ADVANCE YOUR IoT SECURITY LEVERAGING HARDWARE PROTECTED KEYS

DONNIE GARCIA
NXP IoT SECURITY SOLUTIONS
JUNE 2019
**Hardware Protected Keys Webinar Series**

This webinar meets 3 times.

Tue, Apr 16, 2019 10:00 AM - 11:00 AM CDT
Tue, May 21, 2019 10:00 AM - 11:00 AM CDT
Tue, Jun 18, 2019 10:00 AM - 11:00 AM CDT

**Part 1: Utilizing hardware protected keys on broad market Microcontrollers**

Recording

For the IoT Edge device, the cryptographic keys used to perform the services such as encrypted boot, onboarding, and over the air updates are critical components that must be protected. Chip level hardware protected keys are the standard for achieving strong security protection for embedded designs. This session will define what a hardware protected key is and show several examples of how these keys are realized on NXP processors. The LMX RT 1050 family of devices will be used as a real world example of how Intrinsic ID Broadkey® SRAM based PUF can advance your IoT Security.

**Part 2: Using hardware protected keys on state of the art Microcontrollers**

Recording

For the latest microcontrollers addressing IoT applications, hardware protected keys address critical security functions to protect application integrity, software confidentiality and encrypt data at rest. This session will explore the ability of the recently launched NXP IoT microcontroller, LPC5500 series. This family of devices will work as the main processing unit for a broad range of IoT applications and integrates breakthrough capabilities with regards to security. Along with Arm TrustZone technology the SRAM PUF based key management makes security easy to use and easy to deploy.

**Part 3: Advanced IoT application key management based on hardware protected keys**

The recently launched NXP IoT microcontroller, LPC5500 series, works as the main processing unit for a broad range of IoT applications. Along with Arm TrustZone® technology the chip supports SRAM PUF based key management. The product includes a software development kit (MCUXpresso SDK) that contains prebuilt applications to demonstrate edge to cloud connections out of the box. With the integrated security technology and software enablement, the LPC5500 makes security easy to use and easy to deploy. Join this session for a quick run through the demo applications available to kickstart your next IoT designs.
Agenda

• Quick recap and highlights
• Example IoT Device and Enablement
• Key Use Exploration
  – Secure boot
  – Software IP protection
  – Encrypted Execution
  – Device Data Confidentiality
  – Secure Connections
  – Cloud based OTA
  – Authenticated Debug
• Key Management Table Summary
• Conclusions
QUICK RECAP & HIGHLIGHTS
System Level Security Goals Depend on Cryptography

- Cryptography is a fundamental capability needed to address edge device security
  - Basis for protecting data at rest and in transit
  - Provides robust identity for the end device by cryptographic authentication
- The key material used for cryptographic operations must be protected by hardware
  - Attacks against Confidentiality/Integrity/Authenticity are aimed at attaining the Cryptographic Key
Protected over the lifecycle* of the Cryptographic keys

- Key Lifecycle
  - Generation
    - Who/what creates the key material
  - Establishment
    - How the key material is shared or signed between entities
  - Storage
    - Where the key material is placed for future access
  - Use
    - How the key is utilized during the cryptographic processing
  - Decommission
    - Revocation and destruction of key material

*Key Lifecycle https://community.nxp.com/docs/DOC-333095
SRAM PUF Overview

Leverages the intrinsic entropy of the silicon manufacturing process

Device unique, unclonable fingerprint derived on every activation of the PUF

PUF master key is used to protect other secrets

1. Process Variation
   Naturally occurring variations in the attributes of transistors when chips are fabricated (length, width, thickness)

2. SRAM Start-up Values
   Each time an SRAM block powers on the cells come up as either a 1 or a 0

3. Silicon Fingerprint
   The start-up values create a random and repeatable pattern that is unique to each chip

4. SRAM PUF Key
   The silicon fingerprint is turned into a secret key that builds the foundation of a security subsystem
Exploring Protected Key Options

1. **External Security IC**
   - **NXP IoT Security ICs:** A71CH, A100x Authenticator, SE050
   - **Strongest protection across all key life stages**
   - **Uses:**
     - Device identity and establishing TLS/onboarding
     - NXP Trust provisioning reduces overhead for key generation and establishment

2. **Security Hardening on MCU/MPU**
   - **Provides runtime application security**
   - **Uses:**
     - Secure boot
     - Bulk data protection
     - Enforces security policies (Roles)
     - Firmware updates

3. **Software SRAM PUF**
   - **Uses Incremental**
   - **Security w/SRAM PUF**
     - **Uses:**
     - Key Generation and establishment
     - Device identity
     - Assist with TLS/onboarding

4. **Security w/OTP Keys**
   - **Uses:**
     - Assist with early key life stages and improves protection for keys
   - **Hardware PUF (Intrinsic ID QuidiKey): LPC5500 Family**
     - **Links advantages of PUF to runtime application security**
     - **Uses:**
     - PUF protected keys used for secure boot, etc.
     - PUF for Key generation and establishment protects early life stages
NXP Security Technology

1. Secure Boot
2. Secure Storage
3. Key Protection
4. Key Revocation
5. Secure Debug
6. Tamper Detection
7. Virtualization/HW Firewalls
8. Manufacturing Protection

Documentation & Certifications
- Security Users Manual
- App. Notes
- Community

Software & Tools
- MCUXpresso
- Code Signing Tools
- Manufacturing Tools
- Serial Download Tools

Hardware
- Casper, Prince
- MPC, PPC
- PUF
**NXP LPC5500 MCU SERIES**

**SECURITY SUBSYSTEM OVERVIEW**

- **ROM supporting**
  - Secure Boot, Debug Authentication & DICE Engine

- **TrustZone for Cortex-M33**
  - Arm’s Security Attribution Unit (SAU)
  - Arm’s Memory Protection Unit (MPU): Secure & Non-Secure
  - NXP’s (implementation) Defined Attribution Unit (using IDAU interface)
  - NXP’s Secure Bus, Secure GPIO & Secure DMA Controllers

- **Cryptography Accelerators**
  - Symmetric (AES-256) & Hashing (SHA2) engine
  - On-the-fly flash encryption/decryption engine (PRINCE)
  - Asymmetric engine for RSA and ECC (CASPER)
  - Random Number Generator (RNG)

- **Secure Storage**
  - Physically Unclonable Function (PUF)
    - Device unique root key (256 bit strength), 64-4096 bit key size
  - Protected Flash Region
    - RFC4122 compliant 128-bit UUID per device
    - Customer Manufacturing Programable Area (Boot Configuration, RoT key table hash, Debug configuration, Prince configuration)
      - PUF Key Store (Activation code, Prince region key codes, FW update key encryption key, Unique Device Secret)
    - Customer Field Programable Area (Monotonic counter, Prince IV codes)
LPC5500 EXAMPLE IOT DEVICE
Complimentary with Extensive Support

- MCUXpresso SDK
- MCUXpresso IDE
- MCUXpresso Config Tools

Hardware Platform for Ease of Development

- On-board debug circuit
- PCB Layout, Schematic and Board Files Available

Simplify secure embedded development; Reduce time to market.

LPC5500 MCU Series
Amazon FreeRTOS at the Device

The FreeRTOS kernel is now an AWS open source project, and these pages are being updated accordingly. AWS are pleased to announce immediate availability of the MIT licensed Amazon FreeRTOS operating system, built on the FreeRTOS kernel v10.
AWS IoT Device and Cloud Views
MCUXpresso SDK Examples
KEY USE
EXPLORATION
Exploring Embedded Cybersecurity Functions & associated Keys

- Secure boot
- Secure Connections
- Encrypted Execution
- Data Confidentiality
- Cloud OTA Update
- Debug Authentication
- Software IP Protection
Cryptographic validation of application code before allowing execution

Attacks mitigated, Security policy and Benefits
- Protection from malware injection from local or remote attacks
- Enforce authenticated boot, authenticated debug and secure OTA process

Creator, storage, and protection
- Created by product owner on a host machine (e.g. HSM)
- Public key data is part of boot image. Protection is achieved through cryptographic validation. In addition Root of Trust Public key is bound the device through Protected flash

Name, type and functions
- Minimum of 2 asymmetric key pairs, Root of Trust and Image key pairs (4 total)
- Private keys are used by host machines. RoT is used to sign Image Certificates, Image is used to sign image data
Protection of software in transit by use of symmetric cryptography

Attacks mitigated, Security policy and Benefits

Interception of software binaries in transit (at manufacturing or in the field)

Protect Software Intellectual Property, Prevent product clones

Creator, storage, and protection

PUF chip master key is created by LPC5500, other subordinate keys are created by a host machine

PUF key is ephemeral and maintained by the LPC5500 key store in Protected Flash. This key is used to protected others pre-shared keys.

Name, type and functions

Four (4) Symmetric Keys, PUF Chip master key, SB Key, SB MAC Key, and SB Data encryption key

PUF is used to decrypt SB Key from protected Flash, SB key is used to decrypt MAC and Data keys passed in binaries.
LPC5500 Series: Secure Binaries

• The Secure Binary (SB) image format is a command-based firmware update image
• The SB 2.0 and 2.1 file format also uses AES encryption for confidentiality and HMAC for extending trust from the signed part of the SB file to the command and data part of the SB file. These two keys (AES decrypt key and HMAC key) are wrapped in the RFC3394 key blob, for which the key wrapping key is the SBKEK key

• User application receives an encrypted SB file containing new firmware and stores it in external SPI flash, or a similar memory.
• Use API to authenticate SB file.
• Use API to decrypt and load the SB file.
• If also using secure boot, the API can be used to authenticate the new firmware in flash before rebooting into it. If this final authentication fails, the new firmware should be made non-executable by erasing and writing over critical regions of it such as the vector table. Even if not using secure boot, the code written to flash can still be signed to support this final authentication step.
Protection of software *at rest* by use of symmetric cryptography

**Attacks mitigated, Security policy and Benefits**

- Chip reverse engineer by physical means. Extraction of software through logical interfaces.
- Protect the confidentiality of software property at rest

**Creator, storage, and protection**

- PUF chip master key is created by LPC5500, PRINCE symmetric key is created by LPC5500
- PUF key is ephemeral and maintained by the LPC5500 key store in Protected Flash. This key is used to protect the PRINCE keys stored in protected flash

**Name, type and functions**

- Minimum 2 symmetric keys (PUF Chip Master and PRINCE region key)
- PUF is used to decrypt PRINCE Key from protected Flash, PRINCE key is instantiated by ROM to support encrypted execution
Protection of data with hardware diversified keys

Attacks mitigated, Security policy and Benefits

- Protect from data extraction via logical interfaces or reverse engineering
- Strong protection for application data (e.g. Sensor data, WiFi credentials, passwords)

Creator, storage, and protection

- PUF chip master key is created by LPC5500, User Data Encryption key is created by LPC5500
- PUF key is ephemeral and maintained by the LPC5500 key store in Protected Flash. This key is used to protected the User Data Encryption Key

Name, type and functions

- Two, (2) Symmetric Keys, PUF Chip master key and the User Key
- PUF is used to decrypt User Key and User Key is used to encrypt sensitive data for the device (Passwords, WiFi credentials, etc.)
Achieve Transport Layer Security for device to cloud connections

<table>
<thead>
<tr>
<th>Attacks mitigated, Security policy and Benefits</th>
<th>Creator, storage, and protection</th>
<th>Name, type and functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection from snooping and man-in-the-middle attacks</td>
<td>Depends on multiple entities to create the Public Key Infrastructure (CA, Server, Client). With PKI in place, key agreement protocols are used to create session keys.</td>
<td>Minimum three (3) asymmetric key pairs, CA, Server and Client. One Symmetric key that is reached by key agreement</td>
</tr>
<tr>
<td>Trusted services through validation of the authenticity of the device to cloud connection and confidentiality through key agreement</td>
<td>For the device (LPC5500), Pub/Priv key material is protected by secure boot and Software IP protections.</td>
<td>LPC5500 has access to CA public key to validate server public key. Also LPC5500 uses Client Private key to respond to server challenge and reach key agreement</td>
</tr>
</tbody>
</table>
TLS Handshake (Source: AN12131)
Cloud based fleet management services for secure OTA

Attacks mitigated, Security policy and Benefits
- Prevent tampering of device firmware updates
- Implement a secure OTA for group of devices

Creator, storage, and protection
- Host machine is used to create an asymmetric key pair for Cloud OTA
- OTA signing key is protected by host machine, OTA public key is protected by LPC5500 Secure boot

Name, type and functions
- Asymmetric key pair OTA private and OTA public keys
- OTA public key is present in the device so that the OTA image can be validated
OTA Jobs from AWS

CREATE JOB
Select a job

AWS IoT Device Management job orchestration and notification service allows you to define a set of remote operations called jobs that are sent to and executed on one or more devices connected to AWS IoT.

Create a custom job
Send a request to acquire an executable job file from one of your IoT buttons to one or more devices connected to AWS IoT.

Create an Amazon FreeRTOS OTA update job
This On-Device OTA update job will send your firmware image securely over MQTT to Amazon FreeRTOS-based devices.

CREATE JOB
Create an Amazon FreeRTOS OTA update job

Select and sign your firmware image
Code signing ensures that devices only run code published by trusted authors and that the code has not been altered or corrupted since it was signed. You have three options for code signing. Learn more

- Sign a new firmware image for me
- Select a previously signed firmware image
- Use my custom signed firmware image

Code signing profile Learn more
No code signing profile selected
Create Select

Select your firmware image in S3 or upload it
Image not selected
Select

Pathname of firmware image on device Learn more
e.g. /device/updates
### Use of cryptography to open access to device Debug capabilities

#### Attacks mitigated, Security policy and Benefits
- Prevent firmware tampering at the device or re-profiling of the device with malicious software
- Restrict logical interfaces on the device

#### Creator, storage, and protection
- Debug entity creates the keys related authenticated debug
- Debug entity must protect the private key, the public key is cryptographically validated

#### Name, type and functions
- Authenticated debug private key and public key is an Asymmetric key pair.
- The Private key is used by a host machine to sign a challenge provided by the LPC5500, the public key is passed to the LPC5500 and validated with the Root Of Trust Public key before use
Secure debug
Debug authentication for RMA use case

1. OEM generates RoT key pairs and programs the device before shipping.
   - SHA256 hash of RoT public key hashes
2. Field Technician generates his own key pair and provides public key to OEM for authorization.
3. OEM attests the Field Technician’s public key. In the debug credential certificate he assigns the access rights.
4. End customer having issues with a locked product takes it to Field technician.
5. Field technician uses his credentials to authenticate with device and un-locks the product for debugging.
KEY MANAGEMENT TABLE SUMMARY (REFERENCE)
## Secure Boot

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Type Name</th>
<th>Created By</th>
<th>Function</th>
<th>Storage</th>
<th>Used by</th>
<th>Protected by</th>
<th>Benefit</th>
<th>Security Policy</th>
<th>Attacks Mitigated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymmetric-RSA</td>
<td>Root Of Trust Private Keys</td>
<td>Product Owner - Host Machine, OpenSSL, HSM</td>
<td>Sign Image Certificates, Sign Debug credentials</td>
<td>Example: Host machine</td>
<td>Product Owner or their designated entity</td>
<td>Host Machine</td>
<td>Security Policy, Boot, Debug access, Secure OTA process</td>
<td>Local or Remote malware injection, Logical interface attacks (JTAG)</td>
<td></td>
</tr>
<tr>
<td>Asymmetric-RSA</td>
<td>Root of Trust Public Keys</td>
<td>Same as above</td>
<td>Validate Image Certificate, Debug credentials</td>
<td>Hash stored in Protected Flash space on the LPC5500</td>
<td>Chip itself for secure boot authentication of Debug credentials</td>
<td>Chip firewalls</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric-RSA</td>
<td>Image Private Key</td>
<td>Same as above</td>
<td>Sign boot data including app code</td>
<td>Example: Host machine</td>
<td>Dat</td>
<td>Host Machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric-RSA</td>
<td>Image Public Key</td>
<td>Same as above</td>
<td>Validate boot data including application code</td>
<td>Part of boot data</td>
<td>Chip itself for secure boot authentication of root of trust public key</td>
<td>Cryptographic Validation using root of trust public key</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Secure Connections

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Type Name</th>
<th>Created By</th>
<th>Function</th>
<th>Storage</th>
<th>Used by</th>
<th>Protected by</th>
<th>Benefit</th>
<th>Security Policy</th>
<th>Attacks Mitigated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymmetric</td>
<td>Cloud Provider Certificate Authority Public key</td>
<td>Certificate Authority</td>
<td>Validate the identity of the cloud connection</td>
<td>At the device in Application Code image data as a certificate</td>
<td>Application Code</td>
<td>TLS handshake</td>
<td>Secure Boot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Cloud Provider Certificate Authority Private key</td>
<td>Certificate Authority</td>
<td>Sign Certificates in the Public Key Infrastructure</td>
<td>Certificate Authority</td>
<td>Application Code for TLS handshake</td>
<td>Certificate Authority</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Server Private Key</td>
<td>Cloud Service</td>
<td>TLS Handshake</td>
<td>Cloud Service</td>
<td>Chip for validating Server and reaching key agreement</td>
<td>Secure Boot-CA Public Key-Validation of Server certificate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Server Public Key</td>
<td>Cloud Service</td>
<td>TLS Handshake</td>
<td>Cloud Service</td>
<td>Chip for signing server challenges and key agreement protocol</td>
<td>PUF Chip Master Key encryption of Chip User Key - Chip user key encryption</td>
<td></td>
<td>Validate the authenticity of the cloud connection</td>
<td>Establish Trusted Connections, Protect data in transit</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Client Private Key</td>
<td>Options (Host machine, Chip itself, Cloud Provider)</td>
<td>TLS Handshake</td>
<td>Image Data, encrypted by Chip User Key</td>
<td>Server for validating authenticity of the client and reaching key agreement</td>
<td>Secure Boot at the device - CA public key validation by the server</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Client Public Key</td>
<td>Options (Host machine, Chip itself, Cloud Provider)</td>
<td>TLS Handshake</td>
<td>Image Data</td>
<td>Server for validating authenticity of the client and reaching key agreement</td>
<td>Secure Boot at the device - CA public key validation by the server</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric</td>
<td>Session Key</td>
<td>Key agreement protocols based on Public Key Cryptography</td>
<td>TLS Handshake</td>
<td>Image Data</td>
<td>Shared secret for data confidentiality between server and client</td>
<td></td>
<td></td>
<td>Application Software</td>
<td></td>
</tr>
</tbody>
</table>
## Cloud OTA and Authenticated Debug

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Type Name</th>
<th>Created By</th>
<th>Function</th>
<th>Storage</th>
<th>Used by</th>
<th>Protected by</th>
<th>Benefit</th>
<th>Security Policy</th>
<th>Attacks Mitigated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymmetric</td>
<td>Cloud OTA Image Private Key</td>
<td>Host machine</td>
<td>Sign binaries pushed by the cloud</td>
<td>Host machine</td>
<td>Host machine for providing</td>
<td></td>
<td>Validate the Authenticity of image data sent by the cloud</td>
<td>Secure OTA</td>
<td>Firmware Tampering</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Cloud Based OTA Image Public Key</td>
<td>Host machine</td>
<td>Validate binaries received by the cloud</td>
<td>Host machine</td>
<td>Secure boot</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric-RSA</td>
<td>Authenticated Debug Private Key</td>
<td>Host machine of the Entity whom will debug</td>
<td>Respond to Chip Authenticated Debug Challenge</td>
<td>Host machine</td>
<td>Host machine for providing signatures to the LPC5500 during debug authentication</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asymmetric-RSA</td>
<td>Authenticated Debug Public key</td>
<td>Host machine of the Entity whom will debug</td>
<td>Sent to LPC5500 in order to be used in order to validate a Debug Challenge response</td>
<td>Host machine</td>
<td>Host Machine Signing done by Root Of Trust private key on a host machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Cloud Based OTA Image
- **Asymmetric**: Cloud OTA Image Private Key
  - Created By: Host machine
  - Function: Sign binaries pushed by the cloud
  - Storage: Host machine
  - Used by: Host machine for providing signatures to AWS
  - Protected by: Host machine
  - Benefit: Validate the Authenticity of image data sent by the cloud
  - Security Policy: Secure OTA
  - Attacks Mitigated: Firmware Tampering

### Cloud Based OTA Image Public Key
- **Asymmetric**: Cloud Based OTA Image Public key
  - Created By: Host machine
  - Function: Validate binaries received by the cloud
  - Storage: Host machine
  - Used by: Secure boot

### Authenticated Debug
- **Asymmetric-RSA**: Authenticated Debug Private Key
  - Created By: Host machine of the Entity whom will debug
  - Function: Respond to Chip Authenticated Debug Challenge
  - Storage: Host machine
  - Used by: Host machine for providing signatures to the LPC5500 during debug authentication

- **Asymmetric-RSA**: Authenticated Debug Public key
  - Created By: Host machine of the Entity whom will debug
  - Function: Sent to LPC5500 in order to be used in order to validate a Debug Challenge response
  - Storage: Host machine
  - Used by: Signing done by Root Of Trust private key on a host machine

- **Asymmetric-RSA**: Authenticated Debug Public key
  - Created By: Host machine of the Entity whom will debug
  - Function: Sent to LPC5500 in order to be used in order to validate a Debug Challenge response
  - Storage: Host machine
  - Used by: Signing done by Root Of Trust private key on a host machine

- **Asymmetric-RSA**: Authenticated Debug Public key
  - Created By: Host machine of the Entity whom will debug
  - Function: Sent to LPC5500 in order to be used in order to validate a Debug Challenge response
  - Storage: Host machine
  - Used by: Signing done by Root Of Trust private key on a host machine

### Security Policy
- **Secure OTA**: Protects against firmware tampering
- **Firmware Tampering**: Mitigates attacks against firmware integrity

### Attacks Mitigated
- **Firmware Tampering**: Protects against malicious firmware modifications
# Software protection, Encrypted Execution and Data Confidentiality

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Type Name</th>
<th>Created By</th>
<th>Function</th>
<th>Storage</th>
<th>Used by</th>
<th>Protected by</th>
<th>Benefit</th>
<th>Security Policy</th>
<th>Attacks Mitigated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric-AES</td>
<td>PUF Chip Master Key</td>
<td>PUF on the chip itself</td>
<td>Key encryption key for other keys</td>
<td>Activation code (non secret) stored in Protected Flash, Ephemeral key creation upon software request.</td>
<td>Chip for decoding other Keys and protected data</td>
<td>Protection of other keys which provide confidentiality of data (Application code, etc.)</td>
<td>Protect Software IP, Never store key material in plain text, Enforce key diversity (Unique key per chip/device),</td>
<td>Extracting device key material from logical interfaces</td>
<td>Interception of software in transit (at manufacturing or in the field),</td>
</tr>
<tr>
<td>Symmetric-AES</td>
<td>SB Key</td>
<td>Product owner, host machine</td>
<td>Decrypt the Binary Key</td>
<td>KEY STORE: Stored as a key code which is encrypted by PUF Chip Master key</td>
<td>Chip for decoding Binary Key during bootloading</td>
<td>Confidentiality of Software IP</td>
<td>Protect Software IP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric-MAC</td>
<td>SB MAC Key</td>
<td>Host machine (elftosb)</td>
<td>Check the integrity of the SB file header data</td>
<td>Encrypted by SB Key and part of the SB file</td>
<td>Chip for checking integrity of the Header Data in SB file</td>
<td>PUF Chip Master Key encryption of SB Key and SB Key</td>
<td>confidentiality of Software IP</td>
<td>Protect SW IP</td>
<td>attacks on binaries being passed to the device</td>
</tr>
<tr>
<td>Symmetric - AES</td>
<td>SB Data Encryption Key</td>
<td>Host machine (elftosb)</td>
<td>Encrypt image data in SB files</td>
<td>Encrypted by SB Key and part of the SB file</td>
<td>Chip for decrypting image data for loading</td>
<td>PUF Chip Master Key encryption of SB Key and SB Key</td>
<td>Confidentiality of Software IP</td>
<td>Protect SW IP in transit</td>
<td>attacks on binaries being passed to the device</td>
</tr>
<tr>
<td>Symmetric-PRINCE</td>
<td>PRINCE Key</td>
<td>Chip itself using PUF</td>
<td>Encrypted Execution</td>
<td>KEY STORE: Stored as a key code which is encrypted by PUF Chip Master key</td>
<td>Chip for encrypted execution</td>
<td>PUF Chip Master Key encryption of Prince Key only known by the chip</td>
<td>Confidentiality of Software IP</td>
<td>Protect SW IP in use and storage</td>
<td>Chip reverse engineering or data extraction from logical interfaces</td>
</tr>
<tr>
<td>Symmetric- AES</td>
<td>Chip User Key</td>
<td>Host machine or Chip Itself</td>
<td>Protect data managed by the device</td>
<td>KEY STORE: Stored as a key code which is encrypted by PUF Chip Master key</td>
<td>Application Code</td>
<td>PUF Chip Master Key encryption of Chip User Key</td>
<td>Protect data at rest with diversified keys (Sensor data, passwords, WiFi credentials)</td>
<td>Protect data at rest</td>
<td>Extraction of data from logical interfaces</td>
</tr>
</tbody>
</table>
CONCLUSION
Summary

- It has never been so easy to get a device connected and create an IoT Device
  - This is both amazing and frightening at the same time
- Many device types share a common set of assets that must be protected by security functions
  - Secure devices make proper use of cryptography to perform security functions
  - Both Symmetric and Asymmetric cryptography is used
  - Hardware protection of the keys is essential for protecting the device
- State of the art Microcontrollers like LPC5500 series integrate technology to achieve security functions
  - PUF based key management
  - Enabled by ROM

Thanks!