SmartFusion2 – Embedded system "Root-of-Trust"
Secure Boot Demonstration

Peter Trott
Snr FAE - Microsemi
Agenda

- Embedded System Threats
- Secure Boot / Root-of-Trust (RoT)
- Multi-Stage CPU Boot Process
- Non-Secure Processor Problem
- Microsemi Secure Boot Solution
- Secure Boot Demo
Embedded Systems Threats

- We are under attack!
- Hacking of embedded microprocessor based systems is on the rise
  - Easy to obtain embedded products
  - Readily available tools / techniques on hacker websites
- Improved security measures required to counter these increasing sophisticated attacks.
Secure Boot

- Cornerstone of embedded system security is to ensure that only **authorized** microprocessor code is loaded and executed.

- The process of guaranteeing the execution of authentic code is called ‘Secure Boot’

- Booting processors from known trusted code
  - Prerequisite for the secure operation of embedded systems

- If the initial hardware and boot code can be trusted, then this trust can be extended to code loaded and executed later.

- The foundation for Secure Boot is the Root-of-Trust (RoT)
Root-of-Trust (RoT)

- A root-of-trust is essential to implement secure boot

- Root-of-trust is an entity that can be trusted to behave in an expected manner
  - Supports verification of system, software and data integrity
  - Keeps keys and critical data confidential
  - Its processes are immutable
  - Works in conjunction with other system elements to ensure system security

A Root-of-Trust is the foundation upon which all other security layers are created and trusted.
Root-of-Trust Examples

- Trusted Platform Module (TPM)
  - Industry standard external Root-of-Trust device
  - Provides cryptographic services with static RSA key

- Processors with built in security features
  - One time programmable key storage
  - Immutable on-chip microcode
  - May not support the full suite of security functions needed
    - Anti-tamper, data encryption at rest, DPA resistance, ROM scrambling, etc

- Select SoC FPGAs (SmartFusion2 / Igloo2)
  - Cryptographic services, True random number generator
  - Stronger design security, On-chip oscillators
  - Anti-tamper measures (DPA resistance, Zeroization)
  - On-chip Flash
  - Vastly better computational power, more IOs and a variety of built-in interfaces
CPU Multi-Stage Boot Process

- Typical Multistage Boot (unsecure)
  - From boot to app its non stop execution!

- Secure Multistage Boot
  - Initialize an embedded system from rest with trusted code
  - Validation of each stage performed by the prior stage
    - Establishes a ‘chain of trust’ all the way to the top application layer
Secure Multi-Stage Boot Code Validation

- Code validation can be done by symmetric or asymmetric key cryptography techniques
  - Build an inherently trusted key (public) into the phase-0 boot loader
  - Phase-1 code is digitally signed using key (private)
  - During Phase-0 the system validates the digital signature of the Phase-1 code, prior to execution
    - Boot if signature validates
    - Abort boot apply system penalties if invalid

- Immutability of the Phase-0 Key is critically important
  - Anti-Tamper Monitors
Non-Secure Processors

- Secure boot is challenging when the target processor has no inherent security capabilities
  - Does not have root-of-trust capability
  - Low end Microprocessors, Microcontrollers, DSP’s

- Customers in many market segments face this issue
  - Telecom, Military, Industrial, Medical, Energy

- Processors located in peripheral or remote subsystems are especially vulnerable to attack
Secure Boot Hardware Configuration

- SmartFusion2/Igloo2 can be used as a root-of-trust
  - Assist in securing the multi-phase boot processes

**Diagram Description**

- **Phase 1 - Code**
  1. BIOS
  2. OS Loader
  3. OS
  4. Application code

- **Hardware Root-of-Trust**
  - SmartFusion®2 Secure SoC FPGA
    - Cortex™-M3
    - OSC
    - TRNG
    - eSRAM
    - PUF
    - eNVM
    - FPGA

- **Target Processor**
  - Main MPU
    - CPU
    - SRAM

- **Additional Notes**
  - Tight integration with other board functions such as power management make bypassing the HW root-of-trust more difficult
  - Code loaded into on-chip SRAM is validated before branching to it

© 2015 Microsemi Corporation. COMPANY PROPRIETARY
Secure Boot Process (Phase-0 and 1)

**Phase-0 Boot Code**
- Load/Exec. Phase-0 SW
- Calculate CBC-MAC (w/ Whitebox\textsc{Crypto})
- Unwrap Phase-1 Keys
- Load, Validate & Decrypt Boot-1 SW

**Phase-1 Boot Code**
- Execute Phase-1 SW

**Target Processor** SmartFusion®2

**Nonce, N**
- RSA Public Key
- RSA Private Key

**CBC-MAC Tag**
- {{IV,ESK|N}_{AES-CBC}}_{RSA}

**SRAM Start-up Values**
- Other penalties?

**Integrity Self-test**
- Remove Reset
- Deliver Phase-0 SW
- Generate/ Deliver Nonce
- Calculate & Validate MAC tag
- Decrypt Shared Key
- Wrap Phase-1 Keys
- Apply Penalties if Response is Invalid

**Phase-0 Boot Code & Data**
- (All Phase-0 Boot Code & Data)

**Phase-1 Encrypted / Validated Boot Code**
- IV
- MAC tag

**(MAC)** (ephemeral shared key, ESK)
- (Phase-1 Keys)

(ephemeral shared key, ESK)
Secure Boot in a ‘nut shell’

- Assumes processor has no inherent security capabilities

- Securely boot processor with Phase-0 code

- Establish a AES session key between processor and SmartFusion2 RoT to secure the communication channel

- Transfer subsequent Phase-x code and code validation keys over secure channel to target processor.

- Target processor will validate Phase-x code
  - Execute code if validated
  - Apply system penalties if code does not validate
Load/Validate Phase-0 Code

Load/Exec. Phase-0 SW

Calculate CBC-MAC (w/ Whitebox\(^{\text{CRYPTO}}\))

Nonce, N

RSA Public Key

(All Phase-0 Boot Code & Data)

Phase-0 Boot Code

Phase-1 Boot Code

Target Processor

SmartFusion\(^{\text{2}}\)

Integrity 1

Remove Reset 2

Deliver Phase-0 SW 3

Generate/ Deliver Nonce 5

Calculate & Validate MAC tag 7

Apply Penalties if Response is Invalid 8

Phase-0 Boot Code

(AES Key (same as used w/ Whitebox\(^{\text{CRYPTO}}\))

(Boot Code)

(Nonce)

(Phase-0 Boot Code & Data)

(AES Key (same as used w/ Whitebox\(^{\text{CRYPTO}}\))

(Boot Code)

(Nonce)

(From eNVM)

(From eNVM)
Establish Ephemeral Session Key (ESK)

Phase-0 Boot Code

1. Load/Exec. Phase-0 SW
2. Calculate CBC-MAC (w/ Whitebox CRYPTO)
3. Calculate/Encrypt ESK
4. Nonce, N
5. RSA Public Key

Phase-1 Boot Code

6. (MAC)
7. (based on SRAM entropy)
8. SRAM Start-up Values

Target Processor

SmartFusion®2

1. Integrity Self-test
2. Remove Reset
3. Deliver Phase-0 SW
4. Generate/Deliver Nonce
5. Calculate & Validate MAC tag
6. CBC-MAC Tag
7. {{IV,ESK|N}_{AES-CBC}}_{RSA}
8. AES Key (same as used with Whitebox CRYPTO)
9. RSA Private Key
10. Decrypt Shared Key

© 2015 Microsemi Corporation. COMPANY PROPRIETARY
Share Phase-x Encryption/Validation Keys

(All Phase-0 Boot Code & Data)

**Phase-0 Boot Code**

- Load/Exec. Phase-0 SW
- Calculate CBC-MAC (w/ Whitebox CRYPTO)
- Nonce, N
- RSA Public Key
- Calculate CBC-MAC (w/ Whitebox CRYPTO)
- Calculate/Encrypt Shared Key
- RSA Private Key
- Apply Penalties if response is invalid
- AES Key (same as used with Whitebox CRYPTO)
- RSA Public Key
- Generate/Deliver Phase-0 SW
- **Phase-0 Boot Code**
- **Phase-0 Boot Code**

**Phase-1 Boot Code**

- Unwrap Phase-1 Keys
- {Keys}_{ESK}
- {IV,ESK|N}_{AES-CBC}_{RSA}
- Phase-1 Boot Code Encryption & Authentication Keys
- Phase-1 Boot Code

**Target Processor**

SmartFusion®2

- Remove Reset
- Deliver Phase-0 SW
- Integrity Self-test
- Deliver Phase-0 SW
- Generate/Deliver Nonce
- Calculate & Validate MAC tag
- Decrypt Shared Key
- Calculate CBC-MAC Tag

(from NRBG)

(from eNVM)

(from eNVM)
Load/Validate/Decrypt/Execute Phase-x Code

Phase-0 Boot Code

- Calculate CBC-MAC (w/ Whitebox\textsuperscript{CRYPTO})
- Calculate/Encrypt Shared Key
- Unwrap Phase-1 Keys
- Load/Validate & Decrypt Phase-1 SW

Phase-1 Boot Code

- Load, Validate & Decrypt Phase-1 SW

Target Processor
SmartFusion\textsuperscript{®}2

Other penalties?
(SRAM Start-up values)

CBC-MAC Tag

\{\{IV,ESK|N\}\_(AES-CBC)\_RSA

(Halt if Invalid)

MAC tag

Phase-1 Encrypted / Validated Boot Code

Integrity
- Self-test

Remove
- Reset

Deliver
- Boot-0 SW

Generate/
Deliver
- Nonce

Calculate &
Validate MAC tag

Decrypt
- Shared Key

Wrap
- Phase-1 Keys

Phase-0 Boot Code

AES Key
(same as used with Whitebox\textsuperscript{CRYPTO})

RSA
Private Key

Apply Penalties if response is invalid

Phase-1 Boot Code Encryption & Authentication Keys

(Phase-1 Keys)

(Nonce)

(Boot Code)

(RSA Public Key)

(All Phase-0 Boot Code & Data)

(ephemeral shared key, ESK)

(based on SRAM entropy)

(RSA Public Key)

 MAC tag

(from external PROM)

(from NRBG)

(from eNVM)
Summary

- SmartFusion2/Igloo2 are ideal devices to implement RoT
  - Anti-tamper and Integrity check hardware
  - Data storage with HW firewalls
  - True HW RNG and AES engine
  - Significant FPGA space for system integration, tamper monitoring and strong penalties.

- Secure Boot architecture is scalable
  - Modify based on target processors capabilities

- Single Secure Boot solution
  - Same solution regardless of processor capability
  - Ideal for customers deploying different processors.
Phase-0 Secure Boot Process Flow

Serve up Phase-0 code from eNVM over SPI to CPU.
Data includes:
- 256bit AES key (Whitebox)
- 128 byte nonce
- RSA Public Key
- Application code

In Parallel, Feed data to AES - CBC to complete MAC Tag calculation.

Compare calculated MAC to Received MAC.
If mismatch assert RESET.
Secure Boot Demo

- Demonstrate secure boot of a non-secure processor
  - Microprocessor that has no inherent security capabilities
  - SmartFusion Eval board used for convenience

- Demonstrates Phase-0 and Phase-1 boot stages
  - Boot an unsecure microprocessor with trusted code
  - Validate that microprocessor is executing this trusted code

- Demonstrates use of SmartFusion2 as a root-of-trust
Secure Boot Device Resource Utilization

- Cortex-M3 Based SmartFusion2 Utilization (Phase-0 and 1)

<table>
<thead>
<tr>
<th>Resource</th>
<th>M2S005</th>
<th>M2S050</th>
</tr>
</thead>
<tbody>
<tr>
<td>1170 LUTS</td>
<td>24%</td>
<td>2.4%</td>
</tr>
<tr>
<td>7 uSRAMs</td>
<td>64%</td>
<td>10%</td>
</tr>
<tr>
<td>0 LSRAMs</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>64kB eNVM</td>
<td>50%</td>
<td>25%</td>
</tr>
</tbody>
</table>

- CoreABC based SmartFusion2/Igloo2 utilization (Phase-0 only)

<table>
<thead>
<tr>
<th>Resource</th>
<th>M2S005</th>
<th>M2S050</th>
</tr>
</thead>
<tbody>
<tr>
<td>2300 LUTS</td>
<td>46%</td>
<td>4.7%</td>
</tr>
<tr>
<td>7 uSRAMs</td>
<td>64%</td>
<td>10%</td>
</tr>
<tr>
<td>1 LSRAMs</td>
<td>10%</td>
<td>1.4%</td>
</tr>
<tr>
<td>32kB eNVM</td>
<td>25%</td>
<td>12.5%</td>
</tr>
</tbody>
</table>

- Performance
  - Max SPI Clock = 30MHz (eNVM & AES limited)
Thank You!

Microsemi Corporation
One Enterprise, Aliso Viejo, CA 92656
Ph: 949-380-6100
Fax: 949-215-4996
www.microsemi.com
For more information: Sales.Support@microsemi.com
Backup Slides
Secure Boot Using CoreABC

- SmartFusion2 / Igloo2
- MSS
- FIC AHB Slave
- APB-to-AHB Wedge
- CoreABC
- CoreAHBlite
- SPI Slave Controller
- TinyAES CBC 8-bit
- AES Control SM
- SPI Bus