Future Electronics – NXP
Secure Access Lab
1. Improve your understanding of security/crypto concepts and keywords – Public/Private keys, AES, ECC, etc...

2. Demonstrate the security/crypto concepts using LPC43S57 MCU and the A70CM Secure Element.

3. Demonstrate the NTAG usage
Future Electronics and NXP has collaborated to build this Secure Access Demo Kit. This kit can secure an access via:

1. PIN
2. Finger Print reader
3. NTAG
The Heart of the Kit

The heart of this kit is the:

1. **MCU – LPC4300 Series**
2. **Secure Element Co-processor – A7001**
Two Modes of Operation – How it actually works ...

The kit has 2 modes of operation -

- **Manufacturing Mode** – executed in a secured facility
  1. Generate & store device-specific public & private key pair in A7
  2. Ask A7 to generate device-specific signature & store it in MCU flash
  3. Save keys required to establish a secure connection with A7

- **Normal Mode** – executed in the field
  1. Ask A7 to verify the signature of the Unique ID
  2. Retrieve an AES key from A7 using the encrypted interface
  3. Use the retrieved key to encrypt a block of data in RAM

Example of what it can do in the field
Two Modes of Operation – How it actually works ...

The kit has 2 modes of operation -

- **Manufacturing Mode** – executed in a secured facility
  1. Generate & store device-specific public & private key pair in A7
  2. Ask A7 to generate device-specific signature & store it in MCU flash
  3. Save keys required to establish a secure connection with A7

First set it up: This is in the factory, where keys are generated
The kit has 2 modes of operation -

- **Normal Mode** – executed in the field
  1. Ask A7 to verify the signature of the Unique ID
  2. Retrieve an AES key from A7 using the encrypted interface
  3. Use the retrieved key to encrypt a block of data in RAM
  4. Use the retrieved key to decrypt a block of data in RAM

* AES Key is wrapped in key_wrapping_key
Recap of the Two Modes

Manufacturing Mode

- Configure A7 for ECC
- A7 Generate & Store Public & Private Key pair in A7
- A7 Generate & Store AES Key in A7
- Use A7 to generate the Signed Hash of Unique ID & Store it in MCU Flash

Done

Normal Mode

- Ask A7 to validate the Signed Hash
- Retrieve AES key from A7
- Encrypt Data in MCU RAM with AES Engine
- Decrypt Data in MCU RAM with AES Engine

Done
Setting Up Hardware

Please assemble hardware to look like this:

USB to your laptop
Connecting FTDI Cable

Connector is not goof-proof, please ensure correct orientation

Black wire on top
Setting Up PUTTY

1. Ensure Secure Access Panel is plugged into your laptop USB port
2. Install CDM21214_Setup.exe from the Lab USB Stick
3. Open Device Manager to obtain COM #.
4. Run Putty.exe from the Lab USB Stick
5. In the PuTTY Configuration, select Serial, change Serial Line and Speed.
Lab 1: The Crypto Game

Learn how to utilize Public Key Cryptography capabilities of A7 to verify device signature and retrieve an AES key securely from A7 secure storage

Your goal is to obtain 100 points by completing 10 tasks.

Manufacturing Mode: 60 points

Normal Mode: 40 points
Manufacturing Mode – 60 points

1. Configure A7 for ECC
2. A7 Generate & Store Public & Private Key pair in A7
3. A7 Generate & Store AES Key in A7
4. Use A7 to generate the Signed Hash of Unique ID & Store it in MCU Flash
5. Done
Next: Run through the Tasks, And review Concepts and Code

Go ahead, select “2” to Run Labs:

Go ahead, select “1” to run Lab #1:
Configure A7 for ECC

In Task #1, you had set the A7 to use ECC, which stands for Elliptic Curve Cryptography. It is an approach to public key crypto.

The benefit of ECC is a smaller key size, reducing storage and transmission requirement.

This is an irreversible step for the A7 at the factory.
## Putty Terminal for User Feedback

### SCOREBOARD

<table>
<thead>
<tr>
<th>Task#</th>
<th>Mode</th>
<th>Status</th>
<th>Description</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Irreversible configure ECC NIST 192</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

---

**Total Points:** 60

### Hints:
- Manufacture Mode tasks complete!
- Task# 1 RESETS the game by setting all points to ZERO!
In Task #2, you ask the A7 to generate the Public and Private Key pair. These keys are used for signature generation/verification.
In Task #4, you generate a random AES Key (or set a fixed AES key) – this is later used to encrypt some data in normal(field) mode.
Second Quiz:

1. Is AES Symmetric or Asymmetric?
2. Why don’t we use Public/Private keys in our Home WiFi network?

<table>
<thead>
<tr>
<th></th>
<th>Secret Key (Symmetric)</th>
<th>Public Key (Asymmetric)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Key</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Protection of Key</td>
<td>Must be kept secret</td>
<td>One key must be kept secret &amp; other can be freely exposed</td>
</tr>
<tr>
<td>Best Uses</td>
<td>secrecy and integrity of data</td>
<td>Key exchange, authentication</td>
</tr>
<tr>
<td>Key Distribution</td>
<td>Problematic</td>
<td>Safer</td>
</tr>
<tr>
<td>Speed</td>
<td>Fast</td>
<td>Slow; typically, 10,000 times slower than secret key</td>
</tr>
</tbody>
</table>
Second Quiz: Answers

1. AES is **Symmetric**

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Secret Key (Symmetric)</th>
<th>Public Key (Asymmetric)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Key</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Protection of Key</td>
<td>Must be kept secret</td>
<td>One key must be kept secret &amp; other can be freely exposed</td>
</tr>
<tr>
<td>Best Uses</td>
<td>secrecy and integrity of data</td>
<td>Key exchange, authentication</td>
</tr>
<tr>
<td>Key Distribution</td>
<td>Problematic</td>
<td>Safer</td>
</tr>
<tr>
<td>Speed</td>
<td>Fast</td>
<td>Slow; typically, 10,000 times slower than secret key</td>
</tr>
</tbody>
</table>

2. Public/Private Key is **Asymmetric** and slow
In Task #5, you created a Signed Hash which is stored in the MCU. Later, the A7 will validate that Sign Hash to ensure that it is a genuine MCU. Task #6, store it in MCU Flash.
## Goal Summary – 60 points

### SCOREBOARD

<table>
<thead>
<tr>
<th>Task#</th>
<th>Mode</th>
<th>Status</th>
<th>Description</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Irreversible configure ECC NIST 192</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Generate random Public, Private keys in A7</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Store random Public, Private keys to A7</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Generate random AES 128-bit key store to A7</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Calculate Device Signature (Signed Hash)</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Store Device Signature (Hash) to LPC Flash</td>
<td>10</td>
</tr>
</tbody>
</table>

Total Points: 60

### Hints:
- Manufacture Mode tasks complete!
- Task# 1 RESETS the game by setting all points to ZERO!
Normal Mode – 40 points

- Normal Mode
- Ask A7 to validate the Signed Hash
- Retrieve AES Key from A7
- Encrypt Data in MCU RAM with AES Engine
- Decrypt Data in MCU RAM with AES Engine
- Done
Validate Signed Hash

```c
if (A70UserAction == action_validate_SignedHash) {
    get_signed_hash();
    DEBUGOUT("\n\Retrieved Signed hash from flash\r\n");

    /* Ask A70CM to validate the signature for authentication */
    result = validate_signedhash();
}
```

Now the product is in the field, you need to make sure that the genuine MCU it talking to A7.

Task #7, asks the A7 to validate the Signed Hash.
In order to show what the chip can do, you asked it to retrieve the AES key which was generated in Manufacturing Mode earlier.
Encrypt/Decrypt Data in RAM

```c
if (A70UserAction == action_Decrypt_Data) {

    //clear data out
    memset(Decrypt_CypherText, 0, CYPHER_CT);

    //Initialize Decryption Control parameters
    CRYPT_CTRL_T Decrypt_ctrl;

    //Decrypt the encrypted CypherText
    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 */
    decryption_dma_image(&Decrypt_ctrl);
}
```

In Task #9 and #10, you asked the MCU to encrypt/decrypt data in RAM using the AES key.
## Goal Summary – 40 points

<table>
<thead>
<tr>
<th>Task#</th>
<th>Mode</th>
<th>Status</th>
<th>Description</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Irreversible configure ECC NIST 192</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Generate random Public, Private keys in A7</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Store Public, Private keys to LPC flash</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Generate random AES 128-bit key store to A7</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Calculate Device Signature (Hash)</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>MANUFACTURE</td>
<td>PASS</td>
<td>Store Device Signature (Hash) to LPC Flash</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>NORMAL</td>
<td>PASS</td>
<td>Validate Device Signature (Hash)</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>NORMAL</td>
<td>PASS</td>
<td>Get AES 128-bit key from A7</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>NORMAL</td>
<td>PASS</td>
<td>Encrypt data with LPC AES engine</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>NORMAL</td>
<td>PASS</td>
<td>Decrypt data with LPC AES engine</td>
<td>10</td>
</tr>
</tbody>
</table>

Total Points: 100
Quick Summary

1. Why did we generate Public and Private Keys?
2. Why did we generate the Signed Hash?
3. Why did we ask A7 to validate Signed Hash?
4. Why did we store the AES Key in A7 (instead in LPC Flash)?
5. What is ECC?
Quick Answers

1. Why did we generate Public and Private Keys?
   - Signature generation/verification

2. Why did we generate the Signed Hash? –
   - This is used later by A7 to validate that a clone MCU is not used

3. Why did we ask A7 to validate Signed Hash?
   - This is used to validate that a clone MCU is not used

4. Why did we store the AES Key in A7 (instead in LPC Flash)?
   - The A7 is tamper-proof.

5. What is ECC?
   - Elliptic Curve Cryptography
Lab 2: The NTAG Demo

In Lab 2 – We demonstrate/discuss:
1. Programming NTAG with PINs
2. The Source Code
3. NFC Data Exchange Format (NDEF)

Go ahead, first “EXIT” Lab 1, then “RUN” Lab 2:

*Notice: Lab2 performs all necessary A7 initializations automatically on wakeup
MISSION 1: Store Your Passport in the Safe in 3 mins

You are on vacation. You are in your hotel and wish to store your passport in the safe.

You put your passport in the safe. And close the door.
In order to lock door:

Go ahead Set YOUR 4 digit PIN by selecting option 1 - SET/CHANGE PIN. Your PIN will be automatically saved to YOUR NTAG KEY!
MISSION 2: Your Spouse Wants to Store Her/His Valuables in the Safe in 3 mins

Now your spouse wants to store her/his Diamond ring/Rolex watch in the safe.

Fortunately, the hotel provided a second NTAG. Your spouse will now program her/his own PIN.

Go ahead, set a different 4-digit PIN for your spouse’s key. It will be automatically saved to her/his NTAG KEY!
Matching NTAG-PINs

To demonstrate that only the matching NTAG and PIN will unlock the safe,

Go ahead try to open the safe –

1. Use your NTAG with your spouse’s PIN or

2. Use your spouse’s NTAG with your PIN

The lock should not have opened ....

Because

The MCU reads the PIN which is stored in the NTAG, then compares it to what was typed. Only the matching NTAG and PIN will unlock the safe.
MISSION 3: IDENTICAL PIN - 2016

Now everyone, set your safe with the PIN 2016.

Your mother-in-law which is the other room lost her NTAG but she said that her PIN was 2016 – the same PIN as yours. What a coincidence!

Being the helpful in-law, you try to open her safe to get her passport.

Go ahead – try to unlock her safe with your NTAG
( at this point, go to another demo kit and try to unlock their demo)
Unfortunately, you couldn’t open it - why?
Even though you have the same PIN - 2016, Your PIN was encrypted with a different AES key. Therefore your NTAG-PIN will not work on a different safe.
Code and Concept Review
Before you start running any crypto code, you must perform initialization. In particular for this project, you need to initialize the A7 and AES Engine:

```c
// Init A700x
init_a700x();

// Init AES block to enable AES engine for encryption, decryption
Chip_AES_Init();

// Get the AES key from A7
status = get_aes_key();
```
The NXP AES Engine supports:

- Electronic Code Book (ECB) decode mode with 128-bit key.
- Cypher Block Chaining (CBC) decode mode with 128-bit key.
- CMAC hash calculation (see Section 7.3.4) for the boot image only.

In this lab, you used the ECB mode:

```c
//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 the plain text image using AES ECB */
encryption_dma_image(&Encrypt_ctrl);
```
Configure the DMA channel to process the AES block,
Then enable DMA & start the AES Operation

```c
ctrl->error = Chip_AES_Config_DMA(dma.channel_id);
if (ctrl->error != LPC_OK) return;
/* Configure DMA channel to process AES block */
/* Check for errors */
ctrl->error = Chip_AES_Operate_DMA(dma.channel_id, ctrl->dataOutAddr, ctrl->dataInAddr, ctrl->sizeInBlocks);
if (ctrl->error != LPC_OK) return;
/* Enable DMA, and start AES operation */
while ((Chip_AES_GetStatus_DMA(dma.channel_id)) != 0) {}  
/* Check for errors */
/* Wait for DMA to complete */
/* Set status */
ctrl->status = true;
```

The AES engine performs DMA to achieve efficient real-time encryption.
Write PIN to NTAG

This is the code used to write the PIN to the NTAG.

```c
//Write encrypted 16-byte block to NTAG (first 4-bytes contain PIN, last 12-bytes are spare)
status = NTAGWriteRead(MIFARE_ULTRALIGHT, NTAG_WRITE, EncryptDataArray);
```

The data is packed in format called NDEF

The purpose of writing NDEF messages onto a Tag is to provide some information that every NFC device can read. That information can be a URL or some text.
5.2 NDEF

The NDEF specification (see NDEF: Ref. 5) defines a message encapsulation format to exchange information between an NFC Device and another NFC Device or an NFC Tag.

NDEF is a lightweight, binary message format that can be used to encapsulate one or more application-defined payloads of arbitrary type and size into a single construct called NDEF message (see Figure 2). An application-defined payload is encapsulated within one single NDEF record, or chunked into two or more NDEF records. One or more application-defined payload contains the application data.

Each application-defined payload is described by Type, and an Optional Identifier:

- Type Identifiers may be URIs, MIME media types, or NFC-specific types (see NDEF: Ref. 5 and NFC RTD: Ref. 8).
- Optional Identifier enables association of multiple payloads and cross referencing between them.

Pls refer to source code file: NTAG.c (line 496)
Some NTAG Features & Benefits

NTAG203
NFC Forum Type 2 Tag compliant IC with 144 bytes user memory
Rev. 3.2 — 12 December 2011

2. Features and benefits

2.1 RF Interface (ISO/IEC 14443A)
- Contactless transmission of data and supply energy (no battery needed)
- Operating distance: up to 100 mm (depending on field strength and antenna geometry)
- Operating frequency: 13.56 MHz
- Fast data transfer: 106 kbit/s
- High data integrity: 16-bit CRC, parity, bit coding, bit counting
- True anticollision
- 7 byte serial number (cascade level 2 according to ISO/IEC 14443-3)

2.2 EEPROM
- 168 bytes of total memory, divided in 42 pages (4 bytes each)
- 144 bytes of user r/w memory area, divided in 36 pages (4 bytes each)
- Field programmable read-only locking function per page for first 64 bytes
- Field programmable read-only locking function per block
- 32-bit user definable One-Time Programmable (OTP) area
- 16-bit counter
- Data retention of 5 years
- Write endurance 10000 cycles
In the USB Stick – please find:

1. LPCXpresso – go on-line to activate Free Version 256k
2. Putty
3. FTDI Driver - CDM21214_Setup.exe
4. A copy of this PowerPoint presentation
THANK YOU