2024 Design Document

Secure MISC

0xDACC

February 26, 2024

Header		Payload
Packet Magic	Checksum	Length
(1 byte)	(4 bytes)	(1 byte)
0x4C		0x00

Table 1: List Packet

Header		Payload	
Packet Magic	Checksum	Length	Data
(1 byte)	(4 bytes)	(1 byte)	(4 bytes)
0x4D		0x04	

Table 2: List Response Packet

1 Proposed List Changes

Use standard I2C packet structure

2 Proposed Attest Changes

Store attestation PIN as a hash with enough rounds that it takes approximately 2 seconds.

- Limits brute force attempts
- Makes raw PIN unable to be extracted from flash

Wrap attestation symmetric key with attestation PIN hash

Store attestation data encrypted with unwrapped symmetric key

- Also limits brute force and makes PIN unreadable from flash

Meets SR3 and SR4

3 Proposed Replace Changes

Store replacement token as a hash

Header		Payload		
Packet Magic	Checksum	Length	Data	Signature
(1 byte)	(4 bytes)	(1 byte)	(6 bytes)	(65 bytes)
0xAA		0x06	0x415454455354	

Table 3: Attestation Data Packet

Header		Payload		
Packet Magic	Checksum	Length	Attestation Data	Signature
(1 byte)	(4 bytes)	(1 byte)	(192 bytes)	(65 bytes)
0xFA		0xC0		

Table 4: Attestation ACK Packet

- Makes token unable to be extracted from flash
- Highly unlikely that the token can be brute forced

Verify component authenticity

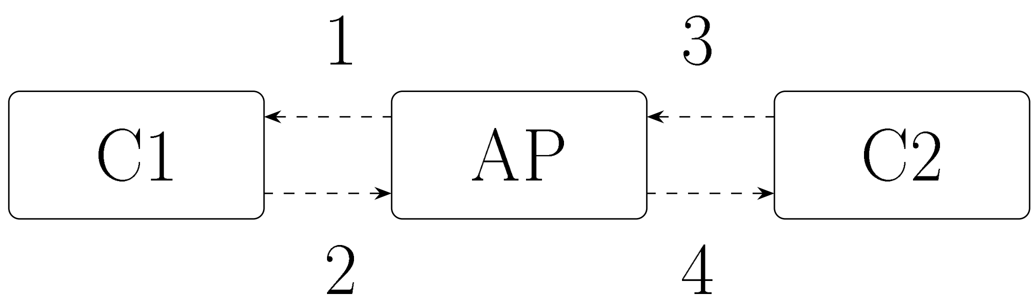
1. Store an asymmetric public key in flash
2. Generate a random number using onboard TRNG
3. Ask new component to sign random number
4. Verify using onboard public key

Meets SR3

4 Proposed Boot Changes

Verify integrity of all 3 boards

- Store public keys C and private key A on AP
- Store public key A and private key C on Component1
- Store public key A and private key C on Component2



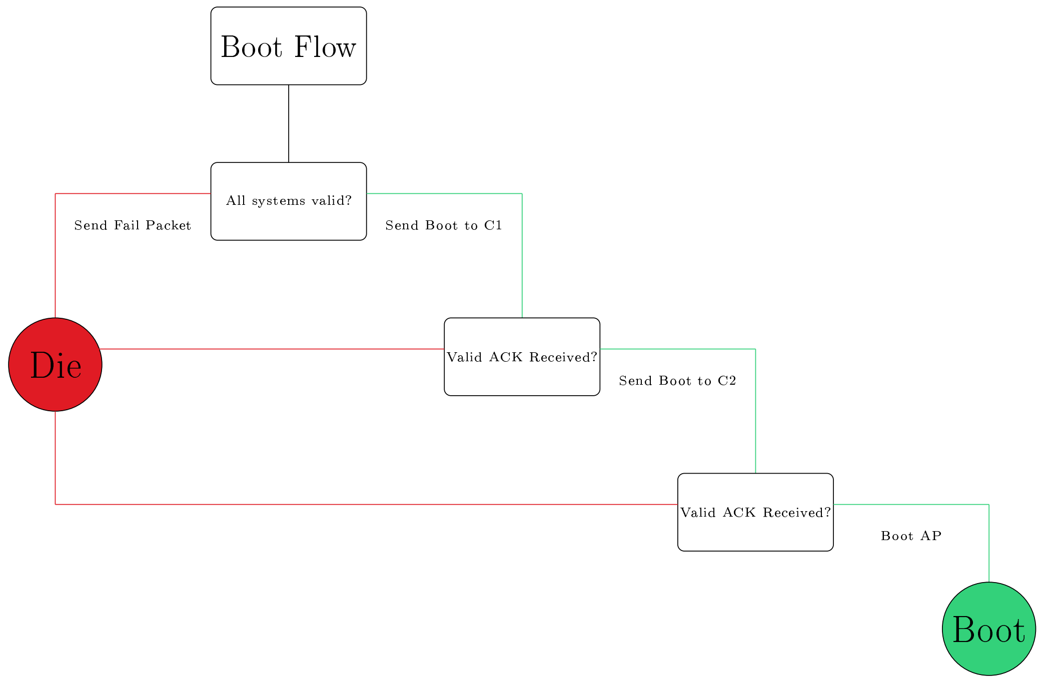
1. AP verifies Component1
   (a) AP generates a random number and asks Component1 to sign with key C
   (b) AP verifies signature using key C
2. Component1 verifies AP
   (a) Component1 generates a random number and asks AP to sign with key C
   (b) Component1 verifies signature using key C
3. Component2 verifies AP
   (a) Component2 generates a random number and asks AP to sign with key A

(b) Component2 verifies signature using key A

4. AP verifies Component2

    (a) AP generates a random number and asks Component2 to sign with key C

    (b) AP verifies signature using key C



If any signatures are invalid, stop immediately and shut down.

Header		Payload		
Packet Magic	Checksum	Length	Data	Signature
(1 byte)	(4 bytes)	(1 byte)	(4 bytes)	(65 bytes)
0xBB		0x04	0x424F4F54	

Table 5: Component Boot Packet

Header		Payload		
Packet Magic	Checksum	Length	Boot Message	Signature
(1 byte)	(4 bytes)	(1 byte)	(64 bytes)	(65 bytes)
0xFB		0x40		

Table 6: Boot ACK Packet

If:

- Packet Magic != 0xBB or 0xAA
- CSUM(Payload) != Expected Checksum
- Length != 0x45
- Data != 0x424F4F54
- recover(signature) != key B or key C
- Startup ACK data == 0xFF and recover(signature) == key A, key B, key C, or key D

Shut down immediately, send fail packet if running on component, and do not continue operation.

Meets SR1 and SR2

5 Proposed Secure TX Changes

ECIES Based Scheme

- Generate private key using RNG
- Create an encrypted channel even though unnecessary.
- Confidentiality will be provided to make RE'ing just a tiny bit harder
- Encrypt packets with negotiated key
- Negotiate HMAC key over new channel
- Append HMAC to all packets before encrypting
- Calculate checksum of encrypted data

Header		Encrypted Payload				
Packet Magic	Checksum	Payload Magic	Length - 1	Nonce	Data	HMAC
(1 byte)	(4 bytes)	(1 byte)	(1 byte)	(4 bytes)	(256 bytes)	(32 bytes)
0xEE		0xDD				

Table 7: Encrypted I2C Packet

Key Exchange				
Packet Magic	Checksum	Length	Key Material	Key Hash
(1 byte)	(4 bytes)	(1 byte)	(64 bytes)	(32 bytes)
0x4B		0x60		

Table 8: Key Exchange I2C Packet

If:

- Packet Magic != Expected Magic
- CSUM(packet) != Expected Checksum
- Payload Magic != Expected Magic
- HMAC(Data) != HMAC or Hash(Key) != Key Hash

- Nonce != Expected Nonce

Shut down immediately, send fail packet, and do not continue operation.

Meets SR5

6 Other

Secure DAPLink firmware for RISC-V chip

- Only execute signed code
- Disable the DAPLINK flashing utility
- Disable code debugging
- Disable MAINTENANCE mode

Secure key storage

- All asymmetric and symmetric keys located on flash will be stored in an encrypted state
- Wrapper keys will be compile-time constants and XOR'ed with another compile-time constant so the raw key will *NEVER* be stored in flash
- By wrapping all keys, a flash dumper payload would not be able to extract the real keys and static reverse engineering would have a similar outcome

All of the above objectives are futile if the attacker can simply modify the flash or just set a breakpoint where the validation happens. By not allowing the chip to be debugged (easily) and only allowing signed code to be run, security becomes a lot more reasonable. After reading through the requirements, some of these "secure boot" steps may be unnecessary, so may or may not be implemented.

7 Summary

7.1 SR1 All components must be valid for AP to boot

- Validate Component1 integrity through signing an arbitrary number
- Validate Component2 integrity in same manner
- Components then validate the AP to make sure all 3 systems are present and valid
- Boot the AP

7.2 SR2 All components must be validated by AP and commanded before booting

- After a successful handshake, it can be assumed that all components are valid
- Send signed boot command to components from AP
- Boot individual components

7.3 SR3 The Attestation PIN and Replacement Token should be kept confidential

- PIN will be stored as a hash with enough iterations to reduce the brute force likelihood
- Replacement Token will also be stored as a hash

7.4 SR4 Component Attestation Data should be kept confidential

- Attestation Data will be stored with symmteric encryption with the key being derived from the Attestation PIN

7.5 SR5 Integrity and Authentication of all communications

- All messages will follow a standard packet format with a negotiated HMAC key and assymetric encryption
- A nonce and ephermeral keys may be included to limit replay attacks