2024 MITRE eCTF Design Document

Team UCI

Richard Sima, Emma Xiao, Zhanhao Ruan, Zhengxuan Li, Jinyao Xu, Zack Wang, Leo Yang, AJ Moon, Zuhair Taleb, Yintong Luo, Xiaocheng Li

Overview 4
Functional Requirements 4
Req. 1 - List Components 4
● Description 4
● Implementation 4
Req. 2 - Attest 4
● Description 4
● Implementation 5
Req. 3 - Replace 5
● Description 5
● Implementation 5
Req. 4 - Boot flash memory 5
● Description 5
● Implementation 5
Req. 5 Secure Send and Receive 6
● Description 6
● Implementation 6
Security Requirements 6
Req. 1 Component Validity Check 6
● Description 6
● Implementation 6
Req. 2 AP Validity Check 6
● Description 6
● Implementation 6
Req. 3 Confidential Pin & Replacement 7
● Description 7
● Implementation 7
Req. 4 Confidential Attestation Data 7
● Description 7
● Implementation 7
Req. 5 Secure Post-Boot MISC Messaging 7
● Description 7
● Implementation 7
Post Boot Requirement 8
Dependencies & Build AP/Component Phase 8
Key Synthesis 9
● Term Definitions: 9
● Algorithm Description: 9
● Visualization: 10
● Security Proof 10

● Robustness..11
Secure Communication Protocol.................................... 12
● Protocol Overview..12
● Term Definitions.. 12
● Protocol Overview..13
● Implementation..13
Security Proof.. 14

Overview

There will be one AP that supports up to two medical components. When prompted to, the AP will send the components' IDs to the host tool to be displayed as its "list device" functionality. This is not protected and can be initiated by anyone. To boot the device, the AP will first validate the expected component(s), and only if this test passes and both components send a boot message to the AP will the AP finally boot. Each component stores attestation data on the flash memory as well as an attestation PIN, only if a valid attestation PIN is provided will the component release the data to the AP that requested it. Components can still only be replaced if a valid replacement token is provided to the AP, after which the expected component ID will be updated to the new one. The AP should now reject any other components. Validation of components relies on the proposed **Key Synthesis** algorithm, and a secure communications channel is provided to the components and the AP through the proposed **Secure Communication Protocol**.

Functional Requirements

Req. 1 - **List Components**

- **Description**

The MISC must be able to list the Component IDs of the Components currently installed on the Medical Device. This command is not authenticated and may be initiated by anyone.

- **Implementation**

Implemented with scan_components() in application_processor.c. The AP goes through a for loop to all possible I2C addresses and sends list requests and waits for responses. The AP prints out the expected component IDs, then sends out a command to a component and waits for a successful response. If all provisioned components return a successful response, the function returns and prints success.

Req. 2 - **Attest**

- **Description**

Attestation data from components must only be able to be retrieved with a valid attestation PIN.

- **Implementation**

The attestation PIN is stored on the flash memory of the components. Since the PIN is always working on the local server, it is not necessary to be encrypted. When a user enters a PIN on the host tool, it is sent to the AP. The component(s) then check the received PIN and if it is correct, will release the attestation data, which is being encrypted by the AES algorithm, to the AP to be returned to the host tool.

Req. 3 - **Replace**

- **Description**

A MISC Component must only be able to be replaced if the user provides an authorized replacement token and the component is authentic.

- **Implementation**

The user enters a replacement token on the host tool which sends it to the AP. If the replacement token is incorrect, the AP rejects the replacement attempt. The component_id will be passed along during the replacement process. Once the validity check is passed, the ID stored in the flash memory will then be changed by AP. If the key is incorrect, the new component is rejected. Given the fact that keys are shared and unique between 3 parts of a system, "replacing" with a fake component will not cause the system to boot.

Req. 4 - **Boot**

- **Description**

The MISC must first ensure the integrity of the device and the Components on it. If the integrity check fails, the boot is aborted.

- **Implementation**

Booting the system involves 2 parts: validation and boot command. The order of boot for our design follows generally with the original design (component 1, component 2, and AP). The whole booting sequence presumes that the secure channel of communication is already established (key synthesis completed where each component and AP has a shared AES128 key).

The AP first validates and then boots the component in order (assuming we have 2 components but the case for one component also holds). Once both components are validated and booted by a valid AP, it sends a booting acknowledgment to the AP, and the AP will then boot as well. Detailed security measures can be found in the security section.

Req. 5 **Secure Send and Receive**

- **Description**

After a successful boot, the MISC must provide a secure communications channel for the AP and components to use.

- **Implementation**

Once key synthesis is complete and devices are all booted, the secure channel of communication is already established so a shared key will ensure secure communication. Detailed specifications can be found in the security requirements section.

Security Requirements

Req. 1 **Component Validity Check**

- **Description**

The Application Processor should only boot if all expected Components are **present and valid**.

- **Implementation**

The AP is flashed with a number of expected components. If the number of components connected differs from the number of expected components, the boot process is aborted. The AP first conducts a scan of existing components just like in the list functionality. If this list is successfully done (under the AES encryption scheme), then validation will be performed.

The AP sends a random number along with an opcode to the components. The components decrypt and validate the boot message as well as the opcode. The component will then send the random number back to the AP. If AP decrypts before its time frame expires then it will boot (given it receives all two components messages).

Req. 2 **AP Validity Check**

- **Description**

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

- **Implementation**

After the component has received something from the AP, it will decrypt the message with the unique key and see if the decrypted message contains a meaningful boot command. If the boot command is confirmed, the component will reply with its component_id and boot confirmation message back to AP in encrypted mode and initialize the booting.

Req. 3 Confidential PIN & Replacement

- **Description**

  The Attestation PIN and Replacement Token must be kept confidential.

- **Implementation**

  All instances of the Attestation PIN and Replacement token that may appear in communication will be encrypted and thus will not be viewable by any attackers that intercept communications.

Req. 4 Confidential Attestation Data

- **Description**

  The Attestation Data must be kept confidential

- **Implementation**

  The Attestation Data is encrypted using the secret key before being flashed onto the memory of a component. Only when a correct attestation PIN is provided will the component then send the data to the AP, which can then decrypt the data using the secret key. All instances of the Attestation Data in communications will be encrypted.

Req. 5 Secure Post-Boot MISC Messaging

- **Description**

  The integrity and authenticity of messages sent and received using the post-boot MISC secure communications functionality should be ensured.

- **Implementation**

  Periodically refresh encryption keys between the Application Processor (AP) and Components (C1, C2). then it uses a hash-based message authentication code for message integrity and authenticity checks. Incorporate sequence numbers in messages to prevent replay attacks. Timestamps are included in messages to verify timeliness. Encrypt messages using AES for confidentiality. Implement error handling and response timeouts for reliable communication. Conduct regular health checks of components. Maintain logs for all communications and monitor for security anomalies.

Post-Boot Requirement

Description

Once the system is active, the communications between the components must be secure from tampering, duplication, or forgery. The utilized four-way communication serves(See Requirement 5) as a validity checker to ensure integrity.

Deployment & Build AP/Component Phase

Deployment:

During the deployment phase, the Makefile will invoke a Python script named import_csv.py to generate a CSV file named cc.csv. This file will contain a list of 300 pairs of Masks and Final Masks, which will be utilized in the subsequent phases of building the Application Processor and Component.

Application Processor/Component:

\`	In the Application Processor and Component building phases, two separate Python scripts, ap_making.py and comp_making.py, respectively, will be called within the project.mk file. These Python scripts will retrieve two rows from the randomly generated CSV file, representing the Mask and Final Mask required for Key Synthesis (for a detailed description, refer to the Secure Communication Protocol documentation). Subsequently, the scripts will write the Mask, Final Mask, and their respective portions of the Key into the key.c file.

Key Synthesis

- **Term Definitions:**

❖ *k1*, *k2*, *k3* - Three 128-bit randomly generated keys
❖ *k* - Final Encryption key obtained by XORing *k1, k2,* and *k3*
❖ *M* - Randomly generated 128-bit mask
❖ *F* - Randomly generated 128-bit mask
❖ *r1, r2* - Random number
❖ *AP* - Application Processor
❖ *Comp1* - Component 1
❖ *Comp2* - Component 2

- ## Algorithm Description:

For devices with the complete two components, they will be 3 key shares generated. The AP stores k2, M, and F. Comp1 stores k1, M, and F. Comp2 stores k3, M, and F. These values are all stored in flash memory in the macro header file. For each round of key synthesis, there will be a random number generated by AP.

1. AP sends *k2r1* to Comp1 and sends *k2r2* to Comp2.
2. Comp1 can now XOR k2r1 to obtain k2. Now Comp1 have k1 and *k2r1*
3. Comp2 can now XOR k2r2 to obtain k2. Now Comp2 have k3 and *k2r2*
4. Comp1 sends *k1M* to AP. Comp2 sends *k3M* to AP.
5. AP can now
    - XOR *k1M* with *M* to obtain *k1*.
    - XOR *k3M* with *M* to obtain *k3*.
    - XOR *k3* with *k2* then with *k1* to obtain *k*

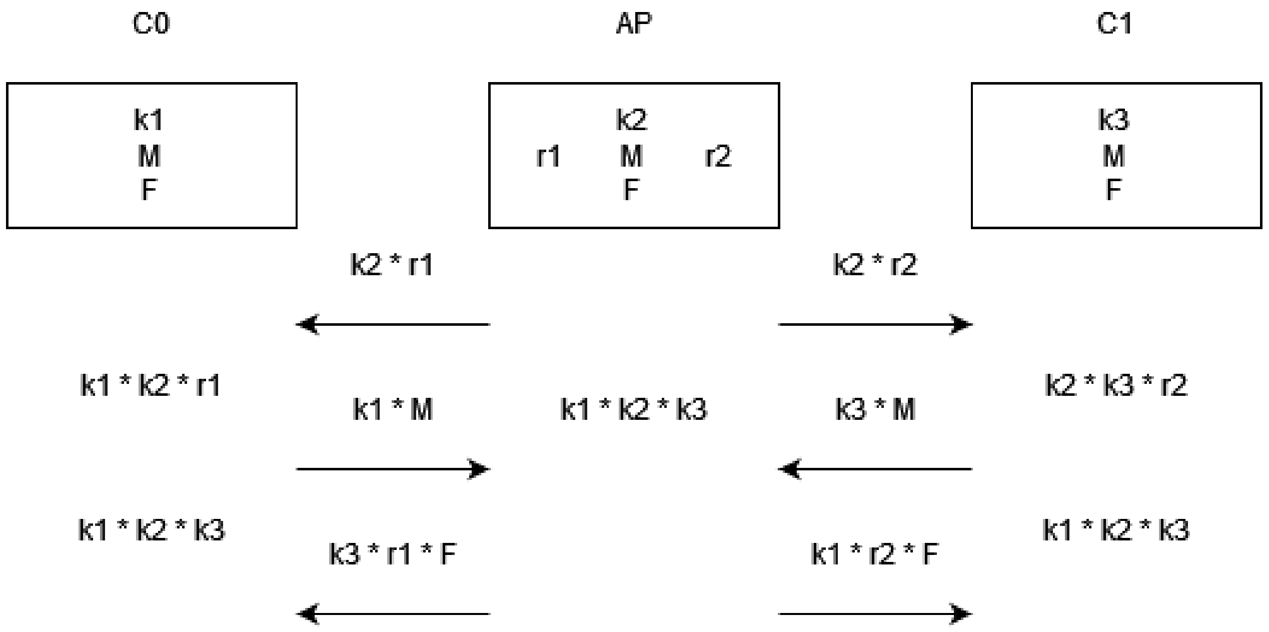
   Now AP have *k1, k2,* and *k3*
6. AP sends *k3Fr1* to Comp1 and *k1Fr2* to Comp2
7. Comp1 can now
    - XOR *k3Fr1* with *k1k2r1* to obtain *k1k2k3F*
    - XOR k1k2k3F with *F* to obtain *k1, k2, and k3*
    - Now Comp1 has *k1, k2,* and *k3*
    - XOR *k3* with *k2* then with *k1* to obtain *k*

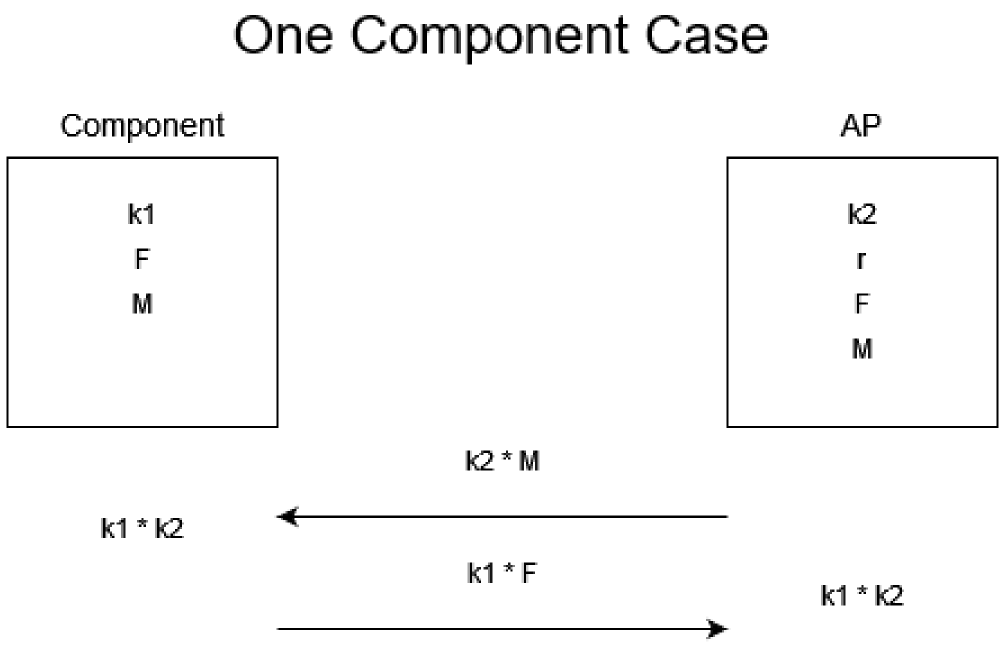
   Comp2 can now
    - XOR *k2k3r2* with *k1r2F* to obtain *k1k2k3F*
    - XOR *k1k2k3F* with *F* to obtain *k1, k2, and k3*
    - Now Comp2 has *k1, k2* and *k3*
    - XOR *k3* with *k2* then with *k1* to obtain *k*
8. AP, Comp1, and Comp2 all now have *k*

For devices with only one component, there will only be 2 key shares generated. There will be no random number used to reduce computational complexity and/or communication errors between the boards. Hence, the algorithm will be as follows:

1. AP sends k2M to Component
2. Component computes k1 * k2 * M * M = K
3. Component sends back its k1 * F to the AP
4. AP computes k1 * k2 * F * F = K

- **Visualization:**





- **Robustness**

The initiation of Key Synthesis within the system is deliberately designed to occur subsequent to the issuance of the first user command. This procedural sequence is embedded within the main function of the `application_processor.c` source file in the Application Processor (AP) module. The commencement of Key Synthesis represents a critical juncture in the operational loop, predicated on the prerequisite that a user-initiated command has been successfully executed. This approach ensures that Key Synthesis transpires within a fully integrated system environment, encompassing the Application Processor and the requisite two component units.

In instances where the component units are not duly connected, the AP engages in the initial round of Key Synthesis and thereafter enters a persistent wait state. This design decision is pivotal to maintaining the integrity and security of the system. By architecturally sequencing the Key Synthesis to follow the user's initial command, the system effectively safeguards against premature or unauthorized activation of Key Synthesis, thereby upholding the overarching security paradigm. Conversely, positioning Key Synthesis as the inaugural step, preceding the execution of the initial user command, could potentially compromise the security infrastructure of the system, leaving it vulnerable to exploitation. This deliberate sequencing thus serves as a fundamental safeguard, ensuring the operational security and integrity of the system from the outset.

All key variables are initialized to random numbers before the Key Synthesis process starts; thus, in case of failure, there will be no "default" synchronized key value.

Secure Communication Protocol

● Protocol Overview

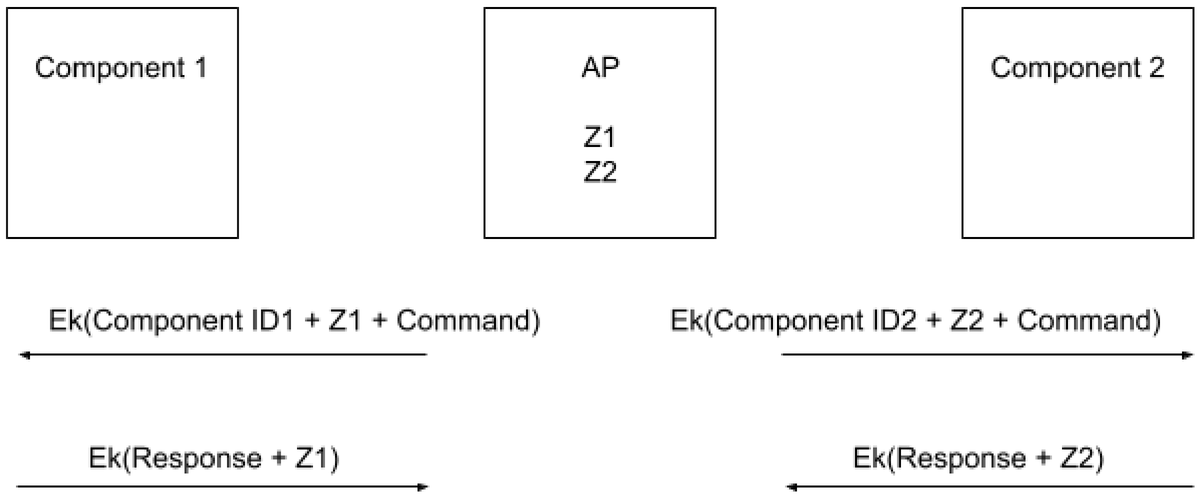
Our secure communication protocol uses a two-way handshake communication process, aiming at the communication between the Application Processor (AP) and both components to make sure it will respond to information correctly to the host machine if and only if it has received encrypted and not expired messages from the component. The functionalities that we need to build a secure communication protocol for are Boot, Attest, and Post-boot communication. These three functionalities use this communication protocol that the AP sends command information to the Component and the Component responds with requested data. All the information flow sent between the AP and the Component will be encrypted and decrypted into the same meaningful message if and only if a unique key is constructed into the board's firmware. The Key Synthesis ensures the encryption and decryption will be secure enough that only three valid devices(one AP and two Components) are valid(by validity it is determined by whether or not the board can decrypt the message on time) and present. If one of the devices is missing or not valid, the communication will be protected since the AP(either valid or malicious) will not decrypt and respond to the host with meaningful information because the key used to decrypt and encrypt messages will be different for the AP and the Components. Because the three functionalities will have the same communication format, in the following documentation, we use the Boot as an example to better explain our implementation.

● Term Definitions

❖ AP, C1, C2 - Application Processor, Component 1, Component 2
❖ *Z1*, *Z2* - 12-byte unsigned integers generated on the AP using the built-in TRNG(True Random Number Generator) function to ensure the freshness of messages.

- E_k() - Encrypt function using ASE, that takes in plain text as input, *k* as the key to use for encryption, constructed in the Key Synthesis step

Note that at this step, it is safe to assume that AP, C1, and C2 have already constructed the complete Key. This Communication Protocol is used for secure communication(Boot, Attest, and Post-boot communication) between AP, C1, and C2.


Component 1
AP
Z1
Z2
Component 2
Ek(Component ID1 + Z1 + Command)
Ek(Component ID2 + Z2 + Command)
Ek(Response + Z1)
Ek(Response + Z2)

Protocol Overview

The Protocol we proposed is a two-way handshake between the AP and both components, which ensures the integrity and presence of the AP and both components before booting any devices. The following procedure will walk through the protocol step by step.

Implementation

Step 1:

AP sends one encrypted message to one component consisting of the Component ID of that component and boot command along with a randomly generated number Z1. AP will keep track of the number Z1 in its memory. The message is encrypted using an Encrypt function E_k. After AP sends out the message, it will start a time window. If the Component fails to respond within that period, the AP will discard the Z and generate a new Z to ignore any messages from the Component that come along with the Z1 to ensure the timeliness of the response. This helps to protect the communication from replay attacks.

Summary: AP sends the encrypted Component ID to C1 to validate the integrity and send the component boot command.

Step 2:

Once AP sends the message, the component should be able to receive, decrypt, and validate the Component ID. If the Component ID matches the ID on the component itself, the component successfully validates the AP and proceeds to boot. After booting, the component will send out another encrypted message containing a confirmation of booting back to AP, along with the same randomly generated number Z1. The message is encrypted by the E_k function.

Summary: C1 receives, decrypts, and validates the Component ID and sends back a response(i.e. Acknowledgment, attestation data, etc.)

Step 3:

AP hears back from the Component and decrypts the message with the key. If the decrypted message contains a meaningful Boot confirmation message, AP will check if the Z1 is the same as it sends out the request. If the Z1 check fails, which means AP doesn't hear back from Components within the designed time window, AP will abort Boot. If the check has passed, the AP will proceed to validate another component.

Summary: AP hears back from the Component and validates the timeliness of the response.

Similarly:

AP repeats Step1-3 with a newly generated random number Z2 to validate another Component. If the validation is successful, the AP proceeds to boot.

Summary: AP validates another component and boots if successful.