

Why Your Computer Cannot Be Trusted — and What We Are Building Instead

An Introduction to the CLOOMC IDE and the Church Machine

The Problem in Plain English

Every computer you use — your phone, your laptop, the server that holds your medical records, the system that controls the traffic lights — runs on an architecture designed in 1945. It was designed for a world where "computer" meant a room-sized machine operated by mathematicians in a government laboratory. It was never designed to be safe.

That is not a criticism of the designers. In 1945, the only people who could access a computer were the people standing in the room with it. Security meant a locked door. The idea that billions of untrusted users would run billions of untrusted programs on billions of connected machines was unimaginable.

But that is the world we live in now. And we are living in it on top of a foundation that was never built for it.

The result is everywhere: ransomware shuts down hospitals. Identity theft ruins families. Critical infrastructure — power grids, water treatment, air traffic control — runs on software that security researchers describe as "held together with duct tape and prayers." The global cost of cybercrime is approximately \$10.5 trillion per year, making it the third largest economy in the world, behind only the United States and China.

This is not a problem that better programmers can fix. It is not a problem that more security software can fix. It is a problem built into the foundation — the architecture itself.

Why Today's Computers Cannot Be Fixed

Imagine a city where every building is made of glass — no walls, no doors, no locks. Every room is visible and accessible from every other room. Now imagine trying to create privacy by hanging curtains. You can hang curtains everywhere, hire guards to watch the curtains, install cameras to watch the guards. But the building is still made of glass. Anyone with a sharp enough tool can cut through.

That is how computer security works today. The glass is the von Neumann architecture — a design where every piece of data in the computer's memory is, at the hardware level, accessible to every program. The curtains are the operating system, the firewalls, the antivirus, the access control lists. They are software — and software can be tricked, bypassed, or broken.

The fundamental problem is simple: today's computers do not know the difference between "data you are allowed to read" and "data you are not allowed to read." The hardware treats all memory the same. The rules about who can access what are written in software — the same software that attackers are trying to subvert. The guards are made of the same glass as the walls.

Every hack, every data breach, every ransomware attack exploits this single architectural flaw. The software says "you cannot access this." The hardware says "I do not know what you are talking about — here is the data."

No amount of better curtains will fix a building made of glass.

What If the Walls Were Real?

Now imagine a different city. This one is built with real walls — not glass, not curtains, but concrete. Each room has a real door with a real lock, and the keys are physical objects that cannot be copied, forged, or stolen by reaching through a wall. If you have the key, the door opens. If you do not, the door stays shut. Not because a guard is watching, but because the lock is built into the wall.

That is the Church Machine.

The Church Machine is a computer where the security is in the hardware, not the software. Every piece of memory is protected by a hardware-enforced "key" called a Golden Token. To read data, you need a token that says you can read. To write data, you need a token that says you can write. To run a program, you need a token that says you can execute. These checks happen in the circuitry of the chip itself — at the speed of electricity, on every single operation, with no exceptions.

There is no operating system to bypass. There is no privilege escalation to exploit. There is no buffer overflow that gives you access to someone else's memory. The hardware simply does not allow it. Asking a Church Machine to read memory without the right token is like asking a wall to stop being a wall. It is not a matter of cleverness — it is physics.

Ada Lovelace Knew This in 1843

In 1843 — 102 years before von Neumann published his architecture — Ada Lovelace wrote the world's first computer program. It was an algorithm for computing a sequence of numbers called Bernoulli numbers, written for Charles Babbage's Analytical Engine.

That program is still correct.

Not "correct for its time." Not "correct with patches." Correct — period. The mathematical reasoning Ada applied 183 years ago has not needed a single update. It has never been hacked. It has never had a security vulnerability. It has never needed to be rewritten because the platform changed.

Why? Because Ada was doing mathematics. Her algorithm is a logical proof — a statement about numbers that is either true or false, and once verified, remains true forever. The Analytical Engine she wrote for was a constrained machine with clear rules. She did not have the "freedom" to make the machine do something it was not designed for. That constraint — that loss of flexibility — is precisely what made her work permanent.

Every program running on every computer in the world today will eventually be rewritten, patched, or abandoned. Ada's program will outlast them all. Not because she was a better programmer — though she was remarkable — but because she was working with the right kind of machine.

The Church Machine is designed to be that kind of machine again.

The Lever and the Fulcrum

There is a simple way to understand why some approaches to computing work and others do not. It comes from Archimedes, who said: "Give me a lever long enough and a fulcrum on which to place it, and I shall move the world."

A lever needs three things: effort (the push), the load (the weight), and the fulcrum (the pivot point). The position of the fulcrum determines everything. Put it in the right place and a child can lift a boulder. Put it in the wrong place and a giant cannot move a pebble.

Computer science has a fulcrum: the boundary between hardware and meaning. Below that boundary, everything is physics — electrical signals, transistors, clock ticks. Above it, everything is logic — your bank balance, your medical records, the instruction that says "transfer \$1,000."

Von Neumann put the fulcrum below meaning. The hardware knows nothing about what the data means — it just moves bits around. All the meaning, all the rules, all the security is built in software above. This means every programmer, for every program, must rebuild the rules from scratch. The lever is too short. No matter how hard you push, the load barely moves.

The Church Machine puts the fulcrum at meaning. The hardware understands ownership, permissions, and boundaries — not as software suggestions, but as physical facts enforced by circuits. This means every programmer inherits the rules automatically. The lever is long. A small push moves a large load.

This is why security has not improved in fifty years despite trillions of dollars in investment: the fulcrum is in the wrong place. Moving the fulcrum — putting the security into the hardware where it belongs — changes the equation entirely.

What Is an Abstraction?

On the Church Machine, programs are not loose collections of code scattered across a hard drive. They are sealed objects called abstractions.

An abstraction is a self-contained unit of software with a defined interface — a list of operations it can perform. Think of it like an appliance. A toaster has a clear interface: put bread in, push the lever, get toast out. You do not need to understand the heating element. You cannot reach inside and rewire it while it is running. It does one thing, it does it reliably, and its internals are protected from interference.

A Church Machine abstraction works the same way. It has a fixed list of entry points — the things it can do. It has boundaries — the memory it owns, and nothing else. It has a seal — a cryptographic stamp that proves it has not been tampered with. And all of this is enforced by the hardware, not by convention.

When you build an abstraction and seal it, it is finished. Not "finished until the next update." Finished in the way that a mathematical theorem is finished — it is a verified fact that does not change. Other programs can use it through its defined interface, but they cannot corrupt it, modify it, or exploit it. The hardware will not allow it.

Why This Means Software Gets Cheaper Over Time

Here is the most important practical consequence of this approach: the cost of building software goes down over time — potentially to zero.

In conventional computing, every new program starts from scratch in terms of safety. It inherits no guarantees from the platform. A perfectly written program can be destroyed by a bug in a library it depends on, or by a vulnerability in the operating system, or by an attacker who finds a flaw in a completely unrelated program running on the same machine. The cost of each new program is the cost of writing it plus the cost of defending it against everything else. This cost never decreases. It only grows.

On the Church Machine, every sealed abstraction adds to a growing library of verified, reusable components. Because the hardware enforces isolation, a correct abstraction cannot be broken by a bug in another abstraction. When you build the thousandth abstraction, you are standing on the shoulders of

999 verified predecessors — each one guaranteed by hardware to work as specified, forever.

The global software industry currently spends approximately \$500 billion per year on maintenance — not building new things, but patching, porting, and securing old things that should have been "finished" years ago. That is the cost of the wrong fulcrum. On a machine where finished means finished, most of that cost simply does not exist.

What Is the CLOOMC IDE?

CLOOMC stands for Capability-Limited, Object-Oriented, Machine Code. The name is the design: capability-limited means every operation is bounded by a hardware-enforced token — the fulcrum that turns unconstrained flexibility into an engineering discipline. Object-oriented means the unit of software is the sealed abstraction, not the loose file. Machine code means these are real instructions executed by real hardware, not interpreted by a software layer that can be subverted.

CLOOMC is both the instruction set — the language the Church Machine hardware understands — and the fulcrum itself: the point where hardware meets meaning. The CLOOMC IDE is the development environment where abstractions are written, compiled, tested, and sealed.

The IDE runs in a web browser. It provides:

- **An abstraction creator** where you design and write abstractions in a high-level language (currently a subset of JavaScript or Haskell) and the compiler translates them into CLOOMC instructions. Each abstraction is a problem-oriented, engineered solution — not a loose script, but a sealed component with a defined interface and a calibrated MTBF.
- **A hardware simulator** that runs your abstractions exactly as the real chip would — complete with Golden Token checks, capability fault detection, and the full security model. If your code violates a security rule, you see it immediately, before it ever touches real hardware.
- **A register dashboard** that shows the state of every capability register, every data register, and every namespace entry in real time. You can watch the Golden Tokens being checked on every memory access.
- **A namespace browser** where you can see the CLOOMC namespace — the engineered library of sealed abstractions, each with its own MTBF, seal, permissions, and relationships. This is not a file system. It is a curated collection of problem-oriented solutions, each one verified and reusable.
- **A fault investigation system** that captures the complete machine state when a security violation occurs, so you can understand exactly what went wrong and why. Every fault contributes to the MTBF measurement of the abstraction that caused it — reliability is calibrated, not assumed.

The IDE targets the Efinix Ti60 F225 FPGA — a chip with 60,000 logic elements that can be physically programmed with the Church Machine architecture. It also supports the smaller Tang Nano 20K for education and IoT applications.

What This Means for You

If you are not a programmer, you might wonder why any of this matters to you. The answer is simple: every digital service you depend on — banking, healthcare, communications, transport, voting — runs on software built on the wrong foundation. That foundation cannot be patched. It must be replaced.

The Church Machine is that replacement. It is not an incremental improvement. It is a different answer to the most fundamental question in computer science: who is responsible for security — the programmer, or the machine?

For eighty years, the answer has been "the programmer." The result is \$10.5 trillion per year in cybercrime, 3.5 million unfilled cybersecurity jobs, and a world where a hospital can be shut down by ransomware because the CT scanner runs software that is two decades old and cannot be safely updated.

The Church Machine answers: "the machine." The hardware enforces security. The programmer writes logic. The two are separated by a boundary that cannot be crossed by clever code, by exploits, or by malice. That boundary is the fulcrum — and for the first time in eighty years, it is in the right place.

Where to Start

The CLOOMC IDE is available now as a web application. You do not need to install anything. You do not need to buy hardware. You can:

- 1. Open the IDE** and explore the simulator — load a program, step through it, watch the Golden Tokens in action.
- 2. Read the handbook** — a guided introduction to the architecture, written for beginners with no assumed background in hardware or security.
- 3. Try the tutorials** — hands-on exercises that walk you through writing your first abstraction, compiling it, sealing it, and running it in the simulator.
- 4. Join the project** — the Church Machine is open source. Every line of code, every document, every hardware design is public. Contributions are welcome from programmers, educators, security researchers, and anyone who believes that computing can be better than it is.

The lever is long enough. The fulcrum is in the right place. What remains is the effort — and that is something we can do together.

The Church Machine IDE is hosted at CLOOMC.org. The project is open source and published under a permissive licence. All hardware designs use the Amaranth HDL framework and target commercially available FPGA development boards.