MITRE eCTF 2024 Design Document

SIGPwny at the University of Illinois Urbana-Champaign (UIUC)
Version: 0.0 (Initial Design)

Table of Contents

Overview

We will be writing our design in Rust, with a custom build system in place to ensure interoperability with post-boot code written in C. We chose Rust for its strict type checking, memory safety, and overall security advantages over C.

HIDE Protocol Communication Layer

“HIDE” is a temporary working name for initial design documentation and will likely be changed in the future. It does not stand for anything.

We define the “application layer” as the messages sent between the AP and Components that directly serve the functional requirements. We are implementing an extra communication layer between the I2C layer and the application layer. This extra communication layer, which we call the HIDE protocol, will ensure that messages maintain confidentiality, integrity, authenticity, and non-replayability.

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.

Although Ascon provides most of the security properties that we need, it provides no guarantees about the “freshness” of messages or their correct routing, resulting in the possibility of replay attacks. To solve this, we implement a three-way handshake that occurs for every message, where a challenge-response system is used to prevent message reuse. Every message is encrypted with Ascon, with a different symmetric key for each direction of communication.

Suppose *A* needs to send an arbitrary message, *m*, to *B*.

1. *A* will first send a message request to *B*. This message is encrypted with Ascon using the key *K_{AB}*.
2. *B* will respond to *A* with a random challenge nonce. This message is encrypted with Ascon using the key *K_{BA}*.
3. *A* will respond to *B* with the solved challenge along with message *m*. This message is encrypted with Ascon using the key *K_{AB}*.

Each message in the HIDE protocol will contain a header, consisting of the HIDE message magic bytes, the sender ID, and the receiver ID. The sender ID and receiver ID are always

validated by the receive function. We define the AP's ID as three null bytes, followed by its I2C address byte.

Our implementation of the HIDE protocol will expose two interfaces, `send(destination ID, message)` and `receive(source ID)`. The `receive` function will always validate the source ID, destination ID, and integrity of incoming messages.

RNG

Although a True RNG (TRNG) component is provided on the boards this year, we are not relying on it and using it as just one source of entropy. To generate secure randomness for encryption and nonces, we plan to initialize a CSPRNG with the secret RNG initial value and implement continuous reseeding using TRNG and other entropy sources such as the CPU temperature sensor.

Additionally, we strive to reduce known plaintext in our messages to prevent novel cryptanalysis or side-channel analysis attacks. We achieve this by randomizing the command magic bytes during the build process for each command sent on the application layer.

Secrets

Deployment global secrets:

- HIDE symmetric key for AP -> Component communication
- HIDE symmetric key for Component -> AP communication
- Randomized command magic bytes

AP secrets:

- RNG initial value
- Attestation PIN salt
- Replacement Token salt

Component secrets:

- RNG initial value

RNG initial values are provided for the individual AP and Components during the build process. These RNG initial values will be constantly updated as the device runs to create a fallback in case TRNG is compromised.

MISC Functionality

For every application protocol listed below, the HIDE protocol will be used for communication.

List Components

This functionality identifies which of the provisioned Component IDs are connected over I2C. The provisioned Component IDs are persistently stored in the flash memory of the AP.

For each provisioned Component ID, the AP will do the following:

1. The AP logs to the host that the Component ID is provisioned.
2. The AP sends a ping request to the Component ID using the HIDE communication layer.
3. If the Component is present, the Component responds with a pong message to the AP. Then, the AP will log to the host that the Component was found.
4. If the AP does not receive a pong message within 1 second, the AP will log to the host that the Component was not found.

Attestation of Components

The hashing function used for validating the Attestation PIN will be Argon2. Argon2 was selected due to its resistance to side-channel attacks. The hash computation will consist of multiple rounds of Argon2, with the exact count to be determined during development. The correct PIN will be appended to a secret Attestation PIN salt and then hashed. This hash is stored in the flash memory of the AP.

1. The host machine sends the AP the Attestation PIN and the Component ID to retrieve attestation data.
2. The AP will wait at least 1 second.
3. The AP will compute the Argon2 `hash(Attestation PIN salt | PIN attempt)` with the configured number of rounds.
4. The AP will compare the computed hash with the stored hash in flash memory.
5. The AP will wait until 2.8 seconds total has elapsed from the start of the transaction.
6. If the PIN is correct, the AP will command the corresponding Component to provide its Attestation Data. Then, the AP will log the Attestation Data to the host.
7. If the PIN is incorrect, the AP will delay an additional 2 seconds before telling the host the PIN is invalid.

Replacement of Components

To replace a Component on the medical device, a valid Replacement Token must be provided. Similar to Attestation, the hashing function used for validating the Replacement Token will be Argon2. The correct Replacement Token will be appended to a secret Replacement Token salt and then hashed. This hash is stored in the flash memory of the AP.

1. The host machine sends the AP the Replacement Token, old Component ID, and new Component ID.
2. The AP will wait at least 3 seconds.
3. The AP will compute the Argon2 `hash(Replacement Token salt | Replacement Token attempt)` with the configured number of rounds.
4. The AP will compare the computed hash with the stored hash in flash memory.
5. The AP will wait until 4.8 seconds total have elapsed from the start of the transaction.
6. If the Replacement Token is correct, the AP will update the provisioned Component ID list stored in flash memory with the new Component ID, replacing the old one.
7. If the Replacement Token is incorrect, the transaction will simply end.

Boot Verification

Before booting the device, the AP will verify that the provisioned Components are both present and valid.

1. The host machine instructs the AP to boot.
2. For each provisioned Component ID, the AP will do the following:
    a. The AP will send a “boot ping” request to the corresponding Component.
    b. If the Component is present, the Component responds with a “boot pong” message to the AP. Then, the AP will continue to the next Component.
    c. If the AP does not receive a pong message within 1 second, the AP will abort.
3. Once the AP has received pongs from all provisioned Components, the AP will wait until at least 2.8 seconds have passed in the current transaction.
4. The AP will send the “go boot” command to each provisioned Component. Each Component will then begin post-boot.
5. The AP begins post-boot.

Post-Boot Secure Communication

We will simply build a C interface for our HIDE communication layer written in Rust for `secure_send` and `secure_receive`. Post-boot communication will use this C interface for the HIDE protocol, ensuring confidentiality, integrity, authenticity, and non-replayability.

Security Requirements

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

SR1

“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 can rely on messages sent on this channel to confirm the active presence of valid, registered Components. A counterfeit Component will not have access to the secret HIDE keys and thus will not even be able to communicate with the AP.

SR2

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

After the AP determines that it is appropriate 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.

SR3

“The Attestation PIN and Replacement Token should be kept confidential.”

We use Argon2 hashes to validate the Attestation PIN and Replacement Token, preventing compromise of the real PIN and Token via leakage. We also use timers and delays to effectively prevent brute force or timing side-channel attacks.

SR4

“Component Attestation Data should be kept confidential.”

Attestation Data is only provided 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 challenge-response properties of the HIDE protocol.

SR5

“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 Ascon symmetric keys to verify the authenticity of the sender and receiver. The challenge-response nature of the HIDE protocol and the use of nonces prevents the duplication of messages.