

csiPass PIV Manual

This guide is for someone who has a csiPass device with a smart-card reader (CCID) and wants to use its **PIV** application: what PIV is, what each of its four key slots is for and what it asks of you, how setup and the administration key work, how the companion and the glass present PIV, how standard middleware reaches it, and how to keep keys, certificates and their history once they exist.

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

Prototype caution. PIV on csiPass is laboratory tooling on development images. What this guide describes has been run on a board: OpenSC and the inbox Windows driver, browser client certificates, historical decryption after a 9D rotation, administration with OpenSC alone, a blocked administration key and the reset after it (test protocols 23d and 23h), certificates issued from each key's attestation (23j), and logon to a Windows domain over Remote Desktop (24c) – a console logon on a domain-joined PC has not been. Use a LAB card image (`esp32s3-lab-card`), test certificates and test accounts, and do not make this the only place a key you care about exists.

What PIV is

PIV (Personal Identity Verification, NIST SP 800-73) is the smart-card profile behind US government ID cards, and the one most operating systems and PKI tools speak without extra drivers. A PIV card holds a few private keys, each in a numbered **slot** with a fixed purpose and a fixed access rule, and the X.509 **certificate** a certificate authority issued for each key. The computer finds a key through its certificate, sends the card a digest or a wrapped secret, and gets back a signature or the unwrapped value. The private keys never leave the card.

People meet it as Windows smart-card logon and certificate selection, as browser TLS client certificates, as S/MIME signing and decryption in mail clients, as document signing, and as SSH keys through a PKCS#11 module (OpenSC). csiPass implements that card application beside FIDO and OpenPGP, over its separate USB smart-card interface.

The four slots

Slot	Name	Typical use	Needs	On csiPass
9A	PIV Authentication	Logon, VPN, TLS client certificates, SSH	The PIN, once per session	PIN session
9C	Digital Signature	Signing documents and mail	The PIN immediately before every signature	PIN always
9D	Key Management	Decrypting mail, key agreement	The PIN, once per session	PIN session ; replaced keys move to an on-card history
9E	Card Authentication	Door readers, a card that must answer without a PIN	Nothing	No PIN

“PIN always” is not a button: every 9C signature needs its own PIN presentation, and csiPass adds no press to using any key. A 9C signature does not end the PIN session the other slots use.

When 9D’s key is replaced while its certificate is on the card, the old key and its certificate move to one of **twenty retired slots** (82 - 95). Mail encrypted to an old key years ago still decrypts, and middleware finds those keys through the card’s **Key History**. When all twenty are taken the card refuses a new 9D rather than discard the oldest, and the companion says so: 9D keeps its key and certificate, and only a PIV reset makes room.

Algorithms

csiPass makes **ECC P-256** (ECDSA in 9A, 9C and 9E, ECDH in 9D), **RSA-2048** and **RSA-3072** – the algorithms with a recorded measurement on this board. The card lists them in its answer to SELECT, and anything else is refused. **ECC P-384** is built in and available now that its measurement is recorded; the companion offers it when the card lists it. Keys are generated **on the device**; there is no import of a private key and no export of one. An RSA-3072 key takes about half a minute to make.

Three secrets, and what each is for

Secret	Format	What it does	If it is lost or blocked
PIN	6 to 8 digits	Opens 9A, 9C and 9D for use	Unblock it with the PUK
PUK	Exactly 8 characters, as SP 800-73 has it; a PUK set shorter before 2026-09-26 still works, but a new one is 8	Unblocks the PIN, nothing else	Only a Full PIV reset , which erases the keys
Administration key	AES-256, made at setup	Makes keys, stores and deletes certificates, writes the card’s data	Only a Full PIV reset , which erases the keys

- The PIN and the PUK must differ. Each has **3 tries** until you set others (**Set PIN tries**, 1 to 15 each), and the administration key has 3; a wrong attempt costs one, and a right one restores them

all.

- The device PIN you unlock the companion with is a different secret, for the device's management – it is none of these.
- The companion never keeps any of them after an operation finishes, and the device never shows them.

The administration key is not something you type. The companion generates it at setup, installs it on the card, and hands you its **only copy** in an encrypted backup file. Every change to the card's keys and certificates opens that file with its passphrase, for that one operation.

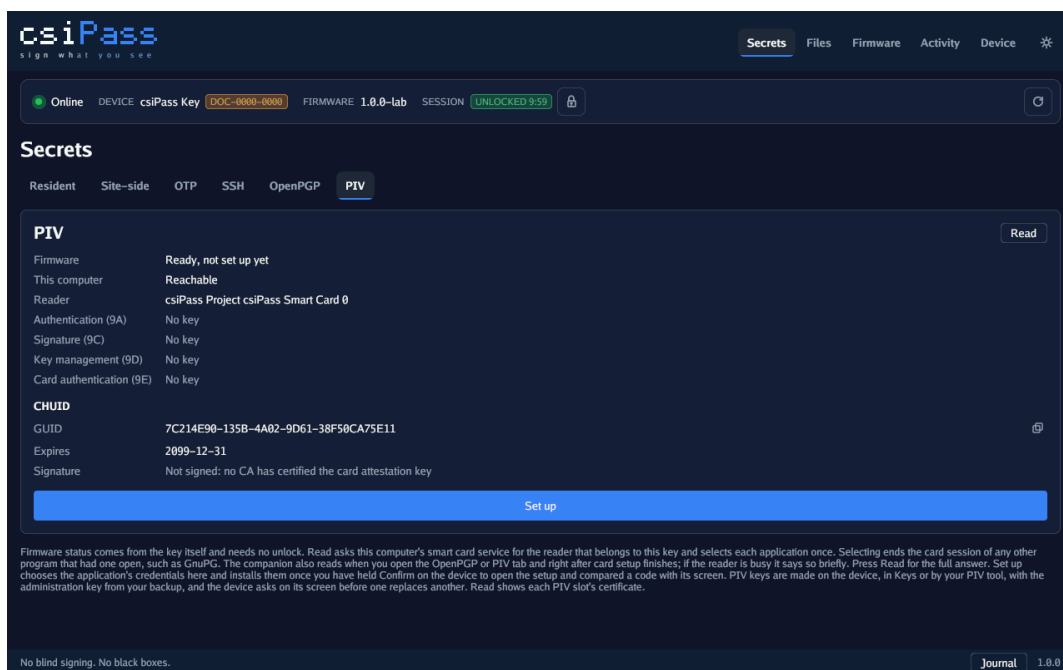
Which firmware carries PIV

The smart-card reader is part of the ordinary application image and of the **lab card** image (`esp32s3-1ab-card`). The ordinary **lab** image (`esp32s3-1ab`) keeps a USB serial log in its place and has no reader, so the PIV and OpenPGP tabs are absent there.

The **CHUID** – the object a PIV card identifies itself with – is answered by every image, in the form a non-federal (PIV-I) issuer uses: the FASC-N of all nines, a GUID that changes with every PIV reset, an expiry of 2099-12-31, and an issuer signature by the card attestation key once a CA has certified that key (empty until then). Windows smart-card logon keys its certificate cache by that GUID. The Card Capability Container, Discovery object and Key History are there on every image too.

Setting PIV up

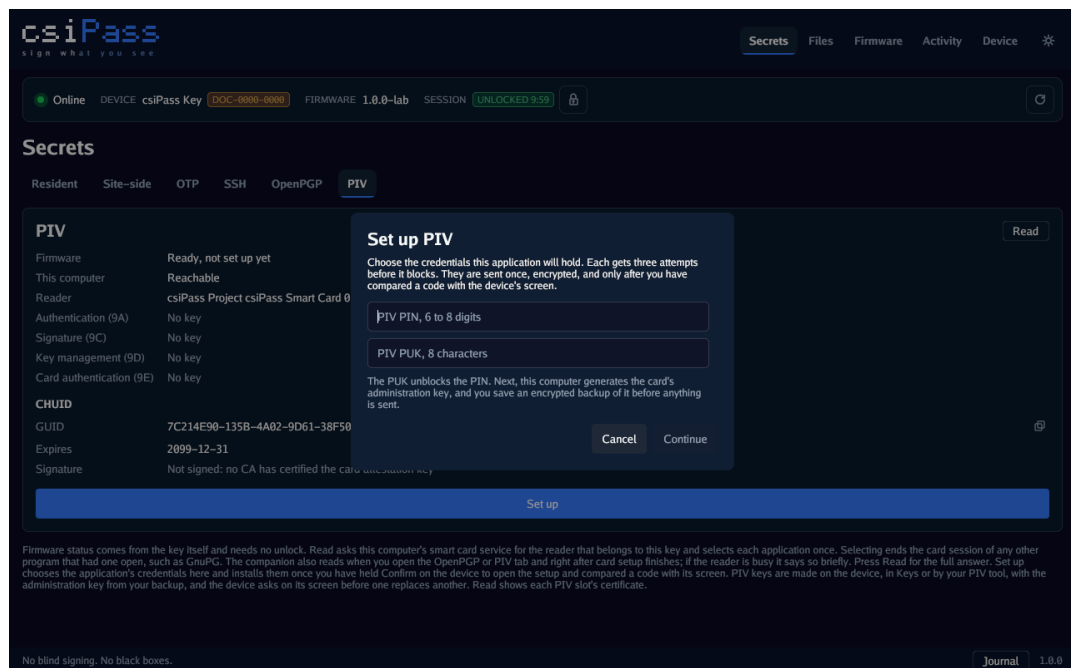
Set up appears on the PIV tab when the application is present, ready and holds nothing, and the companion is unlocked. The card then shows *Ready, not set up yet*, every slot *No key*, and a CHUID that nobody has signed:



PIV before Set up

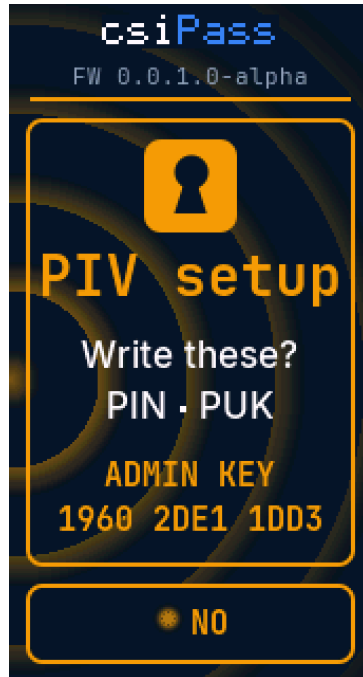
It runs in six steps.

1. **The PIN and the PUK.** Type both. Nothing is sent yet, and no clock runs.



The first step of Set up PIV

2. **The administration key.** The companion generates it and shows its fingerprint – three groups of four hex digits.
3. **Save the backup.** Choose a passphrase of at least 12 characters – not the device PIN, not one you use elsewhere – and where to save the file. Keep it somewhere other than this csiPass, and do not print it. The file is standard [age](#) encryption and opens without the companion: `age -d csipass-piv-admin-key.age` shows the key, its purpose, its algorithm and its fingerprint.
4. **Check the backup.** Type the passphrase again. The companion reads the file back from where you saved it – it asks you to choose the file only if you have already moved it – and compares the whole key with the one it is about to install. Setup goes on only if they match; otherwise it says why and offers **Save again**.
5. **The device.** The glass shows **Hold to set up – to set up PIV. Hold Confirm for five seconds**; that opens a 120-second setup window, which only the device can do. Then both screens show a **six-digit code**: answer **They match** in the companion and **SAME** (Confirm) on the glass, in either order. If the codes differ, answer **They differ / DIFF** and stop – something between this computer and the device is answering for it.
6. **The commit.** The device shows what it is about to install, **naming the administration key by the same fingerprint** the companion showed, and waits for **Confirm**. If the fingerprints differ, answer **No** (BOOT). Only after Confirm is anything written.



What the device is about to write, with the administration key's fingerprint

Keep the backup file. Without it you can still use the keys, change the PIN and unblock it with the PUK, but you cannot make or replace a key or store a certificate short of a Full PIV reset.

If the companion loses the answer at the last step – the cable goes, the wait runs out – it does **not** send again; **Check the device** reads back what landed and changes nothing.

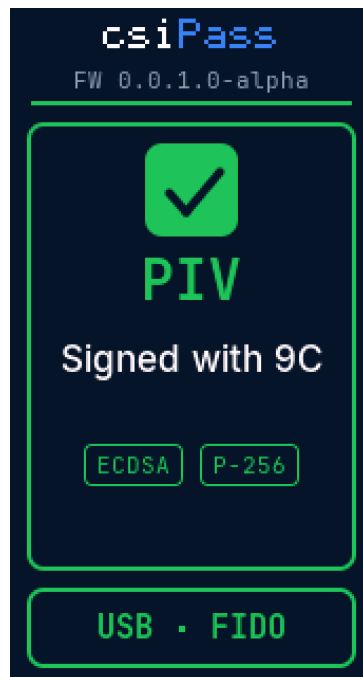
You can also start from the device: on the glass, open **Smart cards** and hold **BOOT** while the footer says **SET UP**. The device sets up the first application that holds nothing yet and waits for the companion, which must be unlocked; *Unlock first* on the glass means it is not.

What the glass shows

PIV asks for a press only where a key would be destroyed. Everything else it shows without asking – blue while the card works, green when it succeeded:



A 9D key agreement in progress



A 9C signature done

On the glass	When	Your answer
PIV , in red – <i>New 9A key / Replaces 9A key, the algorithm beside it</i>	A key is about to replace one the slot already holds	Confirm replaces it; BOOT refuses, and so does the red Cancel (Esc) in the companion's Replace key window while it waits, its button counting the seconds left – with the companion unlocked; locked, press NO on the device. The old key is gone for good once confirmed
<i>Making 9A key and a spinner</i>	A key is being made – into an empty slot with no press asked, or after your Confirm	nothing
<i>Signing with 9C, Logging in with 9A, Agreeing with 9D, Decrypting with 9D and a spinner</i>	A key is in use	nothing; it is information, not a question
PIV – <i>Sign with 9C, Log in with 9A, ...</i>	A key made with Press always or Press once in 15 s is about to be used	Confirm lets it; BOOT refuses. Press once in 15 s asks once, then not again for 15 seconds
PIV , in green – <i>Signed with 9C, New 9A key made, ...</i>	The operation succeeded	nothing; it goes back to Ready by itself
PIN set, PUK set – <i>PIV</i>	After a PIN or PUK change or unblock	nothing; it goes by itself
PIV , in red – <i>Wrong PIN / 2 tries left (or Wrong PUK, Wrong admin key)</i>	A PIN, PUK or administration key was presented and refused, by any program	nothing; it goes by itself, and Activity records it
PIV , in red – <i>PIN blocked / Unblock with PUK; PUK blocked / Reset PIV: keys erased</i>	The last try was spent, or a blocked one was tried again	Unblock the PIN with the PUK; a blocked PUK takes a Full PIV reset
Admin key blocked – <i>Reset PIV: keys erased</i>	The administration key's third wrong attempt	Only a Full PIV reset leaves this state
PIV , in red – <i>Change admin key / Old key stops working</i>	A new administration key was sent, after the current one authenticated	Confirm replaces it – every copy of the old key stops working; BOOT refuses and changes nothing
PIV , in green – <i>Admin key changed / Old key no longer works</i>	The new key is stored	nothing; keep its backup where you kept the old one

On the glass	When	Your answer
PIV , in green – <i>CHUID / Stored, Certificate 9A / Stored or Removed, Attestation cert / Stored, ...</i>	An object was written under the administration key: New CHUID, a certificate imported or deleted, <code>csipass-lab-ca certify</code>	nothing; it goes by itself
PIV , in red – <i>CHUID, Certificate 9A, ... over Refused: malformed, Not this slot's key, Not on this image, Already issued, Retired: kept, Could not be saved</i>	Such a write was refused, and why	nothing; the card kept what it had
PIV , in red – <i>Delete 9A key / Gone for good</i>	A slot's key is about to be deleted, after the administration key authenticated	Confirm deletes the key and its certificate – on 9D too, where nothing is archived; BOOT refuses and keeps both
PIV , in amber – <i>Set PIN tries / PIN 5, PUK 4</i>	New retry limits were sent, after the administration key and the PIN	Confirm sets them, both counters full; BOOT keeps the old ones
Hold to reset – <i>to erase PIV from this card</i>	Full PIV reset	Hold Confirm five seconds

In Activity every refused PIN, PUK or administration key is a row, whatever you chose to record: *Wrong PIV PIN* (with PIN or PUK) or *Wrong PIV admin key* and the tries left; *PIV PIN blocked* or *PIV admin key blocked* for the attempt that used the last try; and the same with *Attempt refused* for any attempt on one already blocked – the card compared nothing, so even the right value is refused, and the journal does not call it wrong.

A replacement waits **30 seconds** for your answer and is refused after that, the same as BOOT. Signing, decrypting and logging in with a key ask for nothing – the PIN is their protection, as the standard defines it – but the glass shows each one while it runs and says when it succeeded, so a key used when you did not expect it is something you can see. A refusal shows nothing. If another question is on the glass at that moment – a passkey, a management confirmation – it keeps the screen, and the key's outcome follows it.

Making a key into an **empty** slot asks for no press on purpose: Windows and OpenSC enrollment do not wait for one, and it destroys nothing. It still needs the administration key, so nobody without your backup can make a key there.

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. The footer names what BOOT does – *NO* on every question – or says *PLEASE WAIT* while the card works, and is empty when there is nothing current to say. A prompt you answer NO to is recorded in Activity as *Declined*, and one nobody answers as *Timed out*.

Press once in 15 s remembers a press for the card session it was made in, the way the PIN is remembered: a program that opens the card again – a second `opensc-too1` run, another application

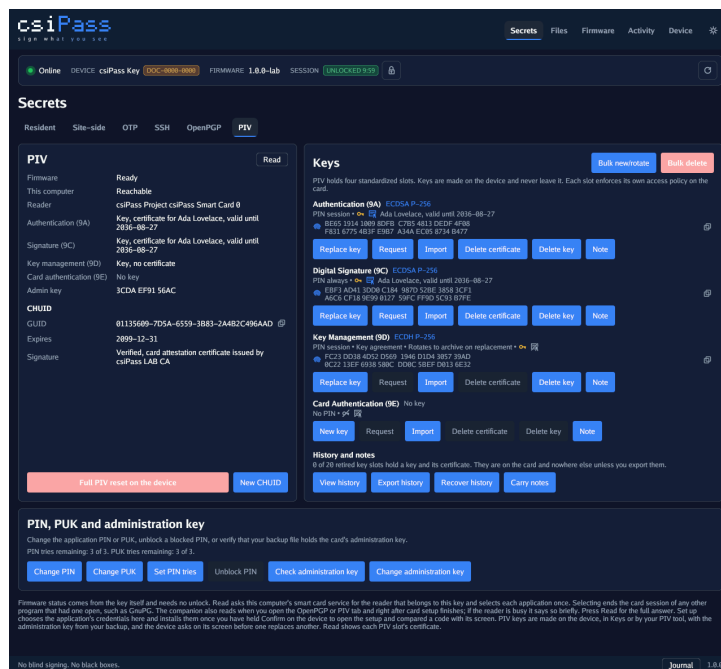
selecting PIV – starts without it and is asked again. Windows also ends a PC/SC transaction left idle for five seconds, after which another program may reset the card and the press with it.

Laboratory images can be set to approve PIV presses by themselves (companion **Device** → **Behaviour** → **PIV auto-confirmation**); product firmware cannot.

Companion: PIV on Secrets

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

The page at a glance



The PIV tab with keys in 9A, 9C and 9D

From the top:

- **The header** is the whole window's: the device and its serial, the **firmware** (a LAB card image ends in `-lab`) and the **session**, **LOCKED** or **UNLOCKED** with the minutes left. Set up, a reset, notes and the administration key's fingerprint need it unlocked.
- **The PIV card** (left) is the application and what the last **Read** found:

Row	What it says
Firmware	The application's state as the device reports it: <i>Ready, Ready, not set up yet, Reset in progress: finish it on the device, Damaged: ...</i>
This computer	What this computer's smart-card service found when it selected PIV: <i>Reachable, Not found on the card, Present, refused (status ...), or Not read</i>
Reader	The reader's name, as Windows or pcsc-lite lists it
Authentication (9A) ... Card authentication (9E)	Per slot: <i>Key, certificate for the subject*, valid until* the date; Key, no certificate; or No key</i>
Admin key	The administration key's fingerprint, while unlocked (below)
CHUID – GUID, Expires, Signature	The card's identifier, with a copy button; its expiry; whether a CA-certified attestation key signed it

Under the rows: **Full PIV reset on the device**, and beside it **New CHUID** on a LAB card image; **Set up** instead while PIV is not set up.

- **Keys** (right) has one block per slot – the slot's name and reference, then its algorithm in blue; a line with its access rule (*PIN session, PIN always, No PIN*; 9D adds *Key agreement* and *Rotates to archive on replacement*), a gold **key** glyph and a blue **certificate** glyph with the certificate's subject and expiry; then the key's **fingerprint** with a copy button; then the slot's buttons. **Bulk new/rotate** and **Bulk delete** are in the heading. **History and notes** closes the card, with how many of the twenty retired slots hold a key.
- **PIN, PUK and administration key** (the full-width band) says the tries left of each and holds their buttons; *Unblock PIN* is live only while the PIN is blocked.
- **The hint** at the bottom says where each line comes from.

A button that cannot act now is greyed out: *Request* and *Delete certificate* on a 9D without a certificate (its key never signs), *Delete key* on an empty slot.

Overview and Read

The PIV card shows whether the firmware serves PIV, what this computer found through its smart-card service, and – after **Read (F5 / Ctrl+R)** – the reader and what each slot holds: its key and the key's algorithm, and the certificate that names it, with its subject and expiry. A slot shows **Key, no certificate** when the device made a key and nothing certifies it yet, and **No key** when it is empty; the device answers that without a PIN (D12, P24). Firmware from before that answer shows only certificates, and a slot without one reads **No certificate**, which says nothing about a key.

Below the slots, while the companion is unlocked, **Admin key** gives the administration key's fingerprint – the three groups of four hex digits the setup glass showed and the key's .age backup carries. The device reports it (never the key), so this is how to tell which of several backups opens this card before handing one to a script or to **Recover history**.

Below that, under its own **CHUID** heading, is the card's CHUID, read with no PIN: **GUID** (8-4-4-4-12 hex, in the order the card stores it, with a copy button), **Expires**, and **Signature** – *Not signed* while no CA has

certified the card attestation key, *Verified, card attestation certificate issued by* the CA's name when the signature is sound, over these bytes and by the certificate inside it, or *Does not verify*. Verified says nothing about whether that CA is one to trust: the companion has no trust anchor for it and only names it. A FASC-N other than all nines – one an issuer wrote – gets a row of its own. The GUID changes with every Full PIV reset and with nothing else on the application image, which takes no CHUID written with PUT DATA. The **lab card** image does, and **New CHUID** – beside **Full PIV reset**, on that image only – gives the card a CHUID with a new random GUID. It asks for the administration key backup; the card then signs the new CHUID with its attestation key if a CA has certified that key (`csipass-lab-ca certify` , which takes the same administration key backup and its passphrase), so **Signature** reads *Verified* with your LAB CA's name, and *Not signed* otherwise. A computer that knew the old GUID reads the card afresh (`deploy\clear-windows-piv-cache.ps1` if Windows holds on to it). Only a Full PIV reset brings back the card's own CHUID. Read selects PIV and OpenPGP once; selecting ends the card session of any other program that had one open. If another program holds the reader, the page says so; close it and press **Read** again.

Keys

Under each slot's name, after its access policy, a **gold key** means the slot holds a key and a **blue certificate** means a certificate names it, followed by whom the certificate is for and until when. A grey glyph struck through is one the device says is not there; a grey glyph that is not struck is one it cannot say about – before **Read**, or with firmware from before the device could report its keys. Hover a glyph for what it stands for: the key's algorithm, the certificate's subject.

Under that line a slot holding a key shows its **fingerprint**, with the fingerprint glyph and a copy button. PIV defines no fingerprint; `csiPass` uses SHA-256 over the key's `SubjectPublicKeyInfo`, so it names the key rather than the certificate and stays the same when a certificate is issued again. It changes when the key is replaced – which is how to tell that a **Replace key** did or did not happen. Copy puts the 64 digits on the clipboard as `openssl` prints them; to compute it from a certificate:

```
openssl x509 -in piv-9a.pem -pubkey -noout | openssl pkey -pubin -outform DER | openssl dgst -sha
```

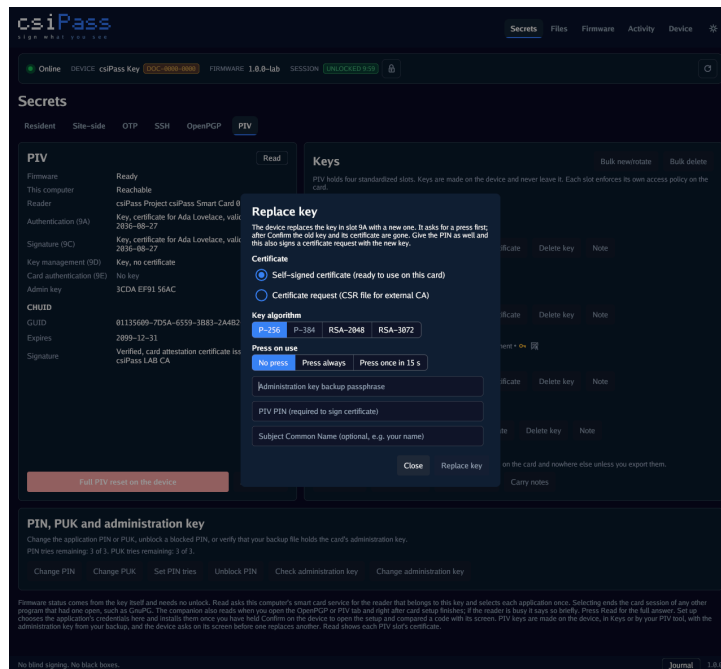
For each slot:

Control	What it does	Needs
New key (a slot holding a key: Replace key ; 9D with a certificate: Rotate key)	The device makes a key in the slot and, as you choose, a self-signed certificate (ready to use), a certificate request (a CSR for your authority), or for 9D its public key for your authority or a certificate from a local one-time issuer (below). Replace key and Rotate key start on the algorithm and the press of the key the slot holds, and a replacement never asks for less of a press than that key: the weaker choices are greyed out, and the device refuses them from any tool. Press on use : No press (the default, what Windows logon and TLS clients expect), Press always before every use, or Press once in 15 s – one press covers every use for the next 15 seconds	Administration key backup; the PIN for a self-signed certificate on 9A or 9C, or to sign a request at once; Confirm on the glass if the slot already holds a key
Request	Signs a PKCS#10 request with the key the slot holds, certified or not (not 9D, whose key never signs)	The PIN, presented just before the signature
Import	Stores the certificate your authority issued. The device checks that it names this slot's key, and refuses it otherwise	Administration key backup
Delete certificate	Removes the slot's certificate; the key stays	Administration key backup
Delete key	Deletes the slot's key and the certificate that names it, for good – on 9D too, where it is not archived. Shown when the card says the slot holds a key	Administration key backup; Confirm on the glass
Note	Your note about the slot's key, kept on the device; a new key starts without one	Companion unlocked

New key, **Replace key** and **Rotate key** open one window. It says what the device is about to do, then asks, top to bottom:

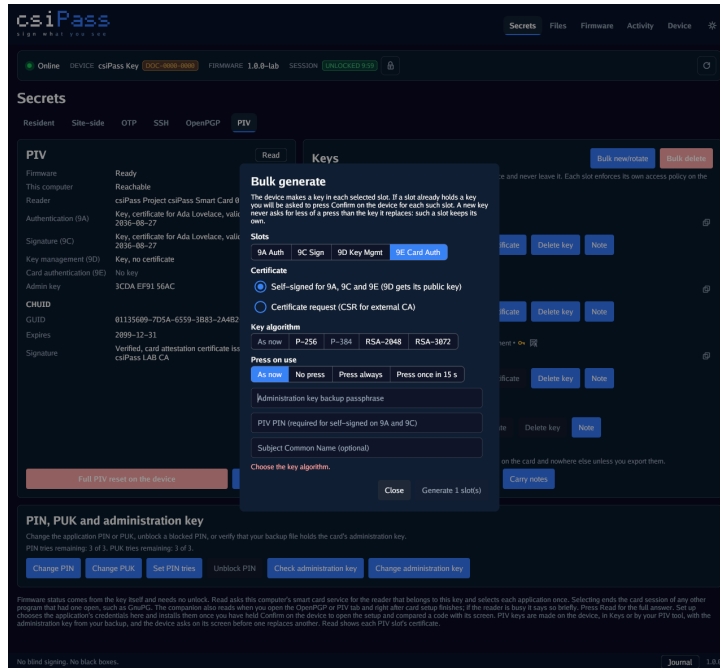
- **Certificate** – *Self-signed certificate (ready to use on this card)* or *Certificate request (CSR file for external CA)*; on 9D also its *public key* for your authority or the *local one-time issuer* (below).
- **Key algorithm** – the ones the card lists; a replacement starts on the current key's.
- **Press on use** – *No press*, *Press always*, *Press once in 15 s*; a replacement starts on the current key's, and a weaker choice is greyed out.
- **Administration key backup passphrase**, the **PIV PIN** where the new key has to sign something at once (a self-signed certificate on 9A or 9C, or a request), and an optional **Subject Common Name** (your name).

The button at the bottom right names the action. Over an existing key the glass asks for Confirm after it; into an empty slot it does not.



Replace key for slot 9A

Bulk new/rotate makes keys in several slots with the backup opened once – self-signed certificates for 9A, 9C and 9E (the PIN is asked for 9A and 9C), and for 9D its public key or a local issuer's certificate – with one **Key algorithm** and one **Press on use** for every key it makes. The algorithm's **As now** makes each new key in the algorithm of the key it replaces, so a P-384 9A stays P-384 beside a P-256 9C; it is offered only when every ticked slot holds a key, and then taken by itself while no other algorithm is chosen; and with an empty slot among them it is greyed out, its tip says why, and the algorithm has to be named. The press's **As now**, the default, keeps each replaced key's own, and a slot whose key asks for more than the choice keeps its own – and **Save all to folder** writes every PEM it produced. **Bulk delete** removes the certificates of the slots you tick; their keys stay. Either asks for a press on the glass for each slot that already holds a key. While **Bulk new/rotate** waits for one, its button counts the press down and its red **Cancel** (Esc) declines it; a press declined – there or with NO on the glass – stops the run, and the slots after it are reported as not started.



Bulk new/rotate, opened on a card whose 9E is empty

The window opens with the empty slots chosen (**9E** here); the **Slots** row toggles each one. What is still missing is said in red under the fields, and the button counts the slots it will make (*Generate 1 slot(s)*).

The local one-time issuer (9D only). For personal or offline use, when there is no certificate authority: the companion makes an issuer, signs 9D's certificate with it, and forgets the issuer's key. You must save the chain – 9D's certificate and the issuer's – before the certificate is installed. Another computer trusts 9D only once you install that issuer certificate there. There is no revocation and no renewal: replacing 9D means a new issuer.

History and notes

- **View history** lists 9D's certificate and every retired key's – subject, serial, end of validity, key – and offers the same file to save. It asks for nothing, so it opens on the list rather than on a form.
- **Export history** saves 9D's certificate and every retired key's to a file, straight away for the same reason. Certificates are public and the keys never leave the card, so the file decrypts nothing; it is what puts a certificate back if the card loses one.
- **Recover history** reads an exported history and writes back each certificate the card has lost – only onto the key it was issued for, and never over a certificate a slot still has. It needs the administration key backup.
- **Carry notes** saves this application's notes to a file before a reset, and after it writes back the ones you tick; each binds to whatever key its slot holds then.

The page shows how many of the twenty retired slots are in use.

PIN, PUK and administration key

Control	What it does	Needs
Change PIN	A new PIN of 6 to 8 digits	The current PIN
Unblock PIN	A new PIN once the old one is blocked	The PUK
Change PUK	A new PUK of exactly 8 characters	The current PUK
Set PIN tries	How many wrong tries block the PIN and the PUK, 1 to 15 each. The values stay; both counters start full. The line above the buttons shows the tries left and of how many	The backup and the PIN; Confirm on the glass. Refused while the PUK is blocked
Check administration key	Asks the card whether the key in your backup is its administration key	The backup; a wrong key costs one of its three tries
Change administration key	Makes a new key, saves its backup where you choose and opens it again, then replaces the card's key with it. From then on the old backup opens nothing	The current backup; a passphrase for the new one, typed twice; Confirm on the glass

As the standard says, a PIN change leaves the PIN verified under its new value, and an unblock leaves it verified if it was before; 9A and 9D need no new PIN either way. The next 9C signature still asks for the PIN.

Full PIV reset

Full PIV reset on the device erases the application: its PIN, PUK and administration key, every key, every certificate, the history and the notes. The glass shows **Hold to reset**; hold Confirm five seconds. Afterwards PIV is empty and ready for **Set up** again, with a new administration key and a new backup – the old backup opens nothing. Export the history and carry the notes first if you want them.

This is the only way out of a blocked PUK or a blocked administration key.

If a reset is interrupted by a power cut, PIV is left half erased and unavailable, and the companion says *Reset in progress: finish it on the device*. The device does not finish it by itself: on the glass, open **Smart cards**, hold **BOOT** while the footer says **FINISH**, and Confirm the red **Finish reset** offer.

Setting up the command-line tools

The board runs used these; what each is for is in the sections after this one.

Tool	For	Where to get it
OpenSC (0.26.1 on the bench) – <code>opensc-tool</code> , <code>pkcs15-tool</code> , <code>pkcs11-tool</code> , <code>piv-tool</code> , and the PKCS#11 module <code>opensc-pkcs11</code>	Reading the card, keys and certificates, administration without the companion, PKCS#11 for browsers, mail and SSH	The OpenSC installer for Windows; <code>opensc</code> on Linux
OpenSSL 3 (3.5 on the bench)	Certificates and requests, checking chains	Git for Windows ships one (<code>C:\Program Files\Git\usr\bin\openssl.exe</code>); any OpenSSL 3 does
libp11 's <code>pkcs11prov</code> (0.4.21 on the bench)	Letting OpenSSL sign with a key on the card	<code>libp11</code>
age	Opening the administration key backup outside the companion	age-encryption.org
<code>certutil</code>	What Windows makes of the card	Windows
<code>csipass-lab-ca</code>	Certificates for every slot from a LAB CA, attestation	This repository (<code>go build ./cmd/csipass-lab-ca</code>); LAB certificates

Windows. The OpenSC installer does not put its tools on the `PATH`. In each PowerShell window, or in your profile:

```
$env:Path += ';C:\Program Files\OpenSC Project\OpenSC\tools'
$env:Path += ';C:\Program Files\Git\usr\bin'      # for openssl, if it is not already there
```

Then check that the card answers, with nothing asked:

```
opensc-tool --list-readers      # csiPass Project csiPass Smart Card 0
pkcs15-tool --list-pins        # PIN (auth ID 01) and PUK (02), each with its tries
pkcs15-tool --list-certificates
```

The inbox Windows driver needs nothing installed for `certutil`, logon and browsers.

Linux. `pcscd`, `libccid`, `opensc` and `openssl`, with the csiPass reader taught to the CCID driver (*Linux*, under *PIV with standard tools*). The same three checks apply.

Keep secrets off the command line. `pkcs15-tool` and `pkcs11-tool` ask for a PIN or PUK they were not given (`--pin`, `--puk`, `--new-pin` left out), so it stays out of your shell history. The administration key reaches OpenSC through a file named by `PIV_EXT_AUTH_KEY` (*Administering PIV with OpenSC*), which you delete straight after.

PIV with standard tools

The companion manages the card; everyday use belongs to your operating system and your applications, which speak PIV themselves. OpenSC paths below have been exercised on a board (test protocols 23d and 23h).

Windows. Windows includes a PIV smart-card driver (`msclmd.dll` , shown as *Identity Device (NIST SP 800-73 [PIV])*). Once slot **9A** holds an RSA key and a certificate, `certutil -scinfo` lists it under both the Base CSP and the Smart Card KSP. Self-signed companion certificates report `CERT_E_UNTRUSTEDROOT` – expected until you trust the issuer. `certutil -scinfo` builds chains with the machine's engine, so for it the issuer belongs in the machine's Root store (`certutil -addstore Root issuer.pem` , elevated); in your user's alone it reports *Missing Issuer* and `CERT_E_CHAINING` , while browsers and mail under your account are satisfied. A LAB certificate names no CRL, so `CRYPT_E_NO_REVOCATION_CHECK` stays: expected. The inbox driver does not enumerate ECDSA slots (**9C** / **9E**); OpenSC still does.

```
certutil -scinfo
```

Before any slot has a certificate, the same command names the card and then stops with `NTE_BAD_KEYSET` – *keyset does not exist*. Windows builds a key container only for a slot that holds a certificate, so an application that is set up but has no certificate yet has none to show; it is not a fault.

After a lab PIV reset or certificate rewrite. Windows caches container maps under the CHUID GUID. Writing certificates with the companion or OpenSC does not refresh that cache. `certutil` then keeps saying `NTE_BAD_KEYSET` while `pkcs11-tool --list-objects` is fine. Even after `SCardSvr` has caught up, browsers can still offer the **previous** certificate until the Certificate Propagation Service (`CertPropSvc`) restarts. Close the companion (and any OpenSC / GnuPG session on the reader), then flush:

```
pwsh -NoProfile -File deploy\clear-windows-piv-cache.ps1 -SubjectMatch 'Bespalov|csiPass' -ScInfo
```

The script briefly disables SmartCardReader PnP devices (to drop open PC/SC handles), stops `SCardSvr` (it persists its RAM map into the registry on stop), removes only `Calais\Cache\Cache` and the PIV ATR cache entries, then starts `SCardSvr` and `CertPropSvc` and re-enables the readers. It never deletes `Calais\SmartCards` . Prefer closing the companion, browsers on the card, OpenSC tools, and GnuPG first. A full PIV **application** reset on a LAB image changes the derived CHUID GUID and often avoids the stale Calais map; rewriting certificates in the same life does not.

OpenSC (Windows, macOS, Linux) provides a PKCS#11 module that browsers, mail clients and SSH use:

```
pkcs11-tool --list-slots
pkcs11-tool --list-objects --type cert
ssh-keygen -D /usr/lib/opensc-pkcs11.so          # the SSH public key of 9A
ssh -I /usr/lib/opensc-pkcs11.so user@server
```

The module path differs by system (`opensc-pkcs11.dll` on Windows). SSH uses 9A, which needs the PIN once per session.

Administration tools (`yubico-piv-tool` , OpenSC's `piv-tool`) can make keys and store certificates, given the administration key from your backup (`age -d` shows it) with algorithm **AES-256** (`0C`). The companion does the same with a check that the certificate belongs to the key and without the key ever sitting in a command line; prefer it where you can. Where you cannot – an organisation administering many cards with scripts – the next section is the whole of it with OpenSC.

Windows logon. A domain logs on with 9A: an RSA-2048 key (the inbox driver shows no other), a certificate from a CA the domain trusts for logon (in NTAUTH) with the user's UPN and, for current domain controllers, the account's SID (`csipass-lab-ca issue -upn ... -sid ...` on a lab; the [LAB certificates guide](#) has the whole path). **Choose the press for 9A with logon in mind:** Windows uses the key several times per logon – **Press always** asked three times at logon and four at unlock on the board – so **Press once in 15 s** (one press) or **No press** is the practical choice; a replacement cannot lower the press, so decide before the key is made. Remote Desktop with smart-card redirection logs on the same way. The card cache it reads is the **client's**: after a PIV change, refresh the client (New CHUID, or `clear-windows-piv-cache.ps1`) or the remote logon reports *No valid certificates were found on this smart card*. Removing the card locks the session where *Smart card removal behavior* says so.

Linux. `pcsc-lite` with the CCID driver (`pcscd` , `libccid`) and OpenSC. Until `csiPass` has a USB identity of its own, the CCID driver does not know the development one: 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` . Where there is no `polkit` – a container – start `pcscd --disable-polkit` . Under WSL 2 the device reaches Linux through `usbipd-win` (`usbipd bind` , `usbipd attach --wsl`), leaving Windows while it is there, and the companion's management needs `modprobe usbhid hid_generic` in the WSL VM first.

When the card changes under a program. Setting PIV or OpenPGP up, resetting either, or recovering the card's data powers the card off before the reader opens again. A program that held the card gets an error on its next exchange – Windows reports `SCARD_E_COMM_DATA_LOST` , `pcsc-lite` `SCARD_W_REMOVED_CARD` – and nothing from the new card until it reconnects; `SCardReconnect` with a reset is enough. On Windows a program that connects afresh gets the same error until one of them does that: the smart card service was never told the card went off. The companion reconnects by itself (from 2026-09-29; before, PIV stayed *Not read* after a set-up until something else reset the card). The Windows PIV driver recovers by itself without unplugging (`certutil -scinfo` reads the card again; see above for its certificate cache). GnuPG does not: see the OpenPGP manual.

Administering PIV with OpenSC, without the companion

Everything the companion does to PIV after setup, done with OpenSC 0.26 (`piv-tool` , `pkcs15-tool` , `pkcs11-tool` , `opensc-tool`) and OpenSSL 3. Every command here was run on a board (test protocol 23h, 2026-09-26) with the companion closed. What stays on the device by design, with no PIV command: **setting PIV up** (PIN, PUK and administration key), a **Full PIV reset**, slot **notes** and **Activity**.

The administration key. `age -d csipass-piv-admin-key.age` prints it on the `Key:` line, 64 hex digits; the fingerprint beside it is the one the companion shows as **Admin key**. OpenSC reads it from the file named by `PIV_EXT_AUTH_KEY` , as printed or colon-separated. Put it in a file only for the commands that need it, and delete the file straight after:

```
$keyFile = Join-Path $env:TEMP 'piv-admin.txt'
Set-Content -LiteralPath $keyFile -NoNewline -Encoding ascii -Value '<64 hex digits>'
$env:PIV_EXT_AUTH_KEY = $keyFile
try {
    piv-tool -A M:9B:0C -n          # checks the key; add the real work here
} finally {
    Remove-Item Env:\PIV_EXT_AUTH_KEY
    Remove-Item -LiteralPath $keyFile -Force
}
```

A wrong key costs one of its three tries, and the third blocks it, which only a Full PIV reset leaves. A script should check the key once and stop if it is refused, rather than present it again with every command.

Reading the card needs nothing: `pkcs15-tool --list-pins` (the PIN is auth ID 01, the PUK 02, each with its tries), `--list-certificates`, `--list-keys`. Whether a slot holds a key, its algorithm and its press:

```
opensc-tool -s 00:A4:04:00:05:A0:00:00:03:08 -s 00:F7:00:9A
```

– `9000` with the key's metadata, `6A82` for an empty slot (and `9C`, `9D`, `9E`, or `80` and `81` for the PIN's and PUK's tries and limits).

Keys. `piv-tool -A M:9B:0C -G <slot>:<alg> -o <slot>.pub` with the algorithm as OpenSC's number: `11` P-256, `14` P-384, `07` RSA-2048, `05` RSA-3072. Into an empty slot nothing is asked; over a key the glass asks, and NO leaves the old key. `piv-tool` exits with `1` after a key it did make, so judge by the public key file, not the exit code. A key made this way asks for no press when it is used; for **Press always** or **Press once in 15 s** send the GENERATE yourself with the touch policy (`AB 01 02` always, `AB 01 03` cached):

```
piv-tool -A M:9B:0C -s 00:47:00:9A:08:AC:06:80:01:11:AB:01:02
```

A replacement never asks for less of a press than the key it replaces; a weaker policy is refused before the glass asks anything.

A certificate for a key on the card. OpenSC shows a PIV key only beside a certificate, so point it at the public key you saved (`PIV_9A_KEY`, and `PIV_9C_KEY`, ... likewise) and sign through OpenSSL's PKCS#11 provider – `pkcs11prov` from `libp11` 0.4.21 was the one used:

```
$env:PIV_9A_KEY = '9a.pub'
openssl req -new -x509 -days 365 -subj "/CN=csiPass 9A" `
    -provider-path '<libp11 directory>' -provider pkcs11prov -provider default `
    -key 'pkcs11:id=%01;type=private;pin-value=<PIN>' -out 9a.pem
piv-tool -A M:9B:0C -C 9A -i 9a.pem
```

–`new` without `-x509` makes a request for your authority instead; `9C` asks for the PIN for each signature, on the console. OpenSC 0.26.1 takes **PEM** for `-C`, not DER. The card refuses a certificate for any other

key. A P-384 key signs a SHA-256 digest – the shorter digest is signed as the same number, as OpenSC sends it – from firmware of 2026-09-26 on.

Delete a slot's certificate (the key stays), with the slot's tag – 05 9A, 0A 9C, 0B 9D, 01 9E:

```
piv-tool -A M:9B:0C -s 00:DB:3F:FF:07:5C:03:5F:C1:05:53:00
```

Delete a key – and its certificate – asks on the glass: `piv-tool -A M:9B:0C -s 00:F6:FF:9E`.

PIN and PUK. `pkcs15-tool --change-pin` (it asks for the old and the new PIN); the PUK the same with `--auth-id 02` (exactly eight characters); `pkcs15-tool --unblock-pin` (the PUK, then the new PIN). `--pin`, `--puk` and `--new-pin` give the values on the command line instead, for scripts that can keep them out of a history. `pkcs15-tool --verify-pin --auth-id 01` checks a PIN, and a wrong one costs a try as anywhere else. **Tries** need the PIN verified in the same run, and ask on the glass:

```
piv-tool -A M:9B:0C -s 00:20:00:80:08:<PIN as hex, FF-padded to 8 bytes> -s 00:FA:05:04
```

sets five for the PIN and four for the PUK, both counters full.

A new administration key – save its backup first; the glass asks, and from Confirm on the old key opens nothing:

```
piv-tool -A M:9B:0C -s 00:FF:FF:FF:23:0C:9B:20:<the new key, colon-separated>
```

9D and its history. A certified 9D replaced with `-G 9D:<alg>` goes into the retired archive, *Retired KEY MAN 1* onwards (IDs 05 and up), and still decrypts what was sent to it (`pkcs11-tool --login --decrypt --id 05` for RSA, `--derive` for ECDH). Twenty retired keys fill the archive; a twenty-first rotation is refused before anything is asked, and nothing is evicted. The Key History Object says how many are held:

```
opensc-tool -s 00:A4:04:00:05:A0:00:00:03:08 -s 00:CB:3F:FF:05:5C:03:5F:C1:0C:00
```

(C1 01 14 is twenty). **Export** is `pkcs15-tool --read-certificate 03` for 9D and 05, 06, ... for the retired ones. **Recover** writes a retired certificate back as its whole container, wrapped in 53, under the SP 800-73 container ID – 1001 for the first retired key – not the 5FC10D tag: `piv-tool -A M:9B:0C -0 1001 -i <container>`. The card takes it only for the key it names; a retired certificate cannot be deleted.

Presses in scripts. A press remembered under **Press once in 15 s** lasts for one card session: keep one program's connection open across the uses, and do not let it sit idle for five seconds (see *What the glass shows*).

Attestation on a LAB image: `csipass-lab-ca verify -dir <LAB CA>` ([LAB certificates](#)) attests every key back to your LAB CA. `yubico-piv-tool -a attest -s 9a` sends the same command in YubiKey's form but has not been run against `csiPass`.

Every LAB certificate at once. `csipass-lab-ca issue` issues and installs the certificates for 9A, 9C, 9D, 9E and OpenPGP's login key in one pass, each for a key the card attests it made, from your LAB CA

or another CA you name (test protocol 23j); its guide is [LAB certificates](#).

Everyday use

1. **Plug the key in.** The host enumerates FIDO and the smart-card reader.
 2. **One program on the reader at a time.** A program that keeps PIV selected loses its session when another selects it; that is normal.
 3. **Logon, TLS, SSH** use 9A: the PIN once, then signatures without it until the session ends.
 4. **Signing documents and mail** uses 9C: the PIN before every signature.
 5. **Decrypting mail** uses 9D, or a retired key for mail encrypted to an old 9D certificate.
 6. **Door readers and other PIN-less uses** use 9E, which answers anyone who holds the device.
 7. **Renewing a certificate – Request** with the existing key, then **Import** what comes back. **New key** instead when the key itself should change.
 8. **Rotating 9D – Rotate key** keeps the old key in the history so old mail still opens; publish the new certificate to your correspondents.
 9. **Backups** – keep the administration key backup and an **Export history** file somewhere safe. Private keys cannot be backed up off the card; a lost or reset device means those keys are gone.
 10. **Blocked PIN – Unblock PIN** with the PUK. A blocked PUK or a blocked administration key means a Full PIV reset.
-

Quick map: goal ↔ companion ↔ glass ↔ standard tools

Goal	Companion	Glass	Standard tools
Set PIV up	Set up (Unlocked)	Hold 5 s → SAME → Confirm, fingerprint compared	Not supported: setup is the companion's
See slots and certificates	Read	—	<code>pkcs11-tool --list-objects /</code> <code>certutil -scinfo</code>
Make a key	New key / Replace key / Bulk new/rotate	Confirm over an existing key; <i>Making ... key, New ... key made</i>	<code>piv-tool -A M:9B:0C -G 9A:11 -o 9a.pub</code> with the backup's key
Use a key	—	<i>Signing with ... / Agreeing with ..., then Signed with ...</i>	Middleware, after the PIN
Get a certificate	Request , then Import	—	CSR tool + authority
Rotate 9D	Rotate key	Confirm	—
Keep the history	Export history / Recover history	—	—
Change or unblock the PIN	Change PIN / Unblock PIN	<i>PIN set</i>	<code>pkcs15-tool --change-pin / --unblock-pin</code>
Change the PUK	Change PUK	<i>PUK set</i>	<code>pkcs15-tool --change-pin --auth-id 02</code>
Set the PIN and PUK tries	Set PIN tries	PIV , in amber: <i>Set PIN tries / PIN 5, PUK 4; then PIN tries set</i>	The raw APDU with <code>piv-tool</code> (test protocol 23h, H16); <code>yubico-piv-tool -a pin-retries</code> in form
Delete a key	Delete key	PIV , in red: <i>Delete 9A key / Gone for good; then 9A key deleted</i>	The raw APDU with <code>piv-tool</code> (23h, H15); <code>yubico-piv-tool -a delete-key</code> in form
Check the backup	Check administration key	—	<code>piv-tool -A M:9B:0C -n</code> with the key from the backup
Replace the administration key	Change administration key	PIV , in red: <i>Change admin key / Old key stops working; then Admin key changed</i>	The raw APDU with <code>piv-tool</code> (23h, H14); <code>yubico-piv-tool -a set-mgm-key</code> in form
Certificates for every slot	New key / Request, Import	<i>Certificate 9A / Stored per slot</i>	<code>csipass-lab-ca issue</code> on a LAB image (23j)
Attest a key	—	—	<code>csipass-lab-ca verify</code> on a LAB image certified by your LAB CA;

Goal	Companion	Glass	Standard tools
			yubico-piv-tool -a attest in form; a production image attests nothing yet

In form means the command is YubiKey's, which csiPass implements, but `yubico-piv-tool` itself has not been run against csiPass; the `piv-tool` forms have. | Wipe PIV | **Full PIV reset** | Hold Confirm 5 s | Not supported |

Related documents

Document	Role
User Manual	FIDO everyday use and glass grammar
Companion Manual	Whole companion, including Smart cards
OpenPGP 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/piv-key-objects.md</code>	Slots, access rules, data objects, commands (implementers)
<code>docs/openpgp-piv/commissioning-channel.md</code>	Why Set up is sealed and compared on glass
<code>docs/conformance.md</code>	Which SP 800-73-5 clauses are implemented, and how each was checked