SOC - Spot the Odd Circuit
Android Software Stack

Applications
Home, Contacts, Phone, Browser, ...

Application Framework
Managers for Activity, Window, Package, ...

Libraries
SQLite, OpenGL, SSL, ...

Runtime
Dalvik VM, Core libs

Linux Kernel
Display, camera, flash, wifi, audio, IPC (binder), ...

Hardware Stack
Cortex®-A53

ARM CoreSight™ Multicore Debug and Trace

- ARMv8-A 32b/64b CPU
- 8-64k I-Cache w/parity
- 8-64k D-Cache w/ECC
- Floating Point Unit
- NEON™ SIMD engine with crypto ext.

ACP  |  SCU  |  L2 w/ECC (128kB ~ 2MB)

Configurable AMBA®4 ACE or AMBA5 CHI Coherent Bus Interface
Why SCA on a SoC might be Hard

Reality we are facing

- High Speed more than one Giga Hertz
  - Acquisition chain and setup matters
  - Identify Crypto activity not that simple
- Nanometer Technology
- Two Cache Levels Cache Activities perturbates the acquisition
- Running in a complex OS context
  - Interruption
  - Multiple Crypto options (SW libraries, HW)
  - Several Application Options
- Access not always easy
- HW AES Enc in 20 cycles ...

For all these reasons, studying one of the most deployed core like the ARMv8 is interesting
First Target for SCA - Hisilicon Kirin 620 SoC on the Hikey board

8 x ARM © A53 Cores with NEON™, Freq 1.2Ghz
On the SCA Bench ...
EM Activity

While AES encryption, using OpenSSL library
EM Cartography - One active among the eight cores

Core activity is visible on the right, but the PMIC (Power Management Integrated Circuit), on the left side is a source of emission

This a small signal processing allow to identify the activity of the active core.

We noticed 2 clusters 0-3 and 4-7 that seems to operate simultaneously

Core 0, 3, and 6 have a stronger emission
Some results in OpenSSL context

❖ Mode of operations
❖ Interruptions
❖ Cache L1/L2
Mode of Operation Matters
Mode of Operation

ECB
Mode of Operation
CBC
At OS Level…
Interruptions, every 4ms
Memory Operations - Cache Effects
Cache Miss

- L2 miss: Reading in DRAM
- L1 miss: Reading in L2
L1 Miss Read in L2
But, ... cache activity (here L1) can be identified
Let's simplify the work, then scale-up ... may the Force be with us!
Classical Approach

Simplify but not too much

❖ Known Inputs or Output of targeted Algorithm
❖ Known Code
❖ Control the Core(s) where the AES is operated
❖ Add triggers to ease identification
Simple AES bare-metal
A Single AES 10 traces synchronized

Looks good!
Attack Result

Remaining Key Entropy evolution - Single Core
Pipeline Effect ?
Pipeline 3 AES bare-metal

```assembly
func decrypt_pipeliined
  ldr x3, =0xf9018810  // Pointer to input data
  ldi {v12.16b}, [x3], #16  // Loading from memory to NEON register
  ldi {v13.16b}, [x3], #16  // 3 blocks at a time
  ldi {v14.16b}, [x3], #16
  aesd v12.16b, v11.16b  // Round 0
  aesinc v12.16b, v12.16b
  aesd v13.16b, v11.16b
  aesinc v13.16b, v13.16b
  aesd v14.16b, v11.16b
  aesinc v14.16b, v14.16b
  .../
  aesd v12.16b, v3.16b
  aesinc v12.16b, v12.16b
  aesd v13.16b, v3.16b
  aesinc v13.16b, v13.16b
  aesd v14.16b, v3.16b
  aesinc v14.16b, v14.16b
  aesd v12.16b, v2.16b  // Last Round
  aesd v13.16b, v2.16b
  aesd v14.16b, v2.16b
  eor v12.16b, v12.16b, v1.16b
  eor v13.16b, v13.16b, v1.16b
  eor v14.16b, v14.16b, v1.16b
  ldr x3, =0xf9018810  // Reload pointer
  stl {v12.16b}, [x3], #16  // Storing result in-place
  stl {v13.16b}, [x3], #16
  stl {v14.16b}, [x3], #16
  ret
endfunc decrypt_pipeliined
```
Attack Result

Pipeline by 3 effect target the first core execution

Remaining Entropy evolution

- Naive exhaust
- Best exhaust
- SKEA exhaust
Attack Result

Pipeline by 3 effect, target the second core execution
Attack Result

Pipeline by 3 effect, target the third core execution
Attack Result

With the proper Selection Function the third AES can be more easy to break
Result

- OS: Capture between interruptions
- Cache: use cache activity as a helper, or remove bad traces
- Pipeline: Leverage architecture and code knowledge to define the proper Selection Function

- Literature says 4 Million trace to attack an A53 we render with less than 40k or 20k to get 32-bit remaining entropy and get the whole key with SKEA
Faults?
Another SoC to play with: iMX8 mini
Product description iMX8 mini

➔ Motherboard Technexion PICO-PI-8M PICO-IMX8M-

➔ Technology 14nm LPC FinFET

➔ SoC: NXP 4x Cortex-A53 core platforms up to 1.8GHz per core

➔ Caches
  ◆ 32kB L1-I Cache/ 32 kB L1-D Cache
  ◆ 512kB L2 Cache
Methodology

➔ Software stack analysis from OS to crypto routine call, understand how to reduce architecture effect.

➔ Dedicated software (bare-metal) for characterization.

➔ PhotoEmission to find area of activity and refine area of interest.

➔ Simulation vs IRL comparison to explain or anticipate fault and leakage model.

➔ Side-Channel - Signal analysis to adjust timing and refine trigger, shooting delay …
Area Of Interest from EM-SCA

I.MX8 Die

Area of interest
Area of Interest from Photonic Emission

**Principle** Capture lots of images and stack them up to get highlighted activity by image processing.

**Not interesting** Area of cores Interconnect, (buses, I/O,...)
Material for Laser Fault Injection
## Example of Fault Injection Parameter set

<table>
<thead>
<tr>
<th>EMFI Parameter</th>
<th>Range</th>
<th>Step (Coarse)</th>
<th>Step (Fine)</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>X pos</td>
<td>1000</td>
<td>2400</td>
<td>50 or 100</td>
<td>10 or 25</td>
</tr>
<tr>
<td>Y pos</td>
<td>2200</td>
<td>3400</td>
<td>50 or 100</td>
<td>10 or 25</td>
</tr>
<tr>
<td>Pulse amplitude</td>
<td>300</td>
<td>700</td>
<td>50 or 100</td>
<td>10</td>
</tr>
<tr>
<td>Pulse width</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Pulse delay</td>
<td>300</td>
<td>500</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Repeat</td>
<td>-</td>
<td>step</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LFI Parameter</th>
<th>Range</th>
<th>Step (Coarse)</th>
<th>Step (Fine)</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>X pos</td>
<td>10 or 6</td>
<td>3, 2, 1</td>
<td></td>
<td>μm</td>
</tr>
<tr>
<td>Y pos</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>μm</td>
</tr>
<tr>
<td>Optispot</td>
<td>0</td>
<td>1500</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>PDM amplitude</td>
<td>20</td>
<td>80</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Pulse width</td>
<td>100</td>
<td>1000</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>Pulse delay</td>
<td>6875</td>
<td>8875</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>Repeat</td>
<td>0</td>
<td>step</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
Fault Injection Exploitation

**Figure:** Pie Char of Faulty outcomes (no effect not represented)

**Figure:** Browse results according to parameters

**Figure:** Fault Visualizer Widget - Click on pts shows pie chart.

**Figure:** Fault effect, back tracking knowing the key
Thank you. Any questions?

- contact@eshard.com
- www.eshard.com
- /company/eshard
- @eshard

**France HQ**
Bâtiment GiENAH
11 avenue de Canteranne
33600 Pessac, France

**France R&D**
7 rue Gaston de Flotte
13012 Marseille, France

**Germany**
eShard GmbH
Beethovenallee 21
53173 Bonn