PROJECT REPORT

Secure Boot and Firmware Update Framework Using Hardware Root of Trust on INDUS Platform

1	Ashank Swaminathan	250250130002
2	Sivaelango N	250250130011
3	Thirumurugan S	250250130012
4	Yashwanth S	250250130013

ACKNOWLEDGEMENT

The project titled " Secure Boot and Firmware Update Framework Using Hardware Root of Trust on STM32 Platform" has been a challenging and enriching journey that brought together the objectives of secure data transmission, real-time connectivity, and embedded system integration. This project represents a significant milestone in our academic and professional growth, and it would not have been possible without the support and guidance we received along the way. We would like to express our heartfelt gratitude to our guide, Mr. Krishna C, for his continuous support, insightful guidance, and encouragement throughout the course of this project. His technical expertise and constructive feedback were instrumental in overcoming the various challenges we faced. We are especially thankful for his assistance in helping us understand complex concepts and for facilitating the resources and direction necessary for the successful execution of this project. This project marks just one of many steps in our journey, and the experience gained here will serve as a strong foundation for the challenges we are yet to face. (PG-DESD FEBRUARY 2025)

Table of Contents

ABSTRACT

This project aims to develop a secure communication framework for IoT devices using the ESP32 microcontroller, Infineon's Trust M secure element, the mbedTLS cryptographic library, and the HTTP protocol. The system is designed to ensure confidentiality, integrity, and authenticity of data exchanged between IoT devices and cloud servers. Secure communication is achieved using Transport Layer Security (TLS) and X.509 digital certificates for mutual authentication. The Infineon Trust M chip is integrated to provide hardware-level key storage and perform cryptographic operations such as signing and encryption, thereby offloading these tasks from the ESP32.

In the current implementation phase, the ESP32 has been successfully configured to communicate with a public HTTP broker over a TLS-secure channel using mbedTLS, with AES encryption applied to transmitted messages. Future phases will focus on full integration of the Trust M module to enhance security and performance. This project offers a scalable and practical solution for secure IoT deployments, with potential applications in smart homes, healthcare systems, and industrial automation.

Title of the project

Secure Boot and Firmware Update Framework Using Hardware Root of Trust on STM32 Platform

01 - Introduction

i) Purpose

This project, titled **"Secure Boot and Firmware Update Framework Using Hardware Root of Trust on STM32 Platform",**focuses on enhancing the reliability and security of firmware updates for embedded IoT devices deployed on the STM32-based development board. The solution implements a minimal secure bootloader designed to support robust firmware management via UART communication.

The core objective is to ensure secure remote firmware updates for IoT devices, which are often deployed in remote or inaccessible locations where manual updates are impractical and error-prone. The secure bootloader enables the device to receive firmware images over UART, verify their integrity, safely store them in flash memory, and securely boot the application only after successful validation.

This framework addresses key security concerns such as firmware tampering, rollback attacks, and unauthorized code execution. By incorporating cryptographic verification and controlled update mechanisms, the system maintains the integrity, authenticity, and reliability of the firmware throughout the device lifecycle.

ii) Document Conventions

1. **SB**: Secure Boot
2. **OTA**: Over-the-Air
3. **HSM**: Hardware Security Module
4. **STM32 Platform:** The target embedded hardware platform used in this project.
5. **PKI**: Public Key Infrastructure.

02 - Objective

The main objectives of this project are:

- To design and implement a secure and reliable Over-the-Air (OTA) firmware update mechanism for the STM32-based development board, ensuring minimal downtime and maximum security.
- To develop a lightweight secure bootloader on the STM32F4 microcontroller that can receive firmware updates via UART, verify their authenticity, and store them safely in flash memory.
- To ensure secure application booting by validating firmware integrity and authenticity before execution, thereby preventing the device from running tampered or unauthorized code.
- To utilize OPTIGA™ Trust M as a Hardware Root of Trust (HRoT) for secure key storage and cryptographic signature verification, enhancing the system's defense against rollback attacks and firmware tampering.
- To offload OTA handling and cloud communication to an external ESP32 module, enabling secure Wi-Fi-based firmware retrieval while keeping the STM32 bootloader minimal and focused.
- To address the challenges of remote firmware updates in inaccessible IoT deployments, providing a secure, scalable, and efficient alternative to manual updates.

03 - Design and Constraints

i) Project/Product Perspective

This project aims to enhance the security of embedded systems on the STM32 platform by implementing a secure boot and firmware update mechanism using a Hardware Root of Trust (HRoT), such as Optiga™ Trust M. The framework ensures that only authenticated and untampered firmware is executed or updated on the device. It addresses the growing need for protecting devices from unauthorized access, tampering, and firmware-level attacks. This solution acts as a core security layer and is designed to be modular, reusable, and easily integrable into various STM32-based applications.

ii) Product Functions

The main functionalities of the project include:

1. Secure Boot Process – Verifies the integrity and authenticity of the firmware during system startup using cryptographic signatures.

2. Secure Firmware Update – Ensures only verified and authorized firmware updates are accepted and installed.

3. Hardware Root of Trust Integration – Uses a secure element (e.g., Optiga™ Trust M) to store cryptographic keys and perform secure operations.

4. Firmware Rollback Protection – Prevents installation of outdated or previously compromised firmware versions.

5. Tamper Detection and Failure Handling – Detects signature mismatches and prevents execution of untrusted firmware.

6. Modular and Scalable Design – Can be integrated with different STM32 platform applications with minimal changes.

iii) Design and Implementation Constraints

- The solution must operate on embedded systems with limited resources, such as low memory (32 KB) and flash storage (128 KB).

- The secure boot and firmware update process should be efficient, minimizing boot time and resource usage.

- **Development Environment Constraints:** The system is developed primarily using C for the embedded firmware, and for server-side automation and firmware signing.

- Limited storage and computational power on embedded devices restrict the ability to run large cryptographic algorithms. The solution must be optimized for performance and memory usage.

Overall this project delivers a secure and lightweight firmware update framework tailored for STM32-based embedded systems. A key component is a **minimal secure bootloader** developed for the STM32F4 microcontroller. It supports firmware reception via UART, verifies digital signatures, securely stores validated firmware in internal flash, and ensures only authenticated and untampered firmware is executed during startup.

To strengthen platform integrity, the solution integrates the **OPTIGA™ Trust M** secure element as a **Hardware Root of Trust (HRoT)**. It facilitates secure key storage and offloads cryptographic operations such as signature verification, significantly enhancing protection against firmware tampering and rollback attacks.

The architecture also includes an **external OTA (Over-the-Air) management module** built on the ESP32 platform. This module handles secure Wi-Fi-based communication with the cloud, manages firmware downloads, performs integrity checks, and transmits validated firmware to the STM32 via UART. This modular offloading approach keeps the STM32 bootloader lightweight and efficient.

The complete solution demonstrates a **fully functional, end-to-end secure firmware update flow**, showcasing cryptographic validation via Trust M and safe firmware installation on the STM32 platform.

04 - Literature Survey

In recent years, the demand for secure firmware update mechanisms in embedded systems has increased significantly, particularly in IoT and STM32trial automation environments. Numerous approaches have been developed to address security challenges, including secure boot, encrypted communication, rollback protection, and the use of secure elements.

i) STM32 Secure Bootloader and X-CUBE-SBSFU

STMicroelectronics introduced the X-CUBE-SBSFU software package, which enables secure boot and firmware update on STM32 microcontrollers. It supports firmware image authentication and secure firmware download over interfaces such as UART, USB, and Wi-Fi. However, the SBSFU framework assumes certain hardware security features or larger memory capacity that might not be available on constrained MCUs like the STM32F4.

Our project draws inspiration from SBSFU in terms of image integrity verification and boot validation but simplifies it for more constrained devices by offloading secure communication to an external ESP32.

ii) Trust M Secure Element

Infineon's OPTIGA™ Trust M is a hardware-based secure element that provides secure storage of cryptographic keys and executes cryptographic operations like ECC-based signature verification. It complies with industrial-grade security standards and is resistant to physical attacks.

In our project, Trust M plays a crucial role in verifying firmware authenticity before boot, thereby establishing a hardware-based root of trust—something lacking in most software-only implementations.

iii) Experimental Bootloader Design on STM32

This work presents the design of a Bootloader experimental platform centered around the STM32 microcontroller. The system is developed to support two key functionalities: boot loading and user interaction. A "multi-system storage" architecture is adopted to implement three distinct operational modes during the

boot phase. Additionally, the user interaction component includes multiple command levels with differing privileges to facilitate learning. The system is tailored for educational purposes, offering hands-on pr actice modules of varying complexity to accommodate learners at different levels. This helps students reinforce theoretical concepts, engage in exploratory development, and foster innovation.Published by Jigao Niu, Aiguo Wu, Mengmeng Liu, Chunhua Xu in *Proceedings of the Third International Conference on Advanced Manufacturing Technology and Manufacturing Systems (ICAMTMS 2024)*, Volume 13226, Article 132264N

Our system differs in focus by targeting real-world OTA use cases with hardware-based signature validation, but similarly explores a layered and modular bootloader design.

iv) Embetronicx STM32 Bootloader Framework

The Embetronicx community offers a step-by-step tutorial and GitHub repository on building custom bootloaders for STM32 devices. Their approach focuses on UART-based firmware updates and includes basic checksum validation, but lacks cryptographic security.

We extend this concept by integrating secure communication (via ESP32) and cryptographic signature validation using Trust M, making our design both robust and modular.

v) ARM TrustZone for Cortex-M33

The ARM Platform Security Architecture (PSA) and TrustZone offer a robust solution for secure boot and isolation on Cortex-M33 MCUs. TrustZone allows separation of secure and non-secure execution environments, but requires hardware support not present in STM32F4.

As our target platform lacks TrustZone, we implement a layered security mechanism using external components (ESP32 and Trust M) to emulate trusted execution and validation externally.

vi) Secure OTA via TLS and HTTP

Secure Over-the-Air (OTA) updates via HTTP over TLS are widely used in commercial embedded platforms. TLS ensures encryption and integrity, while

HTTP provides a lightweight protocol for message delivery. In our project, we use ESP32 as a TLS client using mbedTLS and HTTP to download the firmware securely. This isolates the STM32 from direct exposure to network threats while maintaining low resource usage on the microcontroller.

05 - Hardware and Software Requirements

i) Software

- STM32 Cube IDE
- STM32 CubeMX
- Eclipse Mosquitto
- Open SSL

ii) Hardware

The Proposed system configuration is as follows:

Sl. No.	ITEM	Model Description
01	Target Development Board	STM32F401RE NUCLEO Board
03	Processor	STM32F4
04	Secure element	Optiga Trust M
12	Network	IEEE 802.11
13	Server	Personal Laptop
17	OS	Ubuntu (Debian Based Linux)

iii) Communications Interfaces:

- The client devices must support **reliable internet connectivity** (Wi-Fi) to download firmware updates from the HTTP server.
- **Firmware size** is limited by the storage capacity of the client device. Firmware images should be optimized for smaller sizes without compromising security features.

06 - Technical Stack

Secure Firmware Update Architecture using HTTP and Hardware Root of Trust

The illustrated framework demonstrates a secure and robust method for delivering firmware updates to STM32-based embedded systems, leveraging HTTP protocol and hardware-based security. On the PC/Server side, a pre-compiled firmware binary (.bin) is segmented into smaller chunks and transmitted over a TLS-secured HTTP channel. This process is brokered via a secure HTTP server such as Mosquitto or AWS IoT Core, ensuring encrypted, authenticated delivery of firmware data.

The STM32 client device is equipped with an HTTP client that subscribes to the update topic and begins receiving firmware chunks. These chunks are then written to flash memory in a controlled manner. To secure the communication and update process, the client utilizes an Optiga Trust M secure element, which handles authentication, TLS key storage, and validation of digital credentials.

Once the firmware is received, the system invokes a verification phase. Here, either an authentication token or a digital signature embedded in the firmware is validated using credentials stored in the secure element. The bootloader is then responsible for performing cryptographic integrity checks, such as hash validation or signature verification, before executing the new firmware. This ensures that only trusted and untampered firmware is installed on the device, making it resilient against OTA attack vectors and unauthorized firmware injection.

This architecture provides a scalable and secure approach to firmware updates for embedded systems deployed in the field.

i) Microcontroller with USART

The microcontroller with Universal Synchronous/Asynchronous Receiver/Transmitter (USART) integration, appropriate for a report, consists of a strong set of hardware and software elements made for effective serial operation. The heart of hardware is microcontrollers like the STM32 Nucleo-F401RE or ESP8266, which have integrated USART peripherals that support common baud rates, data formats, and flow control for dependable data transfer.Security modules like Infineon's OPTIGATM Trust M, which improves the stack with cryptographic features including SHA-256 hashing and ECDSA signing for safe data transfer, are used in conjunction with these. In order to configure and handle USART activities, the software stack uses libraries like HAL (Hardware Abstraction Layer) or Arduino's Serial library, which are used in embedded C programming with frameworks like STM32CubeIDE or Arduino IDE. Furthermore, security modules are interfaced with communication protocols like I2C, and smooth development and testing are ensured by debugging tools like ST-Link or UART-to-USB converters. For projects needing authentication and real-time data transfer, this stack offers a safe and scalable base.\

ii) Bootloader in protected flash memory

A bootloader in protected flash memory is a little program that enables firmware updates or application execution and initializes the system. It is kept in a specific, secured area of a microcontroller's flash memory. To guarantee that it stays intact throughout updates or power cycles, the bootloader for microcontrollers such as the STM32 Nucleo-F401RE is usually pre-programmed by the manufacturer in a read-only or write-protected region of the flash. The bootloader can perform duties like UART-based firmware loading (e.g., via USART) or SHA-256 hashing to ensure the integrity of new code thanks to this protected region, which is often defined via option bytes or a specific memory segment (e.g., the first 16 KB on STM32 devices).
By storing cryptographic keys for bootloader update authentication using ECDSA signatures, the OPTIGATM Trust M can improve security by thwarting unwanted changes. In order to provide a safe and dependable boot procedure, you would integrate the bootloader with your application code using tools like

STM32CubeProgrammer and adjust the microcontroller's memory protection unit (MPU) or flash protection settings.

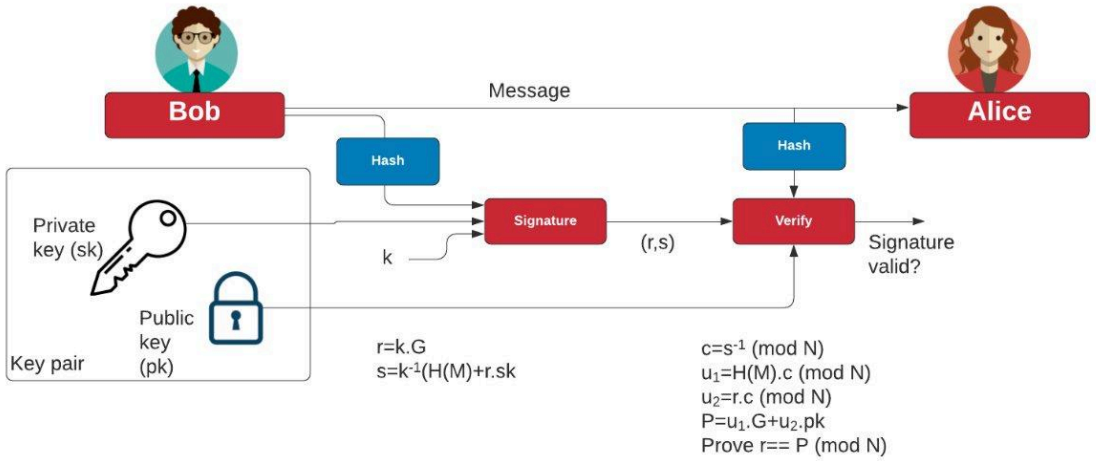
iii) Mbed-TLS SHA256 module for hashing of the firmware

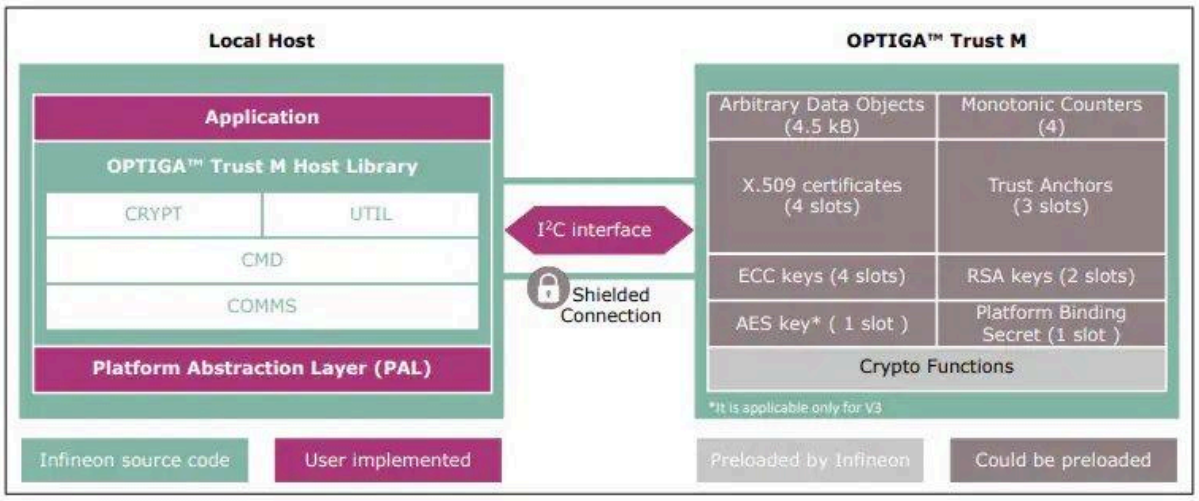
An STM32 Nucleo-F401RE microcontroller's firmware can be hashed using the Mbed TLS SHA-256 module, a powerful cryptographic library component. An open-source, portable TLS/SSL library called Mbed TLS provides a SHA-256 implementation that conforms to the FIPS 180-4 standard, which makes it appropriate for producing a 256-bit hash to guarantee firmware integrity. You would link against the Mbed TLS library and include the mbedtls/sha256.h header in your project, which is usually set up in an embedded development environment such as STM32CubeIDE, in order to integrate it.It is possible to initialize the module using mbedtls_sha256_init(), then set the context with mbedtls_sha256_starts(), process the firmware data in chunks with mbedtls_sha256_update(), and generate the final hash with mbedtls_sha256_finish(). If enabled, the hardware cryptography accelerator (such as the HASH peripheral) can be used by an STM32 to maximize performance. In order to authenticate the firmware during bootloader updates and prevent tampering, the SHA-256 hash can be signed with ECDSA when used in conjunction with the OPTIGA Trust M for secure key storage. As stated in the Mbed TLS manual, make sure that memory allocation and error handling are done correctly to accommodate the resource limitations of your project.

iv) ECDSA module (via OPTIGA Trust M) for digital signature verification

A safe hardware-based method for verifying digital signatures is offered by the ECDSA module via OPTIGATM Trust M. It is perfect for verifying firmware or data on a microcontroller such as the STM32 Nucleo-F401RE. Using elliptic curve cryptography (such as the NIST P-256 curve), the Infineon security controller OPTIGATM Trust M facilitates ECDSA operations and safely maintains private/public key pairs, guaranteeing that the private key is shielded from extraction. Using commands like OPTIGA_CRYPT_ECDSA_VERIFY, the firmware's received SHA-256 hash (for example, produced using Mbed TLS) and matching ECDSA signature are sent to the OPTIGA™ Trust M over I2C for verification.After comparing the signature to the embedded public key,

the module returns a success or failure state. This integration, which uses the STM32's I2C peripheral (PB6 for SCL, PB7 for SDA) to improve security during bootloader upgrades or data exchanges, needs to be configured using Infineon's OPTIGATM Trust M API or library. In order to comply with the security requirements of your project, make sure that the OPTIGATM Trust M datasheet is followed for initialization and error handling. Verification is usually carried out post-reception at 10:21 PM IST on August 04, 2025, or at any operating time.

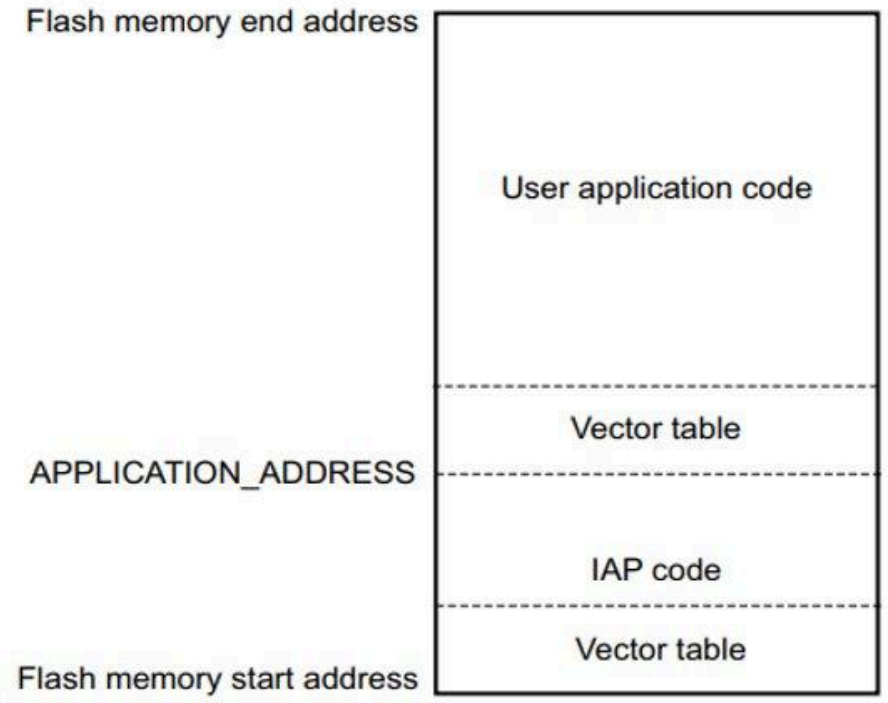




v) YMODEM protocol via USART for robust data transfer

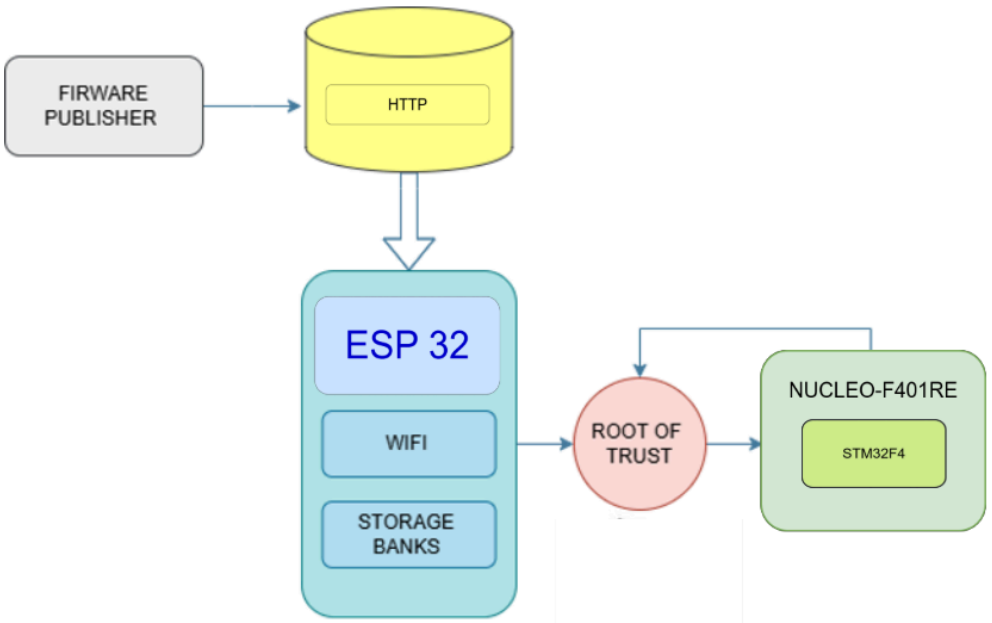
A strong and dependable data communication method, the YMODEM protocol via USART is appropriate for firmware updates or file exchanges on a microcontroller such as the STM32 Nucleo-F401RE. An modification of the

XMODEM protocol, YMODEM improves reliability over the USART interface (e.g., PA9 for TX, PA10 for RX on STM32) by supporting block-based file transfers with 1 KB blocks, cyclic redundancy check (CRC-16), error detection, and retransmission requests. It is perfect for bootloader apps because it has a filename block and supports multiple file transfers. In order to implement it, the STM32's USART peripheral must be configured using the STM32CubeIDE or HAL libraries to handle baud rates (such as 115200 bps), interrupts or DMA must be enabled for effective data handling, and a YMODEM library must be integrated. The transfer is started by the protocol with a "C" character. Data blocks containing sequence numbers and CRC are then sent, and the recipient (such as an STM32 or a PC terminal) either acknowledges or requests retransmission via NAK. When used in conjunction with OPTIGATM Trust M for firmware hashing using Mbed TLS SHA-256 and ECDSA signature verification.



07 - Block diagram

Flow Analysis of the Secure OTA Firmware Update Architecture



The block diagram outlines the secure firmware update mechanism implemented on the STM32 platform, highlighting a well-partitioned system that balances cloud-based communication, hardware-backed security, and microcontroller-level control. The process begins with the Firmware Publisher, typically a developer or automated CI/CD tool, which generates and signs a new firmware version. This firmware is published to an HTTP Broker, a lightweight messaging server designed for reliable communication in IoT systems. The broker acts as a central hub that distributes firmware updates to subscribed clients.

Once the firmware is available, the ESP32 module, functioning as the Wi-Fi-enabled edge device, connects securely to the HTTP Broker using TLS encryption, ensuring that the downloaded data is shielded against eavesdropping and tampering during transmission. The ESP32 temporarily stores the received firmware in its internal storage banks, acting as a buffer stage before the firmware reaches the final device. This approach reduces the flash memory

burden on the STM32 board and supports update resumption if communication is interrupted.

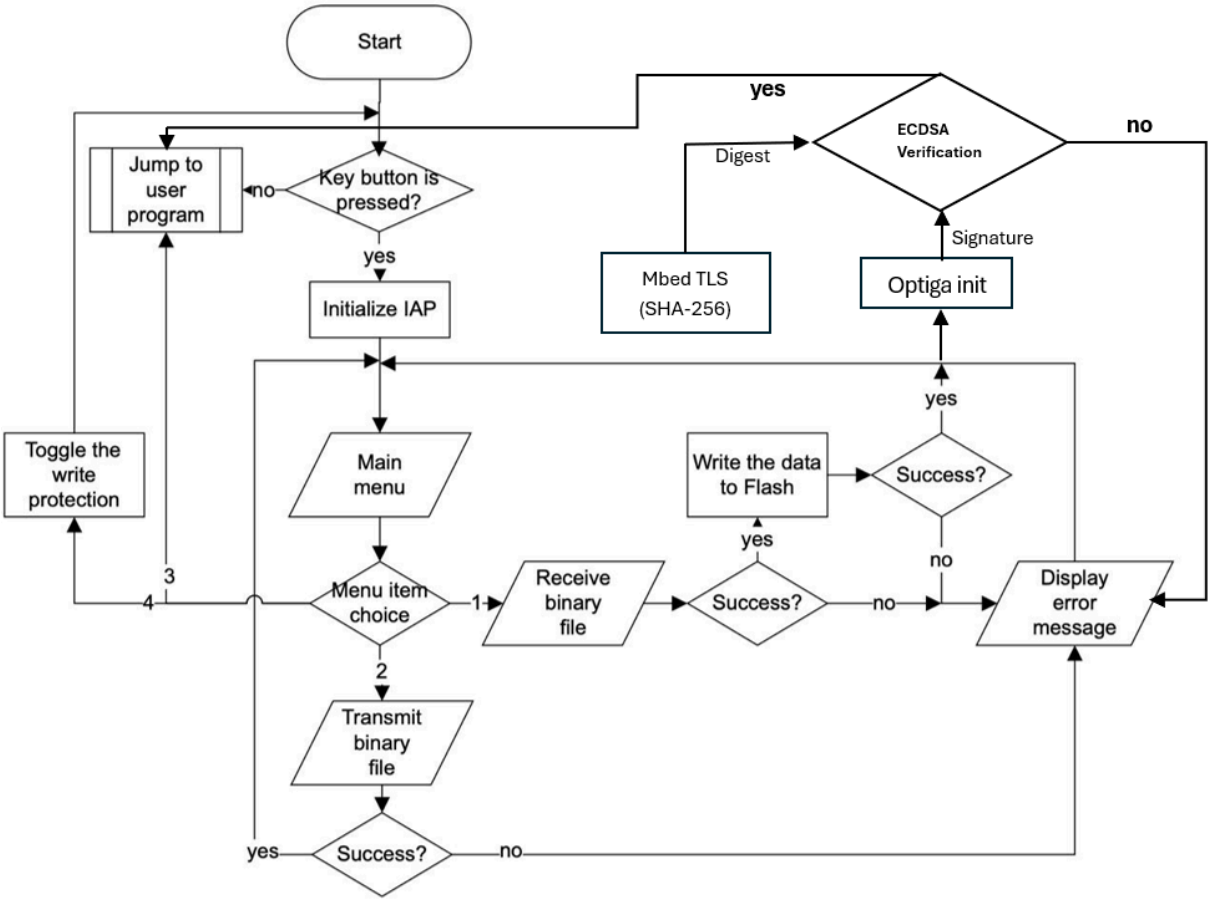
Before any firmware is transferred to the target device, the ESP32 forwards it to the Root of Trust — specifically, the OPTIGA™ Trust M secure element. This hardware security module is responsible for verifying the authenticity of the firmware by checking its cryptographic signature. If the signature verification fails, the process is halted, ensuring that no untrusted or tampered firmware can proceed to execution. On successful validation, the ESP32 initiates the handover of the verified firmware image to the STM32 Development Board, which houses the STM32F4 microcontroller.

The STM32F4, upon receiving the verified firmware, stores it in its flash memory and enters a secure boot phase. The minimal secure bootloader present on the microcontroller is responsible for performing integrity checks (e.g., CRC or signature match) on every boot cycle. If the integrity check passes, the microcontroller jumps to the application start address and executes the new firmware. If the check fails or if the firmware is corrupted, the system either halts or falls back to a previously validated firmware version (rollback protection, if implemented).

A key design advantage here is the air-gap logic, where the ESP32 acts as a security buffer between the internet and the MCU. The STM32 never directly connects to the internet; instead, it receives only verified and signed data through a trusted intermediary (ESP32 + Trust M). This not only reduces the attack surface but also adheres to the principle of least privilege and separation of concerns.

08 - Flow Diagram

i) Flow Analysis of Bootloader Firmware Handling Process



This flowchart outlines a secure firmware update process for an embedded system, integrating cryptographic verification and flash memory operations. The system begins execution by jumping directly to the user program, indicating a lightweight or secondary bootloader structure. A physical key button press serves as the entry point for initiating the firmware update routine, ensuring deliberate user intent before proceeding with sensitive operations.

Upon activation, the system initializes In-Application Programming (IAP), a method for modifying firmware without requiring a full system reset. This step prepares the hardware for flash memory writes, ensuring proper addressing and

avoiding conflicts with active processes. The subsequent stages involve cryptographic verification using Mbed TLS (SHA-256) for hashing and ECDSA (Elliptic Curve Digital Signature Algorithm) for signature validation. These steps ensure firmware authenticity and integrity before any write operations are permitted.

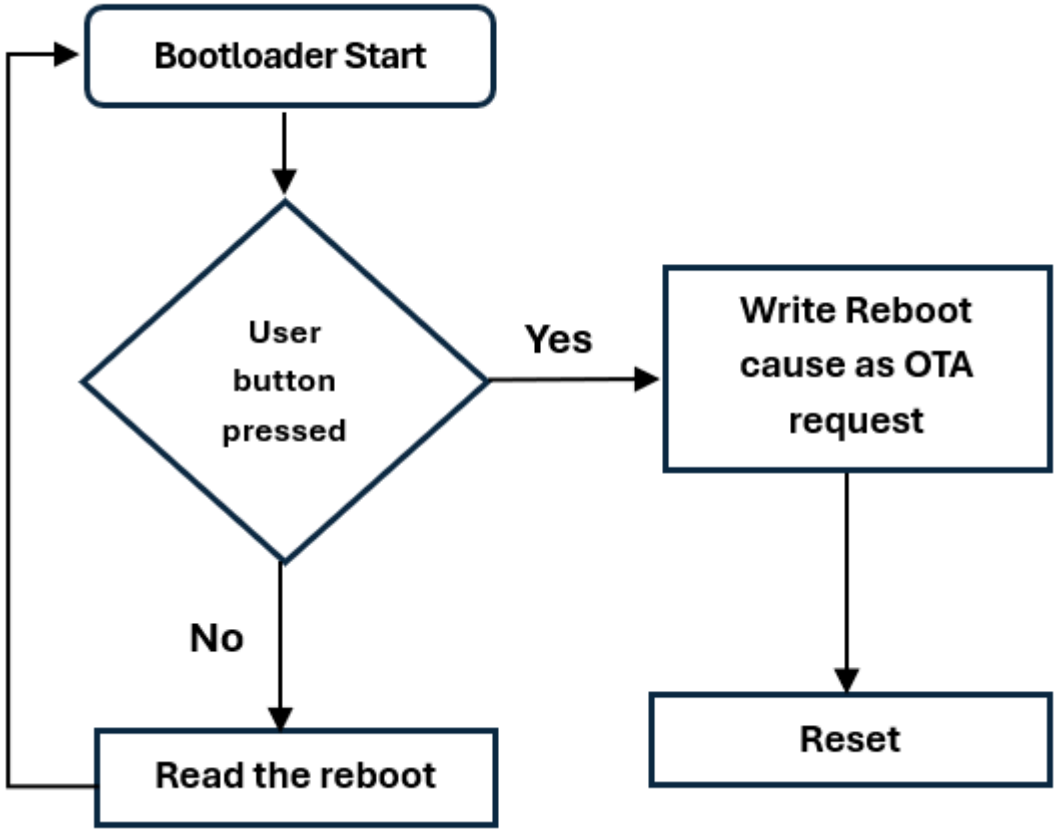
The Optiga initialization phase suggests the use of a dedicated security chip (e.g., Infineon's Optiga Trust) to handle secure key storage and cryptographic operations. Following this, the system toggles write protection on the flash memory, a critical step to prevent unauthorized modifications during the update process. The main menu is then displayed, offering options to receive or transmit a binary file, indicating bidirectional firmware update capabilities.

When receiving a binary file, the system writes the data to flash memory and checks for success. If the operation fails, an error message is displayed, ensuring user awareness. A similar verification occurs when transmitting a binary file, with success/failure paths maintaining system robustness. The flowchart's loop structure (e.g., returning to the main menu) implies a user-retry mechanism for failed operations.

The Mbed TLS (SHA-256) digest generation ensures firmware integrity by producing a fixed-size hash of the binary file. This hash is then validated using ECDSA verification, a public-key cryptography method resistant to tampering. The inclusion of Optiga hardware further strengthens security by offloading sensitive operations (e.g., key storage, signature verification) to a dedicated, tamper-resistant component.

The write protection toggle is a critical safeguard, preventing accidental or malicious flash corruption. This dual-state (enabled/disabled) mechanism aligns with best practices for firmware update security. The flowchart's explicit success/failure checks at each stage (e.g., flash write, transmission) reflect a fault-tolerant design, ensuring system stability even in edge cases.

ii) Application-Level OTA Request Initialization



Following the secure bootloader execution flow outlined in previous diagram,This represents the application-side logic that sets up the conditions for an OTA firmware update. While the bootloader is responsible for firmware validation and execution control, the application layer has the critical role of initiating OTA requests in a controlled and secure manner. This coordination ensures a seamless and secure handoff between runtime operation and firmware update mechanisms.

The sequence begins after the system has booted into a verified application. During runtime — typically under maintenance conditions or user instruction — the application checks whether the user button is pressed, which serves as a physical trigger for initiating an OTA update request. This decision is crucial because it gives the user manual control over when the firmware should be updated, reducing the risk of unintended overwrites.

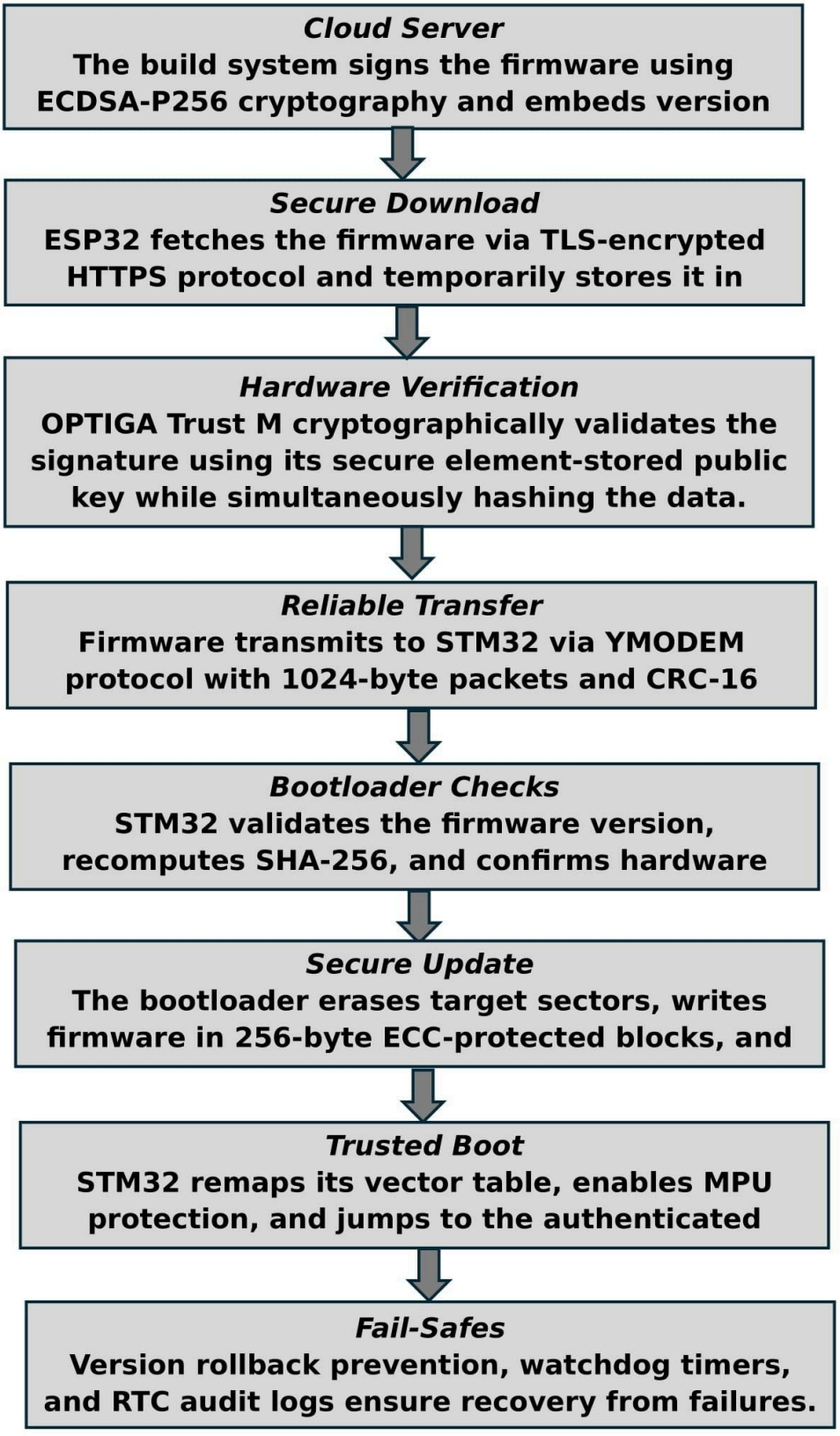
If the user button is pressed, the application proceeds to write a specific reboot cause flag, indicating that the next reboot should be interpreted by the bootloader as an OTA update request. This flag is stored in a persistent register or reserved memory area that the bootloader checks during startup (as shown in previous diagram). Once the flag is set, the system performs a controlled reset, effectively handing control back to the bootloader for the update operation.

If the user button is not pressed, the application bypasses the OTA preparation and continues normal operation. No flags are modified, and the system assumes a standard reboot or power cycle when it restarts.

The elegance of this design lies in its decoupled but coordinated approach: the application layer does not perform the update itself but instead acts as a gatekeeper, instructing the bootloader to initiate the OTA sequence through indirect signaling. This reduces the application's complexity and attack surface while still enabling firmware updates without requiring direct internet access on the main microcontroller.

Moreover, this separation of concerns between application and bootloader not only simplifies validation logic but also aligns with best practices in embedded security — such as minimal trusted code, clearly defined state transitions, and strong access control for update triggers. The system ensures that only intentional, user-approved update operations are carried out, with every step being cryptographically and logically verified.

iii) Overall Flow Diagram



The flow diagram illustrates the complete end-to-end secure firmware update and boot sequence designed for STM32-based systems, with a strong emphasis on cryptographic assurance, modular separation of responsibilities, and hardware-backed trust anchors.

The process initiates at the cloud layer, where the build system compiles the firmware, signs it using **ECDSA-P256** digital signatures, and embeds versioning metadata directly into the binary. This metadata includes information such as firmware version number, target hardware ID, and cryptographic hash summaries, establishing the basis for authenticity and rollback control. The signed image is then deployed to a secure cloud server, ready to be fetched by downstream embedded nodes.

On the client side, the ESP32-based OTA manager initiates the download over a **TLS-encrypted HTTPS** channel, ensuring end-to-end confidentiality and integrity during transmission. The firmware binary is stored temporarily in **SPIFFS** flash memory, isolated from STM32, thus offloading the network and cryptographic processing from the main microcontroller and reducing attack surface on the secure bootloader.

Once the firmware is downloaded, the ESP32 leverages the **OPTIGA™ Trust M** secure element to validate the digital signature. The Trust M performs this verification entirely in hardware using a stored public key, ensuring that neither the key nor the validation process is exposed to tampering. Simultaneously, the secure element computes a SHA-256 hash of the firmware for integrity confirmation, binding the signature validation tightly with content integrity.

After successful verification, the firmware is transmitted to the STM32 microcontroller over a UART interface using the **YMODEM protocol**, which provides reliable data transfer via 1024-byte packets and CRC-16 checks. UART DMA is employed to enhance transfer efficiency while minimizing CPU overhead. Upon receiving the complete image, the STM32 secure bootloader begins its internal validation process.

The bootloader independently verifies the firmware version, recomputes the **SHA-256 hash**, and ensures compatibility with the hardware configuration. Only upon passing these checks does it proceed to the update stage. During the secure update, the bootloader erases target sectors of internal flash and writes the new firmware in **256-byte blocks**, each protected by **ECC (Error**

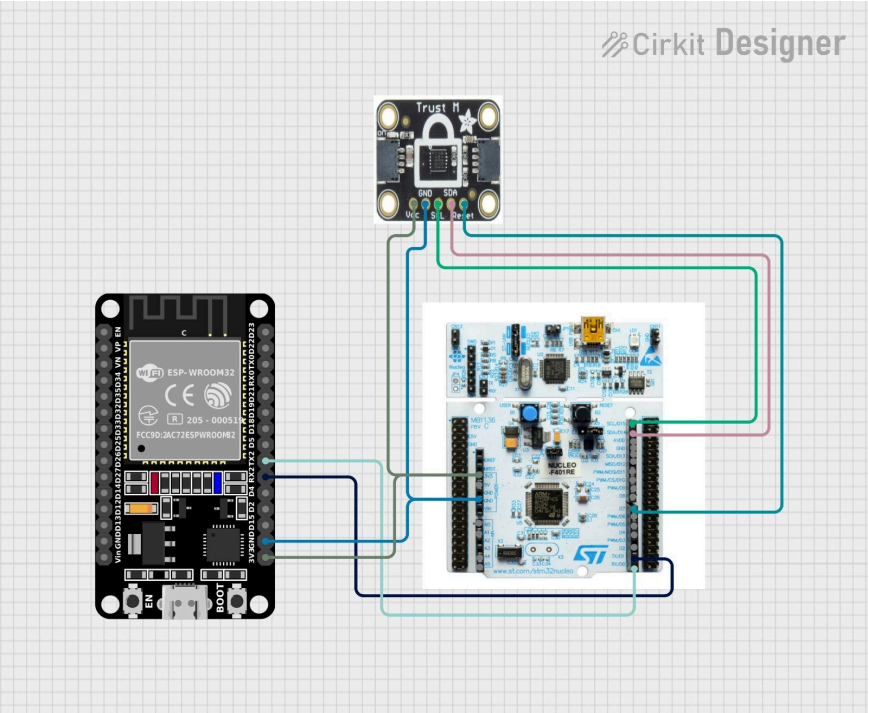
Correction Code). Post-write verification is performed for every block, ensuring storage integrity before execution.

After successful flashing, the STM32 performs secure boot procedures by remapping the **vector table**, enabling **Memory Protection Unit (MPU)** regions, and finally branching to the newly authenticated firmware. This ensures that runtime execution begins only from trusted code.

To support resilience and robustness, the system incorporates multiple fail-safe mechanisms. These include version rollback prevention using stored metadata, **watchdog timers** to recover from stalled updates or crashes, and **RTC-based audit logs** to record update activity for traceability. Together, these components ensure that the system maintains a verifiable chain of trust from firmware compilation to application execution, while providing secure, scalable, and recoverable firmware lifecycle management.

09 - Schematic Diagram

i) Hardware Interconnection Diagram



ESP32 ↔ STM32 Nucleo (UART Connection):

- ESP32 **TX2 (GPIO17)** → STM32 **RX (PA10 / USART1_RX)**
- ESP32 **RX2 (GPIO16)** ← STM32 **TX (PA9 / USART1_TX)**
- ESP32 **GND** ↔ STM32 **GND**
- ESP32 **3.3V** → STM32 **3.3V**

The UART-based communication interface between the ESP32 and the STM32 Nucleo board is established through a direct serial connection that ensures reliable bidirectional data transfer during the firmware update process. Specifically, ESP32's UART2 interface is utilized, with GPIO17 (TX2) connected to PA10 (USART1_RX) on the STM32, and GPIO16 (RX2) connected to PA9 (USART1_TX). This cross-connection aligns the ESP32's transmit and receive lines with the STM32's corresponding receive and transmit lines, forming a standard full-duplex serial communication link.

Both devices are referenced to a common electrical ground by directly connecting ESP32 GND to STM32 GND, ensuring signal integrity and eliminating potential floating ground issues. For logic level compatibility and

stable operation, 3.3V power from the ESP32 is routed to the STM32's 3.3V rail, maintaining uniform voltage levels across both platforms, as both operate natively at 3.3V logic. This configuration enables the STM32 bootloader to reliably receive firmware data via the YMODEM protocol over UART, using DMA for efficient handling, while the ESP32 manages data transmission with minimal CPU overhead. The simplicity and robustness of this hardware-level interface make it well-suited for secure, resource-constrained embedded update workflows.

OPTIGA Trust M ↔ STM32 Nucleo (I²C + Control Lines):

- Trust M **SCL** → STM32 **PB8 (I2C1_SCL)**
- Trust M **SDA** ↔ STM32 **PB9 (I2C1_SDA)**
- Trust M **VCC** → STM32 **3.3V**
- Trust M **GND** ↔ STM32 **GND**
- Trust M **RST** → STM32 **PA8**

The integration of the OPTIGA™ Trust M secure element with the STM32 Nucleo board is achieved through an I²C interface augmented by a dedicated reset control line, enabling secure communication and deterministic device initialization. The I²C bus connects Trust M's SCL line to STM32 pin PB8 (I2C1_SCL) and SDA to PB9 (I2C1_SDA), leveraging the STM32's hardware-accelerated I2C1 peripheral for efficient and reliable data exchange. This configuration allows the STM32 to initiate and manage transactions with the Trust M for cryptographic operations, such as ECDSA signature verification and secure key access.

Power delivery is handled through a direct connection from the STM32's 3.3V rail to the Trust M's VCC pin, ensuring voltage-level compatibility and stable operation, while a common ground (GND-to-GND) maintains signal reference integrity. Crucially, the Trust M's RST (Reset) line is tied to STM32 pin PA8, enabling the host to programmatically assert hardware resets to the secure element. This feature is essential for maintaining a known-good state, particularly after power cycles or in response to fault conditions.

Together, this schematic provides a robust and secure hardware foundation for leveraging the capabilities of the OPTIGA™ Trust M as a Hardware Root of Trust (HRoT), supporting high-assurance cryptographic functions within the STM32 secure firmware update and boot flow.

10 - Implementation and Deployment

i) Implementation Details

A. STM32 Bootloader Development

- Objective: Build a lightweight bootloader that validates firmware before executing.
- Platform: STM32F401RE MCU (512 KB flash)
- IDE: STM32CubeIDE
- Language: Embedded C
- Key Tasks:
    - **UART-Based Firmware Reception**
        - Uses the **YMODEM protocol** over UART to receive the firmware binary and associated metadata (e.g., version, digital signature).
    - **Flash Memory Management**
        - Resides in a **dedicated, write-protected flash region**.
        - Stores received firmware into a separate **application memory section** to ensure safe bootloader operation.
    - **Metadata Parsing**
        - Extracts critical metadata from the firmware payload:
            - **Version number**
            - **Firmware length**
            - **Digital signature**
    - **Firmware Validation (Secure Boot)**

        - Computes the **SHA-256 hash** of the received firmware using mbedTLS.
        - Sends this hash along with the firmware's signature to **OPTIGA™ Trust M** for verification.
        - In initial stages, allows **hardcoded key-based validation** for debugging and testing.
    - **Decision & Execution**
        - If verification succeeds:
            - Flashes the firmware into application memory.
            - **Performs a jump** to the application start address.
        - If verification fails:
            - Remains in bootloader mode and may initiate retry or fallback logic.

B. ESP32 OTA Client

- **Objective: Offload cloud communication and Dual Bank Storage to ESP32.**
- **Platform:** ESP32 Dev Board
- **IDE:** Arduino IDE
- **Libraries:**
    - `WiFi.h` – Connect to Access Point (AP) or Router
    - `SPIFFS.h` – Store firmware binaries in internal flash
    - `ESPAsyncWebServer.h` – Run HTTP server for file upload & control
    - Custom UART routines – Send `.bin` to STM32 via **YMODEM**
- **Functions:**
    - Connect to secure Wi-Fi.
    - Authenticate and download firmware via HTTPS .

- Forward binary data to STM32 with **YMODEM over UART** to transmit firmware.
- Handle retransmissions and acknowledgments.

C. OPTIGA™ Trust M Integration

- Objective: Provide hardware-based firmware signature validation.

- Interface: **I2C communication** between STM32 and OPTIGA Trust M.
- **Libraries:**

  - optiga_util.h — for Trust M utility services (opening application, reading data objects, managing OIDs)
  - optiga_crypt.h — for ECDSA signature verification and cryptographic operations
  - pal.h — platform abstraction layer to connect STM32 GPIO/I2C to Trust M
  -

- **Process:**

  - **Secure Public Key Storage**
    - A public key (typically ECDSA) is securely pre-provisioned or loaded into a protected Object Identifier (OID) inside Trust M.
    - This public key is **never exposed to the MCU** and is used solely within Trust M for signature verification.
  - **Signature Verification**
    - Once the STM32 calculates the **SHA-256 hash** (digest) of the received firmware (via mbedTLS), it passes this digest and the firmware's digital signature to Trust M.
    - The optiga_crypt_ecdsa_verify() function is called with:
      - The computed **firmware hash**
      - The **digital signature**
      - The **OID reference** of the pre-stored public key
  - **Hardware-Based Decision**
    - Trust M performs the **ECDSA signature verification** entirely within its secure element.

- If the signature is valid for the given digest and public key:
  - Trust M returns a **success code** to STM32.
  - This confirms the firmware came from a trusted source.
- If the verification fails:
  - The firmware is considered **tampered or unauthorized**, and execution is aborted.

ii) Deployment Plan

A. Flash Memory Layout

Region	Address Range	Size	Purpose
Bootloader	0x08000000 – 0x08003FFF	16 KB	Performs validation and jump
App Metadata	0x08004000 – 0x08004FFF	4 KB	Version, signature, status flags
Application	0x08005000 – 0x0801FFFF	~108 KB	Verified user application

11 - Unique Selling Propositions (USP)

i) Hardware-Based Root of Trust via OPTIGA™ Trust M Secure Element

Our secure firmware update solution is anchored by a **true hardware Root of Trust**, made possible by integrating the **Infineon OPTIGA™ Trust M** secure element. This approach sets our design apart from conventional software-only bootloaders that store cryptographic keys in internal MCU flash — a method vulnerable to physical extraction and memory attacks.

- **Tamper-Resistant Hardware Key Storage**
  Private keys are stored exclusively within the Trust M chip's **isolated secure memory**, completely inaccessible to the STM32 or any external bus. This eliminates risks associated with key leakage via firmware bugs, memory dumps, or JTAG attacks.
- **On-Chip Signature Verification Using ECC**
  Trust M handles all **Elliptic Curve Digital Signature Algorithm (ECDSA)** operations internally. The STM32 simply passes the firmware and signature to the chip, and receives a trusted validation result — **no secret ever leaves the secure element**.
- **No Onboard Cryptographic Code Needed on STM32**
  By offloading heavy cryptographic operations to Trust M, the STM32 is relieved of complex math libraries and lengthy key parsing logic, further **reducing firmware size, processing time, and power consumption**.
- **Industrial-Grade Security Assurance**
  The OPTIGA™ Trust M is certified under high-assurance standards, including:
  • ISO/IEC 11889 (TPM standard)
  • EAL6+ (Common Criteria)
  • RoHS and platform-agnostic support across ARM MCUs
- **Defense Against Physical Attacks**
  The chip features built-in protections against voltage, timing, glitch, and side-channel attacks — offering security guarantees **not possible in purely software-based solutions**.

This makes our architecture **trustworthy for industrial, automotive, and IoT deployments**, where **firmware authenticity and integrity** are paramount and **physical access cannot be ruled out**.

ii) Modular Security Architecture

Our design separates key responsibilities among different devices:

- STM32 handles boot-time validation and execution.
- ESP32 manages TLS-secured firmware download.
- Trust M ensures authenticity of firmware.

This reduces the burden on the main MCU and allows for future scalability, such as swapping the communication module without altering the secure boot logic.

iii) Robust and Reliable Firmware Transfer via YMODEM Protocol

Unlike raw UART-based firmware updates that are prone to transmission errors and lack structure, our implementation leverages the **YMODEM protocol** between the ESP32 and STM32 for transmitting the firmware binary.

- **Structured Transfer with Metadata**: YMODEM encapsulates firmware in a structured format, including the filename and size, ensuring STM32 receives all necessary context before flashing begins.
- **CRC-Based Integrity Checks**: Every packet is validated using CRC, ensuring that corrupted data is automatically rejected and retransmitted.
- **Automatic Retransmission and Flow Control**: If any packet fails verification, the sender (ESP32) retries transmission, significantly increasing robustness over noisy or unstable UART links.
- **Chunked Transmission**: Firmware is sent in fixed-size chunks (1 KB), which simplifies buffer management on the constrained STM32 and prevents memory overflows.
- **No Dependence on PC Tools**: This approach is fully embedded and autonomous — firmware updates can be performed without needing any desktop software like TeraTerm or XMODEM utilities.

This makes the system **highly reliable and resilient**, especially for use in field-deployed or remote embedded systems where stability and integrity of updates are critical.

iv) Flash-Efficient Secure Bootloader for Resource-Constrained MCUs

Our STM32-side implementation is engineered to occupy **as little flash memory as possible**, without compromising on essential security or boot features.

- **Compact and Efficient Design**: The secure bootloader is custom-built with no dependency on heavyweight libraries (unlike SBSFU, which typically adds 80–100 KB overhead).

- **Lightweight YMODEM Receiver**: Includes only essential components to receive and store firmware efficiently over UART.

- **Delegated Signature Verification**: Offloads complex cryptographic operations like ECDSA to the external Trust M chip, minimizing STM32 flash usage and computational burden.

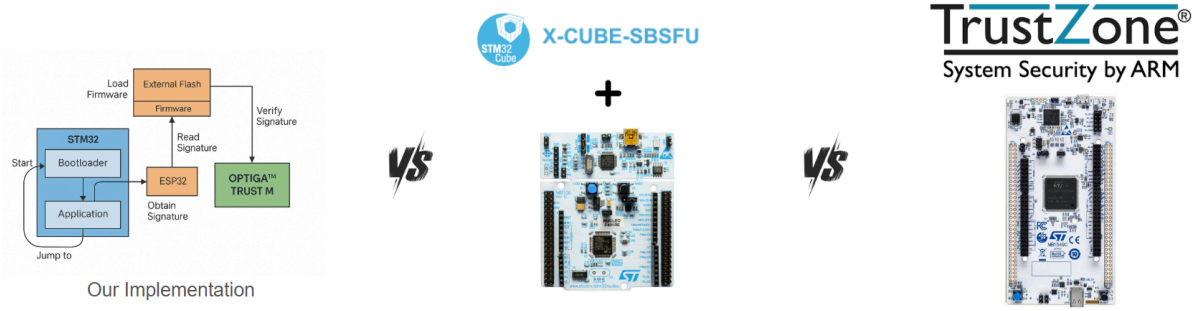
- **Optimized Flash Usage**: With a total flash size of 128 KB on STM32F401RE, our bootloader footprint remains ≤ **64 KB**, preserving at least 50% of flash for the main application.

- **Viable for Cost-Sensitive Applications**: Because of its small footprint, the solution is ideal for **industrial, consumer, or IoT devices** using **low-cost MCUs** that would otherwise be unable to support secure OTA updates.

This lean bootloader architecture strikes a balance between **security**, **reliability**, and **flash efficiency**, making it suitable for high-volume production where every kilobyte matters.

12 - Comparative Study

Comparative Analysis of Secure Firmware Update Architectures



The diagram presents a side-by-side comparison of three different approaches to secure boot and firmware update in embedded systems: the custom implementation developed for the STM32 platform, STMicroelectronics' X-CUBE-SBSFU framework, and ARM's TrustZone technology for Cortex-M devices. Each of these solutions aims to ensure firmware authenticity, integrity, and secure execution, but they differ significantly in architecture, complexity, and hardware dependencies.

On the left, the custom implementation demonstrates a modular architecture built using standard STM32 microcontrollers, an external ESP32, and the OPTIGA™ Trust M secure element. In this setup, the ESP32 handles firmware retrieval from external sources (e.g., HTTP broker), stores it temporarily, and obtains its digital signature. The signature is then validated using Trust M, which serves as a dedicated hardware root of trust for cryptographic operations. Only after successful verification is the firmware handed off to the STM32 bootloader, which then installs and launches it. This distributed model effectively decouples secure communication, storage, and execution, enabling granular control and enhanced flexibility while still offering robust security through hardware-backed validation.

In contrast, the middle section of the diagram highlights the X-CUBE-SBSFU (Secure Boot and Secure Firmware Update) solution provided by STMicroelectronics. This is an officially supported software expansion package

for STM32 microcontrollers that integrates secure boot, secure firmware installation, and version management into a tightly coupled firmware package. It often requires specific STM32 hardware features such as the Memory Protection Unit (MPU), and it supports encryption, anti-rollback mechanisms, and platform attestation. While it offers comprehensive protection out-of-the-box, it is often perceived as more rigid and resource-intensive, which can be a limiting factor in constrained environments.

On the right, ARM's TrustZone for Cortex-M33 introduces a hardware-enforced separation between secure and non-secure execution environments. This allows sensitive operations — such as key handling or firmware validation — to occur in an isolated secure world, even while running on the same core as non-secure application logic. TrustZone provides strong isolation at the hardware level and is highly suited for systems requiring rich OS-level security models. However, it demands microcontrollers with built-in TrustZone support, such as those in the STM32L5 or STM32U5 families, and may be excessive for applications with simpler security requirements.

Overall, the custom implementation strikes a balance between lightweight design and robust security by leveraging off-the-shelf components and discrete trust anchors like the Trust M. Unlike SBSFU and TrustZone, which are more tightly integrated and hardware-specific, this approach offers high adaptability, clearer separation of roles, and easier debugging. It is especially beneficial in IoT and field-deployed systems where OTA updates must be secure, resilient, and hardware-efficient — all while avoiding complex or proprietary ecosystems.

13 - Conclusion

In this project, we successfully designed and partially implemented a modular, secure boot and firmware update framework for the STM32 platform, addressing the growing need for robust, remote firmware management in embedded and IoT devices. With the increasing deployment of edge devices in untrusted or physically inaccessible environments, ensuring secure over-the-air (OTA) updates has become a critical requirement. Our solution was crafted to meet these demands with a lightweight architecture that combines strong security principles, efficient resource usage, and high flexibility.

At the heart of the system lies a minimal secure bootloader running on the STM32 F401-RE microcontroller. This bootloader ensures that only verified and authenticated firmware is executed, thereby protecting the system from unauthorized or malicious updates. The update mechanism is decoupled from the MCU and is handled by an external ESP32, which securely communicates with a cloud-based HTTP broker over TLS. This separation of responsibilities ensures a reduced attack surface, as the STM32 never directly connects to the internet.

Crucially, our framework incorporates OPTIGA™ Trust M, a dedicated hardware root of trust, for storing cryptographic keys and performing signature validation. This enhances security by providing hardware-backed protection against tampering, side-channel attacks, and key extraction. The dual-slot architecture and CRC validation further ensure firmware integrity and offer rollback protection, making the system resilient in case of failed or corrupted updates.

Compared to industry-standard solutions like X-CUBE-SBSFU and ARM TrustZone, our design offers a flexible and component-level approach. While those solutions are robust, they come with hardware constraints or complexity that may not suit all use cases. Our implementation strikes a practical balance by utilizing readily available components and emphasizing modularity, making

it easier to adapt across various STM32-based platforms without compromising on core security requirements.

In conclusion, this project lays a strong foundation for implementing secure and scalable firmware update mechanisms in embedded systems. Future work will focus on completing the integration of flash protection, rollback prevention, and performance benchmarking. By combining secure communication, hardware-based cryptography, and minimal trusted code, our framework aligns with modern IoT security standards and is well-suited for real-world deployments.

14 - References

I. Secure Bootloader and Firmware Update (STM32 and General)

[1] Embetronicx, *Custom STM32 Bootloader Project*, GitHub Repository, [Online]. Available: https://github.com/Embetronicx/STM32-Bootloader

➔ **Demonstrates a minimal YMODEM-based custom bootloader for STM32 MCUs.**

[2] STMicroelectronics, *X-CUBE-SBSFU: Secure Boot and Secure Firmware Update*, [Online]. Available: https://www.st.com/en/embedded-software/x-cube-sbsfu.html

➔ **Official STM32Cube package providing secure boot and firmware update framework.**

[3] STMicroelectronics, *AN4657 - Application Note: STM32 in-application programming (IAP) using the USART*, [Online]. Available: https://www.st.com/resource/en/application_note/dm00122600-stm32-in-application-programming-iap-using-the-usart-stmicroelectronics.pdf

➔ **Detailed description of USART-based IAP and protocol support.**

[4] STMicroelectronics, *RM0090 - STM32F401 Reference Manual*, [Online]. Available: https://www.st.com/resource/en/reference_manual/dm00096844-stm32f401xe-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

➔ Core hardware-level information and boot configuration registers.

[5] STMicroelectronics, *AN2606 - STM32 Bootloader Protocols*, [Online]. Available: https://www.st.com/resource/en/application_note/dm00038536-an2606-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf

➔ Covers supported bootloader protocols and memory mapping on STM32.

II. Cryptographic Libraries and Platform Security

[6] ARM, *Platform Security Architecture (PSA)*, [Online]. Available: https://developer.arm.com/architectures/security-architectures/platform-security-architecture

➔ ARM's official security framework including TrustZone for Cortex-M devices.

[7] MbedTLS Project, *mbedTLS: Open Source SSL/TLS for Embedded*, GitHub Repository, [Online]. Available: https://github.com/Mbed-TLS/mbedtls

➔ Lightweight TLS/SSL library used for encryption and secure communication.

[8] Infineon Technologies, *OPTIGA™ Trust M Secure Element*, [Online]. Available: https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-trust/optiga-trust-m/

➔ Hardware-based secure element used for ECDSA verification and secure boot.

[9] Hackster.io, *Secure Bootloader for STM32 Using OPTIGA Trust M*, [Online]. Available:

https://www.hackster.io/news/secure-bootloader-for-stm32-using-optiga-trust-m-c104c7

➔ Community-driven project showcasing secure boot using hardware RoT.

III. OTA Update Mechanisms and ESP32 Integration

[10] Particle.io, *OTA Firmware Update Tutorial*, [Online]. Available: https://docs.particle.io/tutorials/device-os/firmware-updates/ota-updates/

➔ Concepts and mechanisms of OTA updates in resource-constrained IoT devices.

[11] Arduino-ESP8266 Team, *OTA Updates for ESP8266 (NodeMCU)*, [Online]. Available: https://arduino-esp8266.readthedocs.io/en/latest/ota_updates/readme.html

➔ Practical documentation for implementing OTA updates in ESP-based devices.

[12] Loboris, *ESP32 YMODEM UART Transfer Example*, GitHub Repository, [Online]. Available: https://github.com/loboris/ESP32_ymodem_example

➔ Implementation of YMODEM protocol on ESP32 for UART-based binary transfer.

IV. STM32 Microcontroller Internals

[13] STMicroelectronics, *RM0090 - STM32F401 Reference Manual*, [Online]. Available: https://www.st.com/resource/en/reference_manual/dm00096844-stm32f401xe-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

➔ Technical overview of the STM32F401RE, including flash memory, bootloader behavior, and IAP support.

[14] Embetronicx, *STM32 Reset and Boot Sequence Explained*, [Online]. Available: https://embetronicx.com/tutorials/microcontrollers/stm32/reset-sequence-in-arm-cortex-m4/

➔ Tutorial describing the vector table, reset handling, and startup logic for STM32 Cortex-M4.

[15] G. Dongyuan, F. Xiaojie, C. Aijun, and L. Jin, *Design of Bootloader experimental system based on STM32*, Experimental Technology & Management, vol. 36, no. 11, pp. 89–92, 2019. [Online]. Available: https://doi.org/10.16791/j.cnki.sjg.2019.11.022

➔ Description of a Bootloader experiment system using STM32 with multi-system storage and privilege-based interaction design for student training and innovation development.