2024 eCTF: Design Document

Team IITI

Introduction

This document outlines the design of the solution proposed for the Embedded Capture The Flag (eCTF) competition hosted by MITRE. The solution aims to address the challenges presented in the competition, focusing on secure and efficient strategies for securing the Medical Device. We have given a simple overview of our understanding of the security requirements. We have kept our approach simple and will follow an incremental approach in later stages to use more complex techniques.

We have used the reference design provided by MITRE. We will use the functions provided in the reference design to meet the functional requirements. For implementation of the security requirements, we make a few assumptions according to the preliminary knowledge we have gathered:

- The global secrets file is not accessible to the attacker at any point.
- MAX78000 has hardware support for AES encryption, and two devices will never be provisioned with the same I2C address.
- According to the implementation of the address mask, the last byte of the ComponentIDs will be unique for Component 1 and Component 2.
- We assume the SRAM present on the devices will not reset after the "boot" function is called.
- We also assume that MAX78000 supports true random number generation as stated here.

We also assume that all the processes are atomic and non-interruptible as and when required. During implementation, we will analyze and take care of all such cases. For cryptography, we will initially use the simple crypto interface provided by the organizers. Later, we plan to use WolfSSL Library to meet the requirements.

Security Requirement 1

The Application Processor (AP) should only boot if all expected Components are present and valid.
The following problem requires us to deal with two main parameters:

1. All the expected components are present.
2. We must verify all the components are valid.

This is important to make sure that all the components are valid and expected to ensure unintended actions by the Medical Device. To accomplish this, The global secrets will contain two keys:

1. A key for the components.
2. A key for **AES communication**.

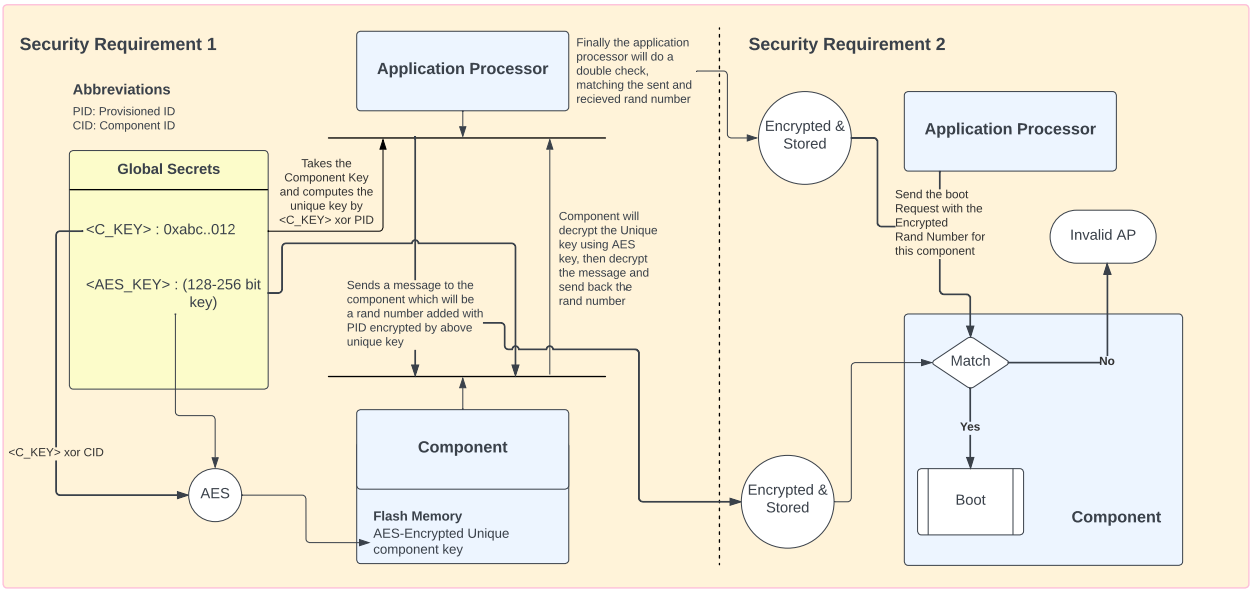

Figure 1: Security Requirement 1 and Security Requirement 2 Flow Chart/s

When we build the .img file, we will also store an encrypted key on the components' flash memory, which will be obtained by taking the XOR of the ComponentID and the component key available in the global secrets. The extra XOR operation between the ComponentID and the component key in the proposed authentication process adds an additional layer of security to the verification mechanism.

Whenever the AP wants to verify the component, it calculates the XOR of the component key and the provisioned ID, which is used to encrypt a puzzle where we add a random number generated by the hardware and the provisioned ID. The component receives the message and first decrypts the message using the XOR of the ComponentID and key. It should only be able to decrypt if the ComponentID and provisioned ID match. Even then, as an extra layer, we subtract the ComponentID from the decrypted message and send it back to the AP. Here, AP compares the returned message, the random number and the original random number. If they are found to be equal, the component is validated.

We perform the above process for all the provisioned devices. Only then can the boot function be called from the AP. To implement this, we will use an atomic counting semaphore.

Security Requirement 2

Components should only boot after being commanded to by a valid AP that has confirmed the integrity of the device.

This requirement should be satisfied after confirming the integrity of the device by integrity; we assume that it refers to security requirement 1. We plan to use the random number used in the puzzle communication. The same random number will be hashed using **SHA-256** stored for each component in a map in the AP, with keys as the provisioned ComponentID and the value as the hashed random number. The component will also store the hashed random number that was sent to it during the validation process.

While booting the components, we will send the hashed random number for a particular component while sending it the boot command; the component will first check it against the one stored in it. If it matches, it will call the boot sequence; otherwise, it will return an error message.

By requiring components to match the hashed random number during the boot process, the sys-

tem ensures that only components are commanded by a correct and validated AP. Intertwining the steps for Security Requirement 1 and Security Requirement 2 creates a cohesive security framework, fortifying the system against potential threats and unauthorized access throughout the lifecycle of the components.

Security Requirement 3

The Attestation PIN and Replacement Token should be kept confidential.
The problem requires us to deal with the issue that if we store the Attestation PIN and Replacement Token directly on the flash memory of the AP, then it will be susceptible to attacking and getting the Attestation PIN and Replacement Token.

So, to keep the Attestation PIN and Replacement Token confidential while building the AP, we will not store them as it is in the flash memory; instead, we will first encrypt the Attestation PIN and Replacement Token using **SHA-256**, a one-way or irreversible hash function. While requesting the Attestion and Replacement of components, we will first encrypt the Attestation PIN and Replacement Token using the same one-way function and compare it with the stored hashed versions of the tokens.

The advantage of using the one-way hash function is that even if the hacker gained access to the flash memory, they would only be able to see the hashed version of the desired input rather than the actual desired input, and as the one-way encryption is not possible to decrypt it will be next to impossible for the attacker to know what the real input should be for attestation or replacement.

Security Requirement 4

Component Attestation Data should be kept confidential.
Here, we have to deal with the issue that Attestation data is stored on the component, and if this data gets attacked, this might lead to the recreation or modification of some critical components. Moreover, this data must be stored so that it can be retrieved in its original form only when a valid request for the same has occurred.

For this, we will keep attestation data in the component in encrypted form using **AES encryption** with the key as XOR of C_key and ComponentID to ensure the key is unique to each component. Firstly, the user requests the attestation data by entering the PIN. The PIN will be stored as a hash, which will be compared with the PIN entered by the user after hashing it. If a match is found, the AP sends a request to the component for the required data. The data will be sent to the AP using the secure send and secure receive functionality. As AP has access to C_key and ComponentID, it can decrypt the received message and display it.

Security Requirement 5

The integrity and authenticity of messages sent and received using the post-boot MISC secure communications functionality should be ensured.
To satisfy the given requirements, we will use **AES encryption/decryption** to secure the channel. An **AES communication key** will be stored in global secrets. Whenever any component wants to send a message to AP, it will encrypt the message with AES encryption key XOR ComponentID. When the AP receives the message, it will decrypt using the same KEY, i.e., AES encryption key XOR ComponentID, and vice-versa.

The benefit of using AES Encryption key XORed with ComponentID as the encryption key for encrypting/decrypting the messages is that the encryption key would be distinct for each component to AP communication. Furthermore, using this way of creating encryption keys will also benefit us

while replacing the components, as the AES encryption key used in generating the actual encryption key will remain constant but the actual encryption key would be distinct and change as per the component. This way, it also aligns with the assumption that the global secrets file will not change after deployment, but still the encryption keys stay dynamic.

Wishlist

Here are a few things we plan to implement, but they require quite more time, so we plan to implement the above discussed initially and later extend to the items mentioned below based on the feasibility in the given time.

1. Instead of XOR, we will explore the use of irreversible functions. XOR is a simple bitwise operation and is not considered a robust encryption mechanism. It may provide a different level of security than more sophisticated cryptographic algorithms.

2. We will implement a timeout to prevent brute force on the SHA256 hashes. This means that for every three incorrect attempts to retrieve the data, say the wrong Attestation PIN, we will lock the function for 30 seconds. This will increase the time required to brute force the hashes to an acceptable level without violating the competition rules.

3. We will explore the possibilities of other symmetric encryption techniques and some asymmetric techniques such as RSA, ECC, etc. In the secure_send and secure_receive functions, we intend to implement a double encryption mechanism. We will first encrypt the message using the private key of the sender. The message is further encrypted using the public key of the receiver. To unlock the message, the receiver first uses the sender's public key and then their private key to decrypt it. This will help to ensure the authenticity of the messages.

4. We will implement a digital signature for the messages. Whenever a message is sent, we will send the hash of the message along with the actual message. The receiver will hash the message on their end and compare the two hashes. A match shows the integrity of the message.

5. We will also try to find some use cases of CNNA (Convolutional Neural Network Accelerator), present in the MAX78000 FTHR board for increasing the speed of encryption/decryption or some other functionality in this project.

6. We will explore the checksum and error correction code provisions by the I2C bus protocol used for board communication. We will implement the same to ensure the integrity of the messages.

Summary

This document provides a brief overview of the design choices we are planning to make. The current idea is simple and might fail in some cases. However, we will follow an incremental approach to improve the design, continuously test our design for various vulnerabilities, and try to improve the same. The wishlist outlines future enhancements, including the exploration of irreversible functions, implementation of a timeout mechanism, investigation into other encryption techniques (RSA, ECC), double encryption in secure_send, secure_receive functions, digital signatures, CNNA utilization, and I2C bus protocol provisions for integrity. We are committed to a continuous learning and development process throughout the completion of this project.