SWITCH Lifecycle & PassKey Install — Release 1

Kenneth J Hamer-Hodges

May 2026

SWITCH Instruction — Full Lifecycle and Chain of Trust

v1.0 — 2026-05-02 CONFIDENTIAL

Overview

SWITCH is the only instruction in the Church Machine that can write to the system-wide privileged capability registers CR13 (interrupt handler) and CR15 (namespace root). Understanding it fully requires tracing three phases: the one-time namespace birth that creates the root of trust, the boot sequence that loads the initial namespace, and the runtime PassKey ceremony that authorises any subsequent privileged register replacement.

The hardware FSM checks (type, sentinel, target) are necessary but not sufficient for security. The full guarantee depends on the chain of trust established in software upstream of the instruction.

Phase 0 — Namespace Birth

Occurs once, at manufacture or first system configuration. Never repeated for the same namespace instance.

```
Hardware RNG
     │
     │  128-bit secret — known to nobody yet
     ▼
Mint.Encode(
    type    = Abstract,
    ab_data = secret,
    perms   = {} (none)
)
     │
     │  produces NS_IDENTITY_GT (32-bit Abstract GT, no NS entry)
     ▼
     ├──► Stored in Mint's private c-list
     │    (Mint's root of trust — never shared further)
     │
     └──► Returned to the namespace creator
          (kept as the "owner credential")
```

After this phase exactly two parties hold the NS Identity: **Mint** and the **original namespace owner**. No other code path can produce it or derive it.

This is the “lead into gold” step. The random number is inert data — lead. Running it through Mint produces an unforgeable Abstract GT — gold — that the hardware will accept as proof of identity.

Phase 1 — Boot (every power-on)

The boot ROM runs with M-elevation permanently asserted, bypassing the L-permission check inside mLoad. NS integrity32 is still verified.

```
Power-on / Reset
     │
     ▼
B:00  FAULT_RST   clear all CRs, DRs, M-elevation ON, LEDs = 0b000001
     │
     ▼
B:01  LOAD_NS     mLoad( NS[0] = Boot.NS ) → CR15
                  ↑ M-elevation bypasses L-perm; integrity32 still checked
                  ↑ CR15 goes from NULL → namespace root
     │
     ▼
B:02  INIT_THRD   mLoad( NS[1] = Boot.Thread ) → CR12
     │
     ▼
B:03  INIT_ABSTR  mLoad( NS[3] = Boot.Abstr ) → CR6
     │
     ▼
B:04  LOAD_NUC    CR14 ← Boot.Abstr code region
     │
     ▼
Boot.Abstr executes
     │
     └──► calls Startup.Config.Execute()
               └──► calls configured main abstraction (NS[entry_slot])
```

After boot:

Register	Contents	Scope
CR12	Thread Stack GT	System-wide; unchanged by CHANGE
CR13	NULL or boot default	System-wide; unchanged by CHANGE
CR14	Boot.Abstr code GT	Per-thread; re-derived on every CALL
CR15	Boot.NS root GT	System-wide; unchanged by CHANGE

SWITCH is **not** used during normal boot. The boot ROM’s M-elevation is the bootstrap mechanism. SWITCH is only needed to replace CR13 or CR15 while the machine is already running.

Phase 2 — PassKey Issuance (runtime)

Required before any SWITCH can execute. The caller must prove namespace ownership to Mint before Mint will produce a PassKey.

```
Caller holds:
  • NS_IDENTITY_GT  (proof of namespace ownership — from Phase 0)
  • new capability they want installed in CR15 (or CR13)
          │
          ▼
Caller → Mint.IssuePassKey( NS_IDENTITY_GT, target = CR15 )
          │
          ▼
     ┌──────────────────────────────────────────────┐
     │  Does NS_IDENTITY_GT match the copy          │
     │  stored in Mint's private c-list?            │
     └──────────────────────────────────────────────┘
          │                    │
        MATCH              NO MATCH
          │                    │
          ▼                    ▼
   Mint.Encode(           REFUSE — fault
     type    = Abstract,
     ab_data = <internal key>,
     word1_location = 0xFFFFFFFF   ← sentinel for CR15
   )                              (0xFFFFFFFE for CR13)
          │
          ▼
   PassKey Abstract GT (32-bit word)
   + sentinel baked into word1_location by Mint
          │
          ▼
   Mint.Transfer( PassKey → caller's CR, e.g. CR0 )
   B=0: the PassKey cannot be copied further by user code (mSave faults)
```

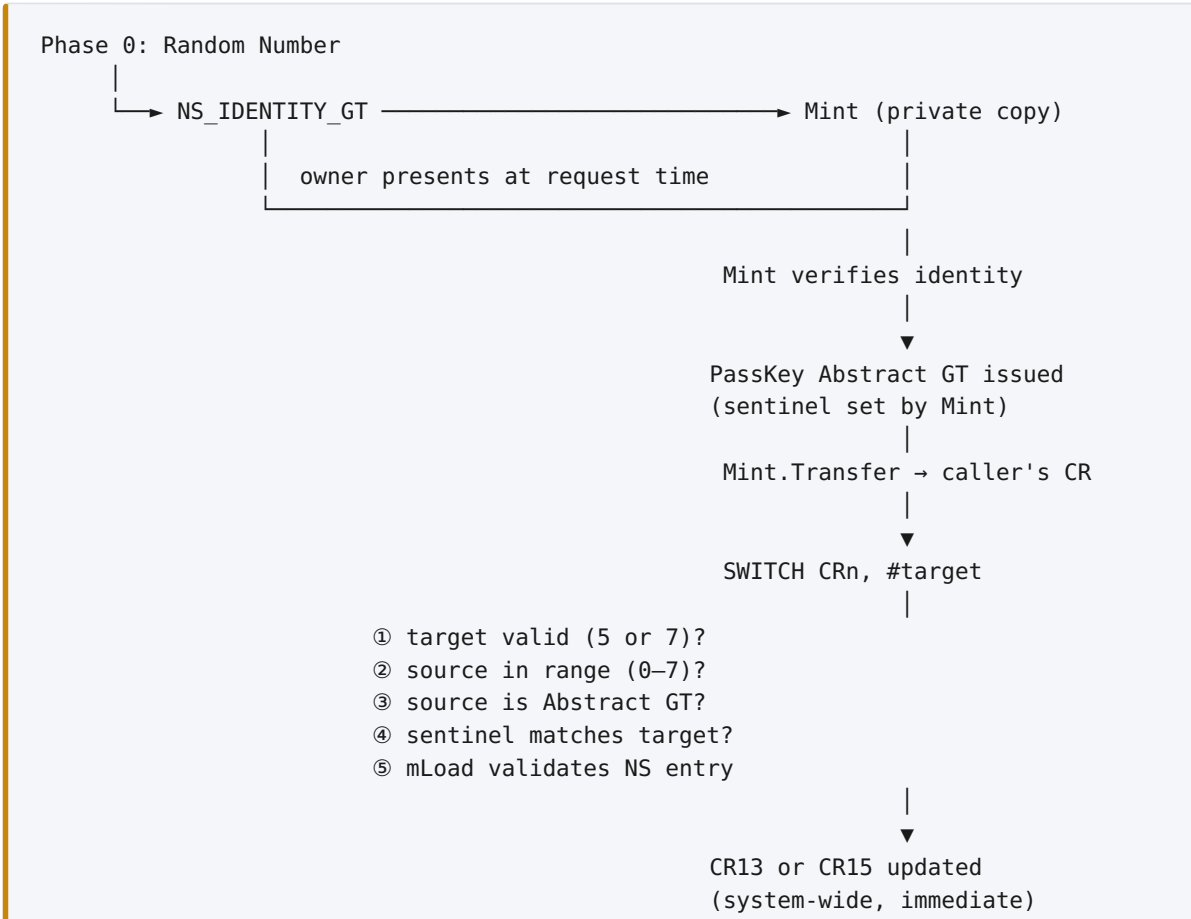
The sentinel values (`0xFFFFFFFF` for CR15, `0xFFFFFFFE` for CR13) sit at the top of the hardware I/O address range — addresses no live lump base can ever occupy — so there is no ambiguity with a real memory capability.

Phase 3 — The SWITCH Instruction Executes

Hardware FSM in *`hardware/switch.py`*. Five sequential checks before mLoad.

```
SWITCH CR0, #7       (CR0 holds PassKey; #7 = Tgt field for CR15)

  dest_cr = CR8 + Tgt = CR8 + 7 = CR15

 ① CHECK_TARGET
       Tgt ∈ {5, 7}?   (5 → CR13,  7 → CR15)
       NO  → INVALID_OP fault
       YES → ②

 ② CHECK_SRC
       source CR index in range 0–7?
       NO  → INVALID_OP fault
       YES → ③

 ③ READ_SRC
       latch full contents of CR0 (the PassKey capability)
       ↓

 ④ CHECK_PASSKEY_TYPE
       CR0.word0_gt.gt_type == 0b11 (Abstract)?
       NO  → INVALID_OP  (Inform, Outform, NULL all rejected)
       YES → ⑤

 ⑤ CHECK_PASSKEY_SENTINEL
       CR0.word1_location == 0xFFFFFFFF  (sentinel for CR15)?
       NO  → INVALID_OP  (wrong target — e.g. a CR13 PassKey cannot SWITCH CR15)
       YES → ⑥

 ⑥ ChurchMLoad( src=CR0, dst=CR15, m_elevated=1 )
       – L-permission check bypassed (m_elevated)
       – bounds check on PassKey's c-list limit
       – NS entry integrity32 verified
       – gt_seq revocation check
       – g-bit reset
       – CR15 ← new capability
       – thread table shadow updated

     SUCCESS:  CR15 now holds the new namespace root capability
               CR0 is UNCHANGED — one-way install, not a swap

     FAULT:    any mLoad validation failure propagates normally
```

The Complete Chain of Trust



Five Guarantees This Chain Provides

Guarantee	Enforced by
Only the namespace owner can trigger a SWITCH	Mint refuses PassKey without NS_IDENTITY_GT
A PassKey for CR13 cannot install into CR15	Sentinel check in hardware FSM (⑤)
Any ordinary capability (Inform/ Outform) cannot be SWITCHed	Abstract GT type check in hardware FSM (④)
The capability being installed is authentic and unrevoked	mLoad NSGate: integrity32 + gt_seq check (⑥)
The old CR15/CR13 value is not secretly moved anywhere	One-way install — source CR unchanged, no swap

What the Hardware Alone Cannot Enforce

The three hardware FSM checks (Abstract type, sentinel, target validity) are necessary conditions, not sufficient ones. They verify the form of the PassKey but know nothing about whether Mint verified the NS Identity before issuing it. That verification is a software-level protocol and depends on:

1. **Mint's private c-list** being unreadable by any user-level code
2. **NS_IDENTITY_GT** never being placed in a bindable (B=1) c-list slot, preventing propagation beyond the initial two holders
3. **Mint refusing** all PassKey requests that do not present the matching NS_IDENTITY_GT

If any of these three software conditions fail, the hardware checks are insufficient — a forged PassKey could reach SWITCH. The hardware is the last line of defence, not the only one.

Simulator Status (D-11 — CLOSED, Task #880)

The simulator `_execSwitch` was rewritten in May 2026 (Task #880) to match hardware exactly. The old atomic CR swap is gone. The new implementation enforces all three hardware checks in the same order as `hardware/switch.py`:

1. **Target validity** — `imm & 0x7` must be 5 (→CR13) or 7 (→CR15); else `FAULT(INVALID_OP)`
2. **PassKey type** — source must be Abstract GT (type 3); NULL and non-Abstract → `FAULT(INVALID_OP)`
3. **Sentinel check** — `source.word1` must be `0xFFFFFFFE` (CR13) or `0xFFFFFFFF` (CR15); else `FAULT(INVALID_OP)`

One-way install: `this.cr[destCR] = { ...srcCR }`. Source CR unchanged.

The assembler also now rejects crSrc > 7 for SWITCH (CR8–CR11 would silently truncate to CR0–CR3 in the hardware 3-bit crSrc field).

D-11 is closed. Simulator and hardware are in agreement.

Related Files

File	Role
`hardware/switch.py`	Hardware FSM — target check, PassKey type check, sentinel check, mLoad call
`hardware/mload.py`	ChurchMLoad — NS validation, g-bit reset, CR write
`hardware/hw_types.py`	Sentinel constants, GT type constants, target Tgt values
`simulator/simulator.js`	`_execSwitch` — PassKey-gated one-way install (D-11 closed)
`simulator/assembler.js`	SWITCH case 5 — crSrc ≤ 7 guard added
`docs/HARDWARE-DEVIATIONS.md`	D-11 — CLOSED; all deviations resolved
`docs/mint.md`	Mint issuance protocol, Abstract GT note (§9)
`docs/abstract-gt.md`	Abstract GT word layout and ab_type registry
`docs/namespace-security.md`	mLoad validation pipeline, NS entry format

Confidential — Kenneth Hamer-Hodges — May 2026