Who we are

Eshard - Embedded Security Company

- Software & Hardware Security

What do we do:

- Tools: Side Channel / Code Analysis
- Consultancy
- Audit
- Training

@eshardNews

https://www.eshard.com

contact@eshard.com

Bordeaux + Marseille

Attack TrustZone with Rowhammer

GreHack 2017

Pierre Carru - eshard
pierre.carru@eshard.com

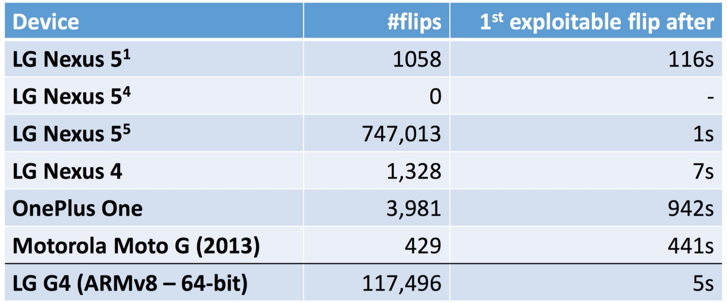
Presentation Summary

$$\left.\begin{array}{l}\text{1. } \textbf{Rowhammer} \rightarrow \text{Corrupt Mem} \\ \text{2. } \textbf{TrustZone} \rightarrow \text{Secure enclave} \\ \text{3. } \textbf{Attack}\text{: Corrupt TrustZone Mem}\end{array}\right\} \sim 25 \text{ min}$$

4. Questions

Context - Existing Rowhammer-based Exploitations

- **Intel**
  - clflush instruction (2014 original attack)
  - cache eviction (rowhammer.js 2015)
  - non-temporal instructions (2016)
  - one location hammering (few days ago)
- **Mobile** (arm):
  no direct way for unprivileged user
  - Drammer (end 2016)
    uses uncached memory region
    → exploit gains root privilege
  - No cache eviction method working yet
    → not enough access/second (yet)?



Device	#flips	1st exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

Context - Existing TrustZone Attacks

- **Software Bugs** in Qualcomm's TEE, and Widevine TA:
  - Dan Rosenberg (2014):
    Integer overflow
    No exploitation
  - Gal Beniamini (2015 - 2016):
    1. Missing parameter validation in Secure Kernel Call
       → Shellcode execution in Secure Kernel
    2. Buffer Overflow in Widevine TA
       → Shellcode execution in TA, and then in Secure Kernel

⇒ **Few TrustZone Attack**

- CLKSCREW (Tang 2017):
  Faults in microarchitecture using frequency and voltage scaling
  → Retrieve private key, Load self-signed TA
- Other ARM Plaforms: undisclosed / unknown?

eshard

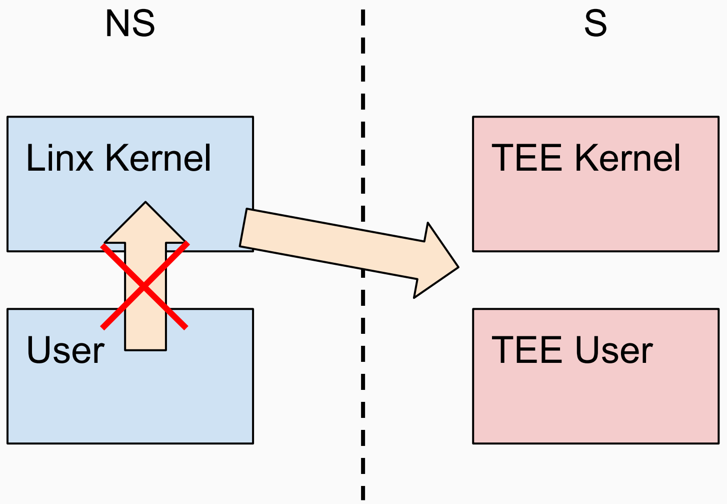
Rowhammer attack against TrustZone

Assumption:

- Rowhammer vulnerable device
- Kernel Privilege in Normal OS

Objectives:

- Corrupt Memory marked Secure
- If possible, exploit corruptions in order to gain more privileges



We focus on the Secure / Non-Secure border

→ **We use maximum privilege in Non-Secure Side**

Realization - Example Exploitation

Platform: Any Cortex-A based ARM Development board with TrustZone Support

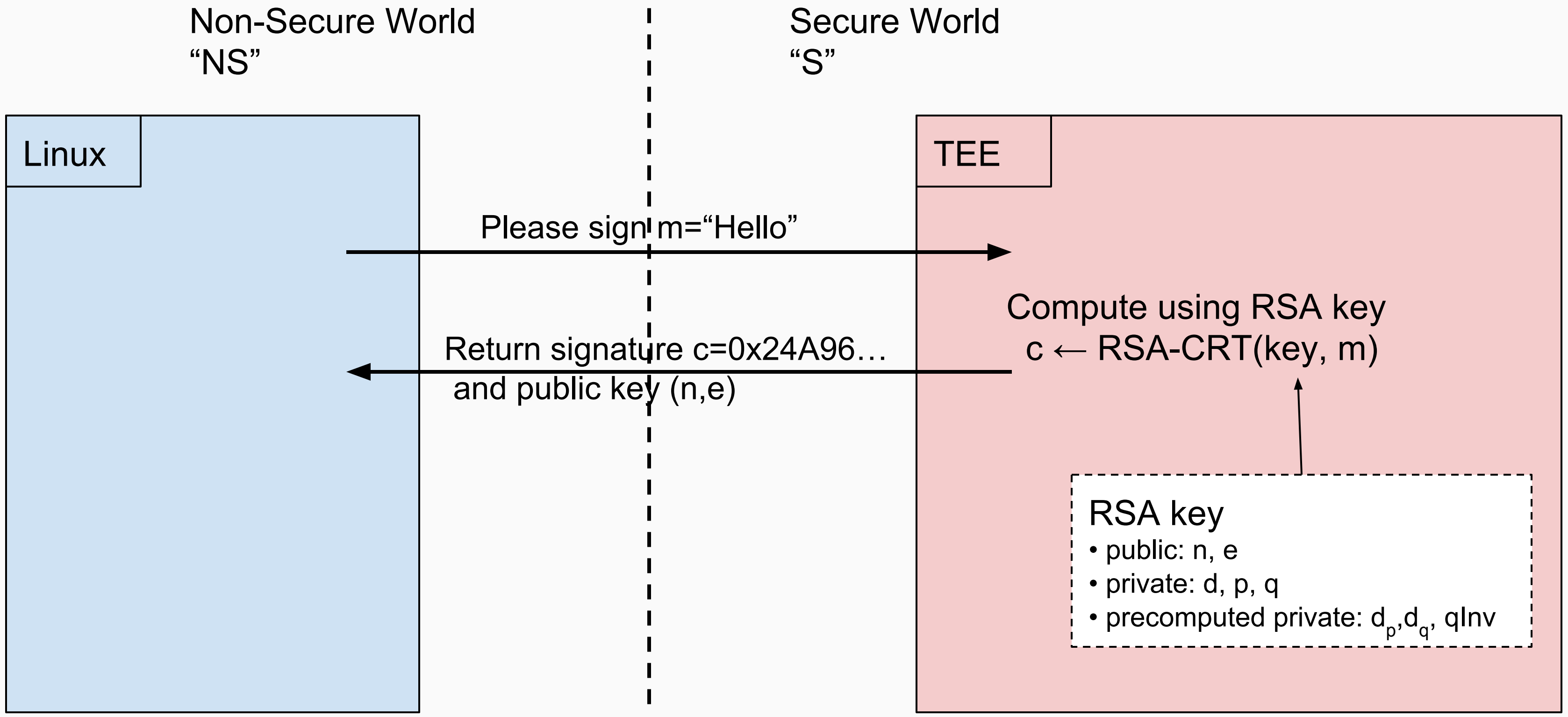
- Linux in Non-Secure Side
- Custom Trusty based TEE

PoC attack:

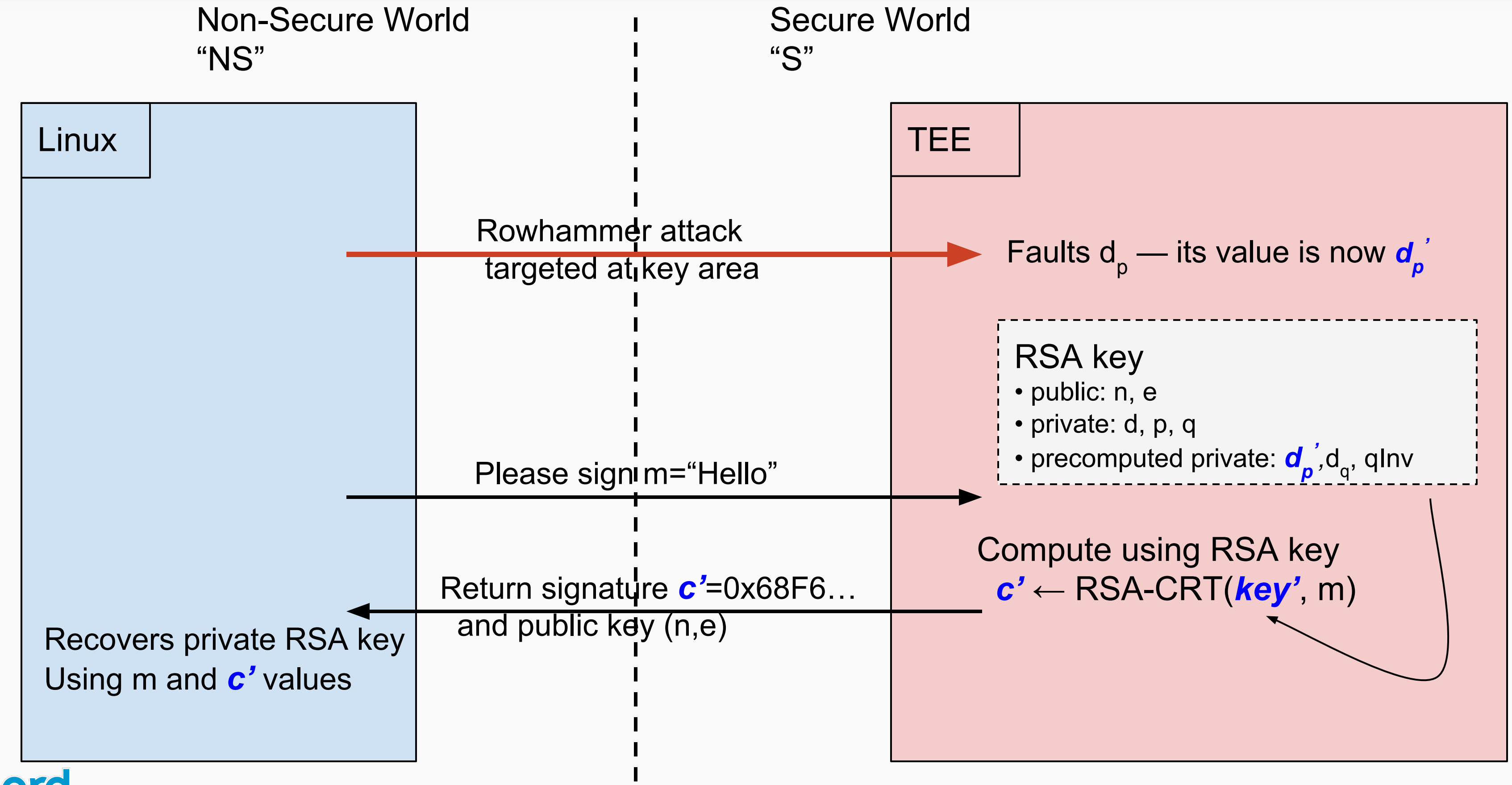
1. TEE provides an RSA-CRT signing mechanism
2. Secret Key stored in S Memory
3. Linux uses Rowhammer to fault the Secret Key (crosses the TrustZone border)
4. Linux uses faulty signature to recover Secret Key “Bellcore”

(*Boneh, DeMillo, Lipton*)

Exploitation Principle (1)

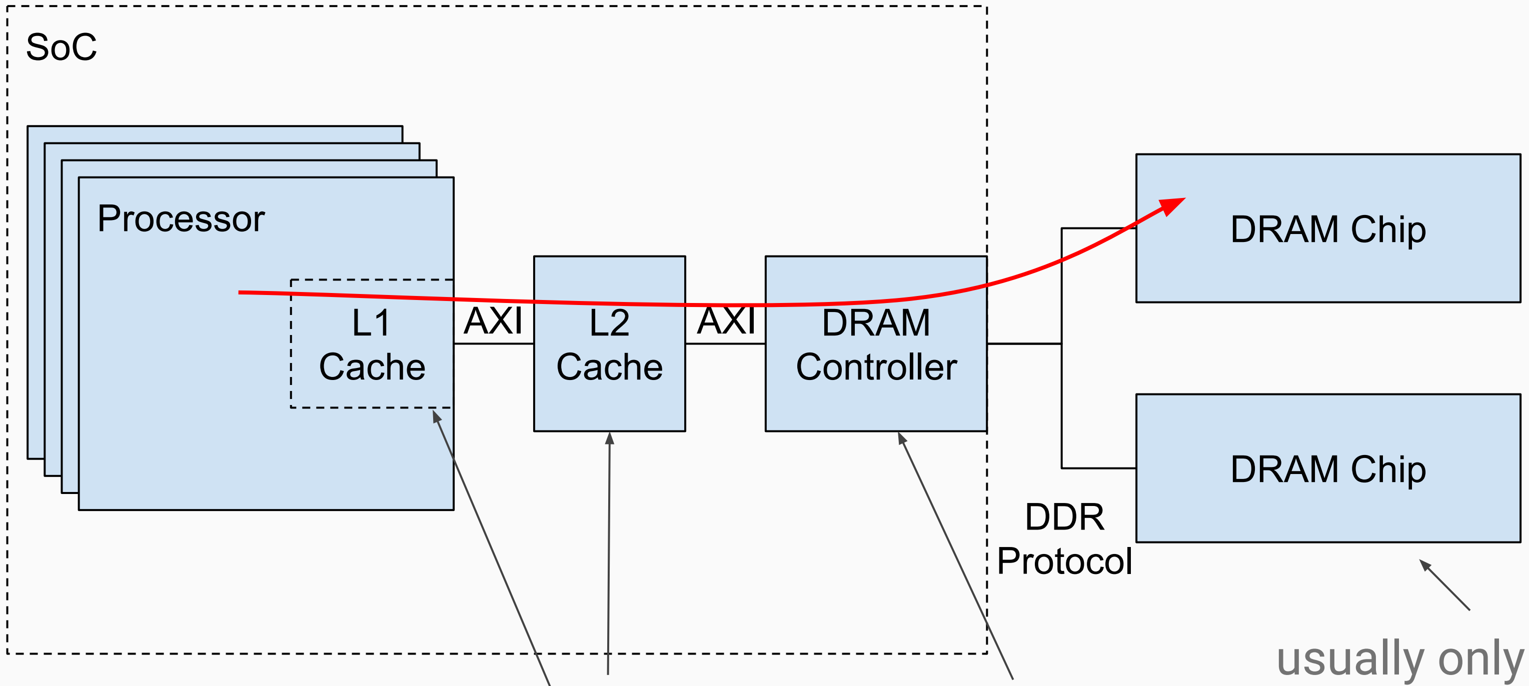


Exploitation Principle (2)


Non-Secure World
“NS”
Secure World
“S”
Linux
TEE
Rowhammer attack targeted at key area
Faults d_p — its value is now d_p'
RSA key
• public: n, e
• private: d, p, q
• precomputed private: d_p', d_q, qInv
Please sign m=“Hello”
Compute using RSA key
$c' \leftarrow$ RSA-CRT(key', m)
Return signature c'=0x68F6… and public key (n,e)
Recovers private RSA key Using m and c' values

Rowhammer

System Architecture



Need to Go through the caches!

May reorder accesses

usually only 1 PoP on mobile

How to generate faults in DRAM

DRAM Storage Cell

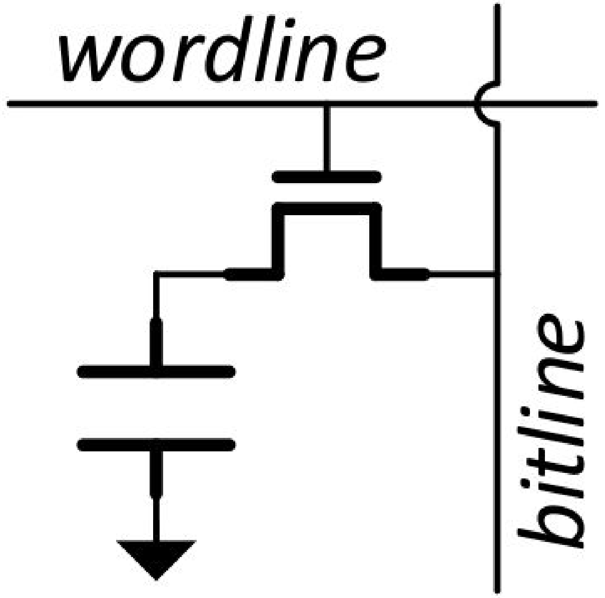
Capacitor as storage mechanism

Capacitor either:

- charged → logic 1
- discharged → logic 0

Capacitors lose their charge over time

⇒ have to be recharged periodically
“refreshed”



A DRAM Chip contains multiple Banks

Usually on Mobile:

1 PoP LPDDR3/4 Chip

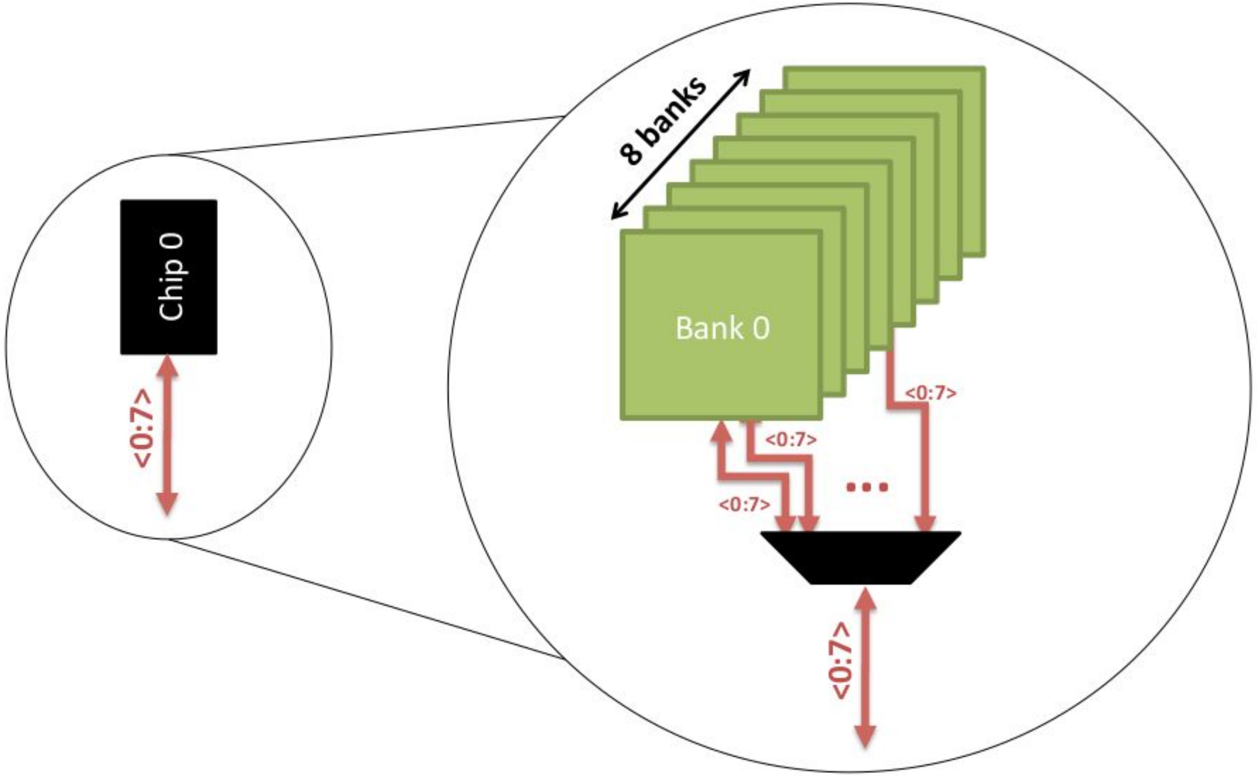


Image: Onur Mutlu

x8 DRAM Bank

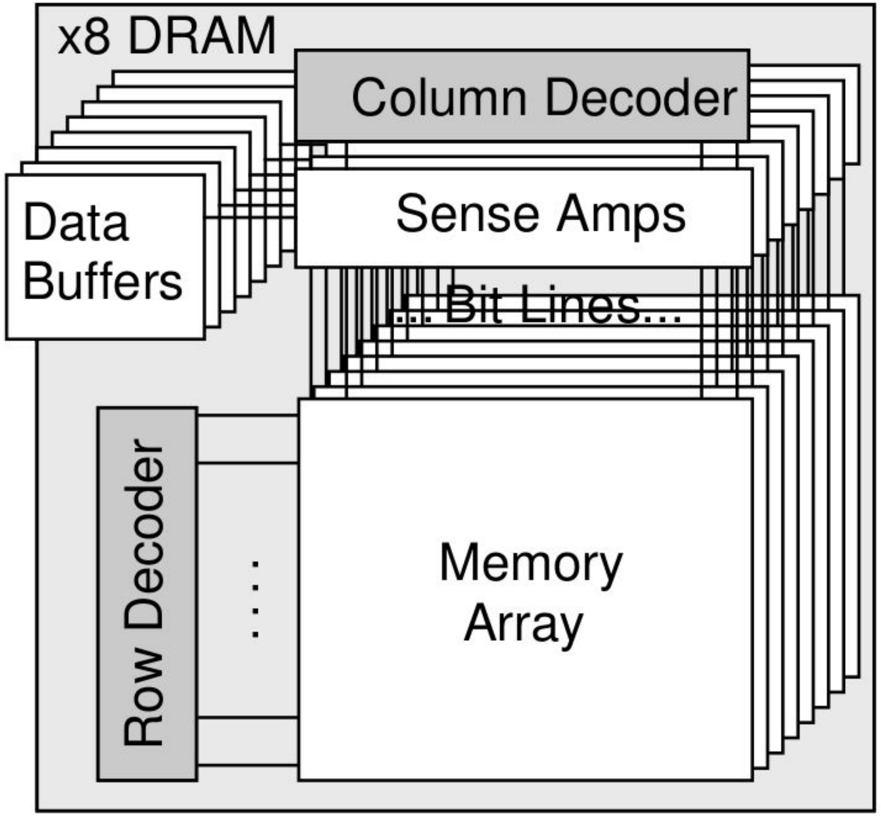
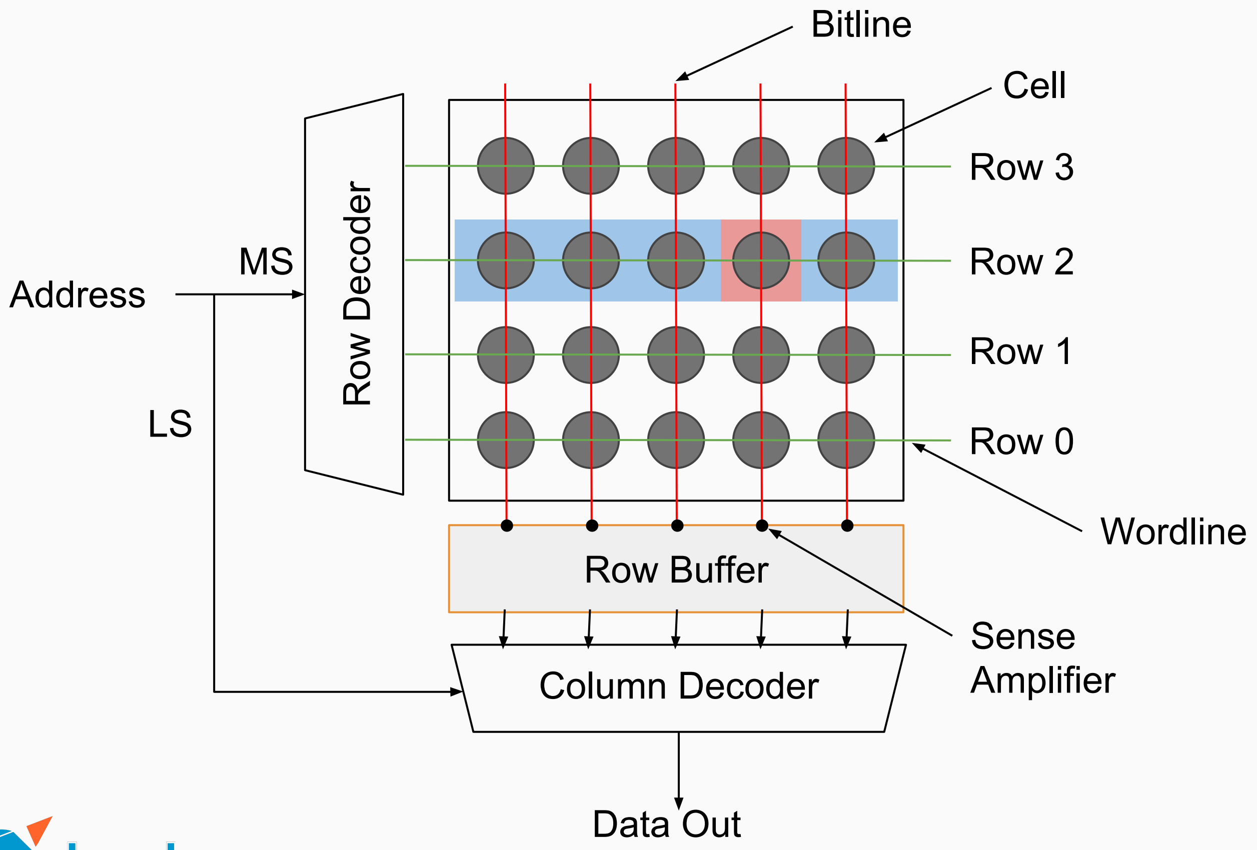


Image: Memory Systems - Cache, DRAM, Disk

DRAM Array



ACTIVATE

Open Row
→ Row Buffer

READ/WRITE

R or W Column
(in buffer)

PRECHARGE

Close Row

REFRESH

Row Access

Access to an opened row:

- No need to ACTIVATE
- Just READ/WRITE to access row buffer

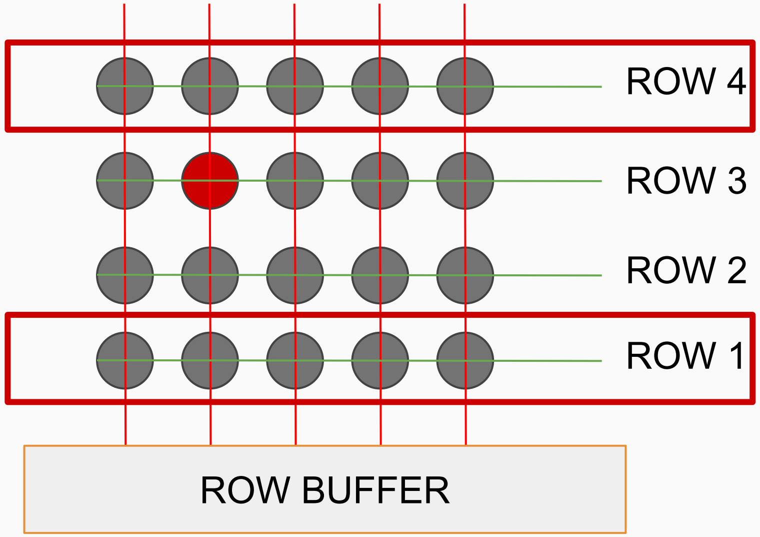
Access to a closed row:

- PRECHARGE current row
- ACTIVATE new row
- READ/WRITE

Simple-Sided Rowhammer

Need to ACTIVATE two distinct Rows in the same Array

Because accessing the same Row consecutively ⇒ hit the row buffer



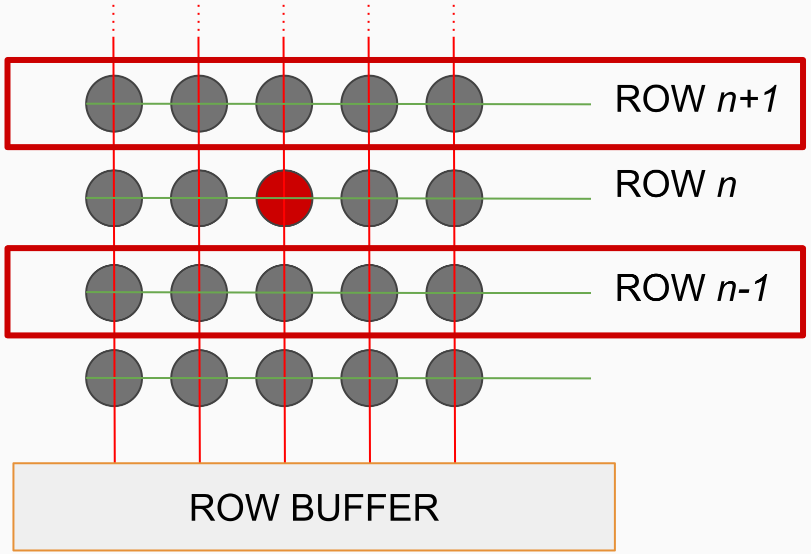
May generate

0 → 1 or 1 → 0 flips

Double-Sided Rowhammer

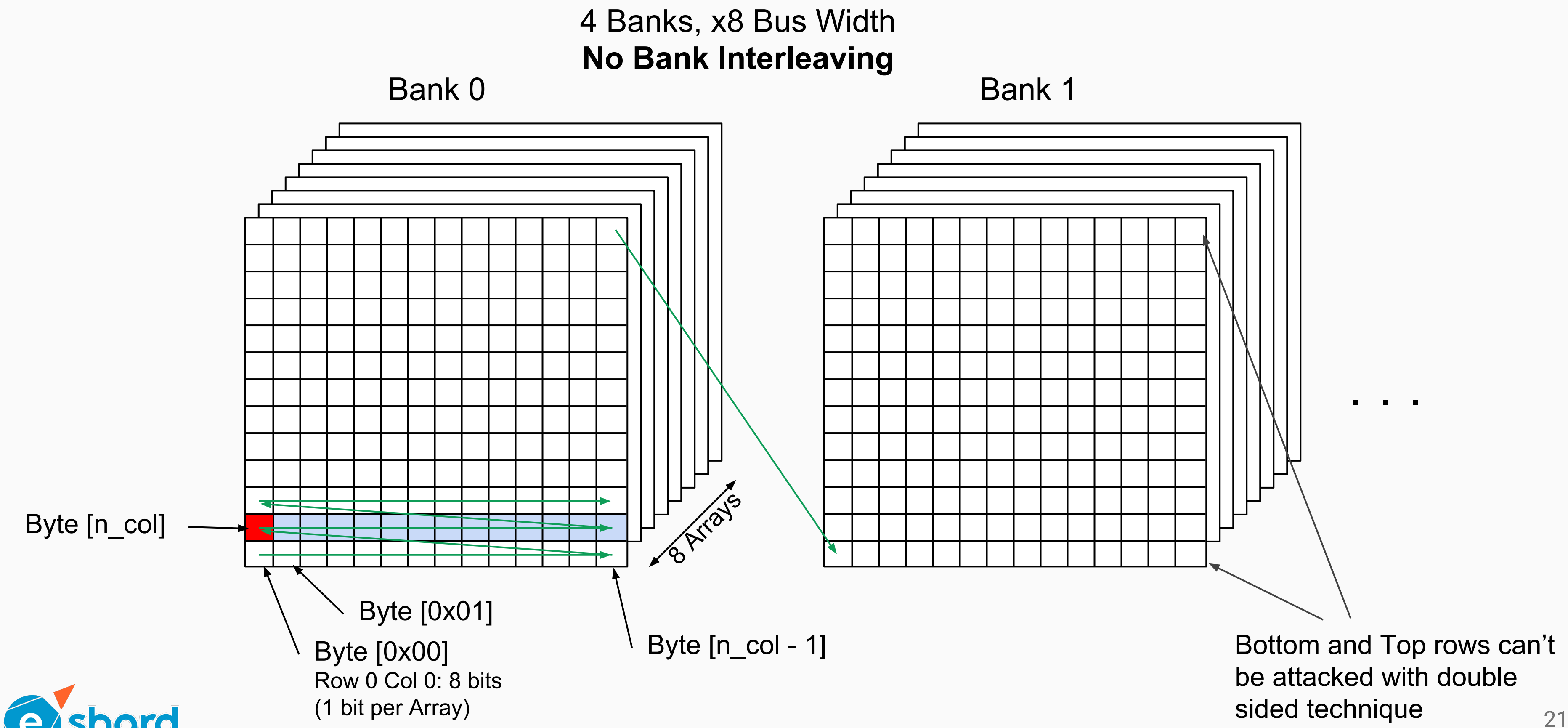
Hammer rows adjacent to the target Row → generates more flips

Flips are reproducible on
 a particular RAM chip
→ due to manufacturing?

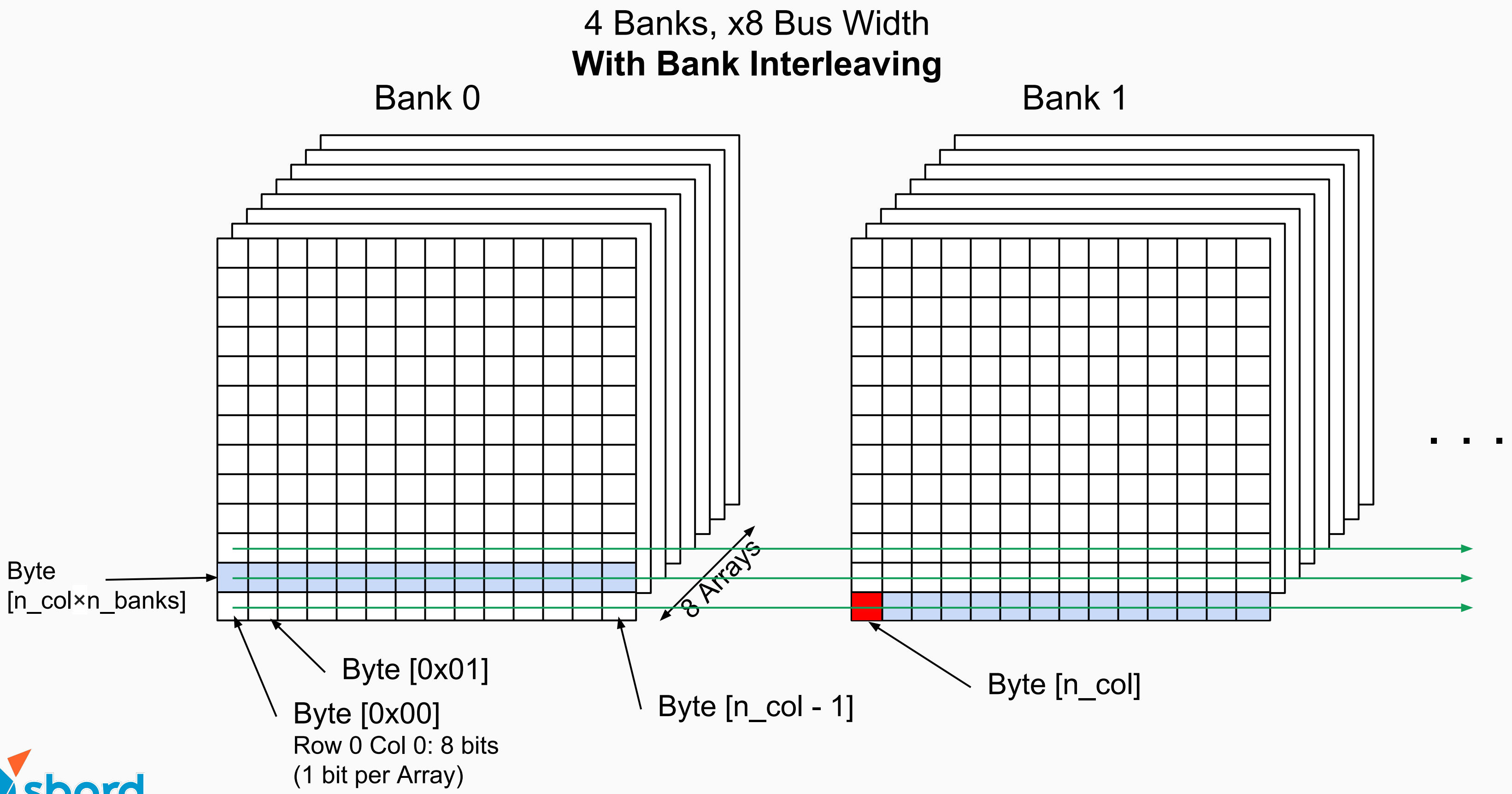


How to address rows from CPU

Memory Mapping - How to address adjacent rows (1)



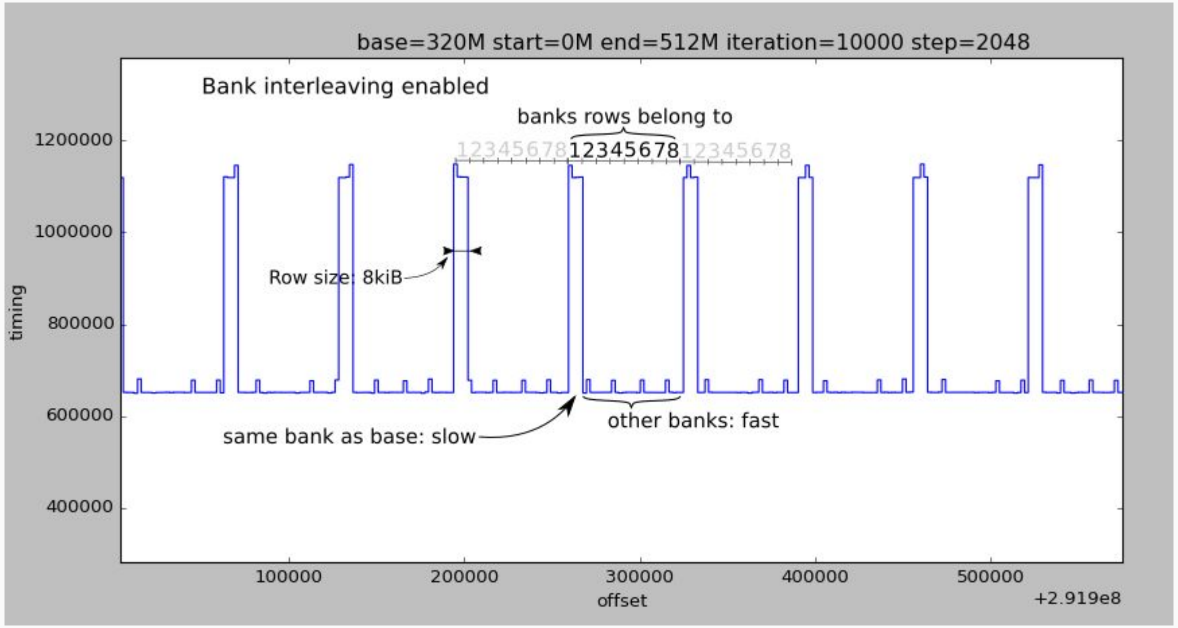
Memory Mapping - How to address adjacent rows (2)



Deduce Memory Characterics & Configuration using Timing Characterization

Pseudo code (simplified)

```
base = 0x…;
for (i = start;
     i < end;
     i += step) {
  ts = start()
  read_at(base)
  read_at(i)
  time[i] = end(ts)
}
```



Can be crossed checked with datasheets if DRAM Chip is identified

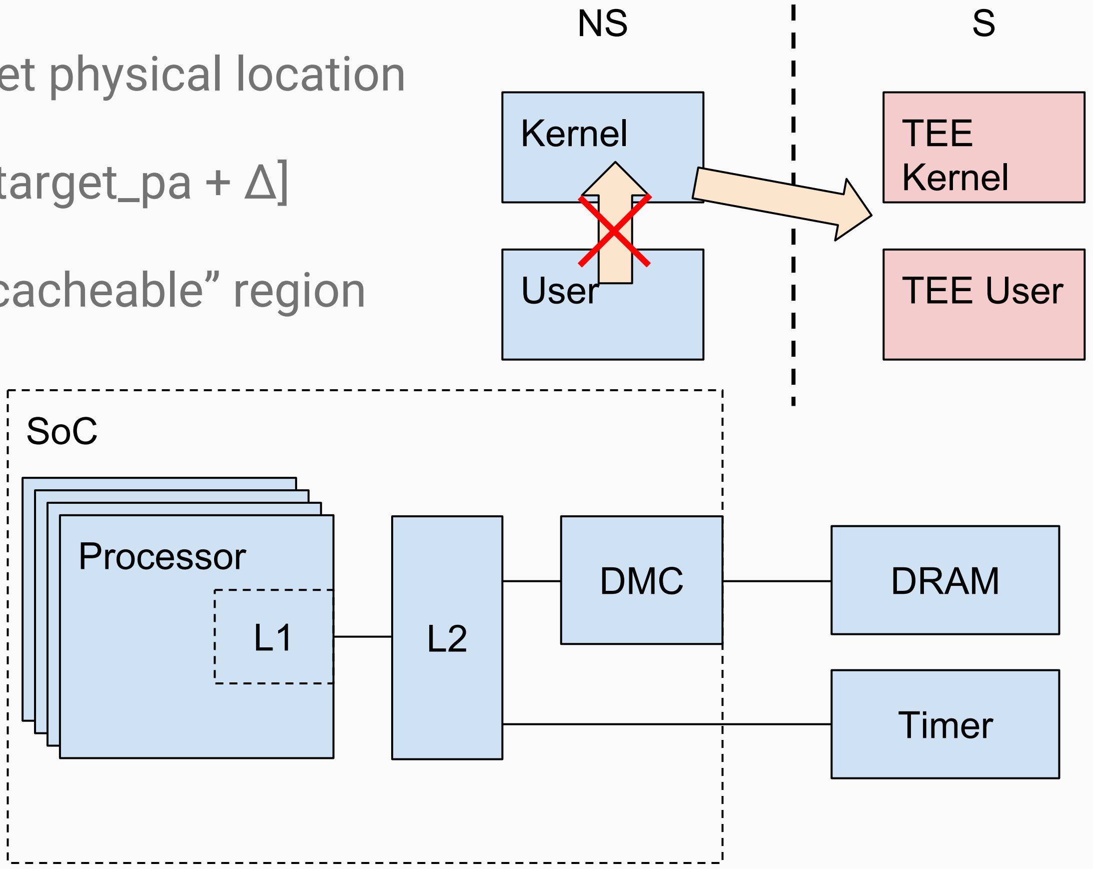
Rowhammer Implementation (2)

Need to map region around target physical location

→ `ioremap` [target_pa - Δ, target_pa + Δ]

Need to bypass the caches: “uncacheable” region

→ `ioremap_nocache`



Rowhammer Implementation (1)

In Kernel Module for simplicity

Code Simplified:

```
/* row before */
addrs[0] = target_va - (mem->n_banks * mem->row_size);
/* row after */
addrs[1] = target_va + (mem->n_banks * mem->row_size);

for (int j = 0; j < iterations; j++) {
    *row_before = pattern; /* write or read */
    *row_after  = pattern;
}
```

TrustZone

TrustZone Rationale

Want:

- Secure processor runs OS with manageable Security
  ≠ Android
- Some hardware resources only accessible to Secure OS

Do not want to:

- Waste silicon space on separate processor
- Hardware duplication

→ TrustZone:

- Time sharing of processor, ≈ **virtually 2 distinct processors**
- Some resources available only to the Secure processor

System Bus - AXI

Masters:

- Read from slaves
- Write to slaves

TrustZone Introduces a new transaction attribute:

NS ∈ {0, 1}

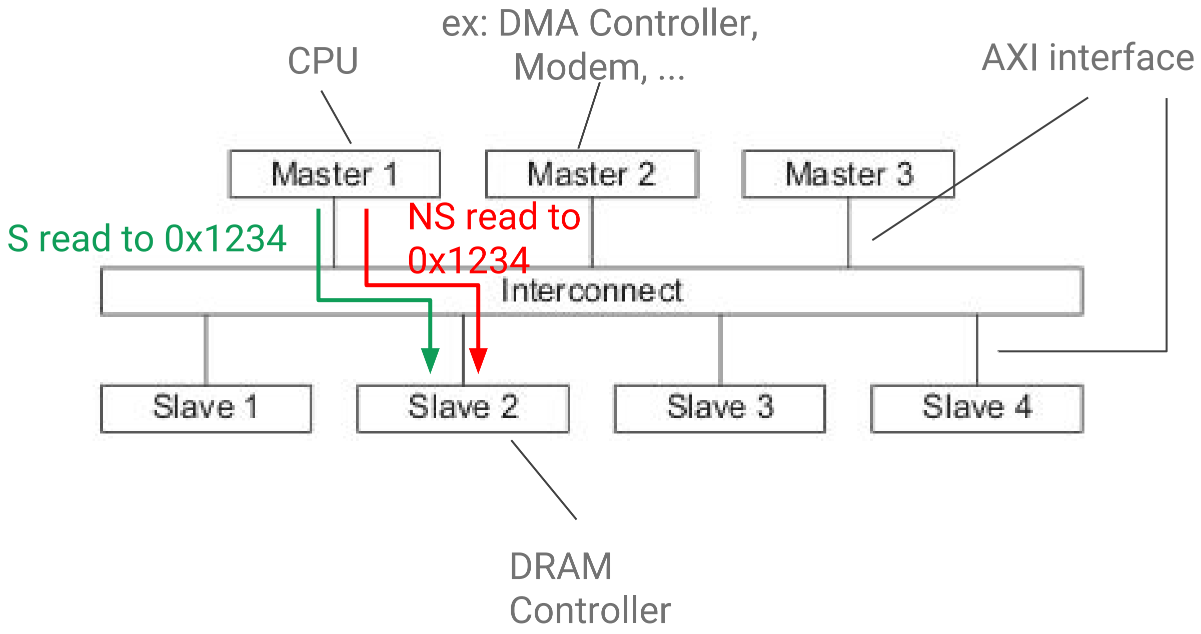


Image: ARM

Adaptations to IPs

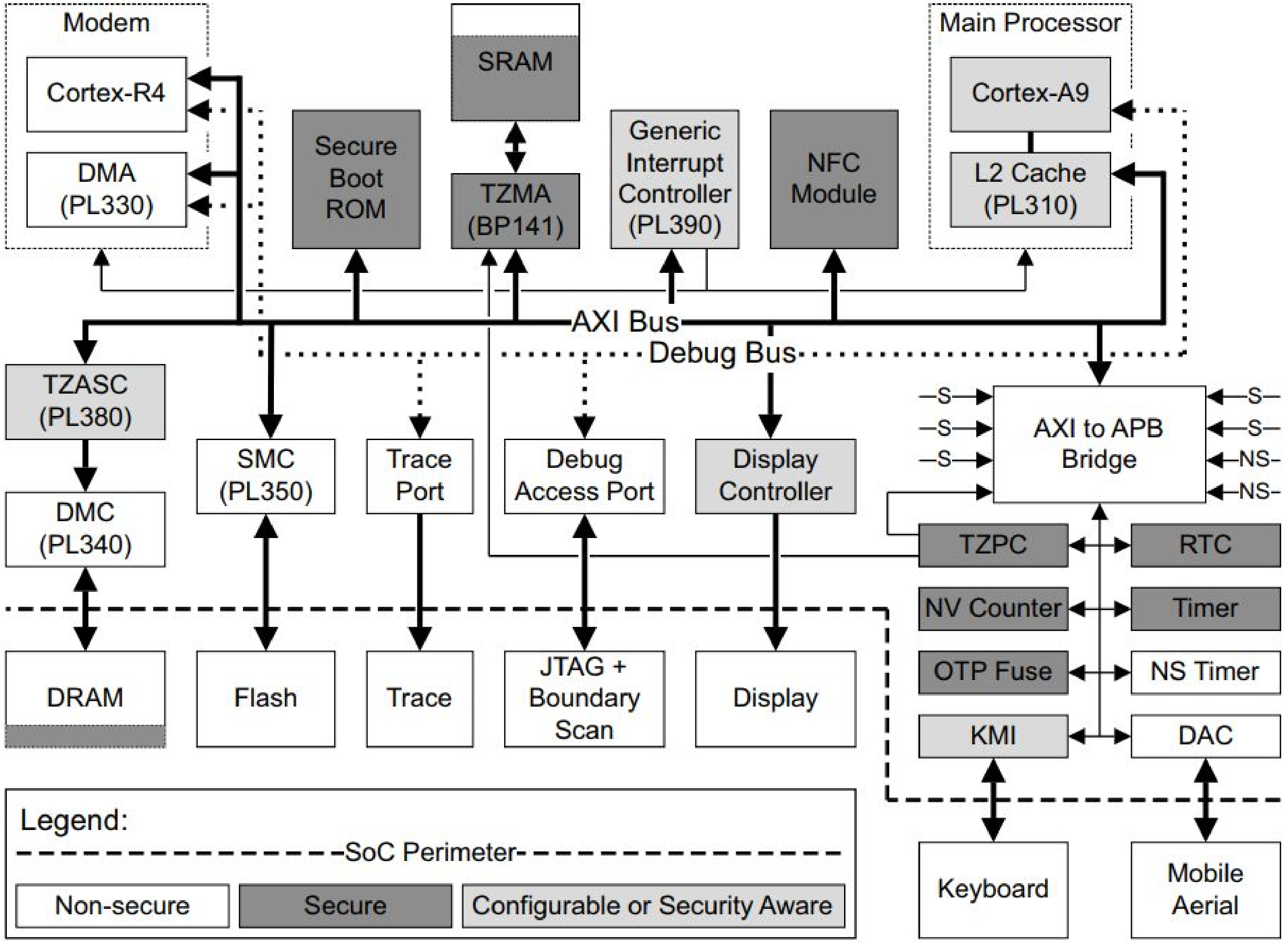
AXI slave responsible to enforce S/NS logic

- L1, L2 Caches
- **Memory controller**
- Touchscreen
- DMA controller
- MMU
- Interrupt controller
- …

Existing devices can be modified to become aware of TrustZone

Or an extra adapter IP can wrap a device to provide S/NS logic

ARM Gadget2008



ARM Procesor Architecture extensions

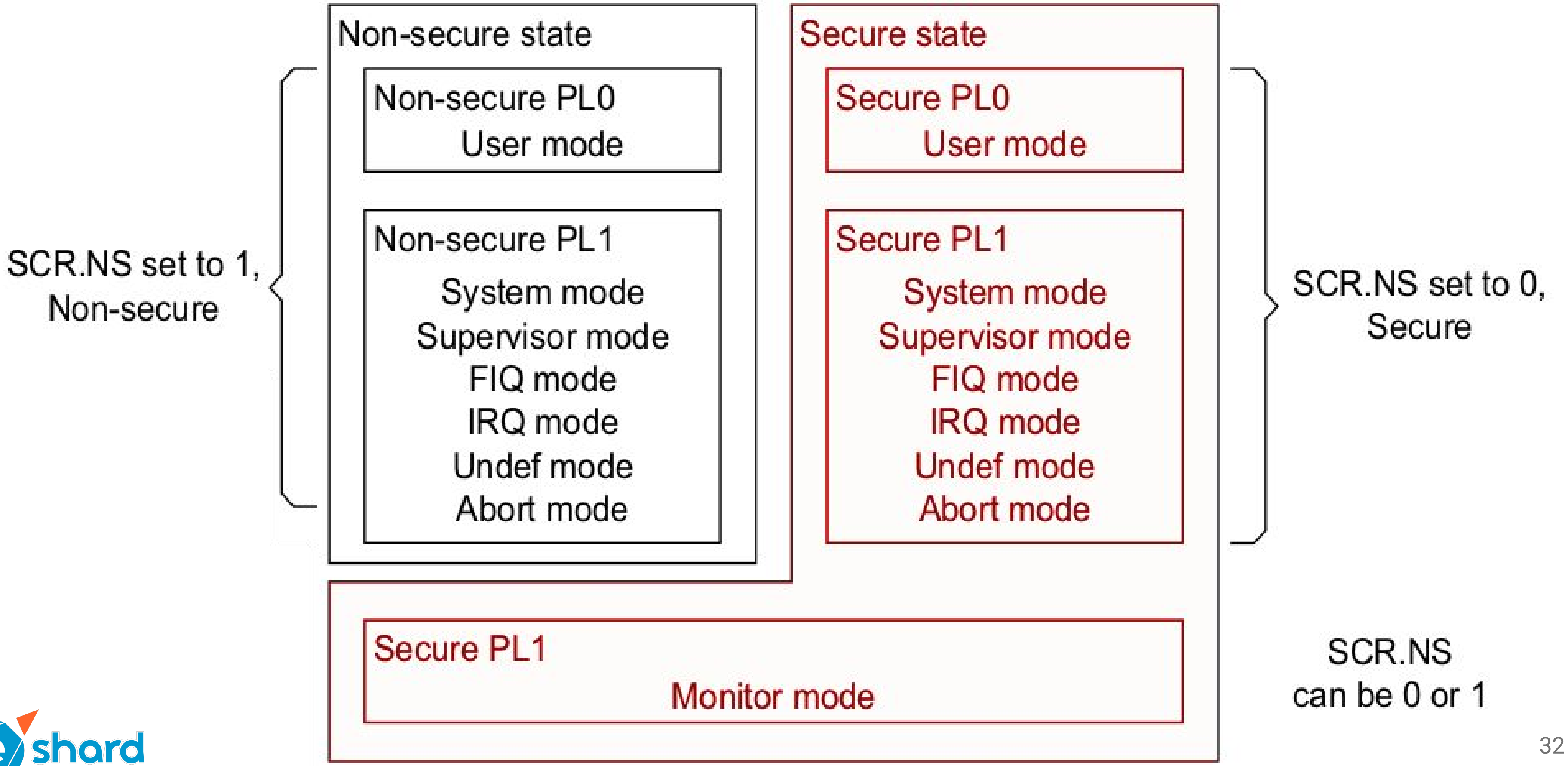
Principles:

Only “secure software” can make S transactions.
NS OS calls “secure software” which checks if call request is legal

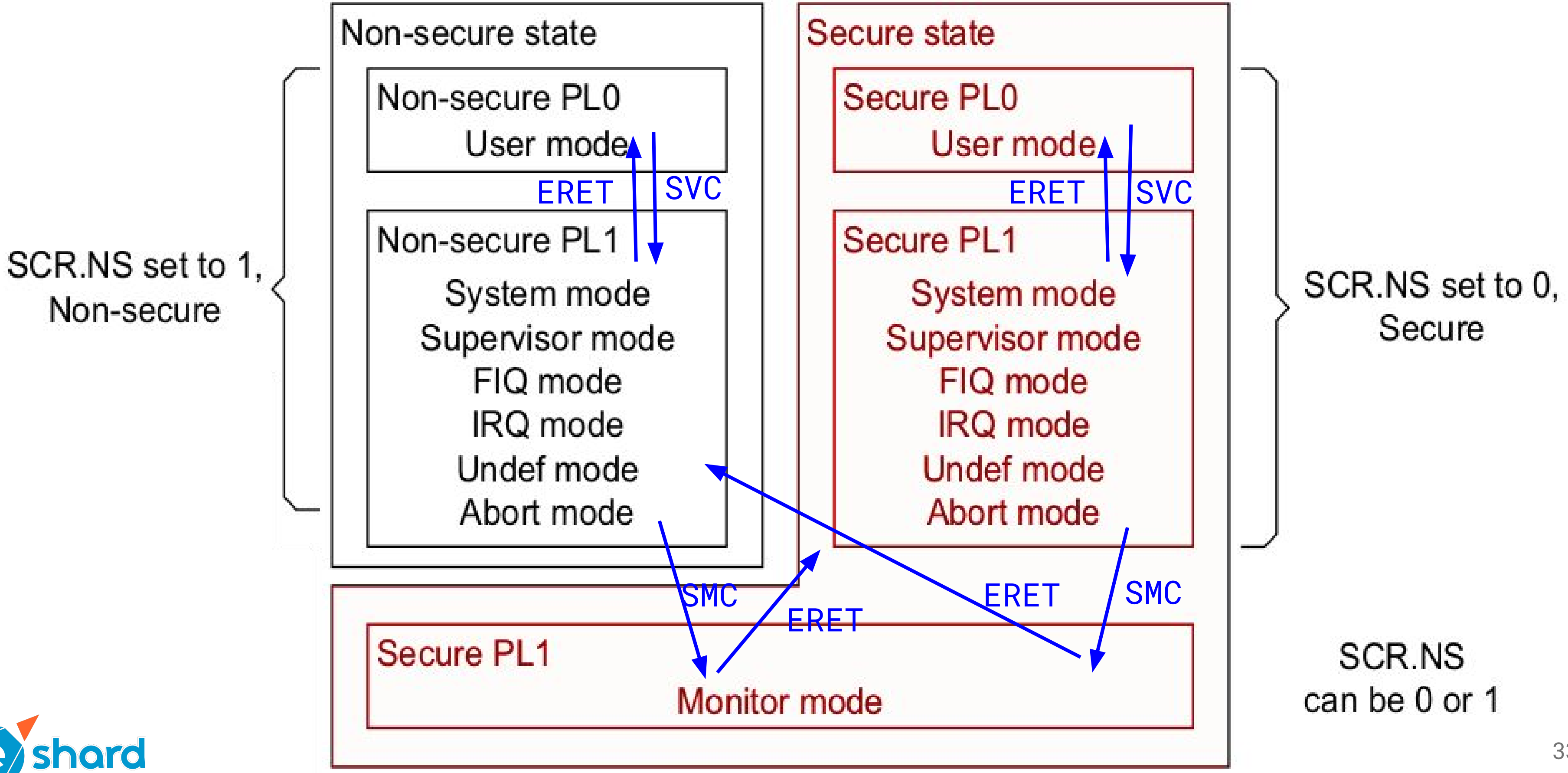
Implementation:

New **state** dimension: **NS** is {0, 1}
New processor **mode**: **monitor** (in addition to usr, svc, …) PL1
New instruction: **SMC**, similar to SVC but for: PL1 → monitor
New system controls (SCR, …), CP15 Register banking

Modes, privilege levels, Security States (Simplified, ARMv7-A)

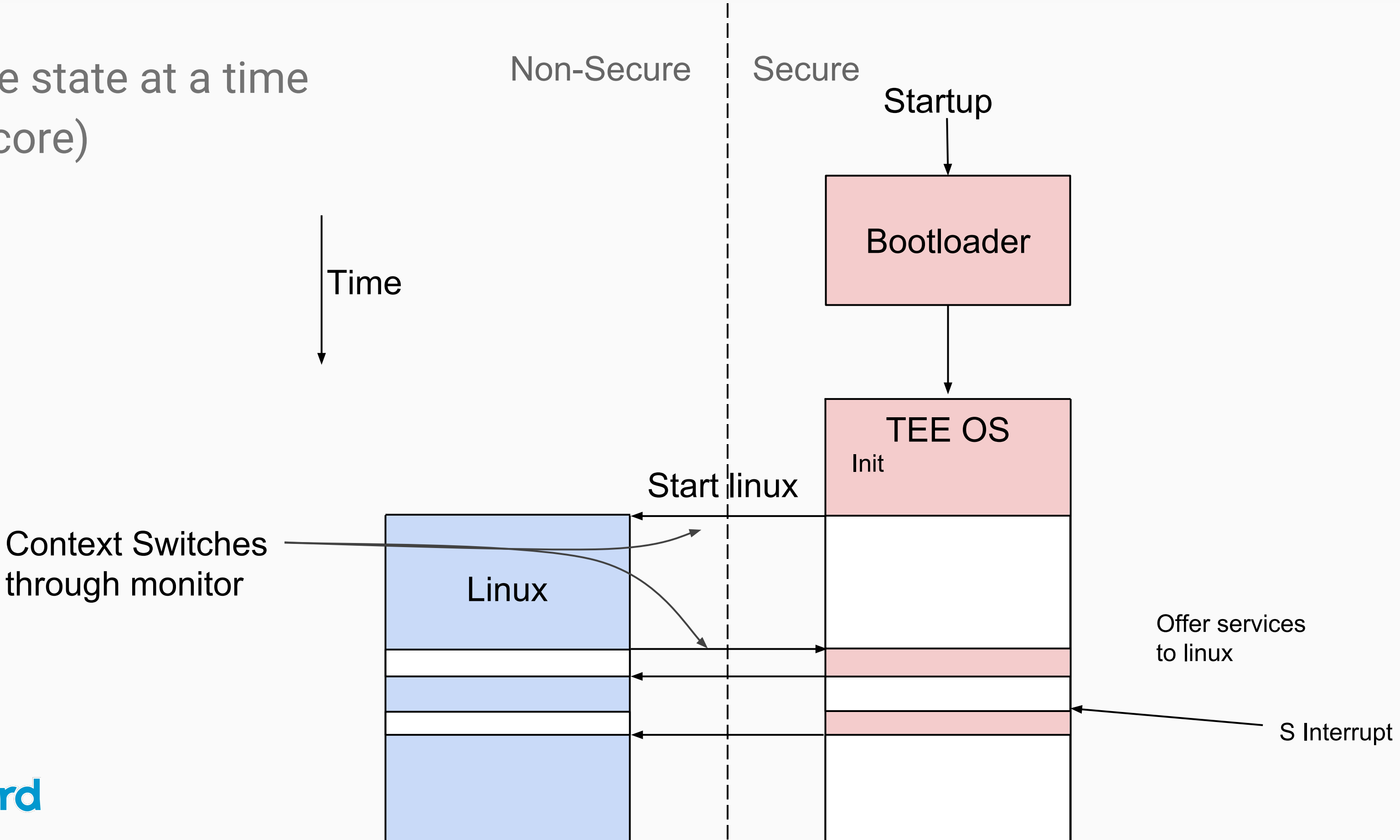


Modes, privilege levels, Security States (Simplified, ARMv7-A)



Execution

In one state at a time (per core)



Attack

RSA-CRT - Fast implementation of the RSA signature based on CRT

Signature s of the message m is defined as:

$$s = m^d \ (mod\ n)$$

Some constants precalculated at key generation

$$d_p = d\ (mod\ p-1)$$

$$d_q = d\ (mod\ q-1)$$

$$q_{inv} = q^{-1}\ (mod\ p)$$

The signature can be calculated:

exponents and modulus are smaller ⇒ faster

$$s_1 = m^{d_p}\ (mod\ p)$$

$$s_2 = m^{d_q}\ (mod\ q)$$

$$h = q_{inv}(s_1 - s_2)\ (mod\ p)$$

$$s = s_2 + hq$$

RSA-CRT Fault Attack - “Bellcore”

On the Importance of Checking Cryptographic Protocols for Faults

Boneh, DeMillo, Lipton 1997

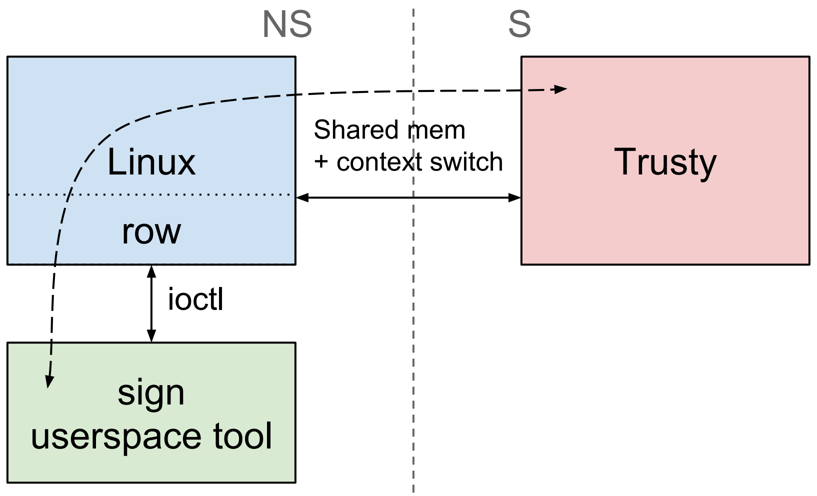
If d_q is faulted and becomes d_q'

The signature calculation become *s’* instead of *s*

p can then be calculated and is: $p = gcd(s'^e - m,\ N)$

The whole private key can then be derived

PoC - Implemented System Overview



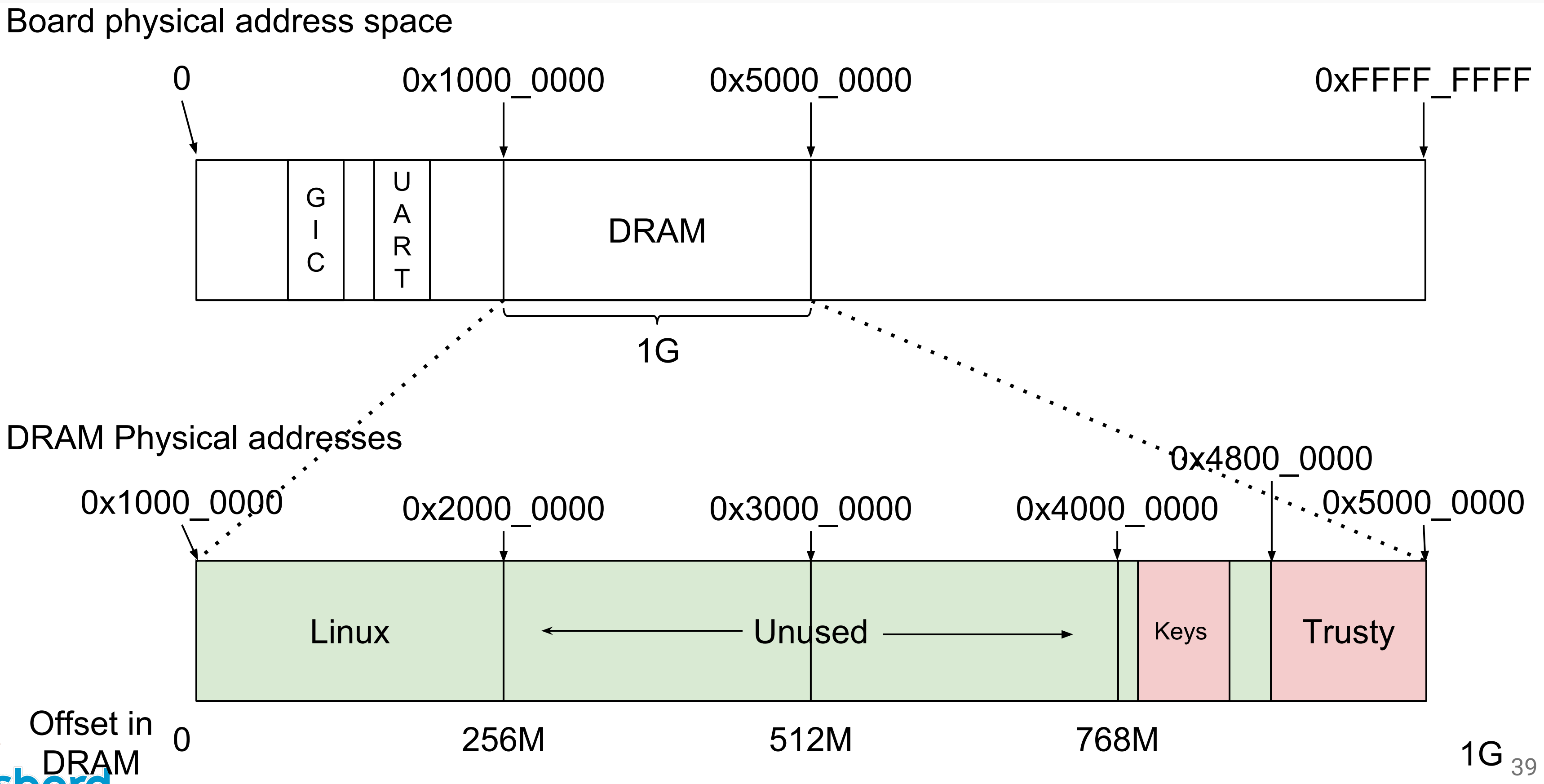
Trusty generates random RSA key in secure memory at boot

Offers signature mechanism to Linux

“**row**” module used to generate faults to a target address using Rowhammer

“**sign**” tool uses Trusty’s signature service and calculates gcd

Memory Setup



Example Session - Sign Message - No Fault in Key

Kernel

Trusty

Userspace

```
[root@alarm ~]# ./sign hello
message: 0x68656c6c6f0000000000000000000000…
[ 5326.601784] row: ROW_IOCTL_SIGNATURE
sign_crt:88: s = 0x7c1a8306e5a4910b3d94d06e62174f4669…
public key:
  e = 0x3
  n = 0xc2c617ed42871bfc97b83cc1e392f0b03323858…
signature: 0x7c1a8306e5a4910b3d94d06e62174f4669…
gcd == n, no fault have happened in the key area
```

Example Session - Hammer

```
[root@alarm ~]# echo 1 > /sys/module/row/params/do_hammer
[ 5343.279638] row: addr[0]=a17f0000 (pa 400F0000)
[ 5343.284277] row: addr[1]=a1810000 (pa 40110000)
[ 5346.779417] dmc: R=2MB   nR=0M  0 MnR/s (29) @ ~0 MB/s
[ 5346.779417]      W=128MB nW=32M 9 MnW/s (4)  @ ~36 MB/s
[ 5346.790429] row: elapsed=42294
```

Memory Controller
Counters

Example Session - Sign Message - Key Faulted

```
[root@alarm ~]# ./sign hello
message: 0x68656c6c6f000000000000000000000000…
[ 5355.711724] row: ROW_IOCTL_SIGNATURE
sign_crt:88: s = 0x657eb547c65344406a9d7f44a58d…
public key:
  e = 0x3
  n = 0xc2c617ed42871bfc97b83cc1e392f0b03323858…
signature: 0x657eb547c65344406a9d7f44a58da72860…
Success: found private factor f:
0xc5d85c20911b6fb56e795d857ea927f28112f7321e713…
other factor of n: n/f = 0xfc069e141107cf589b9464d8341ea18b4c2769513331f…
```

Calculated Signature has changed

Found a factor!

Cannot Access Secure Areas - Protected by TZASC

```
[root@alarm ~]# cat /sys/module/row/params/do_dump_target_pa
[ 5372.191371] Unhandled fault: imprecise external abort (0x406) at 0x76e15004
[ 5372.198354] pgd = 8ced0000
[ 5372.201071] [76e15004] *pgd=1cdd5831, *pte=1b3c175f, *ppte=1b3c1c7f
[ 5372.207400] Internal error: : 406 [#1] SMP ARM
```

Conclusion

- Proof of attackability
- Limitation: Attack memory along S/NS border
- Need to study current TrustZone implementations to determine if exploitable
- Mitigation is simple
- Intern positions open: LLVM Obfuscator / Side-Channel Analysis
  Distributed Computing

eshard

Questions

Remarks (1)

Different point of view compared to other Rowhammer applications:

We are at kernel level, so:

- Easy to access memory using physical addresses
- Easy to bypass caches

This is how drivers for memory mapped devices work

See /proc/iomem

Remarks (2)

Do real world TEE implementations use S regions where Rowhammer is possible?

→ Need to make a mapping of the address space
Easily done from NS space, access to S regions ⇒ external abort

Why Trusty?

Simple & Clean implementation (but no docs)

- Based on LK, nearly vanilla
  - Multiple kernel tasks, preemptive scheduler
  - Memory Management primitives (page tables, ...)
  - Usual primitives: mutexes, timers, …
- Trusty additions in another repo (extensible build system)
  - TrustZone Monitor
  - Userspace applications + syscall interface
  - High Level IPC between S / NS

Trusty - Board Support

- New platform `lk/trusty/platform/`

- Cortex-A9 Support (rough):
  - GICv1
  - Private Timer

- Drivers
  - UART
  - TZASC
  - ...

Annex

Trusty Source Code Organization

- **`external/lk`**: Nearly “normal” LK
- **`lk/trusty`**: additions to LK
    - **`lib/sm`**: TrustZone Monitor
    - **`lib/uthread`**: Userspace threads
    - **`lib/trusty`**: Various
    - **`platform/generic-arm64`**: Support for qemu arm64 virtual board.
    - **`platform/vexpress-a15`**: Support for ARM’s reference board
- **app**: Userspace trusty applications “Trustlets”.

Stdcall / Fastcall calling conventions

SMC, parameters in registers:

- Fastcall: atomic
- Yielding call “stdcall”: can be preempted by a NS interrupt (needs resume)

In Trusty an SMC Number is defined as:

```
#define SMC_FASTCALL_NR(entity, fn)    SMC_NR((entity), (fn), 1, 0)

#define SMC_NR(entity, fn, fastcall)
                         ((fastcall) & 0x1) << 31) | \
                          ((entity) & 0x3F)  << 24) | \
                          ((fn) & 0xFFFF) \
                          )
```

Trusty fastcall

Trusty: register handler to trusty

```
int callback(args) { … }
register_fastcall(call number, callback)
```

Linux: use trusty library in order to issue an SMC with particular call number

```
int ret = trusty_fastcall(call number, args)
```

References

DRAM

- *Memory Systems - Cache, DRAM, Disk*
- *Computer Architecture - Main Memory*, Onur Mutlu
- Rajeev Balasubramonian
- *Main Memory* - Christos Kozyrakis

Rowhammer

- *Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors*, Yoongu Kim
- *Exploiting the DRAM rowhammer bug to gain kernel privileges*, Mark Seaborn and Thomas Dullien
- *Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript*, Daniel Gruss, Clémentine Maurice, and Stefan Mangard
- *Drammer: Deterministic Rowhammer Attacks on Mobile Platforms*, Victor van der Veen

TrustZone

- *Reflections on Trusting TrustZone*, Dan Rosenberg
- https://bits-please.blogspot.com, Gal Beniamini

RSA-CRT Fault Attack

- *On the Importance of Checking Cryptographic Protocols for Faults*, Boneh, DeMillo, Lipton 1997

Trusty

- https://source.android.com/security/trusty/