

The Two-Door Mode Latch for a Multi-Role Hardware Signer

Defensive publication, technical note

Version 1.2, 21 October 2026 Author: Josh Coverley, Favilla Pty Ltd, South Fremantle, Western Australia
Product: Favilla KEY (ESP32-S3 application processor, NXP SE051 secure element, STM32G0B1 isolated signer)

This document is published to place the mechanism described below in the public domain as prior art. It is written from the firmware source that ships on the Favilla KEY at the date above (ESP32 tree FKIO, secure-element module `se_signer.cpp` with `mode_pcr.inc`; STM32G0 tree `favilla-g0-witness-1`, frozen 20 October 2026 at tree hash `c0b94d1b7a9e350d`, self-test `v7`, `src/main.c`) and from the KeyV4 production netlist. It describes what that code does. Where earlier design notes and the code disagree, the code wins, and every such disagreement is listed in section 15.

1. The problem

A hardware signer whose one application processor runs both an always-connected identity role and an offline wallet role has two roles with opposite exposure. (The Favilla KEY has three chips: the ESP32-S3 application processor, the STM32G0 isolated signer and the SE051 secure element. The latch separates the two roles of the application processor; the other two chips are what enforce it.) The connected role (here: a Nostr signer reachable over WiFi by paired phones and relays) is where a remote code execution (RCE) on the application processor is most likely. The offline role (here: a Bitcoin wallet signing PSBTs over QR codes) is where the consequences of an RCE are permanent.

The property wanted is:

1. Code that is running on the application processor in the connected role must not be able to reach the Bitcoin seed, not even by rebooting the processor into the offline role.
2. The only way to open the Bitcoin role is a physical power cycle of the whole device, which evicts any memory-resident RCE before the seed is ever decrypted.
3. Both properties must be enforced by a component the application processor cannot impersonate, not by firmware logic on the application processor itself.

The mechanism below achieves this with one persistent Platform Configuration Register (PCR) object inside the secure element, a secret held only by a second microcontroller, and access policies on the key objects that reference the PCR's value.

2. Hardware facts the design rests on

- The secure element is an NXP SE051 (IoT applet) on an I2C bus shared with the application processor (ESP32-S3) and the isolated signer (STM32G0B1). The ESP32 is the only I2C master. The G0 is an I2C target and cannot reach the SE directly; every SE command the G0 issues is carried by the ESP32 as an opaque, SCP03-encrypted frame (a blind relay).
- The SE051's supply pin is fed from the 3V3 rail through its own regulator (3V3 to VIN, VOUT to VCC). Its ENA pin is driven by G0 port PB2 with a 1 MΩ pull-up. A low ENA puts the SE into deep power-down. Persistent SE objects survive this; only open sessions are lost. An ESP32 software reset does not cycle the SE at all.
- The SE051 cannot tell a soft reset of the host from a power cycle, and every transient object type is refused on this applet (observed on silicon: transient PCR, binary and counter creation all return 6985 or 6A80). The SE therefore cannot witness a power cycle on its own. Everything that must survive a reboot is a persistent object.
- The G0's NRST pin is routed only to the Tag-Connect TC2030 programming pads. There is no connection from the ESP32 to NRST. The ESP32 can send the G0 an I2C RESET command, which the G0 handles in software without re-running `main()`.
- The G0 captures `RCC->CSR` at the top of `main()`, so it knows whether it booted from a power-on or brown-out reset (`PWRRSTF`) or from NRST, software or watchdog. The G0's power-on reset threshold (about 1.6 V) is below the ESP32's brown-out detector (about 2.4 V), so any supply sag that resets the G0 has already reset the ESP32.

3. Components and object identifiers

All objects that a policy names, and all objects that hold wallet secrets, live at a per-wallet address base + (generation << 8), with the generation chosen fresh at each onboarding (1 to 63) and stored in the G0's key page. The G0 reports the live generation to the ESP32 at boot. Generation 0 is the pre-generation layout. No PIN object or authentication object is ever deleted; a wiped wallet leaves them as dead objects and the next onboarding takes a fresh generation. Reason: a PIN or auth object that could be deleted could be re-created at the same identifier with a known value, which would pass every policy that names that identifier.

Object	Base	Type	Role
--------	------	------	------

	identifier		
MODE_PCR	0x20000033	Persistent PCR, 32-byte value	The mode latch
PIN_DAILY ("Nostr PIN")	0x70000003	UserID auth object	Opens the connected role
PIN_SPEND ("BTC PIN")	0x70000005	UserID auth object	Opens the Bitcoin role
AUTH_G0	0x20000030	AES-128 AESKey auth object, key K_SE	The G0's own SCP03 session identity
AUTH_RECOVER	0x20000035	AES-128 AESKey auth object, key K_RECOVER = SHA256(K_SE "favilla-recover-key")[0:16]	PIN-less recovery wipe only
AES_WRAP_SEED	0x20000020	AES-256 symmetric key	Outer layer on the Bitcoin seed
AES_WRAP_DAILY	0x20000022	AES-256 symmetric key	Outer layer on the Nostr scalar
NOSTR_SCALAR	0x20000002	48-byte binary	IV AES(K_DAILY, AES(K_G0, scalar))
PCR_UNLOCK (Door-1)	0x20000031	Persistent PCR	Daily-PIN unlock gate for the Nostr scalar (separate mechanism, see section 11)
DOOR1_SECRET	0x20000036	48-byte binary	Door-1 unlock secret (separate mechanism)

The G0 holds in its last flash page (0x0801F800, programmed by SETKEYS during onboarding and never readable over I2C): K_SE (16 bytes), K_G0 (32 bytes), S (32 bytes), and the generation byte. S is the BTC-door secret. The ESP32 generates S from the SE's TRNG at onboarding, hands it to the G0 in the 81-byte SETKEYS record (K_SE || K_G0 || S || generation, acknowledged by SHA256 of all 81 bytes after the G0 has written and read back its flash page), and zeroises its own copy. The ESP32 persists only the derived latch value (section 4) in nvs_sec namespace mode, key btcval, for verification.

4. The latch values, byte-exact

The SE051 PCR semantics, reverse-engineered from readback and confirmed on silicon:

- A PCR created with initial value I reads SHA256(I) at rest.
- Extending with 32-byte X sets PCR := SHA256(PCR || X).
- Reset (WritePCR with TAG_1 only) returns the PCR to SHA256(I). Reset is governed by the policy's DELETE right. A reset on a PCR already at rest is refused with 6985.
- The extend data goes in TAG_3 (0x43). Sending TAG_2 (the create-time initial value) to an existing PCR is refused with 6A80.
- A freshly created PCR reads the at-rest value but, on this applet, extends from a different internal base than a PCR that has been reset. The firmware therefore never extends a freshly created latch; it always has the G0 reset it first.

With I = 32 zero bytes:

NEUTRAL	=	SHA256(0x00 * 32)	
NOSTR	=	SHA256(NEUTRAL "FAVILLA-MODE-NOSTR" zero-padded to 32 bytes)	
BTC	=	SHA256(NEUTRAL S)	S = 32-byte secret held only by the G0
SPENT	=	SHA256(BTC "FAVILLA-MODE-SPENT" zero-padded to 32 bytes)	
ARM_x	=	SHA256(x "FAVILLA-MODE-ARM" zero-padded to 32 bytes)	for x in {NEUTRAL, SPENT, BTC}

String constants as bytes:

"FAVILLA-MODE-NOSTR"	+ 14 x 0x00	=	46 41 56 49 4C 4C 41 2D 4D 4F 44 45 2D 4E 4F 53 54 52 00..00
"FAVILLA-MODE-SPENT"	+ 14 x 0x00	=	46 41 56 49 4C 4C 41 2D 4D 4F 44 45 2D 53 50 45 4E 54 00..00
"FAVILLA-MODE-ARM"	+ 16 x 0x00	=	46 41 56 49 4C 4C 41 2D 4D 4F 44 45 2D 41 52 4D 00..00

NEUTRAL and NOSTR are public and computable by anyone. BTC, SPENT and ARM_SPENT, ARM_BTC depend on S. The ESP32 knows the BTC value (it sits in the seed key's policy and in NVS) but cannot produce it: producing BTC requires extending from NEUTRAL with S, and only the G0 holds S.

Any value other than the five above matches no policy and is called "wedged" or "non-canonical" in the logs. Reaching one is a denial of service until the next power cycle, never an unlock.

The legacy public value SHA256(NEUTRAL || "FAVILLA-MODE-BTC" padded) still exists in the ESP32 source as a fallback for units onboarded before S was introduced. It is logged as legacy and is not part of the mechanism described here.

5. The MODE_PCR policy

MODE_PCR is created persistent (CLA 0x80, INS 0x01 WRITE, P1 0x09 PCR, P2 0x00) with TAG_POLICY, TAG_1 = identifier, TAG_2 = 32 zero bytes. The policy, as encoded (each plain entry is 08 || authObjId(4) || rights(4), rights bits per AN12543: READ 0x00200000, WRITE 0x00100000, DELETE 0x00040000, ENC 0x02000000, DEC 0x01000000, REQUIRE_PCR_VALUE 0x00010000):

Authenticated as	Rights on MODE_PCR	Meaning
------------------	--------------------	---------

PIN_DAILY	READ, WRITE	May read the latch, may extend it
PIN_SPEND	READ, WRITE	May read the latch, may extend it
AUTH_G0	READ, WRITE, DELETE	May read, extend, and reset (DELETE governs WritePCR reset)
AUTH_RECOVER	DELETE	May delete the object during a recovery wipe
(anonymous)	none	An unauthenticated command can neither read, extend nor reset

Bench-proven: an anonymous WritePCR reset or extend on this object returns 6986. A PIN session's attempt to reset returns 6986. No PIN holds DELETE; a PIN that could delete the latch could re-create it anonymously at the BTC value (a missing object has no policy to refuse a create), which would open the seed key without S. That is why DELETE belongs only to the G0 and to the recovery credential, and the recovery wipe deletes MODE_PCR only after the seed wrap key has been confirmed gone.

Because a PCR is extend-only and SHA256 is one-way, the policy above gives:

- A PIN can move the latch forward (NEUTRAL to NOSTR, or x to ARM_x) but can never move it back to NEUTRAL, and can never land on BTC (it does not know S).
- Only the G0 can reset. The G0 resets only in two circumstances: once per physical boot (section 7.2), and from an armed SPENT or armed BTC value during a Bitcoin life (section 8.3). It never resets from NOSTR.
- Therefore, once a daily-PIN session has extended the latch to NOSTR in a given power-on life, no command from the ESP32, with or without any PIN, with or without a soft reboot, can bring the latch to BTC until the G0 has physically rebooted.

6. Where the gate lives: on the key objects, not the PINs

Silicon finding (11 September 2026): a REQUIRE_PCR_VALUE clause on a UserID (PIN) object is accepted at creation but is not evaluated when a session is opened against that object with VerifySessionUserID. Two throwaway PIN objects, one gated on MODE_PCR == BTC and one on MODE_PCR == NOSTR, both opened a session while the latch read NOSTR. The mode therefore cannot gate the PIN itself; it gates the key operations.

Second silicon finding (11 September 2026): when an object's policy carries two entries for the same auth object, only the first is consulted. Entries for different auth objects are OR'd. The policies below therefore carry at most one entry per auth object.

An extended entry carrying REQUIRE_PCR_VALUE is encoded as 2C || authObjId(4) || rights|0x00010000 (4) || pcrId(4) || expected(32), 45 bytes, and is satisfied only for that auth object's session and only while the named PCR equals the expected value.

AES_WRAP_SEED (Bitcoin seed outer layer)

Authenticated as	Rights	PCR clause
PIN_DAILY	ENC, WRITE, DELETE	none
PIN_SPEND	DEC	MODE_PCR == BTC
AUTH_RECOVER	DELETE	none

The daily PIN wraps the seed at onboarding (an encrypt; harmless) and can delete the key. The spend PIN can perform the only operation that opens the seed, decrypt, and only while the latch reads a value that S alone can produce. The G0's auth object has no entry on this key: before this change the G0's session held DEC, which meant that anyone holding K_SE, S and K_G0 (a readout of the G0 or an I2C tap during SETKEYS) could open the seed with no PIN. Now the SE itself demands a verified spend-PIN session.

AES_WRAP_DAILY (Nostr scalar outer layer)

Authenticated as	Rights	PCR clause
PIN_DAILY	ENC, DEC, WRITE, DELETE	none
AUTH_RECOVER	DELETE	none
AUTH_G0	DEC	MODE_PCR == NOSTR

The G0 strips the outer layer of the Nostr scalar inside its own SE session, and the SE allows that only while the latch reads NOSTR. MODE_PCR == NOSTR can only be reached by a daily-PIN extend from NEUTRAL, so this single clause already implies "the daily PIN was entered in this power-on life and the Nostr door was chosen".

A policy is fixed at object creation on this applet. Any policy change requires wiping the wallet and onboarding again on a fresh generation.

7. Boot and mode selection

7.1 The G0's one-shot arms and life token

In the G0's `main()`, which runs only on a reset of the G0 itself:

```
g_mode_reset_arm = 1    one MODE_PCR reset per physical boot
g_mode_btc_arm   = 1    one extend from NEUTRAL with S per physical boot
g_door           = UNSET no life declared yet
```

These are mirrored into TAMP backup register 0 (TAMP->BKP0R) as a life token: magic 0xA5F1 in bits 31:16, BTC arm spent in bit 11, reset arm spent in bit 10, door in bits 9:8, passphrase budget in bits 7:0. The backup domain survives NRST, software and watchdog resets and is lost only when VDD goes away.

On every G0 boot: if `RCC->CSR` does not show `PWRSTF` and a valid token is present, the token is restored (same life: door kept, spent arms stay spent). If `PWRSTF` is set, the life is fresh regardless of the token, so a battery pull always re-arms even if the rail decayed slowly off the capacitors. None of the three ESP32-reachable actions (I2C RESET command, ESP32's own reboot, Stop-mode sleep and wake) re-run `main()` or touch the arms. `wipe_transient()` and the RESET command deliberately leave the arms and the door alone.

7.2 Cold boot on the ESP32

The ESP32's own reset-reason register (`esp_reset_reason() == ESP_RST_POWERON`) decides only whether the Select Mode screen is shown. It is a UX signal and is forgeable; the authority on "fresh life" is the G0.

Sequence on an onboarded device:

1. ESP32 boots, mounts NVS, probes the G0 and waits out its self-test. The G0 reports the SE object generation; the ESP32 points every identifier at that generation.
2. If the reset reason is `POWERON` and this is not a ceremony boot, the ESP32 peeks the G0's door byte (SELFTEST register byte 19). If the G0 already holds `NOSTR` or `BTC` (an ESP32-only reset while the G0 kept running, for example the module EN pin pulled low), that door is resumed and no screen is shown. Otherwise the Select Mode screen is drawn with two cards, `NOSTR` ("Online identity") and `BITCOIN` ("Offline signing"), and the choice is saved in NVS (`sys/door`) for the soft reboots of this life. If cover mode is enabled the choice is deferred to the game's exit chord and the device boots provisionally as `NOSTR`.
3. If the reset reason is not `POWERON` the saved door is reloaded (or taken from the G0 if it holds one) and no screen is shown. A soft reset inside a Bitcoin life holds a `BITCOIN SESSION` screen ("Nostr is locked until a restart") before any radio is initialised; `ENTER` re-enters the Bitcoin session, holding both left buttons restarts.
4. The ESP32 sends the G0 `D00RNOSTR` (0x11) or `D00RBTC` (0x12). The G0 sets `g_door` if it was `UNSET`, accepts a re-send of the same door, and refuses the other door with 0xEE. On a refusal the ESP32 halts on a `RESTART NEEDED` screen. From this point `door_permits()` on the G0 refuses `MODEBTC`, `D00RKEYS`, `BTCSIGN` and `MSGSIGN` unless `g_door == BTC`, and refuses `SIGN`, `ECDH`, `PUBKEY`, `CACHEKEY`, `PROVISION`, `SESSIONTEST` when `g_door == BTC`. If no door has been declared, the first Nostr key command auto-commits `NOSTR` (`door_autocommit`), so a later `D00RBTC` in the same life hits first-wins.
5. The ESP32 sends the G0 `D00R1LOCK` (Door-1 PCR, separate mechanism) and then `MODERESET` (0x0A). The G0:
 - if `g_mode_reset_arm` is set: opens its own `SCP03` session as `AUTH_G0`, sends `WritePCR(reset)` on `MODE_PCR` (41 04 id), closes, clears the arm, saves the life token. The latch is now at the canonical `NEUTRAL` base.
 - if the arm is spent: does not write. It still reads the latch over its own session and returns its class (1 `NEUTRAL`, 2 `NOSTR`, 3 `BTC`, 4 `SPENT`, 5 `ARMed`, 0 wedged) in the diagnostic, with uptime, `RCC->CSR` at boot and whether this boot resumed a life token.
6. If the reset was refused, the ESP32 compares the latch class with the chosen door. `NOSTR` opens from `NEUTRAL` or `NOSTR`; `BITCOIN` opens from `NEUTRAL`, `SPENT`, `BTC` or `ARMed`. Anything else holds a `RESTART NEEDED` screen ("The key is still in a Bitcoin session", or Nostr) before any PIN is asked for. Every restart screen in the firmware shows the same gesture, holding both left buttons, which is the device's hardware power-off. Only an actual loss of board power re-arms the G0, and the G0 reports whether that happened: its `MODERESET` diagnostic carries the `RCC->CSR` flags captured at the top of `main()` and the uptime, which the ESP32 prints at boot as `[G0] boot evidence: uptime N ms | RCC_CSR ... (POWER NRST)`. That the gesture produces a `POWER` boot of the G0 was verified on the bench by the author; anyone with a unit and a serial monitor can repeat the check by holding the two left buttons and reading that line on the next boot.
7. Only now does a `NOSTR` boot start its radios. A `BITCOIN` boot (`g_ceremonyBoot`) forces the connectivity mode `OFFLINE` in RAM before any radio decision, and the main loop polls `WiFi.getMode()` every two seconds; if a WiFi driver ever appears in a Bitcoin life it is stopped, "RADIO TRIPWIRE" is logged and the session ends. There is no Bluetooth code in the firmware.

7.3 Choosing NOSTR

The daily PIN is entered on the device (or, after an on-device pairing, remotely through the PWA). The SE verifies it and opens a session against `PIN_DAILY`. Then, and only when this boot chose the `NOSTR` door and is not a ceremony boot,

`mode_pcr_set_nostr_in_open_session()` runs:

1. Read `MODE_PCR` in the daily session.
2. If it already reads `NOSTR`: no change (a second unlock in the same life must not extend again, which would wedge the latch).
3. If it reads anything other than `NEUTRAL`: refuse, "power cycle required".
4. Otherwise extend with "FAVILLA-MODE-NOSTR" (padded) and read back; it must equal `NOSTR`.

From here the G0 can strip the Nostr scalar's outer layer (the daily key's G0 entry is satisfied) and the connected role works.

The Bitcoin seed key's spend entry (MODE_PCR == BTC) cannot be satisfied by anything the ESP32 can do for the rest of this power-on life:

- a PIN session cannot reset (no DELETE right);
- a PIN session extending again lands on SHA256(NOSTR || anything), which matches no gate;
- the G0's MODEBTC refuses when its door memory says NOSTR (0xED) and, before that check could ever be bypassed, refuses when the latch reads NOSTR (0xEA); the reset arm was spent at boot, and the BTC arm, although still set in a Nostr life, is only consulted after both checks have passed;
- an ESP32 soft reboot does not re-run the G0's main(), and the life token keeps the door and the spent arms across any G0 reset that is not a loss of VDD.

In a Nostr life the daily PIN is also used purely as a device unlock on a soft reset inside a Bitcoin life (the door is BTC, so the NOSTR commit is skipped and the latch is left as is).

7.4 Choosing BITCOIN

Choosing BITCOIN at the Select Mode screen enters the Bitcoin session in place (no restart): the ESP32 marks the boot as a ceremony boot, radios are never initialised, and the loop runs the Bitcoin door. No daily PIN is involved. The spend PIN alone opens the Bitcoin role (section 8).

8. The Bitcoin door: arm, mint, open, burn

In a Bitcoin life, every operation that needs the seed (door entry, which derives the account xpubs, and every single PSBT input signature, and every BIP-137 message signature) runs the full sequence below. BTC exists in the latch for the length of one SE decrypt plus one G0 read; it is burned to SPENT before any result leaves the G0.

8.1 Arm (ESP32, spend-PIN session)

1. The SE verifies the spend PIN; a session against PIN_SPEND is open.
2. Read MODE_PCR. Classify: NEUTRAL, SPENT or a leftover BTC return 0 ("ask the G0"); an already ARMED value returns 3 (a soft reset between arm and mint); NOSTR or anything else returns -1 and the door is refused ("Door closed, power cycle").
3. Unless already armed, extend the latch with "FAVILLA-MODE-ARM" (padded) and read back; it must equal SHA256(previous || ARM). This is a PIN-session WRITE: it proves to the G0 that the SE verified a PIN since the last burn. The ESP32 cannot forge it (the extend is refused anonymously) and it is useless to an attacker (it does not reach BTC).
4. Close the session. The SE holds one session at a time, and the G0 needs the slot.

8.2 Mint (G0, MODEBTC 0x0B)

The G0 refuses outright if its door memory is not BTC (0xED), if it has no keys (0xE0) or no S (0xEF). Otherwise it opens its own SCP03 session as AUTH_G0 and reads the latch:

Latch reads	Action
ARM_NEUTRAL	Only if g_mode_btc_arm is still set: clear the arm, save the token, WritePCR(reset), then extend with S. Otherwise refuse (0xEE).
ARM_SPENT	WritePCR(reset), then extend with S. No limit: a legitimately entered Bitcoin life burned it after use, and each further signature in the same life goes through the spend PIN again.
ARM_BTC	Treated as ARM_SPENT: reset, then extend with S. A BTC value found lying in the latch (a crash inside the open window) is never accepted as-is.
bare NEUTRAL, SPENT or BTC	Refuse (0xEC): no PIN session armed it.
NOSTR or anything else	Refuse (0xEA). The G0 never resets from NOSTR.

The extend is WritePCR with TAG_1 = id and TAG_3 = S (32 bytes), inside the SCP03 session, so S never appears on the I2C bus in clear. After this the latch reads BTC = SHA256(NEUTRAL || S).

8.3 Open (ESP32, spend-PIN session again)

1. Reopen the spend session. Read MODE_PCR and verify it equals the stored BTC value; refuse otherwise ("Door verify failed").
2. Read the seed blob (96 bytes: IV_lo || AES(K_SEED_SE, inner[0:32]) || IV_hi || AES(K_SEED_SE, inner[32:64])) and its expected inner-layer hash from the G0 (SEEDGET; the blob is stored on the G0's write-once seed page, not on the ESP32).
3. Ask the SE to decrypt both halves with AES_WRAP_SEED. The SE grants this because the session is PIN_SPEND and MODE_PCR == BTC. The result is the G0's inner layer, AES-256-CBC under K_SEEDWRAP = SHA256(K_G0 || "favilla-g0-seedwrap-key"), useless to the ESP32.
4. Check SHA256(inner) against the G0's stored hash, pack it into a 96-byte inner packet, close the session.

8.4 Burn and use (G0, DOORKEYS 0x0E or BTCSIGN 0x0F or MSGSIGN 0x1B)

Before any SE traffic the G0 checks that the packet is the inner layer of the seed it stores (hash match) and of the right kind. Then, in `fetch_blob_se()`:

1. Open its own SCP03 session as AUTH_G0.
2. Read MODE_PCR. It must equal BTC computed from S; otherwise refuse (step 9, "no live BTC door").
3. Extend with "FAVILLA-MODE-SPENT" (padded) in the same session. The latch now reads SPENT, which matches no key gate. If this write fails, no seed is released.
4. Close the session, strip K_SEEDWRAP locally, derive (BIP-32, BIP-84, BIP-48), sign (BIP-143 ECDSA low-S or BIP-137), wipe the seed from RAM.

The ESP32 cross-checks the returned public key and sighash against its own for every input before accepting a signature. The next input repeats from 8.1: spend PIN again, arm from SPENT, G0 resets and re-mints, SE decrypts, G0 burns.

8.5 Leaving the Bitcoin life

The Bitcoin session ends on a SLEEPING screen and deep sleep (the G0 is told to enter Stop mode; it retains RAM, the life token and the door). Waking is an ESP32 reset with the door still BTC, so the BITCOIN SESSION screen is shown. There is no path from a Bitcoin life back to Nostr without a power cycle: the G0's door memory refuses the Nostr-side commands, and the latch reads SPENT, which the daily key's gate does not match.

9. Onboarding order

Onboarding must follow a real power cycle, because it needs the G0's reset one-shot. The boot-time MODERESET is skipped when the G0's SE objects are absent, so the arm is still available. The order, from `main.cpp`:

1. `se_begin_new_generation()`: pick the lowest generation 1..63 with none of its objects present. The G0 must report no live wallet.
2. `se_provision_recovery_auth()`: create AUTH_RECOVER first, because every later object names it (DELETE) in its policy.
3. Create PIN_DAILY and PIN_SPEND (self-policy only: each object's own session holds READ, WRITE, DELETE; no third party may delete a PIN).
4. Open the spend session. `mode_pcr_ensure_exists()`: create MODE_PCR with the section 5 policy, initial value zeros, no mode set. Close the spend session.
5. Open the daily session. `mode_btc_secret_begin()`: S from the SE TRNG; BTC value = SHA256(NEUTRAL || S) persisted to NVS.
6. `se_provision_g0_identity()`: create AUTH_G0 on the SE, send SETKEYS (K_SE || K_G0 || S || generation) to the G0, verify the acknowledgement, zeroise S on the ESP32.
7. `se_provision_aes_wrap_key_seed()` with the section 6 policy (its spend entry embeds the BTC value), then seed store: the G0 applies the inner layer (SEEDSEAL), the SE applies the outer layer in the daily session (ENC has no mode clause), the G0 stores the blob (SEEDSTORE).
8. `g0_mode_reset()`: the G0 resets the freshly created latch to the canonical NEUTRAL base (required: a created-but-never-reset PCR extends from a different base).
9. `mode_pcr_set_nostr_in_open_session()`: extend NEUTRAL to NOSTR on the daily session.
10. `se_provision_aes_wrap_key_daily()` (G0 entry gated on MODE_PCR == NOSTR), then the pair-wrap key and the Nostr scalar.

The device ends onboarding in the Nostr door of that power-on life. Any failure wipes the partial generation's secrets; the next attempt takes the next generation.

10. Wipe

"Delete Key" closes the daily session and asks the G0 for WIPEALL (0x14). The G0 opens a session as AUTH_RECOVER and deletes, in order: NOSTR_SCALAR, NSEC_IMPORT, AES_WRAP_DAILY, AES_WRAP_SEED, PAIR_WRAP, DOOR1_SECRET, MODE_PCR, AUTH_G0, AUTH_RECOVER. MODE_PCR and AUTH_RECOVER are deleted only if the seed wrap key is confirmed deleted or already absent (an R-MAC-verified result inside the SCP03 session; a forged status word from the relay would break the MAC chain). AUTH_G0 is deleted only once every object naming it is gone. The PIN objects and the Door-1 PCR are not deleted (no one holds that right) and remain as dead objects of a dead generation. The ESP32 erases its stored BTC value, then sends WIPEKEYS (0x08): the G0 erases K_SE, K_G0, S and the generation from its key page only if it has itself seen the recovery object deleted under a verified R-MAC in this power-on life (or an authenticated check shows it absent); otherwise the keys are kept so the wipe can be retried. After the wipe the device holds a restart screen, because the next onboarding needs the G0's physical-boot arm.

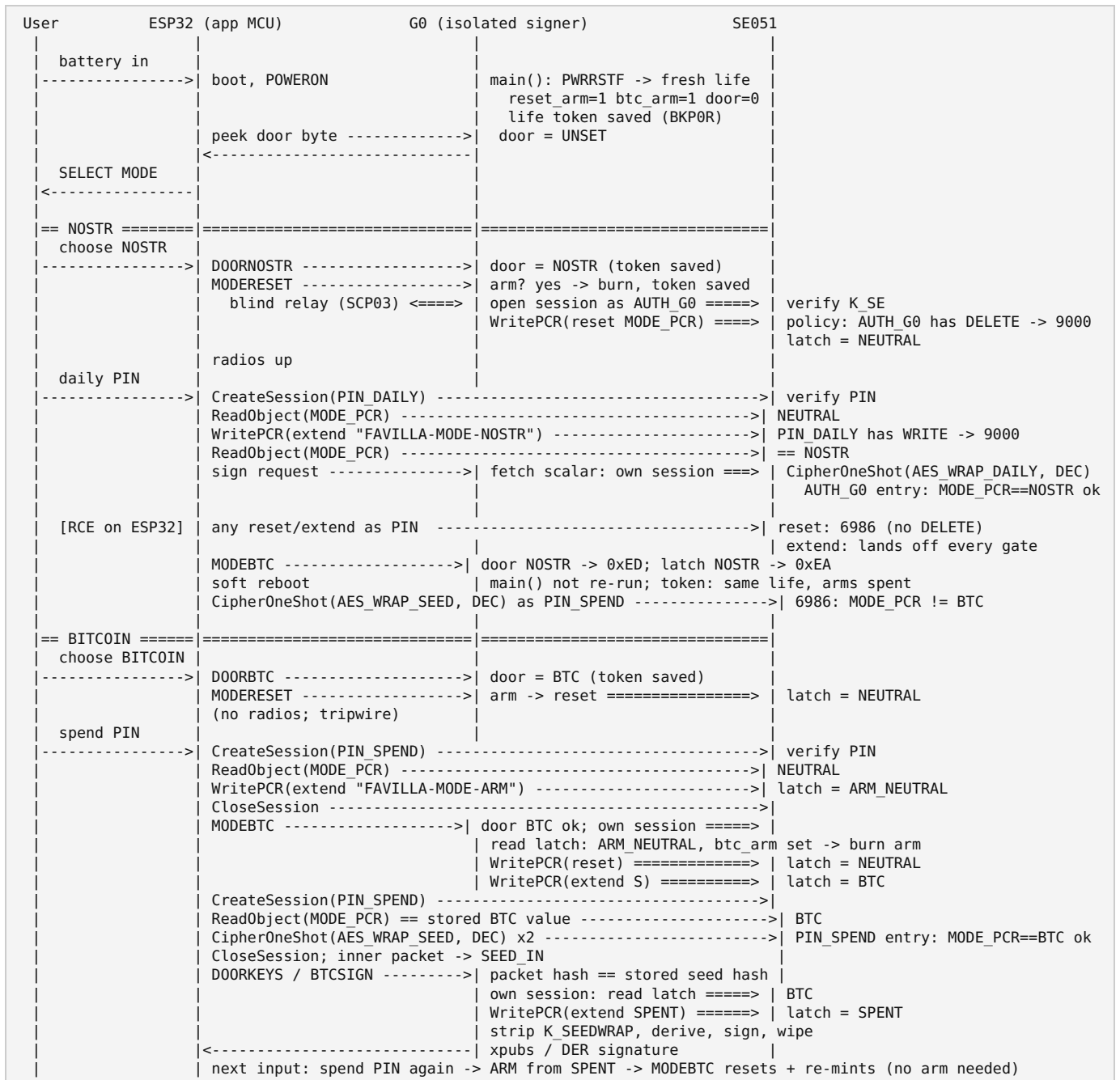
11. What the design does not claim

- It does not use a transient SE object or any SE-side power-cycle witness. All transient object types are refused on this applet, and a PCR-value clause on a PIN object is not evaluated at session open, so the SE cannot refuse the spend PIN by itself. The power-cycle proof lives in the G0's arms and life token, and in NEUTRAL, BTC and SPENT being unreachable

without the G0.

- It does not protect against an attacker with physical access to the device and the spend PIN, nor against readout of the G0's flash before read-out protection is set (RDP level 2 is a production step).
- It does not protect against code that persists in the ESP32's flash across a power cycle and then sends D00RBTC and the spend-PIN flow itself after the user has typed the PIN. That is the domain of ESP32 secure boot and flash encryption.
- It does not close transaction substitution: a signer that signs whatever sighash it is handed during a genuine ceremony. The ESP32 cross-checks pubkey and sighash per input, the user verifies outputs in the coordinator wallet, and multisig is recommended.
- The Door-1 gate (PCR_UNLOCK, DOOR1_SECRET) is a separate mechanism that ties the Nostr scalar to a daily-PIN unlock within a life and locks it at boot. It is not described here beyond the identifiers.
- The SE still opens the seed key for the milliseconds between the G0's extend to BTC and the G0's burn to SPENT. A compromised ESP32 inside a live Bitcoin life, after the user has typed the spend PIN, can obtain the inner layer in that window. The inner layer is still under K_SEEDWRAP, which never leaves the G0.
- The ESP32's reset-reason register and the Select Mode screen are convenience, not security. The G0's door memory is consulted first at every boot.

12. Sequence diagram: cold boot to mode selection to key gating



13. Threat table

--	--

Scenario	What the latch does
RCE on the ESP32 in a Nostr life tries to open the seed	PIN_SPEND decrypt refused 6986: MODE_PCR reads NOSTR, not BTC. The RCE cannot reset (no DELETE), cannot reach BTC by extending (no S), and the G0 refuses MODEBTC by door memory (0xED) and by latch class (0xEA).
RCE in a Nostr life reboots the ESP32 and chooses BITCOIN	The G0's main() did not run, the life token keeps door = NOSTR and the arms spent. DOORBTC is refused 0xEE; the ESP32 halts on RESTART NEEDED. Even if the ESP32 skipped that screen, MODERESET is refused (arm spent) and the latch still reads NOSTR.
RCE in a Nostr life phishes the spend PIN from the user	Same as above: the spend PIN opens a session but its only right on the seed key is DEC gated on BTC, which this life cannot reach. The PIN is useless without a power cycle, which evicts the RCE.
RCE pulls the G0's I2C lines or answers in its place	No S, no SCP03 key: no MODEBTC. The latch stays NEUTRAL, NOSTR or SPENT; none matches the seed key's gate. Bench-proven: SE answers 6986 on the decrypt.
Attacker obtains K_SE, K_G0 and S (G0 readout before RDP, or I2C tap during onboarding)	Can reset and extend the latch to BTC as the G0, but the seed key's DEC right belongs to PIN_SPEND only, so the SE still demands the spend PIN. A stolen inner packet is bound to the stored seed hash and cannot be replayed without a fresh BTC in the latch.
Stolen device, attacker knows the daily PIN	Nostr identity is exposed (that is what the daily PIN protects). The seed is not: the seed key grants the daily PIN ENC, WRITE, DELETE and never DEC.
Stolen device, attacker knows the spend PIN	Out of scope: physical access plus the correct PIN is the owner.
Brown-out induced by the ESP32 to re-arm the G0	The G0's POR threshold is below the ESP32's brown-out detector, so the ESP32 resets first; only flash-persistent code could exploit it, which is secure boot's domain. A real power loss re-arms by design.
Crash or power-off with BTC still in the latch	A found BTC is never trusted: the next door entry arms it and the G0 resets and re-mints. A dead or removed G0 leaves it, and the ESP32's own verify passes, but DOORKEYS/BTCSIGN cannot run without the G0, so no seed.
Malicious SE answers "NEUTRAL" to the G0	The G0's door memory is consulted before any latch read, so a Nostr-life MODEBTC is refused regardless of what the SE says.

14. Numbers and status words seen on silicon

- Anonymous reset or extend of MODE_PCR: 6986. PIN-session reset: 6986. Reset of a PCR already at rest: 6985. TAG_2 on an existing PCR: 6A80. Policy-refused decrypt: 6986. Delete of an absent object: 6985 (not 6A88). CreateSession on a locked PIN object: 6986.
- G0 cost per MODERESET or MODEBTC: about 5 to 7 relay round trips. Per signature in a Bitcoin life (arm, mint, SE decrypt, G0 burn and sign): about 0.8 to 0.9 s on the STM32G0B1 at 64 MHz.
- MODEBTC diagnostic byte 5: 0 ok, 0xEE armed-NEUTRAL without a fresh boot, 0xEC bare value (not PIN-armed), 0xEA NOSTR or non-canonical, 0xED wrong door, 0xEF no S, 0xE0 no keys, 0xEB latch unreadable.

15. Disagreements between the design notes and the code

The design notes (September 2026) and the handover brief describe an earlier version. The code wins; the differences are:

- Notes: MODE_PCR at fixed 0x20000033. Code: 0x20000033 + (generation << 8), generation 1..63 per onboarding.
- Notes: MODE_PCR policy grants READ, WRITE and DELETE to both PIN objects. Code: PINs hold READ and WRITE only; DELETE belongs to AUTH_G0 (which also has READ and WRITE) and to AUTH_RECOVER. No PIN may reset or delete the latch.
- Notes: BTC = SHA256(NEUTRAL || "FAVILLA-MODE-BTC"). Code: BTC = SHA256(NEUTRAL || S) with S a G0-held secret. The public-label value survives only as a logged legacy fallback.
- Notes: setting a mode is "reset then extend" in the PIN session. Code: PIN sessions cannot reset; they extend only from NEUTRAL (NOSTR) or extend to an ARM value (Bitcoin). Resets are the G0's alone.
- Notes: one door per power cycle, BTC persists. Code: BTC is minted per seed use and burned to SPENT by the G0 before any result leaves it; several signatures per Bitcoin life, each behind the spend PIN and a fresh mint.
- Notes: the seed key's spend-PIN entry carries ENC and DEC gated on BTC; the daily PIN holds DELETE as a second entry. Code: daily PIN ENC, WRITE, DELETE ungated; spend PIN DEC gated on BTC; one entry per auth object (same-object entries are not OR'd on this applet).
- Notes: both doors are walked at onboarding (spend session sets BTC). Code: onboarding never sets BTC. The spend session only creates PIN_SPEND and MODE_PCR; the daily session does the rest, with one G0 reset and one extend to NOSTR.
- Handover brief: G0 MODE_RESET (0x0A) is shelved and not in the shipping build. Code: MODERESET 0x0A and MODEBTC 0x0B are in the build and are the mechanism. MODEDEL 0x0C is retired (no handler).
- Notes and brief: "the only way back to BTC is a real power cycle, detected by the ESP32 reset reason". Code: the ESP32 reset reason only decides whether a screen is shown. The G0's arms, door memory and TAMP life token are the authority; a non-power G0 reset resumes the previous life.
- Brief: the SE051 VCC is wired straight to 3V3 and is never power-cycled. Code and G0 notes (13 October 2026): the SE's ENA pin is driven by G0 PB2; deep power-down is used for sleep. Persistent objects survive it, so the argument stands, but the wiring statement in the brief is out of date. The wiring was confirmed from the KeyV4 production netlist (IPC-356,

21 October 2026): U2 pin 12 (VIN) on +3V3; U2 pins 14, 15 and 18 (VCC) on a net carrying only the decoupling capacitor C30; U2 pin 11 (ENA) on net SE051_WAKE with U17 pin 17 (G0 PB2) and R33 (1 MΩ to +3V3).

11. Brief: deviceMode in NVS is retired. Code: deviceMode in NVS remains as the connectivity setting (offline, hotspot, network, auto). The door is a separate value (sys/door on the ESP32, g_door on the G0) and gating is by the SE latch and the G0.
12. Brief: a persistent header mode indicator exists. Code: mode_pcr_current_in_open_session() is defined but has no caller; the mode is communicated by the Select Mode, BITCOIN SESSION and RESTART NEEDED screens.
13. Notes: wipe deletes MODE_PCR via the daily PIN. Code: only the G0's WIPEALL over AUTH_RECOVER deletes it, after the seed wrap key is confirmed gone.
14. Brief: choosing BITCOIN reboots into a ceremony boot. Code: the Bitcoin session is entered in place with no restart (ceremonyEnterInPlace).
15. Brief: the G0 NRST wiring is to be confirmed from a schematic. Confirmed from the KeyV4 production netlist (IPC-356, 21 October 2026): net GO_RST carries exactly three pads, U17 pin 6 (NRST), C46, and J6 pin 3 (the TC2030 footprint). No ESP32 pin and no other component is on that net.
16. The handover brief states the Select Mode screen appears "nowhere else" than on a POWERON reset. Code: it is also suppressed on POWERON when the G0 already holds a door, on a ceremony boot, and in cover mode (deferred to the exit chord).

Revision history

- 1.2, 21 October 2026: title shortened to "The Two-Door Mode Latch for a Multi-Role Hardware Signer". No other change.
- 1.1, 21 October 2026: title and the opening sentence of section 1 corrected. The device has three chips; the earlier wording "single-MCU hardware signer" described the problem the design avoids, not the device. The latch separates the two roles of the one application processor. No change to the mechanism or to any other section.
- 1.0, 21 October 2026: first publication.

Favilla Pty Ltd publishes this note so that the mechanism is freely available for anyone to implement. No patent application has been filed for it as of the date above.