VT
Virginia Tech

Rohit Mehta, Rohit Sathye, Wesley Flynn, Raghav Agarwal
Advised by: Dr. Matthew Hicks, PhD & Dr. David Raymond, PhD
April 22, 2024

Design Overview

This section could include the following:

1. ECDSA Based Authentication (SSL-TLS):

- We have designed our protocol based on the crude working of SSL-TLS.
- We use Elliptic Curve Cryptography for implementing Digital Signature Algorithm.

2. Software/hardware security features:

- On-board TRNG to produce a random nonce
- Key Generation Python script using 'ecdsa' module
- Low power and High Latency ECC and SHA2 Libraries

Defensive Highlight

➢ **Secure Boot (Implemented) :**

- This feature enables a two-way Authentication of Application Processor and the individual Components.
- Keys are provisioned to the AP and the Components during the Build process.
- A pair of Keys for AP and a pair for Components are obtained from the sec256k1 ECC curve during the Make process and stored in header files.
- TRNG based nonce generation offers protection against Replay attacks.
- Secure Boot works due to the TRNG based nonce generation and security offered by ECDSA.

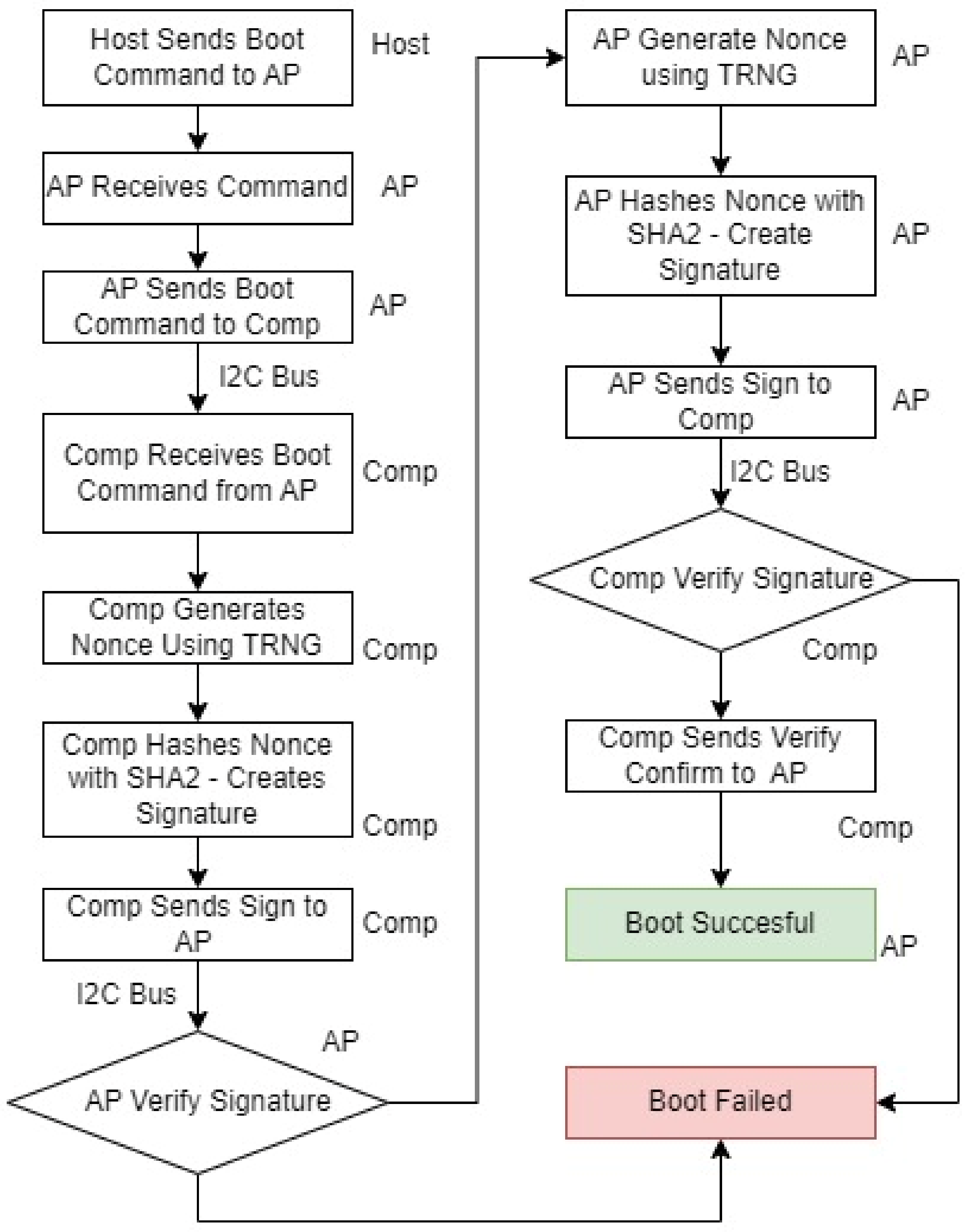
➢ **Secure Attest (Implemented) :**

- Feature Offers protection against AP spoofing.
- Uses Authentication Based on ECDSA used in Secure Boot.
- User Enters Pin and AP verifies the PIN.
- Once PIN is accurately verified, AP sends Attest command to Component.
- Component receives command and generate nonce using TRNG , hashes the nonce using SHA2 and sends to AP for verification. After successful verification, AP initiates the same process.
- Attest Data can be Encrypted using AES by the component and decrypted by the AP **(Suggestive Update)**

➢ **Secure Replace (Suggestive Update)**

- Hash the plain text Replace pin during the Build process and store in Header file.
- Use 'strncmp' to compare hash of user input replace pin and pin present in Header File.
- 'strncmp' provides protection against Buffer Overflow.
- Could not implement in current design due to inconsistency in our own SHA2 library and the 'Hashlib' offered by python

Secure Boot Flowchart


Host Sends Boot Command to AP
Host
AP Receives Command
AP
AP Sends Boot Command to Comp
AP
I2C Bus
Comp Receives Boot Command from AP
Comp
Comp Generates Nonce Using TRNG
Comp
Comp Hashes Nonce with SHA2 - Creates Signature
Comp
Comp Sends Sign to AP
Comp
I2C Bus
AP
AP Verify Signature
AP Generate Nonce using TRNG
AP
AP Hashes Nonce with SHA2 - Create Signature
AP
AP Sends Sign to Comp
AP
I2C Bus
Comp Verify Signature
Comp
Comp Sends Verify Confirm to AP
Comp
Boot Succesful
AP
Boot Failed

Offensive Highlight

This section should include:

➢ **We Developed an attack based on the Replay Attack and exploiting the MD5 Hash Collision**

➢ **Attack Description**

- Victim Team used urandom() function in 'make' file to generate a 12 alphanumeric character nonce (password) for AP and individual components.
- These passwords were hashed using MD5 and stored in header files and sent through I2C in between AP and Comp.
- We serially printed these MD5 hashes using print_info() and then used an online MD5 Hash Decrypting Tool.
- Then we used the output of that tool as the password to spoof AP of the victim design.

➢ **Proposed Fix:**

- Using a safer Hashing algorithm like SHA256 instead of MD5.
- Use 'strncmp' function with a character instead of strcmp

References

1. www.digicert.com/blog/digital-trust-for-connected-medical-devices
2. Waruhari, Philomena and Lawrence Nderu. “An Elliptic curve digital signature algorithm (ECDSA) for securing data : an exemplar of securing patient's data.” (2017).
3. https://github.com/corkami/collisions
4. https://www.dcode.fr/md5-hash
5. https://pypi.org/project/ecdsa/