Secure Bootloader with RBF

By: Mohamed Abo Khalil

Introduction

What is a Bootloader?

- **Definition:**
  - **A bootloader is a small program that initializes hardware and loads the main application firmware on a microcontroller.**

Purpose:

- **it allows for programming and updating the application code after the initial firmware installation.**
- **Ensures the Integrity of the Application**

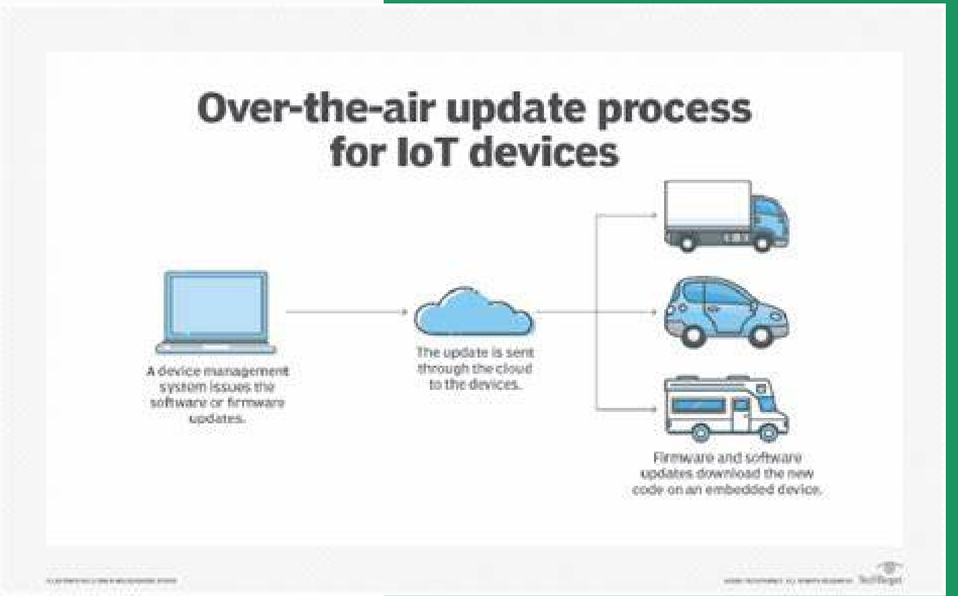
Project Overview

The primary goal of this project is to design and implement a reliable bootloader for the STM32F103C8T6 microcontroller. The bootloader will enable:

- **Firmware Updates:** Seamlessly update the application firmware through UART communication.
- **Integrity Verification:** Ensure that the firmware being uploaded is genuine and free from corruption.
- **Robust Error Handling:** Implement mechanisms to handle failures during flash operations, ensuring system reliability.
- **Security Features:** Protect the firmware and ensure only authorized updates are applied.

Key Features

- **Firmware Update Mechanism:**
- **Support for Over-the-Air (OTA)**
  - Ability to manage firmware sizes and handle partial updates.
- **Application Verification:**
  - Implement CRC (Cyclic Redundancy Check) and digital signatures (using symmetric encryption)
  - to verify the integrity using SHA256 and authenticity of the firmware before installation.
- **Rollback Mechanism:**
  - Provide a fallback option to revert to the previous stable firmware version in case of update failures.
  - Safe State Restoration: The system can restore to a stable state without user intervention, ensuring minimal downtime and maintaining system functionality.



Key Features

- **Security Measures:**
  - **Ensure Confedentiality by using** of AES encryption to secure firmware during transmission to protect against eavesdropping.
  - Authentication mechanisms to verify the sender’s identity before accepting a firmware update.
- **Authorization Mechanism to Read from Memory using seed**
- **Logging Mechanism and Debugging using UART**

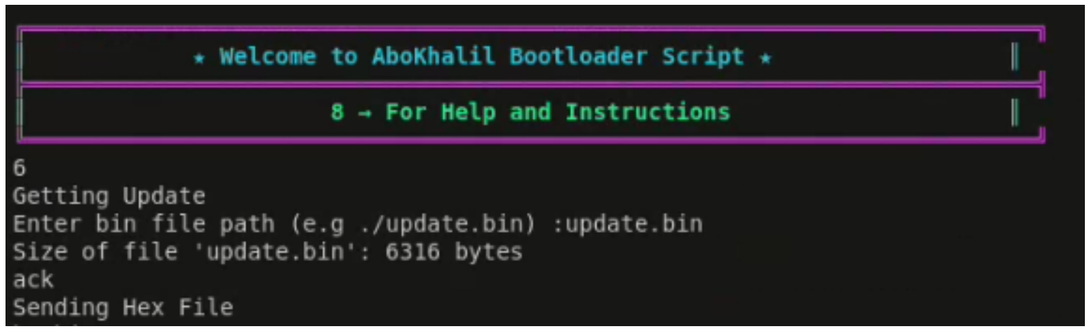
Bootloader Working Process

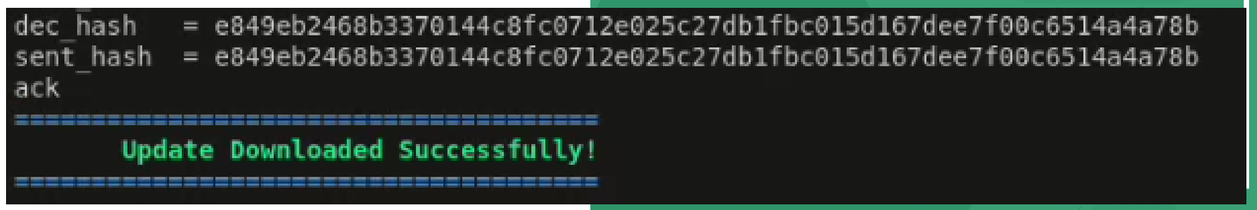
1. Initialization
2. Check APP Integrity
3. **if there's update** : Update and Verify and Jump
   if not :
   if **Valid App**: Jump to It
   if **not**: Check for rollback version
   if **there's no roll back version**
   stuck on Menu waiting for update from Authorized Entity

Note Supports Also Reading and Writing to Memory from Authorized Entity Using Seed Mechanism

Mechanism for Updating Application Firmware

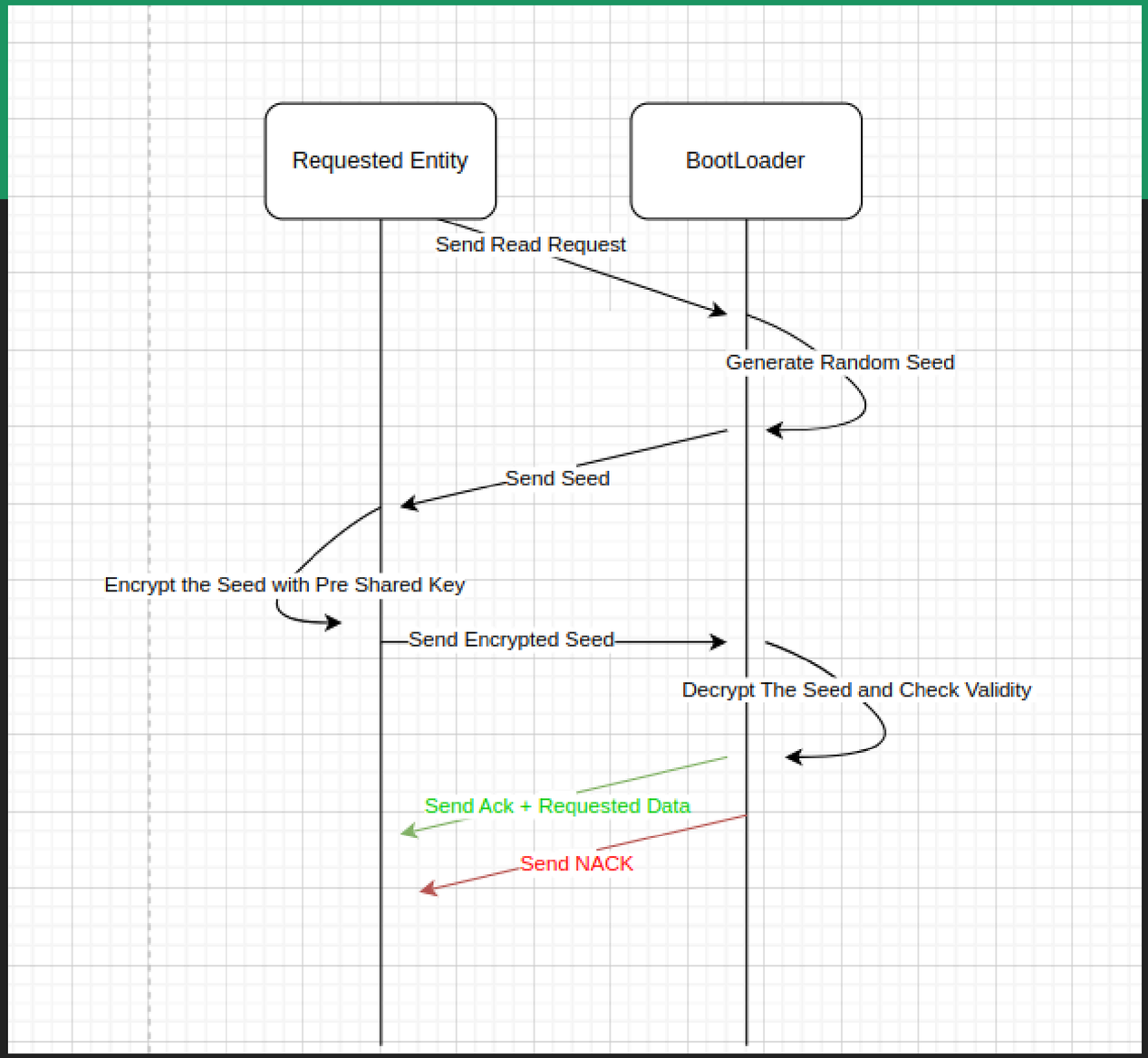
- Start Update Session: prepare for firmware update.
- Retrieve Encrypted Hash: Recieve the AES-ECB encrypted hash - SHA256 from Authenticated Entity
- Receive and Decrypt Data: Receive firmware chunks via UART, decrypt them, and store in decrypted_data buffer.
- Write to Flash Memory
- Hash Accumulation: Accumulate hash of received decrypted data.
- Final Hash Verification: Compare calculated hash with the received encrypted hash to ensure integrity.
- Update Metadata: Update application size and hash using Set_ValidUpdate().
- End Update Session: Clean up cryptographic contexts with End_Updating_Session().





Reading Memory Mechanism

- Verifies the provided address
- Initializes the AES decryption context using the pre-defined read_key.
- Generates a random seed (16 bytes) to enhance security.
- Transmit Seed:
- Sends the generated seed to the host via UART for further processing.
- Receive Encrypted Seed:
- Waits to receive the encrypted version of the seed from the host.
- Seed Decryption:
- Decrypts the received encrypted seed using AES in ECB mode to verify the authorization.
- Authorization Check:
- Compares the decrypted seed with the original seed:
  if Authorized Return Memory Bytes
  Else Send NACK.

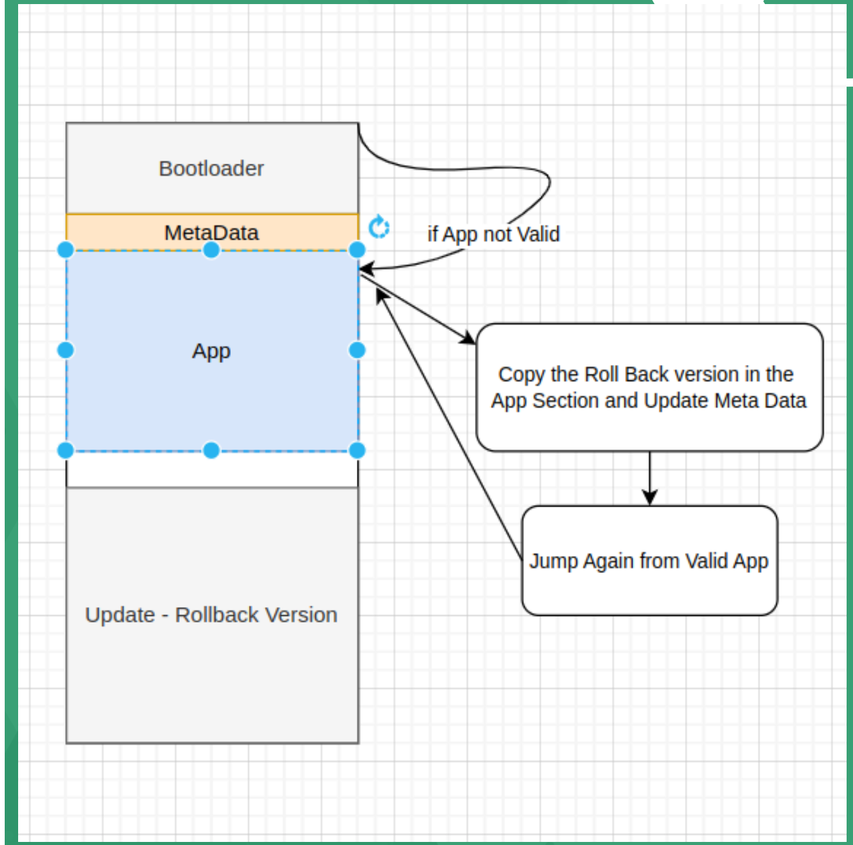


Roll Back Mechanism

Problem: Single Bank MCU

Solution:

- Version Tracking:
  - Each firmware version has it’s own metadata to keep track of the current and previous versions.
- Dual Firmware Storage:
  - Store two firmware images in separate memory regions:
    - Current Firmware: The version currently running on the device.
    - Previous or updated Firmware: The last known stable version.
- Firmware Update Process:
  - During an update, the bootloader first writes the new firmware to a designated area in memory.
  - After writing, the bootloader verifies the integrity of the new firmwar using SHA-256 in mbedtls library.
- Validation Check:
  - If the new firmware passes the validation copy it to the desired app location to avoid the husstle of VTOR and Linker Script Editing Each update.
  - If validation fails, maintain the current firmware and work with normal app.
- Rollback Logic:
  - In the bootloader, implement a check to determine if the current firmware is valid:
    - If the device fails to boot or operates incorrectly after an update, switch back to the second copy of firmware version.
  - The rollback process include:
    - Change App and Updated App MetaData and Store it in Flash.



All Supported Commands

```
8
get help cmd
ack
---------------------------------------
1.  Get Bootloader Version ID           --> 1
2.  Get Chip Identification Number      --> 2
3.  Read Memory Contents                --> 3
4.  Erase Flash Memory                  --> 4
5.  Get Flash Read Protection Level     --> 5
6.  Get Application Update              --> 6
7.  Jump to Application Code            --> 7
8.  Help                                --> 8
---------------------------------------
```

- Get Help CMD

```
1
get version id
ack
Vendor ID: 1F
BootLoader Version: 1:0:0
```

- Vendor iD - Version

```
tx_enc_seed: e5e89171064b9c16e1a8a60efa1335bd
rx_enc_seed: e5e89171064b9c16e1a8a60efa1335bd
ack
AUTHORIZED !!!!
0x00 0x50 0x00 0x20 0x31 0x0c 0x00 0x08 0xd9 0x0b
```

- Reading Memory

```
5
Getting RDP Level
ack
RDP level is : 0
```

- Read Protection Level

```
4
Erasing Flash
STM32f103c8t6 MCU Medium Density has 64 page starting from page 0 to 63
Enter Starting Page: 26
Enter Number of Pages to erase: 38
start:b'\x1a',num:b'&'
ack
Erase Finished ....
```

- Erase Flash

```
7
jumping to app
STM32f103c8t6 MCU Medium Density has 64 page starting from page 0 to 63
Enter App starting page: 27
```

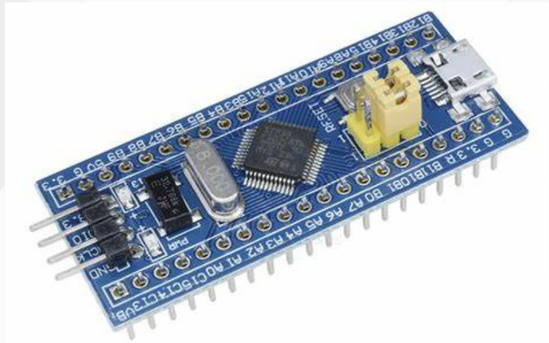
- Jump to Entire Page

```
calc_hash  = 70d45323a54849f91ec3a793ebc93cf37949d5095ad1e5b2fb3bce0ac2956
ack
=======================================
       Update Downloaded Successfully!
=======================================
```

- Update the APP

Deliverables

Developed Python Script to Communicate with BL



Tested Bootloader Project Files

Doxygen Documentation for Bootloader & CRC files

Chellenges and Solutions :

1-Problem: Asymmetric Encryption like ECDSA & ECDH Algorithms for Digital Signature and Key Sharing has very large footprint as stm32f10c8t6 flash size is 64kb

Solution: made Digital Signature using Preshared symmetric key AES which is shared only with authenticated entites.

2- Problem: HAL CRC32 give different result than crc lib in python script

Solution: Studed How CRC32 Works and with Help of Chatgpt i could make Needed Functions.

3- Problem: Mis-Synchronization of BL with python Script

Solution: Made Synchronization Ack

4- Problem: Shifted Recieved Hash

Solution: Flush UART Buffer before recieve the hash

Future Work

Ensure Authentication using PreShared Asymmetric Keys

Digital Signature Using ECDSA

Share Symmetric Key using ECDH

Use AES-GCM and be familliar with it’s operation, authentication tag, etc.

Fix any Bugs found in the bootloader

Thank You