Secure Avionics Flight Firmware Installation Routine System Design

MITRE eCTF 2022
Team **Cacti**
University at Buffalo

1 Introduction

This section presents the entities and communication channels in the system.

1.1 Entities

The following summarizes the entities in the system.

- Host is a general-purpose computer in the secure facility, which is responsible for generating protected firmware and configuration data images.
- A SAFFIRE bootloader is the main entity of the system to ensure the secure loading of flight configuration data and firmware with version updates on the device. It receives commands from the host to boot the firmware, put firmware on SRAM and handover the execution. Moreover, the bootloader is also responsible for providing a readback feature of the firmware and configuration data to an authenticated host.
- The firmware in the avionic device contains the logic to control the flight. The firmware gets the flight itinerary from the configuration files. As the flight travel is dependent on this logic and data the bootloader needs to ensure that no malicious code or data is loaded on the device.
- EEPROM on the Tivaware device is a hardware component accessible to the bootloader, which can be used to store data with access permissions.

2 Attack Models

The attackers can carry out the following attacks:

- Attackers may install custom or modified firmware and configurations to compromise the flight.
- Attackers may install old and buggy version of the firmware.
- Attackers may read firmware and configuration via readback function.
- Attackers may attain the password via side-channel attack.
- Attackers may access sensitive data or disable hardware protection via hardware trojan installed on FLASH.

3 Our Design

The host H has its private key sk_k stored in the host secrets file. The bootloader can access the associated public key pk_k and three AES256-GCM keys for firmware (k_f), configuration (k_c), and version number (k_v) protection, which are stored in the EEPROM. EEPROM also stores additional data or secrets that the bootloader needs at run-time. To protect the EEPROM we utilize EEPROM block hiding feature.

set the flash controller cannot access the flash

MPU is configured to 1) set the FLASH as eXecute-Never (XN) except for the bootloader region, 2) ~~set the FLASH as read and write (RW) for the bootloader only,~~ 3) ~~set the bootloader region as eXecute Only Memory (XOM).~~

ENC and DEC stand for symmetric encryption and decryption with AES256-GCM. AENC and ADEC stand for asymmetric encryption and decryption with RSA512. SIG and AUTH stand for the signature and verification with RSA512.

Protect the FLASH

3.1 Key Protection

The asymmetric pk_k, symmetric k_f, k_c, and k_v are kept in the EEPROM region memory for the bootloader to access at run-time. When the bootloader starts execution it configures the MPU, which allows only the bootloader itself to access the EEPROM memory. After that, it copies the keys from the EEPROM to the flash memory. Immediately, EEPROM blocks containing the keys are locked using the EEPROM block locking feature, which denies read/write access for everyone until reset. Thus, even if attackers can overwrite the MPU configurations, they can not access the EEPROM memory.

3.2 Firmware and Configuration Data Protection

Symmetric crypto is used for firmware, configuration data, version number protection, k_f, k_c, and k_v. Our developed host tools implement two layers of symmetric encryption, the outer layer is to protect the authenticity, integrity of the version number and the inner layer for the firmware body. We keep the firmware and configuration data in encrypted form in flash memory, whereas at firmware boot time we put the decrypted firmware in SRAM and decrypt the configuration data in flash. Figure 1 demonstrates the format of protected images.

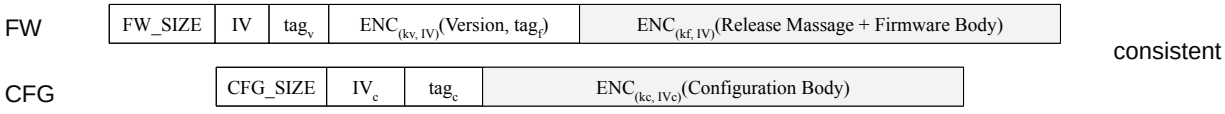

Figure 1: Protected Firmware and Configuration Format

3.2.1 Host tools

Our host tools first use firmware encryption key k_f and IV encrypt the firmware binary and release message: ($\mathcal{C}_{FW}$, tag_f) = ENC$_{k_f,iv}$(Firmware Body + Release Message). Then we use the version encryption key k_v and IV to encrypt the version number and tag_f: ($\mathcal{C}_V$, tag_v) = ENC$_{k_v,iv}$($\mathcal{V} + tag_f$). The finalized protected firmware contains firmware size, IV and tag_v as the header and $\mathcal{C}_V$ and $\mathcal{C}_{FW}$ as the body.

Similar to protect the firmware, host tools use configuration key k_c and IV_c to encrypt the configuration: ($\mathcal{C}_{CFG}$, tag_c) = ENC$_{k_c,iv_c}$(Configuration Body). Then the protected configuration has configuration size, IV_c and tag_c as the header and $\mathcal{C}_{CFG}$ as the body.

3.2.2 Bootloader

Update:

Firmware version checking: Whenever the bootloader receives the update firmware command, it uses the IV and the symmetric key k_v to decrypt the C_V: ($\mathcal{V}'$, tag'_v, tag'_f) = $\text{DEC}_{k_v,iv}(\mathcal{C}_V)$). If tag'_v matches tag_v, it passes version integrity checking. Then it validates the version number with the previously installed one to decide whether accept or reject this update. If valid, bootloader keeps the tag tag'_f, IV and firmware size to check the firmware integrity.

Firmware integrity checking: After validating the version number, bootloader uses IV and the symmetric key k_f to decrypt the $\mathcal{C}_{FW}$: ($\mathcal{P}$, tag'_f) = $\text{DEC}_{k_f,iv}(\mathcal{C}_{FW})$). If tag'_f matches tag_f, it passes firmware integrity checking. Then bootloader updates the encrypted firmware ($\mathcal{C}_{FW}$) in flash memory. Otherwise, the update will be rejected.

Configuration integrity checking. Whenever the bootloader receives the update firmware command. The process is similar to firmware integrity checking.

Boot firmware: At firmware boot time the bootloader decrypts the firmware body and release message with key k_f and IV ($FW_{body}, Rel_{msg}, tag'_f$ = $\text{DEC}_{k_v,iv}(C_{FW})$). The tags tag'_f and tag_f are compared for integrity checking. On successful verification the bootloader puts the firmware plaintext in SRAM.

The configuration data is handled the same way as the firmware with one layer of symmetric crypto with key k_c and IV_c.

3.3 Prevent Version Rollback

Bootloader maintains the version number of firmware in flash memory with MPU configuration. The version number is part of the encrypted body in the protected firmware binary. Only the newer, equal, or zero version of the firmware can be installed. should we store an encrypted version number, keep the version in a register

3.4 Readback Host Authentication

Asymmetric crypto is for readback functionality, the host generates a public key pair (pk_k, sk_k). The secret key sk_k never leaves the host, while the public key pk_k is loaded on the devices as EEPROM data. We implement a digital signature mechanism to ensure that only the valid host can request a readback.

Step 1. Initialize readback function. The host (or attacker) sends the command **R** through UART to bootloader. Bootloader initiates readback and waits for the authentication message.

Step 2. Host sends authentication message. The host ~~generates a digest ($\mathcal{P}_\mathcal{H}$)~~, which is a random plaintext ($\mathcal{P}$) hashed by SHA-256. Then it uses the private key sk_k to encrypt the digest, $\mathcal{C}$ = $\text{ENC}_{sk_k}(\mathcal{P}_\mathcal{H})$. The ciphertext $\mathcal{C}$ & plaintext $\mathcal{P}$ will be sent through UART to the bootloader ($\mathcal{C}$, $\mathcal{P}$).

challenge (nounce) should be provided by bootloader

Step 3. Bootloader authenticates the host. The bootloader uses the associated public key pk_k from EEPROM to decrypt the ciphertext and get the message digest $\mathcal{P}_\mathcal{H}$ = $\text{DEC}_{pk_k}(\mathcal{C})$. Lastly, it generates a second messages digest $\mathcal{P}'_\mathcal{H}$ directly from the received plaintext $\mathcal{P}$ and compares both digests $\mathcal{P}'_\mathcal{H}$. If they do not match, discard. Otherwise: response the readback request.

Step 4. Decrypt the requested memory of the firmware or configuration data and write response on the UART connection for the host.