CLKscrew

Exposing the Perils of Security-Oblivious Energy Management

Adrian Tang, Simha Sethumadhavan, Salvatore Stolfo
Today’s systems cannot exist without Energy Management

Source: Adapted from S. Borkar (Intel)
Today’s systems cannot exist without Energy Management

Industry

Snapdragon 820 Consumes 30% Less Power

Power Consumption Trend
Normalized Real-Life Usage

Enhanced Tuning/Overclocking on 4th Gen Intel® Core™ Processors

New Power Management Capabilities

- Per Core P-States (PCPS)
  - Allows cores to run at individual frequency/voltage
- Energy Efficient Turbo Mode (EET)
  - Core throughput / stall behavior monitored
  - Core frequency is increased only if it is energy efficient
- Uncore Voltage/Frequency Scaling (UFS)
  - Nehalem: Core could turbo up, Uncore at fixed frequency
  - Sandy Bridge: Core and Uncore turbo up/down together
  - Haswell: Each Core & Uncore treated independently
    - Core-Bound Applications: Drive Core frequency higher without needing to increase Uncore
    - LLC/Memory-Bound Applications: Drive Uncore frequency higher without burning core power

Today’s systems cannot exist without **Energy Management**
Today’s systems cannot exist without Energy Management.
Exploiting software interfaces to
Energy Management

Software-based attacker

Stretch operational limits

Induce faults

frequency

voltage

decryption

key
Exploiting software interfaces to Energy Management

Traditional fault attacks
- Need physical proximity
- Need separate equipment
- Soldering, crocodile clips, wire, etc.
**CLKSCREW**: Exposing the perils of security-oblivious Energy Management

**New attack vector** that exploits energy management

**Practical attack** on trusted computing on ARM devices

**Impacts** hundreds of millions of deployed devices

**Lessons** for future energy management designs to be security-conscious
I. DVFS and Regulators

II. The CLKSCREW Attack

III. Attacking ARM Trustzone

IV. Concluding Remarks
Dynamic Voltage and Frequency Scaling (DVFS)

Energy consumption

Frequency

Voltage

DVFS
Hardware & Software Support for DVFS

- Software
- Hardware
- DVFS

- Power Governor
- Vendor Device Driver
- Memory-Mapped Registers
  - Frequency Regulator
  - Voltage Regulator
Hardware Regulators and Software Interfaces

Operating frequency and voltage can be configured via memory-mapped registers from software.
Do hardware regulators impose limits to frequency/voltage changes?
Frequency / Voltage Operating Point Pairs (OPPs)

Legend:
★★★★ Vendor-recommended

Nexus 6

- Android v5 (Lollipop)
- Turbo Charging
- 5.94 inch QHD AMOLED
- 2.7 GHz Process

![Graph showing frequency vs. voltage for Nexus 6]
No safeguard hardware limits

Lower voltage → Lower minimum required frequency to induce instability

Legend:
- ⭐⭐⭐ Vendor-recommended
- ⬠ Max OPP reached before instability
Frequency / Voltage Operating Point Pairs (OPPs)

---

**Device ‘A’**

- **Maximum OPP**
- **Vendor stock OPP**

---

**Device ‘B’**

- **Maximum OPP**
- **Vendor stock OPP**

---

**Frequency (GHz)**

**Voltage (V)**
Does DVFS operate across security boundaries?

Trusted Execution Environments (TEE)
Is DVFS Trustzone-Aware? **No!**

![Diagram showing CPU Core, Frequency & Voltage Regulators, Trustzone Trusted code, Normal Untrusted code, Hardware-enforced isolation, and Regulator HW-SW interface.](image)
I. DVFS and Regulators

II. The CLKscrew Attack

III. Attacking ARM Trustzone

IV. Concluding Remarks

I. DVFS and Attack

II. The CLkscrew

III. Attacking ARM Trustzone

IV. Concluding Remarks
Can we attack Trustzone code execution using software-only control of the regulators?
Induce timing faults

- confidentiality
- integrity
- availability
How do faults occur (due to over-raising frequency)?
How do faults occur (due to over-raising frequency)?

CLK signal

higher frequency

less time for data to propagate

input

output

flip-flop

intermediate logic path

1

'0' \rightarrow '1'

timing violation
How do faults occur (due to over-raising frequency)?

Expected: … a777511b …

Faulty output: … a7775151 …
CLKSCREW Challenges & Solutions

#1: Regulator operating limits

#2: Self-containment within same device

#3: Noisy complex OS environment

#4: Precise timing

#5: Fine-grained timing resolution
CLKSCREW Challenges & Solutions

#1: Regulator operating limits

#2: Self-containment within same device

#3: Noisy complex OS environment

#4: Precise timing

#5: Fine-grained timing resolution

Addressed earlier in DVFS regulators
# CLKSCREW Challenges & Solutions

#1: Regulator operating limits

#2: Self-containment within same device

Cores have different frequency regulators

Core pinning

#3: Noisy complex OS environment

#4: Precise timing

#5: Fine-grained timing resolution
#1: Regulator operating limits

#2: Self-containment within same device

#3: Noisy complex OS environment

#4: Precise timing

#5: Fine-grained timing resolution

CLKSCREW Challenges & Solutions

Core pinning

Disable interrupts during attack

Challenges & Solutions
CLKSCREW Challenges & Solutions

#1: Regulator operating limits

#2: Self-containment within same device

#3: Noisy complex OS environment

#4: Precise timing

#5: Fine-grained timing resolution

High-precision timing loops in attack architecture

Cache-based execution timing profiling

Victim thread

~1,100,000,000,000 clock cycles

~65,000 clock cycles

asm volatile("1: subs %0, %0, #1 \n" " bhi 1b \n":"r" (loops));
I. DVFS and Regulators

II. The CLKSCREW Attack

III. Attacking ARM Trustzone

IV. Concluding Remarks
Subverting Trustzone Isolation with CLKSCREW

**Confidentiality Attack**
infer secret AES key stored within Trustzone

- Trustzone
  - secret key
  - AES decryption
  - ciphertext
  - plaintext

- Normal
  - plaintext

**Integrity Attack**
load self-signed app into Trustzone

- Trustzone
  - signed app
  - app binary
  - public key
  - digital signature
  - SHA-256 hash
  - RSA decryption
  - plaintext hash
  - verify & load app

- Normal
  - signed app
Key Inference Attack: Threat Model

Victim app: AES decryption app executing in Trustzone

Attacker’s goal: Get secret AES key from outside Trustzone

Attacker’s capabilities: 1) Can repeatedly invoke the decryption app  
2) Has software access to hardware regulators
Key Inference Attack: Summary

Idea: Induce a fault during the AES decryption
Infer key from a pair of correct and faulty plaintext

[Differential Fault Analysis [1]]

### Key Inference Attack: CLKSCREW Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base voltage</td>
<td>1.055V</td>
</tr>
<tr>
<td>High frequency</td>
<td>3.69GHz</td>
</tr>
<tr>
<td>Low frequency</td>
<td>2.61GHz</td>
</tr>
<tr>
<td>Fault injection duration</td>
<td>680 no-op loops (~39 μsec)</td>
</tr>
</tbody>
</table>

Differential Fault Analysis needs CLKSCREW to deliver a one-byte fault to the 7th AES round.
Execution timing of Trustzone code can be profiled with hardware cycle counters that are accessible outside of Trustzone.
Key Inference Attack: **Timing Profiling**

How varied is the execution timing of the victim decryption app?

Not too much variability in terms of execution time
Key Inference Attack: Timing Profiling

Can we effectively control the timing of the fault delivery with no-op loops?

Number of no-op loops is a good proxy to control timing of fault delivery
Key Inference Attack: Fault Model

Our fault model requires our attack to inject fault:

- Exactly one AES round at the 7th round
- Corruption of exactly one byte
**Key Inference Attack: Fault Model**

**Precision:** How likely can we inject fault in exactly one AES round?

![Bar chart showing the distribution of faulted AES rounds](chart.png)

More than 60% of the resulting faults are precise enough to corrupt exactly one AES round.
**Key Inference Attack: Fault Model**

**Transience:** How likely can we corrupt exactly one byte?

Out of the above faults that affect one AES round, more than half are transient enough to corrupt exactly one byte.
Key Inference Attack: Results

Controlling $F_{pdelay}$ allows us to precisely time the delivery of the fault to the targeted AES round.

Statistics:
- ~20 faulting attempts to induce one-byte fault to desired AES round.
- ~12 min on a 2.7GHz quad core CPU to generate 3650 key hypotheses.
I. DVFS and Regulators

II. The CLKSCREW Attack

III. Attacking ARM Trustzone

IV. Concluding Remarks
Attack Applicability to Other Platforms

Energy management mechanisms in the industry is trending towards finer-grained and increasingly heterogeneous designs.
Possible Defenses

Hardware-Level

- Operating limits in hardware
- Separate cross-boundary regulators
- Microarchitectural Redundancy

Software-Level

- Randomization
- Code execution redundancy
CLKScrew: Exposing the perils of security-oblivious Energy Management

New attack surface via energy management software interfaces

Not a hardware or software bug
Fundamental design flaw in energy management mechanisms

Future energy management designs must take security into consideration

Adrian Tang - @0x0atang
Simha Sethumadhavan, Salvatore Stolfo