DELAWARE AREA CAREER CENTER

Beau Schwab **Project Manager**	Samuel Goodman **Comms. Director**	Andrew Langan **Lead Developer**	Eli Cochran **Team Advisor**
David Nunley **Developer**	Ezequiel Flores **Developer**	Grayson Seger **Developer**	Henry Reid **Developer**
Cameron Crossley **Developer**	Tyler McColeman **Developer**	Ethan Martindale **Video Editor**	Seth Tydings **Video Editor**

0xDACC

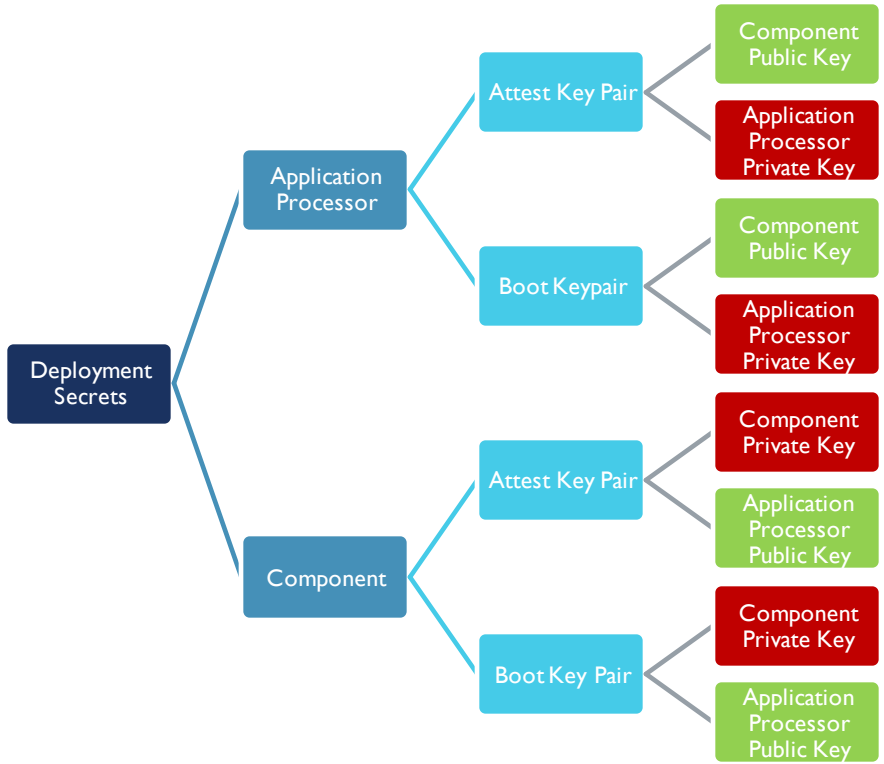
CRYPTOGRAPHIC ALGORITHMS

- AES-128-CTR
- HMAC-SHA256
- SECP256r1 ECC
- SHA256

SECURE SECRET GENERATION

Compile-time Python scripts

- deployment/make_secrets.py
  - Generates shared public/private ECC keys
  - Generates unwrapped attest AES key
  - Generates shared HMAC key
- application_processor/make_secret.py
  - Hashes PIN n times for comparison
  - Hashes PIN n-1 times as wrapper for global attest key
  - Hashes Replacement Token
- component/make_secret.py
  - Encrypts attestation data with unwrapped key



NEW I²C PROTOCOL

- No more registers
- Common packet format
- Packet checksums
- Callbacks instead of polling
- 1:1 Request to Response
- Length encoded along with data

mitre - packets.h

```
1  struct header_t {
2      packet_magic_t magic;
3      uint32_t checksum;
4  };
5
6  template<packet_type_t T> struct __packed payload_t;
7
8  // Encrypted packet payload
9  template<> struct __packed payload_t<packet_type_t::SECURE> {
10     uint8_t magic;
11     uint8_t len;
12     uint32_t nonce;
13     uint8_t data[64];
14     uint8_t __padding[10];  // Pad to multiple of AES block size
15     uint8_t hmac[32];
16 };
17
18 template<packet_type_t T> struct packet_t {
19     header_t header;
20     payload_t<T> payload;
21 };
```

ATTEST

Strengths

- Encryption of data
- Hashed PIN as wrapper
- Signature validation

Weaknesses

- No delays
- Small key space
- Signature reuse

mitre - application_processor.cpp

```cpp
tc_sha256_init(&sha256_ctx);
for (uint32_t i = 0; i < ITERATIONS; ++i) {
    tc_sha256_update(&sha256_ctx, pin, 6);
}
tc_sha256_final(hash, &sha256_ctx);

if (memcmp(hash, ATTEST_HASH, 32) != 0) {
    print_error("Error :(\n");
    return;
}

tc_sha256_init(&sha256_ctx);
for (uint32_t i = 0; i < ITERATIONS - 1; ++i) {
    tc_sha256_update(&sha256_ctx, pin, 6);
}
tc_sha256_final(hash, &sha256_ctx);

unwrap_aes_key(unwrapped_key, ATTEST_KEY_WRAPPED, hash, ATTEST_WRAPPER_NONCE);
```

mitre - application_processor.cpp

```cpp
const char *const fmts[3] = {"LOC", "DATE", "CUST"};

tc_aes128_set_encrypt_key(&aes_key, unwrapped_key);
memcpy(tx_packet.payload.data, "ATTEST", 0x06);

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, tx_packet.payload.data, 0x07);
tc_sha256_final(hash, &sha256_ctx);

if (uECC_sign(ATTEST_A_PRIV, hash, 32, tx_packet.payload.sig,
              uECC_secp256r1()) != 1) {
    return error_t::ERROR;
}

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, rx_packet.payload.data, 64);
tc_sha256_final(hash, &sha256_ctx);

tc_ctr_mode(out, 0x40, rx_packet.payload.data, 0x40, ctr, &aes_key);
print_info("%s>%.64s\n", fmts[i], out);
```

REPLACE

Strengths

- Hashed token

Weaknesses

- Could not get validation to function

mitre - application_processor.cpp

```
1  tc_sha256_init(&sha256_ctx);
2  tc_sha256_update(&sha256_ctx, buf, 16);
3  tc_sha256_final(hash, &sha256_ctx);
4
5  if (memcmp(hash, REPLACEMENT_HASH, 32) == 0) {
6      return error_t::SUCCESS;
7  } else {
8      return error_t::ERROR;
9  }
```

BOOT

Strengths

- Secure key exchange protocol for secure send
- TRNG engine
- Mutual signature validation

mitre - component.cpp

```
// Packet checks and re-construction omitted for brevity.

uECC_make_key(public_key, private_key, uECC_secp256r1());
uECC_shared_secret(rx_packet.payload.material, private_key, shared_secret,
                   uECC_secp256r1());

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, shared_secret, 32);
tc_sha256_final(hash, &sha256_ctx);

memcpy(ctr, "\x00X\xDA\xCC\x00X\xDA\xCC", 8);
memcpy(&ctr[8], &hash[16], 0x8); // AES-128-CTR nonce utilized secure send and receive
memcpy(aes_key, hash, 16);       // AES-128-CTR key
```

Weaknesses

- Only 1 key pair
- Fault injection during RNG

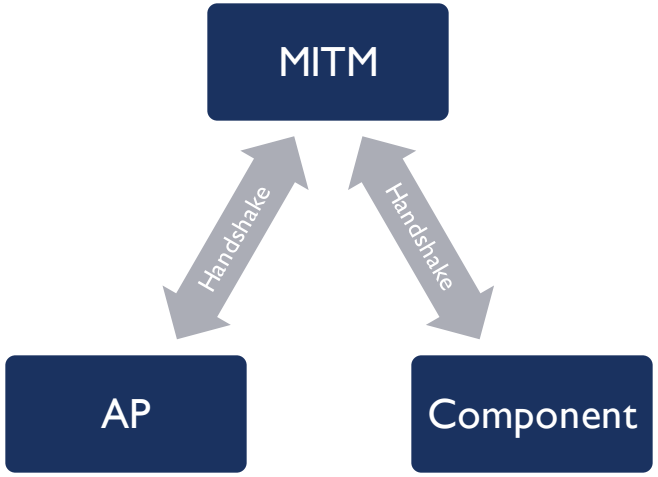
mitre - application_processor.cpp

```
random_bytes(tx_packet.payload.data, 0x20);

if (uECC_sign(BOOT_A_PRIV, tx_packet.payload.data, 0x20,
              tx_packet.payload.sig, uECC_secp256r1()) != 1) {
    return error_t::ERROR;
}

tc_sha256_init(&sha256_ctx);
tc_sha256_update(&sha256_ctx, rx_packet.payload.data, 0x40);
tc_sha256_final(hash, &sha256_ctx);

if (uECC_verify(BOOT_C_PUB, tx_packet.payload.data, 0x20,
                        rx_packet.payload.sig, uECC_secp256r1()) != 1) {
    // Invalid signature
    return error_t::ERROR;
}
```

SECURE SEND & RECEIVE

Strengths

- Ephemeral keys prevent replay across boots ***(Authenticity)***
- Nonces prevent replay during sessions ***(Authenticity)***
- HMAC to prevent modification ***(Integrity)***
- AES encryption to prevent reading ***(Confidentiality)***



Weaknesses

- No MITM protection during KEX
- Leaked HMAC key + MITM = Full compromise

mitre - application_processor.cpp

```cpp
// Send a packet to the specified component
const uint8_t index = addr_to_idx(address);

payload[0] = static_cast<uint8_t>(packet_magic_t::DECRYPTED);
payload[1] = len;
memcpy(&payload[2], &nonces[index], 0x04);
memcpy(&payload[6], buffer, len);

tc_hmac_init(&hmac_ctx);
tc_hmac_set_key(&hmac_ctx, HMAC_KEY, 32);
tc_hmac_update(&hmac_ctx, &payload[0], sizeof(payload) - 32);
tc_hmac_final(hmac, 32, &hmac_ctx);
memcpy(&payload[sizeof(payload) - 32], hmac, 32);

tc_aes128_set_encrypt_key(&aes_key, aes_keys[index]);
tc_ctr_mode(reinterpret_cast<uint8_t *>(&tx_packet.payload),
            sizeof(tx_packet.payload), payload, sizeof(payload),
            ctrs[index], &aes_key);

++nonces[index];
```

NOT SO RANDOM RAND()

Problem

- g_nKey is used to encrypt all communications
- time(NULL) returns -1 without implementation
- srand(time(NULL)) has a deterministic output
- rand() is not a CSPRNG

mitre -

```
1  srand(time(NULL));
2  bzero(g_nKey, BLOCK_SIZE);
3  for(int i = 0; i < KEY_SIZE;i++)
4      g_nKey[i] = rand() % 256;
```

Solution

- Use onboard TRNG hardware
- Remove all rand() based code

mitre -

```
1  bzero(g_nKey, BLOCK_SIZE);
2  MXC_TRNG_Init();
3  MXC_TRNG_Random(g_nKey, KEY_SIZE);
4  MXC_TRNG_Shutdown();
```

BINARY LEAK

- Exploit binary leaked in public channel “uccon_supply_dump.img”
- Written in Rust
- Exploits Mitre-provided I²C Peripheral library
- Unsuccessful RE due to lack of time

FINAL COMMENTS

- Compile-time secrets
- Memory corruption
- Embedded hardware
- Read the documentation

QUESTIONS?

HTTPS://GITHUB.COM/0XDACC/2024-MITRE-ECTF-DACC.GIT

HTTPS://WWW.LINKEDIN.COM/IN/ANDREWLANGAN/

HTTPS://WWW.LINKEDIN.COM/IN/SAM-GOODMAN-CYB/

HTTPS://WWW.LINKEDIN.COM/IN/BEAUSCHWAB26/