

Design and Verification of a Digital Enigma Machine on Embedded Hardware

An embedded systems engineering project, from cipher mathematics to verified hardware

Thalyann Olivo

1. Introduction

The Enigma machine was an electro-mechanical rotor cipher used by German military forces during the Second World War. Its practical key space — the number of settings an operator could actually choose, given known rotor wiring — has been calculated at approximately 1.07×10^{23} , comparable to a 77-bit key. This proposal describes a full-stack reimplementation of that cipher logic: a reference software emulator in C++, a hand-ported embedded implementation targeting the ATtiny85 microcontroller (approximately 6 KB of flash memory), and a rigorous, independently cross-validated verification process for both.

2. Motivation and Project Framing

The original 2016 proposal framed this project as a proof of concept toward a possible future implementation in transistor-transistor logic (TTL) using VHDL. This revised proposal keeps that spirit but sharpens the framing: the project's value lies in resource-constrained embedded engineering, disciplined correctness verification, and demonstrated fluency with classical cryptanalysis — not in producing a stronger cipher.

Quantum key distribution and quantum random number generation derive their security from the physical unpredictability of quantum measurement — secure key exchange and true randomness rooted in physics itself. Enigma, by contrast, derives its (limited, historically broken) security from combinatorial key space: a classical mechanical substitution cipher whose strength depends entirely on how many settings an adversary would have to search. These are distinct technical categories addressing distinct problems. What connects them here is not the cryptography itself, but the engineering discipline underneath it: this project demonstrates the same resource-constrained embedded implementation and rigorous, independently cross-validated correctness verification that any hardware-adjacent cryptography company — quantum or classical — depends on to trust its own systems.

3. Cryptanalytic Weaknesses of the Historical Design

The historical Enigma design contains several well-documented weaknesses, summarized below. This reimplementation reproduces these faithfully, since the goal is historical and technical accuracy for verification purposes — not a hardened cipher.

Weakness	Effect
No letter encodes to itself	The reciprocal reflector pairing guarantees this, which narrows the search space for cryptanalysts.
Regular, predictable stepping	The second wheel steps once every 26 characters; the third wheel barely moves within a typical message.
Double-stepping anomaly	A mechanical quirk of the pawl-and-ratchet design lowers the effective cipher period below its theoretical maximum.
Fixed plugboard cable count	Standard procedure used exactly 10 of 13 possible cable pairs at all times, reducing the achievable key space below its theoretical maximum.

4. Implementation Plan

4.1 Stage 1 — C++ reference emulator

A software emulator implementing the full signal path: entry wheel, three rotors (selectable from historical rotors I–V), reflector (B or C), and plugboard, with ring settings and stepping — including the double-stepping anomaly — modeled precisely. This emulator is the project's source of truth; every later stage is checked against it.

4.2 Stage 2 — Known-answer test suite

Before any embedded work begins, a suite of plaintext/settings/ciphertext test vectors is generated using an independently built and cross-validated Enigma simulator, covering varied rotor orders, ring settings, and plugboard configurations, plus at least one message long enough to exercise the double-step. A dedicated stepping-only test (independent of full encryption) verifies the stepping mechanism in isolation.

4.3 Stage 3 — Embedded port

The verified logic is hand-ported to AVR C for the ATtiny85, working within its approximately 6 KB flash budget (with the ATmega328P as a fallback target if needed). Any point where the embedded implementation must diverge from the C++ reference due to memory constraints is explicitly documented.

4.4 Stage 4 — Hardware verification and demonstration

A minimal, low-cost circuit validates and demonstrates correct operation: a software UART (PB0) streams output to a USB-to-serial adapter for direct comparison against known-answer vectors; three LEDs (PB1–PB3) visualize rotor stepping, including the double-step, for observers; and a pushbutton (PB4) triggers an on-demand known-answer test run. Correctness is defined strictly as a byte-for-byte match against the Stage 2 test suite — not by visual inspection of the demo alone.

5. Verification Methodology

Two independent checks anchor the verification process. First, known-answer testing: every stage of the implementation (software and hardware) is checked against test vectors generated by a separately built, independently cross-validated simulator, so no single source of truth is trusted twice over. Second, Enigma's self-reciprocal property provides a low-cost internal consistency check — running a ciphertext back through the same settings should recover the original plaintext on the same machine, live.

6. Proposed Timeline

At approximately 25 hours per week, the following phased plan is proposed:

Phase	Deliverable
Weeks 1–2	C++ reference emulator: rotor wiring, reflector, plugboard, stepping logic (including double-step).
Weeks 3–4	Known-answer test suite generated and cross-validated against an independent simulator; full test coverage of the C++ emulator.
Weeks 5–7	AVR port to ATtiny85 (or ATmega328P fallback); flash-budget-driven design decisions documented.
Weeks 8–9	Hardware verification build: serial output, rotor-step LEDs, test-vector button; known-answer tests re-run on physical hardware.

7. Relevance to Data Science and Security Engineering Work

Beyond its value as a standalone embedded systems project, this build reflects a set of skills directly relevant to data science and engineering roles at companies working in cryptography, security hardware, and related fields. A few examples of where those skills transfer:

- Randomness and entropy validation: hardware random number generators require continuous statistical testing (e.g. the NIST SP 800-22 test suite) and entropy estimation — core to product credibility, and squarely a data science discipline rather than general software engineering.
- Anomaly and side-channel detection: statistical or machine-learning-based detection of hardware drift, environmental interference, or side-channel leakage in security-critical hardware.
- Usage and performance analytics: modeling adoption, usage patterns, and product performance data as a hardware or security product scales.
- Internal tooling: data pipelines, evaluation frameworks, and dashboards that let a small engineering team move faster without each function rebuilding its own.

Technical foundation

This project draws on a technical foundation that includes coursework in electrical and computer engineering at NYU Polytechnic and an A.S. in Computer Science from Kingsborough Community College, along with daily, highly iterative use of modern AI development tools. The project itself — its staged verification methodology, its independent cross-validation approach, and its embedded implementation plan — is representative of how I approach technical work generally: rigorously, with correctness treated as a first-class requirement rather than an afterthought.

8. Closing

This project is offered as a demonstration of engineering capability — the same resource-constrained implementation and rigorous, independently cross-validated verification discipline that security-critical hardware work depends on, whether classical or quantum.