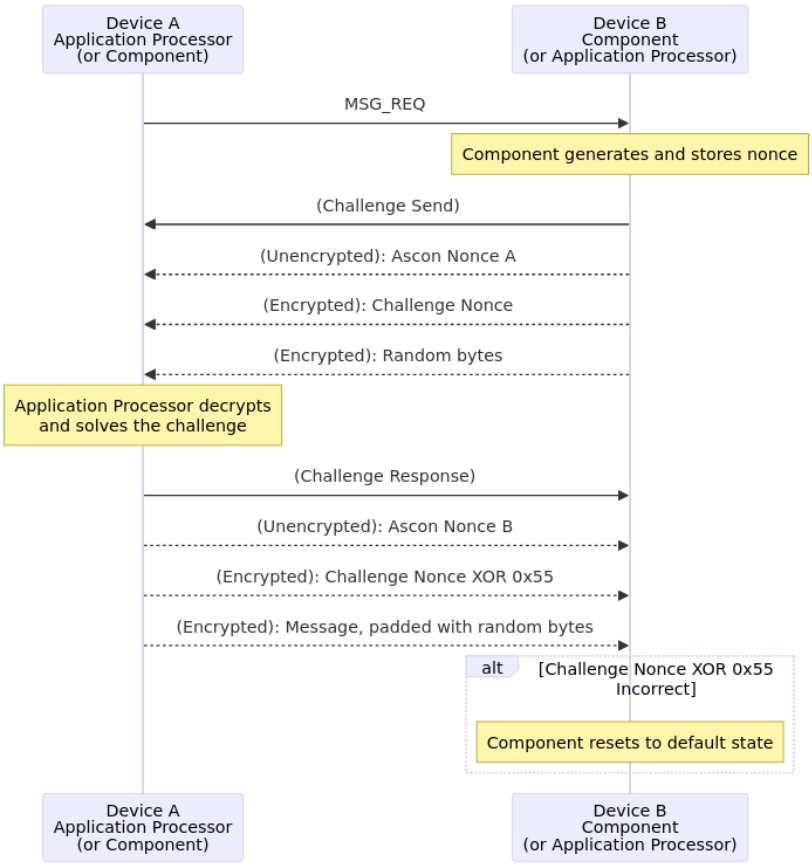
1 HIDE Protocol Communication Layer

We implement an extra communication layer between the I2C layer and the application layer, which we refer to as the HIDE protocol. The HIDE protocol ensures that all messages maintain confidentiality, integrity, authenticity, and non-replayability. We require all messages sent between the AP and the Component to use HIDE.

HIDE effectively turns each message into a three-way challenge-response handshake. The sender first initiates a message request. The receiver will then send a random, encrypted challenge. The sender will then decrypt the challenge, solve it, and encrypt the challenge response to be sent along with the actual message. To solve the challenge nonce, the sender must perform a bitwise XOR of `0x55` with each byte in the challenge nonce.

We use the Authenticated Encryption (AE) cipher, Ascon, for our cryptographic scheme. We chose Ascon since it was selected in the NIST Lightweight Cryptography competition and has a masked software implementation that has been tested against various power analysis and hardware attacks.

1.1 HIDE Protocol



We use Ascon's associated data feature to validate each message. The associated data is 8 bytes, with the first 4 bytes being the component ID, the fifth byte being the HIDE message magic byte, and the last three bytes being null bytes.

Each direction of communication uses a different symmetric encryption key, meaning there are two encryption keys:

- $K_{AP,C}$ is the key for messages sent from the Application Processor to the Component.
- $K_{C,AP}$ is the key for messages sent from the Component to the AP.

Every AP and Components built from the same deployment will share the same keys.

1.1.1 MSG_REQ

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x40`

1.1.2 CHAL_SEND

Description TODO.

Name	Offset	Size (bytes)	Content
Ascon Nonce	`0x00`	16	`\x?? * 16`
Encrypted data	`0x10`	96	Challenge Nonce (16 bytes) + Random bytes (80 bytes)

> [!WARNING] Ascon Nonce should be randomly uniquely generated for all messages

1.1.3 CHAL_RESP

Description TODO.

Name	Offset	Size (bytes)	Content
Ascon Nonce	`0x00`	16	`\x?? * 16`
Encrypted data	`0x10`	96	Solved Challenge Nonce (16 bytes) + Message, padded with random bytes (80 bytes)

> [!WARNING] Ascon Nonce should be randomly uniquely generated for all messages
>
> [!NOTE]
> Application messages can only be a maximum of 64 bytes. We provide up to 80 bytes for posterity.

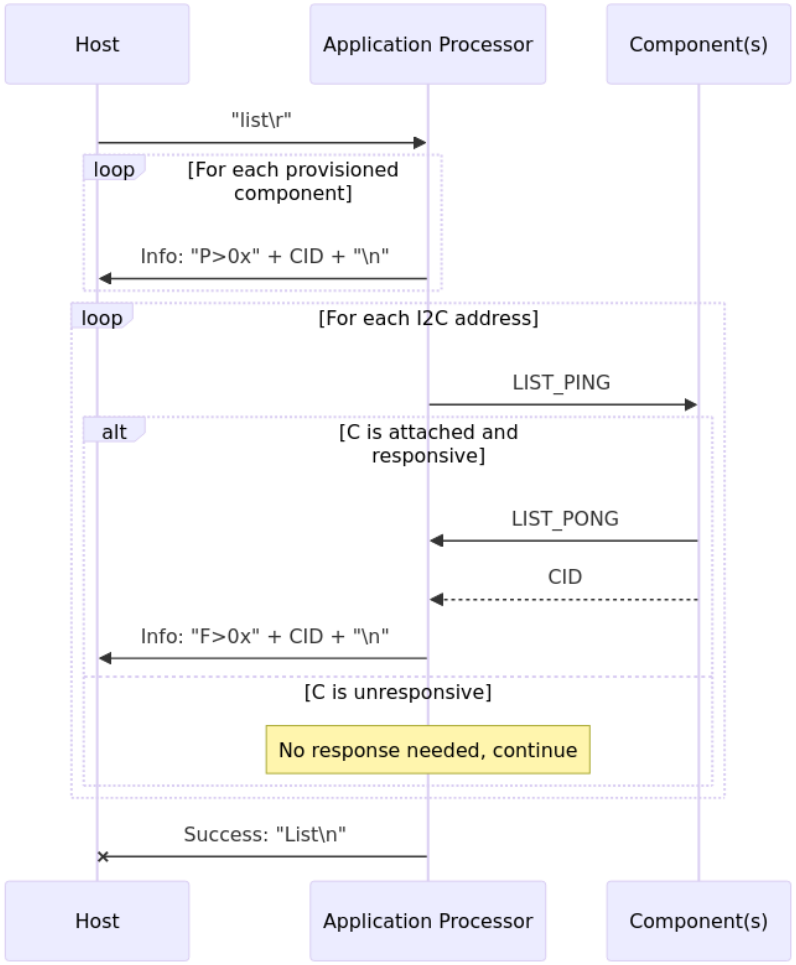
2 MISC Protocol

The Medical Infrastructure Supply Chain (MISC) . All MISC messages are sent over the HIDE protocol.

> [!NOTE]
> "TTT" refers to "total transaction time" and is used to ensure timing functionality requirements are met.

2.1 List Components

The host will ask the Application Processor to "list" its components. The Application Processor, upon receiving the message from the host, will list its provisioned components. It will then send a magic byte as a ping to every I2C address. For components that are present and responsive, they will send a magic byte pong as well as its component ID, which will prompt the Application Processor to send the component ID to the host.



2.1.1 LIST_PING

This byte is sent to every I2C address. For present components, this indicates that the Application Processor is asking for its component ID.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x50`

2.1.2 LIST_PONG

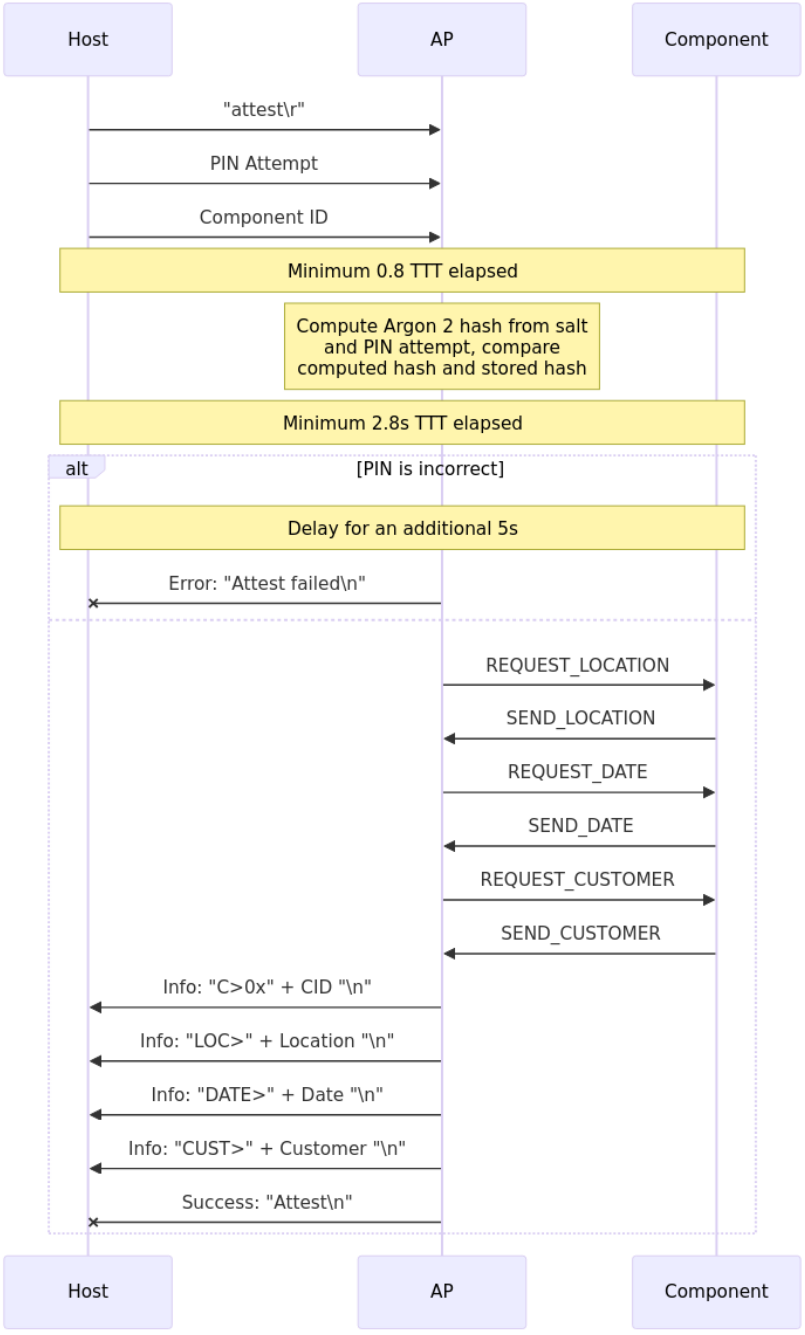
After being prompted by the Application Processor using LIST_PING, an active component will send a response byte and the component ID to the Application Processor.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x51`
Component ID	`0x01-0x05`	4	`\x??\x??\x??\x??`

2.2 Attest Components

Description TODO.

> [!NOTE] The PIN attempt and component ID need to be transmitted at the beginning in a way that the host tool can understand.


Host
AP
Component
"attest\r"
PIN Attempt
Component ID
Minimum 0.8 TTT elapsed
Compute Argon 2 hash from salt and PIN attempt, compare computed hash and stored hash
Minimum 2.8s TTT elapsed
alt
[PIN is incorrect]
Delay for an additional 5s
Error: "Attest failed\n"
REQUEST_LOCATION
SEND_LOCATION
REQUEST_DATE
SEND_DATE
REQUEST_CUSTOMER
SEND_CUSTOMER
Info: "C>0x" + CID "\n"
Info: "LOC>" + Location "\n"
Info: "DATE>" + Date "\n"
Info: "CUST>" + Customer "\n"
Success: "Attest\n"
Host
AP
Component

2.2.1 REQUEST_LOCATION

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x60`

2.2.2 SEND_LOCATION

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x61`
Location	`0x01`	64	`\x?? * 64`

2.2.3 REQUEST_DATE

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x62`

2.2.4 SEND_DATE

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x63`
Date	`0x01`	64	`\x?? * 64`

2.2.5 REQUEST_CUSTOMER

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x64`

2.2.6 SEND_CUSTOMER

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x65`
Customer	`0x01`	64	`\x?? * 64`

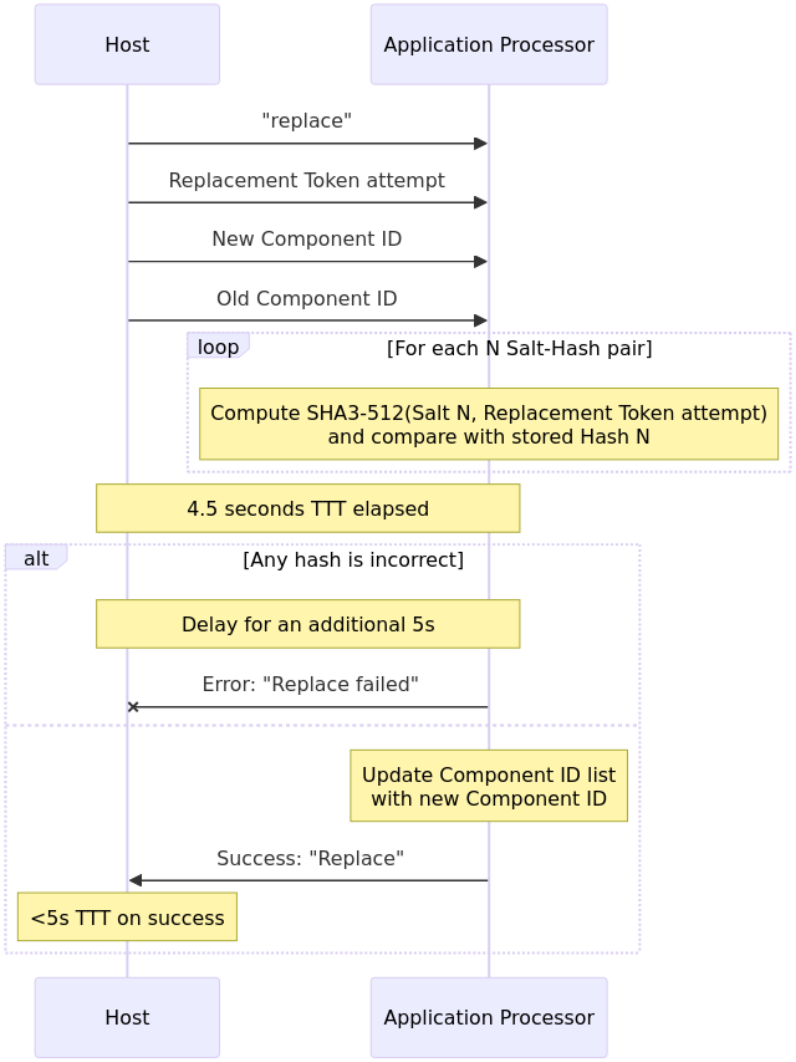
2.3 Replace Components

The AP keeps tracks of Components which are provisioned for it by storing their Component IDs. When a Component is replaced, the AP will update its list of provisioned Components with the new Component ID.

Because this is a sensitive operation, the AP requires a Replacement Token to be sent by the host to validate the replacement. The Replacement Token is a 16-byte string that is assigned during the AP's build process. The Replacement Token is used to ensure that the host is authorized to replace the Component.

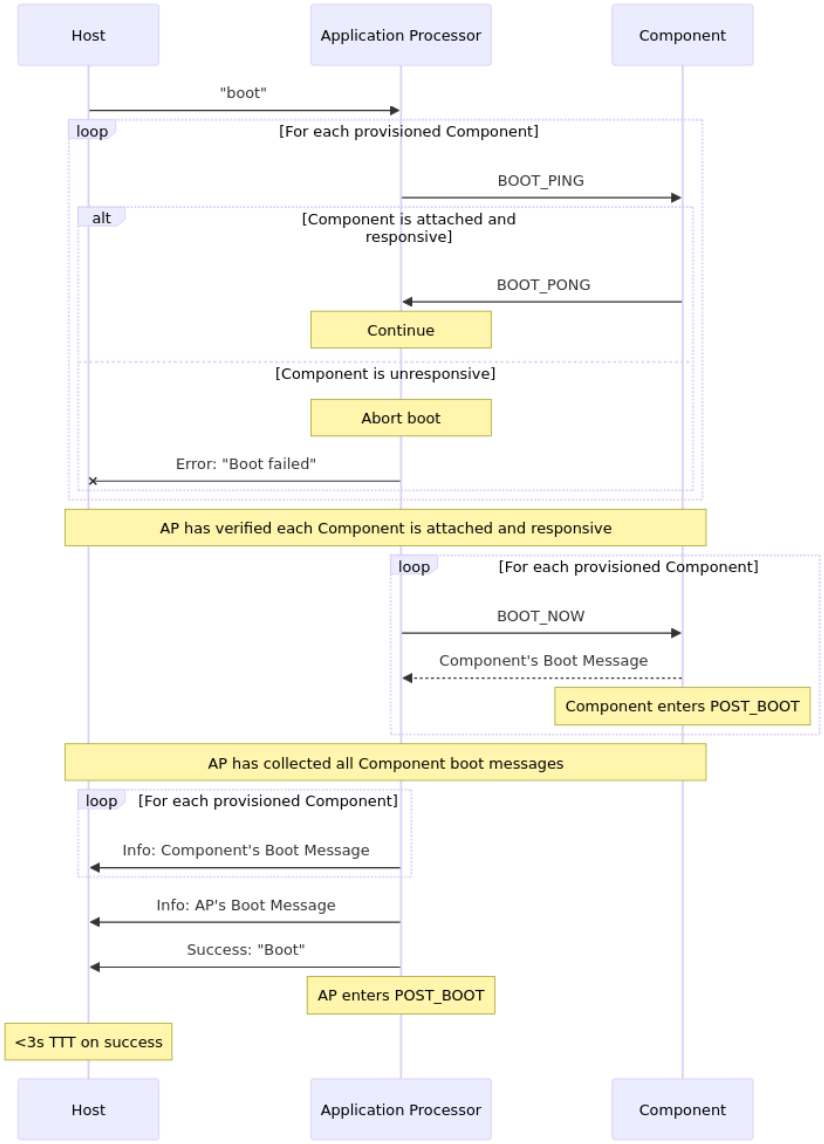
In order to protect the Replacement Token from being compromised, the AP stores the Replacement Token as three uniquely salted SHA3-512 hashes. The salt is a random 16-byte string that is generated during the AP's build process and is prepended to the Replacement Token before hashing. Each salt and hash pair is stored in the AP so that the Replacement Token is never stored in plaintext.

Additionally, a fixed transaction delay of 4.5 seconds is sent in place to prevent brute force attacks.



2.4 Boot Verification

The boot verification process is used to ensure that the Application Processor (AP) only boots if all expected Components are present and valid. The AP will verify that each Component is attached and responsive before booting. The AP will then collect each Component's boot message and send it to the host.



2.4.1 BOOT_PING

Sent from the Application Processor to each Component to "ping" the Component to ensure it is attached and responsive. Expected to receive a BOOT_PONG in response.

Name	Offset	Size (bytes)	Content
Magic	0x00	80	\x80 * 80

2.4.2 BOOT_PONG

Sent from the Component to the Application Processor in response to a BOOT_PING. This indicates that the Component is attached and responsive.

Name	Offset	Size (bytes)	Content
Magic	0x00	80	\x81 * 80

2.4.3 BOOT_NOW

Sent from the Application Processor to each Component to command the Component to boot. The Component will then send its boot message (of length 64) to the Application Processor.

Name	Offset	Size (bytes)	Content
Magic	0x00	80	\x82 * 80

> [!WARNING]
> The component should not respond to a BOOT_NOW if it has not received a BOOT_PING beforehand.

2.5 Post-Boot Communication

Description TODO.

3 Security Requirements

Below we summarize how we achieve the outlined security requirements with our design.

3.1 Security Requirement 1

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

Because of the integrity, authenticity, and non-replayability properties of the HIDE protocol, we ensure that the AP is actively communicating with valid, provisioned Components from the same deployment. A counterfeit Component will not be built on the same deployment and thus will not have access to the Ascon-128 keys required for the encryption layer.

3.2 Security Requirement 2

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

After the AP verifies that each Component is connected and valid, the AP will command each Component to boot. Each Component takes advantage of the integrity, authenticity, and non-replayability properties of the HIDE protocol to confirm that the AP has truly commanded the Component to boot.

3.3 Security Requirement 3

> *"The Attestation PIN and Replacement Token should be kept confidential."*

We use multiple SHA3-512 hashes to validate the Attestation PIN and Replacement Token, preventing compromise of the real PIN and token through side-channel analysis. We also use timers and random delays to effectively prevent brute force and further side-channel attacks.

3.4 Security Requirement 4

> *"Component Attestation Data should be kept confidential."*

Attestation Data is only provided to the AP if the Component receives a command from the AP instructing it to provide Attestation Data. This command is only sent if the AP successfully validates the Attestation PIN. This command cannot be forged due to the properties of the HIDE protocol.

3.5 Security Requiremenet 5

> *"The integrity and authenticity of messages sent and received using the post-boot MISC secure communications functionality should be*

ensured."

The HIDE communication layer uses authenticated encryption with Ascon-128 to verify the authenticity of the sender and receiver. The challenge-response nature of the HIDE protocol and the use of nonces prevents the replay of messages.