

csiPass OpenPGP Manual

This guide is for someone who has a csiPass device with a smart-card reader (CCID) and wants to use its **OpenPGP** application: what OpenPGP is, how csiPass presents it on the glass and in the companion, why so many Confirm presses appear during setup, how the same steps look in GnuPG, and how to use the card once keys are on it.

It assumes you can already unlock the companion and find **Secrets** → **OpenPGP** (**Alt+G**). Everyday FIDO use lives in the [User Manual](#); the full companion surface lives in the [Companion Manual](#); the other card application is in the [PIV Manual](#). Design detail for implementers lives under `docs/openpgp-piv/` (cite via `docs/stubs/pgp-piv.md`).

Prototype caution. OpenPGP on csiPass is laboratory tooling on development images. Use test identities and test mail. Do not make this the only place a key you care about exists. What this guide describes has been run on a board with GnuPG 2.5.22 on Windows and the companion closed – signing, encryption to Curve25519, RSA-2048 and RSA-3072 keys, SSH through `gpg-agent`, the press settings, password tries and KDF (test protocols 23f, 23g and 23i) – and the application image (`esp32s3`) does not yet talk to GnuPG: that waits for a manufacturer ID of its own. Use a LAB card image (`esp32s3-lab-card`).

What OpenPGP is

OpenPGP is a format and a set of practices for:

- **Signing** – proving a message or file came from a holder of a private key
- **Encryption** – sealing a message so only the intended private key can open it
- **Authentication** – proving presence of a key to a service (SSH, mail login, and similar), without necessarily encrypting a long message

People usually meet it as **GnuPG** (`gpg`): software on a computer that holds keys, talks to smart cards, and builds OpenPGP packets. The **OpenPGP Card** profile is a smart-card application: the private keys stay on the card; the host sends digests or ciphertext and gets signatures or plaintext back. csiPass implements that card application beside FIDO, over a separate USB smart-card interface.

Three roles on the card

Role	Typical use	Press on <code>csiPass</code>
Signing (SIG)	Cleartext, detach-sign, certify other keys, signed Git commits	On once a key is made – every signature waits for Confirm
Decryption (DEC)	Open mail or files encrypted to you	Off once a key is made – turn it on if a program holding your User PIN must not decrypt unseen
Login (AUT)	SSH through <code>gpg-agent</code> , challenge-response login; the cardholder X.509 certificate names this key	On once a key is made

The press setting is the card's **User Interaction Flag** (UIF). A key made on `csiPass` takes the defaults above unless you already set that role's flag in this life of the application; a flag you set is never overwritten by a later key.

Each role has its own key material and algorithm attributes. `csiPass` makes Ed25519 / X25519, NIST P-256, and RSA-2048 / RSA-3072 – the card lists what it offers in its algorithm information (`FA`), and GENERATE for anything else is refused. NIST P-384 is built in and offered now that its measurement on this board is recorded. A P-384 key made by an earlier build still signs and decrypts, but an older build that did not list its algorithm may not be recognised by GnuPG: make a new one in an offered algorithm. An empty application starts with Ed25519 for SIG and AUT and X25519 for DEC. Private keys are generated **on the device** and never leave it. There is no export of private material, and there is no import of one either (below).

PIN worlds – do not confuse them

Secret	Where it lives	What it unlocks
Device PIN (management / FIDO Client PIN)	The device	Companion Unlock ; management commands (Set up, reset, device settings, ...); FIDO user verification
OpenPGP User PIN (PW1)	OpenPGP application	Using SIG, DEC and AUT keys: signing, decrypting, logging in
OpenPGP Admin PIN (PW3)	OpenPGP application	Making keys, key attributes, UIF, cardholder fields, the certificate, the signature-PIN policy, unblocking PW1
Reset code (optional)	OpenPGP application	Unblocking PW1 without PW3

- PW1 is **6 or more** characters, PW3 and the reset code **8 or more**, each at most 127. Any characters are allowed. PW1 and PW3 must differ.
 - Each has **3 tries** until you set others (**Password tries**, 1 to 15 each). A blocked PW1 comes back with PW3 or the reset code; a blocked reset code with PW3. **A blocked PW3 has no way back except a Full OpenPGP reset**, which erases the keys.
 - The companion never stores any of these after an operation finishes, and the device never shows them.
-

Which firmware carries OpenPGP

The smart-card reader is part of the ordinary application image and of the **lab card** image (`esp32s3-lab-card`). The ordinary **lab** image (`esp32s3-lab`) keeps a USB serial log in its place and has no reader, so the OpenPGP and PIV tabs are absent there.

GnuPG needs the card's identity – its application identifier and serial number – to recognise it. That identity is issued under a registered manufacturer number, which `csiPass` does not have yet. So today:

- the **lab card** image answers with a clearly labelled test identity (manufacturer `0xFF00`), and GnuPG works with it;
- the **application** image answers no identity at all, and GnuPG does not recognise the card, although the companion can still set it up, make keys and read them.

This is a product gate, not a missing feature, and it lifts once a registered number exists.

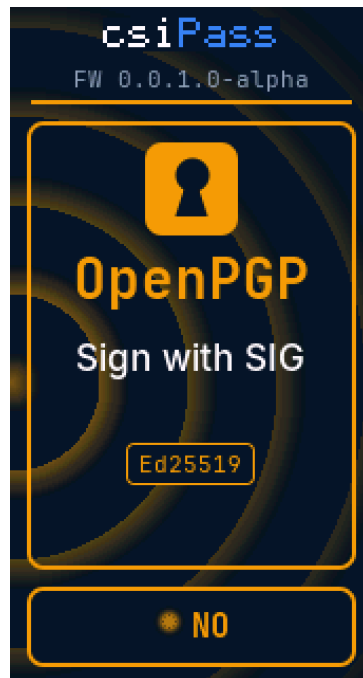
How it works on `csiPass`

1. The firmware runs an **OpenPGP Card 3.4**-shaped application on the CCID interface. Hosts see a smart-card reader beside the FIDO security key.
2. **Commissioning (Set up)** installs PW1, PW3 and an optional reset code over the **management** USB channel, with a sealed handshake and glass confirmation (below). Until that succeeds, the OpenPGP application has no credentials and answers every PIN command that it has none.
3. **Keys** are created later – in the companion **Keys** card or with GnuPG – over **PC/SC**, with PW3 verified and Confirm on the glass for **every** new key, into an empty role as well as over an old one.
4. Day-to-day **sign / decrypt / authenticate** goes through GnuPG or another OpenPGP tool talking to the same reader. The companion does not encrypt or decrypt OpenPGP messages; it manages the card and can sign a PKCS#10 request for the login key's cardholder certificate.

Management HID and the smart-card reader share the device's attention. While the companion holds the reader for **Read** or a key operation, pause other programs that keep the reader (especially GnuPG's `scdaemon`): `gpgconf --kill scdaemon`. The companion never kills it for you.

What the glass shows

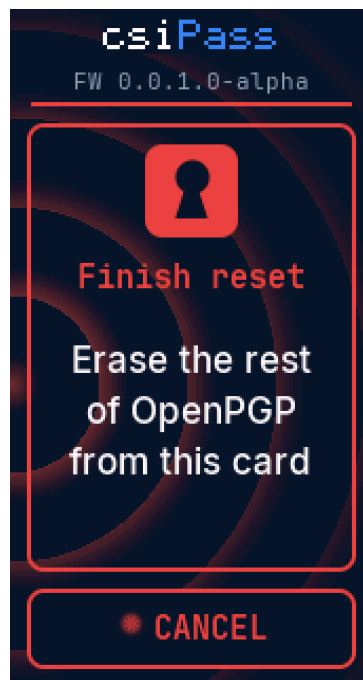
Every question the card asks you is on the device's screen, drawn from what the card itself parsed – never from text the computer sent. Amber asks, blue works, green is done, red is destructive or refused:



A signature waiting for Confirm



The card making a key



A reset left unfinished

On the glass	When	Your answer
OpenPGP – <i>Sign with SIG</i> , the algorithm beside it	A signature, while SIG's press is on	Confirm approves; BOOT refuses
OpenPGP – <i>Decrypt with DEC</i>	A decryption, while DEC's press is on	as above
OpenPGP – <i>Log in with AUT</i>	An authentication, while AUT's press is on	as above
OpenPGP – <i>New SIG key</i>	Any GENERATE into an empty role	as above
OpenPGP , in red – <i>New SIG key / Replaces SIG key</i>	GENERATE over a key the role already holds	as above; the old key is gone once you confirm. the red Cancel (Esc) in the companion's window while it waits refuses it as BOOT does, with the companion unlocked; the running button counts the seconds left
OpenPGP , in red – <i>Always press SIG / Cannot be undone</i>	Making a role's press permanent	as above
OpenPGP , in amber – <i>Set PIN tries / User 5, Admin 6, and Reset 4 beside it</i>	New retry limits were sent after the Admin PIN	Confirm sets them, each counter full; BOOT keeps the old ones
<i>Making SIG key, Signing with SIG, ... and a spinner</i>	After Confirm, while the card works – and while a key whose press is off is used	nothing; RSA-3072 generation takes about half a minute
OpenPGP , in green – <i>Signed with SIG, New SIG key made, ...</i>	The operation succeeded	nothing; it goes back to Ready by itself
User PIN set, Admin PIN set, Reset code set – <i>OpenPGP</i>	After a PIN change or unblock, or the reset code set by the Admin PIN	nothing; it goes by itself
OpenPGP , in green – <i>Reset code / Removed</i>	The reset code removed by the Admin PIN (GnuPG's <code>kdf-setup</code> does it)	nothing
OpenPGP , in green – <i>PIN hashing / Changed (or Off)</i>	GnuPG's <code>kdf-setup</code> finished: new salts, and both PINs are now the defaults	change the PINs
OpenPGP , in green – <i>Certificate stored / Certificate removed</i>	The cardholder certificate was written or deleted under the Admin PIN	nothing; it goes by itself

On the glass	When	Your answer
OpenPGP , in red – <i>Certificate refused / Not one DER cert, Not the login key's, Could not be saved</i>	A cardholder certificate write under the Admin PIN was refused, and why	nothing; the card kept what it had
OpenPGP , in red – <i>Wrong User PIN / 2 tries left (or Wrong Admin PIN, Wrong reset code)</i>	A password was presented and refused, by GnuPG or anything else	nothing; it goes by itself, and Activity records it
OpenPGP , in red – <i>User PIN blocked / Unblock: Admin PIN; Admin PIN blocked / Reset OpenPGP</i>	The last try was spent, or a blocked one was tried again	Unblock the User PIN with the Admin PIN or the reset code; a blocked Admin PIN takes a Full OpenPGP reset

In Activity every refused password is a row, whatever you chose to record: *Wrong OpenPGP PIN* with *User PIN*, *Admin PIN* or *Reset code* and the tries left; *OpenPGP PIN blocked* for the attempt that used the last try; and *OpenPGP PIN blocked / Attempt refused* for any attempt on a password already blocked – the card compared nothing, so even the right one is refused, and the journal does not call it wrong.

A request waits **30 seconds** for your answer; after that it is refused, the same as BOOT, and the computer gets an error rather than a hang. A request the screen cannot show is refused at once rather than left waiting. A key whose press is off asks nothing, but the glass still shows it in use and says when it succeeded; a refusal shows nothing.

The press proves **presence and intent for that operation**, not what exactly is being signed: the glass names the role and the operation, and the computer still chooses the data. Laboratory images can be set to approve by themselves (companion **Device** → **Behaviour** → **PGP auto-confirmation**); product firmware cannot.

Reading a prompt. The moving arcs from the Confirm button's corner are the only sign that Confirm answers yes; the footer never says so. It names what BOOT does – *NO* on every question – or says *PLEASE WAIT* while the card works. A prompt answered NO is recorded in Activity as *Declined*, one nobody answered as *Timed out*. A press remembered for 15 seconds is remembered for the card session it was made in, as the PIN is: another program opening the card starts without it.

When you need management unlock (HID Management)

Unlock in the companion opens a short-lived **management session** with the device PIN. That session is required for anything that goes over the management channel and changes card commissioning state:

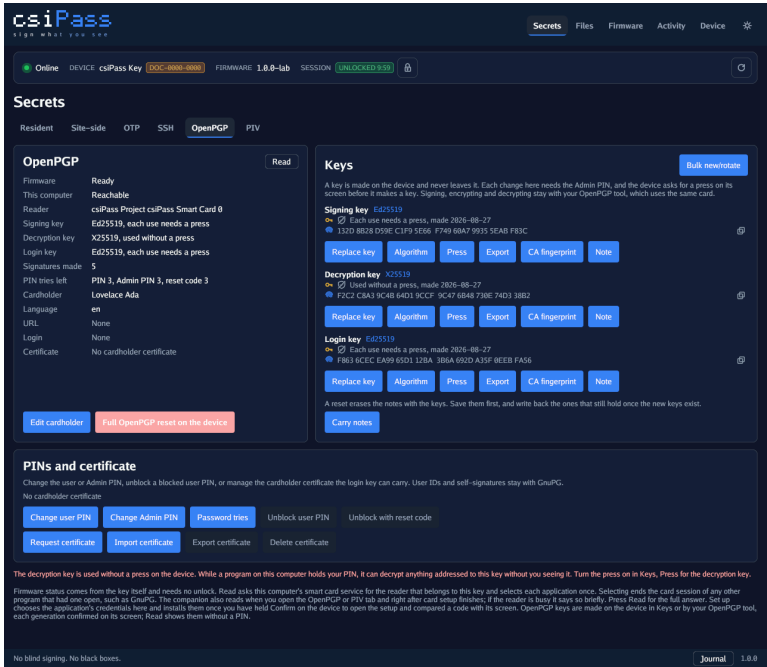
Action	Needs Unlock?	Why
Set up OpenPGP	Yes	Sealed commissioning over management HID; the glass arms and confirms
Full OpenPGP reset	Yes to press (shown while locked, enabled after Unlock)	Erases the OpenPGP application over management
Recover (damaged card journal)	Yes	Erases both card applications and the journal
Note on a role, Carry notes	Yes	Notes live in the device's management store, not on the card
Read, New key, Algorithm, Press, PIN changes, cardholder, certificate	No	These talk PC/SC; the card PINs (PW1/PW3) are what they need

You unlock so the companion may ask the device to **arm setup**, **commit credentials**, or **reset**. You still type PW3 or PW1 when the **card** itself requires them. Unlock does not replace OpenPGP PINs.

Companion: OpenPGP on Secrets

Open **Secrets** → **OpenPGP** (**Alt+G**). The tab is present when the connected image has the card reader.

The page at a glance



The OpenPGP tab with keys in all three roles

From the top:

- **The header** is the whole window's: the device (**csiPass Key** and its serial), the **firmware** it runs – a LAB card image ends in `-1ab` – and the **session**: *LOCKED*, or *UNLOCKED* with the minutes left.

Set up, a reset and notes need it unlocked; everything else on this tab does not.

- **Secrets** has one tab per kind of secret: *Resident*, *Site-side*, *OTP* and *SSH* are the FIDO lists, *OpenPGP* and *PIV* the two card applications.
- **The OpenPGP card** (left) is what the application is and what the card said at the last **Read**:

Row	What it says
Firmware	The application's state as the device reports it: <i>Ready</i> , <i>Ready, not set up yet</i> , <i>Reset in progress: finish it on the device</i> , <i>Damaged: ...</i>
This computer	What this computer's smart-card service found when it selected OpenPGP: <i>Reachable</i> , <i>Not found on the card</i> , <i>Present, refused (status ...)</i> , or <i>Not read</i> before any Read
Reader	The reader's name, as Windows or pcsc-lite lists it
Signing / Decryption / Login key	Each role's algorithm and press, or <i>No key</i>
Signatures made	The card's signature counter
PIN tries left	User PIN, Admin PIN and, once set, the reset code
Cardholder, Language, URL, Login	What GnuPG's <code>name</code> , <code>lang</code> , <code>url</code> and <code>login</code> wrote, or <i>None</i>
Certificate	The cardholder certificate, or <i>No cardholder certificate</i>

Under the rows: **Edit cardholder** and **Full OpenPGP reset on the device** once the application is set up, **Set up** while it is not.

- **Keys** (right) has one block per role – *Signing key*, *Decryption key*, *Login key* – each with its algorithm in blue, a line of glyphs (a gold **key** when the role holds one; the **press** glyph, struck through for *used without a press*; when it was made), and the key's **fingerprint** with a copy button on the right. Under each block are the role's buttons. **Bulk new/rotate** is in the heading, **Carry notes** at the foot.
- **PINs and certificate** (the full-width band) holds the password and certificate buttons. A button that cannot act now is greyed out – *Unblock user PIN* while the PIN is not blocked, *Export* and *Delete certificate* while there is none.
- **A red line under the bands** warns while the decryption key needs no press (below).
- **The hint** at the bottom says where each line comes from.

Overview and Read

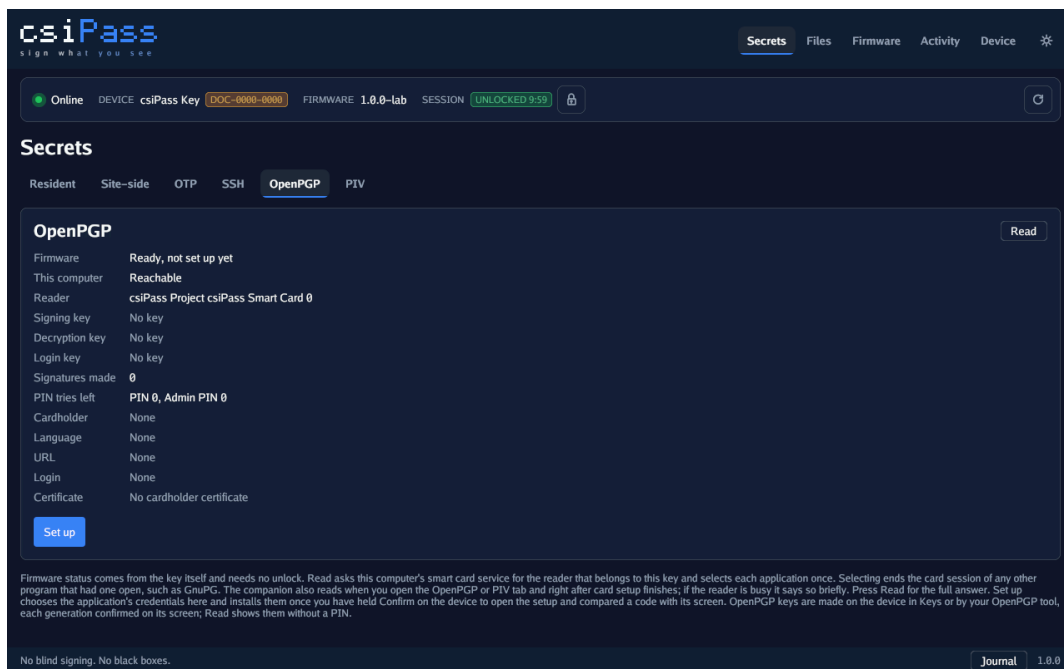
- Firmware readiness for the OpenPGP application comes from management GetInfo (no unlock required to see the tab on a connected key).
- **Read (F5 / Ctrl+R)** asks the host smart-card stack for **this** key's reader, by USB topology, and selects OpenPGP and PIV once. That is how the companion learns the keys, their fingerprints and

presses, the PIN tries, the signature count, the cardholder fields and whether a certificate is stored. Opening the OpenPGP tab and finishing **Set up** also start a Read.

- Selecting the application ends the card session of any other program that had it open, such as GnuPG. If another program holds the reader, the page says the reader is busy. Free it (`gpgconf --kill sdaemon`) and press **Read** again.
- If the decryption key needs no press, the page says so in red: a program that holds your User PIN can decrypt with it unseen.

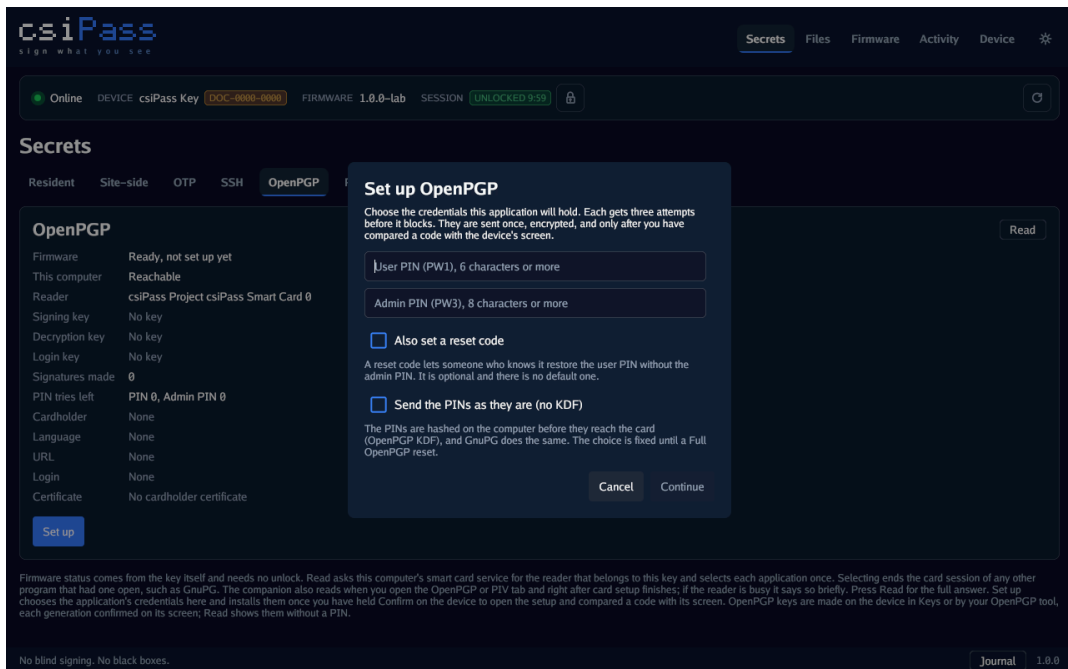
Set up (commissioning)

Set up appears when OpenPGP is present, ready and holds nothing yet, and a management session is open. Before it, the card shows *Ready, not set up yet*, no keys and no tries:



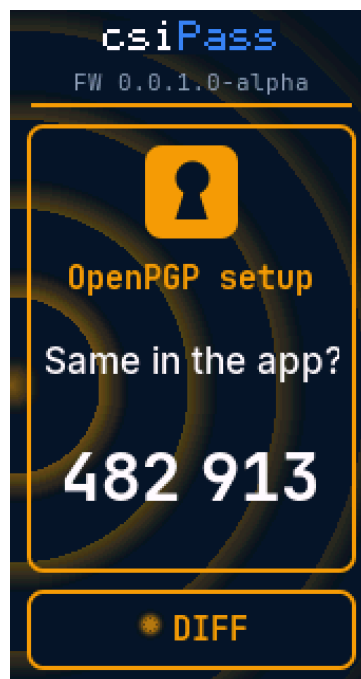
OpenPGP before Set up

1. Enter **User PIN (PW1)**, **Admin PIN (PW3)** and, if you want one, a **reset code** (tick *Also set a reset code*). *Send the PINs as they are (no KDF)* is off, and should stay off: with KDF on, the computer hashes each PIN before it reaches the card, GnuPG does the same, and the choice is fixed until a Full OpenPGP reset. **Continue** is live once the values pass the rules above (lengths; PW1 and PW3 differ). Nothing is sent yet.



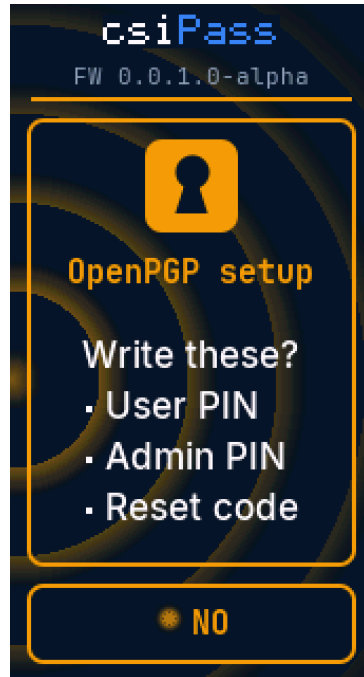
The Set up OpenPGP dialog

- The companion asks the device to open setup. The glass shows **Hold to set up** with *to set up OpenPGP*. **Hold Confirm for five seconds** to arm a 120-second window. Only the device can arm it; nothing the companion sends extends it.
- Both screens show a **six-digit code** in two groups of three. Compare them. Answer **They match** in the companion and **SAME** (Confirm) on the glass – either order. If they differ, answer **They differ / DIFF** (BOOT) and stop; something between this computer and the device is answering for it.



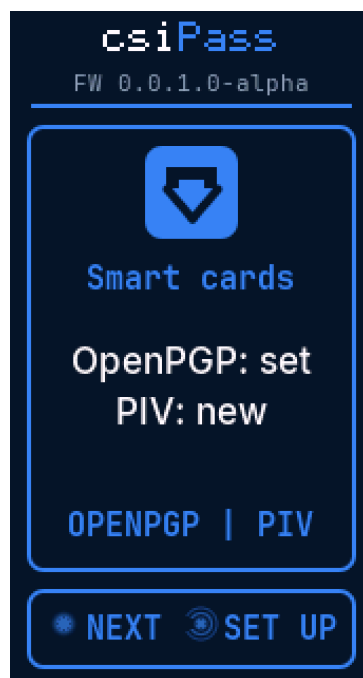
The comparison code on the glass

- The device shows what it will install – *Write these?* and the list – and waits for **Confirm**; **NO** (BOOT) writes nothing. Only then are the credentials written.

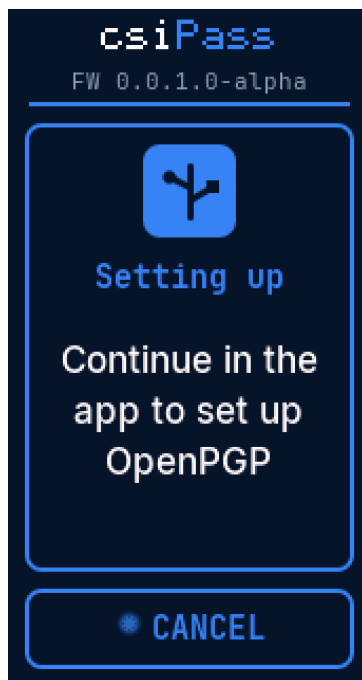


What the device is about to write

You can also start from the device: on the glass, open **Smart cards** and hold **BOOT** while the footer says **SET UP**. The device sets up the first application that has nothing yet and then waits for the companion, which must be unlocked to continue; *Unlock first* on the glass means it is not.



Smart cards on the glass, and the wait for the companion



Setting up, continued in the companion

After success the companion reads the card at once, and the Keys band appears. Make keys there (or with GnuPG, below).

Why so many Confirm presses during Set up?

Each press answers a **different** question. They are not duplicates of one approval:

Glass gesture	What it proves
Hold Confirm 5 s (<i>Hold to set up</i>)	You are physically at this device and you want a setup window opened. The computer cannot invent that window.
Confirm = SAME (six-digit compare)	The digits on the glass match the companion. That binds the USB session to the device you are looking at – not a lookalike answering for it.
Confirm (commit / install)	You accept writing the PINs (and optional reset code) into OpenPGP storage. Until this press, nothing durable was installed.

Later, **each New key** and a permanent press ask for Confirm again: generating a private key, replacing one, or making a press requirement permanent is another irreversible (or hard-to-undo) act. A press means **presence and intent for that operation**, not “I already said yes once at setup.” The device does not reuse an old Confirm for a new request.

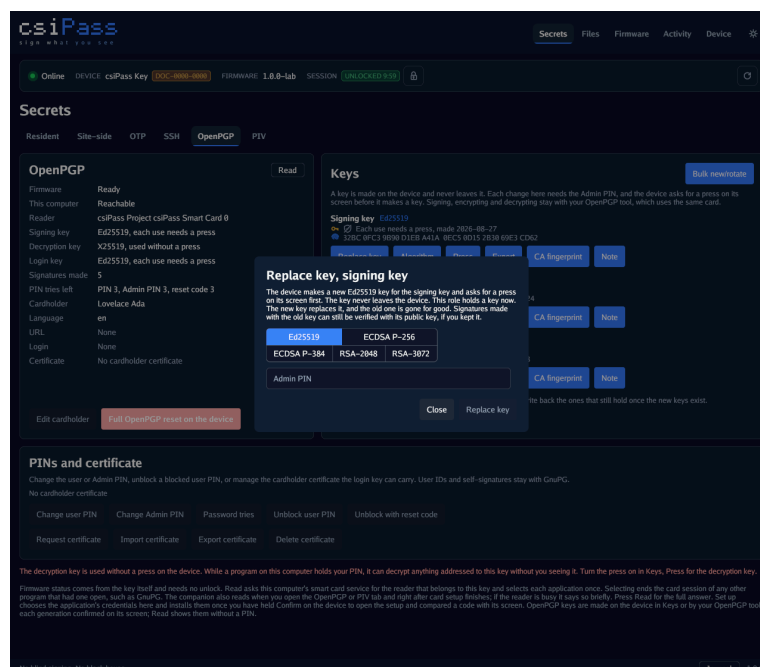
If the companion loses the answer to the sealed install, it does **not** send the envelope again. Use **Check the device** / **Read** to see what landed.

Keys

Keys lists the three roles with each one’s algorithm, whether it needs a press, when its key was made and its fingerprint (with a copy control for comparing with `gpg --card-status`). A fingerprint that does not match the one the card recorded is marked. For each role:

Control	What it does	Glass / PIN
New key (Replace key over a key the role holds)	Chooses an algorithm FA offers (preselected to the current one), writes the attributes if they changed, then GENERATE; records the fingerprint and creation time GnuPG needs	Confirm on glass; Admin PIN
Algorithm	Same path as New key with a different algorithm	Confirm; Admin PIN
Press	Sets the key's UIF: No press , Press always , or Press once in 15 s (uses in quick succession ask once); tick Permanent to make it impossible to change short of a Full OpenPGP reset. A new key has a press on every use for signing and login, none for decryption	Confirm for permanent; Admin PIN
Export	ASCII-armored public key (.asc), no User ID packet	No PIN
CA fingerprint	Sets or clears the recorded CA fingerprint for the role	Admin PIN
Note	Your note about the role's key, kept on the device; a new key starts without one	Management Unlock
Carry notes	Saves this application's notes to a file before a reset, and writes back the ones that still describe a key after it	Management Unlock

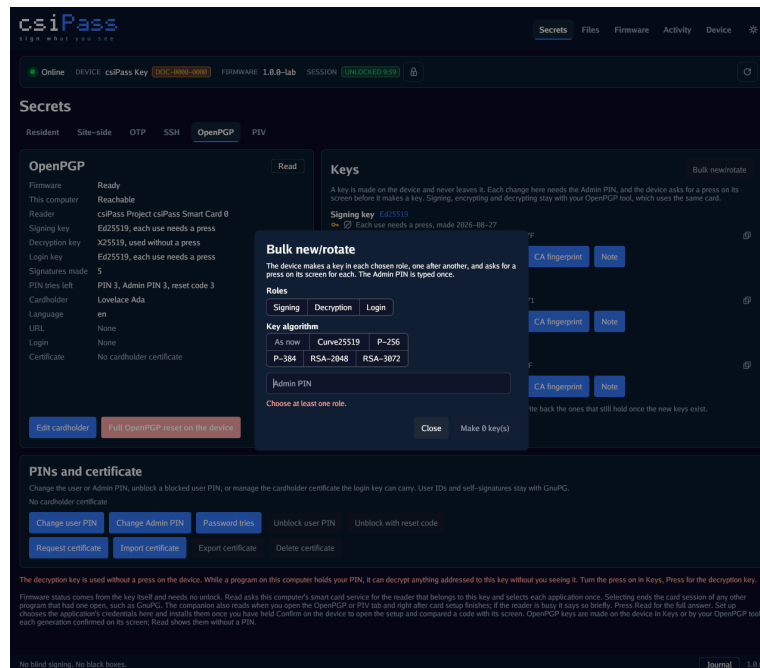
New key (or Replace key) opens a short window: what the device is about to make, the algorithms the card offers with the role's current one chosen, and the **Admin PIN**. The button makes the key; the glass then asks, and the old key is gone once you Confirm there.



Replace key for the signing role

Bulk new/rotate, in the Keys heading, makes keys in several roles with the Admin PIN typed once. The roles with no key are chosen when it opens; with a key in every role none is, so a replacement is always chosen by name. The algorithm is one choice for all of them: **As now** makes each new key in the algorithm of the key it replaces – offered only when every chosen role holds a key, and then taken by

itself while no other algorithm is chosen; with an empty role among them it is greyed out, its tip says why, and the algorithm has to be named – and **Curve25519, P-256, P-384, RSA-2048, RSA-3072** give the decryption key the family's agreement algorithm (X25519, ECDH) and the other two its signing one (Ed25519, ECDSA); a family the card does not offer in every chosen role is greyed out. The roles run one after another, each its own transaction with its own **Confirm** on the glass, and the panel says which one is running. The run stops at the first failure that is not the role's own – a wrong Admin PIN (so it is not tried again on the next role), a press declined or cancelled, a busy reader, or a lost answer – and marks the rest *not started*. **Save public keys** then writes one `.asc` per new key to a folder you choose.



Bulk new/rotate with no role chosen yet

The window says what is missing in red under the fields – *Choose at least one role*, an algorithm, the Admin PIN – and its button counts the keys it will make (*Make 2 key(s)*).

CA fingerprint takes the fingerprint of an OpenPGP key: the one that certifies this role's key, such as your own primary key or your organisation's. The LAB attestation CA (`csipass-lab-ca`) is an X.509 certificate and cannot be used here. For a bench card, `deploy/new-lab-openpgp-ca.ps1` makes a throwaway OpenPGP CA key and prints its fingerprint (`deploy/README.md` , *LAB OpenPGP CA*).

A key already in the role is **gone** when GENERATE succeeds. Old ciphertext to a replaced decryption key cannot be opened. Old signatures still verify with the old public key if you kept it.

Cardholder, PINs, certificate

Control	Purpose	Auth
Edit cardholder	Name, language, salutation, URL, login data; an empty field clears it	Admin PIN
Change user PIN / Change Admin PIN	Replace PW1 or PW3	The current value
Unblock user PIN	New PW1 after the Admin PIN	Admin PIN
Unblock with reset code	New PW1 with the reset code	Reset code
Password tries	How many wrong tries block PW1, the reset code and PW3, 1 to 15 each. The values stay; each counter starts full. A reset code that is not set keeps no limit	Admin PIN; Confirm on the glass
Request certificate	PKCS#10 request signed on the card by the login key; save the PEM for a CA	User PIN ; Confirm if AUT's press is on
Import / Export / Delete certificate	The cardholder certificate (7F21 , one DER, at most 2,560 bytes); it must name the login key when one is present	Admin PIN to import or delete; nothing to export

The unblock buttons are live only while PW1 is blocked.

Full OpenPGP reset

Erases that application's PINs, keys, certificate and notes. The glass shows **Hold to reset – to erase OpenPGP from this card**; hold Confirm five seconds. Afterwards the application is empty and ready for **Set up** again – a new life, not a repair: the old fingerprints are gone for good. Save your notes with **Carry notes** first if you want them.

If a reset was interrupted – a power cut in the middle – the application is left half erased and unavailable, and the companion says *Reset in progress: finish it on the device*. The device never finishes it by itself: on the glass, open **Smart cards**, hold **BOOT** while the footer says **FINISH**, and Confirm the red **Finish reset** offer. (**Recover**, which erases both applications, is the one that carries on by itself after a restart, under *Completing reset*.)

Setting up GnuPG on the computer

GnuPG reaches the card through `scdaemon`, its smart-card daemon, which `gpg-agent` starts when something needs the card. Two lines of its configuration decide whether it and the companion can share the reader.

Windows

1. **Install GnuPG** – GnuPG's own Windows installer or Gpg4win. The board runs used GnuPG 2.5.22, installed in `C:\Program Files\GnuPG`; anything from 2.3 on reads `csiPass` the same way.

2. **Tell `scdaemon` to use the Windows smart card service, shared.** In

`%APPDATA%\gnupg\scdaemon.conf` (create it if it is not there):

```
disable-ccid
pcsc-shared
```

`disable-ccid` stops `scdaemon` driving the USB reader itself, which would take it away from Windows, PIV middleware and the companion. `pcsc-shared` opens the reader shared rather than exclusively: without it the companion's **Read** reports the reader busy for as long as `scdaemon` runs.

3. **The PIN dialog.** `gpg-agent` asks for PINs through a *pinentry* program, named in

`%APPDATA%\gnupg\gpg-agent.conf`:

```
pinentry-program "C:\Program Files\GnuPG\bin\pinentry-basic.exe"
```

The window can open behind the terminal; when `gpg` seems to hang, look for it with Alt+Tab.

4. **Apply** – `gpgconf --kill all` – and check the card:

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

The first lines should read *Reader: csiPass Project csiPass Smart Card 0, Application type: OpenPGP, Manufacturer: unmanaged S/N range* (the LAB image's test identity, `0xFF00`). *Card not present* or *No such device* means the image has no reader (`esp32s3-lab`), the application image (which GnuPG does not recognise yet), or another program holding the reader without sharing.

Linux

The Linux side has been run on a board through `pcsc-lite` (test protocol 23b, H2); GnuPG itself has not.

1. Install `gnupg`, `scdaemon`, `pcscd` and `libccid` (package names differ slightly between distributions).
2. Teach the CCID driver the development reader: add `0xCAFE / 0x4014 / csiPass Project csiPass Smart Card` to the `ifdVendorID`, `ifdProductID` and `ifdFriendlyName` arrays of `/usr/lib/pcsc/drivers/ifd-ccid.bundle/Contents/Info.plist` (the three at the same position) and restart `pcscd` – the [PIV Manual](#), *Linux*, has the details, containers and WSL 2 included.
3. `~/ .gnupg/scdaemon.conf` with the same two lines, `disable-ccid` and `pcsc-shared`; then `gpgconf --kill all` and `gpg --card-status`.

From an empty card to a working key, on the command line

The whole path with GnuPG, the companion used only where nothing else can do it. The User PIN is `PW1`, the Admin PIN `PW3`; GnuPG calls them *PIN* and *Admin PIN*.

1. **Set OpenPGP up in the companion** (*Set up*, above). There is no command for it: the PINs travel sealed over management and the device confirms them on the glass.

2. **Free the reader and look:**

```
gpgconf --kill sddaemon
gpg --card-status
```

Key attributes shows what the roles will generate (`ed25519 cv25519 ed25519` on a fresh card), *PIN retry counter* `3 0 3` (User PIN, reset code – none set – and Admin PIN), *KDF setting*: on.

3. **Cardholder data (optional)**, under the Admin PIN:

```
gpg --card-edit
gpg/card> admin
gpg/card> name          # surname, then given name
gpg/card> lang          # e.g. en
gpg/card> login         # the account name, if a service wants one
gpg/card> url           # where your public key can be fetched
gpg/card> quit
```

4. **The algorithms**, if not the defaults – still in `gpg --card-edit`, `admin: key-attr` asks, role by role, *RSA* or *ECC* and then the size or the curve. Each change needs the Admin PIN. The card lists what it makes; anything else is refused.

5. **Make the keys on the card** and an identity around them:

```
gpg --card-edit
gpg/card> admin
gpg/card> generate
```

Answer **no** to *Make off-card backup of encryption key?* – *csiPass* keeps private keys on the device – and **yes** to *Replace existing keys?* if the card holds any. Give an expiry, a name and an e-mail address. The glass asks **Confirm** for each key, and *RSA* takes some seconds after it. *GnuPG* then signs the identity with the new signing key: the User PIN, and a press while *SIG's* press is on.

6. **Check** that the keyring points at the card:

```
gpg -K --with-keygrip you@example.test
```

`sec>` and `ssb>` – the `>` – mean the secret part is on a card, and the *card serial number* line names it (`FF00 xxxxxxxx` on a LAB image). The keygrips are what *SSH* needs below.

7. **The presses**, if not the defaults (`gpg --card-edit`, `admin`, `uif 1` on, ... – *Presses and the signature PIN*, below).

8. **Publish the public key** – only the public part exists off the card:

```
gpg --armor --export you@example.test > you.asc
```

9. **Use it** – sign, encrypt and decrypt, SSH and Git, below.

To add or replace one role's key later, keeping the identity: `gpg --card-edit` → `admin` → `key-attr` if the algorithm changes, then `gpg --edit-key you@example.test` → `addcardkey` → the role → `save`. With the signature PIN forced (the default) GnuPG asks the User PIN once more after the card has made the key, to certify it with SIG.

The same work with GnuPG

The companion never runs `gpg`. These are the usual GnuPG equivalents once the card is commissioned and visible to the host (a lab card image, see *Which firmware carries OpenPGP*).

Free the reader before the companion **Reads**:

```
gpgconf --kill sddaemon
```

Kill it too **after a Full OpenPGP reset or a new Set up**, and after flashing firmware. `sddaemon` keeps what it read in memory: after a reset it goes on showing the old card's serial number, keys and press settings beside fresh PIN counters, with no error (test protocol 23b, H3), and across a flash it can keep a reader handle that no longer exists and print nonsense. A key in your keyring made on the card before a reset points at keys that are gone, and `gpg` then waits for "the card" forever – in `--pinentry-mode loopback` with no message at all. Delete that key (`gpg --delete-secret-and-public-keys <fingerprint>`; its secret part is only a pointer to the card) and bind the new card keys, below.

One failed GnuPG command after Windows used the card. Windows resets the card when it has finished with a PIV PIN – after a smart-card logon, `certutil -scinfo`, a browser's client certificate – so that the PIN it verified does not stay verified. `sddaemon`'s connection then reports the reset, and the next GnuPG command fails once – *OpenPGP card not available: General error* from `--card-status`, *signing failed: Card error* from a signature – before it asks for a PIN or reaches the card; the one after it reconnects and works. Run it again – nothing on the card changed, and Activity has no row for the failed one. A tool can end more than one PIN session: `certutil -scinfo` verifies once per cryptographic provider, so it resets twice and can cost two such failures. Seen on the board in test protocol 24a, C3: a PC/SC observer recorded `SCARD_W_RESET_CARD` in the middle of `certutil -scinfo`, which asked for the PIN again after it.

One Card error after another tool used PIV. The card has one selected application for everyone, and it cannot tell which program sent a command. A tool that selects PIV and leaves it selected – `pkcs15-tool`, for one – leaves `sddaemon` (with `pcsc-shared`, which the companion needs) sending its next OpenPGP command to PIV; PIV refuses it, and GnuPG reports *signing failed: Card error* without asking for a PIN. The next GnuPG command selects OpenPGP again and works. The companion's **Read** selects OpenPGP last and does not cause it. Any multi-application card behaves so; it is a limit of sharing the reader, not a fault. Test protocol 24a, C4: no reset seen, and a signature after something reselected OpenPGP worked the first time.

Inspect the card

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

Compare fingerprints with the companion **Keys** card (copy control beside each fingerprint). *Signature counter* is the card's count of signatures made.

Change or unblock PINs

```
gpg --change-pin
```

Or interactively:

```
gpg --card-edit  
> admin  
> passwd
```

`passwd` offers changing PW1, unblocking it with PW3, changing PW3 and setting the reset code. The companion's buttons do the same. Setting or removing the reset code this way (PUT DATA `D3`) works from firmware of 2026-09-26; earlier firmware answered it *not found*.

With KDF on – the default – GnuPG sends the card a hash of each PIN, never the PIN, and hashes under the salts the card publishes. A card set up before 2026-09-26 published an object GnuPG did not recognise as KDF, so GnuPG sent PINs as typed and the card refused them: set such a card up again.

Password tries have no GnuPG command: use the companion's **Password tries**, and the glass asks in amber. `ykman openpgp access set-retries 5 4 6` sends the same command (SET PIN RETRIES) in YubiKey's form, which `csiPass` takes, but `ykman` looks for a YubiKey and has not been run against `csiPass`; test protocol 23i sent the command as a raw APDU.

Generate keys on the card (instead of companion New key)

Prefer recognising a key the companion already made:

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

Choose **Existing key from card** (GnuPG 2.3+). Without questions, for keys already on the card in all three roles:

```
gpg --quick-generate-key "Your Name <you@example.test>" card
```

It makes three self-signatures with the signing key, so the glass asks for each while SIG's press is on (one press while a cached one is remembered). Do **not** use `generate` inside `gpg --card-edit` if the companion already created the key – that would mint another key and orphan the recorded fingerprint. These two recognise keys the standard GnuPG way and have not been run against a board; the paths that have are `generate` below (test protocol 23g, G8) and `addcardkey`, which makes a key in one role and adds it to an existing identity (G4, G8).

If you generate only with GnuPG:

```
gpg --card-edit
> admin
> key-attr
> generate
```

Confirm on the glass for each key (GENERATE always asks on csiPass). GnuPG's *make off-card backup* question must be answered **no**: csiPass never lets a private key leave the device.

Presses and the signature PIN

```
gpg --card-edit
> admin
> uif 1 on          # SIG asks for Confirm; 2 is DEC, 3 is AUT
> uif 2 permanent  # cannot be turned off again
```

GnuPG sets on, off and permanent. The cached flags – a press remembered for 15 seconds, within the session – come from the companion's **Press** (Press once in 15 s). They are YubiKey's values (03, and 04 permanent), so `ykman openpgp keys set-touch sig cached (cached-fixed)` sends them too – the same caveat as above: not run against csiPass, where 23i wrote them as raw APDUs. `gpg --card-status` shows *UIF setting* as on or off only, so a cached flag reads *on* (`--with-colons` gives `uif:1:0:1:`); the companion and Activity tell cached from always. Firmware before 2026-09-26 did not say it had a button (DO 7F74), and GnuPG then showed no UIF line and refused `uif`.

```
> forcesig          # toggles: PW1 once per signature, or once per session
```

The card starts with **PW1 valid for one signature** (`--card-status` says *forced*): GnuPG presents the PIN again for every signature, from `gpg-agent`'s cache if it has one. A permanent press asks for Confirm on the glass before the card agrees to it.

Sign / encrypt / decrypt (everyday use)

After the public key has a User ID and is in your keyring:

```
gpg --clearsign message.txt
gpg --encrypt --recipient YOU@example.test secret.txt
gpg --decrypt secret.txt.gpg
```

The card performs the private operation; GnuPG builds the OpenPGP packets. Enter the **User PIN** when `sddaemon` asks. If the role's press is on, Confirm on the glass as well. On Windows GnuPG's PIN dialog can open behind the terminal: when `gpg` seems to hang after the glass has finished, look for a *Pinentry* window. RSA decryption keys follow the standard wire format and were corrected on 2026-09-23; firmware before that refused RSA decryption. Decryption with RSA-2048 and RSA-3072 keys made on the card was checked with GnuPG on 2026-09-29 (a new decryption key beside an identity: *From an empty card to a working key*, the last paragraph).

SSH with the authentication key

`gpg-agent` can be an SSH agent that signs with the login (AUT) key.

1. Add `enable-ssh-support` to `gpg-agent.conf` and `gpgconf --kill gpg-agent`.
2. Tell the agent which key to offer: the AUT subkey's keygrip (`gpg -K --with-keygrip`, the [A] line) goes on a line of its own in `sshcontrol`, in the same directory (`%APPDATA%\gnupg\sshcontrol` on Windows):

```
0123456789ABCDEF0123456789ABCDEF01234567 0
```

3. Check that the agent offers it, and put the public key on the server:

```
gpg-connect-agent "keyinfo --ssh-list" /bye          # the card key, OPENPGP.3
gpg --export-ssh-key YOU@example.test >> ~/.ssh/authorized_keys # on the server
```

4. Point SSH at the agent. On Linux and macOS: `export SSH_AUTH_SOCK=$(gpgconf --list-dirs agent-ssh-socket)`, then `ssh-add -L` lists the key and `ssh user@server` uses it. On Windows the inbox OpenSSH client talks to its own agent's pipe, not to `gpg-agent`: GnuPG 2.4 and later serve that pipe with `enable-win32-openssh-support` in `gpg-agent.conf`, once the Windows *OpenSSH Authentication Agent* service is stopped; PuTTY is served with `enable-putty-support`.

What has been run on the board is the agent's side, with the companion closed: the key listed by `keyinfo --ssh-list`, and a challenge signed through the agent with a press on the glass, Ed25519 (test protocol 23f) and RSA-2048 over SSH's `rsa-sha2-512` hash (23g, G9) – `gpg-connect-agent "SIGKEY <keygrip>" "SETHASH --hash=sha512 <hex>" PKSIGN /bye`. An `ssh` login to a server and the Windows OpenSSH pipe have not been.

Confirm on the glass when AUT's press is on. The card side is INTERNAL AUTHENTICATE with PW1 (reference 82). With an RSA login key the card signs at most 40 % of the modulus (102 bytes for RSA-2048, 153 for RSA-3072), which is more than SSH ever sends: its longest, `rsa-sha2-512`, is 83 bytes, and an RSA-2048 login key signed exactly that on a board (2026-09-29).

Signed Git commits

Git signs with `gpg`, so the card signs each commit, with SIG's press:

```
git config --global user.signingkey <fingerprint of the signing key>
git config --global commit.gpgsign true
git config --global gpg.program "C:\Program Files\GnuPG\bin\gpg.exe" # Windows
```

`git commit -S` asks the User PIN (again for every commit while the signature PIN is forced) and a Confirm on the glass. This is ordinary GnuPG signing, the path `gpg --clearsign` took on the board (23g, G2-G3); Git itself was not part of that run.

Cardholder certificate

Companion **Request certificate** produces a PEM CSR signed by the login key. With GnuPG alone you would use another PKI tool against the AUT key, then store the issued DER. Import it in the companion

(Admin PIN) or with a card tool that writes DO 7F21. On a bench card, `deploy/sign-lab-openpgp-request.ps1` issues that DER from the CSR with the LAB OpenPGP CA key (`deploy/README.md` , *LAB OpenPGP CA*); `csipass-lab-ca issue` does it without a CSR, from the key's attestation, in the same pass as the PIV certificates ([LAB certificates in one pass](#)).

What GnuPG offers that csiPass refuses

GnuPG	On csiPass
<code>keytocard</code> (move an existing private key onto the card)	Refused: keys are made on the device and never imported
<code>kdf-setup</code>	Not needed: the companion's Set up chooses KDF (on by default); <code>gpg --card-status</code> shows <i>KDF setting: on</i> and GnuPG hashes the PINs itself. It works if run, and does what GnuPG says it does: new salts, both PINs back to the defaults (123456 and 12345678), the reset code removed; the glass says <i>OpenPGP – PIN hashing / Changed</i> . Change the PINs afterwards
<code>factory-reset</code> (TERMINATE / ACTIVATE)	Not supported: use Full OpenPGP reset from the companion or the device
Key attestation	Supported in csiPass's own form, not YubiKey's: <code>00 FB <role></code> answers a certificate for the role's key signed by the card's attestation key once that key is certified (a LAB image with <code>csipass-lab-ca certify</code> ; <code>csipass-lab-ca verify</code> checks it), and <code>6D00</code> until then. The certificate is the answer and is stored nowhere, so <code>ykman</code> 's attestation commands do not read it (D11, K2)
Private-use data objects (<code>privatedo</code>)	Not supported

After the profile is set up: everyday use

A finished profile means: OpenPGP is commissioned, at least the roles you need hold keys, fingerprints match what GnuPG sees, and you know your PW1 / PW3 (and reset code if you set one).

1. **Plug the key in.** The host enumerates FIDO and the smart-card reader.
2. **Do not leave `scdaemon` and the companion fighting over the reader.** Use one owner at a time.
3. **Mail and files** – encrypt and sign with GnuPG (or another OpenPGP client) as usual; the private operation happens on the card.
4. **SSH / login** – use the authentication key through your agent stack; Confirm on the glass when AUT's press is on.
5. **Decryption without a press** – the default on csiPass. If that worries you, **Press** on the decryption role (or make it permanent).
6. **Backup of public material** – export `.asc` public keys and keep User IDs and certifications in GnuPG's keyring or a printed fingerprint sheet. Private keys cannot be backed up off the card; a lost or reset device means those private keys are gone. Plan revocation certificates accordingly.

7. **Rotation – New key** or **Algorithm** replaces material on the card. Re-publish the new public key; re-encrypt data that only the old DEC key could open, if you still have plaintext.
 8. **Blocked User PIN** – unblock with the Admin PIN or the reset code in the companion, or `gpg --change-pin`. Three failures block each PIN; the Admin PIN has its own counter, and a blocked Admin PIN means a Full OpenPGP reset.
 9. **Wipe OpenPGP only – Full OpenPGP reset**, then Set up again. FIDO passkeys and PIV are separate and stay unless you erase them.
-

Quick map: companion control ↔ glass ↔ gpg

Goal	Companion	Glass	GnuPG alternative
Commission PINs	Set up (Unlocked)	Hold 5 s → SAME → Confirm	Not via <code>gpg</code> ; the card must be commissioned first
See keys / tries	Read	—	<code>gpg --card-status</code>
Make a key	New key / Algorithm	Confirm (red over an old key)	<code>gpg --card-edit</code> → <code>generate</code> (or Existing key from card)
Set the press	Press	Confirm if permanent	<code>uif N on</code> / <code>uif N off</code> / <code>uif N permanent</code> (cached: the companion)
Signature PIN policy	—	—	<code>forcesig</code>
Export public key	Export	—	<code>gpg --export --armor KEYID</code> after the key is in the keyring
Change PW1 / PW3	Change user / Admin PIN	<i>PIN set</i> notice	<code>gpg --change-pin</code>
Unblock PW1	Unblock user PIN / with reset code	<i>User PIN set</i>	<code>gpg --change-pin</code> → unblock
Set the tries	Password tries	OpenPGP , in amber: <i>Set PIN tries</i> ; then <i>PIN tries set</i>	No GnuPG command; <code>ykman openpgp access set-retries</code> in form (not run against <code>csiPass</code>)
Sign / decrypt mail	— (use GnuPG)	Confirm if the press is on	<code>gpg --sign</code> / <code>--encrypt</code> / <code>--decrypt</code>
SSH login	—	<i>Log in with AUT</i> if its press is on	<code>enable-ssh-support</code> , the AUT keygrip in <code>sshcontrol</code> , <code>gpg --export-ssh-key</code>
Sign Git commits	—	<i>Sign with SIG</i>	<code>git config user.signingkey</code> , <code>commit.gpgsign true</code>
Cardholder data	Edit cardholder	—	<code>gpg --card-edit</code> → <code>admin</code> → <code>name</code> / <code>lang</code> / <code>login</code> / <code>url</code>
Login-key CSR	Request certificate	Confirm if AUT's press is on	External CSR tool + AUT
Keep notes across a reset	Carry notes	—	—
Wipe OpenPGP	Full OpenPGP reset	Hold Confirm 5 s	Not supported (<code>factory-reset</code>)

Related documents

Document	Role
User Manual	FIDO everyday use and glass grammar
Companion Manual	Whole companion, including Smart cards
PIV Manual	The other card application on the same reader
<code>docs/stubs/pgp-piv.md</code>	M15/M16 plan and acceptance gates
<code>docs/openpgp-piv/openpgp-key-objects.md</code>	Commands, UIF, PW1 modes (implementers)
<code>docs/openpgp-piv/commissioning-channel.md</code>	Why Set up is sealed and compared on glass
<code>docs/conformance.md</code>	Which OpenPGP Card clauses are implemented, and how each was checked