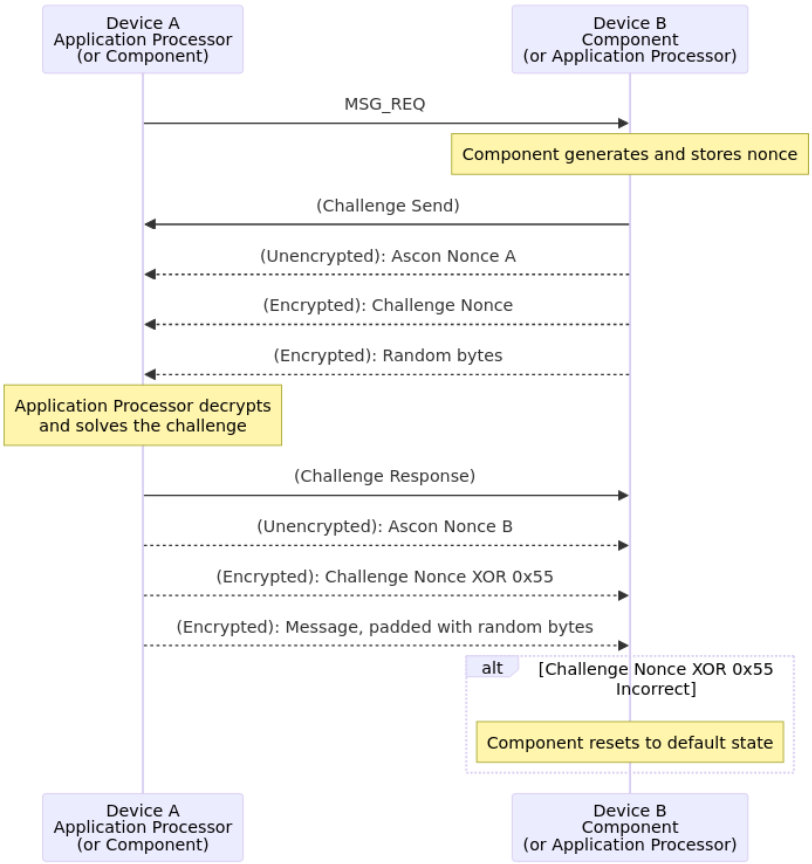
1 HIDE Protocol Communication Layer

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.

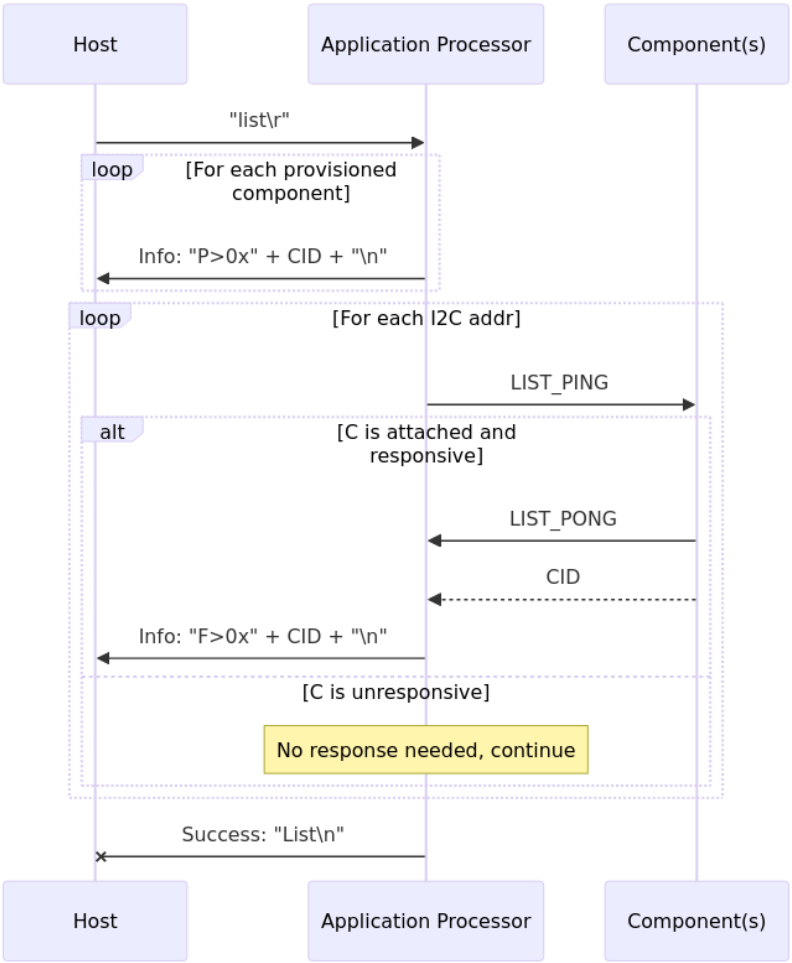
2 MISC Protocol

Description TODO. 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

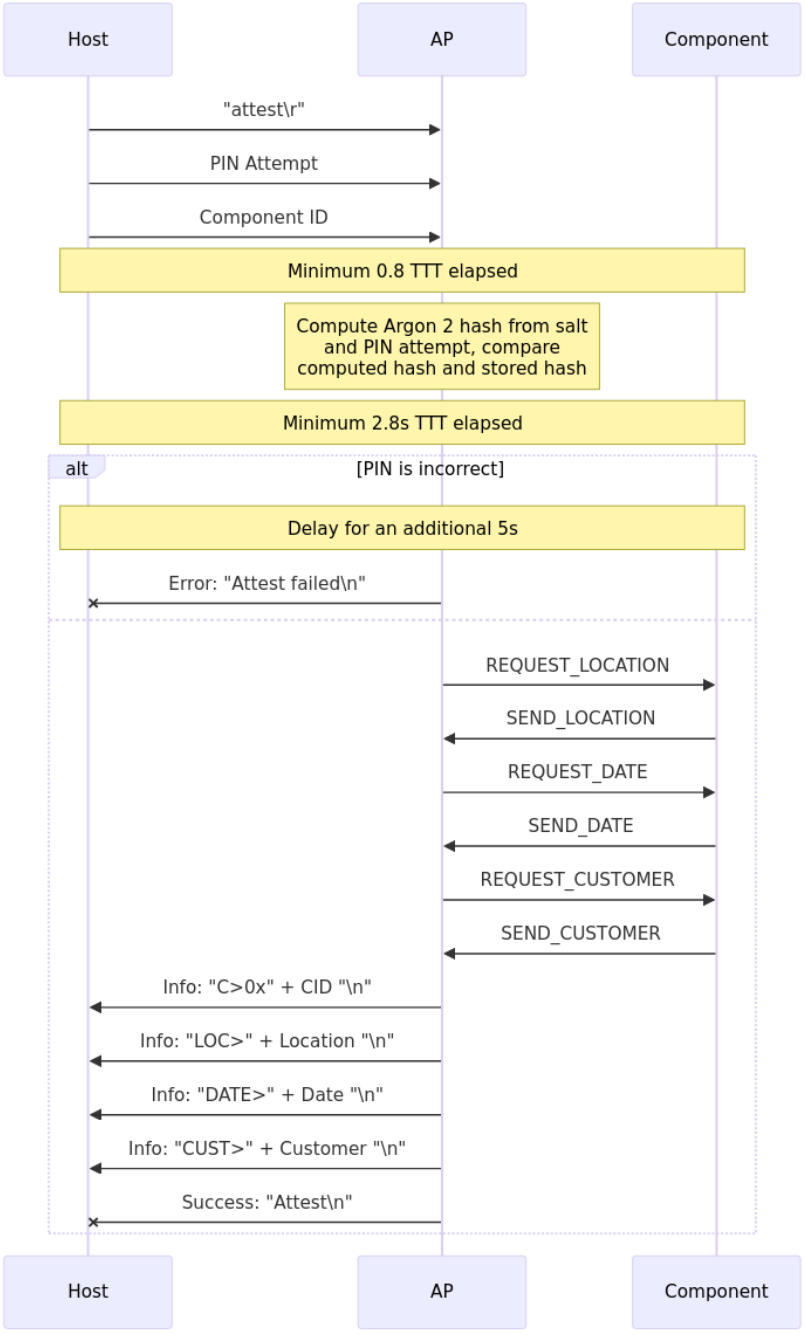
After being prompted by the Application Processor using LIST_PING, an active component will send a response byte and the component ID to the Application Processor.

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`

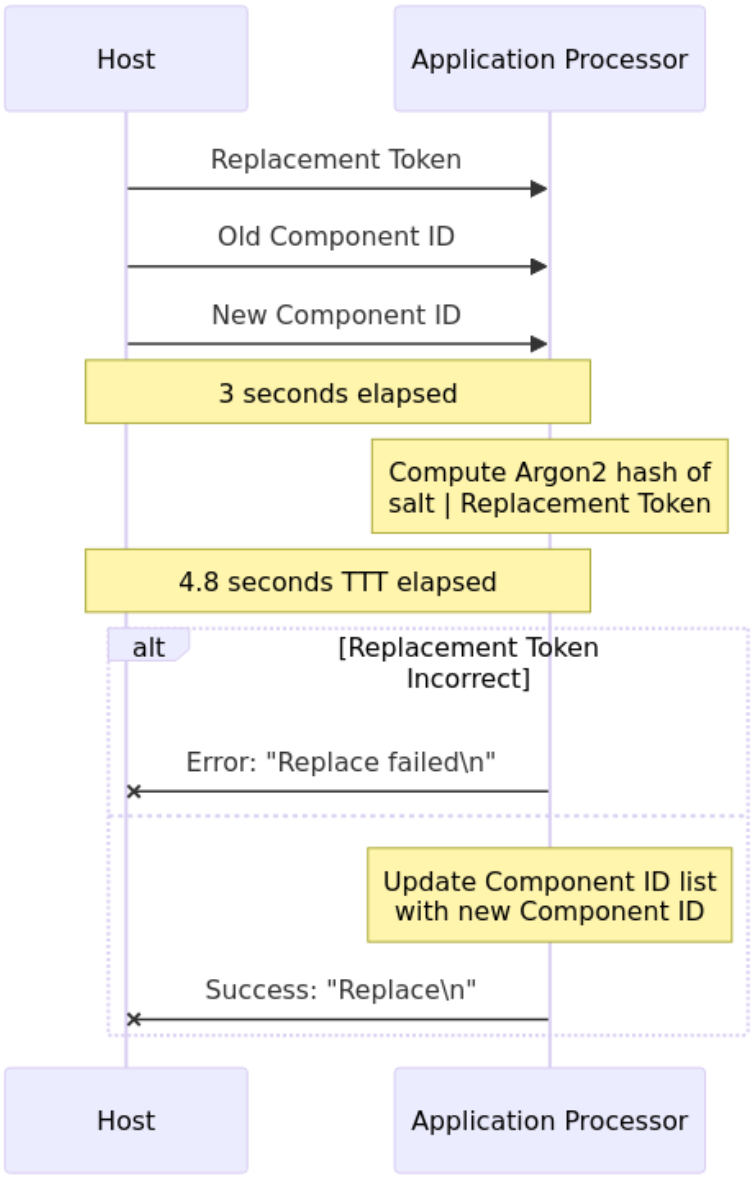
2.2.6 SEND_CUSTOMER

Description TODO.

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

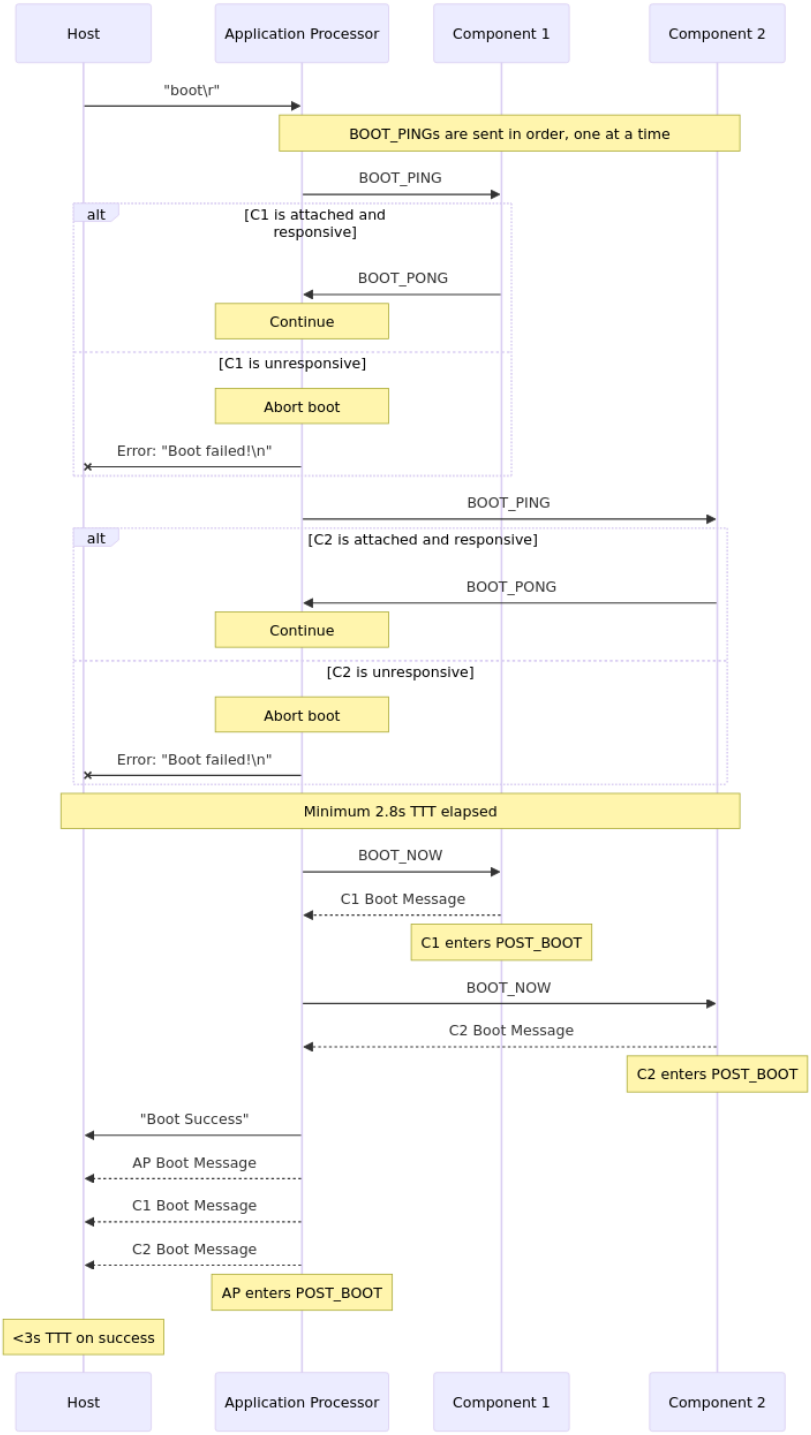
2.3 Replace Components

Description TODO.



2.4 Boot Verification

Description TODO.


Host
Application Processor
Component 1
Component 2
"boot\r"
BOOT_PINGs are sent in order, one at a time
BOOT_PING
alt
[C1 is attached and responsive]
BOOT_PONG
Continue
[C1 is unresponsive]
Abort boot
Error: "Boot failed!\n"
BOOT_PING
alt
[C2 is attached and responsive]
BOOT_PONG
Continue
[C2 is unresponsive]
Abort boot
Error: "Boot failed!\n"
Minimum 2.8s TTT elapsed
BOOT_NOW
C1 Boot Message
C1 enters POST_BOOT
BOOT_NOW
C2 Boot Message
C2 enters POST_BOOT
"Boot Success"
AP Boot Message
C1 Boot Message
C2 Boot Message
AP enters POST_BOOT
<3s TTT on success
Host
Application Processor
Component 1
Component 2

2.4.1 BOOT_PING

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x80`

2.4.2 BOOT_PONG

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x81`

2.4.3 BOOT_NOW

Description TODO.

Name	Offset	Size (bytes)	Content
Magic	`0x00`	1	`\x82`

2.5 Post-Boot Communication

Description TODO.