System Design

Embedded Medical Devices

By

ETSU

Jan 31, 2024

Introduction

In this document outlining the initial design phase, we are detailing the development of a comprehensive and secure supply chain solution for microcontrollers embedded in a medical device. Our primary focus is on ensuring the integrity of the device and safeguarding sensitive data. To address concerns related to counterfeit device components and their communication, our specific objective is to establish the Medical Infrastructure Supply Chain (MISC). This system is designed to verify the authenticity and integrity of device components throughout their lifecycle, including device operation and repair. The goal is to ensure accurate functionality while implementing robust protection against potential attacks.

In our system architecture, the integration of three integral components is designed to fulfill the system's functional requirements (Figure 1). The core elements consist of two components working with an application processor (AP). The host machine dispatches commands to the AP, which, in turn, executes actions aligned with the specified functional requirements. This collaborative setup ensures the seamless functioning of our system, where the host, AP, and components work to meet the defined objectives.

To facilitate communication between the host and AP, UART is employed, ensuring a reliable and efficient data exchange mechanism. Simultaneously, communication from the AP to each component is established through an I2C bus, enabling a synchronized flow of information.

Adopting C as our primary programming language for firmware development is a strategic decision that enhances the security of our code. The language's flexibility and efficiency make it well-suited for embedded systems, allowing us to build robust and reliable software solutions. In conjunction with C, we leverage custom libraries to translate functional requirements into tailored solutions that precisely address the requirements of our system.

In adherence with the Security Requirements, our firmware incorporates advanced encryption and authentication methods. This multi-layered security approach serves as a robust defense design, enhancing the entire system against potential threats and aligning with the highest standards of security, especially in the critical context of medical applications. To summarize, our system architecture, communication protocols, and security measures collectively contribute to establishing a robust and reliable embedded medical system.

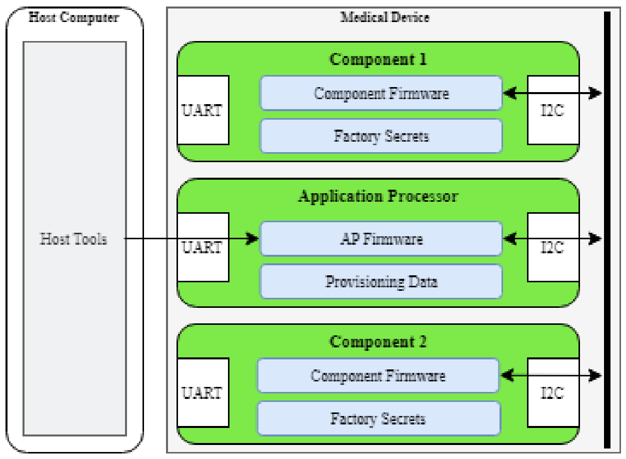


Figure 1: System Architecture: Host, Components, and AP

A. Functional Requirements

To meet the functional requirements, we will create custom libraries in Rust. These libraries will empower the Host to securely initiate the Application Processor (AP), verify the integrity of the components, facilitate encrypted communications for both sending and receiving data, oversee user authentication, and manage error logging and handling—all accomplished within the specified time limits.

List Components

The AP will receive a List command from the Host and return a list of Component IDs currently installed. In event of failure, the AP should display the error message(s). The List command has a time requirement of three seconds to complete the operation.

Attest

Allow for the retrieval of Attestation data if an Attestation PIN is provided and authenticated by the Security Design as outlined below in Security Requirement 3 to verify that a Component is authentic within the three second timeframe.

Replace

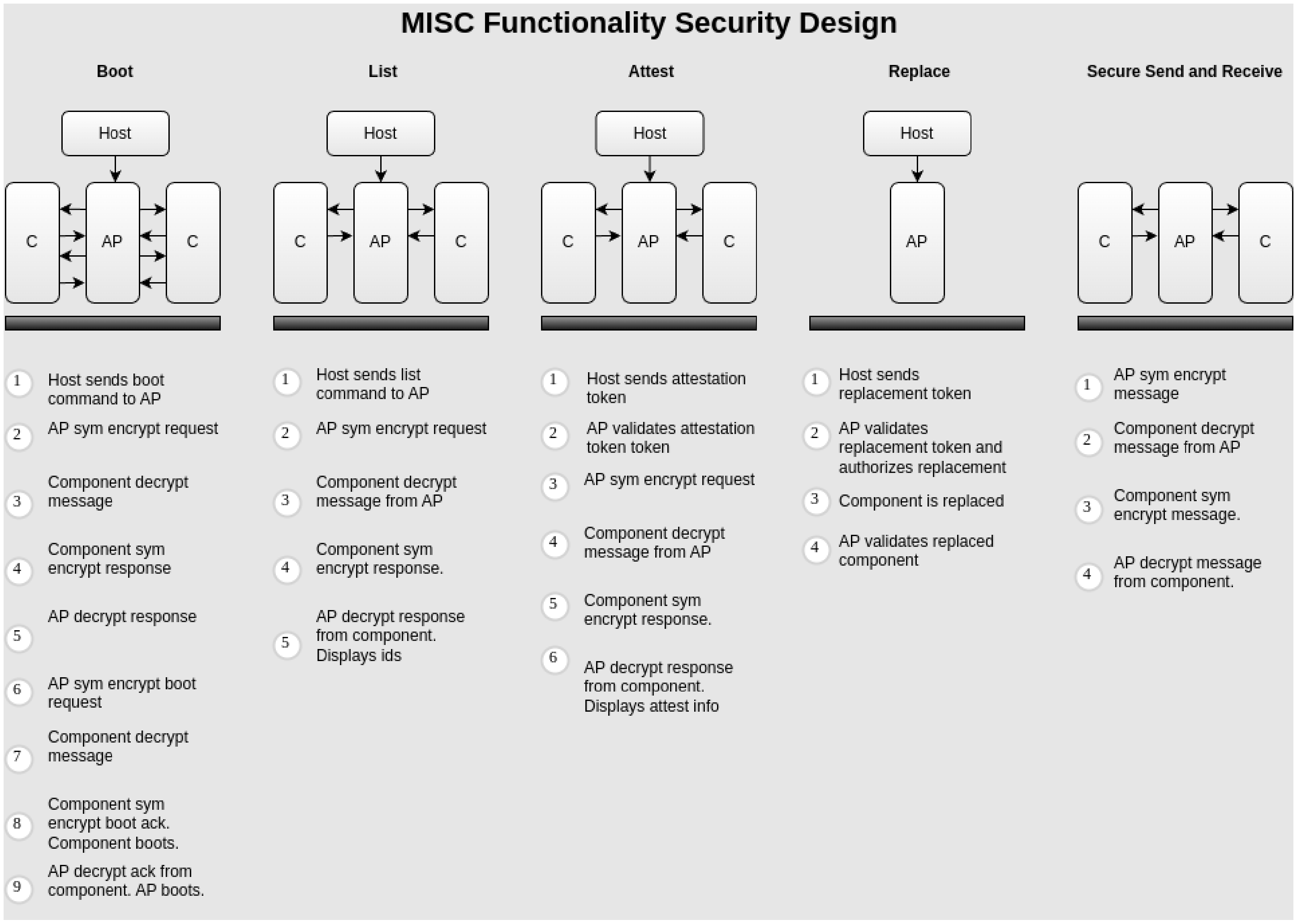
Replace an existing component with a new component validated by the methods outlined in Security Requirement 2 within the five-second timeframe.

Boot

The AP will check that the installed Components are valid, and if both Components pass proceed to boot displaying the relevant boot messages and in event of failure notify the user, within the three second timeframe.

Secure Send & Receive

The MISC will send and receive messages from the AP/Components that have been encrypted and validated as outlined Security Requirement 5 below.


MISC Functionality Security Design
Boot
Host
C
AP
C
1 Host sends boot command to AP
2 AP sym encrypt request
3 Component decrypt message
4 Component sym encrypt response
5 AP decrypt response
6 AP sym encrypt boot request
7 Component decrypt message
8 Component sym encrypt boot ack. Component boots.
9 AP decrypt ack from component. AP boots.
List
Host
C
AP
C
1 Host sends list command to AP
2 AP sym encrypt request
3 Component decrypt message from AP
4 Component sym encrypt response.
5 AP decrypt response from component. Displays ids
Attest
Host
C
AP
C
1 Host sends attestation token
2 AP validates attestation token token
3 AP sym encrypt request
4 Component decrypt message from AP
5 Component sym encrypt response.
6 AP decrypt response from component. Displays attest info
Replace
Host
AP
1 Host sends replacement token
2 AP validates replacement token and authorizes replacement
3 Component is replaced
4 AP validates replaced component
Secure Send and Receive
C
AP
C
1 AP sym encrypt message
2 Component decrypt message from AP
3 Component sym encrypt message.
4 AP decrypt message from component.

B. Security Requirements

To meet the security requirements, we will encompass three key components: validating the origin of boards from the same deployment factory, ensuring data confidentiality, and securing communications between the host, AP, and components against corruption and unauthorized access. Employing encryption and authentication/authorization mechanisms, we utilize AES for symmetric encryption with a rotating key that is generated and distributed every 5 messages. A symmetric encryption key, distributed to all components during deployment, acts as our initial "stamp of validation." When transmitting a packet, we incorporate random padding, standardizing the packet length before encryption.

Security Requirement 1

The Application Processor (AP) should only boot if all expected Components are present and valid.

During the build phase, we will generate a symmetric key dedicated to component boot validation. Upon the system startup, the Application Processor (AP) will dispatch a new symmetric key to the components for further communications. A boot message will next be sent from the host machine, initiating the validation process and kickstarting their individual boot procedures. The validation of each component involves leveraging its unique component ID and padding, which will be securely transmitted to the AP through the distributed encryption key.

Security Requirement 2

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

During the component's pre-boot stage, triggered by the AP, the component initiates the validation process by sending a message to the AP, marking the commencement of the post-boot stage. The validation of the AP involves utilizing the AP ID and padding, and this information is securely transmitted using the most recently distributed symmetric key to the component. This systematic approach ensures a secure and validated boot process for both the AP and components, enhancing the overall integrity and security of our embedded system.

Security Requirement 3

The Attestation PIN and Replacement Token should be kept confidential.

There is no requirement to disclose the attestation pin or replacement token, so our focus will be on preventing any inadvertent leaks. The same cautious approach applies to our encryption key. This stance might change if the requirement for a "valid component" is removed in the future, potentially altering our considerations for replacement scenarios.

Security Requirement 4

Component Attestation Data should be kept confidential.

Enforcing authentication/authorization

To ensure proper authentication and authorization has occurred, we will ensure that the authorization and authentication are tightly coupled with replacement and attestation functions, not allowing the functions to be called without authentication and authorization.

Timing and protecting against brute forcing

As part of our plan, recognizing the limitations in modifying the attestation pin, replacement token, or timing requirements, we acknowledge the theoretical resilience of approximately 10^7 years for the attestation pin and around 10^{24} years for the replacement token, assuming perfect randomness in their generation. In the interest of enhancing security, we are open to lowering the response time to an order of magnitude while maintaining a consistent enforcement of this revised response time for all attestation and replacement requests. Additionally, the use of rotating keys will defer brute force attacks, as every five messages, a new key will be required. This strategic adjustment aims to improve our security measures without compromising the integrity of our system.

Security Requirement 5

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

As part of our design, during the deployment phase, we will utilize symmetric keys generated in the build process. Additionally, we will generate public and private keys for both the Application Processor (AP) and its components at this stage. At runtime, the AP will dynamically generate a session key at regular intervals. To initiate communication, the AP will transmit the first session key encrypted with the symmetric key. Subsequent communication between the AP and components will be encrypted using the established session key, and each message will be signed with the private key of the respective device. Periodically, every n messages, the AP will establish a new session key signed with its private key and encrypted with the old session key. This new session key will then be distributed to components for future communication. This cryptographic protocol will ensure secure and authenticated communication between the access point and its components.

C. Testing and Handoff

We will strictly test our designed system and conduct a thorough code review, ensuring its reliability and completeness according to functional and security requirements. During the code review, our focus will be on identifying issues related to functionality, readability, and security feature implementation. Following this comprehensive review, we will create a structured plan for necessary adjustments and refining. Through an iterative process, our goal is to eliminate potential bugs and enhance the overall quality of our design. The finalized version of the completed design will be submitted for the handoff phase.

Conclusion

In the development of the embedded medical system, choosing Rust for secure firmware development is our strategic decision. Rust's memory safety features play a vital role in robust code, which is crucial for maintaining the integrity of medical systems. The use of custom-built libraries further ensures tailored solutions to meet the specific functional requirements of our system.

Security holds crucial importance in the medical domain, and we prioritize it by implementing robust measures such as encryption and digital signing to safeguard sensitive data. Protection against memory corruption is integral in preventing vulnerabilities that could potentially compromise the stability of the system. Additionally, we employ authentication mechanisms to further enhance security, ensuring that only authorized entities have access to and can interact with the medical system. These comprehensive security measures collectively enhance our commitment to safeguarding the integrity and confidentiality of medical data and system functionality. Through the adoption of these security measures, our embedded medical system not only meets functional specifications but also adheres to security requirements, promoting the reliability and trustworthiness of the overall solution.