



Pillaging and plundering SGX with Software-based Fault Injection Attacks

Kit Murdock

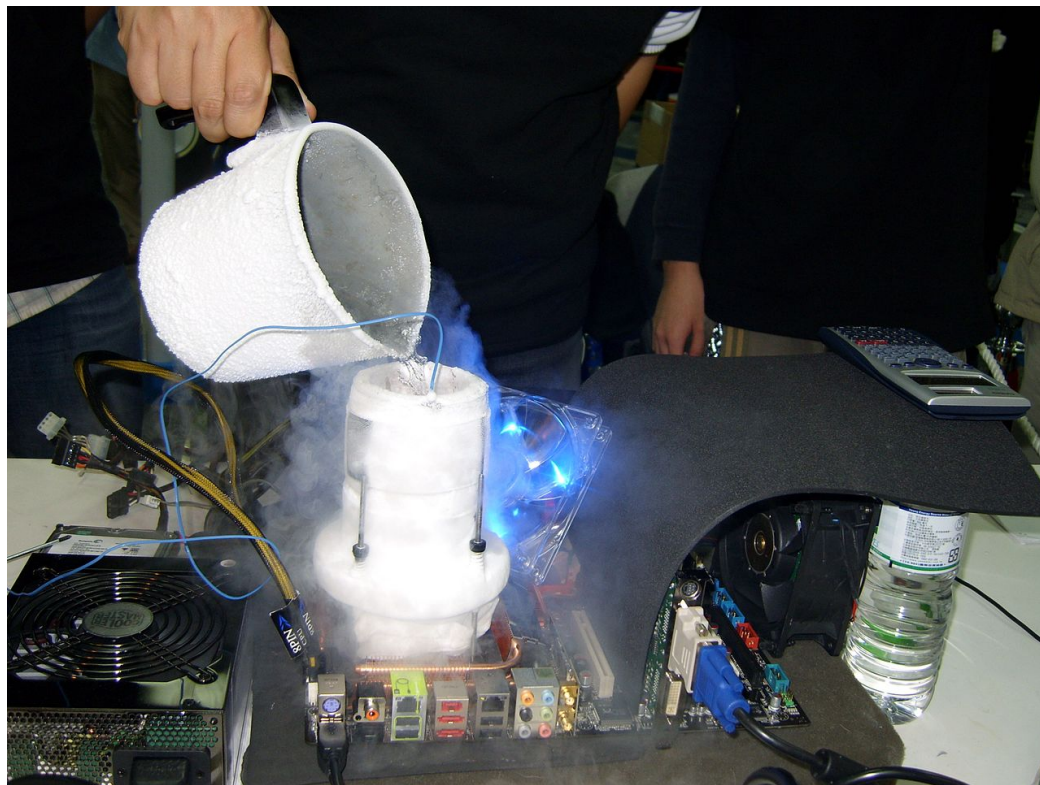


Pillaging and plundering SGX with Software-based Fault Injection Attacks

Kit Murdock

Overclocking

CPU Operating Speed	User Define
- External Clock	148 MHz
- Multiplier Factor	x16.5
AGP Frequency	72 MHz
CPU FSB/DRAM ratio	Auto
CPU Interface	Enabled



[Image attribution: Rico Shen](#)

Pillaging and plundering SGX with Software-based Fault Injection Attacks



[Image attribution: Charles Gaudette](#)

Pillaging and plundering SGX with Software-based Fault Injection Attacks

TS 8.70 - Monitoring

TECHPOWERUP

Performance

Settings

- Clock Modulation 100.0%
- Chipset Modulation 100.0%
- Set Multiplier 63 T
- Power Saver
- Disable Turbo SpeedS
- BD PROCHOT C1E
- Task Bar On Top
- Log File More D

RightMark CPU Clock Utility

CPU info

AMD Athlon X2

CPU model: AMD
CPU core: Mancl
PM features:
Core clock: 2010.30
Throttle: 2010.30
Core temp.: 51.2°
Multiplier (FID): 10.0x
Req.Vcore (VID): 1.200V

CPU 0 CPU 1

CPU Cache Mainboard Memory SPD About

Processor

Name: AMD Opteron Processor
Code Name: Toledo
Package: Socket
Technology: 90 nm
Specification: Dual Core AMD
Family: F Mode
Ext. Family: F Ext. Mode
Instructions: MMX (+), 3DNow! (+)

Clocks (Core#0)

Core Speed: 2651.4 MHz
Multiplier: x 10.0
Bus Speed: 265.1 MHz
HT Link: 795.4 MHz

Selection: Processor #1

FX VISION AMD OverDrive

Status Monitor

- CPU Status
- GPU Status
- Board Status
- Logging

Performance Control

- Clock/Voltage
- Memory
- BEMP
- Fan Control
- AMD Smart Profiles
- Benchmark
- Stability Test
- Auto Clock

System Information

- Basic
- Detailed
- Diagram

CPU-Tweaker 2.0

CPU

Model: AMD Phenom(tm) II X4 965 Processor CPUID: F43 Rev: C3
Socket: AM3 (941) Tech: 45 nm Cores/Threads: 4 / 4 VCore: 0.000 V

MotherBoard

Vendor: ASUSTeK Computer INC. Model: M4A88TD-M/USB3
Chipset: AMD 785GX BIOS version: 0902 Date: 12/10/2010

Memory

Type: DDR3 Manufacturer: Part Nb.:
Size: 2 x 4096 Speed: 1000 (63MHz) @ 7.5.5.17- Chan.: Unganged

System Frequency

BCLK: 200.9 MHz
Cores: x 4.00 803.6 MHz
UnCore: x 10 2008.9 MHz
HT: x 10 2008.9 MHz
RAM: 3:10 669.6 MHz

Timings

Channels: A VDimm: 0.000 V
CAS# Latency (CL): 7
RAS# to CAS# Delay (TRCD): 9
RAS# Precharge (TRP): 9
Precharge Delay (TRAS): 24
Command Rate (CR): 1T

Profile Information

Profile:

Core 0 Multiplier Core 1 Multiplier Core 2 Multiplier Core 3 Multiplier
Core 4 Multiplier Core 5 Multiplier

HT ref. Clock PCIe@ Speed IGP Speed SidePort Speed
CPU VID NB VID Mem VDDQ Mem VTT
CPU VDDC NB Core Voltage NB PCIe@ Voltage CPU HT Voltage
Memory Clock RAS to CAS Delay Command Rate Row Cycle Time

Pillaging and plundering SGX with Software-based Fault Injection Attacks

DVFS

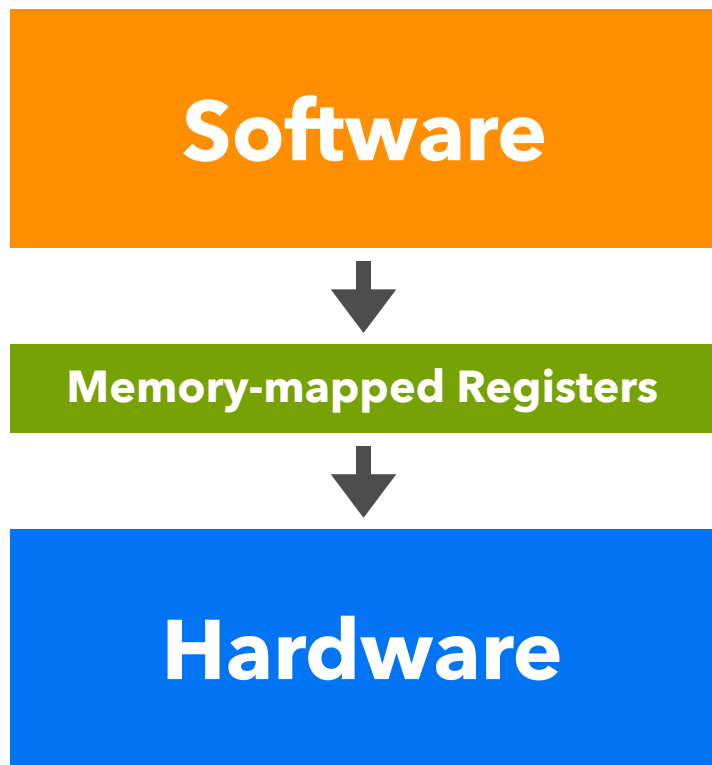
Dynamic Voltage and Frequency Scaling

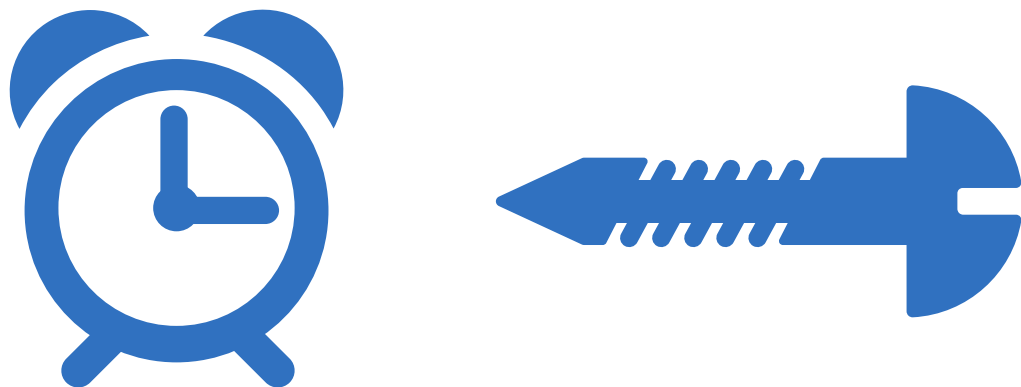


Resilient and reliable

Very fast responses

High-assurance and
low running costs

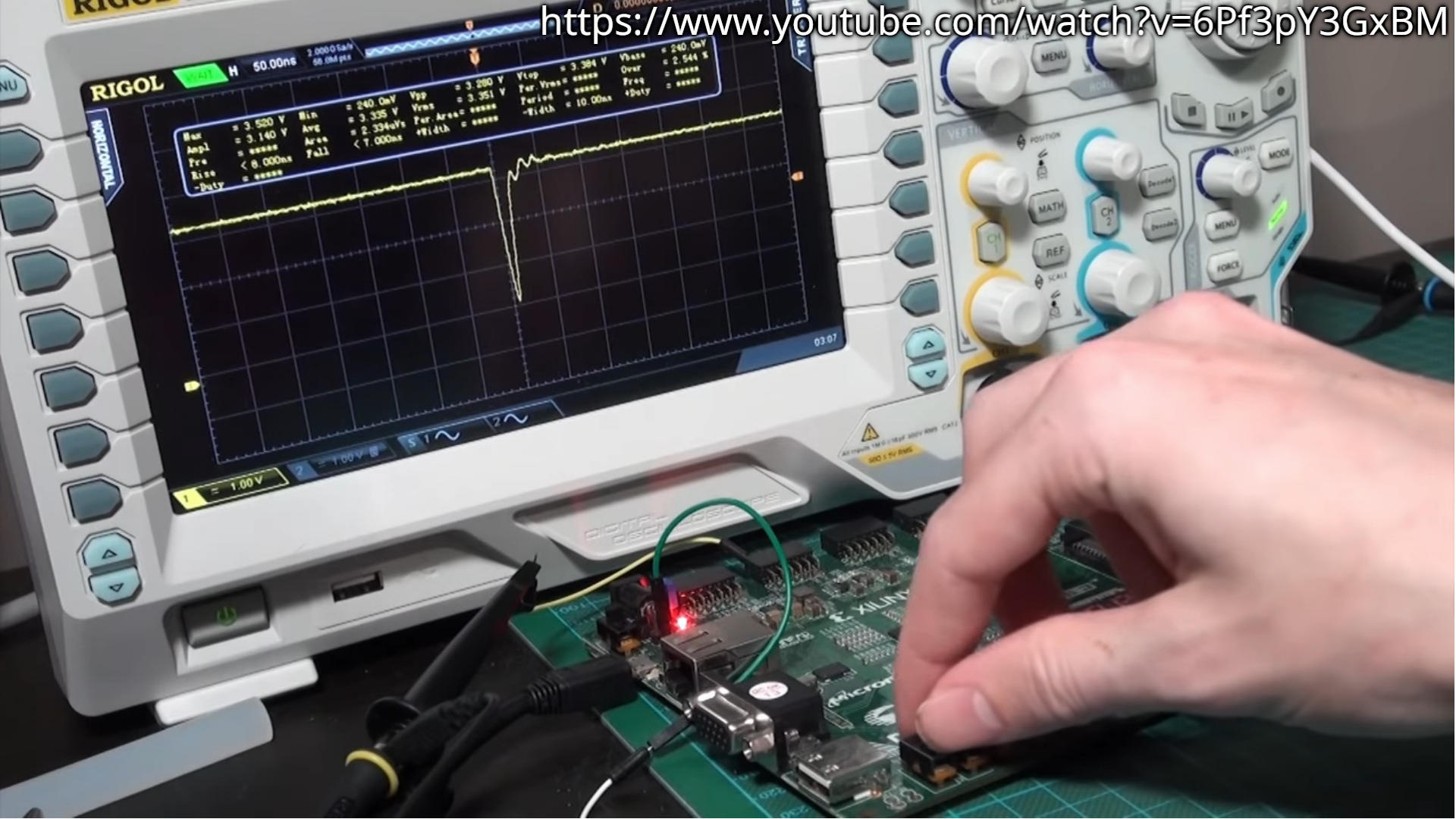


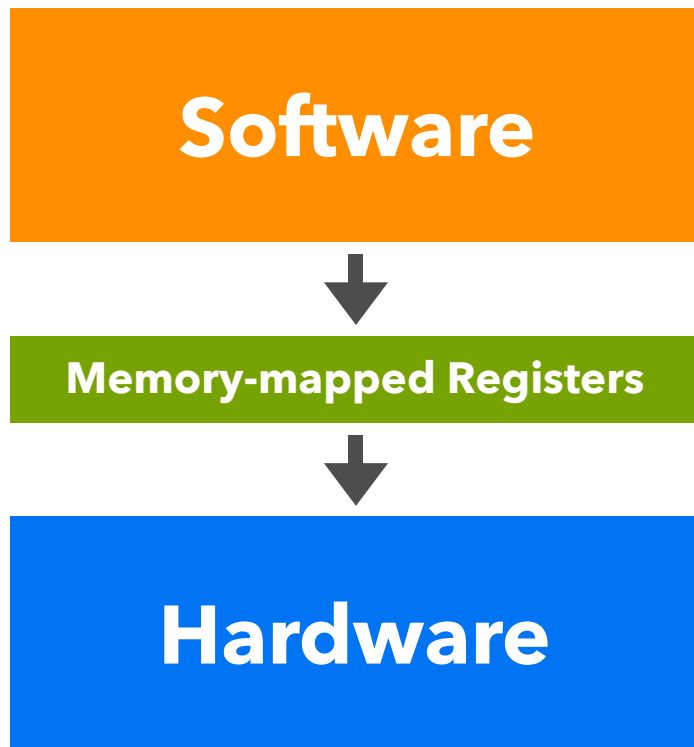


Adrian Tang et al. "CLKSCREW: exposing the perils of security-oblivious energy management"

In: USENIX Security Symposium. 2017

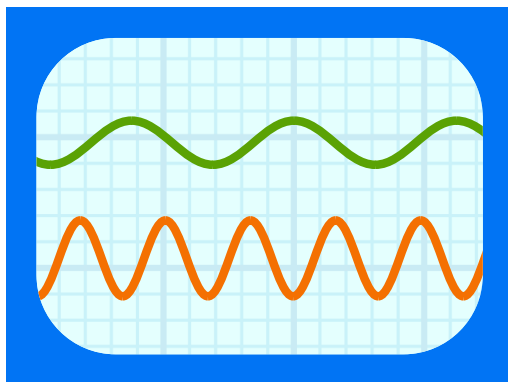
**A new class of
fault attacks**



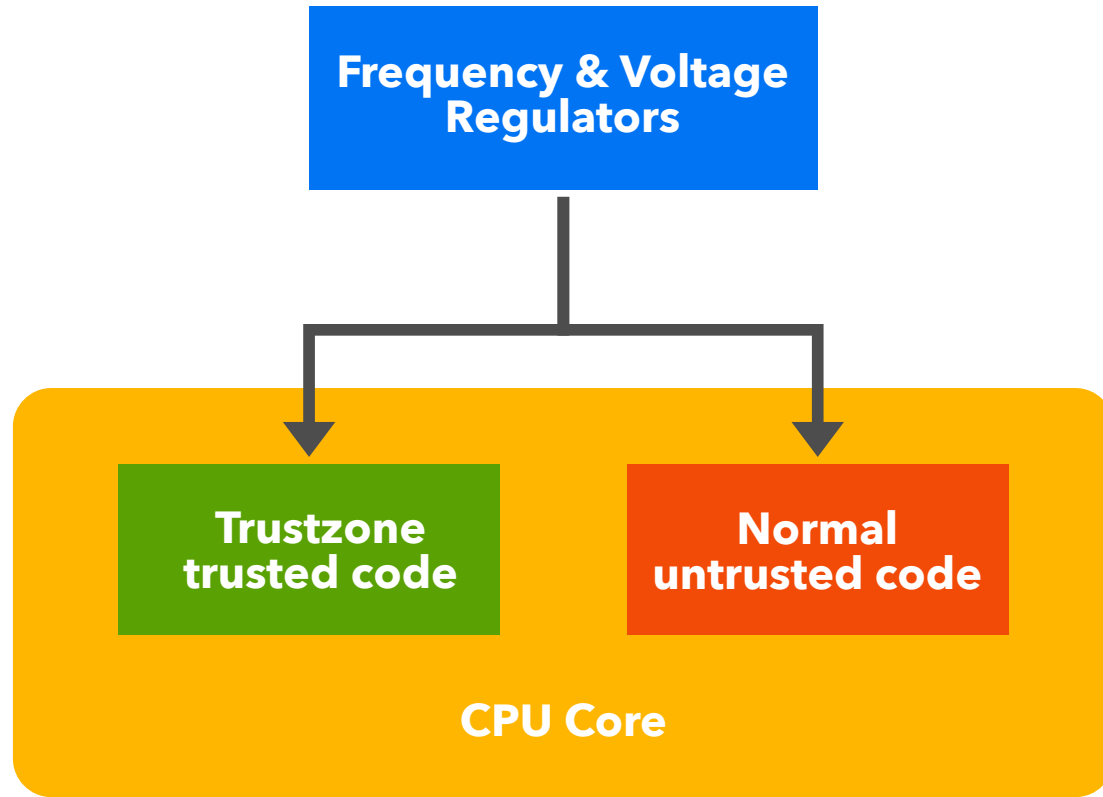


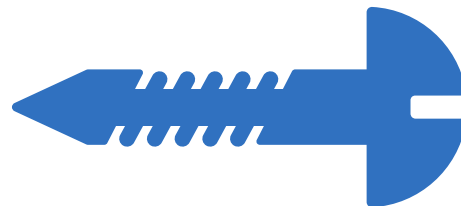
CLKscrew attack

```
add.w (a0)+,d1  
cmp.l a0,d0  
bcc.s loop  
movea.l #$18E,a1  
cmp.w (a1),d1  
bne.w WrongChecksum
```



2 + 2 = 5

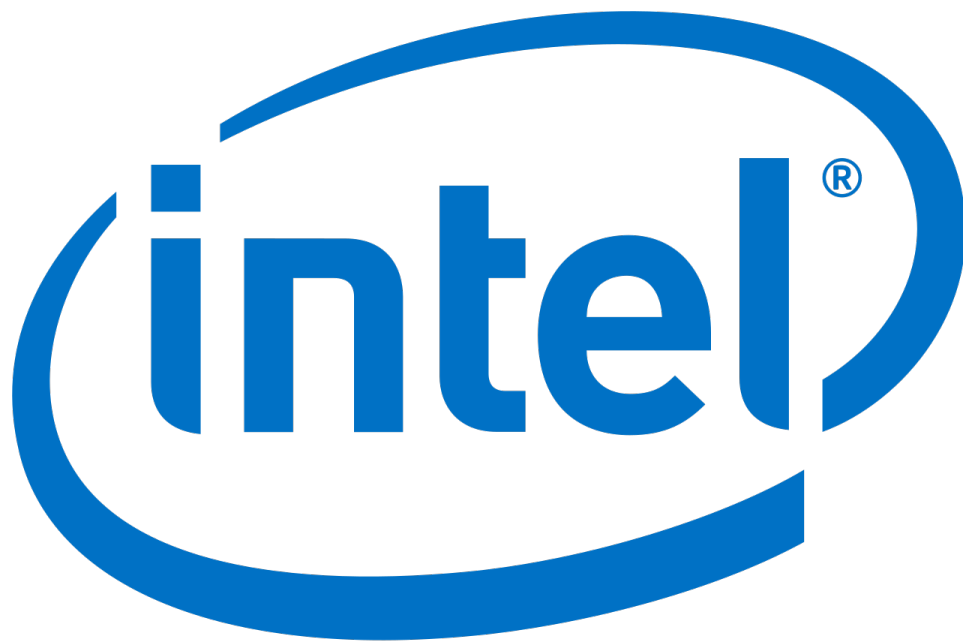




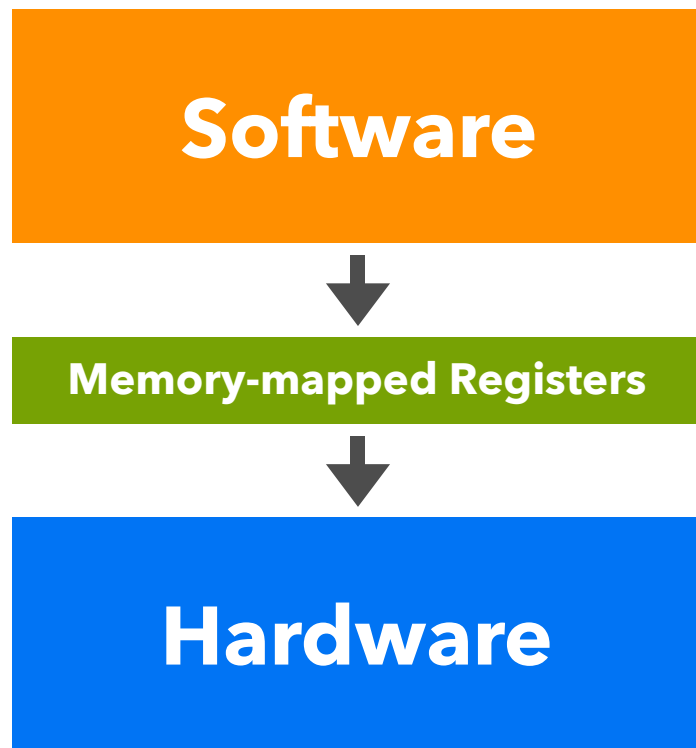
- Infer secret AES key that was stored within Trustzone
- Trick Trustzone into loading a self-signed app

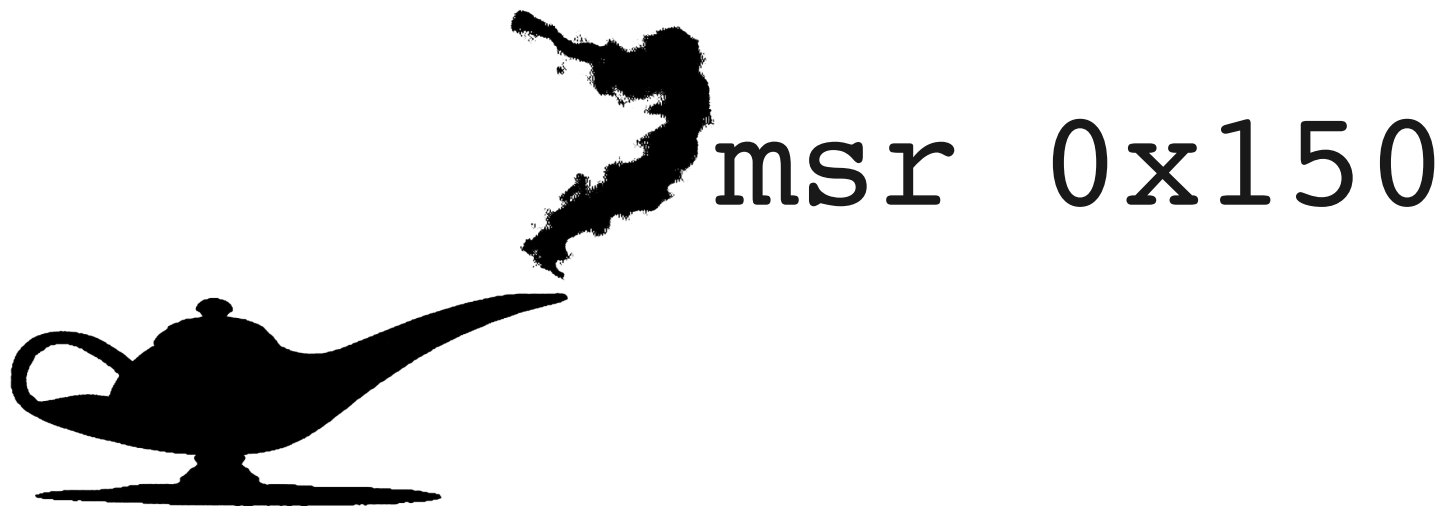
ARM

What about Intel?

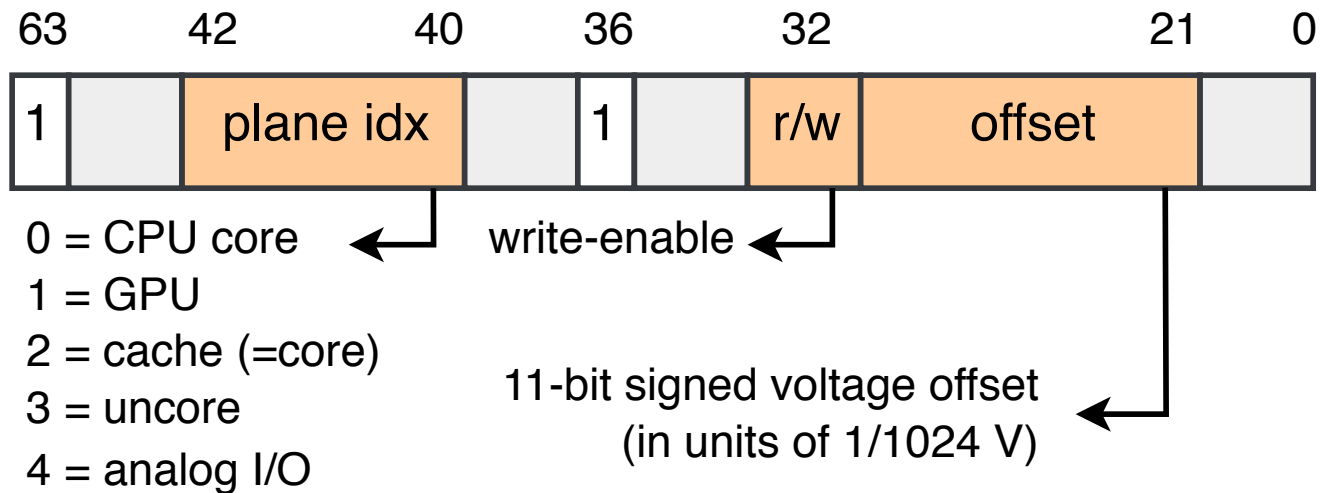


Pillaging and plundering SGX with Software-based Fault Injection Attacks

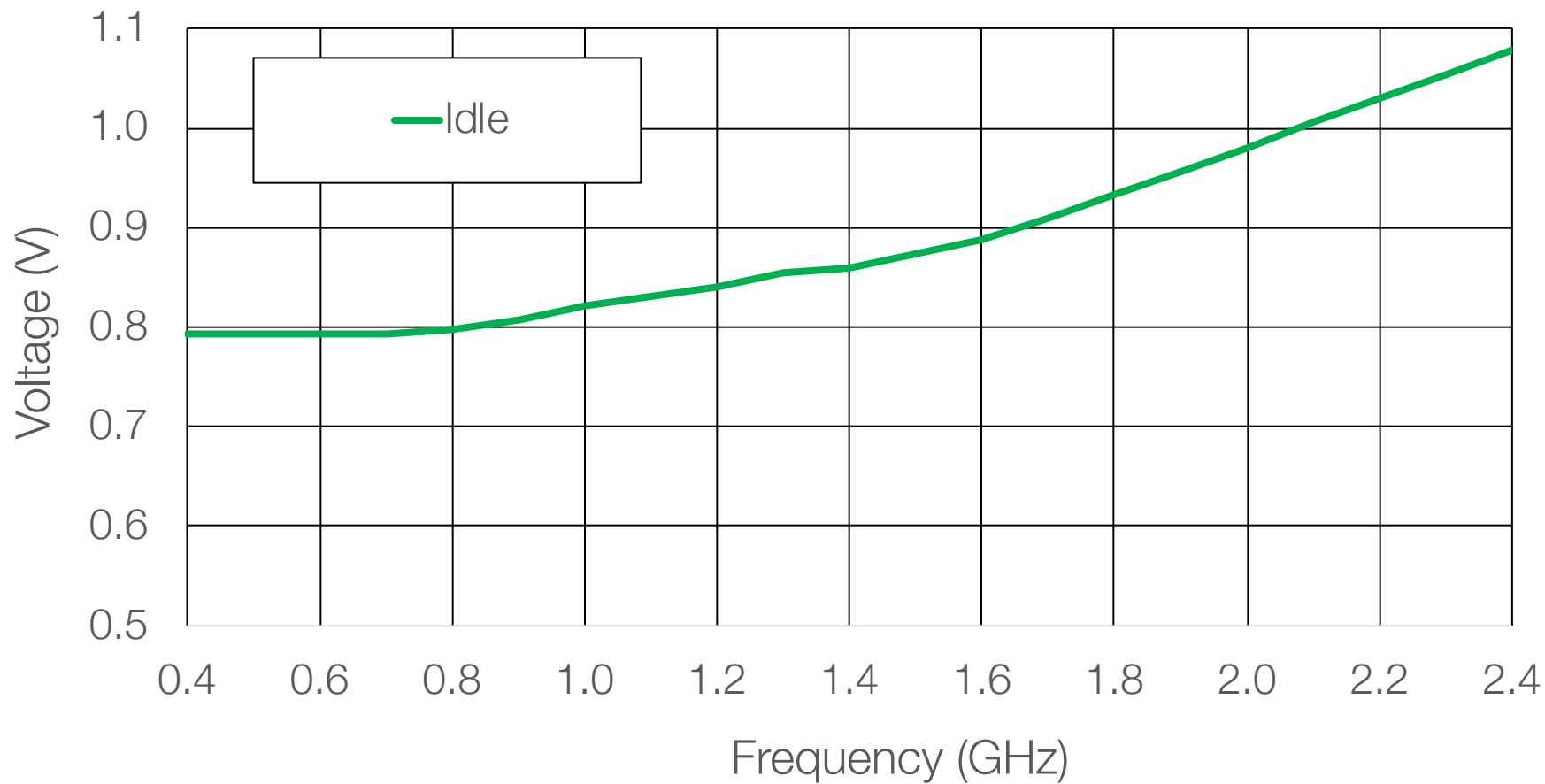




Undervolting Intel CPUs



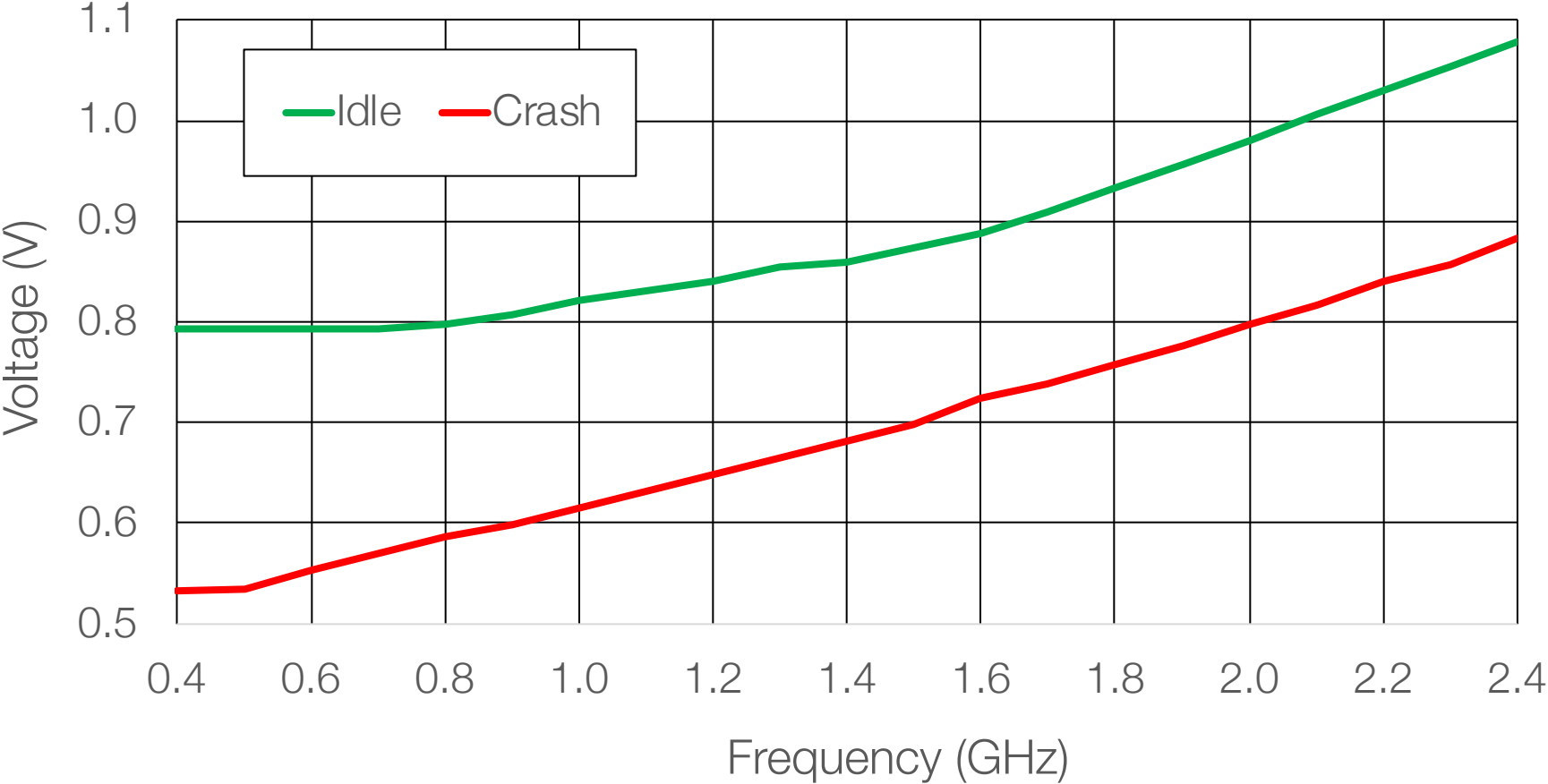
Idle voltage – Intel(R) Core(TM) i3-7100U CPU



bagger> █

⌘

Idle and crash voltages – Intel(R) Core(TM) i3-7100U CPU



**Can we
fault it?**

Will it fault?

```
correct          = 7 * 3
```

```
my_value         = 7 * 3
```

```
// Start undervolting
```

```
while ( my_value == correct )
```

```
{
```

```
    my_value = 7 * 3
```

```
}
```

```
// Can we ever get here?
```

Will it fault?

```
uint64_t multiplier = 0x1122334455667788;
uint64_t correct    = 0xdeadbeef*multiplier;
uint64_t var        = 0xdeadbeef*multiplier;

// start undervolting

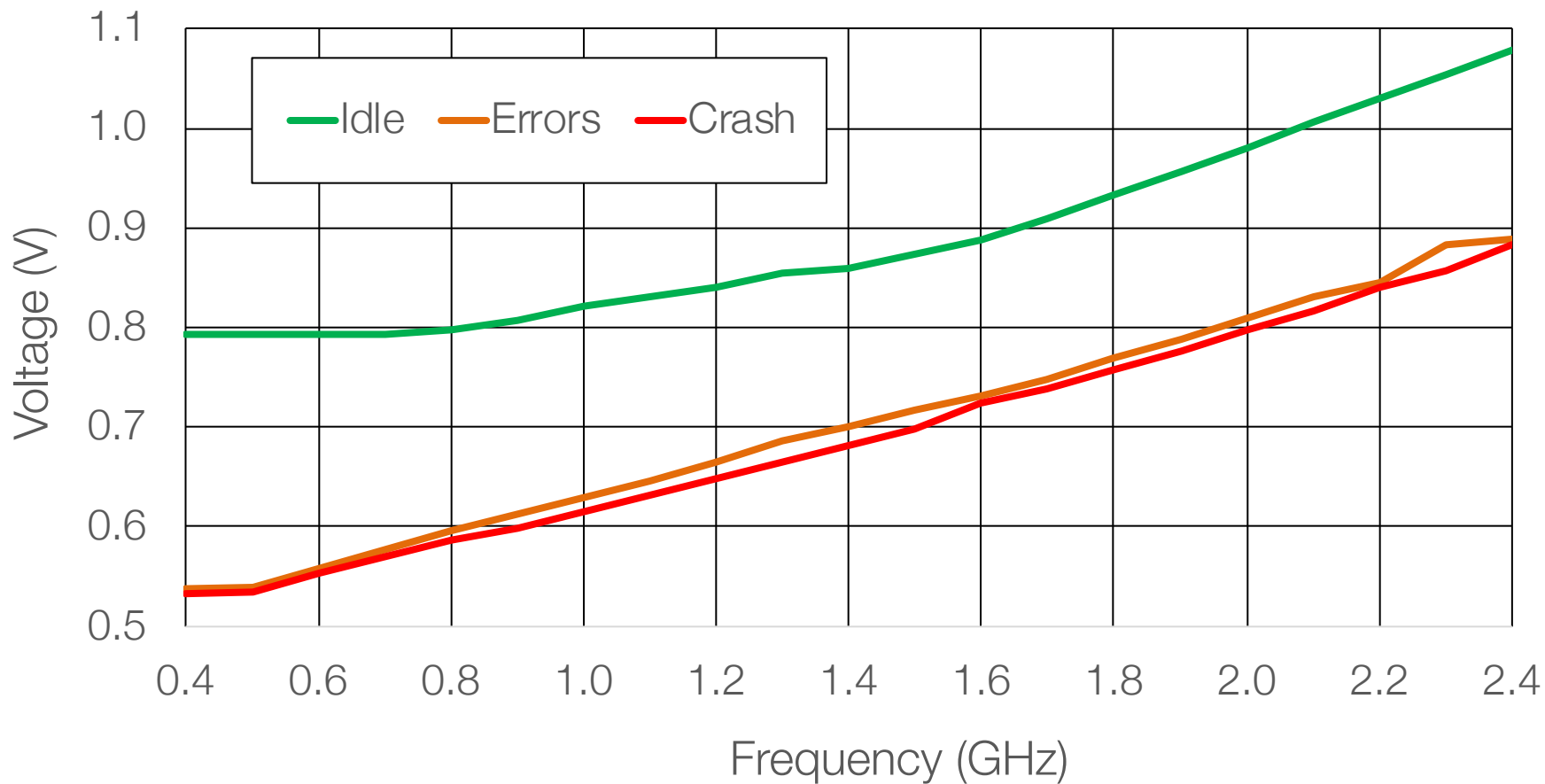
while ( var == correct )
{
    var = 0xdeadbeef * multiplier;
}

// stop undervolting
// Can we ever get here?
uint64_t flipped_bits = var ^ correct;
```

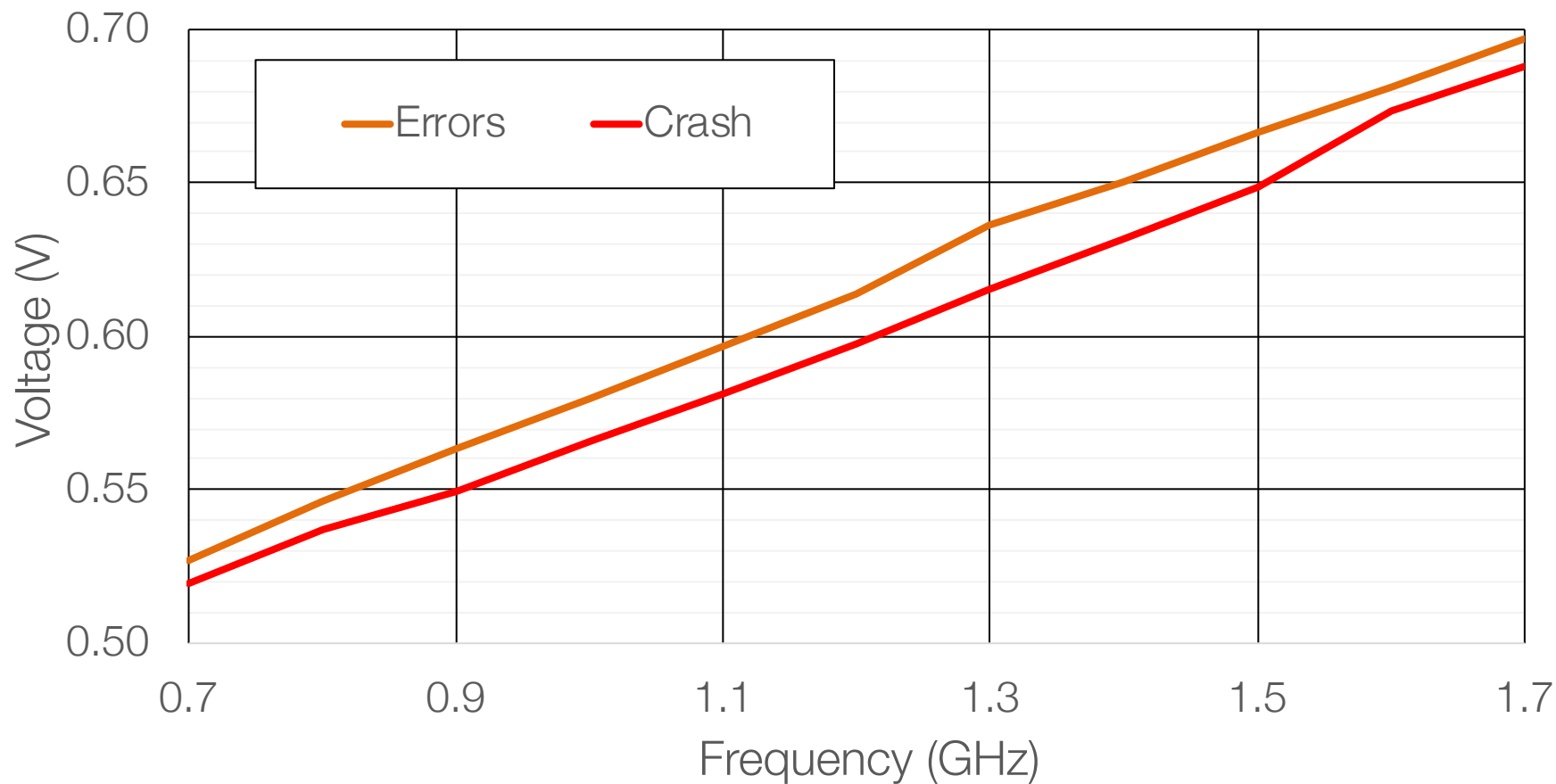
bagger> █

ÿ

Idle, error and crash voltages – Intel(R) Core(TM) i3-7100U CPU



Error and crash voltages – Intel(R) Core(TM) i3-7100U CPU





Carmen Crincoli, but Fhqwhgads

@CarmenCrincoli

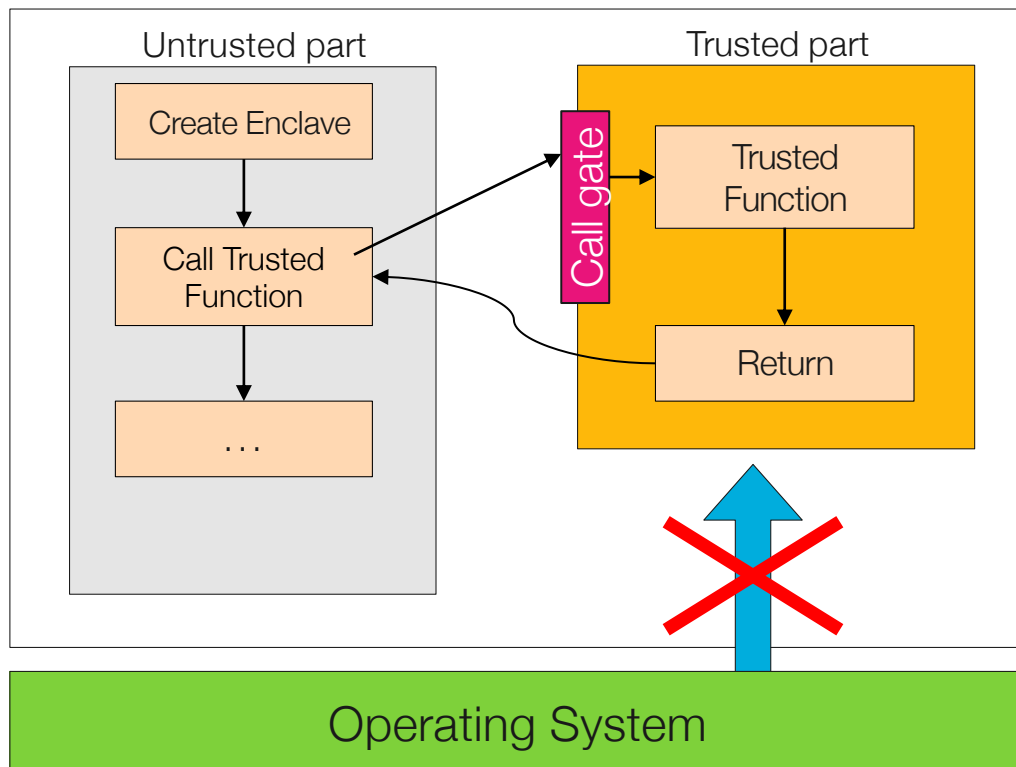


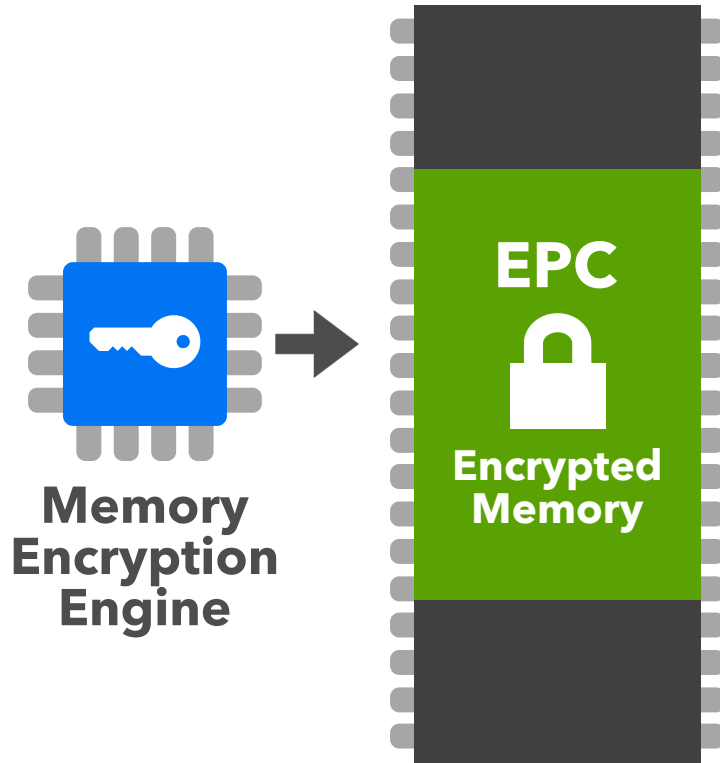
"Plundervolt" sure does have a catchy name and logo for an exploit that... <checks notes> ...requires you to be running as root already.

1:59 AM · Dec 11, 2019 · [Twitter Web App](#)

**Let's meet
SGX**

Application





We can bypass the SGX integrity checks!



Larry Osterman

@osterman

Replying to [@CarmenCrimcoli](#) and [@MT6572A](#)

Since the threat model for SGX assumes that the attacker has root access (and I believe also has physical control over the hardware), this is actually a bigger deal than you make it out to be.

2:07 AM · Dec 11, 2019 · [Twitter Web App](#)

	A	B	C	D	E
1	Operand 1	Operand2	xor answer	undervolting	temperature
5525	0x9e2d4a	0x0024	ffffffffe000000	-272	+36.0C
5526	0x9e2d51	0x0024	ffffffffe000000	-272	+36.0C
5527	0x9eb497	0x0024	ffffffffe000000	-272	+37.0C
5528	0x9eb49e	0x0024	ffffffffe000000	-272	+36.0C
5529	0x9f3bf2	0x0024	ffffffffe000000	-272	+37.0C
5530	0x9f3c15	0x0024	ffffffffe000000	-272	+37.0C
5531	0x9f3c23	0x0024	ffffffffe000000	-272	+37.0C
5532	0x9f3c2a	0x0024	ffffffffe000000	-272	+37.0C
5533	0x9f3c5b	0x0024	ffffffffe000000	-272	+37.0C
5534	0xa04b2d	0x0024	000000002000000	-272	+37.0C
5535	0xa0d2f1	0x0024	000000002000000	-272	+37.0C
5536	0xa0d306	0x0024	000000002000000	-272	+37.0C
5537	0xa269cd	0x0024	000000002000000	-272	+37.0C
5538	0xa269fe	0x0024	000000002000000	-272	+36.0C
5539	0xa61e0e	0x0024	000000001000000	-272	+37.0C
5540	0xa61e38	0x0024	000000001000000	-272	+37.0C
5541	0xa61e3f	0x0024	000000001000000	-272	+36.0C
5542	0xa61e46	0x0024	000000001000000	-272	+36.0C
5543	0xa72d34	0x0024	000000001000000	-272	+36.0C
5544	0xa72d5e	0x0024	000000001000000	-272	+36.0C
5545	0xa8c3ae	0x0024	000000002000000	-272	+37.0C
5546	0xa94b25	0x0024	000000001000000	-272	+37.0C
5547	0xa9d2b1	0x0024	000000001000000	-272	+37.0C
5548	0xaa59fe	0x0024	000000001000000	-272	+37.0C
5549	0xaa5a0c	0x0024	000000001000000	-272	+37.0C
5550	0xaa5a13	0x0024	000000001000000	-272	+37.0C
5551	0xaa5a21	0x0024	000000001000000	-272	+37.0C

Multiplication faults

1st_operand * 2nd_operand = result


Smallest

0x89af


Smallest

0x1


Smallest

0x200000

0x80000 * 0x4

`1st_operand * 2nd_operand = result`

`0x80000 * 0x4 = 0x200000`



`0x4 * 0x80000 = 0x200000`

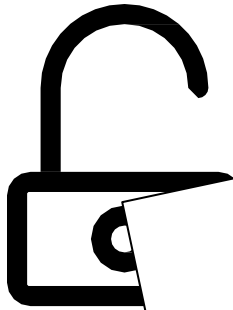


Multiplication faults

Operand 1	Operand 2	Flipped Bits
0xacff13	0x00ee	0x000000003e000000
0xa7fcc	0x0335	0x0000000010000000
0x9fff4f	0x00b2	0x0000000020000000
0x2bffc0	0x0008	0x0000000001000000
0x0b7a04	0x0087	0x0000000004000000
0x080004	0x0008	0xffffffffff000000
0x0022b2	0x6c3a	0x00000000000000700



**You promised
plundering...**



- Public Key Cryptography
- Untrusted channel
- Encryption

Many RSA implementations use the Chinese Remainder Theorem optimisation

```
// Start undervolting  
uint8_t rsa_dec_ecall(int iterations)  
{  
    //Waitforfirstfault  
    trigger_fault(iterations);  
  
    //Actualdecryption  
    ippsRSA_Decrypt(ct, dec, pPrv, scratchBuffer);  
}  
// Stop undervolting
```

```
bagger> dog Enclave/encl
```



And the
pillaging...?

AES-NI (New Instructions)

Instruction	Description
AESENC	Perform one round of an AES encryption
AESENCLAST	Perform the last round of an AES encryption
AESDEC	Perform one round of an AES decryption
AESDECLAST	Perform the last round of an AES decryption
AESENCGEN	Perform an AES encryption generation
AESDECLGEN	Perform an AES decryption generation
AESIMC	Perform AES Inverse Mix Columns
CLMUL	Carryless multiply (CLMUL)

AES can be attacked if you get a fault in the 8th round.


```
// Start undervolting
```

```
do
```

```
{
```

```
    plaintext= <randomlygenerated>;
```

```
    result1=aes128_encryption(plaintext);
```

```
    result2=aes128_encryption(plaintext);
```

```
} while(result1 == result2)
```

```
// Stop undervolting
```

```
bagger> sudo ./aes-encrypt 100000 -262
```

```
|
```



**Hold
tight...**

But what about memory corruption?

```
struct_foo_t *foo = &arr[offset];  
foo->foo = enclave_secret;
```

But what about memory corruption?

`foo = arr + offset`  `0x24`

Creating enclave...

==== Victim Enclave ====

[pt.c] /dev/sgx-step opened!

Enclave Base: 0x7f001a000000



Voltage

0.584V

Undervolting

-235mV



- A new type of attack against Intel
- Breaks the integrity of SGX
- Within SGX
 - Retrieve keys using AES-NI
 - Retrieve RSA key
 - Induce memory corruption in bug free
 - Make enclave write secrets to untrusted memory

Voltage
0.594V

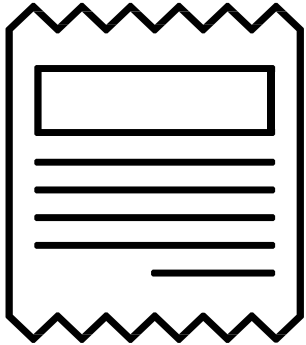
Undervolting
-240mV



Kit Murdock, David Oswald, Flavio D. Garcia,
Jo Van Bulck, Daniel Gruss, and Frank Piessens.

"Plundervolt: Software-based Fault Injection Attacks against Intel SGX".
In S&P 2020

Acknowledgements



This research is partially funded by the Research Fund KU Leuven, and by the Agency for Innovation and Entrepreneurship (Flanders). Jo Van Bulck is supported by a grant of the Research Foundation – Flanders (FWO). This research is partially funded by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/R012598/1, EP/R008000/1, and by the European Union’s Horizon 2020 research and innovation programme under grant agreements No. 779391 (FutureTPM) and No. 681402 (SOPHIA).



Thank you