Designing with Security in Mind
FTF-INS-N2039

Daniel Azimov
Martin Bernier
Motivation for Security
Hardware Security
Demo Platform
Firmware Implementation
Q&A
Motivation for Security
Risks

• Attacking the device
  – Tampering with the device
  – Counterfeit device

• Attacking the device link
  – Stealing information (Eavesdropping)
  – Modifying information (or Fabrication)

• Attacking the system (Denial of Service)
Security by Design

• Think about attackers
  – How can someone abuse of the system
• Think about “openness”
  – Can some features make our system less secure
Requirements

• Safety
  – Do what you are supposed to do

• Privacy
  – Restrict access to user data

• Access control
  – Restrict access to authorized persons
Key takeaway

- **HW Integrity**
- **Secure Access**
- **Data Encryption**
- **HW Authentication**

Each measure requires secure storage of keys or identification assets.
Hardware Security
Main Security Services

Data protection
- Confidentiality
  - Encryption
- Integrity
  - Hashing

Authentication & Authorization
- Authentication
  - Password
  - Biometry - Token
- Authorization
  - Access rights

Software protection
- Code Integrity
  - Code signature
  - Code verification
- Runtime integrity

Logging & Auditing
- Security log
- Remember actions
- Auditor access
- Log interpretation

Provisioning
- Code Update
  - System upgrade
  - App upgrade
  - Bug fixing
Hardware Security Solution

- Authenticate boot software
- Key storage for encrypted firmware
- Secure Firmware Update
- Node Authentication
  - Use pre-stored cert or hash to authenticate without cloud connection
- Cloud Authentication
  - Use PKI structure for mutual authentication
- Tamper resistant
Hardware Implementation
Secured Access Demo Platform

Multi-factor authentication to support user access

★ PIN using PCA8885 (capacitive touch keyboard)
★ Fingerprint reader FPC1011F3 (Fingerprints)
★ NFC reader CLRC663
Core Security

- 3-Axis Accelerometer FXO8700
- Secure Element A70CM
- MCU with integrated security LPC43S57
LPC43S57 MCU Features

- Dual Core MCU
- 1MB Flash
- 136kB SRAM
- High-speed Connectivity
- Advanced Peripherals
LPC43S57 Security Features

- Unique Device ID
- Secure Boot from encrypted image
- True Random Number Generator
- Hardware-accelerated AES-128 Engine
- Two 128-bit nonvolatile OTP memories for encrypted keys
What is an A70CM?

• An integrated system with enhanced security
  – Anti-cloning
  – Key storage
  – Asymmetric/Symmetric key encryption, decryption and generation
  – Signature generation and verification
  – Authentication based on PKI

• Security OS JCOP 2.4.2 OS – Smart Card Operating System

• Card Manager Applet
  – Configuration of the cipher suites
  – Cryptographic operations
  – Trust Provisioning at different stages
MIFARE on A7005/6 depending on the configuration
A70CM Key Features

• Public Key Infrastructure (PKI) authentication to support TLS session
• RSA/ECC key-pair generation and signature generation/verification
• RSA encryption/decryption
• AES algorithm: AES-128/256
• Total 78 AES keys in the key store.
  – 26 Key sets in the key store. Accessible to users
  – 1 default key-wrapping key. Invisible to user
  – 1 local encryption key. Invisible to users
• Key wrapping
• Two formats of key set
• Secure remote key management
• Trust Provisioning service in NXP certified and secure environment
# A70CM Keys and Certificates

<table>
<thead>
<tr>
<th>Key ID</th>
<th>Object type/purpose</th>
<th>NXP Provisioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device ID1, $K_{pr}/K_{pub}$</td>
<td>(ECC/RSA) public/private key pair for Device Authentication (TLS)</td>
<td>Created by NXP and injected by NXP at Wafer Level</td>
</tr>
<tr>
<td>Device ID2, $K_{pr}/K_{pub}$</td>
<td>(ECC/RSA) public/private key pair for Device Authentication (TLS)</td>
<td>-</td>
</tr>
<tr>
<td>Device certificates $K_{root CA}$</td>
<td>2 certificates for Device Authentication corresponding to Dev ID1 and Dev ID2</td>
<td>Optional: creation and injection by NXP</td>
</tr>
<tr>
<td>$K_{AES,1}$</td>
<td>2 Public key (ECC/RSA) Server/client certificate checking</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$K_{AES,24}$</td>
<td>AES Key store: default AES 128 bits key set (triplet) (<em>), AES Key store: default AES 256 bits key set (triplet) (</em>)</td>
<td>Initialized by NXP to Random</td>
</tr>
<tr>
<td>SM $K_{ADMIN}$</td>
<td>AES Key store: AES key Set 1 (triplet) (*)</td>
<td>Initialized by NXP to Random</td>
</tr>
<tr>
<td>SM $K_{WRAP}$</td>
<td>...</td>
<td>Initialized by NXP to Random</td>
</tr>
<tr>
<td>SM $K_{MK}$</td>
<td>AES Key store: AES key Set 24 (triplet) (*)</td>
<td>Initialized by NXP to Random</td>
</tr>
<tr>
<td></td>
<td>Public key (ECC/RSA) Remote key/certificate mgnt (access control)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>AES-128 key Encrypt keys exchanged on SM IF</td>
<td>Initialized by NXP to Random</td>
</tr>
<tr>
<td></td>
<td>AES Key for Secure Module Upgrade (Card Manager Key)</td>
<td>Unique by Secure Element, Available through NXP Key Delivery Service (KDS).</td>
</tr>
</tbody>
</table>

Note: Device = OEM product

(*) Eg DLMS keys ($K_{MK}, K_v, K_Eu$) or Mbus keys ($K_M, K_C$ - )
A70CM Security in Hardware

CPU with Glue Logic
+ Memory Scrambling
+ Active Shield

Active Shields

Memory: holding secret data

(Co-)Processor, Logic: operating on secret data
A70CM: System Implementation

- A70CM Applet
- A7001 HW
- Application
- A70CM Host API
- SCI2C
- Board dependant code /HAL
- Host CPU Hardware platform

I2C
LPC43S & A70CM Firmware

Presenter: Daniel Azimov
Agenda

• A70CM Secure Module (SM) Life Cycle
• Manufacturing Mode Firmware
• Normal Mode Firmware
• Tips & Tricks
Configure the A7 to use ECC:
- **Elliptic Curve Cryptography** - Public Key encryption mechanism

Or, you may choose to configure RSA encryption

**Benefit of ECC:** smaller key size compared to RSA, reducing storage and transmission size

```c
/*---------- Beginning of irreversible steps if using production version of A7 chip ----------*/

// Following two steps are irreversible - once performed, it permanently changes the A70CM and cannot be reverted.
// Configure A7 for ECCurve NIST P192

result = configure_ecc(ECCurve_NIST_P192);

// Switch A7 to CONFIGURE state
result &= switch_to_configure(SST_ECC_KEYPAIR_ECCurve_NIST_P192);
```
Ask A7 to generate and store Asymmetric Public and Private key-pair

The Public & Private key-pair will be used for digital signing and authentication

```c
if (A70UserAction == action_generate_Keypair) {
    //****Generate key pair in A7 */
    result = generate_key_pair(SM_CONFIGURE, ECCCurve_NIST_P192, action_generate_Keypair);
}

if (A70UserAction == action_store_Keypair_A7) {
    //****Store Public and Private key pair to A7 at index 0 */
    result = generate_key_pair(SM_CONFIGURE, ECCCurve_NIST_P192, action_store_Keypair_A7);
}
```
OEM – Life Cycle: Configure Step 2

OEM - Life Cycle:
• Configure Init
• Configure

Ask A7 to generate/set and store Symmetric AES-128 bit Key

The AES Key will be used for data encryption and decryption in normal (field) mode

```c
if (A70UserAction == action_generate_Random_AES_Key) {
    //Generate random AES key
    generate_random_aes_key();
}
else if (A70UserAction == action_set_Fixed_AES_Key) {
    //Set fixed AES KEY
    set_fixed_aes_key();
}
```
LPC creates hash of the Device ID & A7 signs hash with its Private Key
LPC Hashes the Device ID & A7 Signs Hash with its Private Key

The Signed Hash of Unique ID is stored in LPC flash and will be used to prevent hardware cloning

```c
if (A7UserAction == action_calculate_SignedHash) {
    /* Calculate the signed hash */
    result = calculate_SH_init();
}

if (A7UserAction == action_store_SignedHash) {
    /* Store the signed hash */
    put_SH_in_flash_init();
}
```
A70CM Life Cycle Recap

NXP

OEM

Field

RSA 1024
RSA 2048
ECC 192
ECC 224
ECC 256

RSA 1024
RSA 2048
ECC 192
ECC 224
ECC 256

Reconfigure (crypto)

Operate

Success

Assessment

Dead

- Access control enabled for PKI crypto/key management
- Key wrapping enabled

Pre-Personalization

Personalization phase

Trust Provisioning

Inject public & private keys in secure environment

Configure Init

Selects crypto (ECC/RSA, key length, curve)

Sets other keys/cert without access control

Configure

Operate

Attack Detection

Reconfigure (crypto)

www.FutureElectronics.com
After LPC43S boots ask A7 to validate Signed Hash of Unique ID

Your product is in the field ... Verify that the MCU & A7 are a genuine pair

// In normal mode, the valid signed hash value needs to be at a known location.
if (A70UserAction == action_validate_SignedHash) {

    get_signed_hash();
    DEBUGOUT("\n\rRetrieved Signed hash from MCU flash\r\n");

    /* Ask A70CM to validate the signature for authentication */
    result = validate_signedhash();
}
OEM – Life Cycle: Operate Step 2

Normal Mode

Ask A7 to validate the Signed Hash

Retrieve AES Key from A7*

*AES Key is wrapped in Key-Wrapping Key

```
if (A70UserAction == action_get_AES_Key) {
    // Get the AES key from A7 using the Key Wrapping Key
    result = get_aes_key();
}
```
Use LPC43S AES Engine to Encrypt data with the AES key retrieved from the A7

LPC AES Engine supports:
- Electronic Code Book (ECB) 128-bit
- Cipher Block Chaining (CBC) 128-bit

```c
if (A7UserAction == action_Encrypt_Data) {
    // Initialize Encryption Control parameters
    CRYPT_CTRL_T Encrypt_ctrl;
    Encrypt_ctrl.encryption = MODE_ECB; // set encryption mode
    Encrypt_ctrl.key_src = KEY_SW; // set encryption key source
    Encrypt_ctrl.dataOutAddr = CypherText; // set pointer to output data
    Encrypt_ctrl.dataInAddr = PlainText; // set pointer to input data
    Encrypt_ctrl.sizeInBlocks = 1; // encrypt 1 data block of 16 bytes

    /* Encrypt the plain text image using AES ECB */
    result = encryption_dma_image(&Encrypt_ctrl);
}
```
OEM – Life Cycle: Operate Step 4

Use LPC43S AES Engine to Decrypt data with the AES key retrieved from the A7

LPC AES Engine Key Selection:
- Key1 – user defined for secure boot (in OTP)
- Key2 – user defined stored (in OTP)
- Software defined key

```c
if (A7UserAction == action_Decrypt_Data) {
    //Initialize Decryption Control parameters
    CRYPT_CTRL_T Decrypt_ctrl;

    //Decrypt the encrypted CyberText
    Decrypt_ctrl.encryption = MODE_ECB;
    Decrypt_ctrl.decryption = MODE_ECB;
    Decrypt_ctrl.key_src = KEY_SW;
    Decrypt_ctrl.dataOutAddr = Decrypt_CypherText;
    Decrypt_ctrl.dataInAddr = CypherText;
    Decrypt_ctrl.sizeInBlocks = 1;

    /* Decrypt the image using AES ECB */
    result = decryption_dma_image(&Decrypt_ctrl);
}
```
A70CM Debug Version

A70CM Configuration (ECC/RSA) is Irreversible

Alternative for Developers:
• Develop Code with A70CM Debug Version to avoid irreversible configuration
• Reconfigure RSA or ECC as often as you want
• No restriction on number of times you execute SWITCH_Configure
Recall: I2C bus between LPC43S & A70CM is **NOT** secure

**Transmit Keys Securely**

How to send keys securely on I2C?

**Solution:**

• Use **Key-Wrapping Key** to encrypt keys before transmitting!
Key-Wrapping Key

How to Use Key-Wrapping Key?

- Save Symmetric Key-Wrapping Key to:
  - A70CM
  - OTP key in LPC43S

- LPC43S requests AES key from A7
- A7 Key-Wraps AES key & sends to LPC43S

- LPC43S decrypts AES key with Key-Wrapping Key in OTP
- AES engine uses decrypted AES key to encrypt/decrypt
Private Key Storage

Where to Store Private Keys on MCU?

• SRAM – Bad Idea …

• Non-Volatile Memory – Even Worse!

• There is NO good place to store the private key in the MCU!

• So, the answer is …

  The private key MUST REMAIN in the A70CM
  … NEVER store your private keys in the MCU!!
FREE HANDS-ON WORKSHOP:
One day free hands-on workshop focused on NXP’s secure MCU product line through Security Access System reference designs.

FREE HANDS-ON WORKSHOP:
One day free hands-on workshop introducing attendees to three key technologies for the internet of things: NFC, Wi-Fi and Bluetooth Smart.
Thank you !!