DESIGN DOCUMENT

Version 1.0

Cheah King Yeh
Won Ying Keat

Advisor: Prof. Lee Yeow Leong
Singapore Management University

Overview

Summary

SAFFIRe consists of both the bootloader as well as host tools. This design document will provide a brief overview of how each aspect of SAFFIRe can ensure that it meets both the functional and security requirements.

Firstly, encryption. Our design utilises proven libraries to allow for Authenticated Encryption, using a variant which allows for associated data (AEAD). Encrypting and verifying the firmware and configuration data would offer resistance to forgery, tampering and plaintext recovery.

To achieve the above mentioned goal, we have detailed it under the Implementation segment. In brief, we plan to regenerate fresh secrets during every SAFFIRe build shared between the host tools and the bootloader. It would be stored in the *secrets* volume and the *EEPROM* for access by the host_tools and the bootloader respectively. Modern AEAD would be used to securely encrypt the resources to be transported, validated and decrypted before loading into the device.

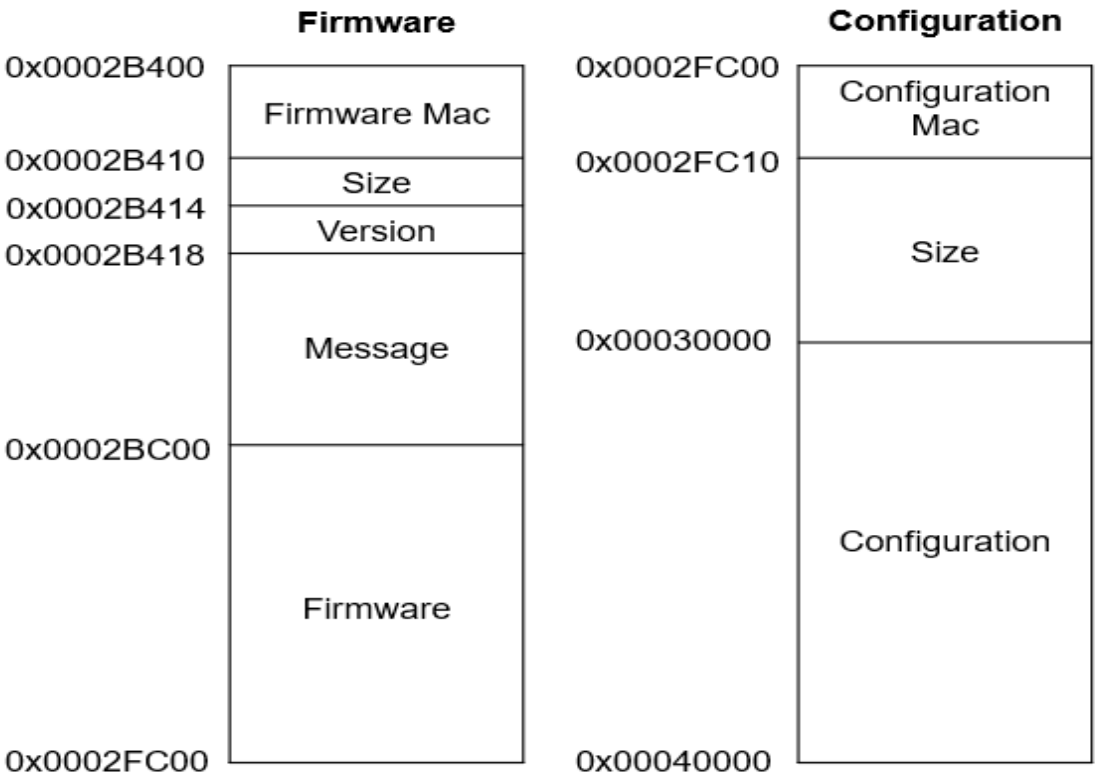
Next, we mitigate replay attacks by using a freshly generated nonce per encryption as well as using counters to prevent nonce reuse. Details will be elaborated in the segments below.

Finally, upon every boot, the SAFFIRe bootloader will ensure that the contents of the firmware (prior and post copy), and configuration data are valid before actually booting the firmware.

Implementation

Space configurations

To accommodate the space and size requirements, we have structured the data to be placed in the bootloader in the following manner.



Firmware

For the firmware, the MAC, size, version and part of the release message (if applicable) will be stored in a single page. If the size of the release message with the metadata exceeds the page size, it will be stored in the next page. Extra space in the second page is padded with 0xFF. Another 16 pages will be allocated to the actual storage of the firmware data itself.

Configuration

For the configuration, the configuration MAC and its size is stored in a single page with padding while an additional 64 pages are allocated for the rest of the configuration data.

Security requirements

Confidentiality

The firmware and configuration binaries are to be securely encrypted and only decryptable by the intended recipient (the bootloader or host tools). To achieve this, we would use symmetric encryption schemes. Note that the firmware metadata (size, release message and version) will not be encrypted but would be taken into account by validating its integrity.

Upon a failed decryption, the bootloader will return an error status code of 0x2 and will stop its current operation. Eventually, the next boot will also fail since the firmware loaded previously is invalid and/or tampered with.

Firmware Integrity and Authenticity

Both firmware and configuration files will be protected against tampering and replay attacks. Authenticated Encryption with Associated Data (AEAD) is used to prevent adversaries from modifying data without detection. The bootloader will check for the authenticity of the data prior to loading the firmware into the device using the various Message Authentication Codes (MACs).

The MACs are validated during the loading of the firmware and configuration and would be used once again during boot. Tampering of the firmware metadata and/or the MAC transmitted across the UART socket would throw an error and the ongoing operation would be stopped. In this case, an error code of 0x2 would be returned from the bootloader.

Firmware Versioning

The versioning will be embedded in the *associated data* portion of the AEAD scheme as it is packed with the firmware. Since both data are MACed, it is unlikely that it can be tampered with. The bootloader will also ensure that the version is later or equal to the current installed version. The various rules for versioning as documented in the *rules* document are complied with. They include:

- Setting the minimum version to the OLDEST_VERSION when it is a clean slate
- Preventing loading of versions lower than the current version unless it is version 0
- Store the current firmware version in flash memory, unless it is version 0, of which the previous version will be stored

Readback Authentication

A challenge-response scheme is used to ensure that only authenticated personnel can read the data. It requires the personnel to generate a valid authenticated MAC for the challenge. No secrets will be revealed within this 3-way process and only an authenticated personnel can generate a valid MAC.

Miscellaneous

Using the recommendations from the crypto library, secret keys and other sensitive information are wiped from the memory (via zeroing) when no longer needed as much as possible.

Security Details

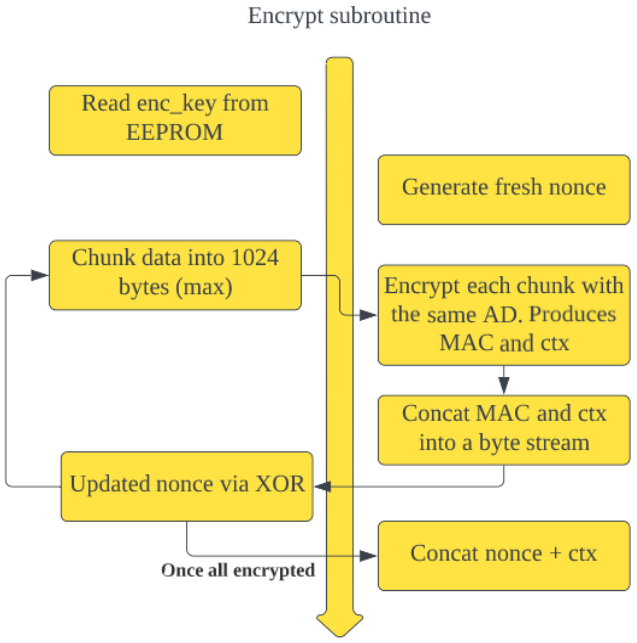
We plan to use state-of-the-art XChaCha20-Poly1305 (RFC 8439) for AEAD. These are chosen because they are proven to be computationally secure and are generally trusted. For the encryption of data in bulk, a fresh nonce will be chosen by obtaining them from /dev/urandom in the host tools.

Encryption and Decryption Subroutine

To allow for efficient encryption, the data are first broken into chunks such that the maximum size per chunk is the PAGE_SIZE (1024 bytes). Each chunk will be encrypted with the encryption key with the random nonce. Upon every encryption, the 10th byte of the nonce will be XORed with a running counter, starting from 1. This guarantees that the nonce will not be reused when encrypting up to a maximum of 256 chunks. Since this number is larger than the maximum firmware or configuration size, this approach is used.

When encrypting each chunk, a MAC and ciphertext is returned. They are concatenated into a single byte string. Upon encryption of all chunks, the original nonce is concatenated to the byte stream and returned to the *host_tools*. Similarly, the bootloader will unpack the byte stream into the nonce and the respective chunks and perform decryption and integrity checks per chunk, calculating the nonce in the same way as the *host_tools*.

The libraries used are [monocypher] in the SAFFIRe bootloader as well as [pymonocypher] (Python wrapper) for the host tools to perform the above mentioned tasks.



Functional Requirements

Our team has decided to keep the design simple and easy to understand. Complexity might make a system more robust but we believe that a simple system will be less prone to human and coding errors. Hence, we have included nice looking diagrams for the various functions.

System Build

The Dockerfile *host_tools/1_create_host_tools.Dockerfile* is edited to install the required tools such as (but not limited to) *python3*, *pip3* and *pymonocypher*. During this build, the secrets generated will be stored in the *secrets* volume and subsequently in the EEPROM of the bootloader. These secrets will be used mainly for encryption and authentication.

The remaining functionality to pack and include of other standard components will be similar to the reference code provided.

Launch Bootloader

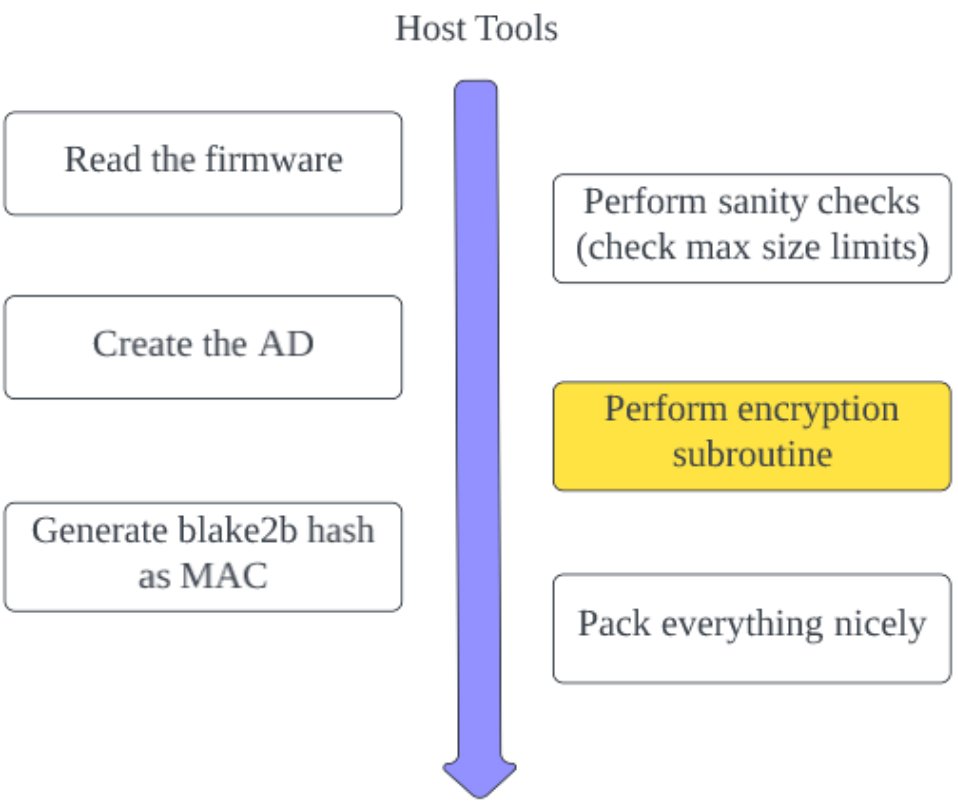
No change required as per the rules and requirements.

Firmware Protect

Building upon the reference design, the main difference would be an additional function call to perform authenticated encryption. The shared key of the host tools in the *secrets* volume will be used. A nonce will be generated randomly by the host_tools by reading from /dev/urandom. When performing multiple firmware protections, a fresh nonce will be used every time. However, within each encryption routing, the nonce will be changed by XORing the 10th byte with a running counter.

The firmware version, release message and raw binary will be packed into a single package. Only the actual binary is encrypted. The metadata will be available in clear text and form the “associated data” portion of AEAD.

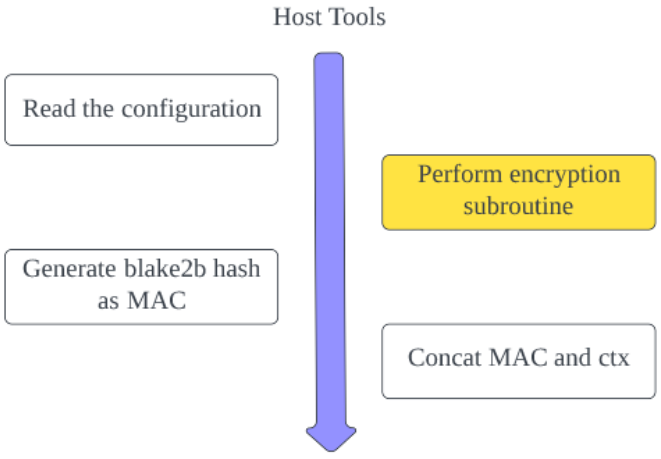
Moreover, a MAC will be generated on the unprotected firmware and included together in the packed firmware data. This MAC will come in handy during the *device boot* process. The tool for MAC will be Blake2b.



Configuration Protect

The same process will be used for mission configurations. However, the function will not be required to pack any other information other than the raw firmware. The calls to perform authenticated encryption will still be similar to *firmware protect*.

As usual, a new nonce will be used for every call to perform configuration protection, generated using similar methods as described in *firmware protect*. Similarly, a MAC will be generated on the unprotected configuration and appended to it.

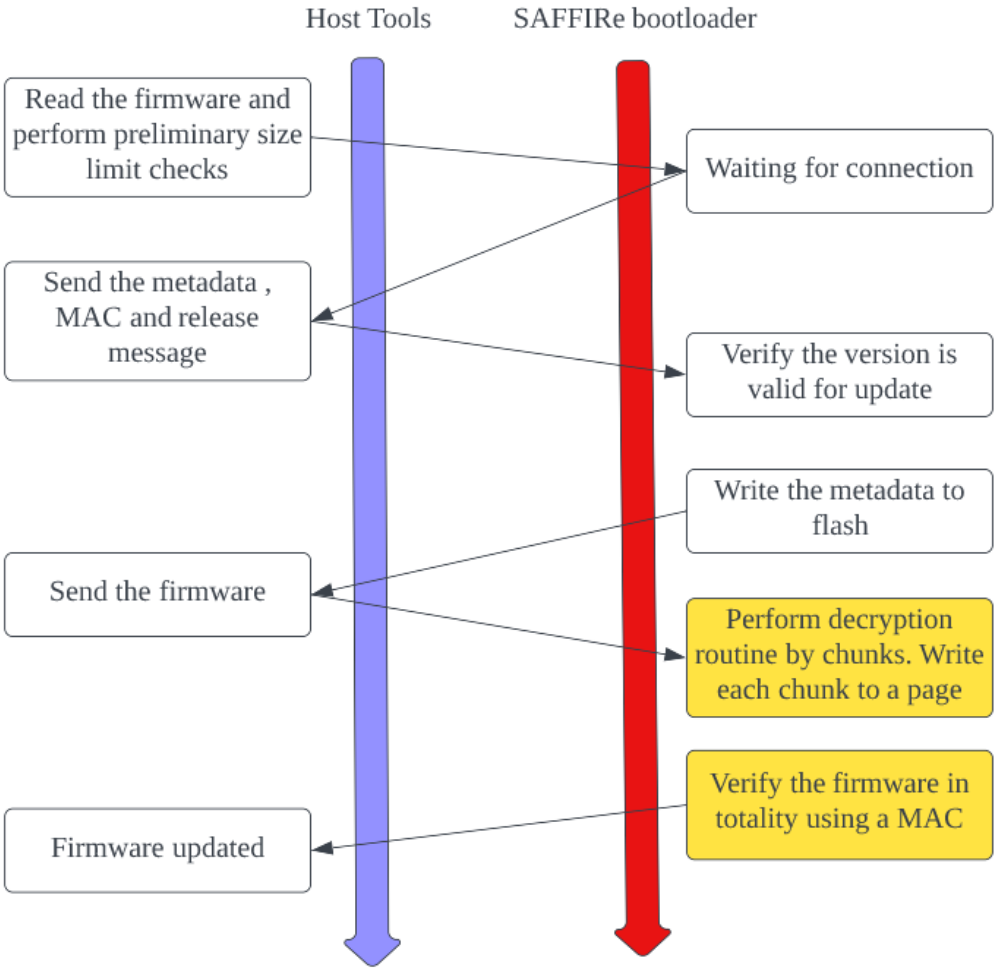


Firmware Update

The host tools will simply send the protected firmware across to the SAFFIRe bootloader. The bootloader will be in charge of ensuring the integrity of the firmware by using the secrets stored in EEPROM and performing decryption and integrity checks.

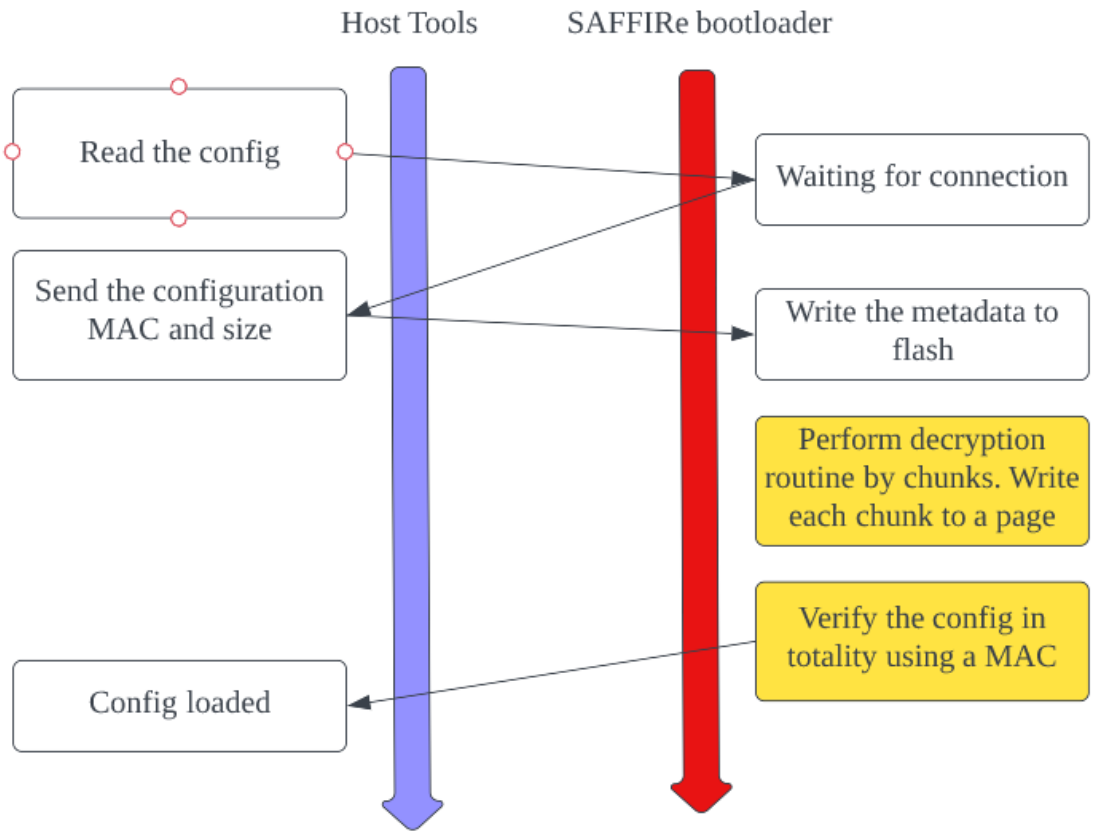
It will first proceed to check the version number of the firmware sent to it. Should the version number fulfil the requirements as stated in the Firmware Versioning section above, it will proceed to read and write those data into the flash memory. Then, it will read the firmware and perform decryption chunk by chunk, similar to how the encryption is carried out.

The decrypted data will be written directly to the flash memory once the chunk is verified. Should all decryption end successfully, a final check using the appended MAC as described in *firmware protect* will be performed to ensure the decrypted blob's identity has not been compromised.



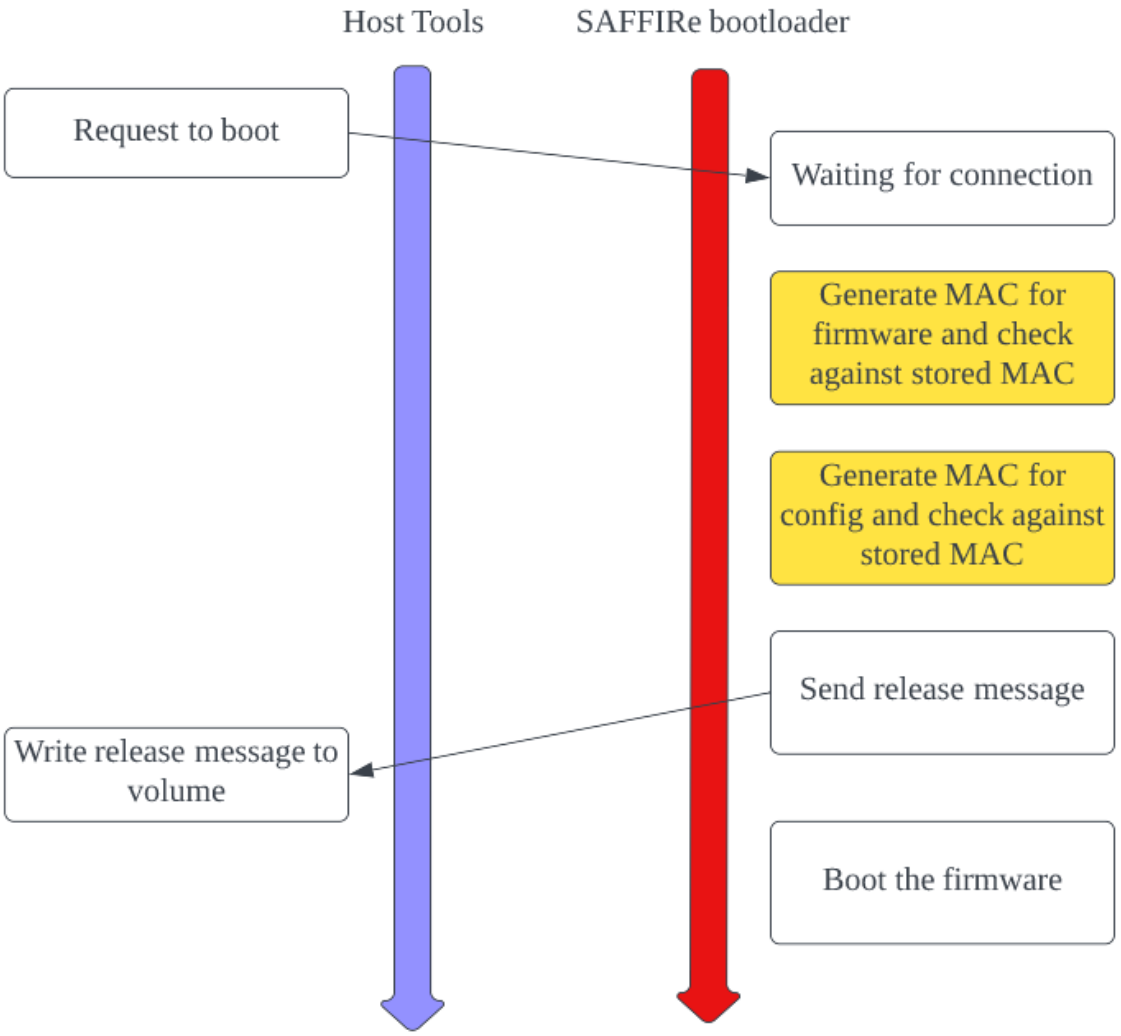
Config Load

The configuration will be similar to that described in *firmware update*.



Device Boot

When loading the data, we will once again ensure that the firmware integrity is valid by calculating the MAC using the secrets found in EEPROM and subsequently load the data into SRAM if the checks pass for both the firmware and the configuration data.



Readback

During initialization, some random bytes will be stored in the EEPROM. This data will become the challenge which is presented to the personnel. To be authenticated, the personnel will need to MAC the challenge using the shared secret key. This piece of data will then be sent to the bootloader which will verify it. Should the response received be correct, the SAFFIRe bootloader will proceed to return the requested data - firmware or configuration. Note that the MAC should produce a 32 bytes result instead of the 16 byte MAC used throughout the rest of the design. Regardless of success or failure, the bootloader will generate a fresh challenge by repeatedly hashing the original challenge using the stored secret key, storing the result in EEPROM for the next readback.