Declaration of Authorship

I, Antony Johan Karels, hereby declare that I am the sole author of the work entitled:

“*From Concept to Console: FortressFlash, a Secure Field Deployment Tool for FortiGate and Beyond*”

here enclosed, and that I have compiled it in my own words, that I have not used any other than the cited sources and aids, and that all parts of this work, which I have adopted from other sources, are acknowledged and designated as such. I also confirm that this work has not been submitted previously or elsewhere.

Signed in Stillwater, Minnesota, United States, on 25 August 2025.

[illegible]

Antony Johan Karels

Integrity Note:

For verification purposes, the SHA-256 hash of the officially submitted PDF is archived at:

https://cyberus.lu/fortressflash/karels_fortressflash_thesis_2025_sha256sum.txt

https://gist.github.com/aku762/0217ada7fad11812173f52e991dc14e9

UNIVERSITY OF LUXEMBOURG

FACULTY OF SCIENCE, TECHNOLOGY AND MEDICINE

ERASMUS MUNDUS JOINT MASTER IN CYBERSECURITY (CYBERUS)

Master's Thesis

From Concept to Console: FortressFlash, A Secure Field Deployment Tool for FortiGate and Beyond

AUTHOR:	Antony Johan Karels
STUDENT NO:	0232759843
PROGRAM SUPERVISOR:	Assoc. Prof. Tegawendé Bissyandé
ACADEMIC SUPERVISOR:	Assoc. Prof. Christian Fisch

SUBMITTED: **25 August 2025**

PAGES: 74 WORDS: 20,579 CHARACTERS: 139,848

Acknowledgements

This thesis marks the culmination of my journey through the CYBERUS Erasmus Mundus Joint Master's Degree program, and I am deeply grateful to those who supported me along the way.

To my fellow **2023-2025 CYBERUS** cohort — thank you for the shared experiences, encouragement, and solidarity throughout this international academic adventure. From Lorient to Tallinn to Luxembourg and beyond, your presence made the challenges lighter and the memories brighter

I extend my sincere thanks to **Professor Guy Gogniat** of the University of Southern Brittany for his support during the first year of the program, and to **Associate Professor Tegawendé F. Bissyandé** at the University of Luxembourg for his flexibility and trust in allowing me to pursue this thesis independently. I am also grateful to my academic supervisor, **Associate Professor Christian Fisch**, for his guidance and encouragement throughout this project.

The initial spark for this project came from hands-on experience in a previous workplace, where I was fortunate to have colleagues who supported early exploration. In particular, I would like to thank **Beth Taylor**, whose encouragement to pursue the CYBERUS program gave me the confidence to take that opportunity despite the uncertainty of a new position. In the months that followed, she continued to support the idea of this project by helping to procure initial hardware and allowing time for some of the early development work, contributions that ultimately shaped the direction of this thesis.

I also wish to acknowledge the use of OpenAI's ChatGPT as a writing and brainstorming companion during many solitary sessions. Its assistance in refining drafts, exploring phrasing, and maintaining momentum was a valuable aid in bringing this work to completion.

The direction, design, and execution of this project have been entirely my own, yet they were built on a foundation shaped by many others. I am thankful for every challenge that helped forge this outcome, for every opportunity that made it possible, and for the set-apart grace that carried me through to its completion. To all who offered guidance, support, or simply a word of encouragement along the way: **Thank You.**

Abstract

Managed-service providers often ask Level-1 technicians to stage FortiGate firewalls under tight change windows. In practice, first-boot turn-up is slowed by repeated laptop tethering, reconfiguring the admin PC onto the factory management subnet, and other setup friction before a vetted baseline can be applied. FortressFlash addresses this usability gap with a self-contained, Raspberry Pi Pico–based appliance that lets L1 staff deliver a *pre-vetted, version-controlled* baseline configuration over UART—without a laptop—while embedding hardware-rooted security. The current prototype couples a Waveshare RP2040-Plus, capacitive-touch display, and microSD storage with modular MicroPython firmware. Security primitives—planned for migration to Raspberry Pi Pico 2 W—include signed/attested boot, OTP-sealed keys, and SHA-256 integrity checks.

A pragmatic three-phase methodology guided development: rapid hardware familiarization, iterative firmware prototyping, and STRIDE-aligned threat modeling. Bench trials on FortiGate 60E units validate stable console-session establishment (20/20 successful logins without resets), authenticated CLI interaction, SD-backed profile selection, and live scroll logging. At submission time, integration of the end-to-end *automated* first-boot push is not yet complete; the evaluation therefore reports component-level behavior and documents the dispatcher design intended to enable fast, repeatable deployment once integrated.

Threat analysis motivates PIN-gated access, file-level encryption of removable media, console-only service accounts, and optional device–firewall binding to mitigate spoofing, tampering, and credential-exposure risks. A near-term roadmap details configuration signing, attested boot/firmware, and role-based access. Beyond its academic contribution, FortressFlash's portable, zero-laptop workflow and planned cryptographic safeguards are designed with regulatory frameworks such as NIST 800-171/CMMC in mind. The underlying architecture remains vendor-agnostic, and its UART-driven approach can be extended to other CLI-centric platforms that share similar staging challenges.

In sum, this thesis documents the design, partial implementation, and security analysis of a field-oriented tool intended to turn error-prone, laptop-dependent staging into a repeatable, auditable process—while charting a clear path to multi-vendor expansion.

Keywords: Fortinet, FortiGate, firewall configuration, CLI automation, Raspberry Pi Pico, UART, MicroPython, STRIDE threat modeling, secure boot, CMMC, embedded systems security, managed service providers (MSP)

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background and Motivation

Initial Experience in FortiGate Deployment. I was first introduced to the FortiGate platform in spring 2023 while working as a project engineer for a managed service provider (MSP) in Minnesota that offered IT support and cybersecurity services to small and medium-sized enterprises (SMEs). As a Fortinet partner, the company standardized on FortiGate appliances across all client networks. My role involved deploying new devices during customer onboarding by following internal documentation to apply a base configuration and adapt it to each client's network. After gaining confidence with features like intrusion detection (IDS), DNS filtering, and SSL VPNs, I was assigned a more advanced setup: configuring SSL VPN access via FortiClient with Azure Entra ID authentication and multi-factor login using Microsoft Authenticator.

Most configurations were handled through the GUI, but key steps in Microsoft and Fortinet documentation often required command-line interface (CLI) commands—some buried deep in the interface, others unavailable outside the terminal. With so many features and options, the CLI often proved faster and more precise, especially for tasks like setting SAML URLs and group mappings. While I initially preferred the GUI as I learned the platform, I quickly saw that mastering the CLI would boost both efficiency and understanding. That became clear when I learned how to transfer configs between devices by copying CLI blocks from one file into another and loading them directly.

FortiLab and Multi-VLAN Testing. To sharpen my FortiGate skills—and give our help-desk a safe playground—I scavenged a decommissioned Netgear GS724TP switch and three spare FortiGate appliances to build "FortiLab" (Figure 1.1). Each firewall sat on its own tagged VLAN, with WAN links back to the office network, letting us flash different firmware versions and hardware models in complete isolation.[1]

The lab's sandboxed topology meant we could rehearse upgrade paths and experiment with policy sets without risking production. Documenting the wiring, VLAN plan, and backup configurations in the company's configuration management database (CMDB)—and labeling every port—sharpened my network-design discipline and left a reusable test bed the team still relies on today.

[1] Admittedly, the VLANs weren't strictly necessary—each FortiGate had spare ports. But the extra complexity made things more interesting, and gave me an excuse to laminate a port mapping table.

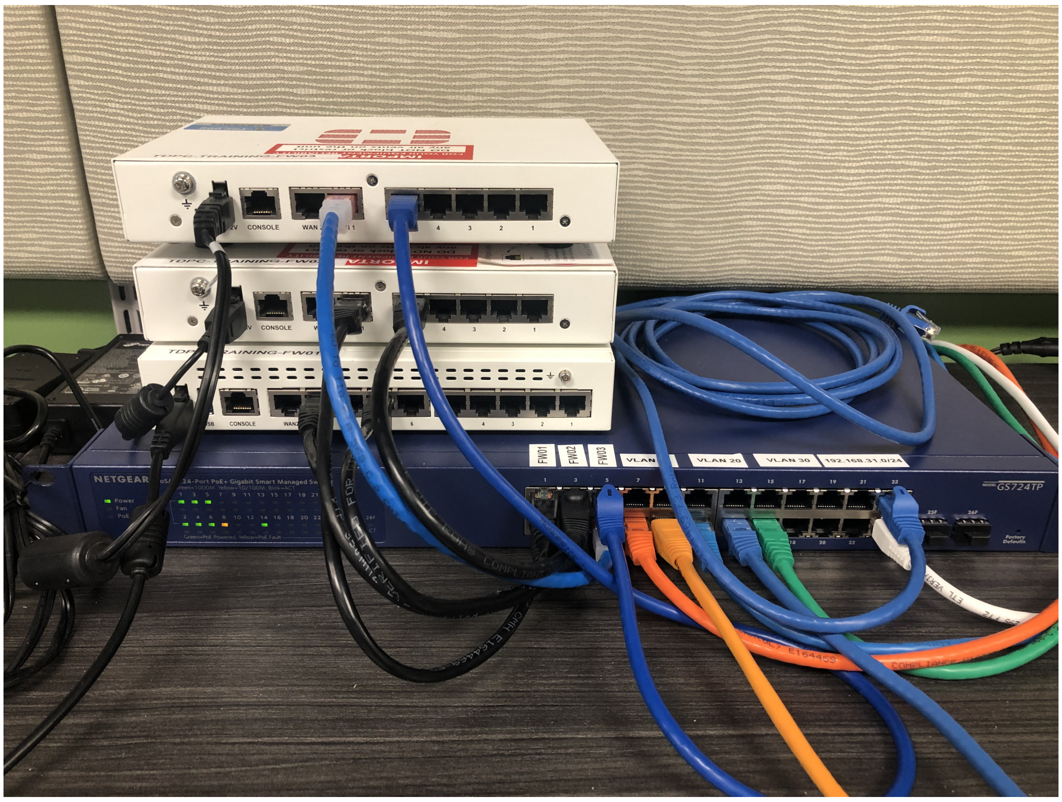


Figure 1.1: "FortiLab" — three stacked FortiGate firewalls linked to a Netgear GS724TP switch, each on its own VLAN for safe experimentation.

ICS Pentesting Lab. During the second semester of the Cyberus program, while studying in Lorient, France, I participated in a group project focused on industrial control system (ICS) security. Our task was to demonstrate practical pentesting techniques within a simulated operational technology (OT) environment. I led the lab design and implementation as part of a group assignment, documented in our internal ICS/OT pentesting report.[2] Naturally, I incorporated the FortiGate firewall that my employer had provided for hands-on practice.

The lab environment combined a Raspberry Pi 2 B+ running OpenPLC, a breadboard circuit simulating digital I/O, and the FortiGate 60E acting as a network gateway and security perimeter. The OpenPLC was programmed with ladder logic to control LEDs and receive inputs from physical buttons, mimicking basic industrial processes. This hands-on project deepened my understanding of microcontroller integration in OT networks and re-ignited my interest in small-form-factor hardware (Figure 1.2).

Deploying FortiGate in FIPS-CC Mode. Following the ICS/OT pentesting experience, I returned to Minnesota for a summer internship that focused on helping a client prepare for the Level 2 Cybersecurity Maturity Model Certification (CMMC). Although the project primarily addressed

[2]Available at https://cyberus.lu/reports/Karels_ICS-Pentest_Cyberus_2024.pdf. See also [1].

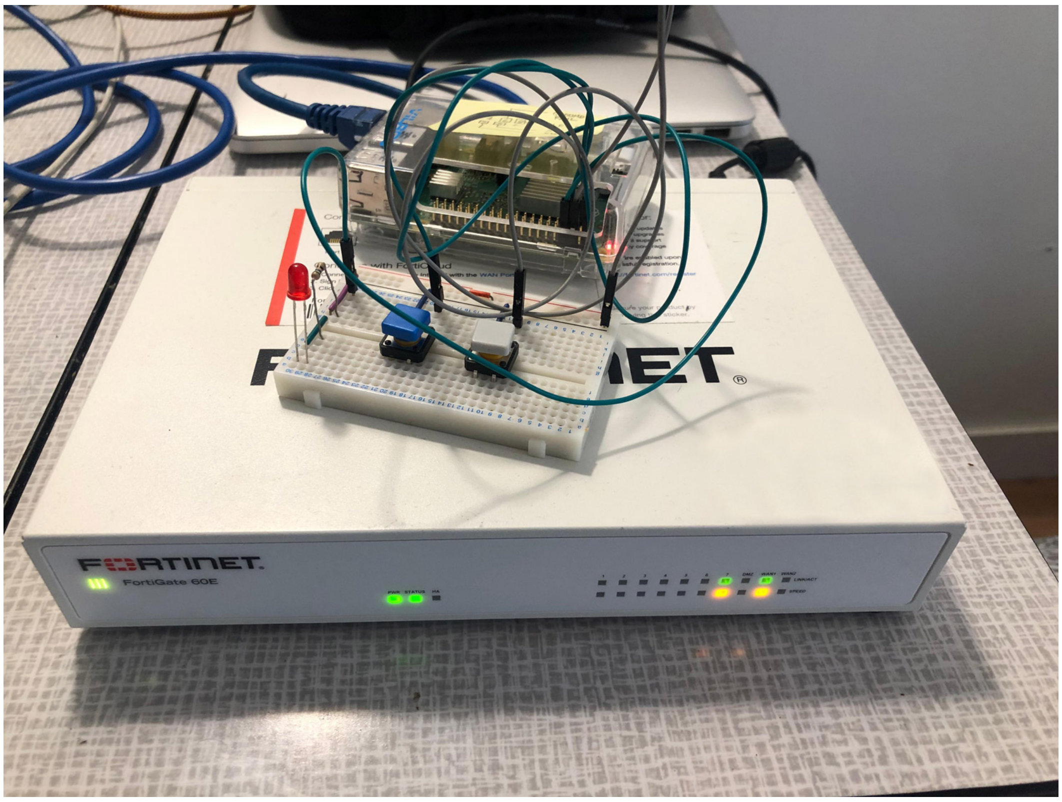


Figure 1.2: ICS/OT pentesting lab setup, featuring a Raspberry Pi 2 B+ running OpenPLC, breadboard with I/O circuit, and a FortiGate firewall for segmentation and perimeter security.

broader compliance topics, my final assignment required directly working with FortiGate firewalls in a CLI-only environment. The task involved cloning a configuration from a standard device onto a new unit running the FIPS-CC validated firmware, which enforces command-line usage by design and disables the graphical interface during writing of the new firmware.

This hands-on experience not only deepened my understanding of FortiGate's CLI structure and constraints, but also highlighted the operational challenges faced by small teams when deploying secure configurations under regulatory frameworks like CMMC. The concept for a portable CLI automation device began to take shape during this phase, inspired by the desire to streamline such deployments in resource-constrained environments.

During this period I also completed both the Fortinet Certified Fundamentals (FCF) and Fortinet Certified Associate (FCA) certifications, which cover core principles of network security and foundational skills for configuring and managing FortiGate devices.[3] A copy of the earned certificates, along with configuration tasks, the original proposal, and additional project notes, are included in the final internship report.[4]

[3]Completing both certifications qualified me to receive a FortiGate 70F appliance bundled with a one-year subscription to Fortinet's security services (e.g., FortiGuard updates, support, and licensing) through their promotional program — a generous offer that, despite my enthusiastic compliance, mysteriously never materialized.

[4]Available at `https://cyberus.lu/reports/Karels_CMMC-Internship_Cyberus_2024.pdf`. See also [2].

Automating FortiGate Setup over Serial. While working on CMMC-related tasks during my internship, I simultaneously began experimenting with serial communication to the FortiGate using a USB console cable. What began as a simple Python script—logging into the CLI, issuing basic commands, and printing the output—quickly evolved into more advanced automation. I developed additional scripts to enable or disable GUI access via HTTPS on the WAN port and to modify LAN/WAN IP settings. These small utilities made initial configuration easier by avoiding the need to manually reconfigure a workstation's NIC to match the firewall's default IP range.

As noted earlier, deploying new FortiGate devices involved following a standardized base configuration. While this responsibility usually fell to Project Technicians, there were frequent situations where Helpdesk staff could have benefited from the ability to perform the task themselves. Unfortunately, due to limited experience with the devices, this responsibility was rarely delegated—despite the organization's need to more efficiently distribute onboarding tasks.

With this challenge in mind, I aimed to leverage the serial connection to automate deployment of the base configuration using Python. The idea was simple: Helpdesk staff could unbox a new device, plug in the console cable, run a script, and have the base configuration applied automatically. Any unique values—such as licensing credentials or client-specific settings—could then be verified and adjusted via the GUI. I also introduced interactive prompts to collect dynamic values like the hostname and IP addresses, with the goal of gradually expanding this capability to support more flexible and parameterized deployments.

Exploring MCU Options for Serial Control. Following my experiments using Python to interface with the FortiGate over a console cable—and still inspired by the ICS OpenPLC demonstration—I began exploring a variety of microcontroller unit (MCU) platforms. Initially revisiting the ICS lab, I rebuilt the breadboard created in the University's lab using personally owned components. I then transitioned to a TinkerKit LCD unit based on the Arduino Leonardo, with the basic goal of re-familiarizing myself with the hardware and the Arduino C environment. Early experiments included output to LCD screens, reading sensor data, button input, and controlling LEDs. These quickly evolved to include Ethernet shields and the development of simple web servers capable of reading and writing data over TCP/IP to interact with connected peripherals.[5]

Building on my initial success communicating with the FortiGate using Python and a console cable, I wanted to explore whether similar functionality could be achieved using a microcontroller. With several platforms already in possession—and an eagerness to expand my options—I acquired additional development boards to evaluate their suitability for the task (Figure 1.3; see also Section 2.2,

[5]Examples included reading temperature sensors, toggling LEDs, and triggering a PowerShell script (source code at https://github.com/aku762/listenlock) to remotely lock my workstation via a network-connected button.

Candidate MCU Platforms and Selection for a detailed comparison).[6]

The result of this exploration culminates in this project's deliverable: a purpose-built prototype initially nicknamed the *FortiProgrammer*[7] during early testing, but ultimately branded as **FortressFlash** to establish its product identity. The device is introduced briefly below and described in full detail in Chapter 2, Hardware Design.

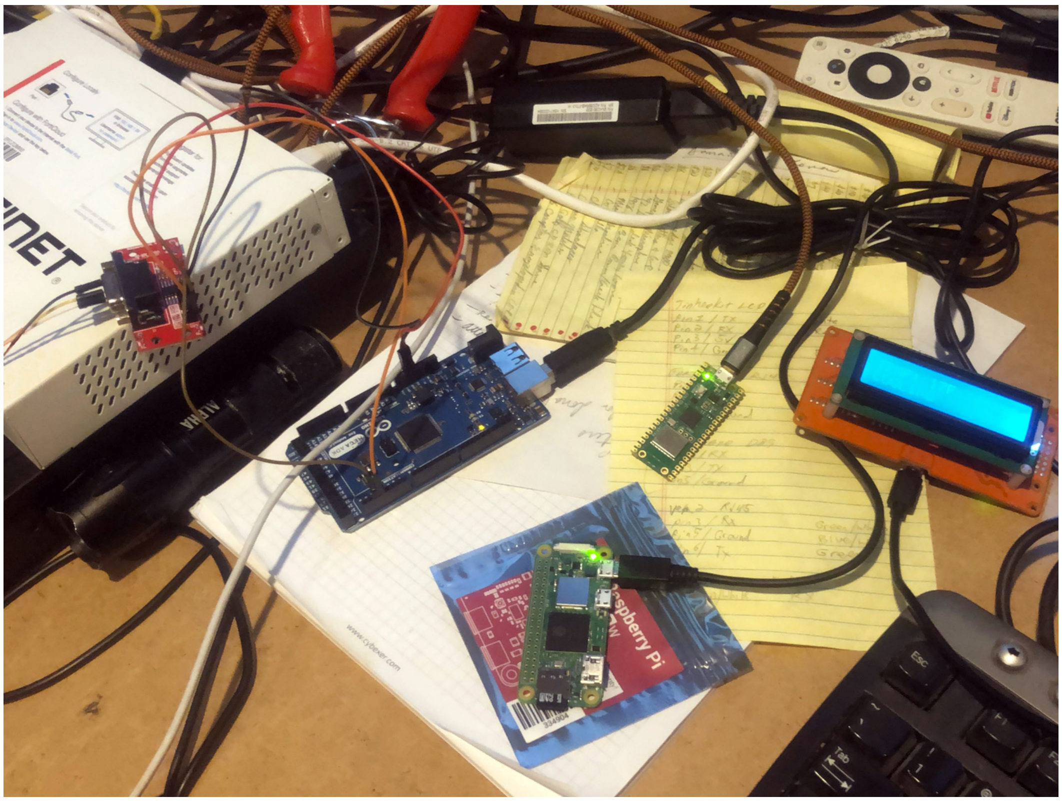


Figure 1.3: Early prototyping chaos: various MCUs and serial experiments in progress. From left to right: Arduino Mega, Raspberry Pi 2 B+, Raspberry Pi Pico W, and a TinkerKit LCD module based on the Arduino Leonardo architecture.

Entrepreneurial Case Study Influence. During the third semester of the Cyberus program, while attending in-person courses in Luxembourg, our cohort participated in an entrepreneurship course taught by Prof. Christian Fisch. One of the assigned case studies[8] focused on Fortinet and its co-founder and CEO, Ken Xie [3]. The study explored Xie's decision to resist market pressure from analysts encouraging a shift toward cloud-first messaging and splashy marketing. Instead, he remained committed to a hardware-centric strategy, emphasizing Fortinet's long-term investment in ASIC development and product integration. The hardware-first strategy described in the case

[6]I may have immediately gone to MicroCenter and impulse-bought half the microcontroller aisle.

[7]*FortiProgrammer* was an informal name used during early prototyping. It is not affiliated with or endorsed by Fortinet, Inc. Normally I'd worry about trademark issues, but given their usual lack of response, I felt pretty safe. The final product name **FortressFlash** is discussed in Chapter 6, Prototype to Product, as part of the branding process.

[8]Fortinet Leadership Case Study, shared in Cyberus Entrepreneurship Course (Fall 2024).

paralleled the *FortiProgrammer* prototype's architecture, which prioritized direct serial access and physical control over configuration workflows. This perspective reinforced Fortinet's relevance—not only as the technology I had been working with extensively, but also as a model of focused, long-term design philosophy.

Project Realignment and Scope Refinement. The original internship, scheduled for my return to Minnesota, was supposed to revolve around developing a custom security-data dashboard for my former employer. However, a company merger soon consumed the very managers who had committed to mentor the project. With no supervisor available and key decisions—such as the choice of monitoring tools—still frozen by re-organization, the dashboard concept stalled.

Rather than wait for corporate alignment, I redirected my thesis focus to a topic that had been a recurring theme over the past two years: efficient FortiGate CLI administration. In consultation with Prof. Tegawendé Bissyandé, I presented the *FortiProgrammer* prototype—an embedded, self-contained tool for pushing standardized configurations over the FortiGate console port—as a new research focus. The faculty approved this pivot, and because I already owned the necessary hardware, I stepped-away from the organization and continued the thesis as an independent, self-directed effort.

Early drafts of the proposal had included an ambitious natural-language–to-command-line (NLP→CLI) module, but expanding the scope to full language processing would have derailed the already tight development timeline. Consequently, NLP has been reframed as future work (see Section 7.6, Intent-Based Networking): the present thesis concentrates on the practical design, implementation, and evaluation of a secure hardware–software stack, while reserving NLP-based command generation for future exploration.

1.2 Problem Statement

Across the small-to-medium-enterprise (SME) sector, firewalls are rarely deployed by dedicated network engineers. Managed service providers (MSPs) must bring up dozens of client sites, each with its own topology, change window, and compliance rules. The business pressure is simple: *onboard the new box tonight; billable traffic must flow tomorrow at 08:00.* In that setting, any mis-typed CLI flag or missed firmware step is not just an inconvenience—it is downtime, SLA penalties, and reputational damage.

FortiGate appliances are popular precisely because their GUI lowers the entry barrier for routine tasks, yet the platform's most powerful features—custom SAML endpoints, FIPS-CC flash images, granular policy objects—still surface only through the command-line interface (CLI). Those commands are peculiar enough that even senior technicians keep a cheat sheet on a second monitor.

Worse, first-boot workflows occur *out-of-band* on the serial console, making them invisible to remote-management platforms that MSPs rely on for audit trails and script automation.

Operational pain points

1. **Error-prone:** A single syntax typo can drop VPN tunnels or expose default services.
2. **Inconsistent:** Step fatigue produces configuration drift, complicating long-term maintenance.
3. **Opaque:** Serial sessions leave no logs in remote monitoring and management (RMM) tools, hampering compliance under frameworks such as CMMC.

Security compliance gap – NIST 800-171 controls enforced via CMMC now require MSPs to prove that cryptographic modules (e.g., FIPS-CC firmware) load correctly, that default passwords are rotated, and that no credentials linger on removable media. A throw-away serial script on a help-desk laptop does not satisfy those controls; a signed-boot microcontroller with one-time-programmable (OTP) stored keys can.

Research challenge. Can a *portable, microcontroller-driven appliance* inject a cryptographically verified baseline configuration over the console port—*quickly, repeatably, securely*—while remaining usable by junior technicians?

1.3 Objectives and Scope

Primary Objectives. This thesis documents the design and evaluation of a self-contained, microcontroller-based tool that prepares FortiGate devices for first use over the serial console. Concretely, the work aims to:

- Design and assemble a working hardware prototype (Waveshare RP2040-Plus + ResTouch LCD + MAX3232 + microSD) capable of stable UART console I/O with FortiGate devices.[9]
- Implement a minimal, field-oriented firmware stack in MicroPython that provides: (i) on-device UI for profile selection, (ii) SD-card profile discovery, (iii) a console login routine with bounded retry logic, and (iv) a serial dispatch pipeline in progress for configuration lines with prompt-aware pacing.
- Define configuration-bundle conventions (file organization, placeholders, guardrails) suitable for lab deployments and future signing.
- Produce a threat-model–driven security plan for the device (tamper, credential handling, roll-back), mapping mitigations to Pico 2 W hardware primitives for a later migration.

[9]See Chapter 2, Hardware Design.

Secondary/Translational Objective. In addition to the technical aims, the thesis documents a strategic innovation analysis (Chapter 6, Prototype to Product). This includes a visualized Business Model Canvas, pricing hypotheses, and go-to-market framing. The purpose is to contextualize the prototype in terms of value creation for managed service providers and to reflect the program's entrepreneurship emphasis. While this analysis shaped priorities such as secure boot, credential handling, and UX polish, it does not alter the primary technical success criteria of the prototype.

Success Criteria. The prototype is considered successful for the thesis if it:

- boots to a functional UI, mounts SD, lists profiles, and exchanges CLI I/O with a FortiGate over UART in a lab setting;
- demonstrates an end-to-end configuration push *or* a verified, prompt-paced dispatch pipeline with logging ready for end-to-end trials;
- reports quantitative reliability or partial timing data when end-to-end push is not yet feasible, with the $< 120\,\mathrm{s}$ target applied in future integration trials.

Scope Delimitations.

- **Device/OS:** FortiOS 6.x–7.x on models with RJ-45 console ports.
- **Config lifecycle:** Initial push and sanity verification only; continuous configuration management is out of scope.
- **Deferred features:** Multi-vendor support, Wi-Fi/BLE, TCP/IP API mode, and NLP/LLM intent translation are documented as *Future Work* (Chapter 7, Future Work), not deliverables for submission.
- **Exploratory work captured:** A Strategic Innovation track (branding, Business Model Canvas, pricing hypotheses, go-to-market framing; see Chapter 6, Prototype to Product) and limited LoRa tests (Section 7.4) are included as documented explorations that informed design but are not core technical objectives of this thesis.

1.4 Methodological Approach

The project followed a pragmatic, iterative process—build, test, refine—prioritizing a continuously working prototype over speculative architecture. Tasks were managed on a physical whiteboard and grouped into must/should/nice-to-have buckets. The resulting phases, along with their key activities and current status at submission, are summarized in Table 1.1.

Phase 1 — Requirements & Scoping. Define the “minimum viable deployer,” success metrics (e.g., $< 120\,\mathrm{s}$ time-to-first-push), assumptions, and constraints for a field-usable tool. Capture initial risks with a STRIDE pass and explicitly delimit out-of-scope features (multi-vendor, TCP/IP API mode, NLP) for later exploration.

Phase 2 — Hardware Selection & Assembly. Select and bench-assemble the RP2040-based stack (Waveshare RP2040-Plus, MAX3232, 2.8” LCD/touch, microSD). Validate RS-232↔UART signaling to the FortiGate console, confirm stable power and cabling, and bring up display/touch and storage to ensure a reliable platform for firmware iteration.

Phase 3 — Firmware Development. Establish a rapid MicroPython loop (REPL + on-device execution). Implement core subsystems: stable console login with bounded retry logic, scrolling serial log, main menu touch UI, and SD profile discovery; a prompt-paced dispatch pipeline for CLI blocks remains in progress. Evolve UX from whiteboard flows to Figma frames and a linked navigation diagram; maintain small, testable commits with bench validation on a FortiGate 60E.

Phase 4 — Laboratory Evaluation. Exercise subsystems on hardware to verify serial I/O, UI responsiveness, SD handling, and console login reliability; collect quantitative success rates and buffer traces; observe error cases (prompt mismatches, CR/LF quirks, buffer limits). Schedule full end-to-end timing once the dispatch path stabilizes; retain logs/screens for reproducibility.

Phase 5 — Security Hardening. Deepen STRIDE analysis and define mitigations. Where timeline limits implementation, map controls to Pico 2 W primitives (signed boot, OTP-backed secrets, SHA-256, TRNG) and specify near-term measures (PIN-derived keys, bundle signing format, log redaction) as integration points for the next sprint.

Parallel Explorations (Non-core). Two investigations informed design but remain outside the core submission deliverables: (i) a Strategic Innovation track (branding, Business Model Canvas, pricing tiers, go-to-market framing) aligned with program outcomes, and (ii) preliminary LoRa tests to evaluate a future out-of-band control/receipt channel. These are documented to capture the full scope of inquiry without altering the technical success criteria.

Table 1.1: Project phases and current status

Phase & Goals	**Key Activities**	**Status**
1. Requirements & Scoping: Define minimum viable deployer; set $< 120\,\mathrm{s}$ metric; enumerate STRIDE threats	Draft objectives/scope; establish timing/error criteria; capture assumptions/constraints	Complete
2. Hardware Selection & Assembly: Stable console I/O with peripherals	Select RP2040-Plus; integrate MAX3232, LCD/touch, microSD; bench-verify UART	Complete
3. Firmware Development: Field-usable prototype UI + I/O	Implement console login routine with retry logic; scrolling log (done); main menu UI (done); SD profile enumeration (done); dispatcher for CLI blocks (in progress)	Core subsystems complete (login, log, menu, SD); dispatch in progress
4. Laboratory Evaluation: Validate subsystems; collect reliability metrics	Verify serial I/O, UI, SD handling; validate console login across 20 trials; document buffer traces and error cases; schedule end-to-end push timing once dispatch stabilizes	Console login validated (20/20); partial evaluation complete; full push pending
5. Security Hardening: Apply STRIDE; define Pico 2 W integration points	Plan PIN-derived keys, file-level encryption, signed bundles, encrypted logs; map to Pico 2 W secure boot/OTP/SHA-256	Planned
	— Non-core investigations —	
Parallel Explorations: Business/LoRa inquiries	Strategic Innovation track (branding, BMC, pricing, GTM); preliminary LoRa range/control tests for future OOB channel	Documented; not part of core deliverables

Chapter 2: Hardware Design

2.1 Goals and Constraints

FortressFlash is a hand-held, field-deployable tool whose primary job is to push vetted baseline configurations to FortiGate appliances over the console port, reliably and repeatably, in the hands of a level-1 technician. The hardware must be simple enough to operate under time pressure in a wiring closet, yet robust and extensible enough to support future security features (e.g., signed bundles, OTP, PIN-gated access, and secure boot). This section states the goals and constraints that shaped the design; the remainder of the chapter shows how each decision traces back to these requirements.

Design goals.

- **Deterministic serial provisioning.** Establish a clean UART↔RS-232 path that negotiates a console session quickly and completes a baseline push without human timing tricks.
- **Field usability.** Hand-held form factor; daylight-readable 2.8 in touchscreen; single-purpose UI with clear status and error states; operable from a USB power bank or internal LiPo battery.
- **Profile portability.** Load signed configuration bundles from microSD; support multiple customer/device profiles without reflashing firmware.
- **Security readiness.** Minimize attack surface (console-only path); enable PIN lock and bundle signing now; keep an upgrade path for advanced security features (signed boot, OTP, SHA-256, TRNG) without re-architecting the board.
- **Extensibility.** Use standard buses (SPI for LCD/touch and SD; UART for console) and modular drivers so features like audit logging, BLE/Wi-Fi convenience links, or vendor-agnostic serial templates can be added later.
- **Serviceability and manufacturability.** Prefer commercial off-the-shelf (COTS) modules, straightforward wiring, and a printable enclosure; document cutouts and fasteners so another engineer can reproduce the unit.

Key constraints.

- **Cost and time.** Prototype BOM and build time must remain modest to meet the thesis schedule and demonstrate repeatability (no custom PCB required).
- **Power.** Operates from standard 5 V USB sources (FortiGate, wall adapter, or portable/internal battery pack). Current draw is modest enough to run for many hours from a small power bank. Detailed electrical characterization is outside the scope of this thesis.
- **Performance headroom.** MicroPython execution and I/O throughput must keep the UI responsive while sustaining console transfers; performance measurements guide (not dictate) platform choices.
- **Physical I/O.** Honor standard DE-9[1] console wiring with a documented null-modem crossover; provide mechanical strain relief and basic ESD protection at the connector.
- **Reliability.** Operate without thermal hotspots or counterfeit-IC fragility; fail safely if storage is missing/corrupt or if a session is interrupted.
- **Scope limits.** No Ethernet, TCP/IP, or cloud control in this chapter; those appear in Future Work and require additional networking and API surfaces.

System overview. At a high level (see block diagram in Figure 2.1), USB-C power and an optional LiPo battery (6,7) supply the RP2040-Plus MCU module (2). The MCU drives an SPI[2] bus to the Waveshare Pico-ResTouch carrier (1), which integrates the 2.8 in LCD, XPT2046 touch controller, and the removable microSD socket; the microSD card (5) holds configuration bundles and profile data. The MCU also exposes a UART that interfaces to a MAX3232 RS-232 transceiver breakout (3), which provides a DE-9 connection; a male–male null-modem adapter (4) and the standard DE-9 console cable (8) implement the required crossover wiring to the FortiGate console port (9). This partition keeps the ±RS-232 domain electrically isolated from the MCU's 3.3 V logic and makes each subsystem independently verifiable.

The following sections walk through the numbered subsystems: the MCU (2) in Section 2.2; display and storage (1,5) in Section 2.3 (Waveshare ResTouch 2.8" — overview) ; power (6,7) in Section 2.3 (Power and battery); the serial interface and cabling (3,4,8,9) in Section 2.4; and enclosure and assembly notes in Section 2.5.

[1]Formally the nine-pin D-sub shell size is DE-9; "DB-9" is a widely used colloquialism inherited from the common DB-25 connector label.

[2]SPI (Serial Peripheral Interface) is a synchronous serial bus used for short-distance, high-speed connections between a master (MCU) and one or more slaves (displays, SD cards, sensors). Typical lines: `SCLK`, `MOSI`, `MISO`, `CS`.

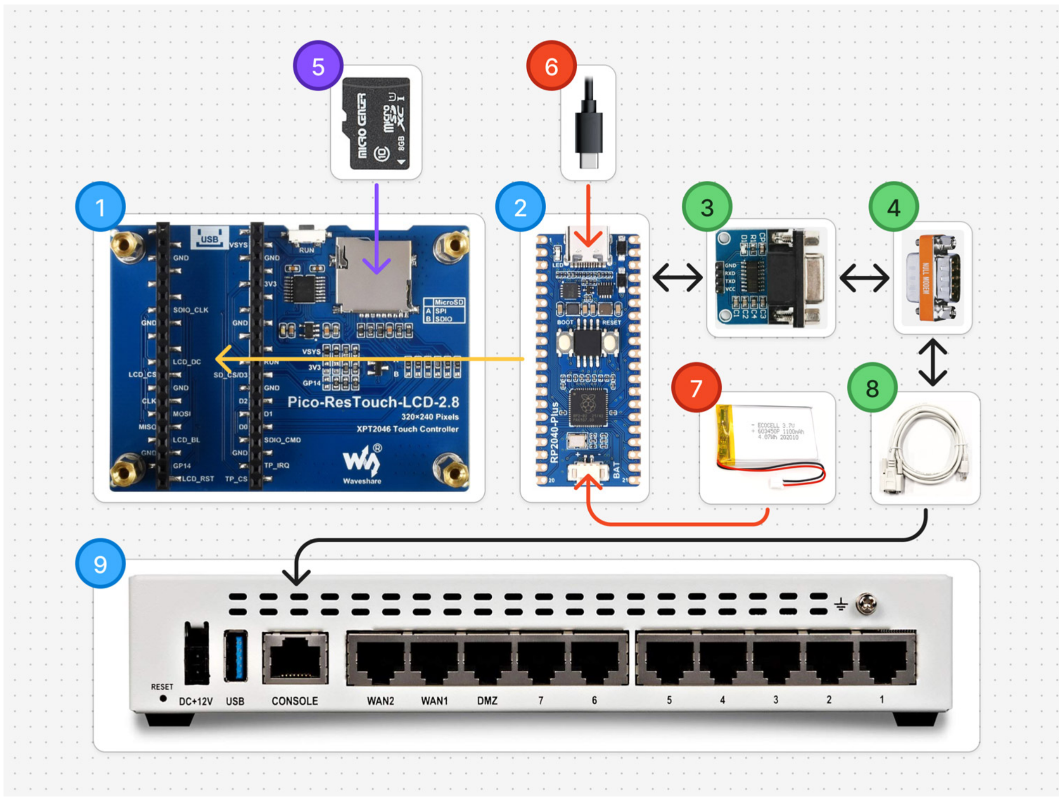


Figure 2.1: FortressFlash hardware block diagram, showing major subsystems: (1) 2.8 in LCD + touch controller + microSD carrier (Waveshare Pico-ResTouch), (2) RP2040-Plus MCU module, (3) MAX3232 RS-232↔TTY breakout, (4) DE-9 male↔male null-modem adapter, (5) microSD card for configuration bundles, (6) USB-C port for power and programming, (7) optional LiPo battery, (8) standard DE-9 console cable, and (9) FortiGate firewall console port. Data flows from the MCU to the FortiGate console via UART→RS-232 conversion, while SPI buses connect the LCD, touch controller, and SD storage. Power can be provided by USB-C or battery.

2.2 Candidate MCU Platforms and Selection

Several development platforms (see Figure 1.3) were evaluated for suitability with the FortressFlash prototype, including the TinkerKit LCD, Arduino Mega, the Raspberry Pi Zero 2 W, the Raspberry Pi Pico (RP2040 family), and the RP2040-Plus module that was chosen for the prototype (Table 2.1). The core selection criteria emphasized reliable UART and SPI I/O, a compact handheld form factor, and a modest BOM cost to enable rapid, reproducible prototyping. During the hands-on evaluation phase, the RP2040's MicroPython ecosystem proved invaluable for rapid firmware development, and the later appearance of the Pico 2 W indicated a plausible upgrade path to hardware security primitives—both benefits that reinforced the RP2040-based choice. The block diagram in Figure 2.1 shows the final architecture; the remainder of this section explains why the RP2040-Plus best balanced the stated constraints and how that choice shaped downstream subsystem design.

TinkerKit LCD (Arduino Leonardo). Early testing was performed using the TinkerKit LCD module, which includes a built-in 16×2 LCD and servo headers for quick GPIO prototyping [4]. This legacy board, based on the Arduino Leonardo, was convenient to experiment with because several units were available from prior stock—despite it being discontinued and no longer supported by TinkerKit [5]. However, its single UART interface quickly proved insufficient: the board could not simultaneously communicate with the FortiGate and also initiate a connection to the Arduino IDE for debugging. This prompted investigation of another board already on-hand, the Arduino Mega.

Arduino Mega. A slight upgrade to the Arduino Mega ADK R3, with its four hardware UARTs, immediately solved the serial-interface limitation [6]. Its larger I/O footprint also made it easy to integrate additional peripherals. However, the Mega lacks a built-in display and, as a design, also represents a legacy-class board—large and relatively expensive for its capabilities compared to newer options. More broadly, both the Mega and other Arduino boards such as the TinkerKit LCD share the platform's underlying 8-bit AVR architecture and limited flash storage, which impose tight constraints on program size and have become increasingly outpaced by modern microcontrollers.

Raspberry Pi Zero 2 W. Attention then turned to the much more robust Raspberry Pi boards; and the next to be evaluated was the Zero 2 W, which proved to be an excellent platform—more than capable of meeting all the project's requirements with its quad-core CPU, 512 MB of RAM, and microSD storage [9]. However, the fact that it is a full Linux system meant that it was, in a sense, *too capable*, introducing additional considerations such as OS selection, updates, and patches that would complicate the build. In a context where simplicity and deterministic behavior were priorities, this level of complexity would have introduced unnecessary maintenance overhead and potential points of failure over time.

Raspberry Pi Pico W. Ruling out the Zero 2 W left the Pico W as the next candidate. Although this board includes on-board Wi-Fi/BLE, wireless functions were not used in this project; evaluation focused on the RP2040 platform itself. The Pico W offered an effective balance of capability and simplicity: its dual-core Arm Cortex-M0+ at 133 MHz and ample SRAM provided far more headroom than 8-bit Arduino-class boards while retaining a deterministic, bare-metal feel without the overhead of a full Linux system [7]. Its low cost, small footprint, and mature MicroPython/toolchain ecosystem made it a solid base for the prototype, with broad community adoption indicating good long-term ecosystem support.

Table 2.1: Comparison of candidate boards by specification. See notes for feature-specific remarks. Technical specifications derived from official datasheets and product briefs [4, 5, 6, 7, 8, 9].

Spec	TinkerKit LCD (Leonardo)[a]	Arduino Mega 2560	Raspberry Pi Pico (Pico 2) W	Raspberry Pi Zero 2 W
CPU	ATmega32u4 @ 16 MHz	ATmega2560 @ 16 MHz	Dual ARM M0+ @ 133 MHz (M33 @ 150 MHz)[b]	Quad ARM-A53 @ 1 GHz
RAM	2.5 KB SRAM	8 KB SRAM	264 KB (520 KB) SRAM	512 MB RAM
Flash	32 KB onboard	256 KB onboard	2 MB (4 MB) onboard	No onboard Flash
Security	None (no hardware security)	None (no hardware security)	None (Pico); Pico 2: Signed boot, OTP key storage, SHA-256, TRNG[c]	OS-level only; no dedicated MCU security features
Interfaces	1 × UART, I^{2}C, SPI	4 × UARTs, I^{2}C, SPI	2 × UARTs, I^{2}C, SPI, PIO	2 × UARTs, I^{2}C, SPI, USB
Expansion	None	None	External SD via add-on	microSD slot for OS and storage
WiFi / BT	No / No	No / No	Yes (802.11n) / Yes (BT 5.2)[d]	Yes (802.11n) / Yes (BT 4.2)
Logic Voltage (V)	5	5	3.3	3.3
Draw (mA)[e]	~50 typical	~70 typical	~90 typical + peripherals	~100–160 idle, higher under load
Board Size (mm)	68 × 53	101 × 53	51 × 21	65 × 30
Languages	C / C++ (Arduino IDE)	C / C++ (Arduino IDE)	C / C++ (SDK), MicroPython, CircuitPython	Full Linux stack (Python, C/C++, Node.js, etc.)
Price (USD)[f]	~ \$20	~ \$25	~ \$6 (\$7)	~ \$15

[a] The TinkerKit LCD was actively sold prior to 2015 and is now discontinued.
[b] Pico 2 uses the RP2350 chip, featuring dual Cortex-M33 or selectable Hazard3 RISC-V cores.
[c] Pico 2 introduces enhanced security including signed boot support, 8 KB OTP storage, hardware SHA-256 accelerator, and a hardware true random number generator.
[d] Wi-Fi (802.11n) and Bluetooth 5.2 are available only on the Pico W and Pico 2 W models.
[e] Power consumption values are approximate typical current draw measured at 5 V input (USB or 5 V pin), even for boards with 3.3 V logic internally.
[f] Prices shown are launch MSRPs; current availability and market prices may differ.

Waveshare RP2040-Plus. While the Raspberry Pi Pico platform was selected for its balance of simplicity and capability, a third-party variant—the Waveshare RP2040-Plus—was ultimately chosen for the prototype due to several practical enhancements. It retains the RP2040 MCU but expands on-board flash to 16 MB (versus 2 MB on a standard Pico), providing ample space for firmware, fonts, and user data. It also integrates Li-ion charge/discharge for untethered field operation. Notably, it has *no on-board radio*. These upgrades, combined with drop-in compatibility with Pico headers and peripherals, made it a strong foundation for this build (Figure 2.3).

Upgrade Path (Pico 2 W). Mid-project, Raspberry Pi introduced the Pico 2 W (RP2350), which adds a faster core (150 MHz), expanded RAM (520 KB) and on-chip flash (4 MB), plus security features absent on Pico 1. Because it is pin-compatible, migrating is trivial (reflash the UF2 and reboot). Although time constraints prevented a full migration for this thesis, the Pico 2 W's security primitives—signed boot, hardware-backed OTP key storage, on-chip SHA-256, and a TRNG—directly support planned milestones such as a verifiable secure-boot chain, sealed credential vaults, and integrity-checked configuration bundles. Adopting the Pico 2 W in future iterations would therefore boost performance and provide a hardware-rooted trust anchor that aligns with embedded-security best practices.

Alternate Upgrade Path (ESP32-S3). While the Pico 2 W is a drop-in upgrade, the ESP32-S3 is a credible alternative when integrated radios and memory options are desired. It offers dual-core Xtensa, 2.4 GHz Wi-Fi + BLE, plentiful I/O, and mature MicroPython support, plus SoC-level secure-boot and flash-encryption in the ESP-IDF toolchain. A representative board-level reference is Heltec's WiFi LoRa 32 V3 (ESP32-S3), which couples Wi-Fi/BLE with LoRa and common QSPI flash/PSRAM configurations [10]. Practically, for this project's UART-driven, UI-light workflow, the Pico family's determinism and the RP2350's hardware security features make it the default; the S3 is best kept for radio-heavy or OTA-first variants, e.g., future additions as discussed in Section 7.4, Long-Range Wireless: LoRa.

Benchmark snapshot (MCU-only). To keep comparisons apples-to-apples, this snapshot covers *MCU-class* boards only (RP2040/Pico, RP2350/Pico 2, ESP32 family); *Arduino-class 8-bit* and *Linux-class* boards (e.g., Raspberry Pi Zero 2 W) are excluded. Table 2.2 summarizes representative MicroPython results that primarily compare Pico 1 (RP2040) and Pico 2 (RP2350); any ESP32 rows in the table refer to earlier ESP32 (LX6) devices and are provided for context—not the ESP32-S3. For a direct Pico W vs. Pico 2 W vs. ESP32-S3 view, the time-based benchmarks reproduced in Figure 2.2 show workload-dependent rankings: `bubble.py` favors Pico 2 W over ESP32-S3 and Pico W, while `fibo2.py` favors ESP32-S3 over Pico 2 W and Pico W [11]. Additional arithmetic MOPS charts for RP2040↔RP2350 appear in Appendix Figures A.5–A.6, which support the Pico 2 headroom claims without implying a universal ordering across all workloads [12, 13].

Table 2.2: MicroPython Benchmark Results Across Selected Platforms

Board	**CPU**	**MHz**	**PS/s**[a]	**i-add**	**f-add**	**i-mul**	**f-mul**	**i-div**	**f-div**
PC	Intel	–	200	–	–	–	–	–	–
Pico	RP2040	133	1.08	6.95	1.36	6.94	1.38	3.22	1.23
Pico 2 (M33)	RP2350	150	2.48	9.96	9.97	9.97	9.34	8.30	5.34
Pico 2 (Hazard3)[b]	RP2350	150	2.48	11.49	2.41	11.49	2.93	4.67	0.74
Wemos	ESP32	–	1.04	–	–	–	–	–	–
M5Stack Core	ESP32	240	0.66	7.23	7.02	7.45	7.02	6.81	3.14
Wio Terminal	ATSAMD51	120	–	7.05	6.31	7.04	5.99	5.70	3.74

[a] PS/s = Pystones per second.
[b] Pico 2 Hazard3 scores are for the secondary RISC-V core on the RP2350.
Sources: Data compiled from Ardusimple [12] (Pystones) and Cheppali [13] (MOPS benchmarks).

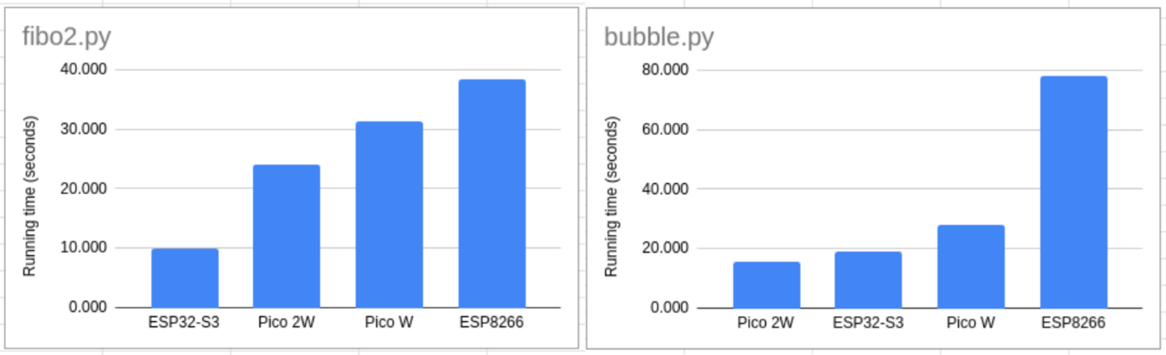


Figure 2.2: Time-based MicroPython benchmarks on MCU-class boards (Pico W, Pico 2 W, ESP32-S3): left, `fibo2.py`; right, `bubble.py`. Reproduced from Grinberg [11]. Rankings vary by script; see text for discussion.

2.3 Subsystems: Display, Touchscreen, Storage, and Power

Waveshare ResTouch 2.8" — overview. The prototype uses the Waveshare Pico-ResTouch 2.8" carrier as the integrated display, touch, and removable-storage subsystem. This carrier plugs directly onto the Pico-style header of the RP2040-Plus module, minimizing wiring and simplifying assembly. The reader can find the prototype's GPIO and peripheral-bus assignments for the carrier in Table 2.3.

LCD (display). The ResTouch provides a 2.8-inch IPS panel with a 320×240 QVGA pixel matrix that is sufficient for rendering FortiGate CLI text and simple status screens [14]. In practice the display is driven over an SPI bus from the MCU and uses a modest framebuffer strategy to keep redraw latency low while preserving MicroPython responsiveness; implementation details and driver notes are discussed in Section 3.5. The display hardware and required SPI signals are summarized in Table 2.3.

Touch (XPT2046). Touch input is provided by an XPT2046 resistive touch controller on the carrier. The touch controller communicates via SPI (MISO is read by the XPT2046 only) and raises an IRQ line when contact is detected; a lightweight calibration routine maps raw ADC touch coordinates into screen pixels. The firmware implements debouncing and a simple two-point calibration workflow to keep touch calibration fast and robust for field technicians. Relevant pin assignments and the active IRQ polarity are listed in Table 2.3.

Removable storage (micro-SD). Configuration bundles and user profiles are stored on the carrier's removable micro-SD socket; the prototype uses FAT32 for interchangeability across platforms. Firmware remains on the RP2040-Plus module's onboard flash (separate from the SD card), while the SD card holds text-based CLI bundles that the device loads at run time. The card is mounted at

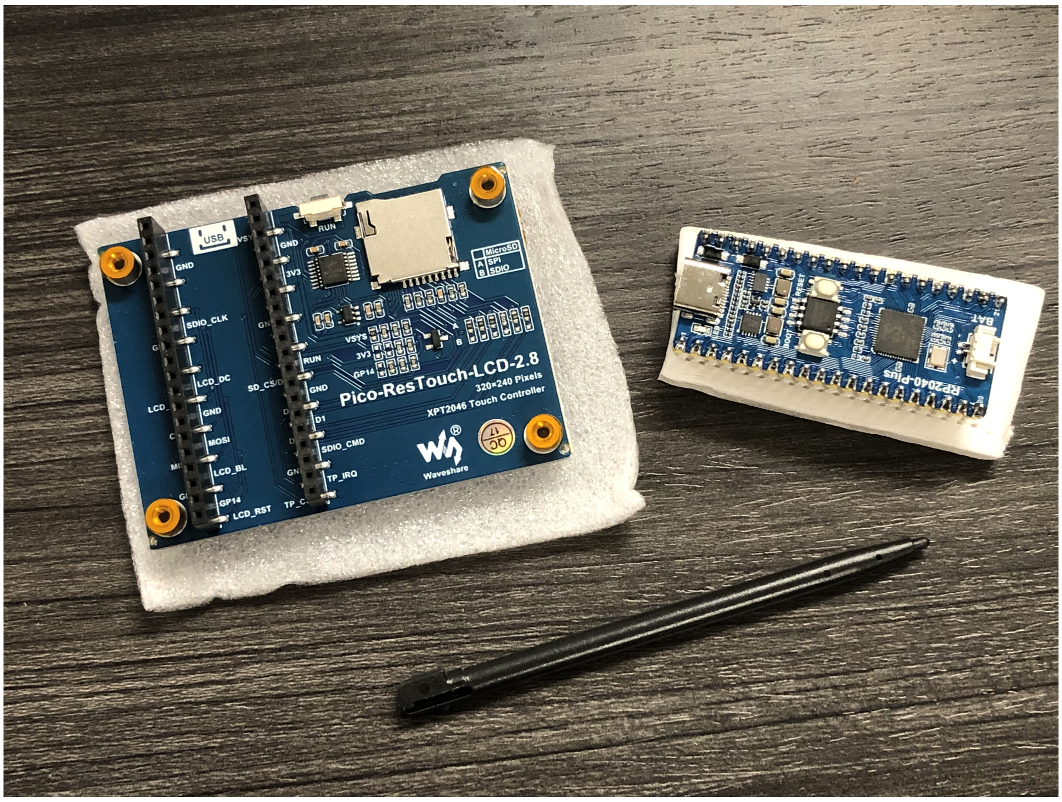


Figure 2.3: Waveshare RP2040-Plus microcontroller (right) and Pico-ResTouch-LCD-2.8 touchscreen module (left), showing the dual-row header that mates the two boards, the onboard micro-SD socket, and the resistive stylus used for input.

`/sd`; the current firmware presents a clear "no card" error state when a card is absent. SD handling code can be extended to verify bundle integrity (for example by checking a checksum or signature) and to detect corrupt or partially written files. Typical CLI bundles are modest in size (tens to a few hundred kilobytes), so SD transfer latency is unlikely to be a user-visible problem for configuration pushes. Migration to a Pico 2 W would preserve this storage layout while enabling optional signing and encryption of bundles as discussed in Chapter 4, Threat Modeling.

Power and battery. FortressFlash is designed to run from universal 5 V USB sources (for example a USB-C power bank, a wall adapter, or even the USB port on a FortiGate appliance or laptop) and may also be fitted with an optional LiPo cell for untethered use. The RP2040 core and peripherals operate at 3.3 V; the RP2040-Plus MCU module used in the prototype includes an on-board switch-mode regulator rated well above the device's steady-state draw (the regulator is specified to supply up to 2 A at 3.3 V) [15].

In practice the Waveshare ResTouch + Pico stack runs comfortably from common USB supplies—Waveshare documentation and hands-on testing with a vanilla Pico suggest the combined draw is on the order of a few hundred milliamps under normal UI and console-transfer activity [14]. Future hardware expansions—such as LoRa, Wi-Fi, or higher-brightness backlights—could exceed this

Table 2.3: Prototype GPIO and Peripheral-Bus Assignments

RP2040-Plus Pin(s)	**Signal (bus)**	**Connected Peripheral / Purpose**
		Serial console to FortiGate (via MAX3232 level shifter)
GP0	UART0_TX	TXD $\rightarrow$ RS-232 pin 2 (Null-modem crossover to FortiGate RxD)
GP1	UART0_RX	RXD $\leftarrow$ RS-232 pin 3 (from FortiGate TxD)
3V3 (pin 36)	VCC	Powers MAX3232 board (3 V logic side)
GND (pin 28)	Ground	Common return for MAX3232 and display stack
		ResTouch 2.8 in. LCD (ST7789) — SPI 1 bus
GP10	SPI1 SCK	LCD CLK
GP11	SPI1 TX	LCD MOSI
GP12	SPI1 RX	LCD MISO (only used by touch/XPT2046)
GP8	LCD_DC	LCD Command/Data pin
GP9	LCD_CS	LCD chip-select
GP15	LCD_RST	LCD hardware reset
GP13	LCD_BL	PWM back-light drive
		XPT2046 Touch-controller (shares SPI 1)
GP16	TP_CS	Touch-controller chip-select
GP17	TP_IRQ	Touch IRQ (active-low pen-down)
		On-board micro-SD slot (4-bit SDIO interface)
GP5	SDIO_CLK	SD clock
GP18	SDIO_CMD	SD command (DI)
GP19	SDIO D0	Data bit 0 (also SPI MISO fallback)
GP20	SDIO D1	Data bit 1
GP21	SDIO D2	Data bit 2
GP22	SD_CS/D3	Data bit 3 / chip-select in SPI-mode
		Miscellaneous / power
VSYS (pin 39)	5 V input	Powers ResTouch via header (USB-C or Li-ion pack)

Notes:

- For annotated pin-out diagrams of the Waveshare RP2040-Plus and the ResTouch 2.8 in. LCD, see Appendix A.2, Pinout Diagrams.
- The ResTouch header hard-wires every line shown; no additional jumper wires are required once the RP2040-Plus is inserted.
- SPI1 is used for both the ST7789 LCD and the XPT2046 touch-controller; chip-selects keep the buses independent.
- The MAX3232 board's DE-9 connector follows a DCE pin-out and mates to the FortiGate rollover cable through a male–male null-modem adapter, as illustrated in Fig. 2.4.

envelope when powered from a standard Pico or Pico 2. In such cases a dedicated 5 V rail or an external regulator may be necessary to maintain stability under peak loads. For field use a modestly rated USB supply (1 A or higher at 5 V) provides useful headroom; many modern phone chargers and power banks provide 2 A or more.

2.4 Subsystems: Serial UART

As with most professional-grade networking gear, FortiGate appliances expose a serial console for direct access to the command-line interface (CLI). On earlier mid-range models—such as the **60D**—the console appears on a mini-USB `USB MGMT` port that contains an on-board serial-to-USB bridge. Fortinet dropped that feature in later revisions. The **60E** unit used throughout this project instead presents the console on an RJ-45 (Cat 5) jack; Fortinet ships it with an RJ-45-to-DE-9 (female) rollover cable for connection to a standard RS-232 port. The console ports are visible on the rear panels of the three stacked firewalls in Figure 1.1 and the block diagram in Figure 2.1, labeled `CONSOLE` and located to the right of the DC jack.

While initial testing was performed using a USB-to-RJ-45 console cable (occasionally included with newer FortiGate units), a standard USB-to-DE9 (male) serial adapter was used when a direct RS-232 connection from a workstation was required. This arrangement allowed for quick changeovers between the development workstation and the prototype without needing to swap console cables, since both relied on the same DE-9 connector and rollover cable. What follows is a breakdown of how the serial interface was integrated into the prototype hardware, along with issues that emerged during testing and iteration.

Pinout Mapping and Cable Configuration Figure 2.4 illustrates the complete console wiring path used for the prototype. The FortiGate's RJ-45 console port connects to a Fortinet rollover cable terminating in a DE-9 (female) DCE connector. This connects through a DE-9 male-to-male null-modem adapter, which crosses the transmit and receive lines. The other end of the null-modem feeds into a MAX3232-based RS-232 level shifter, which presents another DE-9 (female) DCE interface. The level shifter converts standard RS-232 voltage levels to 3.3 V TTL, which are connected to UART0 on the RP2040-Plus. In addition to the signal lines, the RP2040-Plus provides a regulated 3.3 V output and ground connection to power the `HX-004` level shifter directly. This chain enables seamless communication between the FortiGate console and the microcontroller while maintaining signal integrity and logic-level compatibility.

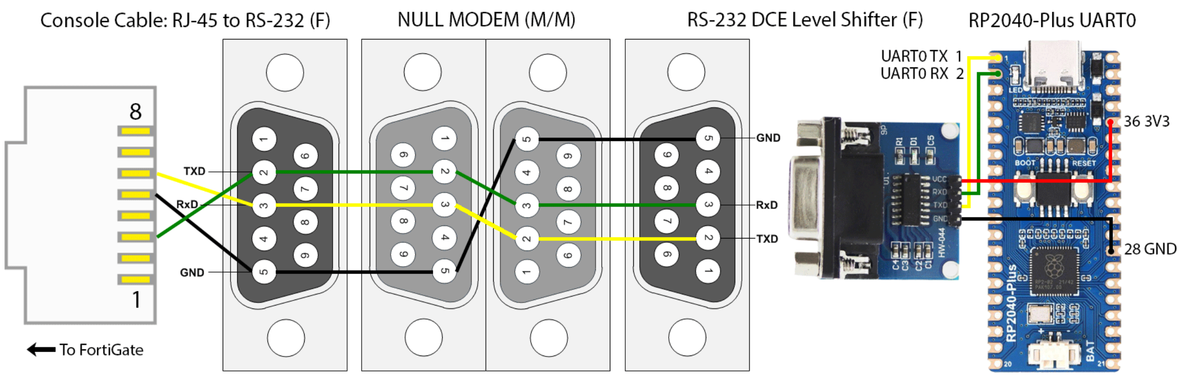


Figure 2.4: Signal flow from FortiGate RJ-45 console port to UART0 on the RP2040-Plus MCU, via standard RS-232 and `HX-004` level-shifting components. Adapted from sources [16, 17, 15].

MAX3232 Converter Thermal Faults

Symptom. Early designs used a postage-stamp `HW-027` "RS-232↔TTL" board (15 mm × 9 mm) to level-shift the FortiGate console. Even with $\texttt{V}_{\texttt{CC}}$ (3.3 V) and `GND` correctly wired on the logic edge *and* no load on the RS-232 side, the board overheated within seconds. This behavior contrasted sharply with the low standby current (~1 mA) specified in the official TI MAX3232 datasheet [18], indicating the presence of a faulty or counterfeit chip (see Figure 2.5).

Root–cause analysis. Two factors remained:

a) **Mirrored silkscreen.** The board prints *"VCC"* on *both* long edges; the RS-232 edge is actually the $\texttt{V}_+$ charge-pump node ($V_{CC} + 6$ V). Powering this pad latches the chip in a high-current state.

b) **Counterfeit silicon.** Many low-cost breakouts ship with look-alike chips that omit the tiny internal bias resistors. With those resistors missing, unused inputs float, the device switches itself rapidly, and the package overheats—regardless of perfect pinout [19, 20].

Attempted community fix. An Amazon reviewer (*ET*) published a "corrected" pinout after tracing the IC leads with a multimeter [21]. Replicating that wiring eliminated the obvious latch-up mistake, yet the module still idled hot—evidence that the problem was deeper than silkscreen confusion.

Resolution. The prototype now uses a larger `HX-004` breakout (see Figure 2.4). Although it still carries a low-cost, non-TI MAX3232 clone, the board includes all five 100 nF charge-pump capacitors and a tidier layout, so it idles cool. Its right-angle DE-9 *female* connector is wired for the standard *data-terminal-equipment* (DTE) pinout (pin 2 = RxD, pin 3 = TxD). The FortiGate console cable behaves as *data-communication-equipment* (DCE) (pin 2 = TxD, pin 3 = RxD). To create the required crossover and match connector genders, a 6 cm *male-to-male* null-modem adapter that swaps pins 2↔3 is inserted in-line.

Future-proof option. Should the design be miniaturised again, a quality-controlled module such as the *Adafruit RS-232 Pal* (Product 5987, $3.95) provides a TI-branded MAX3232E and clear silkscreen, eliminating clone-silicon risk entirely [22].

Lessons learned.

- **Source traceable components.** The $1 clone modules consumed more engineering time than a $4 proven breakout.

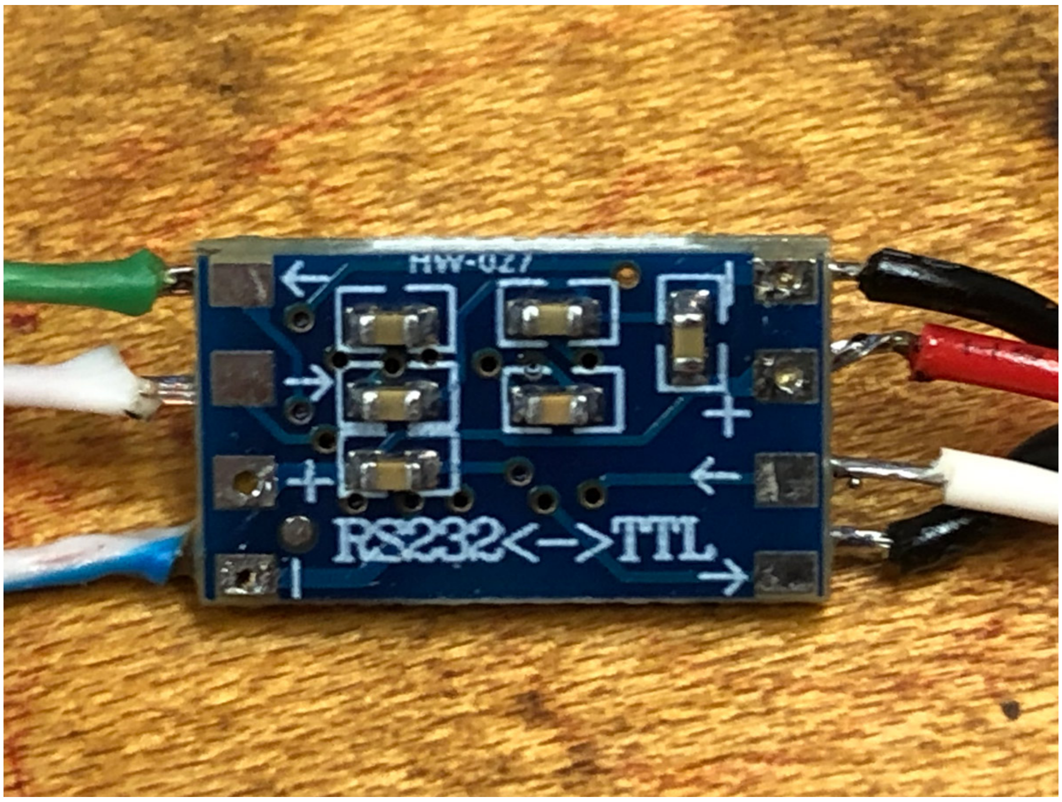


Figure 2.5: Close-up of the `HW-027` RS-232↔TTL level-shifter board implicated in early thermal issues.

- **Document hidden pin conventions.** A single BoM note about the mirrored silkscreen and DTE pinout would have halved debug time.

2.5 Enclosure and Connector Integration

The prototype is housed in a 3D-printed enclosure based on a design published on Thingiverse by Tasuku Suzuki [23]. The model, originally intended for the Waveshare 2.8 ” capacitive touchscreen and Raspberry Pi Pico, was selected due to its close alignment with the chosen hardware stack and its availability under a permissive CC BY-SA license.[3] The enclosure was printed in-house using red ABS filament from eSUN on a Prusa i3 MK3S+, with only minor fit adjustments, as shown in Figure 2.6.[4]

During early design planning, an RJ45 keystone jack was considered for the RS-232 interface, allowing the use of standard CAT5 cables for console connections. This would have provided a clean, low-profile rear connector. As discussed in Section 2.4, MAX3232 Converter Thermal Faults, however, the unbranded MAX3232 module used in this configuration exhibited unsafe thermal behavior and was ultimately rejected.

[3]For a render of the original enclosure model, see Figure A.3 in the appendix.

[4]For a view of the enclosure mid-print on the Prusa i3 MK3S+, see Figure A.4 in the appendix.

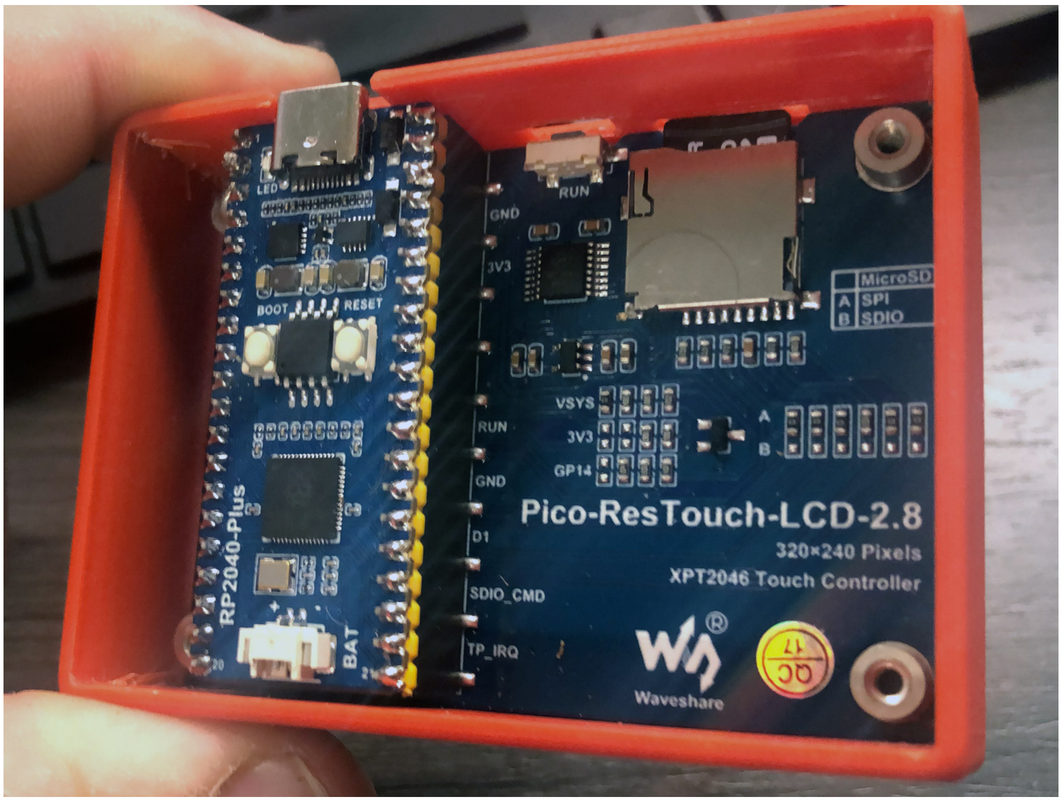


Figure 2.6: Waveshare 2.8" LCD and RP2040-Plus mounted in 3d-printed ABS case.

A more robust MAX3232 breakout board with a built-in DE-9 female connector was adopted to serve as the stable, long-term solution for the console interface. As the enclosure did not feature a cutout for either connector type, the DE-9 port was temporarily hand-fit during prototyping to create a side opening. While this modification enabled functional access to the serial port, it prevented the rear panel from being re-attached. Due to time constraints, a proper CAD remix of the original STL file was deferred, but future revisions will likely address this with a clean rear-mounted DE-9 cutout or a revised modular panel system. Despite this compromise, the prototype remains fully operational, and the enclosure provides sufficient structural protection and screen stability for lab testing and demonstration purposes.

2.6 Complete FortressFlash Assembly

With all major subsystems integrated—MCU, display, storage, UART interface, and enclosure—the prototype reaches its final field-ready form. The completed device unites these elements into a single hand-held tool, compact enough for deployment by entry-level technicians while robust enough to support future enhancements. Figure 2.7 shows the complete FortressFlash assembly from both front and rear perspectives.

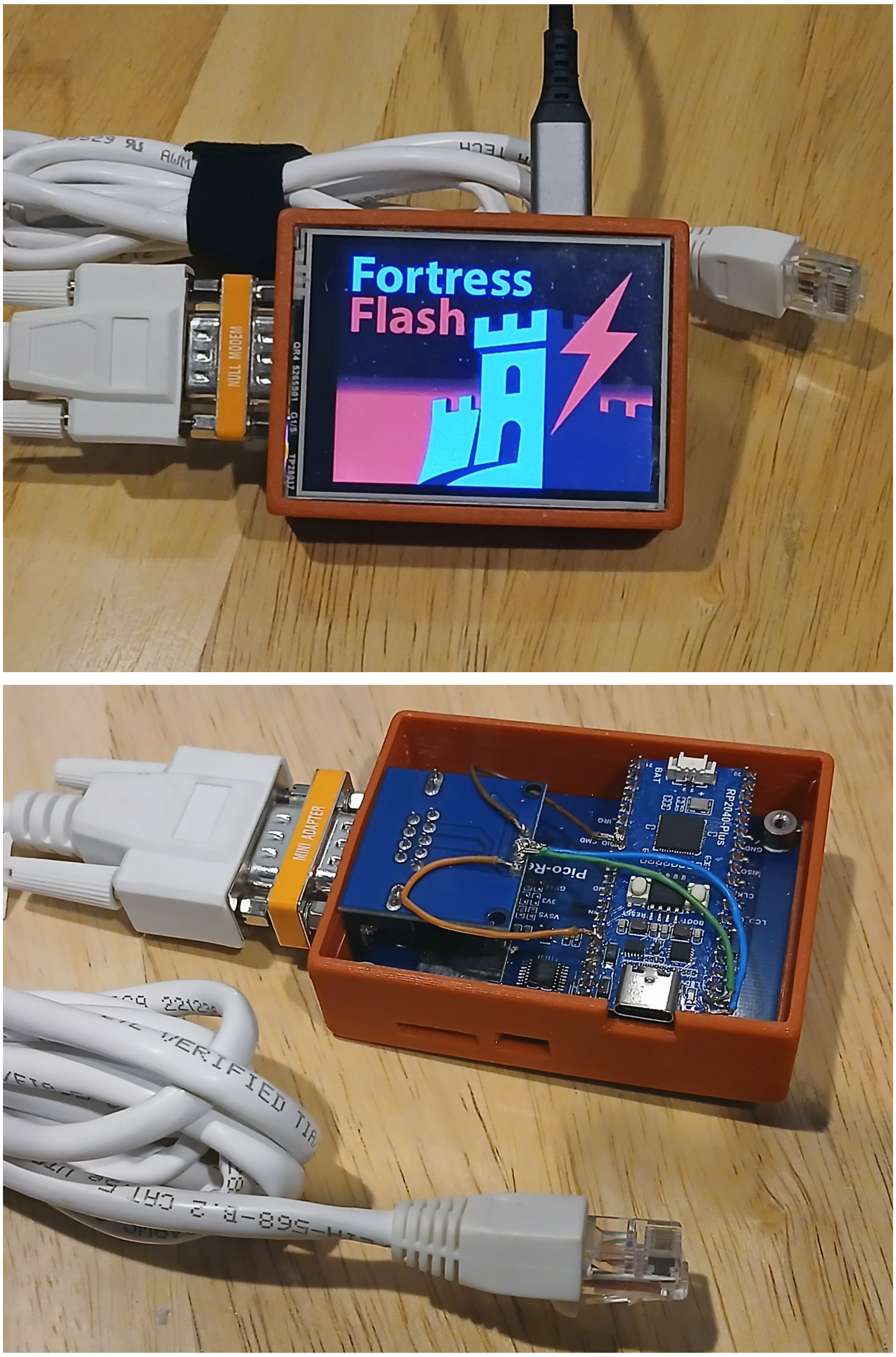


Figure 2.7: Complete FortressFlash assembly. Front view powered on with splash screen and attached DE-9 console cable via in-line null-modem adapter (top), and rear view exposing the RP2040-Plus module, wiring, and DE-9 connector integration (bottom).

Chapter 3: Software Design

3.1 MicroPython for Rapid Prototyping

When evaluating firmware options for the device, MicroPython emerged as the most practical choice for rapid prototyping. Unlike traditional C/C++ development with the vendor SDK—which requires a full toolchain setup, compilation cycles, and more rigid debugging—MicroPython provides an interactive, high-level environment directly on the microcontroller. Its REPL (Read–Eval–Print Loop) enables immediate feedback from the hardware, allowing code to be tested, modified, and re-deployed in seconds. This dramatically shortens the iteration cycle compared to a compile–flash–reset workflow, and its Pythonic syntax minimizes repetitive setup code, making it easier to express hardware interactions such as GPIO, UART, SPI, and I^{2}C in clear, concise statements.

The ecosystem around MicroPython is mature and well-supported for common prototyping hardware, including the Raspberry Pi Pico series. Libraries for sensors, displays, and communication peripherals are readily available and easily imported, accelerating development and reducing the need to write low-level drivers from scratch. For these reasons—immediacy of testing, reduced complexity, and a robust library ecosystem—MicroPython offers the fastest path from concept to working prototype while still leaving room to migrate to lower-level C/C++ implementations if needed for production.

3.2 Firmware Architecture and Operational Flow

The firmware was envisioned as a modular stack, with each feature designed to be self-contained yet extensible. Table 3.1 summarizes all major features, their intended purpose, and current status.

Scrolling Terminal (Implemented) The firmware maintains a circular buffer that captures each line of FortiGate CLI output received over UART. Lines are rendered in a fixed-width font and auto-scroll upward, providing visual confirmation of progress and easy troubleshooting without an external console. Future revisions will optionally save this buffer to encrypted storage for post-deployment auditability with additional command softkeys (status, etc).

Table 3.1: Firmware feature inventory and implementation status.

Feature	**Description**	**Status**
Scrolling Terminal	Displays real-time CLI feedback from the FortiGate.	Implemented
Main Menu UI	Primary navigation touchscreen interface.	Implemented
SD Profile Loading	Enumerates configuration files from microSD and maps them to a file picker interface.	Implemented
Basic Error Handling	Handles missing SD card or missing files.	Implemented
Serial Dispatch Engine	Sends selected profile line-by-line over UART.	In Progress
PIN-based Unlock	Require PIN prior to access (rate throttling and lockout).	Planned
File Verification	Handle Bad or empty .config files	Planned
Device Verification	Confirm FortiGate serial number.	Planned
Secure Passwords	Secure FortiGate credentials handling using Pico 2 OTP	Planned
Zero-Touch Provisioning	Configure new FortiGate out-of-the-box.	Planned
Config File Signing	Verify cryptographic signatures before deployment.	Planned
Encrypted SD Storage	AES-encrypts logins, configs, logs, other data at rest using PIN + OTP device key.	Planned
Logs & Audit Trail	Store full output and deployment logs (signed).	Planned
Device Provisioning App	Setting-up and provisioning the device itself.	Planned
Expanded Device Support	Support different versions, manufacturers, and devices (e.g., Cisco, Palo Alto, Switches)	Future
Users & Groups	Implement individual users and groups (helpdesk, admin)	Future
BLE/Wi-Fi/LoRa Updates	Load new configs wirelessly from admin device.	Future
Firmware Validation	Use Pico 2 secure boot validation	Future
Firmware Updates	Method for patches, fixes, and updates	Future
Firmware Deployment	Update device (i.e., FortiGate) Firmware	Future
API Mode over TCP/IP	Alternative mode using FortiGate REST API.	Future
Intent-Based Networking	Use NLP to create/deploy configs.	Deferred

Main Menu UI (Implemented) A lightweight UI framework draws four capacitive touch-zones across the lower third of the 320 × 240 LCD. Button labels map dynamically to profile names or actions and the menu logic is state-driven, simplifying future expansion to multi-page menus or role-based layouts.

SD Profile Loading (Implimented) At startup the `sd_loader` module scans the `/configs/` directory for files with a `.conf` extension, alphabetically sorting them before registering up to four per page. Current work focuses on graceful handling of long filenames and UTF-8 characters, plus a pagination routine for deployments with more than four profiles.

Basic Error Handling (Implimented) Initial guards check for SD-card presence and non-zero profile count, presenting modal error dialogs (‘No SD Detected’, ‘No Profiles Found’). Next tasks include CRC checks on file open and user-cancelable retry loops.

Serial Dispatch Engine (In-Progress) Upon profile selection the dispatcher opens the file in buffered mode and transmits each line with CR/LF terminators, inserting adaptive inter-command delays based on prompt detection (>). Next milestones include per-command timeout handling and early-abort on detect-error strings (e.g. “syntax error”).

PIN-Based Unlock (Planned) A pre-menu screen will prompt for a 4–8 digit PIN. The PIN concatenates with a device-unique salt stored in Pico 2 OTP to derive an AES-128 key via PBKDF2. Three consecutive failures trigger a 30-second exponential back-off; ten failures lock the device until power-cycle.

File Verification (Planned) Each configuration file will undergo a lightweight sanity scan (regex for `config` / `end` blocks and size > 1 kB). Malformed or empty files raise a blocking alert and are skipped in the menu list.

Device Verification (Planned) Before deployment the FortiGate’s serial number (obtained via `get system status`) will be hashed with a public salt and compared to an allow-list stored on the SD-card. Mismatch triggers a user override prompt or aborts deployment—preventing accidental pushes to the wrong appliance.

Secure Passwords (Pico 2 OTP, Planned) Console credentials will be AES-encrypted on SD-card using a key resident in the Pico 2’s OTP region. Decryption occurs only in SRAM; plaintext is zeroed immediately after use, limiting exposure even if RAM is inspected via debug probes.

Zero-Touch Provisioning (Planned) A wizard mode will detect an unconfigured FortiGate (factory default banner) and automatically push a bootstrap template that enables GUI, sets admin password, and reboots—all without manual intervention.

Config File Signing (Planned) Profiles will be wrapped in a JSON manifest containing SHA-256 hash and an ECDSA signature. The device will verify the signature against an embedded public key before transmission, ensuring integrity and origin authenticity.

Encrypted SD Storage (Planned) File-level encryption (AES-GCM) will protect configuration bundles and logs stored on the SD card. Keys derive from the user PIN combined with a hardware salt. A shadow header stores an encrypted backup of the key-encryption-key to permit secure PIN rotation.

Logs & Audit Trail (Planned) After each deployment the scrolling buffer and a structured JSON log (timestamp, profile, target SN, hash) will be appended to an encrypted `deploy.log`. Logs will be signed with an HMAC to prevent repudiation.

Device Provisioning Application (Planned) A cross-platform host-side loader will provision devices and prepare removable media for field use. In the lab it flashes firmware, derives and burns OTP salts and device identifiers, and writes an initial encrypted credential bundle—establishing a trusted supply-chain baseline before units leave the bench. For field operations it standardizes SD-card formatting and on-disk layout, packages and signs configuration profiles, and optionally enables at-rest encryption with keys sealed on-device and unlocked by a PIN at use time. The loader performs pre-flight validation (target model/firmware, prompt expectations) and emits a preparation log designed to pair with on-device console logs. A simple GUI supports bench technicians, while a CLI enables scripted pipelines and integration into CI/CD workflows.

Expanded Device Support (Future) Abstracting the serial dispatcher and command templates will allow support for Cisco IOSstyle switches, Juniper SRX, Palo Alto, and other CLI-centric platforms (see Section 7.1. Vendor modules will live behind a common "device driver" interface.

Users & Groups (Future) Role-based menus will appear after PIN entry. Help-desk users get limited profile sets; admin users can push firmware or run diagnostics. Group policy will be enforced by encrypted role tokens on SD.

BLE / Wi-Fi / LoRa Updates (Future) A background task will advertise a BLE service that accepts signed profile bundles. Wi-Fi or LoRa mesh could permit over-the-air updates in remote sites; all transfers will use Ephemeral-Elliptic-Curve Diffie-Hellman for key exchange.

Firmware Validation (Future) The Pico 2 secure-boot chain will verify a signed firmware image before execution, blocking tampered binaries and enforcing an anti-rollback monotonic counter.

Firmware Updates (Future) Firmware updates will be delivered as signed UF2 images once migrated to C/C++. These are written over USB using the Pico's native bootloader, providing a simple and reliable recovery path.

FortiGate Firmware Deployment (Future) An advanced workflow will upload FortiOS images via TFTP or USB and issue CLI commands to flash them, followed by post-upgrade configuration injection—streamlining major version jumps.

API Mode over TCP/IP (Future) A separate transport layer will leverage HTTPS REST calls to the FortiGate API. Credential handling, session tokens, and JSON payload construction will reuse the security primitives built for serial mode but operate over Ethernet or Wi-Fi.

Intent-Based Networking (Deferred) If pursued in the future, natural-language prompts ("open port 443 to DMZ") would be handled by a companion wizard service (e.g., FortressFlash Cloud), shaped by vendor support and standards rather than embedded on the device itself (see Section 7.6).

3.3 Touchscreen UI and User Flow

The design of the user experience (UX) began with low-fidelity sketches on a whiteboard, where the overall workflow was first mapped out. These initial diagrams focused on the logical sequence of events rather than visual details—starting from device boot, SD card mounting, and configuration discovery, through profile selection, validation, deployment, and error handling. Branching conditions such as *no SD card inserted*, *no profiles found*, and *hash or decryption failure* were represented with simple decision nodes, so that key paths through the interface were considered early. The runtime follows a stepwise control flow as shown in Figure 3.1: boot leads to the splash screen, then to a PIN entry gate (planned) that enforces attempt throttling before revealing the main menu. From there, users can branch into Deploy Config (Figure 3.2), Terminal, or Shutdown, with additional features marked as planned.

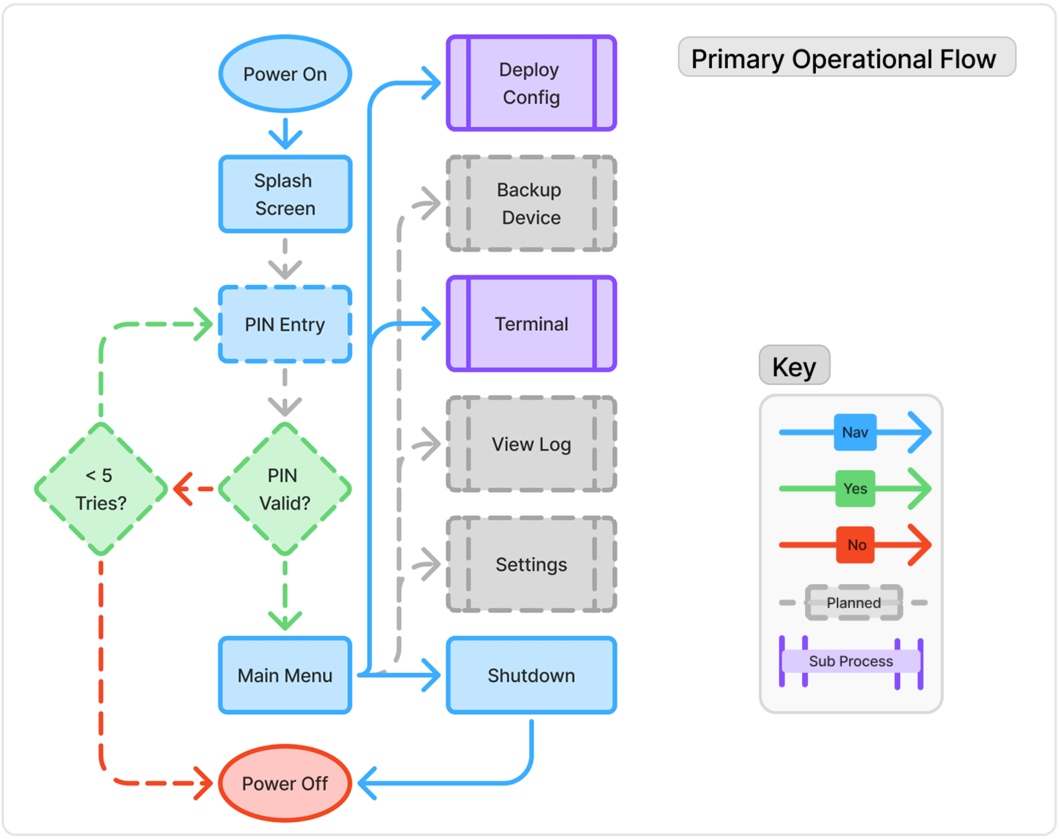


Figure 3.1: Primary operational flow. System startup proceeds from power-on and splash to a planned PIN entry gate, which enforces limited attempts before granting access to the main menu. Menu branches lead to Deploy Config, Terminal, and Shutdown; greyed items denote planned features.

Once the flow was conceptually stable, the design was translated into **Figma**, an online collaborative UI and prototyping tool. Each screen was recreated as a separate frame in 320 × 240 px resolution, matching the physical dimensions of the ResTouch 2.8 in. LCD. Core functional screens—such as the main menu, profile picker, confirmation dialog, and deployment success screen—were linked together using Figma's prototyping mode, while alternate branches for error states and special conditions were placed in a separate page for clarity.

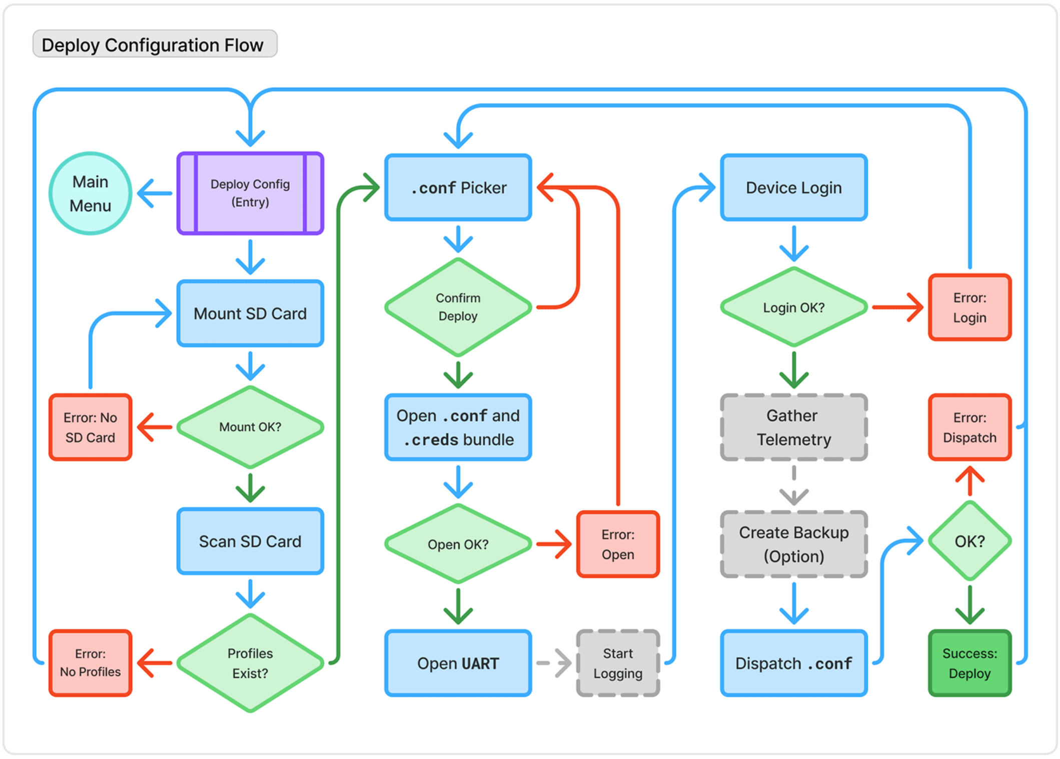
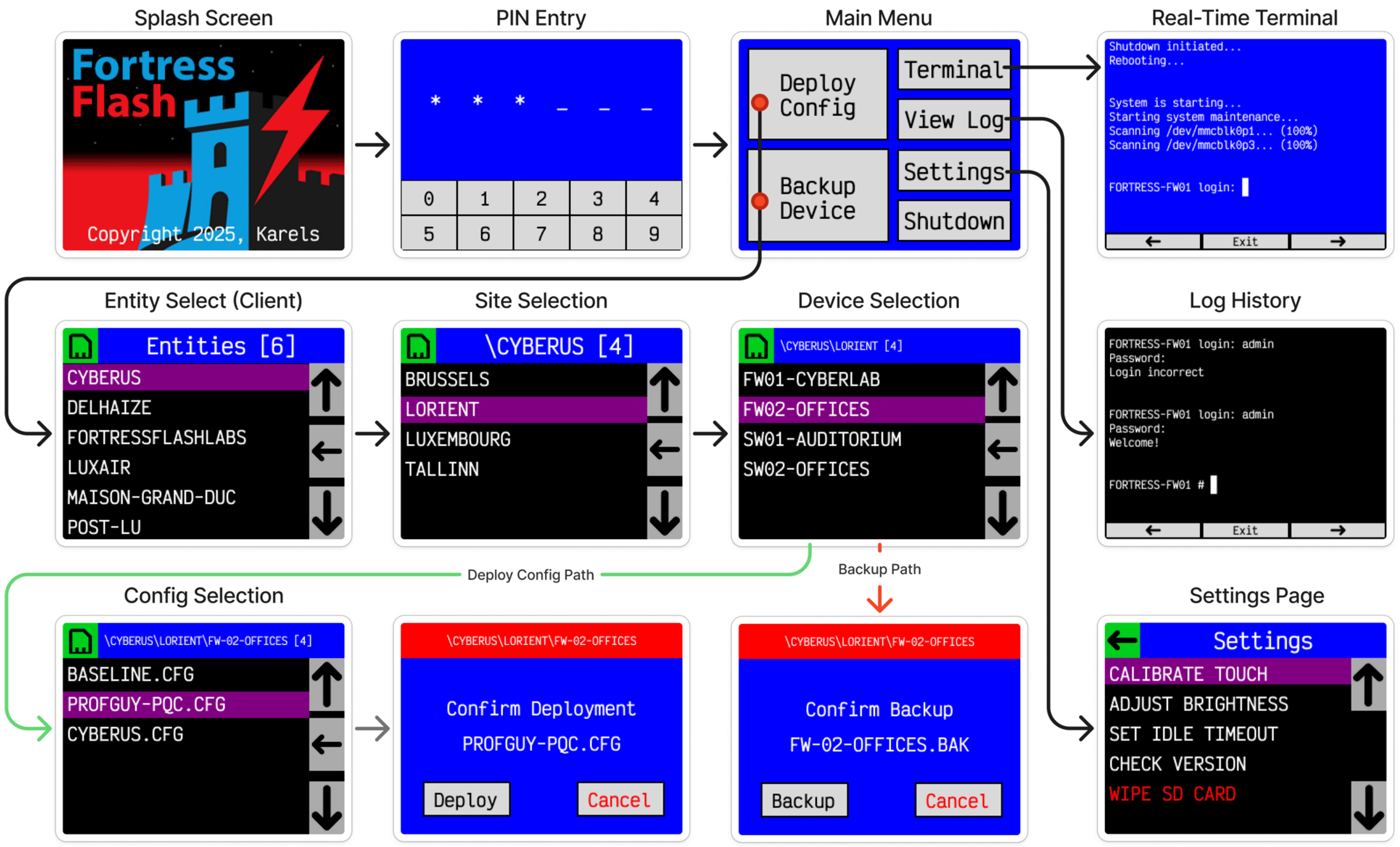


Figure 3.2: Deploy Config subprocess. SD mount/scan, profile selection, bundle open, UART open/login, optional telemetry/backup (planned), prompt-paced dispatch, and OK/error rails.

To connect these screens into a unified navigation diagram, **FigJam** was used. Frames from Figma were imported into a FigJam board and connected using labeled arrows to show the intended user paths. This allowed rapid iteration on interaction logic and highlighted redundant paths that could be simplified. Because error, loading, and confirmation states would clutter the primary diagram, these screens were documented separately in Appendix Figure A.7, forming a secondary UX chart dedicated to non-standard states.

By following this staged approach—from whiteboard sketches, to high-fidelity screen mockups, to automated flow mapping—the final UX diagram in Figure 3.3 captures both the logical structure of the application and the intended user journey, while the appendix diagrams ensure all secondary states are fully documented for reference. The resulting prototype closely guided implementation: the main menu and file picker UI on the actual device follow the Figma design with only minor adjustments, as shown in Appendix Figures A.8-A.10.



```
Shutdown initiated...
Rebooting...


System is starting...
Starting system maintenance...
Scanning /dev/mmcblk0p1... (100%)
Scanning /dev/mmcblk0p3... (100%)


FORTRESS-FW01 login: █
```

```
FORTRESS-FW01 login: admin
Password:
Login incorrect


FORTRESS-FW01 login: admin
Password:
Welcome!


FORTRESS-FW01 # █
```

Figure 3.3: FortressFlash UX flow illustrating primary screens, navigation paths, and core functions. Icons from [24, 25].
(Rotated for clarity—please tilt head responsibly.)

3.4 SD Card Configuration Management

The FortressFlash system uses a removable SD card as the primary medium for storing configuration data. Upon initialization, the device attempts to mount the SD card; if no card is present, the user is prompted to insert one. The directory structure under the `/configs` root is hierarchical and reflects the logical organization of client environments:

- **Entity** – The top-level folder corresponds to a client organization.
- **Site** – Each entity contains one or more site subfolders, representing physical or logical locations.
- **Device** – Each site folder contains one or more device subfolders, named according to the specific hardware asset.
- **Configuration Files** – Within each device folder, configuration files with the `.conf` extension store CLI command sequences in standard FortiGate/FortiOS format.

When the SD card is scanned, the system recursively indexes this hierarchy, building a structured mapping of all available configurations. Users can navigate this tree via the touchscreen interface to select a target device and configuration file.

Future enhancements will expand each device folder to include:

- A `.json` metadata file containing structured information about the device, including entity, site, device type, serial number, and credential mapping.
- An encrypted `.creds` credentials file, stored alongside the configuration, which will be decrypted at runtime using a combination of the operator's PIN and a device-unique key provisioned in OTP memory. This capability is planned for implementation on the Raspberry Pi Pico 2 platform, as it requires additional cryptographic capabilities not present in Pico 1.

This design allows configuration files to remain simple, portable, and human-readable while enabling secure storage of sensitive authentication material. The clear separation between operational configuration (`.conf`) and device metadata/credentials supports both ease of deployment and strong security controls.

3.5 Firmware Architecture and Developer Workflow

The prototype firmware is implemented in MicroPython v1.25.0 (release 2025-04-15) for a Raspberry Pi Pico (Waveshare 16 MB) paired with a 2.8" ResTouch LCD and an integrated microSD slot. Development and on-device execution began in the Thonny IDE for rapid bring-up (REPL, file transfer, and live execution over USB). As the codebase grew, the workflow migrated to Visual Studio Code with the MicroPico extension to gain first-class Git integration, project-wide navigation, and more reliable per-module upload/run from within the editor. This section reflects the submission build; minor divergences may occur as bug fixes land post-freeze, and Section 5.2 contains the latest bench outcomes. The complete lab and test hardware configuration is itemized in Appendix Table A.3.

Developer Workflow. Development followed an iterate-on-hardware loop: edit, sync to device, execute under the MicroPython REPL, observe serial output, and adjust. Early bring-up in Thonny prioritized quick iteration. The VS Code/MicroPico toolchain then standardized the cycle with integrated REPL, one-click upload/run, and source control (branching, diffs, commit history) for the growing project. Debug output was captured via the USB serial console; LCD rendering, touch events, and SD mounts were verified on the bench; and end-to-end UART tests were performed against the FortiGate console to validate login handling and prompt-paced dispatch.

Hardware Interface Drivers. Low-level device communication is handled by dedicated driver modules:

- `sdcard.py` implements a full SPI protocol stack for SD cards, including initialization sequences for both SDSC and SDHC/SDXC formats, block read/write operations, and error handling.
- `st7789.py` (Russ Hughes) controls the ST7789 TFT display, supporting rotation, RGB565 color, bitmap rendering, and multiple font formats.
- `xpt2046.py` manages the XPT2046 touch controller, providing calibrated coordinate mapping and sampling for stable input detection.
- `display.py` wraps the ST7789 driver with simplified methods for clearing the screen, drawing text, and managing fonts, while handling color constants and centering logic.
- `sd.py` offers higher-level SD card mounting, SPI reinitialization, and card presence detection.
- `touch.py` provides rotation-aware coordinate mapping for the touch interface, translating raw ADC readings into screen coordinates.

Service Layer. The service modules orchestrate higher-level application behavior:

- `error_dialog.py` presents interactive modal dialogs when hardware faults occur, such as missing or unreadable SD cards, with touch-enabled retry logic.

- `config_loader.py` scans the `/sd/configs` directory for site-specific `.conf` files, raising exceptions if no valid profiles are found.
- `config_indexer.py` builds and displays a hierarchical tree of configuration files, supporting both console and on-screen output (i.e., `/configs/{entity}/{site}/{device}/`).

Deployment Execution Flow. The deploy sequence can be verified step by step as follows:

1. On boot, SPI and hardware drivers initialize and the splash screen is displayed, leading to the main menu.
2. Selecting *Deploy Config* triggers an attempt to mount the SD card; if no card is present, an error dialog appears.
3. If the card is present, the configuration directory is scanned; an empty directory results in a "No Profiles" error.
4. Available profiles are listed for selection via the touch interface.
5. When a profile is chosen, its configuration file is opened and checked for integrity (hashing or decryption planned).
6. After operator confirmation, the configuration is transmitted over UART to the target device, with success or failure reported.

Design Considerations. This modular approach ensures that each hardware component can be developed, tested, and replaced independently. By isolating low-level SPI and GPIO operations within drivers, and encapsulating user interaction in service modules, the firmware remains maintainable and adaptable for future expansions, such as secure boot, encrypted profiles, or network-based configuration retrieval.

Chapter 4: Threat Modeling

4.1 Overview

FortressFlash was designed with security as a foundational requirement, despite significant hardware constraints. Rather than treating protection mechanisms as an afterthought, the design process integrated security considerations from the outset. To guide this process in a structured and repeatable way, the OWASP Threat Modeling Cheat Sheet [26] was adopted as the primary framework. This reference encourages practitioners to approach systems from an adversarial perspective and to iterate through four key questions: *What are we working on? What can go wrong? What are we going to do about it? and Did we do a good enough job?*

In practice, the device must handle sensitive assets—most notably, login credentials and configuration files for multiple FortiGate firewalls. If compromised, these assets could allow an attacker to map network topology, extract administrative secrets, or gain privileged access to customer infrastructure. Because the device is designed for use in the field, often in environments with minimal physical safeguards, the possibility of theft or unauthorized physical access by a motivated adversary is not merely theoretical but an expected threat scenario.

To identify and categorize these threats, the STRIDE model was applied as a high-level analytical lens. STRIDE encourages systematic consideration of spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege. By examining the device through these categories, specific threat scenarios and mitigations could be mapped more effectively. The following sections of this chapter are organized around the four OWASP threat-modeling questions, presenting the system model, outlining potential threats, describing the mitigations implemented, and evaluating whether those measures are sufficient.

4.2 What Are We Working On?

The device under analysis is a custom tool built on the Waveshare RP2040-Plus module, which provides an RP2040 microcontroller, USB-C programming interface, microSD storage, and general-purpose I/O. A small touchscreen serves as the primary user interface, allowing navigation of menus and selection of configuration profiles. An onboard LED is present but currently functions only as a power indicator. Although the RP2040 exposes other interfaces, such as standard GPIO pins, these are not expected to play a meaningful role in normal operation or in the security model. The USB-C port is used for firmware loading and development and, as with many low-cost microcontrollers, cannot be easily locked down or disabled.

The intended purpose of the device is to streamline deployment of FortiGate firewall configurations in the field. Sensitive data, including administrative credentials and configuration files, will be stored on a removable microSD card and loaded into the firewall via a serial console connection. In the current design, data may be prepared using an external "loader" application that encrypts configuration packages before they are transferred to the device. Alternatively, future firmware updates may allow the device itself to encrypt new or updated configuration files when they are detected on the SD-card in plaintext form. The touchscreen provides menu-driven selection of profiles and operational modes.

A simplified data flow can be described as follows: *User* → *touchscreen* → *firmware* → *SD-card* → *serial interface* → *FortiGate*. When using the external loader, an additional flow precedes this: *User* → *loader application* → *SD-card*. Trust boundaries are present between user input, device firmware, removable storage, and the external serial connection, each representing a potential avenue for attack or misuse.

The design also envisions a PIN-based unlock step during startup. This feature, not yet implemented, would allow multiple credential sets to be stored securely and unlocked dynamically. Including this planned behavior in the system model is important, as it affects how credentials are accessed and how future mitigations may be applied. The device is expected to be operated by helpdesk staff, who may or may not have direct knowledge of the underlying credentials depending on organizational policies. Because the device is intended for use in uncontrolled environments—such as customer sites or during travel—its exposure to opportunistic theft or unauthorized physical access must be assumed in the threat model.

4.3 What Can Go Wrong?

Applying the threat model approach, several attacker scenarios were considered. For example, an adversary might obtain only the removable storage (SD-card) and attempt offline analysis or brute-force decryption of configuration files. A more capable attacker might obtain both the device and the SD-card, enabling inspection of firmware, extraction of salts or other secrets, and manipulation of stored data. Because the device lacks strong physical protections, these scenarios are realistic for field use where the device could be lost, stolen, or left unattended.

To systematically identify threats, the STRIDE framework was applied and tailored to the device context:

Spoofing (authenticity). The device currently has no way to authenticate itself or validate that a given SD-card originates from a trusted source. Without signed or encrypted configurations, a rogue SD-card could be inserted containing a malicious configuration. Simply replicating the hardware would not be sufficient for an attacker; the device's security value is tied to the data on the SD-card, so an attacker would also need access to the original or cloned storage contents. An additional scenario arises after legitimate use: once a helpdesk user unlocks the device and establishes a console session with the firewall, nothing prevents them from physically rerouting the console cable to a laptop and issuing arbitrary commands outside the intended workflow. A robust PIN or user-level access model could reduce this risk, but it remains a potential vector for misuse.

Tampering (integrity). If configuration files on the SD-card are not encrypted or signed, an attacker could alter them to open backdoors, enable unwanted services, or otherwise weaken firewall security. Firmware tampering is also possible via the USB-C port; modified firmware could exfiltrate credentials, bypass the PIN entry process, or alter logs. Because the RP2040 platform lacks secure boot, no built-in integrity check can prevent this.

Repudiation (non-repudiation). Currently there is no implemented audit trail of which configurations were deployed, when, or by which user. Planned logging features could address this, but without protection those log files could themselves be deleted or altered by anyone with access to the SD-card. A future design might link logs to individual PINs or sign them to ensure integrity.

Information Disclosure (confidentiality). Sensitive data exists at multiple points: configuration files on the SD-card, salts in firmware, and decrypted credentials temporarily in RAM. Without encryption, these assets can be extracted directly. Even with encryption, physical access allows an attacker to dump firmware and brute-force the PIN. Debug interfaces (USB-C, SWD/JTAG) further increase exposure, and during loading via a PC the plaintext configuration may briefly reside on another system. Console output on the touchscreen could also leak partial credentials or configurations if not carefully managed.

Denial of Service (availability). The device is not strictly required to configure a firewall, but in workflows where helpdesk staff depend on it, loss of availability would cause operational delays. An attacker could remove or damage the SD-card, overwrite critical configuration files, or load invalid firmware that prevents the device from operating. Malformed configurations could also cause software lock-ups during deployment.

Elevation of Privilege (authorization). Although the device itself does not currently enforce privilege separation, a future multi-PIN design could introduce user-specific access levels. In the current design, brute-forcing a PIN and extracting the firmware salt could allow access to read-only credentials, which in turn could reveal a firewall's serial number and be combined with a second-phase (see Section 4.4, Constraining Credential Use) key to obtain full administrative credentials. Firmware tampering could also bypass access controls and escalate privileges.

4.4 What Are We Going To Do About It?

The threat modeling process identified multiple attack surfaces and potential misuse scenarios. Mitigations focus on three major areas: protecting stored secrets, constraining the use of those secrets, and ensuring integrity and accountability of device operations. Some measures are already implemented in the prototype, while others remain conceptual or partially implemented as future enhancements.

Protecting Stored Secrets Sensitive data such as firewall credentials and configuration files are stored on a removable microSD card. To prevent direct plaintext access, the design introduces a PIN-based access control system. At startup, the user must enter a 6-digit PIN, which is combined with a device-specific salt stored in firmware to derive an AES-128 key. All credentials and configurations on the SD-card are encrypted using this key, ensuring that offline analysis of the card alone yields only ciphertext.

Because the device currently lacks hardware secure boot or readout protection, an attacker with physical access could still dump the firmware and extract any secrets stored directly in flash memory, such as the device-specific salt. In the current design the PIN is a simple 6-digit numeric value, so brute-forcing remains a realistic risk—especially for offline guessing if ciphertext and the salt are obtained—while online mitigations (e.g., attempt limits and back-off) only slow guessing rather than prevent it outright. However, without the correct salt, the encrypted data on the SD-card remains computationally infeasible to decrypt. This layered approach still raises the bar for offline attacks, but it does not eliminate the fundamental weakness of storing sensitive material in firmware.

Future iterations can incorporate a secure element to protect cryptographic material. A secure element is a dedicated hardware component designed to store secrets and perform cryptographic operations without ever exposing raw keys to the main microcontroller. By storing the salt or a master key within such a component, an attacker dumping the firmware would not be able to extract these values directly, even with full physical access to the board.

One example of this approach is the Adafruit ATECC608A breakout board[1], shown in Figure 4.1.

[1] Adafruit ATECC608A Secure Element Breakout, available at https://www.adafruit.com/product/4314

This breakout uses I^2C to interface with a microcontroller and provides protected storage for keys along with hardware support for AES-128, ECDH, ECDSA signing, random number generation, and hashing. Once provisioned, the chip can perform challenge–response operations without ever revealing the stored key. Breakout boards like this are inexpensive, easy to integrate, and compatible with 3.3 V or 5 V logic, making them a practical enhancement for future revisions of the device.

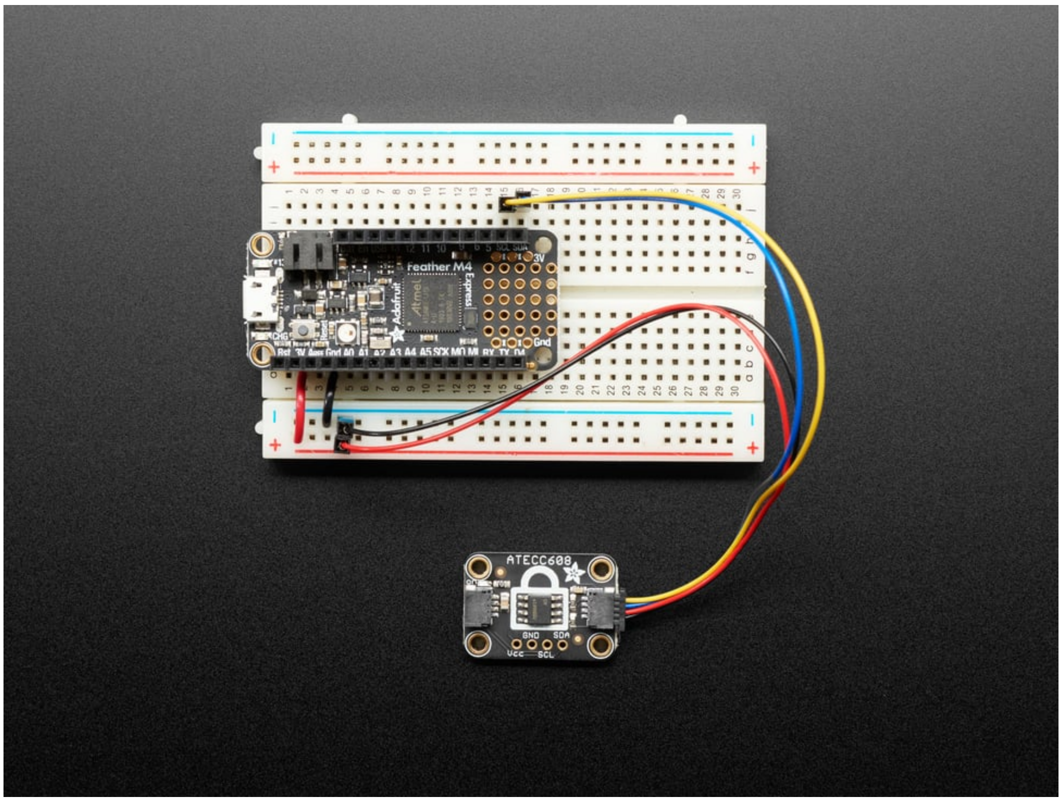


Figure 4.1: Adafruit ATECC608A secure element breakout board connected via I^2C (credit: Adafruit).

Constraining Credential Use To reduce the impact of credential compromise, all firewall accounts used by the device are restricted to console-only access. Neither SSH nor web interfaces are enabled for these accounts. This means that even if credentials are brute-forced, leaked, or extracted from the device, they cannot be used remotely and require physical access to the target firewall. This restriction directly narrows the attack surface exposed by the device.

An additional conceptual enhancement under exploration is a two-phase unlocking process. In this model, the device first authenticates with a low-privilege read-only account to retrieve a non-sensitive identifier such as the firewall's serial number. This identifier, combined with the user's PIN, would then derive a second key used to decrypt the administrator-level credentials. While this adds complexity, it would tie high-privilege credentials to a specific firewall, further limiting misuse in case the device or SD-card is stolen.

To further reduce misuse risk, the device will be designed to minimize the time spent logged in with administrative privileges. When idle or browsing available configurations, the device remains logged in with a low-privilege, read-only account. Administrative credentials are used only at the moment a configuration is deployed, and the device immediately logs out once the operation is complete. This limits the exposure window and prevents users from repurposing the active console session for unauthorized manual commands via another device. By tightly controlling session duration and separating read/write functionality, the system follows a least-privilege model even during normal use.

Ensuring Integrity of Configurations Unsigned or unverified configuration files present a risk of tampering or malicious injection. The envisioned design includes cryptographic signatures for configuration packages. Before applying any configuration, the device would verify its signature against a trusted public key embedded in firmware. This prevents attackers from inserting rogue configuration files even if they gain access to the SD-card. Implementation of signing is still under consideration due to time constraints and the need to balance complexity with available hardware resources.

Maintaining Auditability To address repudiation risks, a logging feature is planned that records which configuration was deployed, when, and under which PIN. These logs would be stored on the SD-card and encrypted or signed to prevent alteration. If a log file is missing or invalid, the device could refuse to operate, providing a basic safeguard against tampering or log deletion. Although logging is not yet implemented in the current prototype, the design anticipates this feature to support operational accountability.

Future Enhancements and Limitations Additional measures under consideration include binding configurations to specific hardware identifiers (e.g., the firewall's serial number) to mitigate theft scenarios, rate-limiting PIN attempts to slow brute-force attacks, and adding tamper-detection circuitry. However, due to the RP2040 platform's lack of secure boot or readout protection, some attack paths—such as extracting the firmware salt—cannot be completely eliminated without hardware changes. These limitations are acknowledged in the design and inform recommendations for future iterations.

4.5 Did We Do A Good Enough Job?

The device's security was evaluated against the identified threats using the STRIDE framework and the OWASP threat modeling process. The analysis shows that, within the constraints of the hardware platform, the implemented design significantly reduces the most immediate risks. PIN-derived encryption protects configuration data at rest, console-only firewall accounts reduce remote attack surfaces, and careful handling of trust boundaries limits casual misuse.

However, residual risks remain. The RP2040 platform lacks hardware-enforced secure boot and true readout protection, meaning that a determined adversary with full physical access could still extract the firmware and attempt offline brute-force attacks against encrypted data. Additionally, because configuration signing and detailed logging are not yet implemented, integrity verification and non-repudiation remain incomplete.

Future improvements have been identified as part of this threat modeling exercise. Integrating a secure element such as the ATECC608A would provide a hardware-backed root of trust, protecting the salt or master key from firmware extraction. Tamper detection features, such as case-open sensors or erased-on-tamper keys, could raise the cost of physical attacks. Configuration signing and audit logging, once implemented, would strengthen defenses against unauthorized modification and enable accountability for deployments. Formal penetration testing by a third party would further validate the security posture.

Limitations and Future Work At this stage of development, the prototype embodies the layered mitigations discussed above but remains a work in progress. Some features—such as multi-user PIN support, configuration signing, and secure-element integration—are still in the design phase and may not be fully operational by the time of the defense demonstration. This is consistent with the iterative nature of threat modeling: the process has informed design decisions and prioritized enhancements even if they are not yet implemented.

Despite these limitations, the current build envisions meaningful improvements over an unsecured deployment workflow. The design targets eliminating plaintext storage of sensitive data, constraining credential use to the physical console, and introducing signed configuration verification and structured audit logging. These measures are documented and prioritized in this chapter but are not yet implemented. The near-term roadmap shifts from adding a discrete secure element to migrating to Raspberry Pi's Pico 2 W (RP2350), which became available mid-way through this work and provides a built-in hardware root of trust.

Its security architecture directly addresses several of the limitations noted in this chapter: signed and attested boot with a key fingerprint stored in 8 kB antifuse OTP; optional encrypted boot with keys held in OTP; a hardware TRNG; a SHA-256 accelerator; glitch-fault detectors; and Arm TrustZone-M for isolating sensitive code and data. Together, these features provide per-device secret anchoring and reduce reliance on firmware-resident material. Migration would focus on the boot chain, OTP key enrollment, and compartmentalizing sensitive routines, while console, storage, and UI paths can remain largely unchanged—and the platform natively supports the signed-bundle verification and audit logging planned above.

Chapter 5: Evaluation and Use Cases

5.1 Evaluation Setup

This section documents the bench environment used to exercise the prototype UI, SD handling, and UART console I/O before attempting full configuration dispatch. The setup mirrors the hardware chain and lab context described earlier in the build chapters and appendix (see Appendix Table A.3).

Bench topology. A FortiGate 60E on the lab bench is cabled from its RJ-45 `CONSOLE` port through the bundled RJ-45→DE-9 rollover cable, a short DE-9 male↔male null-modem adapter (pins 2↔3 crossed), and an HX-004 MAX3232 level shifter into the RP2040-Plus UART0. Power for **FortressFlash** is provided by USB-C (wall adapter, power bank, or development workstation); the firewall is powered normally. The RP2040-Plus drives the Waveshare 2.8 ” LCD/touch carrier and mounts a removable microSD card for configuration bundles.

Network topology. The test FortiGate 60E `WAN` is cabled into a production Netgear M4100-26G switch on the LAN side of a production FortiGate 60D. The test firewall's `LAN` serves a small lab segment at `192.168.1.0/24` with several fixtures for policy/routing checks (see Figure 5.1):

- HP Windows 11 workstation (user validation / GUI reachability).
- Buffalo LinkStation NAS (fileshare access (FTP/SMB), ACL tests).
- PIX-LINK 2.4 GHz access point (wireless client path, DHCP/DNS).

Note: The test LAN is isolated from production; no bridging is enabled. Upstream access, when required, traverses the production path (double NAT may occur).

Development host & tooling. Most firmware work was performed on a Raspberry Pi 5 (16 GB), connected to the production LAN via the Netgear switch. For console sanity checks, the Pi uses a USB→RS-232 adapter directly into the FortiGate 60E's console. The Pi connects to **FortressFlash** over USB-C; deployment is via VS Code with the MicroPico extension (MicroPython workflow).

Firmware under test. MicroPython firmware corresponds to the “core subsystems”: stable console login with bounded retry logic (implemented), scrolling serial log (implemented), main menu touch

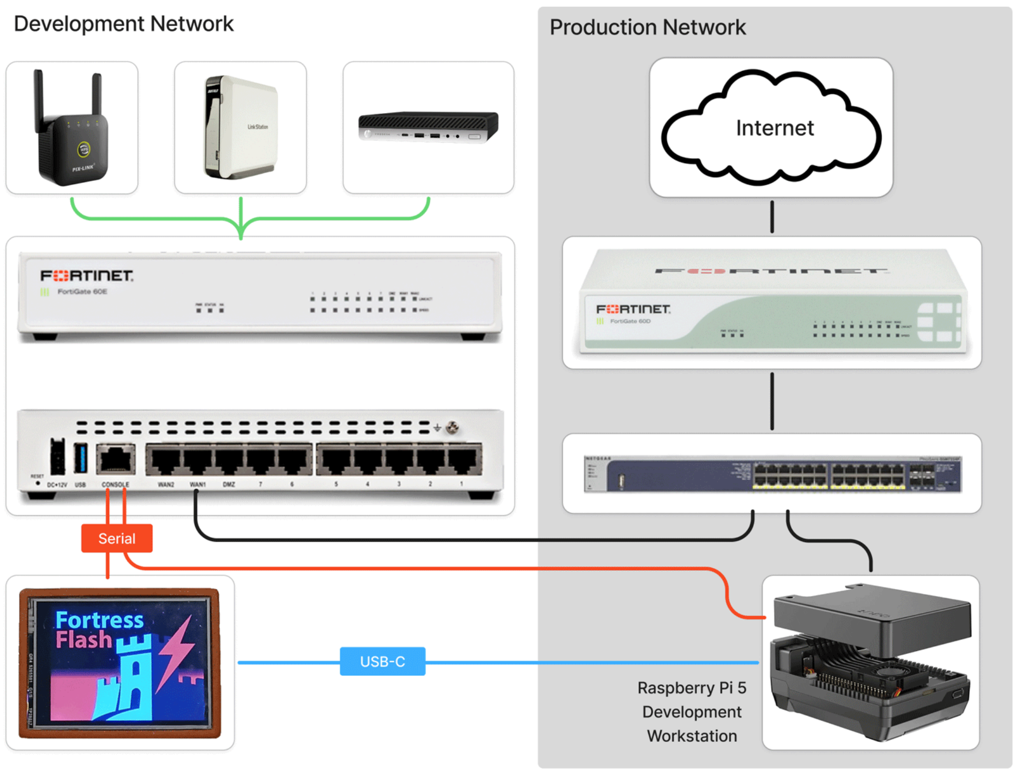


Figure 5.1: Evaluation bench topology used in Section 5.1. The FortiGate 60E under test anchors the *Development Network* (PIX-LINK AP, Buffalo LinkStation NAS, HP Win 11) while its WAN uplinks through the *Production Network* (Netgear M4100 → FG-60D → Internet). **FortressFlash** connects to the 60E via the serial console (red), and the Raspberry Pi 5 development host connects over USB-C for firmware deployment (blue) and uses RS-232 for console sanity checks (red).

UI (implemented), and SD profile enumeration (implemented). A prompt-paced dispatcher is in progress. Error dialogs for "no SD" / "no profiles" are present; PIN unlock and encrypted logging are planned.

Media layout. For the purposes of evaluation, the removable microSD card is freshly formatted with FAT32 and organized under `/configs/{entity}/{site}/{device}/`. Files are plaintext FortiOS CLI bundles with `.conf` extension; a future loader application will package signed/encrypted manifests, but the present evaluation uses unsigned test bundles to exercise discovery, menuing, and line pacing.

Assumptions. Console is used with the vendor's default serial parameters (9600 baud, 8N1, no flow control); the device aborts if no FortiGate banner is detected to protect against mis-cabling. Tests were run against a FortiGate 60E running FortiOS v7.2.4 build 1396. No network services (API/TCP-IP) are used in this chapter.

5.2 Preliminary Observations and Results

The following summarizes the bench outcomes as of submission and maps them to the Success Criteria stated in §1.3. Where end-to-end timing is not yet available, partial timings and functional observations are reported.

What works now.

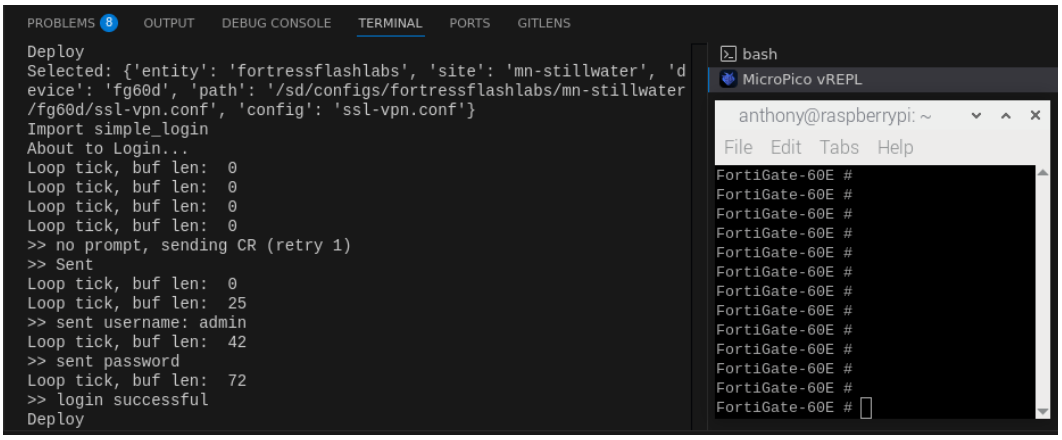
- **Boot → UI ready:** Device reliably boots to the main menu and renders the scrolling serial log without visual artifacts on the 320×240 LCD. Touch input is stable after two-point calibration.
- **SD detection and mount:** Cards mount at `/sd`. "No SD" dialog behaves as expected.
- **Console session:** UART0 consistently establishes a console session on FortiGate 60E through the RJ-45→DE-9→null-modem→MAX3232 chain; login succeeds and CLI output is captured in the scroll buffer (Figure 5.2).
- **Menu navigation:** Profile listing renders and paginates for long lists (UTF-8 labels handled).

Login validation. To validate reliability, two sets of ten login attempts were executed on the FortiGate 60E, with each session independently verified via a USB–RS232 adapter and `screen` to confirm arrival at the # prompt, followed by manual logout. In the first set (n=10), all ten logins succeeded; the initial firmware waited the full two-second timeout before issuing a carriage return, thereby exercising the retry logic. After updating the routine to send an immediate initial carriage return, a second set of ten logins again achieved 10/10 success. Across twenty consecutive trials, the prototype achieved a 100% success rate under bench conditions, completing all sessions end-to-end without resets or crashes.

Instrumentation also captured consistent buffer lengths at each stage of the exchange: 25 bytes after the initial carriage return, 42 bytes after username submission, and 72 bytes after password entry. These stable values confirm deterministic session behavior and provide a baseline for future audit-logging features.

In progress / known gaps.

- **Dispatcher:** Line-by-line transmit with prompt-aware pacing is implemented but not yet validated end-to-end against a full baseline bundle; early-abort on detect-error strings and per-command timeouts are being finalized.
- **Logging at rest:** On-device encrypted log archival is planned; current build provides live screen scroll only (no encrypted `deploy.log` yet).



```
Deploy
Selected: {'entity': 'fortressflashlabs', 'site': 'mn-stillwater', 'd
evice': 'fg60d', 'path': '/sd/configs/fortressflashlabs/mn-stillwater
/fg60d/ssl-vpn.conf', 'config': 'ssl-vpn.conf'}
Import simple_login
About to Login...
Loop tick, buf len:  0
Loop tick, buf len:  0
Loop tick, buf len:  0
Loop tick, buf len:  0
>> no prompt, sending CR (retry 1)
>> Sent
Loop tick, buf len:  0
Loop tick, buf len:  25
>> sent username: admin
Loop tick, buf len:  42
>> sent password
Loop tick, buf len:  72
>> login successful
Deploy
```

```
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
FortiGate-60E #
```

Figure 5.2: Successful FortiGate login demonstration. Left: FortressFlash REPL output showing username/password dispatch and login confirmation. Right: independent `screen` session over USB→RS-232 verifying the FortiGate CLI prompt (#).

- **PIN unlock & file verification:** PIN-gated unlock, signed bundle verification, and device-SN checks are specified but not yet implemented in the prototype.

Matrix vs. Success Criteria (as submitted).

Criterion (from §1.3)	Status	Notes
Boots to UI, mounts SD, lists profiles, exchanges CLI I/O over UART	**Met**	Console and UI verified on FG-60E.
Reliable console login with bounded retries	**Met**	20/20 logins successful; retry logic validated; no resets or crashes.
End-to-end configuration push *or* verified prompt-paced dispatch with logging	**Partial**	Dispatcher present; e2e push not yet executed; logging live-only.
Time-to-first-push < 120 s (when feasible)	**N/A**	Not measured pending e2e push.

This satisfies the first and second success criteria and partially satisfies the third; the fourth is deferred until the full dispatch path is validated.

Interpretation. The lab outcomes indicate the hardware chain, UI, storage, and serial I/O are stable and usable by entry-level staff (cf. Scenario A/B), while security and audit features are correctly staged for a Pico 2 W migration (Chapter 4). This is consistent with the staged development and “Phase 4 Laboratory Evaluation” described in the methodology.

5.3 Deployment Scenario

FortressFlash is designed for the *real* world of managed-service providers (MSPs), where the only on-site resource may be a level-1 help-desk technician armed with a label printer and a mild sense of panic. Two field scenarios capture its intended use:

Scenario A: Rapid Configuration Restore

Context. A previously deployed FortiGate at *Client Alpha* has lost its running configuration after a power spike. Workstations are offline, VLANs are collapsed, and the client's SLA clock is ticking.

Actors. A single L1 technician is dispatched; all senior engineers remain remote.

Procedure.

1. **Power** the firewall as usual. FortressFlash boots from an external USB power bank (or the FortiGate's rear USB-A port).
2. **Connect** the RJ-45 console lead to the FortiGate's `CONSOLE` port, and the DE-9 to the FortressFlash.[1].
3. **Authenticate.** The technician enters a six-digit PIN; three consecutive failures invoke a 30-second back-off.
4. **Navigate Menu.** The main menu appears and displays available options and settings, `Deploy Config` is selected.
5. **Select profile.** FortressFlash presents a hierarchical file picker to select an entity (client), site (location), and device. A list of encrypted `.conf` bundles preloaded on the microSD card is shown for the specific device. The tech taps `Client_Alpha` → `HQ` → `fw01` → `master_settings.conf`.
6. **Push configuration.** FortressFlash logs into the device, loads the configuration, dispatches it to the device, and reloads.
7. **Verify.** FortressFlash tail-follows the boot log until the "`FGT...login:`" banner reappears, then prints a green `PASS`. The tech launches the GUI in a browser; traffic to critical hosts (e.g., DC01, AP01) is restored.

[1] A null-modem crossover is permanently mated to FortressFlash.

Success criteria. End-to-end recovery completes in under five minutes, a 90% reduction versus re-building by hand. An encrypted session transcript is archived to the SD card for compliance auditing.

Scenario B: Zero-Touch First-Boot

Context. A brand-new FortiGate is unboxed at *Client Bravo*. It must ship with the MSP's hardened baseline—DNS filtering, IDS sensors, and policy templates—before any internet access is allowed.

Actors. The same lone L1 technician performs the install.

Procedure.

1. **Power and link** the FortiGate; connect FortressFlash as in Scenario A.
2. **Authenticate** with PIN, select `ZTP` from the menu, and choose the `MSP_Baseline` profile (common to all fresh units).
3. **Personalize.** A guided prompt asks for site-specific values: `Hostname`, `LAN_CIDR`, and the first admin email. FortressFlash merges these into the baseline JSON before deployment.
4. **Apply configuration** and allow the FortiGate to reboot.
5. **Verify.** FortressFlash again verifies the "`FGT...login:`" banner, and prints a green `PASS`. Tech launches the GUI in a browser and verifies workstation and device connectivity.

Success criteria. The firewall is policy-enforced and cloud-manageable within one visit, sparing the MSP a second truck roll. Interactive time for the technician is limited to PIN entry and three site fields; the remainder is fully automated. These scenarios illustrate how FortressFlash converts what was once a two-hour, error-prone CLI session into a guided, five-minute workflow—putting zero-touch deployment within reach of entry-level staff while preserving the MSP's security standards.

> **Note on current prototype:** as of submission, the device reliably establishes a console session and presents the guided workflow; the end-to-end configuration push is under active integration and is reported in Section 5.2 as preliminary results.

Chapter 6: Prototype to Product

6.1 Brand Identity and Positioning

While the *FortiProgrammer* prototype originated as a hands-on experiment with microcontrollers and serial interfaces, it quickly evolved into something more: a real-world tool with tangible market potential. As development progressed, the scope expanded beyond simple configuration delivery into an integrated solution emphasizing security, ease-of-use, and field readiness. Recognizing these characteristics as desirable in the commercial IT services market—particularly for managed service providers (MSPs)—the project adopted a parallel track to explore its viability as a deployable product. This chapter outlines the broader innovation strategy that emerged as the project matured, including brand identity, user experience, product-market fit, and initial go-to-market considerations.

Name and intent. A product with real-world aspirations demands not only technical merit but also clear communication of purpose and trust. To that end, the solution was rebranded from the internal codename *FortiProgrammer* to **FortressFlash**, a name intended to evoke both cybersecurity strength and rapid field deployment. **Fortress** conveys protection and resilience—a safe refuge from external threats, symbolizing a firewall defending a private network. ***Flash*** emphasizes not only speed and responsiveness but also the act of updating firmware and configurations. Together, **FortressFlash** provides a name that visually reflects its purpose while resonating with the imagery and expectations of its operators.

What it is, for whom. FortressFlash is a pocketable, PIN-guarded, SD-backed configuration appliance that lets L1 technicians deploy a vetted FortiGate baseline in minutes—no laptop required. Although the prototype was built and tested primarily with FortiGate firewalls, the underlying architecture is not vendor-specific; the same model can extend to other serial-managed platforms. It targets small-to-mid MSPs that need repeatable, low-error, on-site turn-ups when senior engineers are unavailable, while remaining compatible with the deeper workflows of larger providers. For a complete technical description of FortressFlash's hardware and firmware, see Chapter 2, Hardware Design and Chapter 3, Software Design.

Visual identity. The logo, a fortified castle struck by a lightning bolt, visually represents the concept of “secure infrastructure deployed with speed” (see Figure 6.1). The primary palette echoes Luxembourg’s flag: blue `#0C9DDE` and red `#EA141D`, paired with neutral dark grays for contrast. The castle motif also nods to Luxembourg’s historic reputation as a “fortress city,” a fitting reference given the author’s dual U.S.–Luxembourg citizenship and family ties to Wahl, Luxembourg. The brand identity was also extended into physical artifacts such as a cap and polo shirt (see Appendix A.6, Brand Assets), underscoring that the project was presented not only as a functional prototype but as a recognizable product. The cap, embroidered with the phrase “FortressFlash Labs,” further hints at the project’s evolution towards an innovation space, later referenced in Section 7.4 (LoRa Exploration).

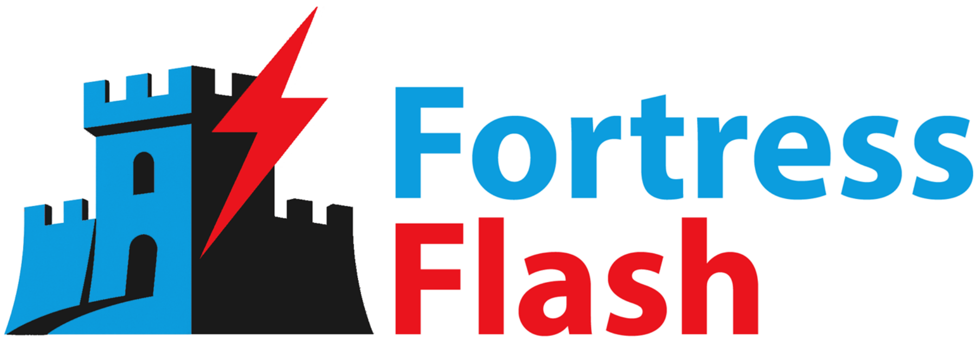


Figure 6.1: FortressFlash brand identity mark. The castle-and-lightning motif symbolizes security and speed, reflecting the core values of the device’s design.

Digital presence. In support of the brand, the domain `FortressFlash.lu` was launched as a countdown landing page, offering a polished entry point for demonstration and future updates. To further strengthen the project’s identity and ensure broader accessibility, `FortressFlash.com` was also secured for global presence and is currently configured to redirect to the primary Luxembourg-based site. This dual-domain setup helps maintain a local innovation focus while remaining approachable to an international audience.

Naming hygiene. This work is an independent academic project and is *not* affiliated with or endorsed by Fortinet, Inc. “FortiGate” and other vendor marks are the property of their respective owners.[1]

6.2 Business Model Canvas

To explore the product’s potential beyond academia, a business model canvas was constructed (see Figure 6.2) to provide a conceptual starting point should the project be incubated or spun out.

[1] A general trademark notice also appears in Chapter 1, Introduction; it is reiterated here for completeness in the branding context.

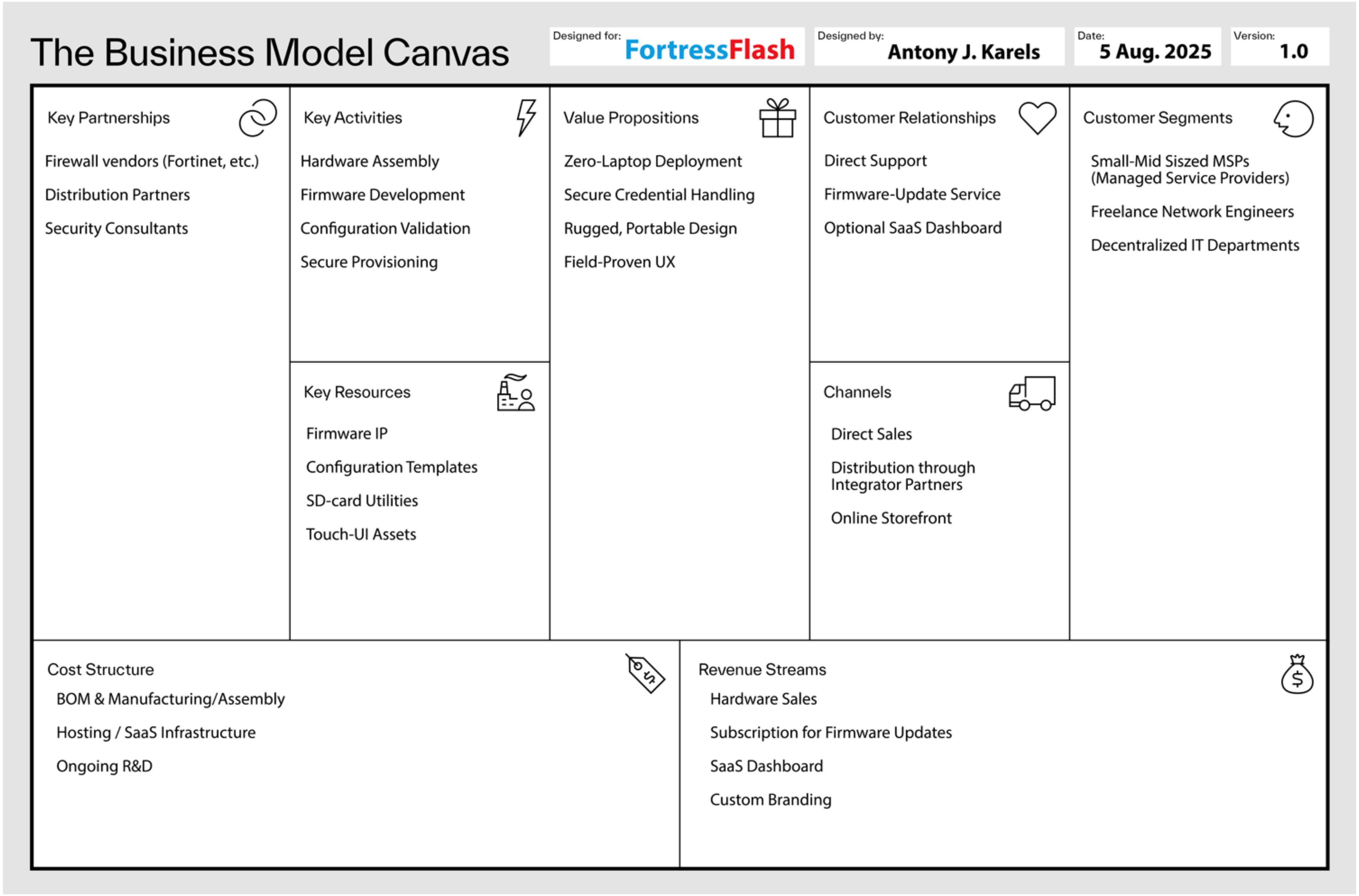


Figure 6.2: Business Model Canvas for FortressFlash, adapted from [27]. Licensed under CC BY-SA 4.0.

6.3 SWOT Analysis

To complement the Business Model Canvas, Table 6.1 summarizes FortressFlash's strengths, weaknesses, opportunities, and threats. The analysis points to a single operational priority: *firmware polish is the keystone*. A lean BOM and a crisp "push-button firewall" story already resonate with MSPs; real adoption hinges on shipping battle-tested firmware. Near-term sprints should therefore center on design-partner pilots and hardening secure boot, credential vaulting, and UX flows.

Strengths	**Weaknesses**
• *One-touch deployment*: even a Level-1 tech can load a vetted configuration.	• Firmware still MVP-grade; secure-boot and Pico 2 W port remain TODOs.
• Ultra-low BOM (˜$25) and no laptop required; $249-$299 retail pays for itself in $\sim$1 roll-out.	• Market pull unproven until one killer workflow is production-ready.
• Built on ubiquitous RP2040 hardware—easy sourcing and customisation.	• Current UX feels "maker-ish"; needs polish for field techs.
• Founder has 8 + years MSP/ISP experience and Fortinet certs.	• Single-product focus; no diversification yet.
Opportunities	**Threats**
• Niche global MSP market hungry for FortiGate time-savers.	• Low barrier to entry—clones could out-iterate on software.
• Adjacent devices (switches, routers) and add-ons (Wireless, API-mode, LLM intent).	• Compromised device with weak SD-card crypto would hurt credibility.
• Channel sales via computer retailers and integrators.	• Vendor indifference; lack of Fortinet endorsement may slow uptake.
• Regulatory tailwinds (PCI DSS 4.0, EU NIS2) reward built-in credential hygiene.	• Microcontroller supply volatility always a lurking risk.

Table 6.1: SWOT analysis for the FortressFlash concept.

6.4 Go-To-Market Strategy

Market size and target customers. A practical addressable market for FortressFlash is the managed-service-provider (MSP) channel. Industry estimates place the number of companies participating in the United States MSP market at roughly 40,000, with a much larger global pool (estimates vary by source). This fragmentation indicates low concentration and numerous small-to-medium buyers that can benefit from a simple, auditable tool for first-boot provisioning and recovery. The overall managed-services market is also large and growing—measured in the hundreds of billions annually—providing a favorable backdrop for a focused, vertical device-plus-SaaS offering.[2]

Value proposition and differentiation. FortressFlash reduces on-site deployment risk for junior technicians by delivering signed, PIN-guarded, SD-backed baseline configurations with an auditable console log—no laptop required. This security-first posture (signed boot, OTP-sealed keys, hardware RNG) provides a compliance-credible alternative to ad-hoc scripting, while the console-first design generalises across vendors that expose standard serial workflows. In bench testing, the same rollover cable and null-modem chain used for FortiGate successfully interfaced with a Palo Alto PA-200, demonstrating cross-vendor applicability without hardware changes (see Figure 7.1). In GTM terms, this supports a beachhead in Fortinet-heavy MSPs with adjacent expansion to other firewall ecosystems.

Channels and launch motion. Initial reach relies on direct sales to regional MSPs and integrators, supported by a public storefront for small orders. The near-term motion prioritises design-partner pilots (5–10 MSPs) to validate workflows, documentation, and subscription attach; success is measured by time-to-baseline, first-pass success rate, and the share of deployments executed by L1 staff. Community presence (vendor-agnostic how-to guides, conference demos) feeds top-of-funnel interest, while a light channel program (logo/packaging customisation, volume pricing) captures larger fleets. With target segments, messaging, and motion defined, Section 6.5 details the corresponding pricing and monetization model.

6.5 Pricing and monetization.

Pricing roadmap. Although the bill of materials is lean ($\approx$ \$25), the value proposition is in the firmware: signed boot, OTP-sealed credentials, audited console workflows, and a field-tested UX that reduces truck-roll risk.[3] Production pricing therefore targets a professional MSRP in the \$249–\$299 range—positioned well below enterprise out-of-band (OOB) gear yet clearly distinct from hobbyist

[2]Representative industry sources include market analyses and MSP surveys; see Mordor Intelligence [28], MSPAlliance summaries [29], and Infrascale reporting [30] for recent estimates.

[3]All amounts are stated in USD given the U.S.-based BOM and primary channel.

adapters. Recurring revenue comes from optional device subscriptions ($9–$12 per month) that enable over-the-air (OTA) updates, audit log retention, and fleet dashboards; when FortressFlash is embedded per firewall, a mandatory SaaS license captures ongoing value. This staged approach preserves Lean discipline: keep the first sale friction-free, prove value quickly, and layer subscriptions once FortressFlash becomes mission-critical in an MSP's workflow. The stage-gated framework in Table 6.2 illustrates this progression.

Table 6.2: Stage-gated pricing framework for FortressFlash

Stage	**Buy-in Pattern**	**Monetization Lever**	**Indicative Price**	**Rationale**
MVP Launch	1 unit ≈ 1 field tech	Hardware margin only	$249 per device	Priced as a professional tool; pays for itself by avoiding a single truck roll; validates core value.
Early Adoption	1 unit ≈ 1 branch/van	HW + optional update/support plan	$249 device + $9 / mo	Begins recurring revenue while retaining a low operational barrier.
Embedded Mode	1 unit ≈ 1 firewall (permanent)	HW + mandatory SaaS licence	$299 device + $12 / mo	Repositions as an appliance; higher ARPU, stickier contracts.

Add-on revenue streams. Beyond the staged core pricing, FortressFlash can be extended through optional upsell levers that add margin, increase stickiness, and differentiate SKUs for niche buyers. Not every customer requires the full security stack from day one, so a split-tier model supports a lower-cost "Standard" unit while compliance-heavy MSPs can opt into enhanced firmware features such as secure SD encryption. A loader application discussed in Section 3.2 (Device Provisioning App.) is included for all customers; Secure-tier features (profile encryption and policy enforcement) are activated by license. Additional levers include fleet-oriented custom branding, remote-site telemetry modules, and developer-focused API licenses. These options—summarized in Table 6.3—extend lifetime value while positioning the hardware as a flexible platform rather than a commoditized adapter.

Table 6.3: Optional and future upsell levers

Add-on / Tier	**Target Users**	**Price Idea**	**Benefit**
Secure SD encryption firmware (with pre-provisioned cards)	Compliance-heavy MSPs	+$50 over Standard SKU, or $19/card	Ensures configs at rest are encrypted; positions "Secure" tier without raising BOM.
Custom branding	Large MSP fleets	$1 k setup + $10/unit	Strengthens MSP brand; locks in fleet orders.
LoRa/BLE telemetry module	Remote sites	+$49	Extends range; creates a "Pro" SKU.
API access licence	DevOps teams	$99 / yr	Enables CI/CD-style mass roll-outs.

6.6 Strategic Insights

What began as a field experiment in embedded development has evolved into something more: a thesis project with credible product potential. At this stage, FortressFlash shows how a technical prototype—paired with clear product identity and market context—can begin to bridge the gap from lab bench to business case. The aim here is not to declare a launch, but to frame a plausible path from prototype to product and surface the assumptions that merit further validation.

The work also reflects the learning ethos of the CYBERUS program. FortressFlash required designing and testing secure embedded systems, analyzing vulnerabilities, and applying countermeasures in real hardware—while exercising horizontal skills such as problem solving, communication, and interdisciplinary coordination. Notably, the program's inclusion of an entrepreneurship course in each of the three in-classroom semesters signals its intent: to connect research and innovation practice with venture-minded execution. In that sense, the project operates as both device and demonstration—aligning technical depth with R&I/D practice and linking low-level engineering choices to higher-level strategy.

More broadly, the chapter suggests that security innovation gains traction when usability, trust, and organizational fit accompany cryptographic rigor. A project that starts with microcontrollers and UART workflows can, with intentional framing, evolve into a coherent product story for a specific channel. Future exploration may focus on multi-vendor extensions and feature enhancements that strengthen the offering if it proceeds in that direction, keeping the emphasis on practical value as much as technical merit. These observations also frame the next chapter, which looks forward to potential extensions and future research directions.

Chapter 7: Future Work

7.1 Expansion to Other Vendors or Platforms

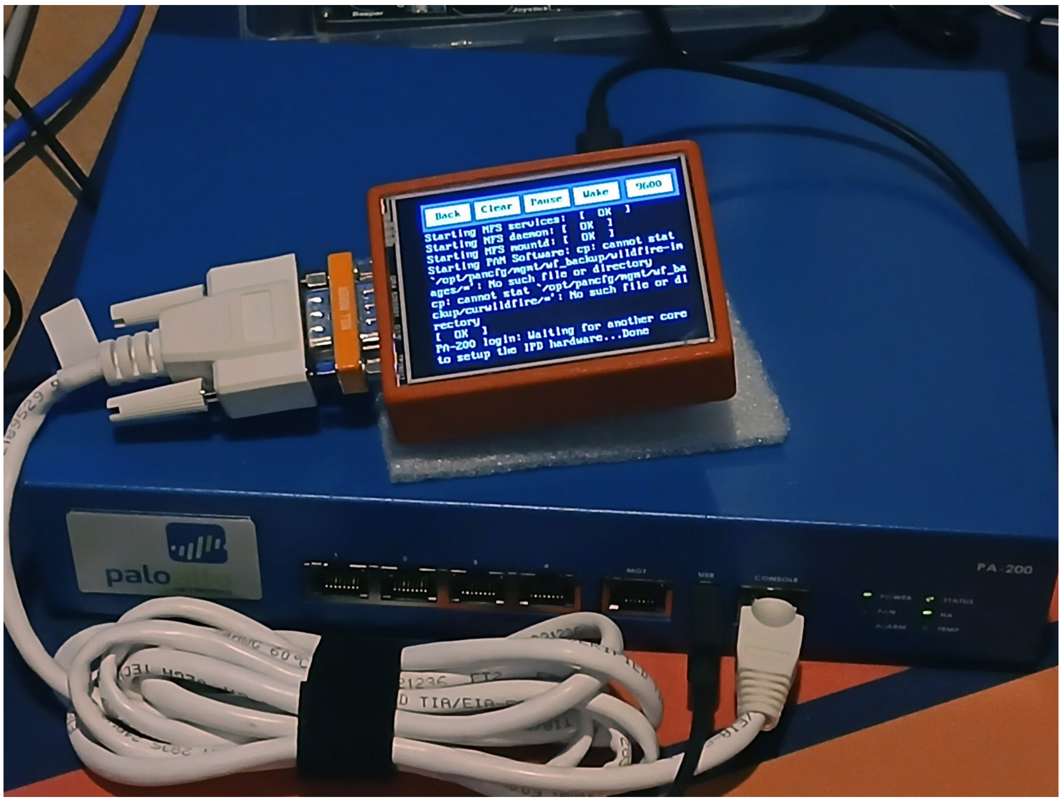
Expanding FortressFlash support to additional platforms is a natural and necessary step toward broader adoption. While FortiGate firewalls are widely used, the current prototype primarily serves market segments such as MSPs, multi-site enterprises, and independent technical support contractors. Broader applicability can be achieved by extending compatibility to other major vendors—such as Cisco, Juniper, and Netgear—whose devices also feature serial-based boot and recovery workflows. Beyond firewalls, many switches, routers, modems, and embedded appliances could benefit from scripted, automated provisioning or recovery capabilities.

Even prosumer-grade hardware, such as the Linksys WRT32X and WRT3200ACM, often includes internal serial headers that can be used for low-level access during firmware recovery or de-bricking procedures (see Figure A.12). Supporting such devices would require the ability to interface with a variety of connector types, voltage levels, and pinouts. A commercial version of FortressFlash could therefore include swappable or modular connector kits, or offer vendor-specific adapters as add-ons. To further extend its reach, the platform could support custom user-defined modules, enabling advanced users to tailor workflows for specialized or uncommon hardware. Notably, many recovery processes—such as firmware restoration via TFTP—also require network connectivity, reinforcing the importance of later-stage TCP/IP enhancements already proposed for API-mode operation (see Section 7.5).

To validate the multi-vendor potential beyond Fortinet equipment, a Palo Alto Networks PA-200 firewall was unexpectedly acquired during fieldwork and tested.[1] Using the identical RJ-45 rollover console cable, null-modem adapter, and MAX3232 level shifter chain employed with the FortiGate 60E, FortressFlash successfully captured the PA-200's boot log and presented its console output without modification. This confirmed that the device's serial workflow is vendor-agnostic at the electrical and protocol layer, and that only firmware-side adjustments (command templates, prompt handling) would be needed to extend support. Figure 7.1 shows the PA-200 connected to FortressFlash during bench testing.

Status:* *Short-term.

[1] In a stroke of serendipity, the unit was discovered at a local Goodwill store on half-price day for $7.50.



```
[  OK  ]
Starting NFS services: [  OK  ]
Starting NFS daemon: [  OK  ]
Starting NFS mountd: [  OK  ]
Starting PAN Software: cp: cannot stat
`/opt/pancfg/mgmt/wf_backup/wildfire-im
ages/*': No such file or directory
cp: cannot stat `/opt/pancfg/mgmt/wf_ba
ckup/curwildfire/*': No such file or di
rectory
[  OK  ]
PA-200 login: Waiting for another core
to setup the IPD hardware...Done
```

Figure 7.1: FortressFlash connected to a Palo Alto PA-200 firewallFortressFlash connected to a Palo Alto PA-200 firewall during interoperability testing. The device successfully displayed the PA-200 console boot sequence using the same RJ-45 rollover cable and null-modem chain as with the FortiGate 60E, confirming cross-vendor compatibility at the console layer.

7.2 Out-of-Band Management Gateway

One of the earliest conceptual extensions of the FortressFlash prototype was to operate not only as a local deployment tool, but also as a remote access appliance. By embedding Wi-Fi connectivity and bridging the console UART into a secure web service, the device could function as an *Out-of-Band Management Gateway*. In this role, administrators would be able to access the firewall CLI through a browser-based terminal, relayed via the cloud application, even when the firewall's primary interfaces were misconfigured or offline.

Out-of-band (OOB) management refers to maintaining a dedicated path for recovery and low-level administration that is independent of the normal data plane. Integrating this capability into FortressFlash would extend its utility beyond one-time configuration pushes and provide a general-purpose remote console channel. From a security standpoint, the Pico 2 W platform is particularly suitable for this purpose, as its hardware secure boot, true random number generator, and OTP-based key storage could support a trustworthy cryptographic tunnel between the field device and the cloud service. A lightweight web application—for example, a browser terminal backed by a WebSocket relay—would provide the administrator interface.

Two primary usage scenarios illustrate the value of this feature. In the first, a level-1 technician is dispatched onsite and connects FortressFlash to the firewall, then links the device to a temporary Wi-Fi hotspot provided by their mobile phone. A remote administrator can immediately take over the CLI session through the web interface without needing direct physical presence. In the second, the device is deployed permanently alongside the firewall, provisioned with a dedicated 4G/5G modem or secondary WAN uplink. This configuration transforms FortressFlash into a standing OOB channel, ensuring that administrators retain access to the console even during network outages or configuration errors.

While realizing this feature would require careful design of the relay service and access controls, it would transform FortressFlash from a point solution for baseline configuration into a versatile remote management platform.

Status:* *Mid-term.

7.3 Short-Range Wireless: BLE & Wi-Fi

Field technicians will not always have a laptop—or even a stable LAN drop—when they unbox a firewall in a new comms closet. Adding short-range wireless links to FortressFlash therefore remains an attractive avenue for future work, not as a replacement for the secure serial channel but as a convenience layer for importing profiles, viewing logs, and performing last-second edits.

BLE Companion App (Phone ↔ Device). Because both the Pico 2 W and ESP32-S3 expose Bluetooth Low Energy, the most straightforward enhancement is a lightweight mobile application that speaks a custom GATT profile. The app would scan for the devices advertised UUID, establish an encrypted link, and expose three primary services: `profile_upload` (writeonly, chunked transfer of new configuration bundles), `status_log` (notify, live scroll of CLI output), and `pinpad_proxy` (write, for remote PIN entry or menu navigation). This keeps the on-device UI minimal while giving help-desk staff a familiar touchscreen keyboard and file picker.

Wi-Fi SoftAP + Captive Portal. When the embedded radio is idle, the device could launch a temporary 802.11 *soft access point* named `FortressFlash-SETUP`. Upon connection the user is redirected to a single-page, password-gated web UI served over HTTPS (self-signed or Let's Encrypt if WAN reachable). Features mirror the BLE app—upload profile, view logs, trigger "dry-run" syntax checks—but bandwidth and MTU ceilings are far higher than BLE, making firmware updates or large audit logs feasible. The portal drops back to monitor-only mode after a configurable timeout to limit the radio's attack surface.

Phone-as-Keyboard (Web HID over Wi-Fi). Text entry on a 2.8 resistive display is painful; shipping a Bluetooth HID stack adds size and pairing complexity. An elegant alternative is to expose a Web HID endpoint through the same Wi-Fi portal. When the user taps an input field (e.g., hostname), the browser captures keystrokes locally and streams them as UTF-8 packets to the device, which echoes them into the CLI buffer. This approach avoids classic Bluetooth while still giving the operator a full touchscreen or hardware keyboard.

Over-the-Air (OTA) Firmware Updates. Using the same SoftAP stack, the device could periodically poll a vendor-hosted manifest over HTTPS, download a signed-image delta, and stage it in inactive flash. Because secure boot hardware already enforces signature checks, the risk surface is limited to rollback attacks—mitigated by a monotonic version counter in OTP or eFuse.

Research Roadmap. Prototype efforts should first harden the BLE GATT service with authenticated pairing (LE Secure Connections, numeric comparison) and rate-limit write operations to foil fuzzing attempts. Next, port the web portal to a minimal React or Svelte bundle served from SPIFFS/LittleFS so it fits in flash. Finally, run comparative usability tests: manual touchscreen only vs. BLE app vs. Wi-Fi portal, measuring task-completion time and error rate. Results will guide whether both radio modes ship in parallel or if a single "phone-first" workflow is sufficient.

Status: Mid-term.

7.4 Long-Range Wireless: LoRa

What is LoRa? LoRa (Long Range) is a sub-GHz, low-power wide-area radio technology that uses chirp-spread-spectrum (CSS) modulation to achieve very high receiver sensitivity and robust operation in noisy RF environments [31]. Compared with Wi-Fi or Bluetooth, LoRa deliberately trades raw data rate for extended range, making it suitable for intermittent, small-payload telemetry rather than large file or streaming transfers. Regulatory allocations differ by region (for example EU deployments typically use 868 MHz plans while North American deployments use 902–928 MHz), so any prototype or production radio must be configured for the correct band and comply with local rules [32]. In practice, modern LoRa transceivers such as the Semtech SX126x family support transmit powers up to +22 dBm and are optimized for multi-kilometer links (line-of-sight) or hundreds of meters in obstructed urban settings when used with modest antennas [33].

LoRa Exploration. To evaluate the feasibility of integrating long-range wireless updates into the FortressFlash platform, a pair of Heltec WiFi LoRa 32 V3 development boards (ESP32S3 + SX1262, 915 MHz) were acquired, along with 10 dB gain whip antennas [10]. Basic experiments were conducted to assess both range and programmability. The boards were assembled into their included protective cases with internal 1100 mAh Li-ion batteries. The upgraded antennas were then mounted to the external SMA connectors to maximize transmission distance and signal quality (see Figure 7.2).

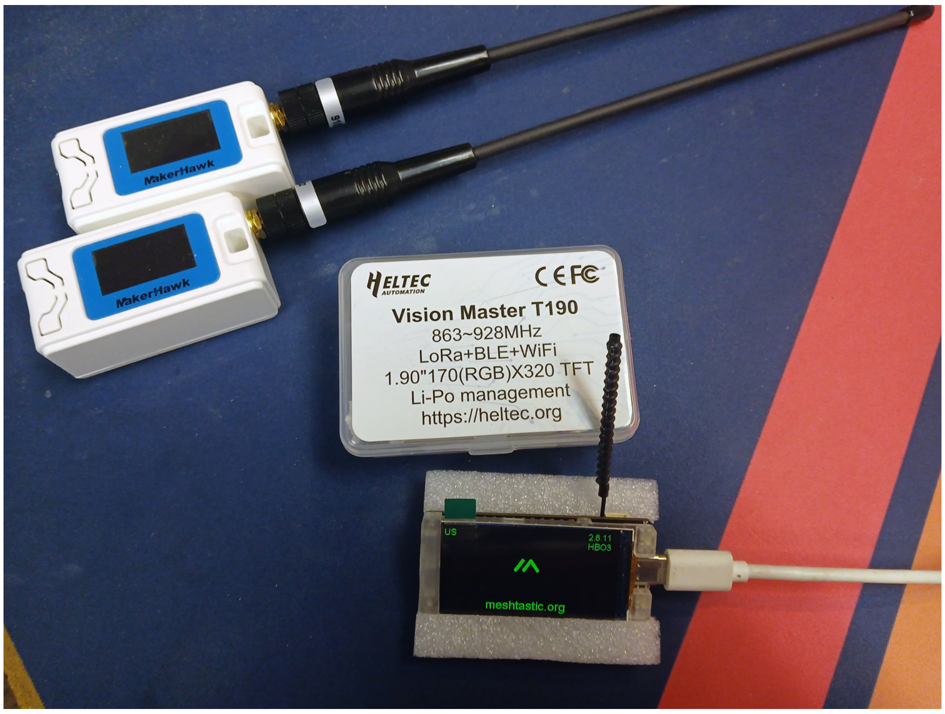


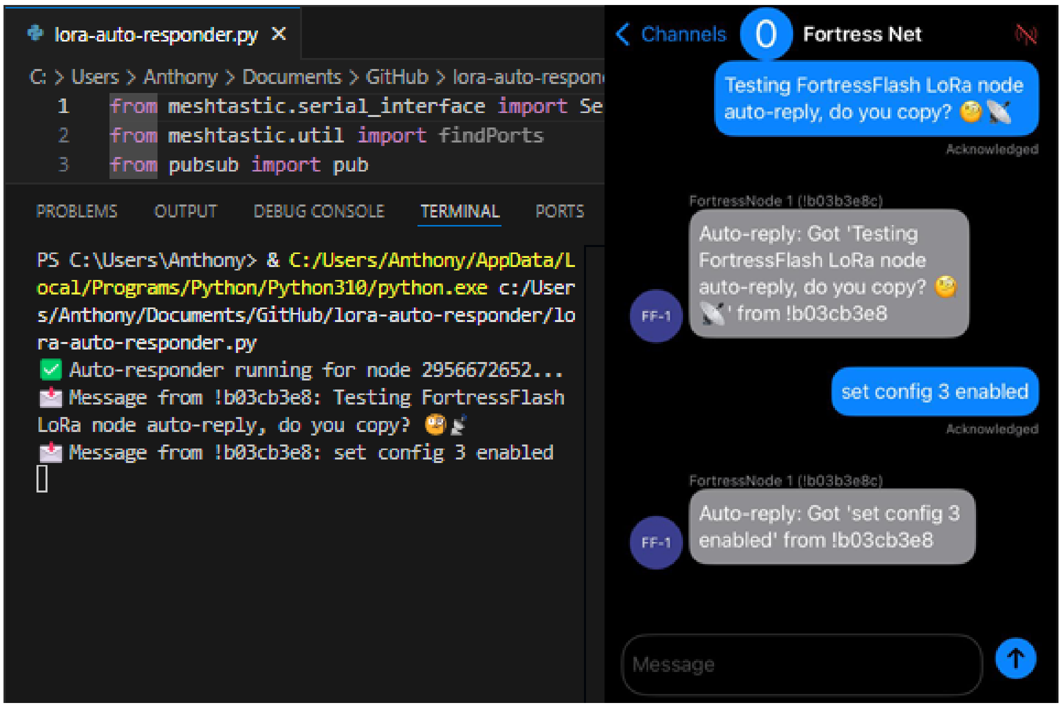
Figure 7.2: Various LoRa MCUs including a pair of Heltec WiFi LoRa 32 V3 boards in cases with whip antennas, and a single Heltec Vison Master T190 unit [10].

Flashing the devices with the latest Meshtastic `2.6.11.60ec05e` beta firmware using the official web flasher[2] was a delightfully simple process, thanks to HTML5 support for USB-C serial communication. Once the firmware was loaded, the devices could be controlled via several interfaces: the Meshtastic smartphone app (iPhone, in this case) over BLE; a workstation connected via USB serial using the Meshtastic web client[3]; the official Python libraries; or over Wi-Fi—though BLE and Wi-Fi cannot be active simultaneously.

After configuring both devices, one was connected to a workstation via USB-C and the other paired to an iPhone running the Meshtastic app. Initial communication was confirmed using the web client, after which a simple Python auto-responder was written to help gauge the effective communication range.[4] The script listens for incoming text messages over the USB serial interface and, if a cooldown timer has elapsed, automatically replies with the same message content—enabling hands-free, round-trip testing while in motion (see Figure 7.3).

The base station was positioned in the center of the FortressFlash Labs HQ, inside a windowless,

[2]https://flasher.meshtastic.org
[3]https://client.meshtastic.org
[4]Source code available at https://github.com/aku762/LoRa-Auto-Responder



```python
from meshtastic.serial_interface import Se
from meshtastic.util import findPorts
from pubsub import pub
```

```
PS C:\Users\Anthony> & C:/Users/Anthony/AppData/L
ocal/Programs/Python/Python310/python.exe c:/User
s/Anthony/Documents/GitHub/lora-auto-responder/lo
ra-auto-responder.py
✅ Auto-responder running for node 2956672652...
📩 Message from !b03cb3e8: Testing FortressFlash
LoRa node auto-reply, do you copy? 🫡📡
📩 Message from !b03cb3e8: set config 3 enabled
```

Testing FortressFlash LoRa node auto-reply, do you copy? 🫡📡
Acknowledged

FortressNode 1 (!b03b3e8c)
FF-1
Auto-reply: Got 'Testing FortressFlash LoRa node auto-reply, do you copy? 🫡📡' from !b03cb3e8

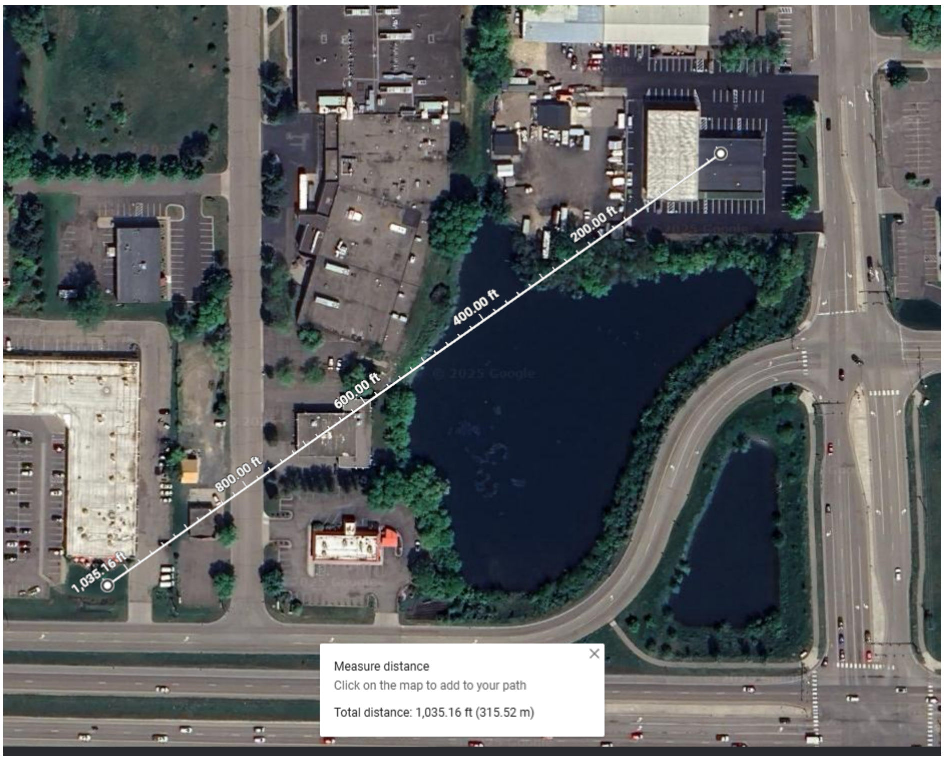
set config 3 enabled
Acknowledged

FortressNode 1 (!b03b3e8c)
FF-1
Auto-reply: Got 'set config 3 enabled' from !b03cb3e8

Figure 7.3: LoRa auto-responder test in action. The left pane shows the Python script running on a Windows workstation, receiving messages via USB serial and echoing them back. The right pane shows the Meshtastic mobile app connected to a second LoRa node over BLE, displaying both the original messages and the automated replies.

steel-roofed room approximately 1 meter (3 feet) above the floor—matching the height of the lab's test firewall. This setup represents a realistic deployment environment, especially for networking gear installed in telecom closets or data rooms. A basic range walk-test was then conducted: beginning south toward the nearby highway, then east to a small shopping plaza. The farthest successful reply was received at approximately 315 meters (1,035 feet). On the northbound street just west of the lab, signal loss occurred due to obstructions from intervening buildings. However, other locations roughly 300 meters north, along with the entire return leg on the road east of the lab, remained within functional range (see Figure 7.4).

Use-cases. These initial tests demonstrate the viability of using LoRa as a low-power, long-range communication channel for future iterations of FortressFlash. Despite the challenging indoor environment, reliable communication was achieved across distances exceeding 300 meters. Incorporating an external antenna—such as one mounted to the roof or extended outdoors—would further improve signal quality and effective range. In a production setting, a LoRa-enabled FortressFlash unit could remain installed at a customer site or secured within an equipment rack, while mobile technicians—equipped with paired nodes in service vehicles or handheld devices—could interact with it remotely during field visits or drive-by diagnostics. Such a setup would allow periodic transmission of status updates, configuration receipts, or alerts back to a central office.



Measure distance

Click on the map to add to your path

Total distance: 1,035.16 ft (315.52 m)

Figure 7.4: Google Maps screenshot showing the measured straight-line distance between the Fortress-Flash HQ base station and the farthest location where a successful auto-reply was received, totaling 315 meters (1,035 feet).

Beyond point-to-point use, LoRa's support for mesh networking and relay nodes enables more sophisticated multi-hop deployments. A nearby relay device can transparently forward packets on behalf of isolated or obstructed nodes, extending coverage without requiring any infrastructure changes to the target environment. Unlike Wi-Fi or cellular links, LoRa offers a lightweight, infrastructure-independent channel for asynchronous telemetry—particularly useful in locations where internet access is unavailable, unreliable, or intentionally segmented.

Status:* *Mid-term.

7.5 API Mode over TCP/IP

Modern FortiGate (and many competing) platforms expose a REST API for programmatic control and configuration. Although this pathway is acknowledged, it remains largely out of scope for the current hardware prototype. Reaching the API requires (i) a TCP/IP link into the same L2/L3 domain as the target firewall and (ii) pre-provisioned API tokens with sufficient privilege. While both Pico 1 and Pico 2 include on-board 802.11n radios, a given customer site may lack a usable access-point—or firewall rules may block wireless ingress altogether. Even when Wi-Fi is available, radio links introduce additional attack surface and key-management overhead.

A wired-Ethernet daughter-board (e.g., W5500 or ENC28J60) or an alternative MCU with native MAC/PHY would be the most robust research platform for future API exploration. CLI over UART, however, still offers three advantages the REST API cannot yet match:

1. **Zero-touch first boot.** The serial console is active before any network interface is up, enabling out-of-box provisioning and FIPS-CC image loading.
2. **Firmware-level tasks.** Operations such as loading factory test images or unlocking hidden partitions often remain CLI-only.
3. **Break-glass scenarios.** When the network plane is down (mis-routed VLAN, bad ACL, etc.), the console remains reachable.

Conversely, an API-centric workflow could provide a "clean" channel for long-horizon maintenance—status polling, incremental policy tweaks, or inventory queries—once the firewall is in steady state. A future revision might therefore adopt a dual-mode architecture: UART for bootstrap and recovery, REST for day-2 automation. Prototype work could compare request latency, error handling, and security posture (TLS mutual auth vs. console-only accounts) to quantify when the extra complexity is justified. It would also open the door to embedding tools commonly required during firmware upgrades, such as TFTP servers and other lightweight file transfer daemons.

Status:* *Long-term

7.6 Intent-Based Networking

Early scoping day-dreamed about typing "open 443 to the DMZ for web servers" and watching FortiOS commands spill out on the console. After short prototyping and workstation-class LLM experiments, it became clear that this rabbit hole is larger than a one-person sprint. The thesis therefore kept the core deliverable device-first and treated intent automation as long-range, cloud-assisted work.

What IBN is (and where it lives today). Intent-based networking seeks to let operators express desired outcomes while the system handles the "how." The canonical split is threefold—*intent ingestion*, *translation*, and *assurance*—with feedback loops to verify that realized state matches the stated intent [34]. In practice, most demonstrations assume a controller-centric fabric (e.g., SDN/virtualized data centers) that has a global view of topology and policy.

Why that is hard in a single-firewall, MCU context. A handheld MCU on a FortiGate console port sees only one device: its configuration, selected runtime counters/logs, and any context the operator supplies. There is no direct visibility into adjacent switches, VLANs, DHCP scopes, identity sources, or east–west flows. Under such constraints, open-ended NLP→CLI is ill-suited to on-device execution. The literature also warns about reliability: configuration synthesis remains a "moon-shot," and even 99.9% accuracy can yield hundreds of wrong lines across long configurations [35]. Any attempt at intent translation is therefore better delegated to a cloud service—with guardrails—if pursued at all.

Pragmatic compromise: a task-oriented "AI wizard" (cloud sidecar). A more realistic trajectory is to keep the device deterministic and use a companion service ("FortressFlash Cloud") as a guided assistant for *one task at a time*—e.g., block a website, publish a game server (VIP/NAT), set up SSL-VPN, or add an IDS profile. The assistant should ground suggestions in vendor documentation and vetted examples via retrieval, validate proposals in a staging/twin when feasible, and package only small, signed configuration change sets. The handheld's job does not change: verify, apply over UART, and log; later, pull over TCP/IP once API mode is available (Section 7.5) [36, 37].

Per-intent operator workflow (single list).

1. Choose a task category (e.g., site-to-site IPsec, SSL-VPN, VIP/NAT, URL/DNS block).
2. Provide minimal context (objects, addresses, ports/services; optionally relevant excerpts of the running config).
3. Receive a small, readable configuration change set with short citations to vendor docs or prior vetted snippets (retrieval-grounded) [36].
4. Dry-run in a digital twin or staging check; reject with reasons if post-conditions fail [37].
5. On approval, emit a signed bundle (manifest + CLI/API). The device verifies, applies, and records a simple audit trail linking the request to the applied lines.

Vendor alignment and integration stance. Vendors are best positioned to formalize intent for their own platforms and to expose safe translation paths via GUI/API as their AI tooling matures. The appropriate stance here is integrator, not inventor: track standards and published techniques [34], adopt the pieces that prove reliable (retrieval-grounded prompts, twin-based checks), and surface them through FortressFlash Cloud as optional, per-intent helpers—while the handheld remains the trustworthy actuator between natural language and the wire.

Status: Deferred indefinitely

Chapter 8: Conclusion

Problem and Aim. Small MSPs need a repeatable, low-friction way to provision firewalls without hauling a laptop or granting broad credentials. Entry-level technicians often lack the training or confidence to configure devices directly, leading to errors and truck rolls. The goal of this thesis was to design and build a handheld device that streamlines FortiGate bring-up while setting a credible path toward stronger security.

What Was Built. The FortressFlash prototype delivers a pocketable console tool with a touch UI, SD-backed profiles, and a prompt-paced dispatcher over UART. It boots reliably, mounts media, navigates via an L1-friendly menu, and establishes console sessions on target devices. The software architecture and flows are documented end-to-end, with verification checklists and bench evidence.

Results vs. Success Criteria. The primary criterion—guided, repeatable on-site provisioning for L1 technicians—has been substantially demonstrated. The prototype reliably boots, mounts SD profiles, and establishes console login sessions on a FortiGate target. A full end-to-end configuration push has not yet been executed, but the dispatcher and verification flows are implemented and undergoing validation. Secondary security objectives are partially satisfied in design (PIN gating, staged integrity checks), with a concrete hardening roadmap (Pico 2 migration, secure boot, OTP-backed keys, signed UF2 updates).

Limitations. The submission build targets rapid iteration in MicroPython; code updates are script-level and the end-to-end "push" path is still being validated. Credentials-at-rest encryption and secure firmware updates are intentionally deferred to the Pico 2 hardening phase and a compiled, signed C/C++ image.

Practical Viability. This work is presented as a proof-of-concept and a foundation for further R&D rather than a market-ready product. Commercializing a hardware appliance would require sustained firmware engineering, broad device/OS testing, support processes, and market validation in the face of vendor trends toward API/cloud management. A sensible next step—if pursued—would be a small paid pilot with MSP partners to validate time savings and support impact before any productization.

Process and Lessons (AI as Force Multiplier). An unplanned but central outcome was learning how a single engineer, paired with an AI assistant, can compress timelines without sacrificing rigor. ChatGPT accelerated refactors, unblocked driver bring-up, and turned sketches into implementable steps; judgment, validation, and design trade-offs remained human. In that sense, the project became a case study in modern solo engineering practice.

Author's Reflection. In an earlier chapter of my life, digital audio workstations (DAWs) and affordable DJ gear gave me a way to create music with tools that once required a full studio. That experience impressed on me how shifts in accessibility can change who gets to build and perform. FortressFlash became a mirror of that same pattern: a toolchain that let one engineer, working out of a "couch lab," build a system that once would have needed a team, lab access, and months of runway.

Enduring Lesson. AI was not the author of this work, but it was a catalyst. Like the DAWs that democratized music production or DSLRs that put cinematic tools in the hands of independent filmmakers, AI assistance compressed timelines and expanded scope without replacing judgment. The enduring contribution of FortressFlash is not only the device itself but the demonstration that a single motivated engineer, equipped with modern platforms and guided AI support, can deliver a professional-grade prototype in weeks. In that sense, FortressFlash is both a device and a case study in the new reality of engineering practice.

Closure. Taken together, the work is complete in scope: a working prototype, documented architecture, security roadmap, and a prioritized future-work path. While it is unlikely to be commercialized in its present form, it achieved the intended academic and professional value—demonstrating systems integration, security reasoning, and disciplined technical writing on a tight schedule. The FortressFlash identity will be retained as a personal banner for future builds, ensuring this work remains a foundation rather than an endpoint.

Availability. Architectural artifacts, figures, and deployment flows are documented herein; source code remains private for IP reasons but can be shared with evaluators upon request.

From sketch to silicon, from console to castle.

Appendix A: Supplementary Materials

A.1 Bill of Materials and Equipment

Table A.1: Bill of Materials for FortressFlash Prototype

Item	Part No.	Supplier	Cost (USD)
Waveshare RP2040-Plus[a]	RP2040-Plus-16MB-M	Amazon	\$17.80
Pico-ResTouch 2.8" LCD[b]	Pico-ResTouch-LCD-2.8	Amazon	\$23.99
MAX3232 RS232 To TTL[c]	Unbranded	eBay	\$0.82
DE-9 Mini Null Modem (M/M)	AD-N05M	CablesOnline	\$3.79
MicroSD Card 8 GB[d]	Microcenter 8 GB	MicroCenter	\$3.99
3D-Printed Enclosure[e]	Thingiverse Model 6128955	Self-printed	~ \$0.25

[a] Primary MCU, purchased via Amazon Waveshare Store. Mfg. website lists at \$7.99+s/h.
[b] Touchscreen, purchased via Amazon Waveshare Store. Mfg. website lists at \$14.99 plus shipping.
[c] Level shifter breakout module, DE-9 female connector.
[d] Configuration storage medium.
[e] Enclosure printed in-house using ABS on a Prusa i3 MK3S+, based on Tasuku Suzuki's Thingiverse model 6128955 [23] (licensed under CC BYSA), with minor fit adjustments.

Table A.2: Components Evaluated but Not Incorporated in Final Design

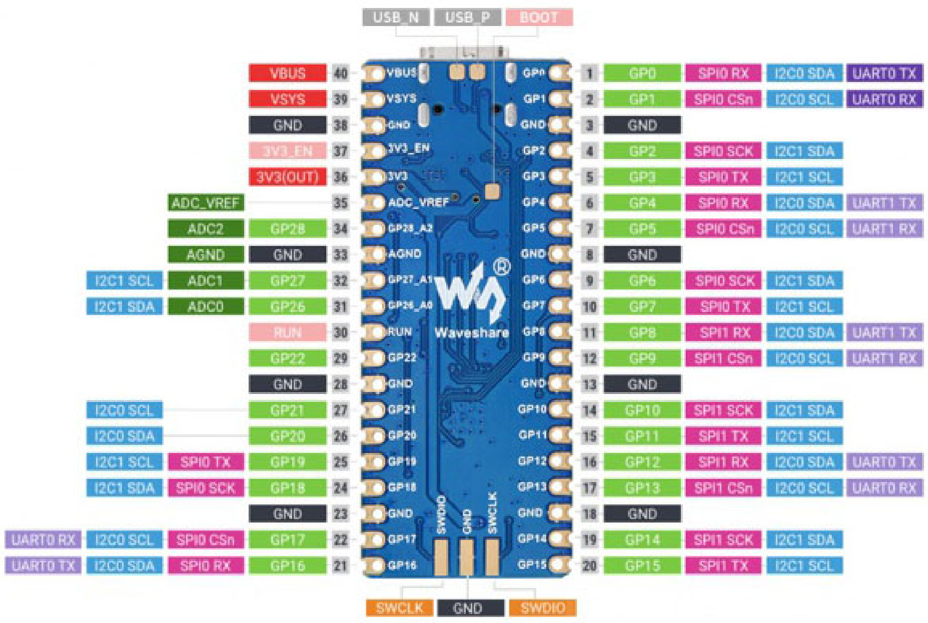
Item	Reason for Exclusion
TinkerKit LCD (T110061)	Limited peripheral interfaces (single UART), insufficient memory, and inadequate processing throughput; discontinued product.
Arduino Mega ADK R3	Physical footprint too large, insufficient memory, and inadequate processing throughput; discontinued product.
Raspberry Pi Pico W	Lacks native battery management and offers only 2 MB of onboard flash storage; while adequate for minimal use cases, the selected Waveshare board offers 16 MB of storage.
Raspberry Pi Zero 2 W	Classified as a single-board computer (SBC) rather than a microcontroller; introduces unnecessary complexity, operating-system overhead, and a larger attack surface.
RS232 to TTL Module (MAX3232, unbranded)	Exhibited intermittent and excessive overheating, indicating potential substandard manufacturing or incomplete supporting circuitry.
Leviton CAT5 Vertical Keystone Jack (8P8C)	Omitted as it was originally paired with the MAX3232 module; a full breakout board was used instead.

Table A.3: Lab and Test Environment Hardware

Item	Part No.	Source	Notes
FortiGate Firewall	FG-60E	Employer-provided	Target device for testing
Console Cable	RJ45-to-DE9	Included with FG-60E	Used for console access
Raspberry Pi Unit	Raspberry Pi 5 (16 GB)	Personal purchase	Development platform
USB–Serial Adapter	Ativa 26847	Personal purchase	For UART debugging
Pico Breadboard Plus Kit	Sapi EP-0172	Personal purchase	Prototyping base
SchmartModule RS-232	710-0001-01 Rev. A	Personal purchase	TTL to Serial Board

A.2 Pinout Diagrams

Pinout diagrams for both the RP2040-Plus microcontroller and the ResTouch 2.8" LCD are provided below for reference.



USB_N
USB_P
BOOT
VBUS 40 — 1 GP0 SPI0 RX I2C0 SDA UART0 TX
VSYS 39 — 2 GP1 SPI0 CSn I2C0 SCL UART0 RX
GND 38 — 3 GND
3V3_EN 37 — 4 GP2 SPI0 SCK I2C1 SDA
3V3(OUT) 36 — 5 GP3 SPI0 TX I2C1 SCL
ADC_VREF 35 — 6 GP4 SPI0 RX I2C0 SDA UART1 TX
ADC2 GP28 34 — 7 GP5 SPI0 CSn I2C0 SCL UART1 RX
AGND GND 33 — 8 GND
I2C1 SCL ADC1 GP27 32 — 9 GP6 SPI0 SCK I2C1 SDA
I2C1 SDA ADC0 GP26 31 — 10 GP7 SPI0 TX I2C1 SCL
RUN 30 — 11 GP8 SPI1 RX I2C0 SDA UART1 TX
GP22 29 — 12 GP9 SPI1 CSn I2C0 SCL UART1 RX
GND 28 — 13 GND
I2C0 SCL GP21 27 — 14 GP10 SPI1 SCK I2C1 SDA
I2C0 SDA GP20 26 — 15 GP11 SPI1 TX I2C1 SCL
I2C1 SCL SPI0 TX GP19 25 — 16 GP12 SPI1 RX I2C0 SDA UART0 TX
I2C1 SDA SPI0 SCK GP18 24 — 17 GP13 SPI1 CSn I2C0 SCL UART0 RX
GND 23 — 18 GND
UART0 RX I2C0 SCL SPI0 CSn GP17 22 — 19 GP14 SPI1 SCK I2C1 SDA
UART0 TX I2C0 SDA SPI0 RX GP16 21 — 20 GP15 SPI1 TX I2C1 SCL
Waveshare
SWDIO
GND
SWCLK
SWCLK GND SWDIO

Figure A.1: Waveshare RP2040-Plus pinout diagram. Source: Waveshare documentation [15].

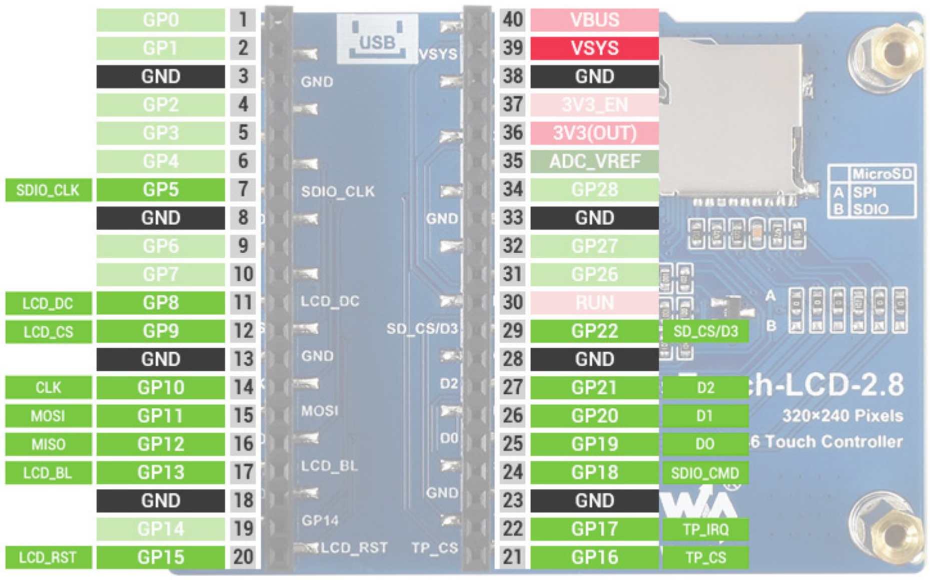
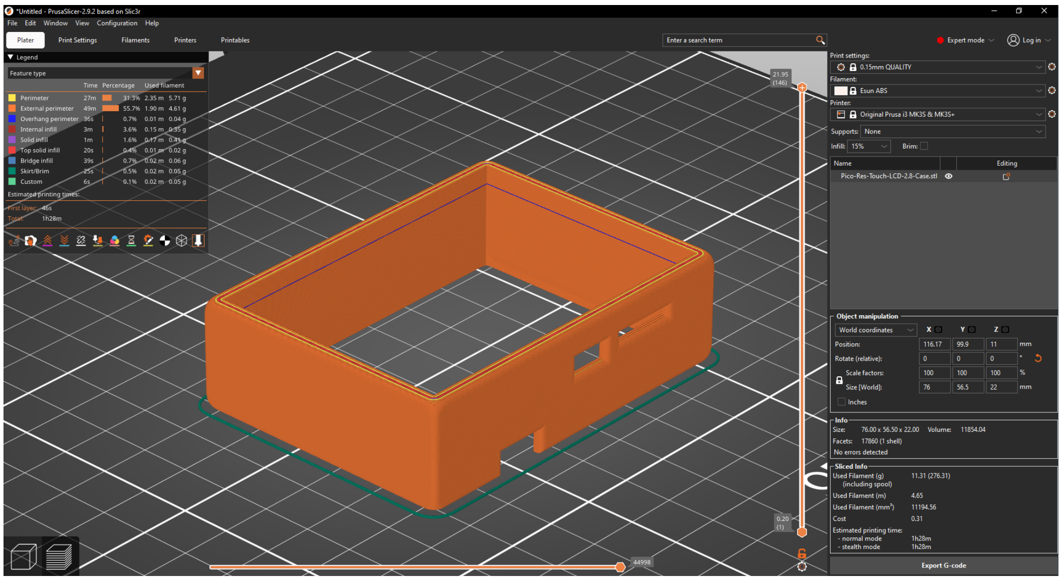


Figure A.2: Waveshare ResTouch 2.8" LCD pinout diagram. Source: Waveshare documentation [14].

A.3 3D Model Reference



Legend

Feature type

	Time	Percentage	Used filament	
🟨 Perimeter	27m	31.3%	2.35 m	5.71 g
🟧 External perimeter	49m	55.7%	1.90 m	4.61 g
🟦 Overhang perimeter	36s	0.7%	0.01 m	0.04 g
🟥 Internal infill	3m	3.6%	0.15 m	0.35 g
🟪 Solid infill	1m	1.6%	0.17 m	0.41 g
🟥 Top solid infill	20s	0.4%	0.01 m	0.02 g
🟦 Bridge infill	39s	0.7%	0.02 m	0.06 g
🟩 Skirt/Brim	25s	0.5%	0.02 m	0.05 g
🟩 Custom	6s	0.1%	0.02 m	0.05 g

Estimated printing times:
First layer: 46s
Total: 1h28m

Print settings: 0.15mm QUALITY
Filament: Esun ABS
Printer: Original Prusa i3 MK3S & MK3S+
Supports: None
Infill: 15% Brim: ☐

Name		Editing
Pico-Res-Touch-LCD-2.8-Case.stl		

Object manipulation

World coordinates	X	Y	Z	
Position:	116.17	99.9	11	mm
Rotate (relative):	0	0	0	°
Scale factors:	100	100	100	%
Size [World]:	76	56.5	22	mm

☐ Inches

Info
Size: 76.00 x 56.50 x 22.00 Volume: 11854.04
Facets: 17860 (1 shell)
No errors detected

Sliced Info

Used Filament (g) (including spool)	11.31 (276.31)
Used Filament (m)	4.65
Used Filament (mm³)	11194.56
Cost	0.31
Estimated printing time:	
- normal mode	1h28m
- stealth mode	1h28m

Figure A.3: Original STL model loaded in PrusaSlicer prior to printing. Model by Tasuku Suzuki, licensed under CC BY-SA [23].

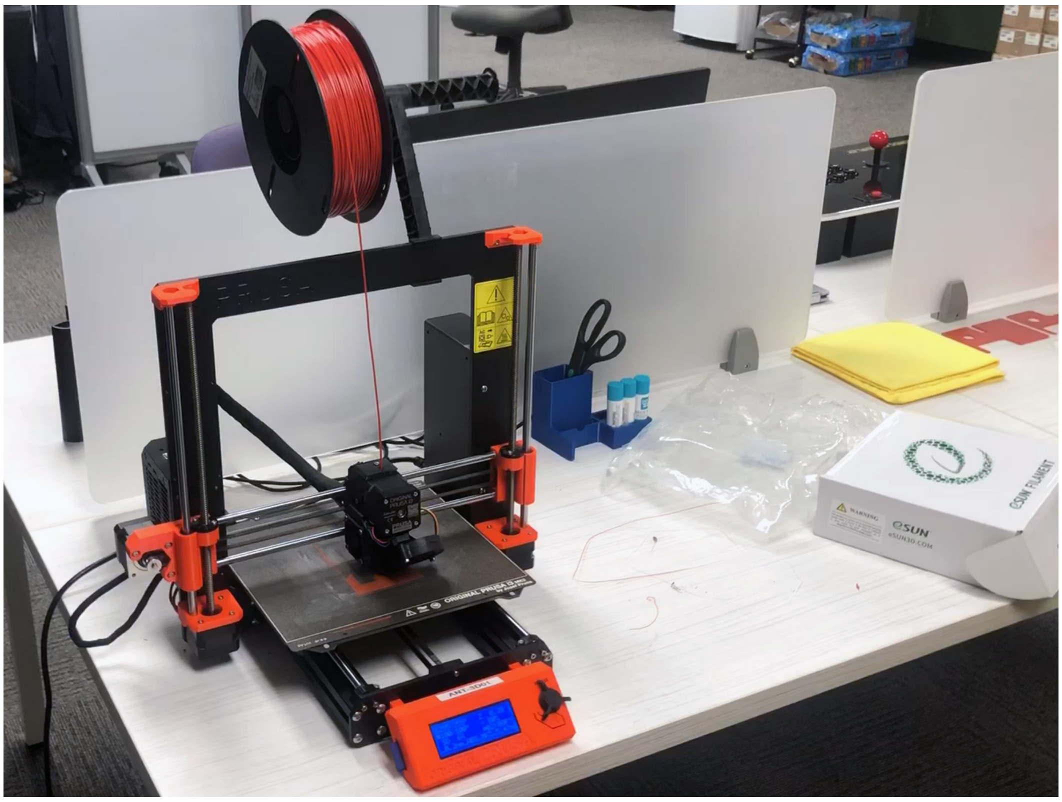


Figure A.4: Enclosure being printed on a Prusa i3 MK3S+ using eSUN red ABS filament.

A.4 Benchmarks

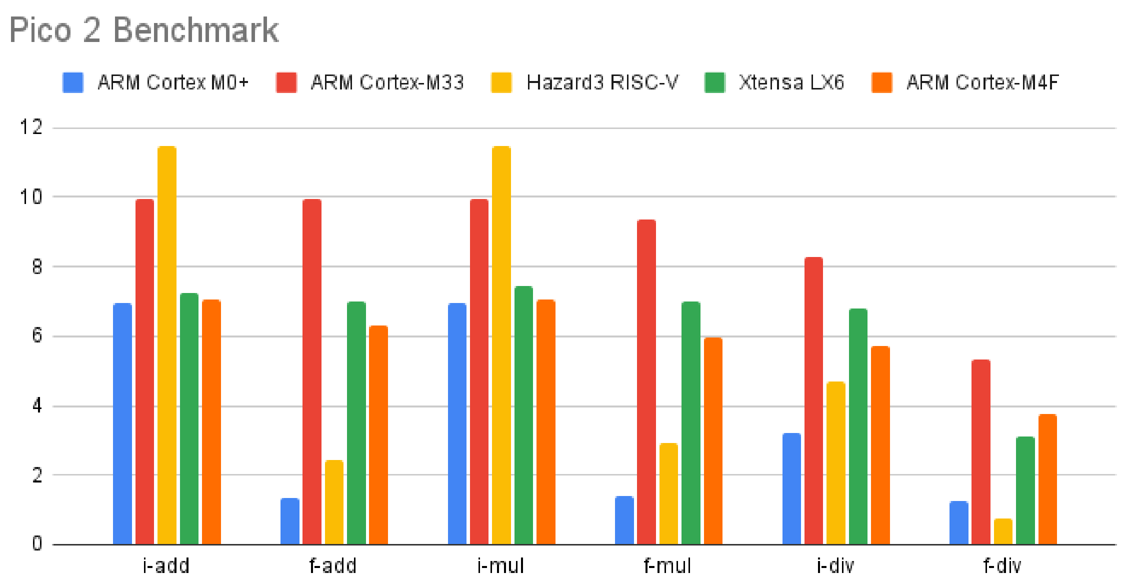


Figure A.5: Benchmark results (MOPS) comparing Raspberry Pi Pico and Pico 2, and other selected boards at their native clock speeds. Figure reproduced from Cheppali [13].

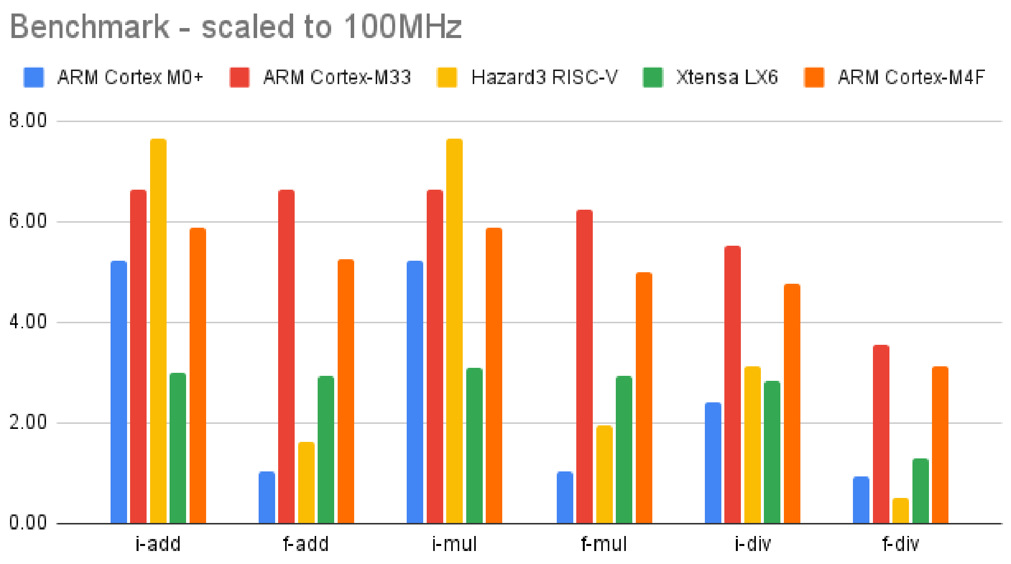
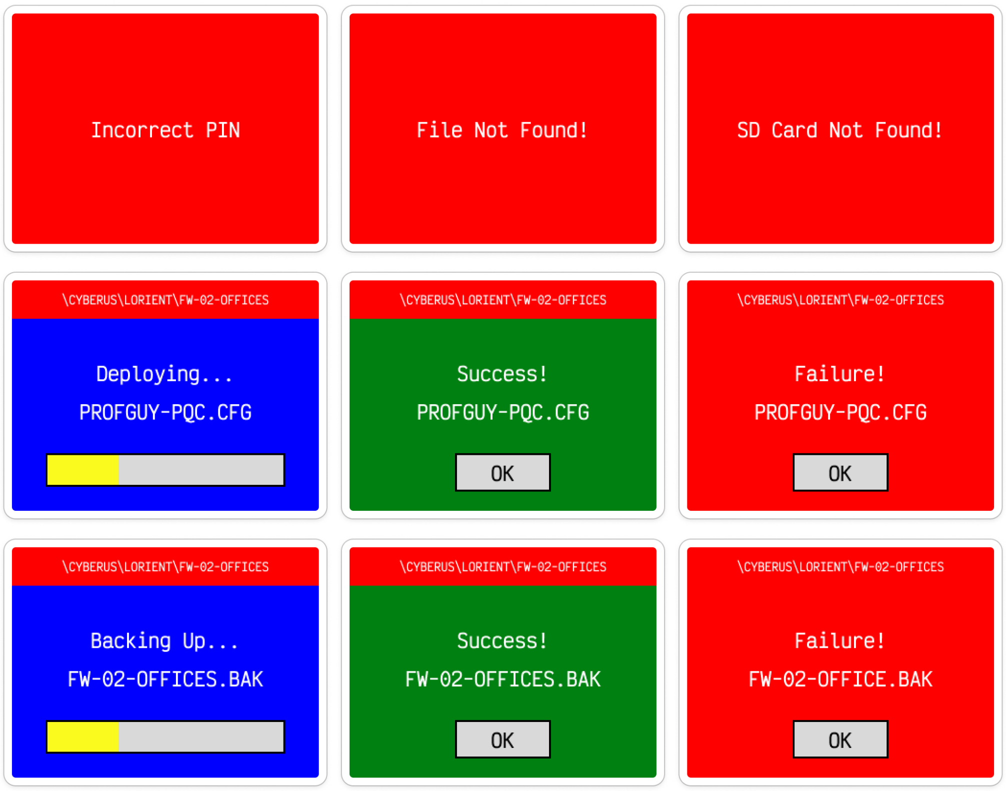


Figure A.6: Benchmark results (MOPS) normalized to 100 MHz for architectural efficiency comparison. Figure reproduced from Cheppali [13].

A.5 User Experience and Interface


Incorrect PIN

File Not Found!

SD Card Not Found!

\CYBERUS\LORIENT\FW-02-OFFICES
Deploying...
PROFGUY-PQC.CFG

\CYBERUS\LORIENT\FW-02-OFFICES
Success!
PROFGUY-PQC.CFG
OK

\CYBERUS\LORIENT\FW-02-OFFICES
Failure!
PROFGUY-PQC.CFG
OK

\CYBERUS\LORIENT\FW-02-OFFICES
Backing Up...
FW-02-OFFICES.BAK

\CYBERUS\LORIENT\FW-02-OFFICES
Success!
FW-02-OFFICES.BAK
OK

\CYBERUS\LORIENT\FW-02-OFFICES
Failure!
FW-02-OFFICE.BAK
OK

Figure A.7: FortressFlash status and error screens for PIN validation, file operations, and deployment/backup processes.

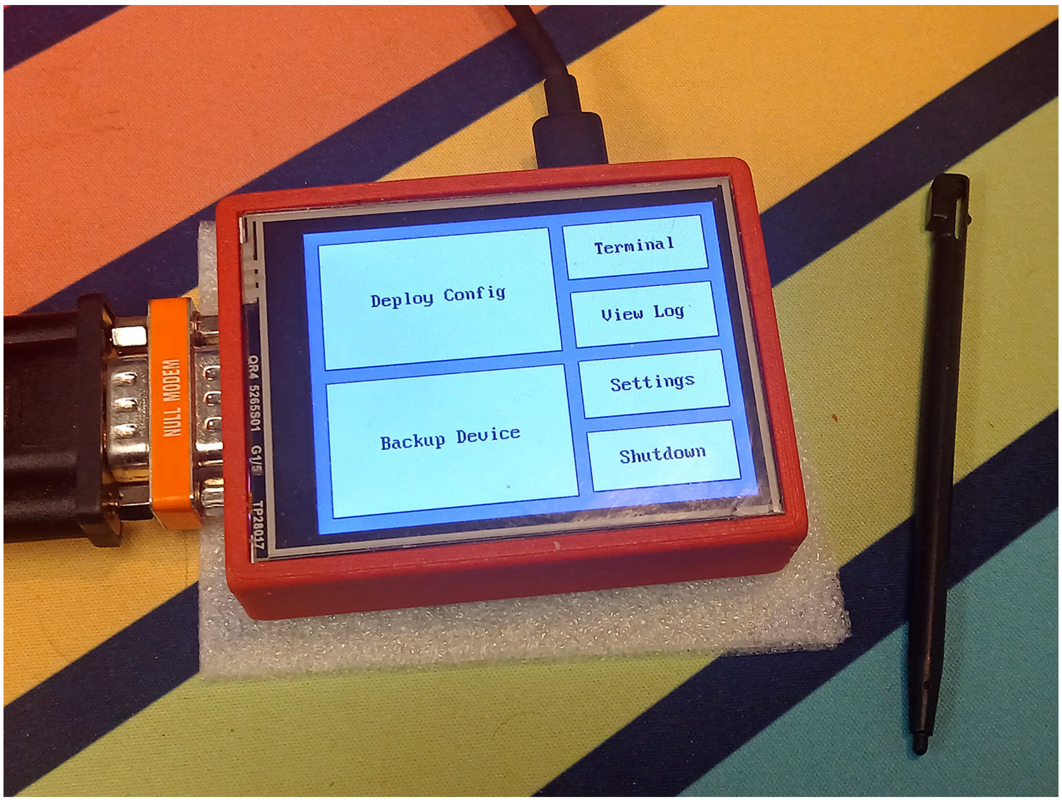
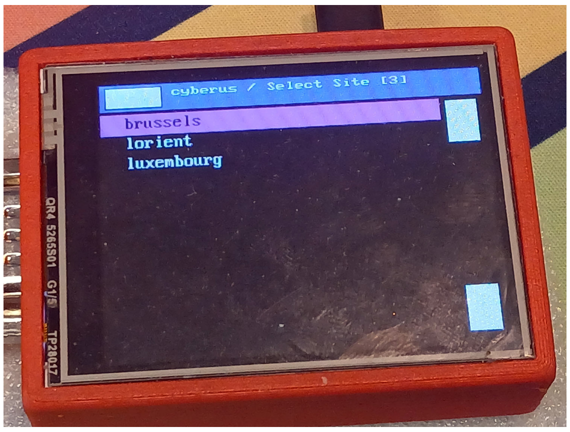
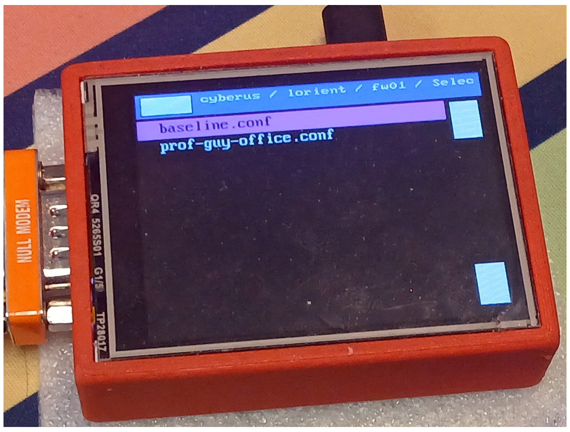


Figure A.8: FortressFlash main menu displayed on the 2.8 ” touchscreen, showing options for deploying a baseline configuration, opening a terminal, viewing logs, backing up a device, accessing settings (including calibration), and shutting down.

Configuration Picker



(a) Site selection menu.



(b) Configuration file picker.

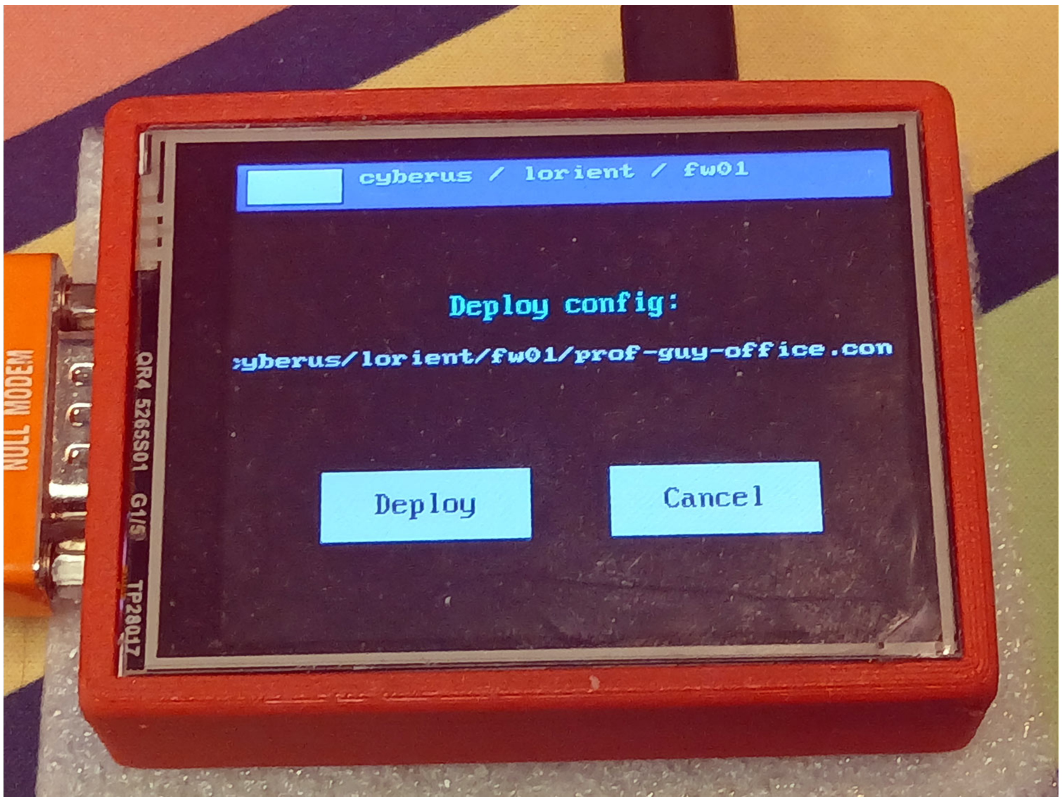


Figure A.9: FortressFlash profile-selection workflow: (a) site selector, (b) configuration picker, and final (c) deployment confirmation screen showing the chosen profile ready to push.

Touch Calibration Wizard

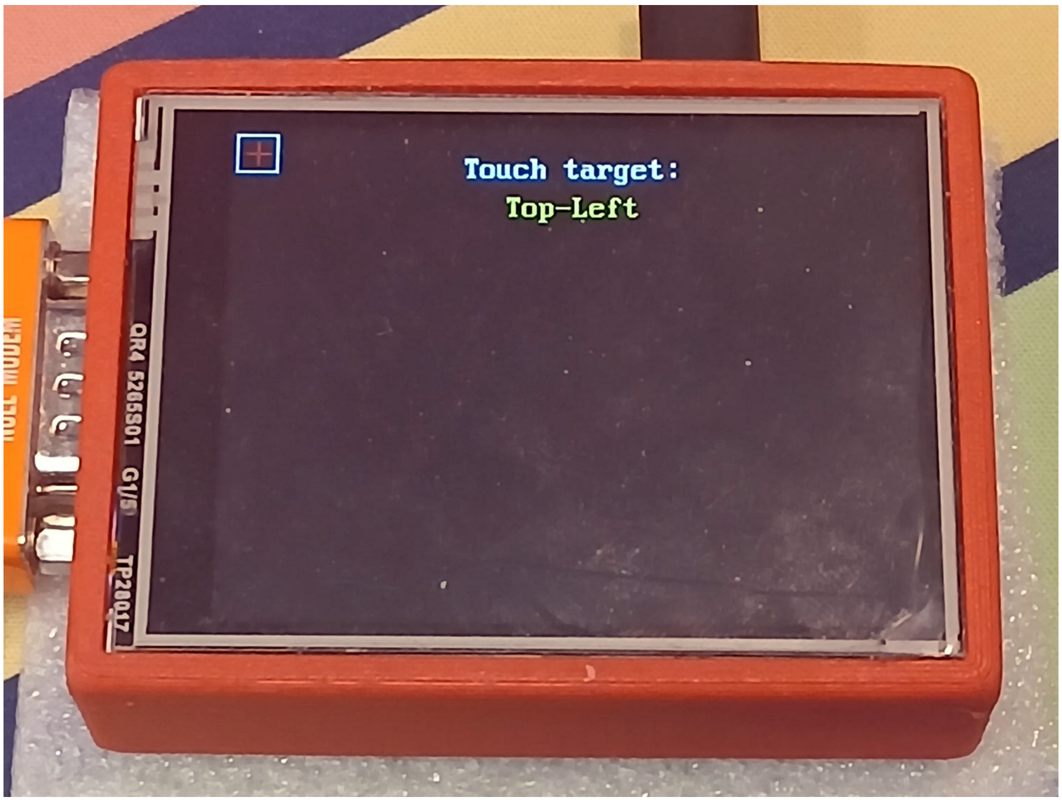


Figure A.10: Touchscreen calibration utility displayed on the FortressFlash prototype. The LCD's native orientation is vertical (portrait), but the device enclosure was built for landscape use. In the initial layout, the DE-9 console connector was positioned on the right edge. After assessment, it was relocated to the left side to improve accessibility and cable management, aligning it with the USB-C port, reset button, and microSD slot along the top edge. This hardware reorientation disrupted the default touch mapping, with both axis rotation and scaling out of alignment, producing severe pointer offsets. Intensive calibration and repeated testing were therefore required to restore accurate touch behavior. The utility shown here, implemented as a standalone `touch_wizard.py` script, was used during development to iteratively adjust the calibration matrix until stable and precise input was achieved. It is not yet integrated into the planned *Settings* menu or set up for auto-configuration, but provided the baseline for eventual inclusion.

A.6 Brand Assets

Figure A.11: Branded FortressFlash technical polo and cap, created to reinforce the project's identity and communicate professionalism during demonstrations and thesis defense. The cap is embroidered with "FortressFlash Labs," a phrase later echoed in Section 7.4 (LoRa Exploration) to suggest continuity as an ongoing innovation space.

A.7 Additional Images

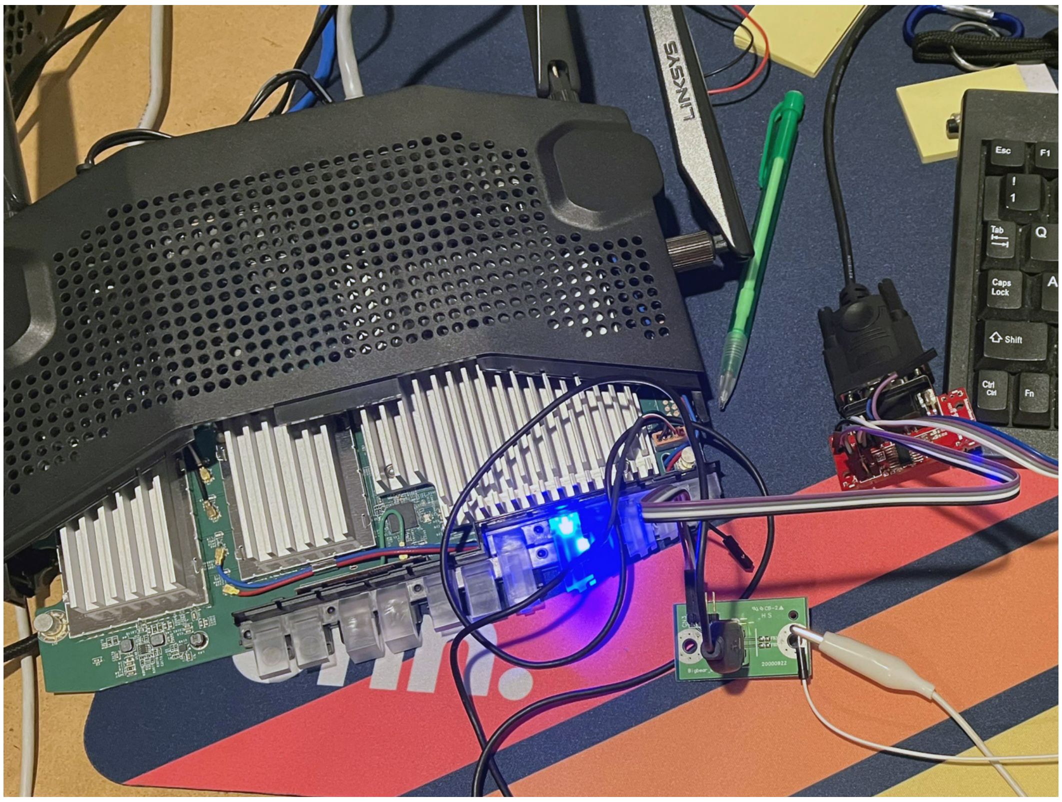


Figure A.12: Recovering a Linksys WRT32X using its internal serial header, connected via external USB-to-Serial and RS-232-to-TTL adapters, along with various breakout boards.

References

[1] Anthony Karels et al. *ICS Pentesting Report*. Project report, M1 Cyberus. Pentesting course supervised by O. Mawloud and S. Bouchelaghem. Université Bretagne Sud (UBS), Apr. 2024. URL: https://cyberus.lu/reports/Karels_ICS-Pentest_Cyberus_2024.pdf.

[2] Anthony Karels. *Cybersecurity Maturity Model Certification Project Report*. Internship Report, M1 Cyberus. CMMC Level 2 certification preparation and FortiGate CLI configuration in FIPS-CC mode. Top Dog PC Services, Université Bretagne Sud (UBS), June 2024. URL: https://cyberus.lu/reports/Karels_CMMC-Internship_Cyberus_2024.pdf.

[3] Tsedal Neeley, Jeff Huizinga, and Emily Grandjean. *Fortinet: Cybersecurity Pioneer Ken Xie Considers the Long Game*. Harvard Business School Case 424-016. Revised March 2024. Oct. 2023. URL: http://hbr.org/product/Fortinet--Cybersecurity-P/an/424016-PDF-ENG.

[4] TinkerKit. *LCD Module*. Archived on September 23, 2014 via the Internet Archive. 2014. URL: https://web.archive.org/web/20140923070802/http://www.tinkerkit.com/lcd/ (visited on 07/30/2025).

[5] Arduino. *Leonardo Board Specifications*. URL: https://docs.arduino.cc/hardware/leonardo (visited on 07/30/2025).

[6] Arduino. *Arduino Mega ADK Rev3 Board Specifications*. Last revised March 14, 2024. 2024. URL: https://docs.arduino.cc/retired/boards/arduino-mega-adk-rev3/ (visited on 07/30/2025).

[7] Raspberry Pi Ltd. *Raspberry Pi Pico W Product Brief*. Apr. 2024. URL: https://datasheets.raspberrypi.com/picow/pico-w-product-brief.pdf (visited on 07/30/2025).

[8] Raspberry Pi Ltd. *Raspberry Pi Pico 2 W Product Brief*. Nov. 2024. URL: https://datasheets.raspberrypi.com/pico/pico-2-product-brief.pdf (visited on 07/30/2025).

[9] Raspberry Pi Ltd. *Raspberry Pi Zero 2 W Product Brief*. Apr. 2024. URL: https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf (visited on 07/30/2025).

[10] Heltec Automation. *WiFi LoRa 32 V3 (ESP32-S3) – HTIT-WB32LA V3.2 Datasheet*. Accessed Aug 2025. Heltec Automation. Sept. 2022. URL: https://resource.heltec.cn/download/WiFi_LoRa_32_V3/HTIT-WB32LA_V3.2.pdf.

[11] Miguel Grinberg. *Benchmarking MicroPython*. https://blog.miguelgrinberg.com/post/benchmarking-micropython. Accessed Aug 2025. 2025.

[12] ArduSimple. *MicroPython Platforms Benchmarking*. 2023. URL: https://www.ardusimple.com/micropython-platforms-benchmarking/ (visited on 07/30/2025).

[13] Sankar Cheppali. *Benchmarking Raspberry Pi Pico 2*. 2024. URL: https://icircuit.net/benchmarking-raspberry-pi-pico-2/3983 (visited on 07/30/2025).

[14] Waveshare Ltd. *ResTouch 2.8 Product Page*. Product imagery and board layout adapted under fair use for educational purposes. URL: https://www.waveshare.com/pico-restouch-lcd-2.8.htm (visited on 08/04/2025).

[15] Waveshare Ltd. *RP2040-Plus Product Page*. Product imagery and board layout adapted under fair use for educational purposes. URL: https://www.waveshare.com/product/raspberry-pi/boards-kits/raspberry-pi-pico-cat/rp2040-plus.htm (visited on 08/04/2025).

[16] Cody Hyman. *RS-232 DE-9 Connector Pinouts*. Licensed under CC BY-SA 3.0. 2013. URL: https://commons.wikimedia.org/wiki/File:RS-232_DE-9_Connector_Pinouts.png.

[17] jobefox. *UTP Cat5E, RJ45 Wiring Diagram*. Originally published on Openclipart. Public domain. 2018. URL: https://commons.wikimedia.org/wiki/File:UTP_Cat5E_wiring.svg.

[18] *MAX3232: 3-V to 5.5-V Multichannel RS-232 Line Driver and Receiver With ±15-kV ESD Protection*. Rev. O. Texas Instruments. Dallas, TX, June 2021. URL: https://www.ti.com/lit/ds/symlink/max3232.pdf (visited on 07/31/2025).

[19] Pete. *Investigating Fake MAX3232 TTL-to-RS-232 Chips*. Sept. 2016. URL: https://blog.heypete.com/2016/09/11/investigating-fake-max3232-ttl-to-rs-232-chips/ (visited on 07/31/2025).

[20] Sodrohu. *MAX3232 overheating/burnt after connecting to PC*. Electrical Engineering Stack Exchange post. July 2014. URL: https://electronics.stackexchange.com/questions/122769/max3232-overheating-burnt-after-connecting-to-pc (visited on 07/31/2025).

[21] ET. *Confusing but works, see *CORRECT* pinouts attached.* Amazon customer review of HiLetgo MAX3232 Level Converter Board with Corrected Pinouts. Oct. 2022. URL: https://www.amazon.com/review/R3LES2QASI591Y (visited on 07/31/2025).

[22] Adafruit Industries. *RS232 Pal – Two Channel UART to RS-232 Level Shifters (MAX3232E), Product 5987*. 2023. URL: https://www.adafruit.com/product/5987 (visited on 07/31/2025).

[23] Tasuku Suzuki. *3D Printable Case for WAVESHARE 2.8inch Touch Display Module for Raspberry Pi Pico*. https://www.thingiverse.com/thing:6128955. Licensed under Creative Commons Attribution-ShareAlike (CC BY-SA). July 2023. (Visited on 07/30/2025).

[24] Gregor Cresnar. *Storage Card*. https://thenounproject.com/browse/icons/term/storage-card/. Icon from Noun Project, licensed under CC BY 3.0.

[25] DOUBLE SLASH. *Back*. https://thenounproject.com/browse/icons/term/back/. Icon from Noun Project, licensed under CC BY 3.0.

[26] OWASP Foundation. *OWASP Threat Modeling Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html. 2025. (Visited on 07/20/2025).

[27] Alexander Osterwalder and Yves Pigneur. *The Business Model Canvas*. PDF handout retrieved from Strategyzer Library. Strategyzer. Feb. 11, 2025. URL: https://www.strategyzer.com/library/the-business-model-canvas (visited on 08/05/2025).

[28] Mordor Intelligence. *United States Managed Services Market - Growth, Trends, and Forecasts (2025–2030)*. Accessed: 2025-08-20. 2025. URL: https://www.mordorintelligence.com/industry-reports/united-states-managed-services-market.

[29] CloudSaver. *2023: Year of the Cloud Managed Service Provider*. Cites MSPAlliance estimate of ˜40,000 U.S. MSPs. 2023. URL: https://www.cloudsaver.com/resources/articles/2023-year-of-the-cloud-managed-service-provider/.

[30] Infrascale / MSP Resources. *Disaster Recovery for Managed Service Providers (MSPs): Key Statistics in the U.S.* Reports approximately 40,000 MSPs in the United States. 2023. URL: https://www.infrascale.com/draas-msp-statistics-usa/.

[31] Semtech Corporation. *LoRa® and LoRaWAN®: Technology Overview*. https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf. White paper / technology overview. Accessed: 2025-08-17.

[32] LoRa Alliance. *RP002-1.0.2 LoRaWAN® Regional Parameters*. Tech. rep. Defines regional frequency plans (EU868, US915, etc.). Accessed: 2025-08-17. LoRa Alliance, 2020. URL: https://lora-alliance.org/wp-content/uploads/2020/11/RP_2-1.0.2.pdf.

[33] Semtech Corporation. *SX1261/62 Transceiver Datasheet*. Datasheet (SX1262 supports up to +22 dBm). Accessed: 2025-08-17. Semtech. Dec. 2017. URL: https://www.mouser.com/datasheet/2/761/DS_SX1261-2_V1.1-1307803.pdf.

[34] A. Clemm et al. *Intent-Based Networking – Concepts and Definitions*. RFC 9315. 2022. URL: https://doi.org/10.17487/RFC9315.

[35] Zied Ben Houidi and Dario Rossi. “Neural language models for network configuration: Opportunities and reality”. In: *Computer Communications* 193 (2022), pp. 118–125. DOI: 10.1016/j.comcom.2022.06.035.

[36] Salwa Mostafa et al. “RAG-Enabled Intent Reasoning for Application-Network Interaction”. In: *arXiv* (2025). eprint: 2505.09339. URL: https://arxiv.org/abs/2505.09339.

[37] Kristina Dzeparoska, Ali Tizghadam, and Alberto Leon-Garcia. “Intent Assurance using LLMs guided by Intent Drift”. In: *arXiv* (2024). eprint: 2402.00715. URL: https://arxiv.org/abs/2402.00715.

Witness

This work was shaped not only in study, but in the space between—those quiet hours when the page remained untouched out of conviction, not delay. In that rhythm, insight came differently: not forced, but given. I am grateful to the One who governs both time and timing, who speaks even in the pauses, and strengthens resolve when the world insists on speed.

The path here was also a return—across oceans, archives, and ancestral names. Reclaiming what was nearly forgotten became more than an administrative act; it became a kind of witness. The red lion still stands, even if surrounded by forgetting. To those few who still prepare—not just in mind or skill, but in loyalty—I offer this as a small report from the field.

The sky will crack open. Until then, may we be found ready.