0xDACC

Delaware Area Career Center

eCTF 2021

Design Document

Version 1.1

(MITRE 2021 Collegiate Embedded Capture-The-Flag)

1. Build Process

Our build process supports 3 commands:

1.1 make create_deployment

- Create the deployment by building the SSS (See 2.1 SSS)
- Prepare to add SEDs by creating a base Docker container for Scewl Bus Controllers
- Generate secret data which can later be used to cryptographically guarantee security requirements
    - Generates 256 secret files, one for each depl_id. Each includes:
        - Constants, including a deployment id (depl_id)
        - Authorization to register and deregister
        - ECC public keys, a private key, and a broadcast keypair
            - Locked; must be unlocked during registration
        - Entropy
            - To be modified during registration

```
$ make create_deployment DEPLOYMENT=<DEPL>
```

1.2 make add_sed

- Add an SED to the deployment
- Distribute the appropriate secret keys and data to the SCEWL Bus Controller
    - Assigns a depl_id and its associated secrets file to this SED
- Build the SED CPU and Controller code
- Inform the SSS of the pairing between the depl_id and SCEWL_ID

```
$ make add_sed DEPLOYMENT=<DEPL> SED=<SED> SCEWL_ID=<SID> \
               NAME=<NAME> CUSTOM=<CUSTOM>
```

1.3 make remove_sed

- Permanently remove an SED from the deployment
- Inform the SSS that this SED may no longer be registered or deregistered and that its deployment id shall be mapped to no SCEWL id
- Invalidate the SED's keys so it may no longer communicate on the SCEWL network

```
$ make remove_sed DEPLOYMENT=<DEPL> SCEWL_ID=<SID> NAME=<NAME>
```

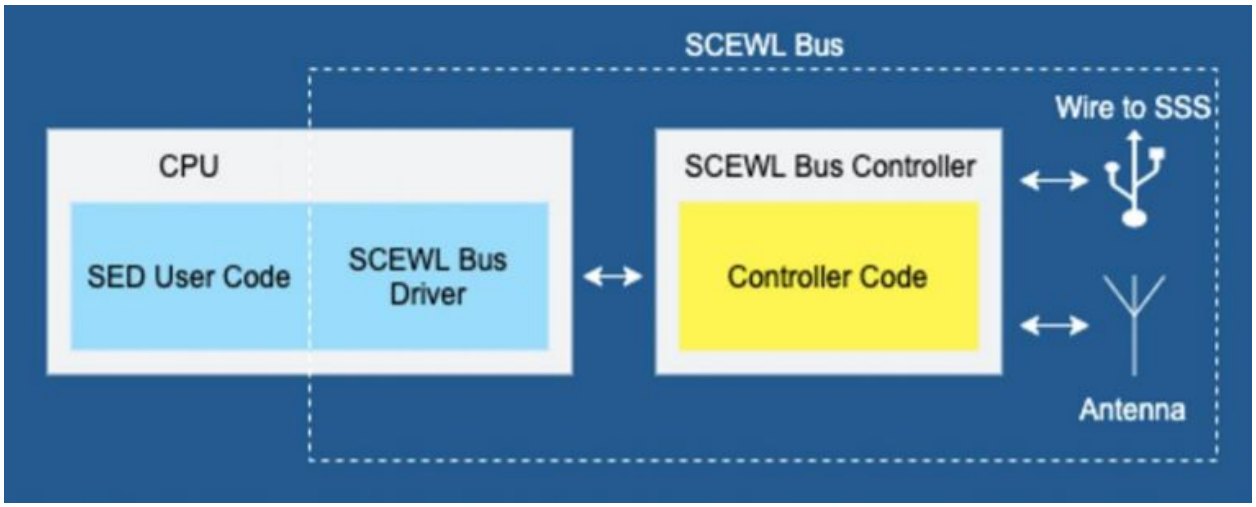
2. Components

Our design implements 2 hardware components:

2.1 SSS (SCEWL Security Server)

- The SCEWL Security Server is in charge of registering and deregistering SEDs
- Launched during `make deploy`, at which time the SSS generates a nonce to identify this launch
- Acts as a certificate authority, storing public keys and providing them upon registration
- Registers and deregisters only properly provisioned devices; those which were created during the build process via the `add_sed` command (see 5.1 Authentication Keys)
- **Registration**
    - Accepts authenticated registration requests, and responds with:
        - A mapping between layer 2 SCEWL_IDs and layer 3 DEPL_IDs
        - A sequence number, equal to the last stored sequence number (see Deregistration below), defaulting to 1
        - A list of last-seen sequence numbers, equal to the last stored list, defaulting to all 0s
        - A key to unlock the ECC keys, enabling communication
        - Random data ensuring entropy is at least 128-bit security level
        - A nonce to prevent injection of packets from previous launches
- **Deregistration**
    - Accepts authenticated deregistration requests, storing the supplied data:
        - Sequence number
        - Last-seen sequence numbers from each SED in the deployment
- Supports up to 16 devices simultaneously registered

2.2 SCEWL Bus Controller



- Our SCEWL Bus Controller provides SEDs with the ability to communicate using the SCEWL network protocol
- The Controller creates and processes SCEWL packets, which add security to the messages contained. These packets are formatted with a 112B header and a ciphertext:

uint8_t sig[64]	uint16_t src	uint16_t tgt	uint16_t ctlen	uint16_t padding	uint64_t seq	uint8_t key[16]	uint8_t iv[16]	uint8_t ct[ctlen]

- It encapsulates (and decapsulates) the SCEWL packets (**body** below) into message frames with a 50-byte message header

char 'S'	char 'C'	uint16_t dst_id	uint16_t src_id	uint16_t body_len	char body[body_len]

    - These frames are transmitted as bytes on an interface
- The Controller handles commands from the SCEWL Bus Driver interface
- This enables each of the communication types specified in **Section 3. Communications**

3. Communications

Our design supports several classes of communication on the SCEWL network.

3.1 Registration and Deregistration

Registration and Deregistration requires interaction with the SSS, and this interaction uses a dedicated wire between the SED and the SSS. No radio communication is needed.

Requests to register and deregister are authenticated to prevent malicious SEDs from attempting either action. See 2.1 SSS for a description of how the SSS handles registration and deregistration requests.

The controller must register before it can engage in network communications. During registration, the controller unlocks its keys, loads an ID lookup table and a list of sequence numbers, scrambles its entropy, and seeds its random generator.

A SED must deregister before powering off in order to store sequence number information and make room for other SEDs to register (limited to 16 simultaneous). After deregistration, the controller hangs until the device is powered down.

3.2 Direct Transmission

Direct transmissions are intended for a specific target SED.
They are encrypted, signed, and authenticated to meet the specifications of ***Section 4. Security Requirements***. See also ***Section 5. Keys*** for a breakdown of the key infrastructure required to support direct transmissions.
Direct Transmissions are sent over the radio interface.

3.3 Broadcasts

Broadcast transmissions are intended for all properly registered SEDs on the network.
They are encrypted, signed, and authenticated to meet the specifications of ***Section 4. Security Requirements***. While broadcast functionality generally mirrors direct transmission, it uses a separate keypair; all registered devices possess the broadcast public and private keys.
Broadcast transmissions are sent over the radio interface.

3.4 FAA

FAA transmissions are sent in plaintext, with no added authentication or assurance of integrity. Although this may present a security risk, it is necessary for compliance with FAA regulation.
FAA transmissions are sent over the radio interface.

4. Security Requirements

Our design fulfills several important security requirements to defend against potential attacks.

4.1 Confidentiality

- SCEWL Transmissions are encrypted to prevent messages from being read by unintended targets
- Because we are using radio waves as our transmission medium, we assume that all signals are intercepted by a Man in the Middle (MitM). However, this MitM will not be able to recover the message contents without the key.

4.2 Integrity

- The integrity of SCEWL Transmissions is always verified through digital signatures, and modified transmissions will be discarded

- Due to the potential for side-channel attacks, we verify the integrity of encrypted messages before attempting decryption
    - This way, maliciously crafted packets targeting a side-channel weakness in decryption will be rejected

4.3 Authentication

- The source of SCEWL Transmissions is always verified through digital signatures, and spoofed transmissions will be discarded
- Valid transmissions are sent by currently-registered (and therefore properly provisioned) devices whose identity matches that which their message claims
- Invalid (spoofed) transmissions are sent by:
    - Attackers directly over the radio, without the use of an SED
    - Devices that are not currently registered
        - (due to deregistration or failure to register)
    - Devices whose identity is not that which their message claims
- Due to the potential for side-channel attacks, we verify the authenticity of encrypted messages before attempting decryption
    - This way, packets from illegitimate sources targeting a side-channel weakness in decryption will be rejected

4.4 Replay Protection

- Each SED's SCEWL Bus Controller maintains a sequence number, which it increments each time it sends a message
- Controllers also record the sequence number of the most recently accepted message from each SED in the deployment
- Received messages whose sequence number does not exceed the most recent sequence number from its source will be rejected, as they must be expired or duplicate.
- Replay protection is provided for up to 2^{64} messages

4.5 Defense in Depth

- Our SCEWL Bus Controller is engineered to resist attacks from the SCEWL Bus Driver interface. In the event that the CPU is compromised, it will be unable to recover the secrets held by the SCEWL Bus Controller

5. Keys

Our design requires several keys to be shared between various parties.

5.1 Authentication Keys

These keys are used to prove that the holding SED is the properly provisioned SED which it identifies itself to be, allowing registration and deregistration.

Holder	Validator	Negotiated	Purpose
SED	SSS	During add_sed; Invalidated during remove_sed	(2.1) Only register properly provisioned SEDs

5.2 Signing Keys

These keys are used to sign messages, ensuring their integrity and authenticity.

Holder	Validator	Negotiated	Purpose
SED A	SED B	During registration, at which time the SSS functions as a Certificate Authority	(4.2) Messages may not be modified in transit
SED	Broadcast	During registration, at which time the SSS functions as a Certificate Authority	(4.2) Messages may not be modified in transit

5.3 Encryption Keys

These keys are used to encrypt messages, ensuring their confidentiality.

Sender	Receiver	Negotiated	Purpose
SED A	SED B	Via shared secret during send and recv	(4.1) Messages may not be read by unintended targets
SED	Broadcast	Via shared secret during send and recv	(4.1) Messages may not be read by unintended targets

6. Cryptographic Implementation

To ensure theoretically secure communications, we use Cryptography to support Confidentiality, Integrity, and Authenticity of messages.

6.1 ECC

We use `secp256r1` as our elliptic curve. Our intention with this curve is to provide about 128 bits of security, which we believe is accomplished by the prime near 2^{256}. This curve is recommended in SEC 2, v2, and is widely implemented. Our design relies upon ECDH to establish shared secrets and upon ECDSA to sign messages.

6.2 AES

We use AES with 128-bit key size in CTR mode to encrypt messages. This mode of operation is not padded and reveals the exact message length, which we consider acceptable. Keys and nonces are randomly generated for each message. Keys are then hidden by applying a simple Key Derivation Function to the shared secret and encrypting the AES key with this value.

6.3 SHA_256

We use SHA_256 as a hash function to provide collision resistance and preimage resistance. Our message digest construction prevents length extension attacks by validating length.

6.4 CSPRNG

We use a Cryptographically Secure Pseudorandom Number Generator to generate randomness on the SCEWL Bus Controller. This is seeded by a secret from the SSS. We use HMAC_DRBG with SHA-256, as defined by NIST Special Publication 800-90A Revision 1. Our Elliptic Curve Cryptography implementation also makes use of the random generator to provide side channel resistance.