Golden Tokens — Release 1

Church-Turing Meta-Machine

Kenneth J Hamer-Hodges

May 2026

Golden Tokens

v1.0 — 2026-04-29 CONFIDENTIAL

What Are Golden Tokens?

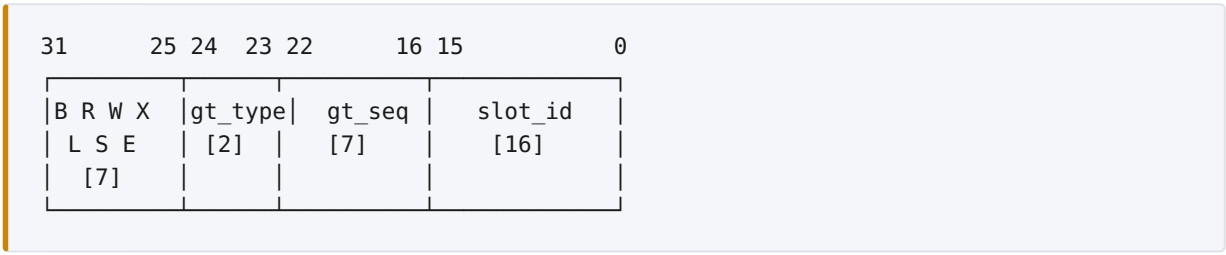
Golden Tokens (GTs) are the fundamental unit of access control in the Church Machine architecture. Every access to a resource – whether loading data, calling a service, or switching privilege levels – requires a valid Golden Token that grants the necessary permissions. Golden Tokens are unforgeable: they cannot be fabricated by software, only created and managed through hardware-enforced mechanisms.

A Golden Token encodes three things: 1. **What** resource it refers to (via `slot_id` — the namespace slot ID) 2. **What operations** are permitted (via permission bits) 3. **Whether it is authentic** (via integrity32 parallel check on the NS entry)

Without a valid Golden Token, no operation proceeds. Any attempt to use an invalid, expired, or insufficient token results in a FAULT.

GT Format

The Church Machine uses a 32-bit Golden Token with a precisely defined bit layout:



Bits	Field	Width	Description
[15:0]	`slot_id`	16	Namespace slot ID (0–65,535)
[22:16]	`gt_seq`	7	Revocation sequence counter; must match NS Entry Word 1 `gt_seq`
[24:23]	`gt_type`	2	GT class (NULL / Inform / Outform / Abstract)
[30:25]	`perms`	6	R, W, X, L, S, E (see below)
[31]	`b_flag`	1	Bind flag — 1 = GT may be propagated via mSave

Each capability register in Church Machine is 128 bits wide (4 x 32-bit words):

Word	Content
word0	The 32-bit Golden Token
word1	Lump base address (from NS Entry Word 0)
word2	NS Entry Word 1: `spare[31:29] | g_bit[28] | gt_seq[27:21] | limit_offset[20:0]`
word3	NS Entry Word 2: integrity32 check (32-bit parallel check)

GT Permission Bits (Church Machine)

The GT stores exactly 6 permission bits in bits [30:25]. These are the mutually exclusive access rights – three for data operations (Turing domain) and three for capability operations (Church domain):

Bit (in perms field)	Absolute bit	Name	Domain	Description
0	25	R	Turing	Read data from the referenced resource
1	26	W	Turing	Write data to the referenced resource
2	27	X	Turing	Execute code at the referenced location
3	28	L	Church	Load a Golden Token from a C-List via mLoad
4	29	S	Church	Save a Golden Token to a C-List via mSave
5	30	E	Church	Enter an abstraction (call a service)

Domain Purity

A GT may carry Turing permissions (R, W, X) **or** Church permissions (L, S, E), but **never both**. This is enforced in hardware at TPERM time — any attempt to create a mixed-domain GT raises a DOMAIN_PURITY fault.

E Isolation

Within the Church domain, E (Enter — invoke an abstraction) must be **standalone**. E may not be combined with L (Load from c-list) or S (Save to c-list). A token that combines E with L or S would allow its holder to both traverse the nodal c-list and enter the abstraction it contains — an attack path that bypasses the separation between the capability list and the code it holds. E is the entry key to a function; L and S are the keys to the capability list that owns it. They must never be the same key.

```
Valid:   R, W, X, RW, RX, RWX             (Turing pure)
Valid:   L, S, E, LS                       (Church pure — E standalone)
Invalid: RL, WL, XE, RE, WS, RWXE, RWXL  (cross-domain — any mix of {R,W,X} with
{L,S,E})
Invalid: LE, SE, LSE                       (E isolation — E combined with L or S
exposes abstraction internals)
```

B Flag — Bind

`B` (bit 31 of GT Word 0) controls whether the GT may be propagated to another c-list via mSave:

- **B=0** (default): the GT cannot be copied out of its current c-list — mSave FAULTs.

- **B=1**: the GT is bindable — mSave permits the write.

B is set by the IDE at lump creation time and may be cleared by CALL on preserved CRs passed to the callee (“no bind by default”).

M Permission – Transient Microcode Elevation

M is **not stored in the GT**. It exists only as a transient signal (`sub_m_elevated`) that microcode asserts during mLoad execution. When mLoad completes, M is gone. No user instruction can set, test, or observe M. This prevents privilege escalation.

GT Type Field (`gt_type`, bits [24:23])

The Church Machine includes a 2-bit type field classifying the nature of the referenced resource:

Value	Type	Description
00	NULL	Empty / invalid — always faults on use
01	Inform	GT points to a lump or data object in local memory via an NS entry
10	Outform	GT references an IDE-managed dependency (lazy-loaded via Locator)
11	Abstract	GT IS the value — constants, immutable credentials, PassKey tokens

NULL is architecturally distinct from all valid reference types. Any GT with `gt_type=00` immediately faults at ChurchNSGate before any NS lookup is performed.

---

gt_seq Field (bits [22:16])

Church Machine includes a 7-bit `gt_seq` field in bits [22:16] of the Golden Token. This revocation counter is critical for namespace integrity and garbage collection safety:

- Each namespace entry stores a corresponding `gt_seq` value in NS Entry Word 1 bits [27:21].
- When a GT is used (LOAD or CALL), ChurchNSGate compares `gt_word0.gt_seq` against the NS entry’s `gt_seq`. A mismatch means the GT has been revoked — access FAULTs immediately.

- During garbage collection, reclaimed entries have their `gt_seq` incremented. All outstanding GTs referencing that entry become stale instantly.
- 7 bits gives 128 revocation generations before wraparound.

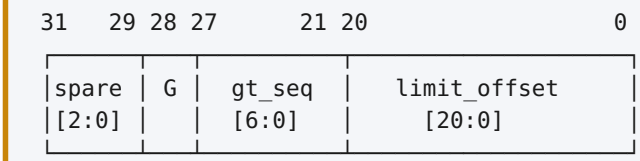
Namespace Entry Format

Each namespace entry occupies **3 consecutive 32-bit words** (12 bytes). The slot byte address is `slot_id × 12` (or equivalently `slot_id × 3` words) from the NS table base. The NS table supports up to **65,536 entries** (bounded by the 16-bit `slot_id` field).

Word 0 — Base Address

The 32-bit lump base byte address.

Word 1 — Limit + gt_seq (WORD2_LAYOUT)



Bits	Field	Description
[20:0]	`limit_offset`	Object size in words minus 1 (21-bit)
[27:21]	`gt_seq`	Revocation counter; compared against GT `gt_seq` by ChurchNSGate
[28]	`g_bit`	GC mark bit — may be set by GC; masked before integrity32 check
[31:29]	spare	Reserved

Word 2 — integrity32 Check

The 32-bit integrity32 parallel check result, computed over NS Entry Word 0 and Word 1 (with `g_bit` masked to zero before the check).

integrity32 Integrity

ChurchNSGate recomputes integrity32 over NS Entry Word 0 and Word 1 (`g_bit` cleared) and compares against NS Entry Word 2. A mismatch faults with `SEAL` error, preventing use of any tampered NS entry.

Capability Registers

The Church Machine provides 16 capability registers (CR0–CR15), divided into two groups:

Instruction-Addressable Registers (CR0–CR11)

These registers are directly accessible by Church instructions through a 4-bit register encoding field. Software can freely read and manipulate GTs in these registers using LOAD, SAVE, and other Church instructions.

Privileged Registers (CR12–CR15)

These registers are protected from direct instruction access. The only way to write to a privileged register is through the SWITCH instruction, which requires appropriate permissions. This architectural constraint prevents privilege escalation through direct register manipulation.

Special Register Roles

Register	Name	Role
CR6	C-List	Current capability list — the set of capabilities available to running code
CR12	Thread Stack	Thread stack capability (privileged, system-wide; loaded at boot B:02)
CR13	Interrupt	System-wide interrupt handler capability (privileged, unchanged by CHANGE)
CR14	Code/CLOOMC	Current code GT — instruction fetch source (privileged, per-thread; re-derived by CALL)
CR15	Namespace	Namespace root — defines the security boundary of the entire system

CR6 and CR14 are re-derived by CALL/RETURN via mLoad. CR12 is saved and restored by CHANGE (thread switching). CR13 and CR15 are system-wide and unchanged by CHANGE. The privileged zone (CR12–CR15) cannot be addressed by normal programmer instructions.

Cross-references

- `architecture.md` — Overall Church Machine architecture
- `Lump-Architecture.md` — Accessible overview of the Lump object model
- `CM_LUMP_SPECIFICATION.md` — Authoritative binary encoding and field specification

---

Confidential — Kenneth Hamer-Hodges — April 2026