MIT eCTF 2024 Design Document

Summary

Our defenses consist of three main components:

1. Messaging scheme with authenticated encryption and associated data
2. Challenge-response for verifying liveliness of AP and Components
3. Redundant checks & random delays along critical paths to frustrate glitching attacks

The messaging scheme is explained in more detail at the end of the document, but the key aspects that provide us with integrity, authenticity, and protecting us from replay attacks are:

- All devices in a deployment use a shared secret key for ChaCha20-Poly1305
- Randomly generated nonces per-boot are also used as sequence numbers
- Messages with unexpected nonces/sequence numbers are rejected

Validity and integrity of devices is measured by their ability to communicate within the authenticated messaging scheme, and by their ability to answer random challenges.

Security requirements

Here we briefly justify our adherence to the various security requirements.

SR 1

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

The AP requires that all provisioned Components correctly respond to a random challenge before commanding Components to boot, and subsequently booting itself.

SR 2

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

Components require that an AP correctly responds to a random challenge before it will follow commands to boot.

SR 3

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

The Attestation PIN and Replacement Token are never stored in plaintext on the AP firmware. Each are salted and hashed at compile-time. Guesses from the user are hashed with the corresponding salt, and compared with the expected result using constant-time comparison functions. This eliminates any timing attacks, and greatly reduces the utility of any power-analysis attacks of the memory comparison operations.

SR 4

> Component Attestation Data should be kept confidential. Attestation Data should only be returned by the AP for a valid Component if the user is able to provide the correct Attestation PIN.

Redundant checks for a correct Attestation PIN are made on the AP before commanding the component to provide its Attestation Data. Additionally, Components require the AP answers a random challenge before providing its Attestation Data.

SR 5

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

Our use of ChaCha20-Poly1305 provides for integrity and authenticity of messages. By generating random initial nonces at boot, and using incrementing nonces as sequence numbers for our messages, we prevent replay attacks, even across reboots!

Messaging scheme

See `common/include/common_msg.h` for relevant data structures.

Packet structure

All messages, except for the `LIST` command, are sent within packets.

Each packet has three fields:

- **Authenticated Data** (19 bytes)
- **Authentication** Tag (16 bytes)

- **Ciphertext** (up to 220 bytes)

The **Authenticated Data** field has five sub-fields:

- `opcode` (1 byte): Designates command type and content of ciphertext field
- `nonce` (12 bytes): The nonce used in making this packet
- `component_id` (4 bytes): The component the packet is either to or from
- `length` (1 byte): The length of the ciphertext field
- `for_ap` (1 byte): Boolean designating if sent to AP or sent to Component

The **Authentication Tag** is generated as part of the ChaCha20-Poly1305 encryption function, and is used to verify the integrity of authenticity of both the **Authenticated Data** field and the **Ciphertext** field.

The **Ciphertext** field is encrypted message content that varies depending on the messages opcode. For an `Attest` command, it may be encrypted Attestation Data, while for a `BootReq` command, it may contain an encrypted challenge.

Devices first verify that a received packet has **Authenticated Data** fields it expects, then verifies the entire packet against the **Authentication Tag**, then decrypts the **Ciphertext** field and acts upon the received `opcode`.

Opcodes

There are 6 valid opcodes:

1. `Init`: Initiates a new session with a device. Used once per power-cycle per component.
2. `AttestReq`: Prelude to `Attest`, carries challenges between AP and component.
3. `Attest`: Carries an AP's response to a challenge, or Attestation Data.
4. `BootReq`: Prelude to `Boot`, carries challenges between AP and Component.
5. `Boot`: Carries an AP's response to a challenge, or a Boot Message.
6. `PostBoot`: Designates a message is part of the Post-Boot scheme.

Session initialization

See `application_processor/src/ap_session.c` for relevant code.

A "session" refers to the set of an `outgoing_nonce` and an `incoming_nonce` an AP associates with each provisioned Component. Before any authenticated message can be sent to a Component (i.e., any opcode besides `Init`), the AP must initialize a session. This is not explicitly commanded by any user and is handled automatically.

The AP will generate a random `outgoing_nonce` and include this nonce in its `Init` packet to a Component. Upon receiving this packet, the Component will save the included nonce as its `incoming_nonce`, and return its own randomly generated `outgoing_nonce` in its response `Init` packet. The AP verifies that, in its received response `Init` packet, the Component has included the nonce the AP originally sent. If so, the `AP` saves the incoming nonce from the Component.

Authenticated packets sent to a Component are ignored when a session has not been initialized. Additionally, packets are ignored when the packet's nonce does not match the stored expected incoming nonce.

Outgoing nonces are incremented after every successful packet construction (i.e., after a successful call to `make_mit_packet`), and incoming nonces are incremented after every valid packet is received from a give component (see end of `issue_cmd`).

User commands

Here we detail the messaging sequence for each of the Attest, Boot, List, and Replace commands.

On Challenges

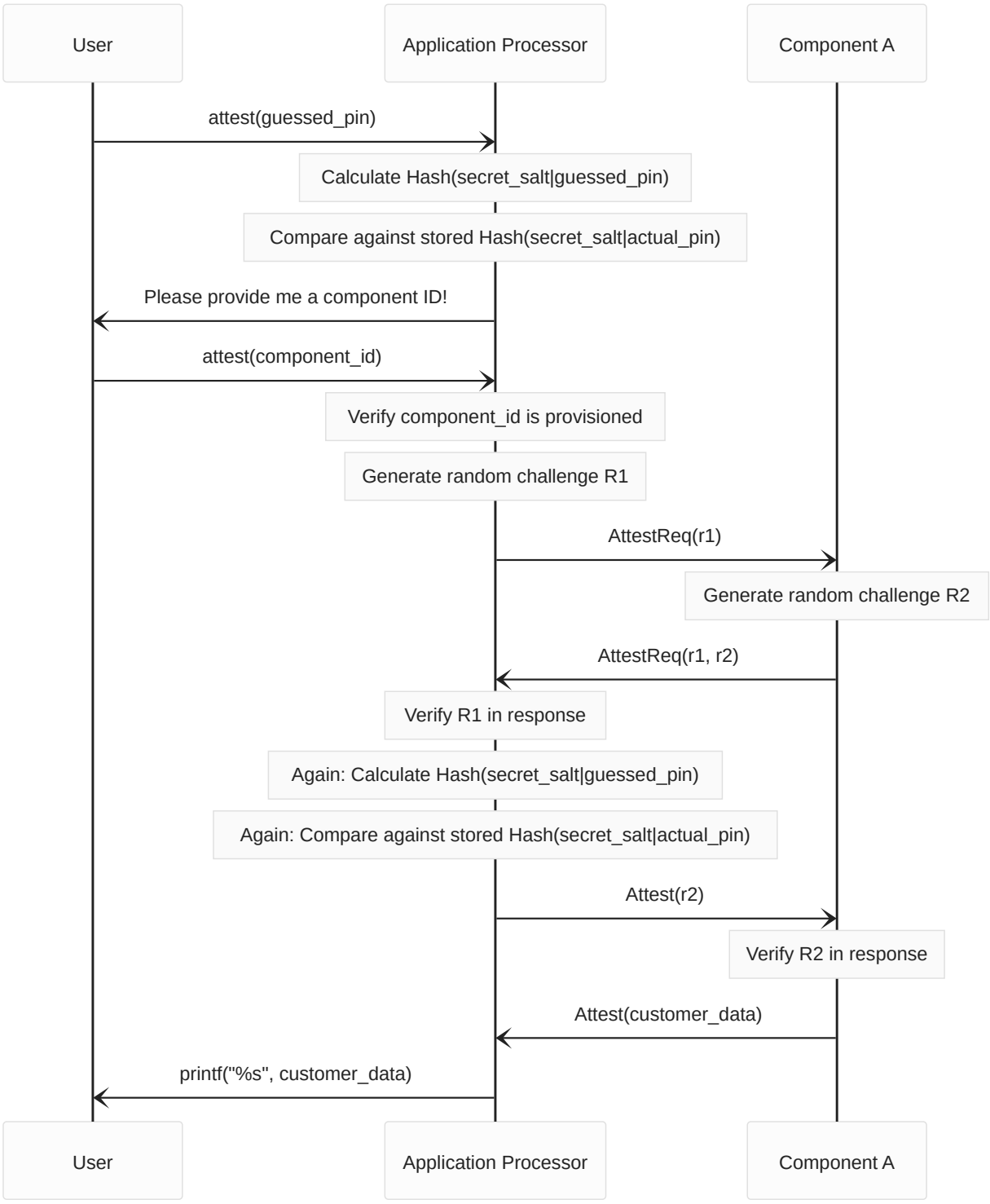
Note that our "challenges" are simply generating a random sequence of bytes, and verifying that that same random sequence is included in the next returned message. If we receive a valid (i.e. authenticated) message from a device with the same sequence included, it means that we just talked to a live & active device as part of our current per-power-cycle session.

These challenges ensure that a replayed sequence of packets to or from a device, perhaps in an attempt to fake the presence of a component that is not actually connected, won't succeed, as they will include a different response to the live device's challenge.

Attest

The attest command is for Users to fetch Customer Data from Components.

This scheme puts all trust on the AP to validate the guessed_pin.

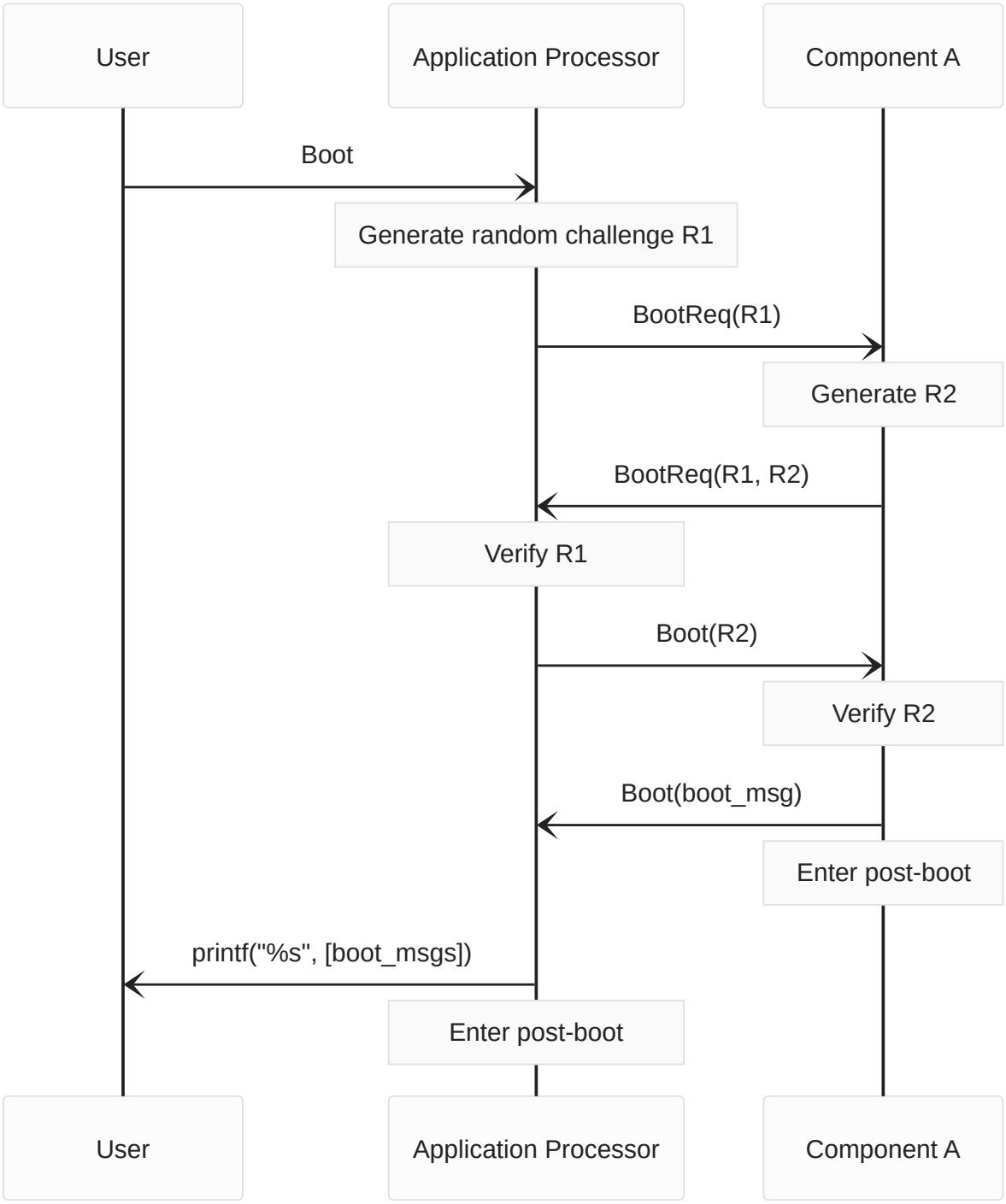


Boot

This command instructs the AP to enter the post-boot phase.

The example below is shown only for the case of an AP provisioned with one component. For an AP with multiple components, the AP would challenge all components with `BootReq`s before it proceeds to send any `Boot` commands.

This scheme puts all trust on the AP to correctly determine all components are connected before sending `Boot` commands.

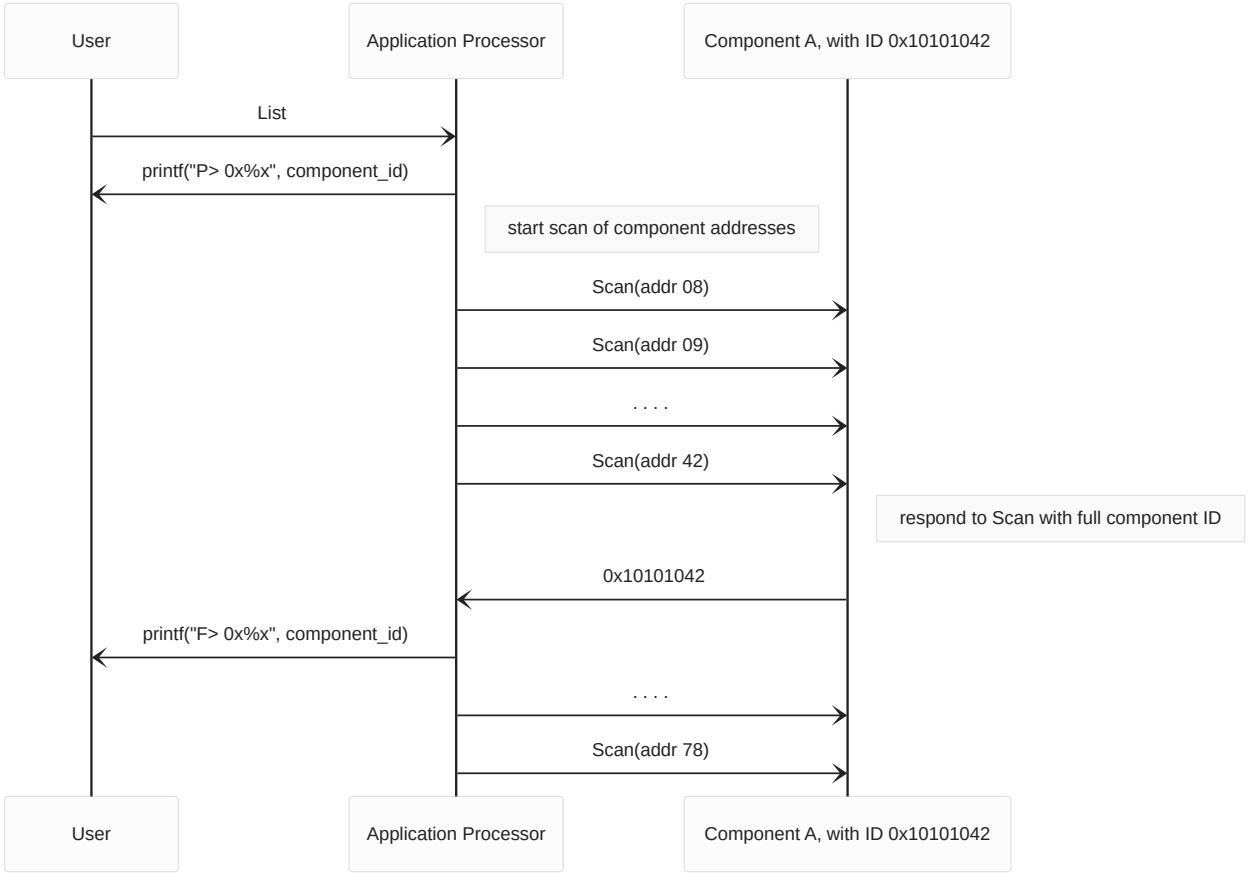


List

This is an unauthenticated command, in order to reduce the complexity of our session management.

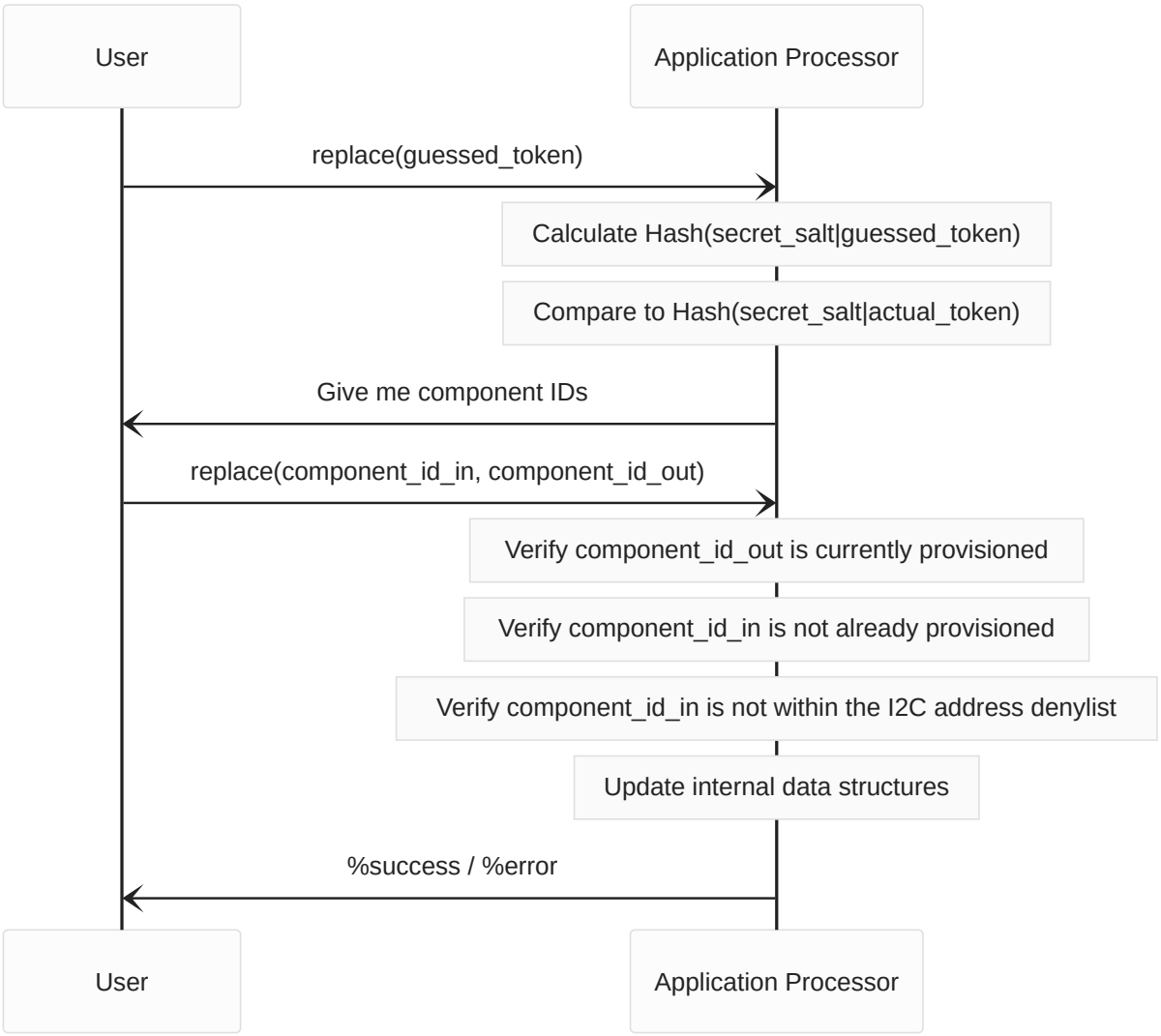
Scan packets from the AP are 4-bytes long, with the last byte being the I2C address.

If a component receives a 4-byte message, then it will respond with its component ID (which is 4-bytes) and not attempt any sort of decryption, authentication, or increase a nonce.



Replace

This command lets a user swap out a provisioned component for another component.



Additional libraries used

We use wolfSSL for generating our runtime SHA256 hashes, for encryption & decryption in ChaCha20-Poly1305, and for constant-time memory comparisons.

We use sha256_literal for generating SHA256 hashes of our Attestation Pin and Replacement Token at compile-time.