240103032C97E1A67CBF0000
Available keys:
  (1) 348411953EBC6DE6416D40A7048F5C5795A956A2 OPENPGP.1 rsa2048 (cert,sign)
  (2) 1066E2EC6FB7F21738C010D62676CA64FDD5001F OPENPGP.2 rsa2048 (encr)
  (3) 73921B6FC73851E61AE9A0196003BE9516B916A0 OPENPGP.3 rsa2048 (sign,auth)
Your selection? 1
Please specify how long the key should be valid.
       0 = key does not expire
    <n>  = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (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: testkey
Email address:
Comment:
You selected this USER-ID:
```

```
 "testkey"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
gpg: xxxx/manual-tests/gnupg/trustdb.gpg: trustdb created
gpg: key 05DC472DA6F6A13B marked as ultimately trusted
gpg: directory 'xxxx/manual-tests/gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as 'xxxx/manual-tests/gnupg/openpgp-revocs.d/FE936FEC13BEBDA
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   rsa2048 2024-01-18 [SC]
     FE936FEC13BEBDAAA0C63E7205DC472DA6F6A13B
uid                      testkey
```

Step 3: Import the Encryption subkey

Next, add the **encr** key as subkey of this master key.

Use the ID of the master key that was printed in that final **pub rsa2048** block to start editing it, along with the creation dates from `card-status`.

Note: Please ensure the creation date is the same, or update the command line accordingly!

```
$ gpg --faked-system-time "20240118T100841!" --edit-key FE936FEC13BEBDAAA0C63E7205DC472DA6F6A13
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: WARNING: running with faked system time: 2024-01-18 10:08:41
Secret key is available.

gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
sec  rsa2048/05DC472DA6F6A13B
    created: 2024-01-18  expires: never       usage: SC
    card-no: 2C97 E1A67CBF
    trust: ultimate      validity: ultimate
[ultimate] (1). testkey

gpg> addkey
Secret parts of primary key are stored on-card.
Please select what kind of key you want:
  (3) DSA (sign only)
  (4) RSA (sign only)
  (5) Elgamal (encrypt only)
  (6) RSA (encrypt only)
 (14) Existing key from card
Your selection? 14
Serial number of the card: D2760001240103032C97E1A67CBF0000
Available keys:
  (1) 348411953EBC6DE6416D40A7048F5C5795A956A2 OPENPGP.1 rsa2048 (cert,sign)
  (2) 1066E2EC6FB7F21738C010D62676CA64FDD5001F OPENPGP.2 rsa2048 (encr)
  (3) 73921B6FC73851E61AE9A0196003BE9516B916A0 OPENPGP.3 rsa2048 (sign,auth)
Your selection? 2
Please specify how long the key should be valid.
        0 = key does not expire
     <n>  = key expires in n days
     <n>w = key expires in n weeks
     <n>m = key expires in n months
```

```
    <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y
Really create? (y/N) y

sec  rsa2048/05DC472DA6F6A13B
    created: 2024-01-18  expires: never       usage: SC
    card-no: 2C97 E1A67CBF
    trust: ultimate      validity: ultimate
ssb  rsa2048/7E6F425E7A90EE9E
    created: 2024-01-18  expires: never       usage: E
    card-no: 2C97 E1A67CBF
[ultimate] (1). testkey
```

Step 4: Import the Authentication subkey

Finally, add the **sign,auth** key as subkey of this master key.

Use the ID of the master key that was printed in that final **pub rsa2048** block to start editing it, along with the creation dates from `card-status`.

Note: Please ensure the creation date is the same, or update the command line accordingly!

```
gpg> addkey
Secret parts of primary key are stored on-card.
Please select what kind of key you want:
  (3) DSA (sign only)
  (4) RSA (sign only)
  (5) Elgamal (encrypt only)
  (6) RSA (encrypt only)
 (14) Existing key from card
Your selection? 14
Serial number of the card: D2760001240103032C97E1A67CBF0000
Available keys:
  (1) 348411953EBC6DE6416D40A7048F5C5795A956A2 OPENPGP.1 rsa2048 (cert,sign)
  (2) 1066E2EC6FB7F21738C010D62676CA64FDD5001F OPENPGP.2 rsa2048 (encr)
  (3) 73921B6FC73851E61AE9A0196003BE9516B916A0 OPENPGP.3 rsa2048 (sign,auth)
Your selection? 3
Please specify how long the key should be valid.
       0 = key does not expire
    <n>  = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y
Really create? (y/N) y

sec  rsa2048/05DC472DA6F6A13B
    created: 2024-01-18  expires: never       usage: SC
    card-no: 2C97 E1A67CBF
    trust: ultimate      validity: ultimate
ssb  rsa2048/7E6F425E7A90EE9E
    created: 2024-01-18  expires: never       usage: E
    card-no: 2C97 E1A67CBF
ssb  rsa2048/4A6FEB8C0F7676FD
    created: 2024-01-18  expires: never       usage: SA
    card-no: 2C97 E1A67CBF
[ultimate] (1). testkey
```

```
gpg> save
gpg> quit
```

Now you should be able to see your imported key by running this command:

```
$ gpg --list-secret-keys --with-keygrip
xxxx/manual-tests/gnupg/pubring.kbx
------------------------------------------------------------------------
sec>  rsa2048 2024-01-18 [SC]
     FE936FEC13BEBDAAA0C63E7205DC472DA6F6A13B
     Keygrip = 348411953EBC6DE6416D40A7048F5C5795A956A2
     Card serial no. = 2C97 E1A67CBF
uid           [ultimate] testkey
ssb>  rsa2048 2024-01-18 [E]
     Keygrip = 1066E2EC6FB7F21738C010D62676CA64FDD5001F
ssb>  rsa2048 2024-01-18 [SA]
     Keygrip = 73921B6FC73851E61AE9A0196003BE9516B916A0
```

6 Tools

There are 2 tools provided:

- `gpgcli.py`: General test tool
- `backup.py`: Backup and Restore of the configuration

If you encounter an error when performing the backup/restore, reload your scdaemon with `gpgconf --reload scdaemon`

6.1 Test command line tool

A test tool is provided under the directory `pytools`. There may be some dependencies package to install. Thus, don't forget to use the command:

```
pip install -r requirements.txt
```

This tool allows to execute lots of test with the device, like key generation, metadata modification, or simply get the information on the configuration and keys.

Its usage is:

```
$ ./gpgcli.py --help
usage: gpgcli.py [-h] [--info] [--reader READER] [--apdu] [--slot {1,2,3}]
[--reset] [--pinpad] --adm-pin PIN --user-pin PIN [--new-user-pin PIN]
[--new-adm-pin PIN] [--reset-code RESET_CODE | --reset-pw1 RESET_PW1] [--serial
SERIAL]
          [--salutation {Male,Female}] [--name NAME] [--url URL] [--login
LOGIN] [--lang LANG] [--key-type {SIG,DEC,AUT}] [--key-action {Export,Generate,Read}]
[--set-fingerprints SIG:DEC:AUT] [--set-templates SIG:DEC:AUT] [--seed-key]
          [--file FILE]

Manage OpenPGP App on Ledger device

options:
 -h, --help            show this help message and exit
 --info                Get and display card information
 --reader READER       PCSC reader name (default is 'Ledger')
 --apdu                Log APDU exchange
 --slot {1,2,3}        Select slot (1 to 3)
 --reset               Reset the application (all data will be erased)
 --pinpad              PIN validation will be delegated to pinpad
 --adm-pin PIN         Admin PIN (if pinpad not used)
```

```
--user-pin PIN        User PIN (if pinpad not used)
--new-user-pin PIN    Change User PIN
--new-adm-pin PIN     Change Admin PIN
--reset-code RESET_CODE
              Update 'PW1 Resetting Code'
--reset-pw1 RESET_PW1
              Reset the User PIN
--serial SERIAL       Update the 'serial' data (4 bytes)
--salutation {Male,Female}
              Update 'salutation' data
--name NAME           Update 'name' data
--url URL             Update 'url' data
--login LOGIN         Update 'login' data
--lang LANG           Update 'lang' data
--key-type {SIG,DEC,AUT}
              Select key type SIG:DEC:AUT (default is all)
--key-action {Export,Generate,Read}
              Generate key pair or Read public key
--set-fingerprints SIG:DEC:AUT
              Set fingerprints for selected 'key-type'
              If 'key-type' is not specified, set for all keys (SIG:DEC:AUT)
              Each fingerprint is 20 hex bytes long
--set-templates SIG:DEC:AUT
              Set template identifier for selected 'key-type'
              If 'key-type' is not specified, set for all keys (SIG:DEC:AUT)
              Valid values are rsa2048, rsa3072, nistp256, ed25519, cv25519
--seed-key            Regenerate all keys, based on seed mode
--file FILE           Public Key export file (default is 'pubkey')
```

Sample output to get Card information:

```
$ ./gpgcli.py --adm-pin 12345678 --user-pin 123456 --info
Connect to card 'Ledger'...
Verify PINs...
Get card info...
=============== Application Identifier ===============
# AID                  : D2760001240103032C97E1A67CBF0000
 - RID                : D276000124
 - Application        : 01
 - Version            : 3.3
 - Manufacturer       : 2C97
 - Serial             : E1A67CBF
=============== Historical Bytes ===============
 - historical bytes    : 0031c573c00180000000000000059000
=============== Max Extended Length ===============
 - Command            : 254
 - Response           : 254
=============== PIN Info ===============
 - PW1                : UTF-8 (12 bytes), Error Counter=3, Validity=Several
PSO:CDS
 - Reset Counter      : UTF-8 (12 bytes), Error Counter=0
 - PW3                : UTF-8 (12 bytes), Error Counter=3
=============== Extended Capabilities ===============
 - Secure Messaging    : ✗
 - Get Challenge       : ✓ (Max length: 254)
 - Key import          : ✓
 - PW status           : Changeable
 - Private DOs         : ✓
 - Algo attributes     : Changeable
```

```
 - PSO:DEC AES         : ✓
 - Key Derived Format  : ✗
 - Max Cert len        : 2560
 - Max Special DO      : 512
 - PIN 2 format        : ✗
 - MSE                 : ✓
=============== Hardware Features ===============
 - Display             : ✗
 - Biometric sensor    : ✗
 - Button/Keypad       : ✓
 - LED                 : ✗
 - Loudspeaker         : ✗
 - Microphone          : ✗
 - Touchscreen         : ✗
 - Battery             : ✗
=============== User Info ===============
 - Name                :
 - Login               :
 - URL                 :
 - Salutation          :
 - Lang                :
=============== Slots Info ===============
 - Number of Slots     : 3
 - Default Slot        : 1
 - Selection by APDU   : ✓
 - Selection by screen : ✓
 - Current             : 1
=============== Keys Info ===============
 - CDS counter         : 8
 - RSA Pub Exponent    : 0x010001
SIG:
  - UIF                : ✗
  - Fingerprint        : fe936fec13bebdaaa0c63e7205dc472da6f6a13b
  - CA fingerprint     : N/A
  - Creation date      : 2024-01-18 10:08:41
  - Attribute          : RSA-2048, Format: standard with modulus (n), Exponent
size: 32
  - Certificate        :
  - Key:
    * OS Target ID     : 0x33100004
    * API Level        : 12
    * Public exp size  : 4
    * Public exp       : 0x010001
    * Private key size: 1040
DEC:
  - UIF                : ✗
  - Fingerprint        : cd29b086fe233dad3d51b7137e6f425e7a90ee9e
  - CA fingerprint     : N/A
  - Creation date      : 2024-01-18 10:08:41
  - Attribute          : RSA-2048, Format: standard with modulus (n), Exponent
size: 32
  - Certificate        :
  - Key:
    * OS Target ID     : 0x33100004
    * API Level        : 12
    * Public exp size  : 4
    * Public exp       : 0x010001
    * Private key size: 1040
```

```
AUT:
 - UIF               : ✗
 - Fingerprint       : 218f67fb85771df160c1cfe04a6feb8c0f7676fd
 - CA fingerprint    : N/A
 - Creation date     : 2024-01-18 10:08:41
 - Attribute         : RSA-2048, Format: standard with modulus (n), Exponent
size: 32
 - Certificate       :
 - Key:
  * OS Target ID    : 0x33100004
  * API Level       : 12
  * Public exp size : 4
  * Public exp      : 0x010001
  * Private key size: 1040
```

6.2 Backup tool

The tool usage is the following:

```
$ ./backup.py --help
usage: backup.py [-h] [--reader READER] [--slot {1,2,3}] [--pinpad] --adm-pin
PIN --user-pin PIN [--restore] [--file FILE]

Backup/Restore OpenPGP App configuration

options:
 -h, --help       show this help message and exit
 --reader READER  PCSC reader name (default is 'Ledger')
 --slot {1,2,3}   Select slot (1 to 3)
 --pinpad         PIN validation will be delegated to pinpad
 --adm-pin PIN    Admin PIN (if pinpad not used)
 --user-pin PIN   User PIN (if pinpad not used)
 --restore        Perform a Restore instead of Backup
 --file FILE      Backup/Restore file (default is 'gpg_backup')
 --seed-key       After Restore, regenerate all keys, based on seed mode

Keys restore is only possible with SEED mode...
```

To perform a backup, simply use the tool like this:

```
$ ./backup.py --adm-pin 12345678 --user-pin 123456
Connect to card 'Ledger'...
Configuration saved in file 'gpg_backup'.
```

To *restore* a backup, simply use the tool like this:

```
$ ./backup.py --restore --adm-pin 12345678 --user-pin 123456 --seed-key
Connect to card 'Ledger'...
Configuration saved in file 'gpg_backup'.
```

7 Annexes

7.1 Trouble/FAQ

Q: It may happens the reader is no more visible with `gpg` tool, whereas it can be seen by `pytools`.

R: In such case (which seems to be linked to the PC configuration), one solution is to re-install the tool packages and libraries:

```
sudo apt remove --purge libpcsclite-dev scdaemon pcscd opensc pcsc-tool
sudo apt autoremove
sudo apt install libpcsclite-dev scdaemon pcscd opensc pcsc-tool
```

Q: gpg-connection agent failed

R: Check that you don't have multiple running agents. After setting-up all SSH stuff, try to fully logout/login

Q: It does not work at all, HELP ME!!!

R Please keep calm and do not cry. Add the following option to `~/.gnupg/gpg-agent.conf`

```
debug-level guru
log-file /tmp/gpgagent.log
```

Add the following option to `~/.gnupg/scdaemon.conf`

```
log-file /tmp/scd.log
debug-level guru
debug-all
```

Make a nice issue report under github providing log and and command line you run.

WARNING: This may reveal confidential information such as key values. Do your log with a test key.

Q: I'm having issue when using SSH, there is no pinpad prompt. (`sign_and_send_pubkey: signing failed: agent refused operation`)

R: You might need to add this command to your `.bashrc` or `.zshrc`:

```
gpg-connect-agent updatestartuptty /bye >/dev/null
```

Q: My mac is not able to see my `Ledger Token`

R: For some reason, SC communication on Mac takes some times or mess it up sometimes.

To troubleshot those issues, you can try to reload the `scdaemon` using this command:

```
gpgconf --reload scdaemon
gpgconf --reload gpg-agent
```

If not successful, you can try to trigger daemons to restart by sending a **SIGTERM** like so:

```
kill -TERM $(pgrep gpg-agent) $(pgrep scdaemon).
```

Changing USB port might also help sometimes. Do not hesitate.

Q: My mac is **STILL** not able to see my `Ledger Token`

R: This might be related to your CCID drivers. You can manually install a more recent version from this https://ccid.apdu.fr/files/ and install it this way:

```
CCID_VERSION=1.5.4
wget https://ccid.apdu.fr/files/ccid-${CCID_VERSION}.tar.bz2
tar xzvf ccid-${CCID_VERSION}.tar.bz2
cd ccid-${CCID_VERSION}
./MacOSX/configure
make
make install
```

Installing the driver depends on `libusb` which can be installed using the following `brew install libusb`. It also requires static linking against it, if you use dynamic linking you will have the following output when using the `./MacOSX/configure` step:

```
/usr/local/Cellar/libusb/1.0.23/lib/libusb-1.0.0.dylib
/usr/local/Cellar/libusb/1.0.23/lib/libusb-1.0.dylib
****************************
Dynamic library libusb found in /usr/local/Cellar/libusb/1.0.23/lib
```

```
****************************
Rename it to force a static link
```

You can use the following:

```
LIBUSB_VERSION=1.0.23

for f in /usr/local/Cellar/libusb/${LIBUSB_VERSION}/lib/*.dylib; do
  mv $f $f.fake
done

./MacOSX/configure

for f in /usr/local/Cellar/libusb/${LIBUSB_VERSION}/lib/*.dylib.fake; do
  ORIG="$( echo $f | sed 's#.fake##g' )"
  mv $f ${ORIG}
done
```

7.2 References

G10CODE *The OpenPGP card application*, https://g10code.com/p-card.html

GPG *The GNU Privacy Guard*, https://gnupg.org/

GPGADD *The OpenPGP card application add-on*, https://github.com/LedgerHQ/app-openpgp/blob/master/doc/developer/gpgcard-addon.rst

GPGSC *The GnuPG Smartcard HOWTO*, https://gnupg.org/howtos/card-howto/en/smartcard-howto.html