Laser Fault Injection on Mobile phone

FDTC – 25/09/2017

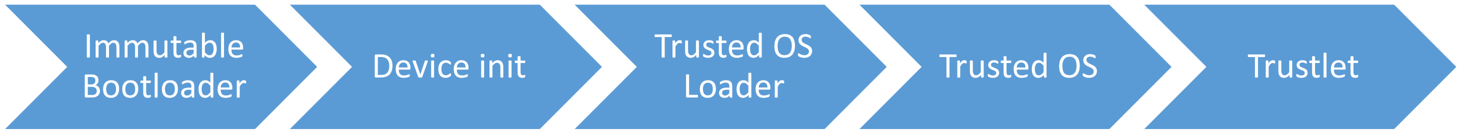
ALPhA NOV
Optics & Lasers Technology Center

eshard

How secure is a secure boot?

Secure Boot: TrustZone (TZ) & Trusted Execution Environment (TEE)

Chain of trust:



Can we target a <u>widespread device</u>, off the shelf
And <u>break its secure boot implementation </u>?

State of the Art

- Quarkslab 2017 : i.MX6 single core Cortex A9 – Dev. Board
    - Buffer overflow on X509 parser (CVE-2017-7932, -7936)
- Riscure : i.MX6 single core Cortex A9 – Dev. Board
    - EMFI (Thessalonikefs – 2014)
    - Voltage glitching (Timmers, Spruyt – 2012..2016)

→ Targeting Secure boot / TEE

- Laser practical results: not much on complex SoC or recent technology nodes
    - Importance of IR Drops on the Modeling of Laser-Induced Transient Faults - 2017

Real-world attack scope

Protagonist

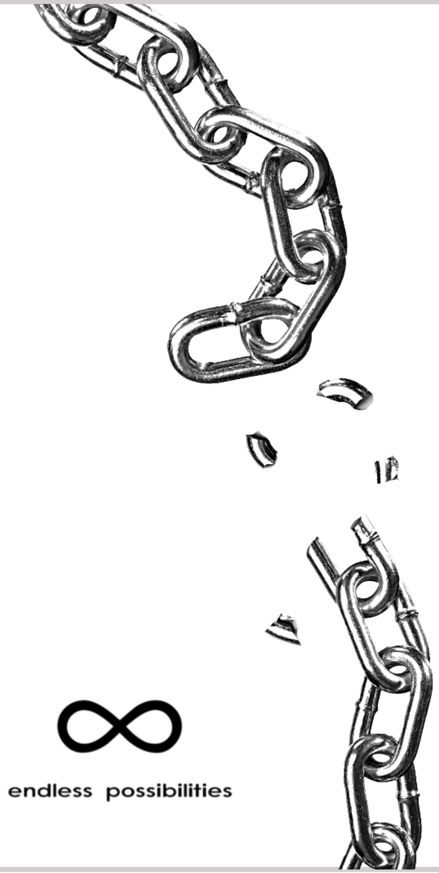
- Well equipped, unlimited samples
- **Undocumented**

Attack

- **Breaking** the chain of trust **only once**

Could lead to

- Reverse engineering : Opening doors
- Fault characterization : Injection possibilities
- Permanent privilege escalation : Signature forgery

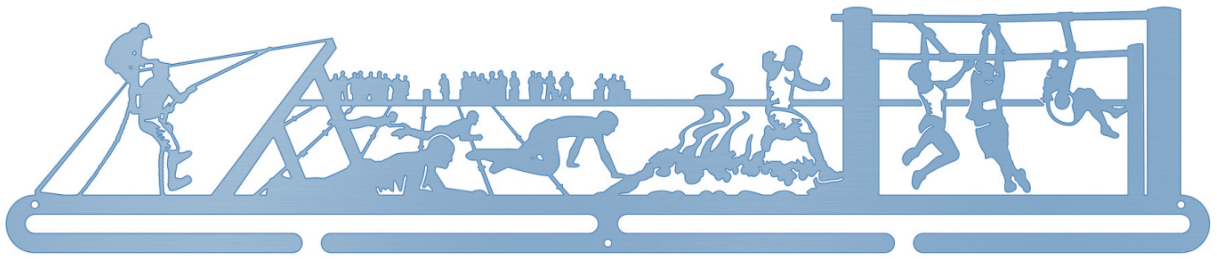


Starting Point: Facing reality

Access Information
External : Find documents
Internal : Reverse engineering

Access Debug Resources
Diversified HW/SW protections

Access Silicon
Product Integration



6 months

Reverse / Understand	Hack	Laser

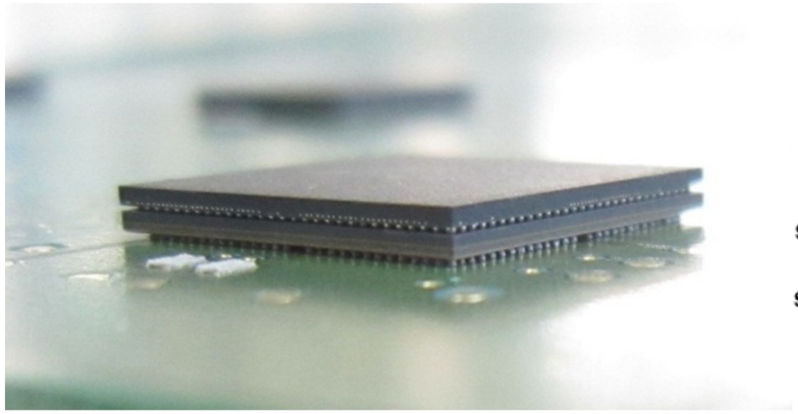
Laser campaign
Dedicated experiment

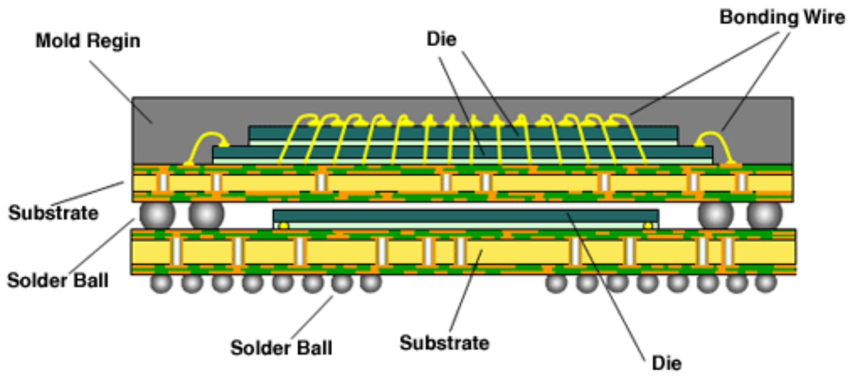
Understand Architecture
Process the gathered data

Understand Security
Cryptography, Protocol, Flaws

Embedded industry standard

- Package-on-Package intricacy





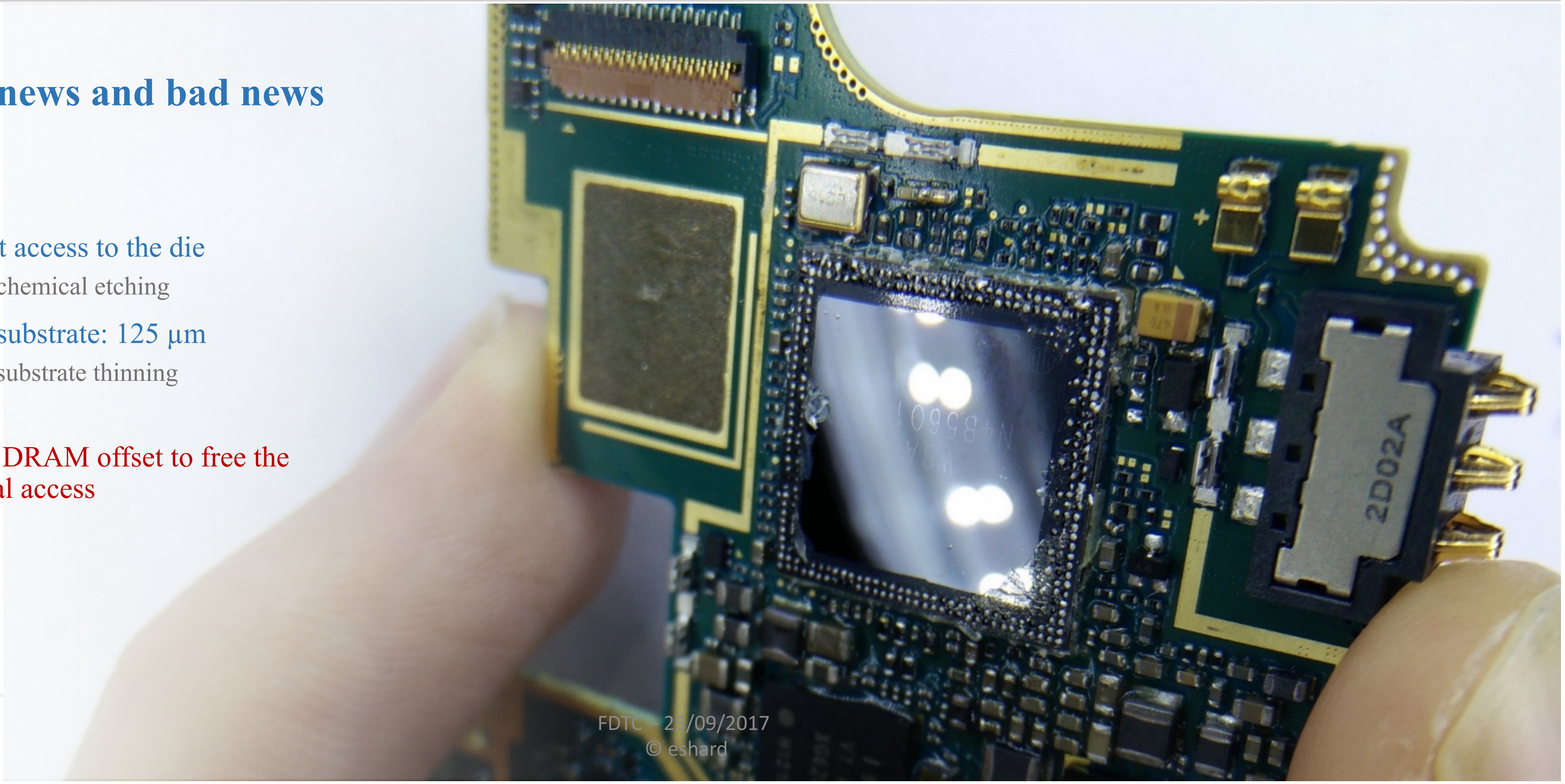
The attack will occur on back-side through the silicon substrate

PoP Opening

Good news and bad news

- Direct access to the die
  No chemical etching
- Thin substrate: 125 µm
  No substrate thinning

! - Need DRAM offset to free the optical access

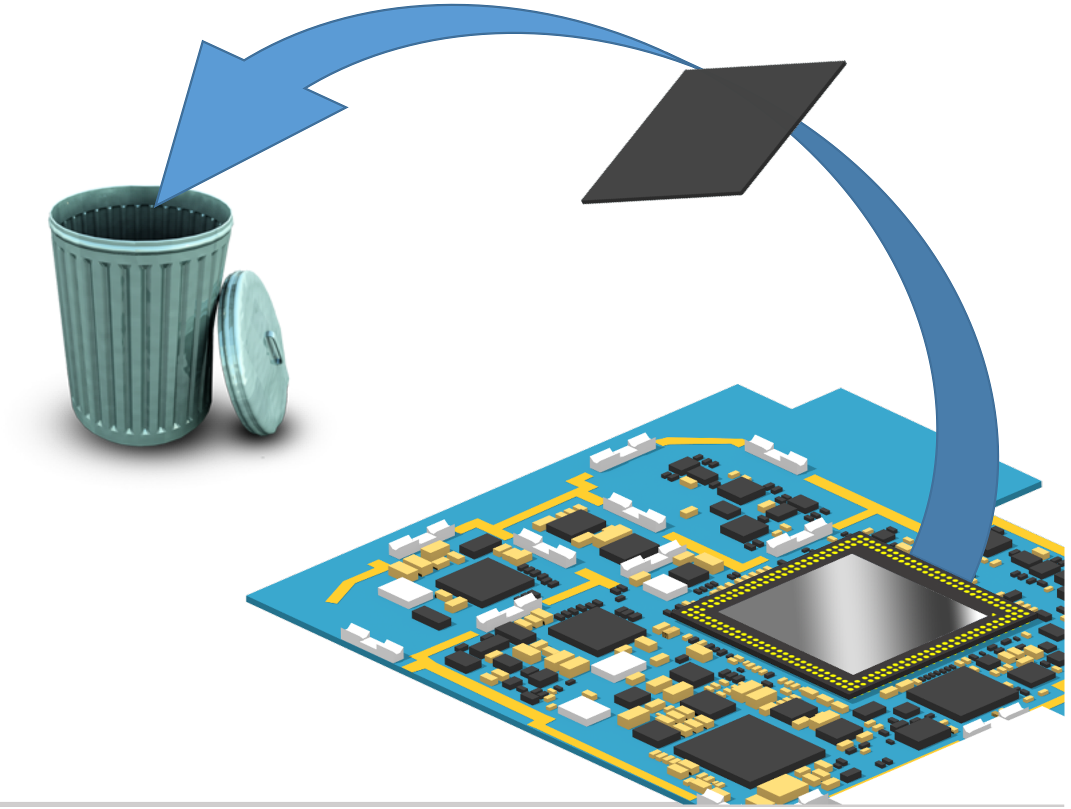


Solution: Removing DRAM

Relatively fast operation

Minimal cost

High chances of success

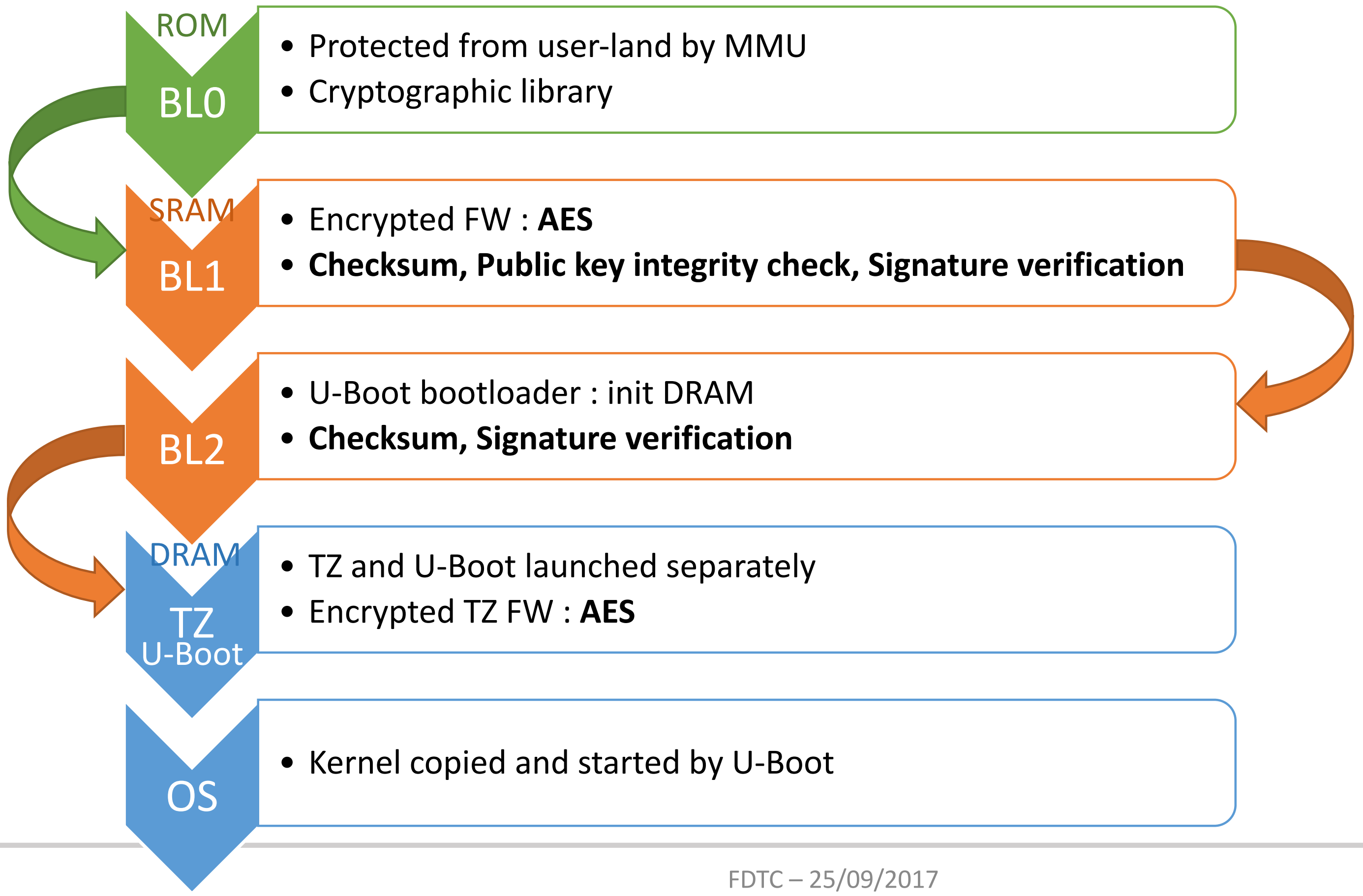


Incomplete boot – Partial solution

Require software exploits

Impossible to estimate the required time

Attack scenario unfolding



- Protected from user-land by MMU
- Cryptographic library

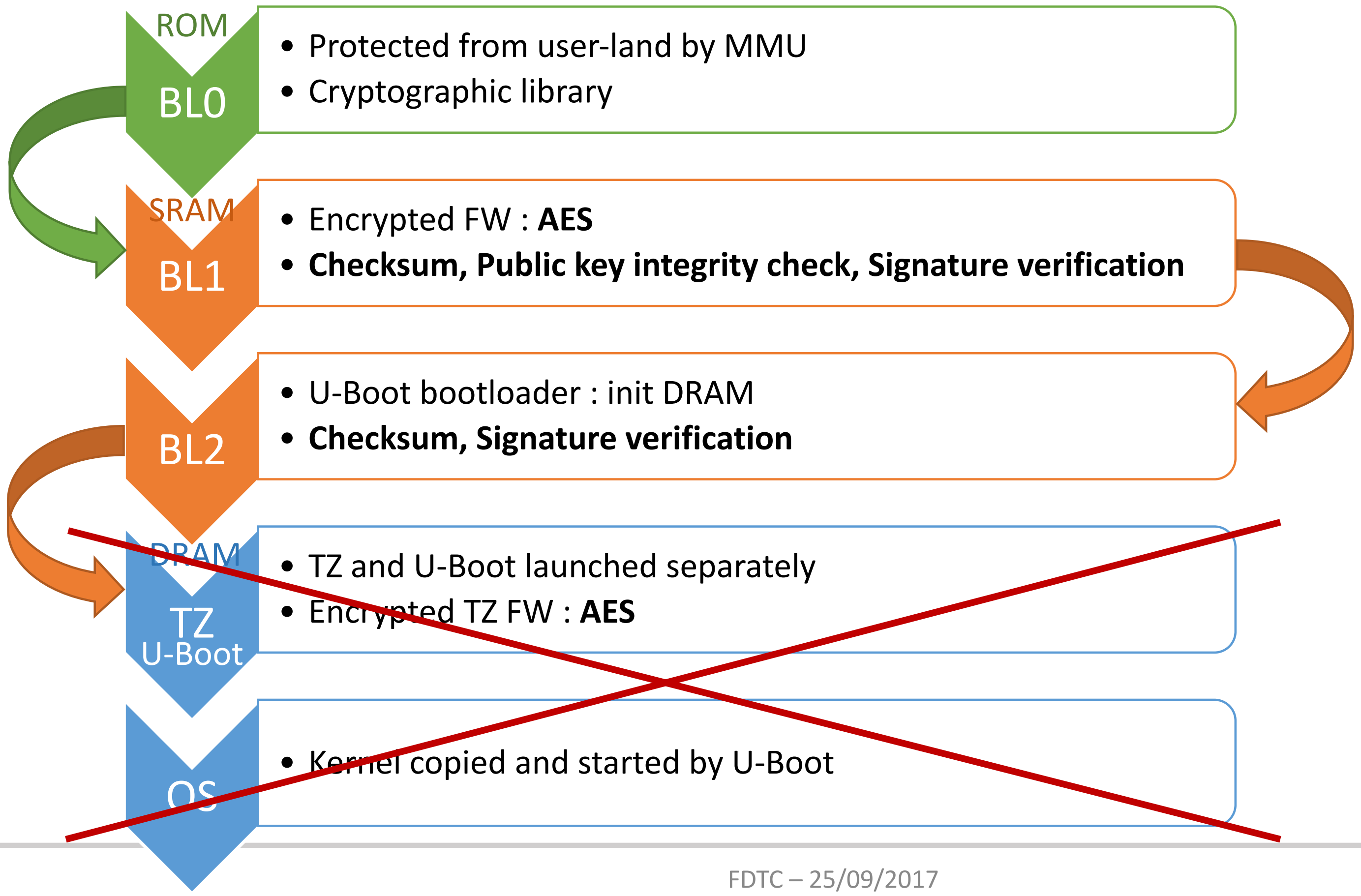
- Encrypted FW : **AES**
- **Checksum, Public key integrity check, Signature verification**

- U-Boot bootloader : init DRAM
- **Checksum, Signature verification**

- TZ and U-Boot launched separately
- Encrypted TZ FW : **AES**

- Kernel copied and started by U-Boot

Attack scenario unfolding



- Protected from user-land by MMU
- Cryptographic library

- Encrypted FW : **AES**
- **Checksum, Public key integrity check, Signature verification**

- U-Boot bootloader : init DRAM
- **Checksum, Signature verification**

- ~~TZ and U-Boot launched separately~~
- ~~Encrypted TZ FW : **AES**~~

- ~~Kernel copied and started by U-Boot~~

Avoid DRAM operations

Characteristics of the attack

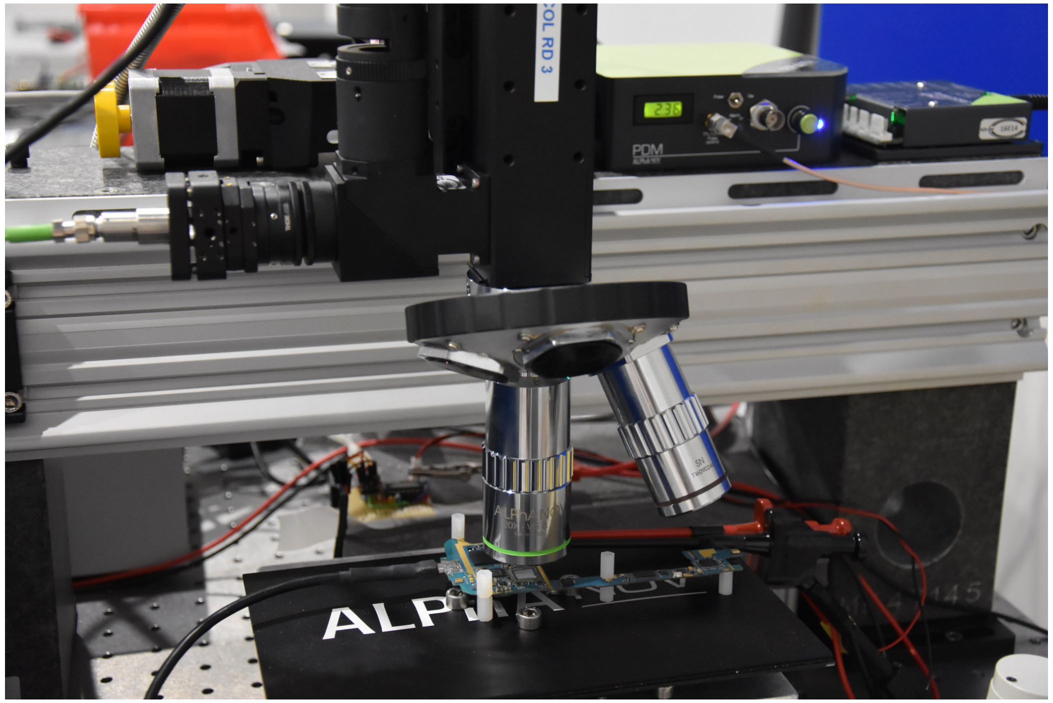
Spot source: 980nm

Spot power: 250mW

Sport duration: 100ns to 200ns

Spot size: 1 to 10 microns

Custom trigger generator



Target System on Chip

- On-die block analysis

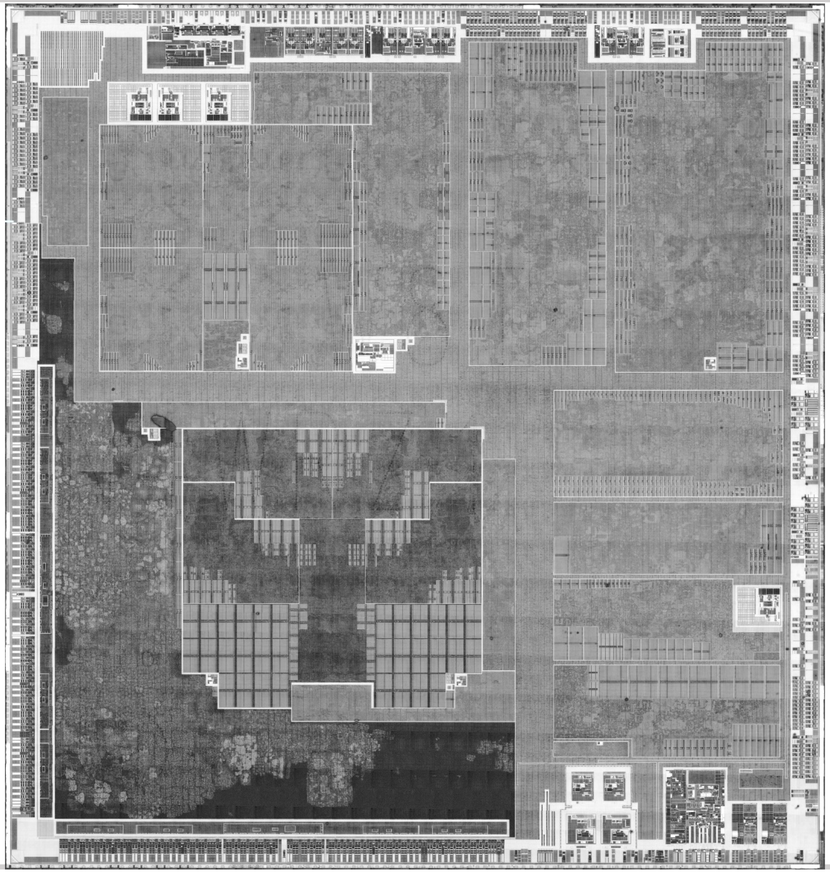
  32nm node – 1 x 1 cm

  Quad-core CorteA9 @ 1,4GHz – 0.71ns clock period
  8-Stage pipeline (variable length, speculative execution)

  27 million logic gates in CPU

  64KB ROM and 256 KB SRAM

  40+ integrated peripherals / interfaces



Light sensitive areas

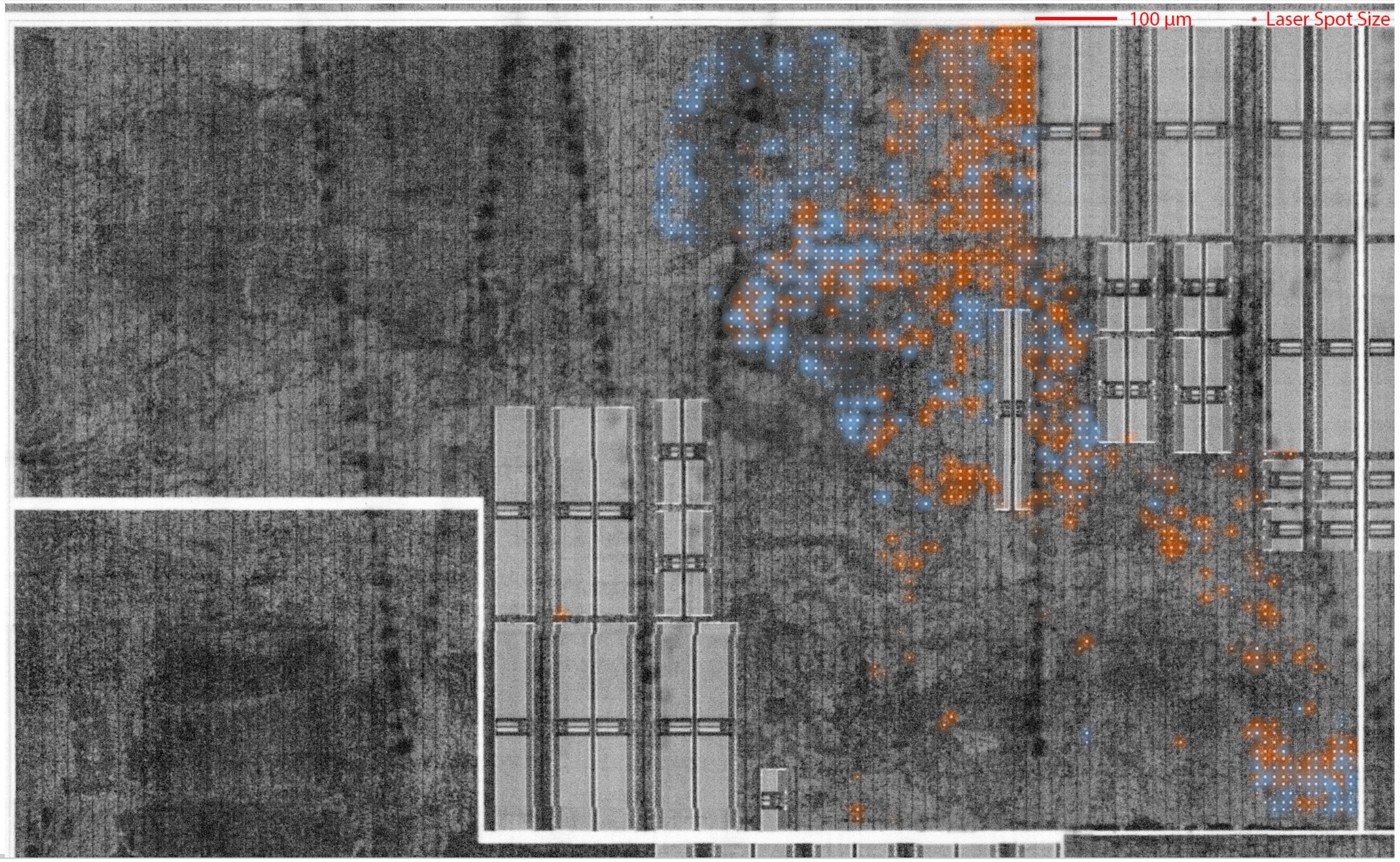
5 to 20% "faults" depending on the area and instructions used

50% crash

50% misbehavior

Sensible area mapping

CPU0 / HW IPs / Buses



Finding vulnerabilities

- Non resilient ROM code :

Timed fault attack

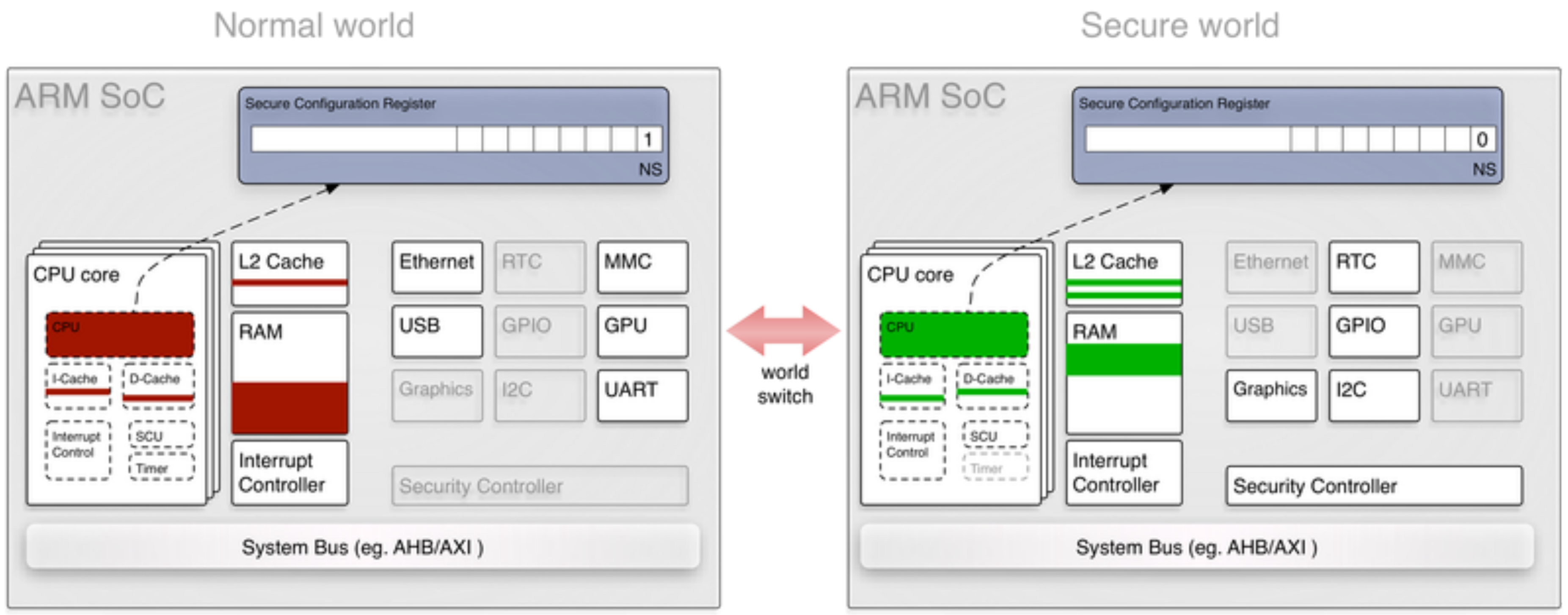
```
Init ( );

BL1_Verify_checksum ( );
If ( secure_boot == 1 ) {
        BL1_Verify_pubkey ( );
        BL1_Verify_signature ( );
        BL1_decrypt ( );
}
BL1_Jump ( );
```

Register ← secure_boot_flag

Register == 1 ✓

Hard to achieve in practice due to timing considerations

ARM Security architecture



State is hold by the **NS bit** inside each core's Secure Configuration Register (SCR)

Finding vulnerabilities

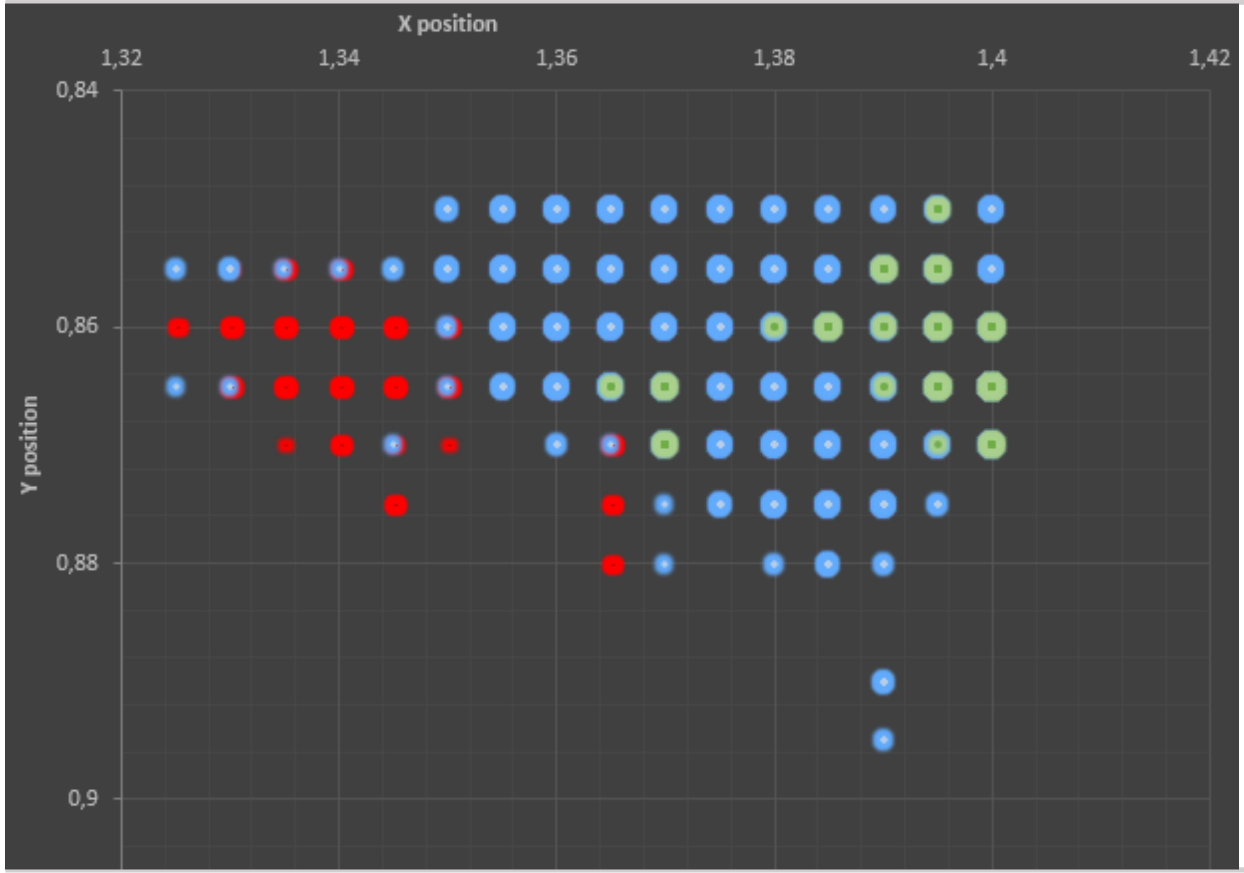
- Non robust implementation: Static fault attack

Simple scenario: Direct state change – Fault Injection on SCR bits

Is there anything behind the specs more than

NS = 0 or NS = 1 ?

Zoom on Fault Effects



Very high reproducibility : >90%

Crash

Non Secure → Secure

Other SCR bit flips

Mitigations

ALPhA NOV Optics & Lasers Technology Center

eshard

- HW Level protections

Fault detection or filtering	!
Redundancy	✓
Error correcting codes	✓

- SW Level protections

Attack surface is already limited	✓
Software redundancy, Resilience	✗
Fixing more complex attack scenarios	✗

Realisation remains challenging → Simple protections should be enough

Can only be confirmed by a **security evaluation**

Conclusion

ALPhA NOV Optics & Lasers Technology Center

eshard

Achieved goals

Reverse engineering Fault characterization Privilege escalation

Risk assessment

High-level attack with several critical steps

Mono-bit fault is possible on complex hardware

Static fault injection is more likely to succeed (simplicity, efficiency)

Complexity != Security

Practical feasibility is proven. Exploitation remains.

Optics & Lasers Technology Center