Fundamentals of HW-based Security

Terry Kim
Senior FAE, ARM

ARM Tech Forum 2016 – Singapore
August 30th 2016

©ARM 2016
What is system security design?

- Every system design will require a **different set** of security properties, depending on the type and value of the assets it is trying to defend against malicious attacks.

- Security solutions are designed to defend against only a **subset** of the possible attacks that they may experience.
Limitations of security systems

- A Security solution’s designer must decide which assets he/she wants to protect, and which of the possible attacks he/she wants to protect the assets against.

- **HW Attacks**
  - Physical access to device – JTAG, Bus, IO Pins,
  - Well resourced and funded

- **Software Attacks**
  - Buffer overflows
  - Interrupts
  - Malware

- **Communication Attacks**
  - Man In The Middle
  - Weak RNG
  - Code vulnerabilities
3 Fundamental (system) security properties

- Confidentiality
  - An asset that is confidential cannot be copied or stolen by a defined set of attacks.

- Integrity
  - An asset that has its integrity assured is defended against modification by a defined set of attacks

- Authenticity
  - When integrity can’t be provided, the defender is left with the capability to detect changes before the asset is used
HW Vs. SW

Once the level of security we want/need is known, what should be the split between HW and SW mechanisms?

https://bloggeek.me/hardware-trumps-software/
HW based security -

Part 1: What security mechanisms SW can’t provide?

Part 2: What security mechanisms HW needs to enforce?
Chapter 1: What security mechanisms SW can’t provide?
(SW can’t provide) Entropy*
(*Or Randomness)

• “One thing that traditional computer systems aren’t good at is coin flipping... They’re deterministic, which means that if you ask the same question you’ll get the same answer every time.”

*Prof. Steve Ward, Computer Science at MIT*
Why does security mechanisms require entropy?

- **Security needs unpredictability**, amongst other things, for key generation
- If an adversary predicts a cipher key, he/she needs not to “break” the cipher
What makes one entropy source better than another?

- A source supplying “Full Entropy” would pass strict Auto-Correlation tests
- It will also demonstrate no bias or pattern repetition

Von Neumann’s bias-removal algorithm

Input: 1 0 1 0 1 1 1 1 0 1 0 1 1 0
  0 1 0 0 1 0 1 1 0 1 1 1 1
Output: 1 1 0 0 0 1
   0 1 0
What about “entropy” sources available to SW?

There are known vulnerabilities of schemes which are not based on “True Random Number Generation” (e.g. CPU stats, date, etc.)

---

**Weak randomness in Android’s DNS resolver**

CVE-2012-2808

Roee Hay & Roi Saltzman
<br>roeeh,roisa@il.ibm.com

IBM Application Security Research Group

July 24, 2012

3 Vulnerability

Let $t_{EID}$ be the time of which the victim generates $R_{EID}$ by calling res_randomid, and let $t_{PORT}$ be the time that $R_{PORT}$ is generated. Both time values are in µsec precision, and both random values are generated by calling res_randomid. Since that function is used twice, in a very short time, $t_{EID}$ and $t_{PORT}$ become very much correlated which has a direct impact on the correlation between $R_{EID}$ and $R_{PORT}$.
How do we do it in CryptoCell?

- Use a Physical random number generator, i.e. measure some physical property
- In CC’s case: we measure timing variations due to power supply noise
How do we do it in CryptoCell?
How do we do it in CryptoCell?
ARM’s Beetle test chip’s TRNG @ Embedded World
(SW can’t provide) Tamper Resistance

- Tampering/physically attacking involves sophisticated, expensive kinds of attacks

Physical attacks:

- Usually split into 2 categories:
  - Non-Invasive attacks
    - Side Channel attacks
    - Fault injection attacks
  - Invasive attacks
    - reverse engineering
    - Microprobing
    - chip modification
    - More..
Physical attacks:

- Mitigation of these attacks comes in many shapes and forms
- Most have implications on area and power consumption
  - Some also impact throughout
Chapter 2: What security mechanisms HW needs to enforce?
Introduction to HW enforced security

- Many forums address the need to standardize trust in embedded systems, including:
  - The Trusted Computing Group (TCG) was formed by AMD, HP, IBM, Intel and Microsoft to implement trusted computing concepts across personal computers
  - Open Mobile Terminal Platform (OMTP) included manufacturers such as Huawei, LG Electronics, Motorola, Nokia, Samsung and Sony Ericsson
  - GlobalPlatform (GP) identifies, develops and publishes specifications that promote secure chip technology. Members include AMD, Apple, ARM, Broadcom, Huawei, Samsung, etc.
Introducing TEE

OMTP and GP outlined a framework for sensitive data to be stored, processed and protected in an isolated, trusted environment, a.k.a. Trusted Execution Environment (TEE).

Such an environment is primarily made out of:

- A Separate (on/off die) or virtualized security processor
  - code running on this processor must be loaded securely using assets bound to the SoC and isolated from non-TEEs
- Interconnect capable of isolating the security data/control path from the non secure world
ARM TrustZone based TEE

- ARM TrustZone technology enables this level of system-wide security by integrating protective measures into the ARM processor, bus fabric, and system peripheral IP
- Latest addition to the TrustZone family is CryptoCell, which is designed to help ARM’s Silicon partners reach a robust TEE quickly (more details in following slides)
Trusted Base System Architecture (TBSA)

- TBSA is a document ARM provides its partners to introduce the concepts behind the TrustZone technology from a hardware viewpoint, emphasizing the concept of isolation of the trusted world from the non-trusted world.
- The document describes the minimal features set that the Trusted Base System hardware must fulfill to be compliant with OMTP and GP’s specs.
## Main HW requirements in TBSA

<table>
<thead>
<tr>
<th>Where?</th>
<th>What?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>implementing 2 states: secure and non-secure</td>
</tr>
<tr>
<td>System bus</td>
<td>Distinguishes secure and non-secure transactions</td>
</tr>
<tr>
<td>System memory</td>
<td>Separates secure and non-secure ranges (on-die RAMs, off-die DRAM, cache-lines)</td>
</tr>
<tr>
<td>Interrupt controller</td>
<td>Distinguishes secure and non-secure peripherals</td>
</tr>
<tr>
<td>Debug solution</td>
<td>capable of restricting access to secure assets</td>
</tr>
<tr>
<td>Boot ROM</td>
<td>Must be on-die</td>
</tr>
<tr>
<td>RNG</td>
<td>Implemented according to industry standards</td>
</tr>
<tr>
<td>Non-Volatile Memory</td>
<td>Around 1 Kb, required for storing roots of trust, NV counters, etc.</td>
</tr>
<tr>
<td>Trusted clock, timers and watchdog</td>
<td>Implemented according to specific requirements</td>
</tr>
</tbody>
</table>
ARM’s TrustZone Alliance Program: Compliance Coverage

- More than half the TBSA and TBBR requirements are covered by using Cryptocell + other ARM IPs
  - TBBR: >60% requirements covered
  - TBSA: >50% requirements covered
  - Remaining requirements relate to subsystem integration
- Cryptocell provides a fast and proven path to TEE implementation

<table>
<thead>
<tr>
<th>TBBR Boundary 1 Compliance</th>
<th>TBSA Boundary 1 Compliance</th>
<th>TBSA Boundary 2 Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partially compliant</td>
<td>Partially compliant</td>
<td>Partially compliant</td>
</tr>
<tr>
<td>Not compliant</td>
<td>Not compliant</td>
<td>Not compliant</td>
</tr>
<tr>
<td>Compliant</td>
<td>Compliant</td>
<td>Compliant</td>
</tr>
<tr>
<td>Not addressed (mandatory)</td>
<td>Addressed by ARM</td>
<td>Not addressed (mandatory)</td>
</tr>
</tbody>
</table>

26 ©ARM 2016
TrustZone® CryptoCell – High Level View

Host direct operation (REE, TEE)

Control interface

Security resources
- NVM
- LCS
- RNG
- Key slots
- Secure Boot
- Secure Debug
- Version control
- Secure Storage
- Reflash Protection
- Code decryption
- Provisioning
- Secure Power State Transition

Asymmetric Cryptography
- Configuration
- Arbitration
- Scheme control
- ALU
- Arbitration

Symmetric Cryptography
- Arbitration
- Tasks submission
- Crypto
- DMAs

Data interface
- Master Bus Interface
- FIFOs

Persistent roots of trust
- Always On

System memory

Provisioning
- Secure Storage
- Secure Debug
- Secure Power State Transition
- Code decryption
- Provisioning
- NVM
- LCS
- RNG
- Key slots

Slave Bus interface
- Decode
- Queuing
HW enforcement’s role in the creation of a Root of Trust

HW enforcement of security policies is at the very heart of the trust you need to establish in your device before trusting it with your assets:

- HW enforces the fact only CryptoCell has access to Root keys (like a HW unique Key)
- HW enforces that only secure bus masters may ask CryptoCell to use the HUK
- HW enforces the fact TEE is instantiated securely
Use case example: fingerprint

7.3.10. Fingerprint Sensor

Device implementations with a secure lock screen SHOULD include a fingerprint sensor. If a device implementation includes a fingerprint sensor and has a corresponding API for third-party developers, it:

- MUST declare support for the android.hardware.fingerprint feature.
- MUST fully implement the corresponding API as described in the Android SDK documentation [Resources, 95].
- MUST have a false acceptance rate not higher than 0.002%.
- Is STRONGLY RECOMMENDED to have a false rejection rate not higher than 10%, and a latency from when the fingerprint sensor is touched until the screen is unlocked below 1 second, for 1 enrolled finger.
- MUST rate limit attempts for at least 30 seconds after 5 false trials for fingerprint verification.
- MUST have a hardware-backed keystore implementation, and perform the fingerprint matching in a Trusted Execution Environment (TEE) or on a chip with a secure channel to the TEE.
- MUST have all identifiable fingerprint data encrypted and cryptographically authenticated such that they cannot be acquired, read or altered outside of the Trusted Execution Environment (TEE) as documented in the implementation guidelines on the Android Open Source Project site [Resources, 96].
In summary:

- In some cases, HW based security provides mechanisms SW simply can’t provide
- In others, e.g. TrustZone based TEEs, HW based security can provide a framework that allows a diverse range of secure system architectures to be implemented with minimal impact on the cost of the device
Questions?