Namespace Security — Release 1

Church-Turing Meta-Machine

Kenneth J Hamer-Hodges

May 2026

Namespace and Security Model

v1.0 — 2026-04-29 CONFIDENTIAL

Namespace Table Structure

The namespace is the master directory of all resources in the system. Every Golden Token references an entry in the namespace table. Each entry describes a resource with three 32-bit words:

Word	Content
Word 0	Base address — 32-bit lump base byte address
Word 1	Limit + gt_seq (`spare[3] | g_bit[1] | gt_seq[7] | limit_offset[21]`)
Word 2	integrity32 check (`integrity32(Word 0, Word 1 with g_bit cleared)`)

Namespace Entry Format

Each namespace entry occupies exactly **3 consecutive 32-bit words** (12 bytes). The slot byte address is calculated as:

```
NS_entry_addr = NS_table_base + slot_id × 12
```

The namespace table supports up to **65,536 entries**, bounded by the 16-bit `slot_id` field in the Golden Token.

Word 1 Layout (WORD2_LAYOUT)

```
31    29 28     27      21 20                  0
┌────────┬──┬────────────┬──────────────────────┐
│ spare  │ G │  gt_seq   │   limit_offset       │
│ [2:0]  │   │  [6:0]    │     [20:0]           │
└────────┴──┴────────────┴──────────────────────┘
```

Bits	Field	Description
[20:0]	`limit_offset`	Object size in words minus 1
[27:21]	`gt_seq`	Revocation counter — compared against GT `gt_seq` by ChurchNSGate
[28]	`g_bit`	GC mark bit — cleared on every ChurchNSGate access (G=0 = reachable)
[31:29]	spare	Reserved

Word 2 Layout

Word 2 holds the full 32-bit `integrity32` check value, computed over Word 0 and Word 1 (with `g_bit` cleared before the computation). It is an opaque 32-bit word with no internal sub-fields visible to hardware.

Bits	Field	Description
[31:0]	`integrity32`	32-bit parallel integrity check over Word 0 and Word 1 (g_bit cleared)

Note on B and F flags: The B (Bind) flag lives in **bit [31] of GT Word 0** itself (`b_flag` in GT_LAYOUT) — it is a property of the token, not the NS entry. The F (Far/Foreign) concept is represented by the `gt_type` field in the GT (`gt_type=10` for Outform). Neither B nor F has a dedicated field in the 3-word NS entry.

The mLoad Master Validation Path

All namespace access in the Church Machine architecture routes through a single trusted validation path called **mLoad** (implemented as `ChurchMLoad` + `ChurchNSGate` in Amaranth HDL). This is the fundamental security principle: one master validation pipeline that every Church instruction must use to access the namespace.

Why One Path

Having a single validation path: - **Minimizes the Trusted Computing Base (TCB)**: Only one piece of code needs to be correct for all namespace access. - **Eliminates validation gaps**: No instruction can bypass permission checks, bounds checks, integrity32 validation, or g_bit reset. - **Maps directly to hardware**: In ASIC/FPGA implementations, mLoad + ChurchNSGate form a single pipeline — there is no way to access namespace memory without passing through them.

mLoad / ChurchNSGate Validation Sequence

Every namespace access follows this exact sequence. Any failure at any step triggers an immediate FAULT:

```
mLoad(source_capability, required_permission, index, destCR):

  1. Permission Check (mLoad — CHECK_L state)
     Does the source capability have L or M permission?
     (requiredPerm=null skips this check — used for RETURN context restoration)
     Failure → FAULT

  2. Bounds Check (mLoad — CHECK_BOUNDS state)
     Is the index within the source C-List range?
     Index must be < ns_w2.limit_offset
     Failure → FAULT

  3. Fetch Golden Token (mLoad — FETCH_GT state)
     Read the 32-bit GT from the C-List at the given index.

  4. NS Slot Bounds Check (mLoad — CHECK_NS state)
     Is the GT's slot_id within the CR15 namespace range?
     Failure → FAULT

  5. Fetch NS Entry (ChurchNSGate — FETCH_LOC / FETCH_W2 / FETCH_W3 states)
     Read NS Entry Word 0 (base), Word 1 (gt_seq + limit_offset + g_bit), Word 2
(integrity32).

  6. gt_seq Match (ChurchNSGate — CHECK_VERSION state)
     Does GT gt_seq [22:16] match NS Entry Word 1 gt_seq [27:21]?
     Failure → FAULT VERSION (stale token — entry was revoked or GC'd)

  7. integrity32 Check (ChurchNSGate — CHECK_VERSION state)
     Recompute integrity32 over NS Word 0 (base) and NS Word 1 (limit/gt_seq, with
g_bit cleared).
     Compare against NS Entry Word 2.
     Failure → FAULT SEAL (tampered NS entry)

  8. G-bit Reset (ChurchNSGate — CHECK_VERSION state)
     Clear g_bit=0 on the accessed namespace entry (NS Entry Word 1 bit [28]).
     This is unconditional — happens on every successful access.
     (GC integration: signals that this entry is reachable)

  9. Write to Destination CR (mLoad — COMPLETE state)
     Write the full capability (GT + NS entry data) to the destination register.
     This is the SOLE path for writing to any CR.

 10. Thread Table Shadow Update (mLoad — UPDATE_THREAD state)
     Write the GT word to the thread table shadow.
     Keeps the thread table continuously current.
```

The Golden Rule: mLoad Is the Sole Path for All CR Writes

No instruction directly writes to a capability register. All CR writes route through mLoad, which:

1. **Validates** the GT against the namespace (gt_seq, integrity32, bounds)
2. **Resets g_bit=0** on the accessed namespace entry (GC liveness)
3. **Writes** the validated capability to the destination CR

4. **Updates** the thread table shadow at Thread[CRd]

This means: - **LOAD**: mLoad validates and writes to CRd - **CALL**: cLoad runs NSGate + lump header read → writes CR6 (c-list) and CR14 (code) - **RETURN**: mLoad (direct mode: `sub_direct=1`) revalidates saved E-GT against namespace — catches use-after-free - **SWITCH**: mLoad validates source GT and writes to privileged register CR12–CR15 - **CHANGE**: Saves data registers + PC to thread lump; CRs reloaded via cLoad on next CALL/RETURN

Instructions Using mLoad / ChurchNSGate

Instruction	Validation Path	Destination
LOAD	ChurchMLoad + ChurchNSGate	CRd (user-specified)
CALL	ChurchMLoad → ChurchNSGate → cLoad	CR6 (c-list), CR14 (code)
RETURN	ChurchMLoad (direct mode)	CR6, CR14 (re-derived from saved E-GT)
CHANGE	ChurchMLoad + ChurchNSGate	CR12–CR15 (privileged: CR12=thread stack, CR13=interrupt, CR14=code, CR15=namespace)
SWITCH	ChurchMLoad + ChurchNSGate	CR12–CR15 (system)
SAVE	Bounds check only (no NSGate)	Namespace write (GT Word 0)

---

integrity32 Integrity Validation

integrity32 validation is the mechanism that ensures Golden Tokens and namespace entries have not been corrupted or forged.

When Validation Occurs

Validation occurs inside ChurchNSGate on every mLoad call — which means every Church instruction that reads from the namespace:

Operation	Validation
LOAD	gt_seq match + integrity32 checked
CALL	gt_seq match + integrity32 checked
RETURN	gt_seq match + integrity32 checked (on saved E-GT)

How Validation Works

When ChurchNSGate processes a namespace entry:

1. The **`gt_seq`** in the Golden Token (bits [22:16]) is compared against the `gt_seq` in NS Entry Word 1 (bits [27:21]). If they do not match, the token is stale and a `VERSION` FAULT is triggered.
2. The **integrity32** value is recomputed over NS Word 0 and NS Word 1 (with `g_bit` cleared) and compared against NS Entry Word 2. If they do not match, a `SEAL` FAULT is triggered.

Revocation: increment the `gt_seq` counter in NS Entry Word 1 by 1. All existing GTs for that entry now have a stale `gt_seq` and FAULT on next use. No tracking of outstanding GTs is required — revocation is O(1).

Failsafe Principle

The Church Machine architecture follows a strict failsafe design: **any validation failure triggers a FAULT, handled by a single fault handler**. There are no partial failures, no silent degradation, and no undefined behaviors. The system is either operating correctly or it is faulted.

This applies uniformly to: - Permission violations (missing required permission bit) - gt_seq mismatches (stale Golden Token) - integrity32 failures (corrupted or tampered NS entry) - Bounds violations (index out of range) - Stack overflows (call stack full) - Stack underflows (return with empty stack)

The fault handler is the single point of error management, ensuring consistent and predictable behavior regardless of the failure mode.

Security Invariants

The Church Machine enforces the following invariants at all times:

No Direct Privileged Register Access

Only CR0–CR11 are addressable through the 4-bit register encoding in Church instructions. Privileged registers CR12–CR15 are physically unreachable through instruction encoding. This is an architectural constraint, not a software convention.

Privilege Through SWITCH Only

The SWITCH instruction is the sole mechanism for writing to privileged registers CR12–CR15. It requires appropriate permissions on the source capability.

Capability-Mediated Access Through mLoad

All resource access goes through capability-mediated C-Lists via the mLoad + ChurchNSGate validation path. LOAD reads from a C-List entry. SAVE writes a GT Word 0 back to memory. There is no instruction that can access namespace entries without a valid Golden Token authorizing the operation. The mLoad path ensures that every access is validated, bounds-checked, integrity32-verified, and g_bit-reset.

Mutually Exclusive Permission Domains

The two mutually exclusive permission domains (Turing and Church) cannot be mixed within a single operation context. This prevents confused deputy attacks where a capability intended for one purpose is misused for another.

Domain	Permissions	Operations
Turing	R, W, X	Read/Write data, Execute code
Church	L, S, E	Load/Save Golden Tokens through C-Lists, Enter abstractions

M (Machine) is a transient microcode elevation (`sub_m_elevated`), never stored in the GT. `g_bit` is the GC mark bit in NS Entry Word 1 bit [28], managed by ChurchNSGate.

G-bit Reset as Security Invariant

The G-bit reset on every namespace access is not optional — it is a security invariant enforced by ChurchNSGate. This ensures that the garbage collector can accurately determine which entries are reachable, preventing: - **Use-after-free**: Revoked entries have their `gt_seq` incremented, instantly invalidating stale Golden Tokens. - **Resource leaks**: Unreachable entries are identified and reclaimed. - **GC evasion**: No instruction can access namespace without triggering G-bit reset.

See Also

- **Lump-Architecture.md — Lump object structure, Header Word encoding, and zone layout**

  Confidential — Kenneth Hamer-Hodges — April 2026