THE SIX LAWS OF CLOOMC

An Independent Review -- The Church-Turing Machine Model | Security by Construction, Not Convention

Every computer you use runs on an architecture designed in the 1940s. The Church Machine is a different answer.

I OIL AND WATER

Capabilities and data never mix.

The hard separation between code capabilities and raw data is the cornerstone of the architecture. Where conventional ISAs treat a pointer as just another integer -- and pay dearly for it -- the Church Machine makes the distinction physical. Forging a capability from raw bits is not merely forbidden by policy; it is impossible by construction. This collapses entire classes of exploit (use-after-free, type confusion, capability spray) before a single line of application code runs. The tension: the hardware tagging overhead must be justified by demonstrated performance targets, and the programmer model for interoperating with legacy data formats needs careful documentation.

II DOUBLE CHECKING

Every READ and WRITE is validated by a capability context register.

Rather than a single, monolithic memory-protection unit, the Church Machine places named capability context registers (CR14, CR12, CR5) at each architectural boundary. Every instruction fetch, every heap access, every stack operation is checked against the relevant register independently. The result is defence in depth baked into the cycle path -- not added as a software wrapper. The practical observation: the CR mapping must be stable across privilege levels; the ongoing remap work (CR0-CR15 split) is the right time to lock this down so that toolchain assumptions do not diverge from hardware.

III DISTRIBUTION NOT CENTRALISATION

No kernel, no central authority -- authority lives at the edge.

This law reads as an organisational principle as much as a technical one. By refusing a privileged kernel and distributing capabilities only as far as each abstraction genuinely requires, the architecture eliminates the single point that attackers spend most of their energy targeting. The analogy to networked systems is deliberate: a misconfigured central authority in today's cloud infrastructure can cascade into continent-scale outages. The Church Machine makes that failure mode structurally impossible. The challenge is communicating this to programmers trained on POSIX -- the contributing guidelines and quick-start narrative are the right places to address that gap.

IV DEMOCRATIC NOT DICTATORIAL

No root, no superuser -- every abstraction plays by identical rules.

Democratic is a bold word choice, and intentionally so. The boot firmware itself operates under bounded capabilities; there is no escape hatch, no manufacturer override, no God mode. This is a strong claim and one that will attract scrutiny -- the tapeout verification process and the open-source HDL are the evidence that makes it credible. The strength here is also the political message: the machine cannot be secretly neutered by a vendor after sale. For a broadsheet audience, this is the most immediately compelling of the six laws, and the one most worth foregrounding.

V CALIBRATED AND TRANSPARENT

Every capability carries explicit, inspectable permission bits and bounds.

Ambient authority -- the invisible permission granted by virtue of who you are rather than what you hold -- is the root cause of most privilege-escalation chains. Law V eliminates it: every R, W, X, L, S, E bit is visible in the capability word itself, inspectable at runtime, and bounded by an explicit range. The observation for the project: the IDE's lump metadata viewer is the natural user-facing surface for this law. Showing live capability words alongside human-readable permission summaries would make the law tangible rather than theoretical.

VI OPEN SOURCE

Hardware, toolchain, IDE, and libraries -- all inspectable and buildable.

Open Source as a constitutional law rather than a licensing footnote is the clearest statement of the project's values. No black boxes means the security model can be audited end-to-end -- from FPGA primitive to assembler to abstraction library. This is the foundation that makes the other five laws credible: without it, 'no hidden permissions' is merely a vendor promise. The practical imperative is keeping the build reproducible and the HDL readable. Follow the project, contribute, and verify for yourself at CLOOMC.org.