UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Software-based Undervolting Faults in AMD Zen Processors

Fehler in AMD Zen Prozessoren durch Software-basierte Unterspannung

Bachelorarbeit

im Rahmen des Studiengangs
IT-Sicherheit
der Universität zu Lübeck

vorgelegt von
Anja Rabich

ausgegeben und betreut von
Prof. Dr. Thomas Eisenbarth

mit Unterstützung von
Luca Wilke

Lübeck, den 31. August 2020

Abstract

Dynamic Voltage and Frequency Scaling (DVFS) is a powerful performance enhancement method used by modern processors, allowing them to scale voltage or frequency as needed based on the power requirements of the CPU. This not only saves power, but also prevents processors from overheating. However, the continued integration of software interfaces giving a user direct access to this functionality has been shown to be a potential security risk, allowing a privileged adversary to indirectly tamper with sensitive computations. This thesis summarizes the results of various papers showing that using DVFS features, unsuitable voltage/frequency values can be set for the processor leading to hardware faults and calculation errors which can be used to undermine the integrity of Trusted Execution Environments (TEE). Results are partially replicated for Intel's TEE implementation SGX, followed by extending the same methodology to AMD's Zen Processors, on which there is currently no information. Results show that undervolting is an unlikely attack vector.

Zusammenfassung

Dynamische Spannungs- und Frequenzskalierung (engl. DVFS) ist ein in modernen Prozessoren vorhandener Leistungs- und Stromverwaltungsmechanismus, womit Spannung und Frequenz der CPU je nach Bedarf skaliert werden können. Somit wird nicht nur Strom gespart, sondern auch zusätzlich verhindert, dass der Prozessor überhitzt. Die zunehmende Integration von Softwareschnittstellen zu diesen Mechanismen die dem Nutzer Einstellungsmöglichkeiten anbieten, haben sich zunehmend als potenzielle Sicherheitslücke erwiesen. Ein Angreifer kann möglicherweise Fehler in Berechnungen innerhalb einer Trusted Execution Environment erzeugen. Diese Arbeit fasst mehrere Forschungsergebnisse zusammen, die gezeigt haben dass man mit DVFS Hardwarefehler erzeugen kann, die auch verwendet werden können um die Integrität von vertraulichen Laufzeitumgebungen wie SGX oder TrustZone zu verletzen. Ergebnisse werden teilweise für SGX reproduziert, gefolgt von einer Erweiterung der Methodik auf Zen Prozessoren von AMD, zu denen es noch keine Informationen gibt. Ergebnisse zeigen dass Unterspannung als Angriffsvektor unwahrscheinlich ist.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, 31. August 2020

Acknowledgements

I would like to thank Luca Wilke for our biweekly meetings for his eternal patience and good advice. I would also like to thank everyone at the Institute for IT-Security who kindly took the time to attend my lunch seminar and give me feedback on my research. I thank Kristian Ehlers, who could not make it to the the aforementioned seminar, but took time out of his own busy schedule to listen to my presentation at another time and date. I thank Prof. Eisenbarth, for giving me the opportunity to write my thesis in the first place and for his helpful feedback. I also thank Jan Wichelmann, who also took time out of his schedule to give me advice on how to proceed in my experimentation. I am grateful to Lars B. Vosteen and Jonah P. Heller, for proof-reading and commenting on my thesis. Finally, special thanks to Noni and Lele, who were there for me when I most needed them.

Contents

Glossary

DVFS Dynamic Voltage Frequency Scaling. A technique used by modern processors to ensure that they meet temperature and performance constraints. Allows changing voltage and/or frequency to values that are currently required by the CPU.

TEE Trusted Execution Environment(s). A secured area of a processor, meant to guarantee confidentiality and integrity of any code executed within.

Intel SGX Intel Software Guard Extension. The Intel implementation of a Trusted Execution Environment. SGX also provides enclave functionality.

Enclave A software module that is isolated from the rest of the computing base. An enclave is loaded by a user-level program. Code execution is protected by the CPU. Only defined interfaces may be used to access the enclave.

SMU System Management Unit. A subcomponent of AMD processors from their Bulldozer architecture onwards and situated on the northbridge.

P-State Performance-States. Discrete voltage-frequency pairs that CPUs switch between depending on their current load.

MSR Model-Specific Registers. Registers in the x86 instruction set that allow setting various features of the CPU.

MCA Machine-Check Architecture. A mechanism with which errors such as bitflips are detected and correct by x86 processors. Machine-Check Exceptions (MCE) are thrown and logged by the kernel when such an error occurs.

1 Introduction

1.1 Motivation

Hardware-based vulnerabilities such as Meltdown and Spectre have shown the potential damage that can be done when corners are cut for performance enhancement. The Desktop and Server CPU market is currently still dominated by two major players, Intel and AMD. In addition, x86 is still the standard instruction set used with plenty of historical bloat, while practically all smartphones run on an ARM core. Thus any potential hardware vulnerability found in the products of these three big players comes with grave security implications for numerous devices due to their homogeneous architecture and the sheer size of their userbase.

In particular, the security of Trusted Execution Environments (TEE) such as Intel's Software Guard Extensions (SGX), ARM's TrustZone, or AMD's semi-equivalent Secure Encrypted Virtualization (SEV) is paramount. While these are meant to be environments wherein trusted computing may take place without fear of confidentiality or integrity violations, they also introduce additional attack surfaces; any potential breach of a TEE greatly magnifies the abilities of an adversary, since they are able to execute privileged code.

While Meltdown was simply a bug that could be fixed with a microcode update being rolled out to all Intel CPUs affected, mitigation of Spectre's many variants has proven far more challenging [spe]. Not only was this a mere bug, but a symptom of architectural design decisions made by both manufacturers, in order to produce faster hardware. Thus from a research standpoint, it is worth looking at other potential attack surfaces of x86 processors and in particular, features which may be abused in unexpected ways.

In the past Intel has dominated the CPU market. However with Intel seemingly unable to break the 7nm manufacturing barrier [Ama20] and with the above-mentioned vulnerabilities hitting Intel particularly hard [SLM+19][CGG+19][BMS+20], it is unsurprising to see customers flock to AMD.

One of the newest vulnerabilities found in Intel CPUs are undervolting-based faults. Several researchers were able to show that these can be used to undermine the integrity of SGX. Commonly used encryption algorithms such as AES-NI and RSA-CRT may be faulted while being run inside an enclave, and their keys then derived using differential fault analysis [QWLQ19b][MOG+20]. Induced faults can also be used to alter the control-

flow of code running inside an enclave [KFG+20a].

1.2 Scope

The aim of this thesis is see if AMD Zen processors are vulnerable against hardware faults leveraged using power management interfaces that allow the manipulation of either voltage and/or frequency and to summarise the results of various papers exploiting such hardware faults. As a stepping stone, previous attacks on Intel are reproduced. The aims of this work can be summarised as the following questions:

1. Can we replicate undervoltage-based faults on AMD Zen processors? Do these security issues apply to them as well?
2. Can we also reproduce the results of the above-mentioned papers in Intel processors, in particular, those that undermine the integrity of SGX?

1.3 Organization

Relevant background knowledge and related work is provided in Section 2 explaining the theory behind how such faults or bit-flips can occur. A rundown of different implementations of TEE by different manufacturers is given. The primary defense mechanism against errors such as bit-flips in the x86 instruction set, Machine-Check Architecture, is briefly explained. Section 3 summarises the methodology and results of research papers that leverage either frequency or voltage settings in order to produce faults in ARM as well as Intel processors. Next, results and observations made during the reproduction of faults on a CPU that had not been tested on any of the aforementioned papers is provided in Section 4. Initially, faults in userland are evoked on multiplications, then on AES-NI running inside an SGX enclave. Afterwards, the multiplication testbench used is ported such that it can be run on AMD. Results of various setups that were used are given in Section 5. Finally, the conclusions of this thesis are summarised in Section 6, along with a discussion of the results and open problems.

2 Background

This chapter goes over information pertaining to CPU timing constraints and their relation to voltage, frequency and power. It also briefly explains what a TEE is and implementations that are relevant to the contents of this thesis. Finally, MCA is briefly explained.

2.1 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling or DVFS is a feature of all modern processors. It is meant to be a convenient energy management feature dynamically adjusting the two main factors of how much power a CPU currently requires, voltage and frequency, in order to ensure that the processor is as energy efficient as possible. The following relation holds between power, voltage and frequency:

$$P_t \propto V_t^2 \times F_t$$

where P_t is the instantaneous dynamic power of a CPU, where V_t is the voltage being supplied at time t and F_t the frequency at which the CPU is running at. In other words, power at any given moment t is proportional to the voltage squared provided and the frequency of the CPU cores at any given moment [TSS17][QWLQ19a][MOG+20].
DVFS is usually implemented on both a hardware and software level: hardware regulators exist for both voltage and frequency that communicate with privileged DVFS drivers which in turn can be used by privileged software to regulate voltage and frequency [TSS17][QWLQ19a].
On x86 Processors, Model-Specific Registers (MSR), used for configuring the CPU's operations [CD16, 8], exist which also allow a privileged entity to set and read frequency and voltage values. Reverse Engineering efforts have revealed the undocumented MSR register 0x150 that appears to exist in all Intel processors, although only CPUs from Skylake onwards allow the regulation of their core voltage through this register [MOG+20], which allows setting an offset of the currently supplied voltage [MOG+20][KFG+20a]. Although AMD CPUs also possess MSR which allow setting and reading Performance-State (P-States) values, no documented or undocumented MSR has been found (or even exists) to the best of the author's knowledge that allows setting an offset like in Intel CPU's.
The aforementioned P-States are voltage-frequency pairs which the CPU adheres to as

closely as it can. Ryzen processors, for example, have three with P0 being the highest performing state when the CPU is under heavy load, while P2 is the idle state. Processors will change from one P-State to a higher one if increased performance is required and switch back to a lower one when the load is removed [Int15].

2.2 Timing Constraints

CPU registers are built on sequential logic circuits know as flip-flops or latches [HH13, 193]. Even with modern tricks such as branch prediction and speculative execution, a synchronous clock is nonetheless required to ensure that values are kept coherent. The clock period T_{clk} is defined as the time between two rising edges of a repetitive clock signal [HH13, 142]. Its reciprocal, the clock frequency ($1/T_{clk}$), consequently can be seen to express how any values can be updated or how much work can be done per unit of time. Combinatorial circuits do not produce results instantaneously however, meaning that there is always a certain time delay before an instruction/operation is complete. This delay is made up of several factors: the propagation delay of a flip-flop to its output Q (T_{pd}), the actual time it takes for a computation to finish and propagate to the next latch T_{max_path} and finally the setup time T_{setup} of a flip-flop as the incoming data signal D must be kept stable on an incoming rising edge [TSS17][HH13, 143-144]. If the delay, called the setup time constraint, exceeds that of the clock period, incorrect register values may be read on the subsequent rising edge [HH13, 144]. The total setup time constraint of a circuit can be summarised as

$$T_{clk} \geq T_{pd} + T_{max\_path} + T_{setup}$$

The shorter this timing constraint is, the shorter the maximum delay of consecutive gates can be. Consequently, this delay limits the number of gates that can be placed back-to-back on a circuits critical path [HH13, 145]. The implication is clear: if one can manipulate any of the factors of the equation, one can intentionally induce a fault. Two possibilities present themselves: decreasing T_{clk} by increasing the frequency, i.e. overclocking the CPU or by shortening any of the factors on the right-hand side. As it isn't possible to change either T_{pd} or T_{setup}, which are dependent upon hardware implementation and physical limits, this only leaves the circuit propagation time T_{max_path}. Although this duration is also fixed, one can make it harder for the underlying transistors in the combinatorial circuits to maintain their correct values for this duration by undervolting [TSS17]. The critical path of the circuit in question must also be long enough for such a fault to occur [MOG+20].

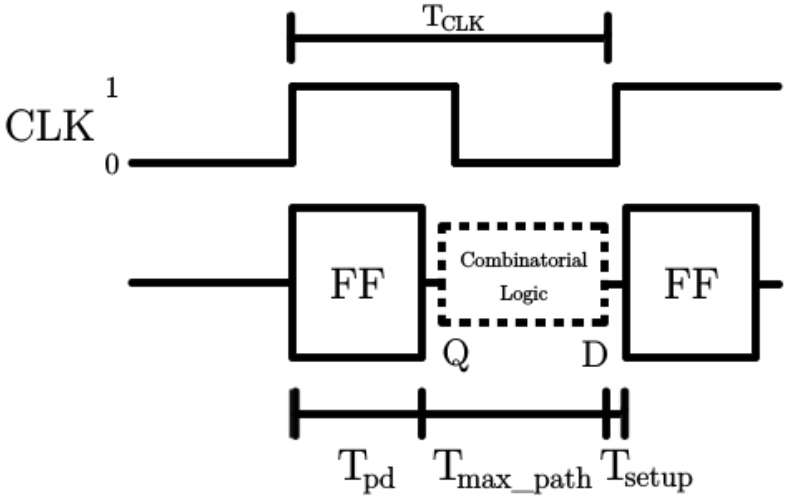

Figure 2.1: Rising edge to rising edge timing constraint

2.3 AMD System Management Unit

In order to manipulate either frequency or voltage values given to the CPU, a suitable interface must be identified first. Both Intel's Extreme Tuning Utility and AMD's Ryzen Master utility allow users to set their CPU's frequency and voltage values for performance enhancements. Much of their functionality is not specified in documentation. While the aforementioned MSR have been identified in Intel processors, currently known voltage altering mechanisms in AMD must go through the System Management Unit (SMU).
The SMU has been present in AMD Processors since its *Jaguar* (16h) and *Bulldozer* (15h) architecture [Mar14, 7]. Information is rather scant in their official documentation however, the official BIOS and Kernel Developer's Guides for families 15h provides the following description:

> "The system management unit (SMU) is a sub-component of the northbridge that is responsible for a variety of system and power management tasks during boot and runtime. The SMU uses two blocks, System Management Controller (SMC) and Platform Security Processor (PSP), in order to assist with many of these tasks. At the architectural level, PSP is known as MP0 and SMC is known as MP1" [AMD18]

In later documentation, the PSP (simply SP in newer Processors) is given its own section [AMD16], leaving only the first sentence. The SP appears to be an ARM Cortex chip with

TrustZone [Mar14, 4]. Both Chips appear to have been taken over in Zen architecture. The Secure Processor offers key generation and management (see Section 2.4.1). How one passes commands to the SMU is described in Section 5.

2.4 Trusted Execution Environments

Trusted Execution Environments aim to guarantee the confidentiality and integrity of computations, with minimal additional hardware while also maintaining performance as much as possible. Different manufacturers provide somewhat different implementations and functionality. Since TEE allow privileged code execution on a system, any vulnerability affecting them comes with dire consequences. As later described in Section 3, several vulnerabilities also affect the integrity of TEE.

2.4.1 AMD SEV & SME

Secure Memory Encryption (SME) uses a key to encrypt system memory with AES-128 [AMD] before committing it to RAM [Kap16][KPW16, 4]. The key is generated and later managed by the Secure Processor on boot [AMD] (refer to Fig. 2.2). SME aims to prevent cold boot attacks and snooping on DRAM [KPW16, 5]. On the other hand, SEV aims to isolate virtual machines from a potentially malicious hypervisor using encryption (see Fig. 2.3). Here, each virtual machine is given its own key, which is also managed by the Secure Processor. These are features currently only supported by AMD EPYC processors.

2.4.2 Intel SGX

Software Guard Extension (SGX) provides secure execution environments known as enclaves. Developers can define trusted and untrusted code to be executed using EDL.
An adversary cannot simply read out values in RAM belonging to an enclave as these sections are encrypted. The integrity of data outside of the CPU package is also checked. Any errors lead to a CPU halt. As we will see in Section 3, this does not pose a problem for undervolting-based faults.

2.4.3 ARM TrustZone

ARM's version of a TEE splits the execution environment into a trusted/secure and a non-trusted/non-secure domain, sometimes also referred to as worlds, on ARM Cortex-A and Cortex-M chips [tru], which are implemented as individual virtual cores running on a physical core [QWLQ19a], similar to SGX [TSS17]. One may switch between them using

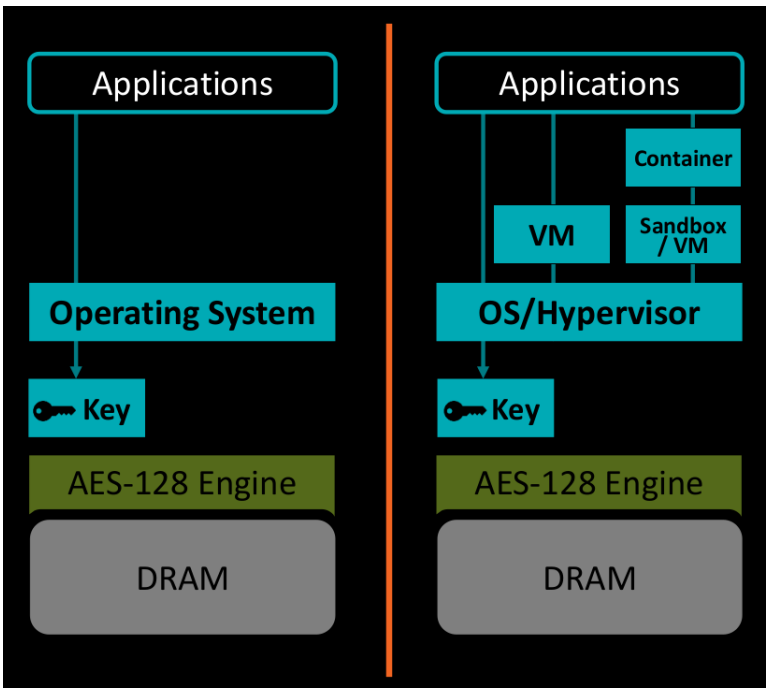

Figure 2.2: Diagram of SME, showcasing how a single key is used to encrypt memory. Figure taken from [Kap16, 7]

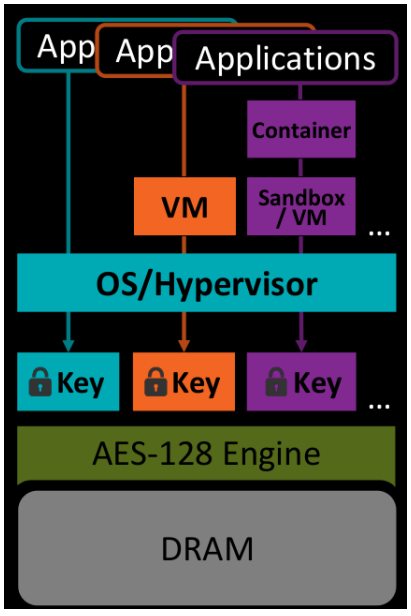

Figure 2.3: Diagram of SEV showing its structure during usage. Each container or virtual machine is given its own encryption in order to protect the guest system from an untrusted hypervisor or administrator. Figure taken from [Kap16, 9]

interrupts [tru]. Software that is meant to be run in the secure world must be authenticated first [QWLQ19a].

2.5 Machine-Check Architecture

With sufficiently long hardware uptime, errors are expected to occur simply due to exterior influences such as electromagnetic radiation or heat. One of the mechanisms for detecting and correcting such errors is the Machine-Check Architecture (MCA), first introduced by Intel in their Pentium processors. MCA monitors the CPU hardware in real-time and logs any detected errors to a special set of registers [KFG+20a, 3]. If an uncorrected hardware error occurs, a Machine-Check Exception (MCE) is raised by the CPU, the interpretation of which may be handled by the operating system [KFG+20a, 4][mceb]. On Linux, *mcelog* or *rasdaemon* may be used for displaying errors that are logged by the kernel.

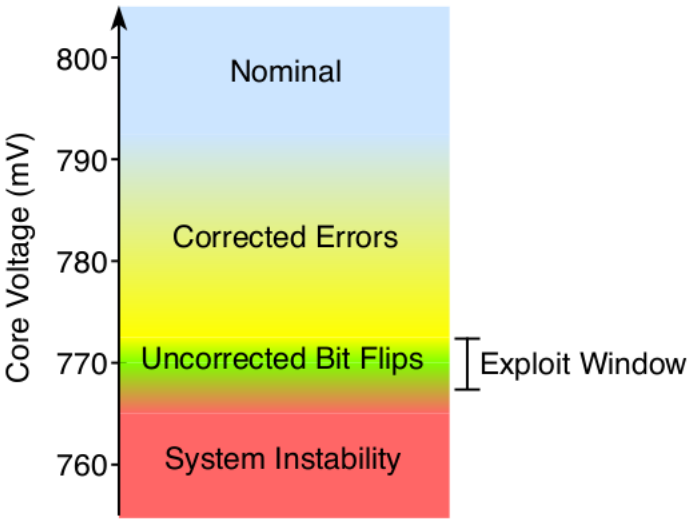

Figure 2.4: Example of how a Processor responds to reducing its voltage. Figure taken from [KFG+20a, 7].

3 Faulting Modern CPUs

The following section summarises the methodology and results of papers using DVFS as an attack vector in order to inject faults into CPU cores.

3.1 Frequency-based Faults

3.1.1 Clkscrew

Tang et al. showed in their paper that it is possible to induce faults in ARM Krait cores, found in several Android smartphones, using overclocking functionality. They were able to show that such faults can be used to undermine the integrity of ARM TrustZone and showed that cryptographic keys can be recovered from within the TrustZone. Since their method is based on overclocking, it decreases the left-hand side of the Equation given in Section 2.2.
The scenario they pictured is the following: an attacker wishes to induce errors in a victim's code execution, such as encryption operations running inside TrustZone, the keys of which can then be inferred using differential fault analysis. Tang et al. made the observation that hardware regulators operate on a core-basis, meaning that even enclaves can be affected by voltage or frequency changes.
To ensure that the attack process isn't affected by the frequency manipulation itself, the victim and attacker process are delegated to their own cores. ARM's big.LITTLE architecture allows having cores run on different voltage and frequency domains, meaning that the attacker core can run on a stable frequency, while the victim core is overclocked [TSS17].

Their methodology can be broken down into the following steps:

1. Residual states are cleared in the cache by evoking the victim and attack thread in quick succession several times.

2. A timing anchor, just before the execution of the target code, must be identified. This may be done using data cache profiling techniques.

3. A wait loop may be added in order to fine-tune the exact delivery time of the fault.

4. The attacker thread delivers the fault by raising the victim core's frequency, holding that frequency for a set duration of time and then restoring the initial frequency.

3.2 Voltage-based Faults

Several research groups independently discovered that the integrity of SGX enclaves can also be undermined using undervolting-induced faults. Their methods and findings are briefly summarised here. A commonality of all three is the usage of MSR to set a voltage offset of what the CPU would normally run on. The findings of all papers have been aggregated into the CVE-2019-11157 [Int].

3.2.1 Voltjockey

In their paper introducing the vulnerability named "Voltjockey", Qiu et al. initially also looked at ARM's Krait architecture. Instead of manipulating the frequency, which has the drawback of requiring that overclocking is enabled (something that is rather uncommon in mobile devices) [QWLQ19a], voltage is used. Similar to Clkscrew, it is assumed that a victim process runs on a single core, with the attacker process being run on a separate core. The process is as follows

1. The victim core must be set to a suitably high frequency value, while all other cores are set to reasonably low frequencies. Residual states in the cache must be cleared.

2. The attacker procedure waits for the victim procedure to enter the point where the fault is to occur.

3. The attacker procedure delivers a low voltage to the entire processor. Proper duration and voltage level require experimentation to be found.

4. Normal voltage is restored such that the victim core can resume its normal operations with the erroneous result.

Their reasoning for setting a high frequency to the victim core and low frequencies to all others (in particular the attacker core) is to ensure that only the victim core malfunctions. This can be done because voltage is regulated for all cores while frequencies may be set individually for ARM CPUs [QWLQ19a].
This was further expanded to Intel SGX, with the methodology remaining largely the same.

3.2.2 V0ltpwn & Plundervolt

The underlying test procedure of V0ltpwn and Plundervolt is based on Algorithm 1. Clearly this should never terminate as the value of var should always be the same inside and outside of the loop. However, if voltage is sufficiently lowered before entering the loop, the multiplication can be made to return an erroneous result and the loop will terminate.

Algorithm 1: Infinite loop

```
1 a = 0xcafec0de;
2 b = 0xdeadbeef;
3 var = a * b;
4 while var == a * b do
5 |   var = a * b;
6 end
```

Both papers explicitly mention and use the undocumented 0x150 MSR (Fig. 3.1) in order to set the voltage offset. The authors of V0ltpwn first disable drivers that automatically configure P-States and other power settings. A P-State is then manually set by writing to the MSR $0x774$ in order to have all cores running at the same initial voltage and frequency. In order to make the attack more reliable, several other adjustments are made:

1. Logical cores are partitioned into a attacker group, containing a single core, and a victim group, with all other cores. Running processes are then delegated only to the attacker core. This is done to minimize noise of the victim cores and to allow individual probing of the fault-prone voltage level, since voltage values cannot be set per core like in ARM.

2. Thermal-based interrupts are disabled in order to prevent hardware interference.

Finally the test program is run on the victim core that is being probed while the attacker core sets the voltage. As temperature also plays a potential role on CPU performance, one can stress the core until the desired temperature is reached on the victim core or an additional stressor section is added to the victim core's logical partner to further achieve an unstable state in the victim core. Although this isn't strictly necessary, Kenjar et al. found that this improved the likelihood of a fault [KFG+20a].
Murdock et al. take a slightly less fine-tuned approach to producing faults in their Plundervolt paper. The CPU frequency is first set to a fixed value. Voltage is then incrementally dropped to a set maximum offset from the starting voltage, while multiplications are run concurrently. Testing is halted if a fault has been found or if all set iterations have been

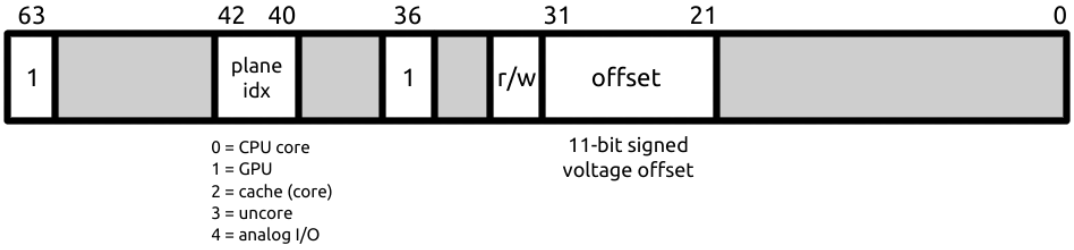

Figure 3.1: MSR 0x150 layout

run through. Normal operating voltage is then restored. Additionally, Murdock et al. found that only instructions with a sufficiently long critical path, such as multiplications, could be faulted, while operations such as addition, subtraction and division, could not [MOG+20].

The focus of both papers diverges from here: Plundervolt shows how encryption schemes such as AES-NI and RSA-CRT can be faulted inside an enclave, the keys of which then recovered with differential fault analysis, while V0ltpwn shows how the bit-flips that occur during faults can be used to hijack control-flow of a program running inside an enclave.

In both cases, the consequences of undervolting are a violation on the integrity of calculations within SGX enclaves. As these faults occur within the CPU package, SGX's memory integrity check does not detect that anything is wrong. Kenjar et al. also note that such faults lead to non-recoverable errors, meaning that MCA cannot correct them [KFG+20a, 4]. They observed that when the voltage was decreased, the processor would start to experience errors that were still correctable by MCA. When voltage was further decreased, the system eventually reached an unstable state where hardware exceptions occur. Between these two states lies a window for uncorrectable faults that remain undetected by MCA, just before the system reaches an unstable state [KFG+20a, 7-8] (refer to Fig. 2.4).

Another key difference is that Plundervolt focuses on mobile processors found in Intel NUCs, while results for V0ltpwn were collected on high-performance i7 processors.

4 Replicating Undervolting Faults in Intel processors

As Plundervolt is, as of writing of this thesis, the only paper to provide proof-of-concept code [1], this was used as an initial point of reference. Results were replicated on an Intel NUC 7i5BNK, with a i5-7260u running Ubuntu 19.10. This CPU was not tested by either paper, however fault-prone voltage-frequency pairs between those of an i3 and an i7 are to be expected.

4.1 Replicating Multiplication Faults

Using the provided PoC code, the following was reproduced on the NUC. First the frequency of the CPU is set to a fixed value. This can be done with $cpupower$ command on Linux. Next, a multiplication thread based on Algorithm 1 is executed while the voltage is decremented in a separate thread. Multiple multiplication threads may be run. Although this is strictly not necessary, it increases the likelihood of a fault as we can test multiple cores at the same time and stop when we have found a fault (Algorithm 1 terminates immediately when one is found). Random integers are generated and used as operands for each multiplication thread.
The following behaviour that was observed by Murdock et al. could be confirmed:

- Erroneous results were either the most significant bits being flipped ($0x0000\ldots \rightarrow 0xffff\ldots$) or incorrect results in the 8-10th position of the hexadecimal result (see Table 4.2).
- The vast majority of faults were a single bit flip in the 8th position. It is possible that the observed larger bit flips are from incorrect carry overs.
- Multiplication errors only ever occurred when the first operand was larger than the second, never the other way around.

The lower the frequency, the harder it became to produce a fault reliably. Either a larger product of the two operands or a larger voltage offset was required to produce a fault. This is in line with the results of both Plundervolt and V0ltpwn. The authors of V0ltpwn reason that it becomes harder to undervolt at lower frequencies because the processor

[1] `https://github.com/KitMurdock/plundervolt`

simply requires less power to run. However, their method using P-States only produced faults when said P-States were between 2.7GHz and 3.6GHz [KFG+20a, 10].
One drawback of specifying offsets is that the same voltage ends up being tested multiple times, making decrements non-linear. Presumably, this occurs because the actual voltage that the cores would normally running on is scaled upwards and one can only define an offset relative to this value using MSR 0x150. This can be seen in Fig. 4.1 and in Fig. 4.3, where during testing the same absolute voltages are tested despite incrementing the offset.
However, using both *rasdaemon* and *mcelog* no Machine Check Exceptions were recorded while running the multiplication testbench, as well as by replacing the multiplication operation with the SIMD inline assembly operation.
While faulting using the Plundervolt proof-of-concept code on the Intel NUC, a total system crash was never observed, only a freeze when a certain threshold was reached. During the collection of crash point data, two types of system failures were observed: an initial freezing of the display where the NUC still showed activity (indicated by its lights), followed by an crash and self-reboot. The other, far more common occurrence, was a system freeze where the NUC showed no further activity and which required a manual reboot.
For the fault values in the graphs given in Section 4.1.1, four multiplication threads were run per frequency such that all cores are used.

4.1.1 Results

	Crash (mV)		Fault (mV)	
Frequency (GHz)	**Voltage**	**Offset**	**Voltage**	**Offset**
1.2	506	-194	519	-194
1.3	521	-192	528	-174
1.4	531	-186	543	-167
1.5	546	-175	558	-164
1.6	555	-166	572	-155
1.7	570	-157	586	-146
1.8	584	-150	601	-139
1.9	599	-150	610	-142
2.0	612	-152	629	-140
2.1	628	-150	648	-140
2.2	643	-148	658	-139
2.3	671	-144	677	-141
2.4	675	-144	696	-137
2.5	700	-146	710	-135
2.6	715	-144	729	-137

Table 4.1: Recorded voltage and offsets at various frequencies for a system crash and for a fault using the testbench provided by the Plundervolt paper.

Operand1	**Operand2**	**Faulty result**	**Flipped Bits**
0xdd551	0x9f	0xffffffff8977d4f	0xfffffffff0000000
0x26a5fbe	0xb9e5	0x000001c0e857f2f6	0x00000001e0000000
0xca2fb31	0xc1ba	0x0000098fedca729a	0x0000001fe0000000

Table 4.2: Examples of faulted multiplications at various frequencies

4.2 Replicating SGX Faults

Using the Plundervolt testbench, faults within an SGX enclave on AES-NI were also reproducible (refer to Fig. 4.4 and 4.5). Similar observations as those made by Murdock et al were made: the multiplication testbench is far more reliable in terms of producing a bit-flip, with a fault being produced with at least 5 runs of the enclave with 100000 iterations per run.

The authors of V0ltpwn claim that their most vulnerable piece of code was the following [KFG+20a, 7-8][KFG+20b]

Listing 4.1: Vulnerable Instructions

```
1 vpxor %xmm1, %xmm2, %xmm3
2 vmovdqu %xmm3, (%rsp)
```

Which is a parallel SSE/AVX logical operation, such as a bitwise XOR, followed by a move instruction [KFG+20a, 7-8]. This is also what is used by the Plundervolt testbench in their AES-NI implementation [do]. The AVX2 intrinsic macros used in Listing 4.2 in line 4 maps to the instruction *vpxor* and in line 25 to *vmovdqu* (provided that the avx2 flag is set when compiling):

Listing 4.2: Partial functionality of the AES-NI implementation used in the Plundervolt testbench

```
1  #define DO_ENC_BLOCK(m, k)                    \
2          do                                        \
3          {                                         \
4                  m = _mm_xor_si128(m, k[0]);          \
5                  m = _mm_aesenc_si128(m, k[1]);       \
6                  m = _mm_aesenc_si128(m, k[2]);       \
7                  m = _mm_aesenc_si128(m, k[3]);       \
8                  m = _mm_aesenc_si128(m, k[4]);       \
9                  m = _mm_aesenc_si128(m, k[5]);       \
10                 m = _mm_aesenc_si128(m, k[6]);       \
11                 m = _mm_aesenc_si128(m, k[7]);       \
12                 m = _mm_aesenc_si128(m, k[8]);       \
13                 m = _mm_aesenc_si128(m, k[9]);       \
14                 m = _mm_aesenclast_si128(m, k[10]); \
15         } while (0)
16
17 static __m128i aes128_enc(__m128i plaintext)
18 {
19         uint8_t pt[16] = {0};
20         uint8_t ct[16] = {0};
21         __m128i m = plaintext;
22
23         DO_ENC_BLOCK(m, key_schedule);
24
25         _mm_storeu_si128((__m128i *)ct, m);
26
27         return m;
28 }
```

Algorithm 2: Pseudocode of AES-NI Encryption run inside the SGX enclave

```
load_aes_key_schedule()
drop_voltage()
while result1 == result2 do
    plaintext = gen_random_plaintext()
    result1 = aes128_enc(plaintext)
    result2 = aes128_enc(plaintext)
end
restore_voltage()
```

Within the enclave, Algorithm 2 is run (the iteration parameter is checked in the actual implementation, such that the while loop will terminate if nothing is found).
In their Usenix presentation video, V0ltpwn co-author Tommaso Frassetto conjectures that the observed faults in AES-NI by Murdock et al. are in fact caused by Listing 4.1 [KFG+20b]. This contradicts what the authors of Plundervolt claim in their paper however, as they state that single bit-flips would occur, always on the leftmost two bytes in an encryption round [MOG+20, 7].
Attempting to reproduce faults on the NUC 7i5BNK by simply running the two instructions of Listing 4.1 within an enclave was also unsuccessful on a modified testbench from Plundervolt. Additionally, the faults produced as shown in Figures 4.4 and 4.5 suggest that faults occurred within an actual AES encryption round. Furthermore, Murdock et al. show in their paper that one can find the key used with differential fault analysis, for which it is most advantageous if the fault occurred during the 8th round of encryption (this is due to the cipher's structure, a more detailed explanation of which is outside of the scope of this thesis). If faults were induced merely by the instruction pattern in Listing 4.1, this would not have been possible.

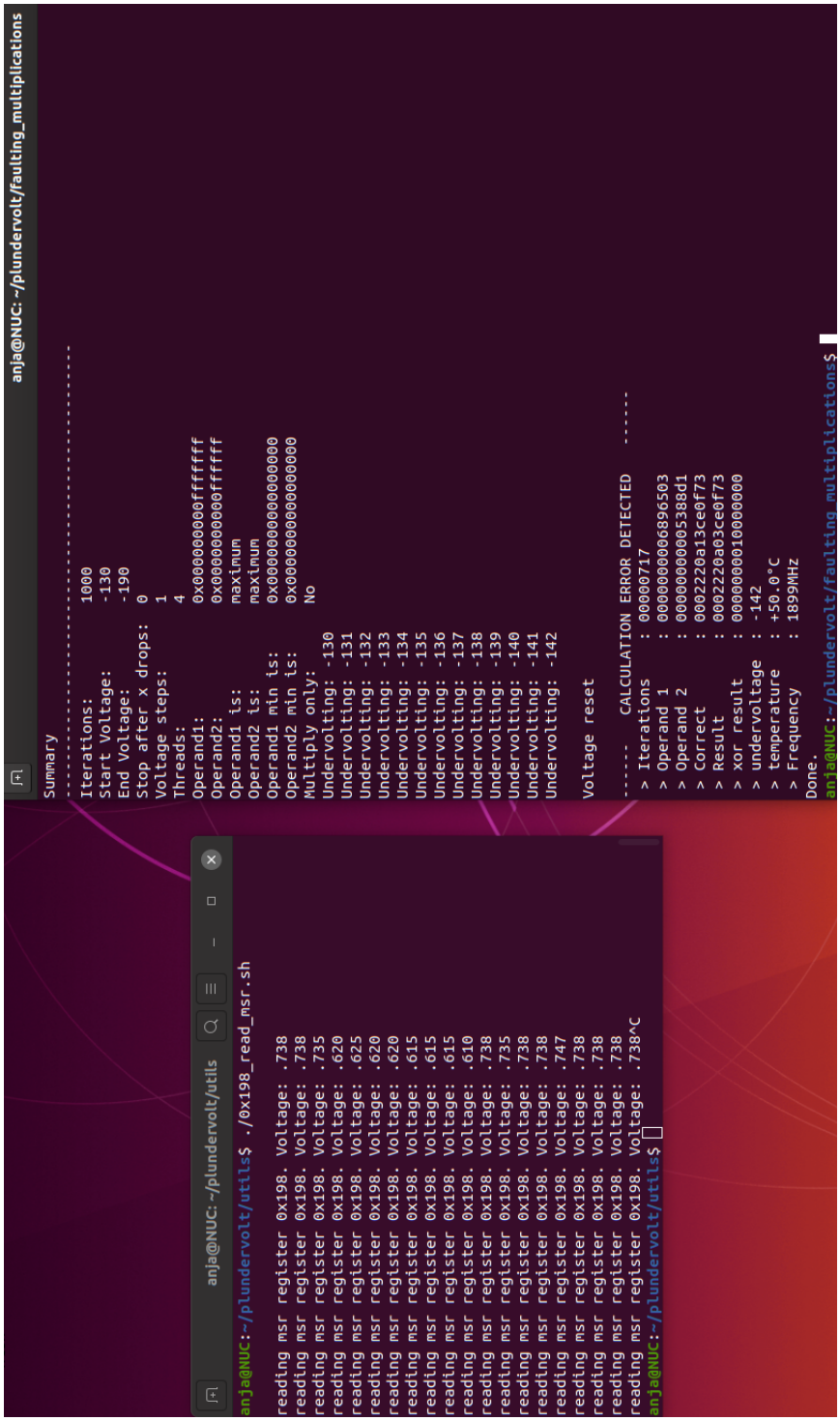

Figure 4.1: A screenshot showcasing the methodology for collecting data. The left window reads out the current voltage from the MSR *0x198*. The starting point and termination of the undervolting testbench are noticeable from the sudden voltage drop and gain.

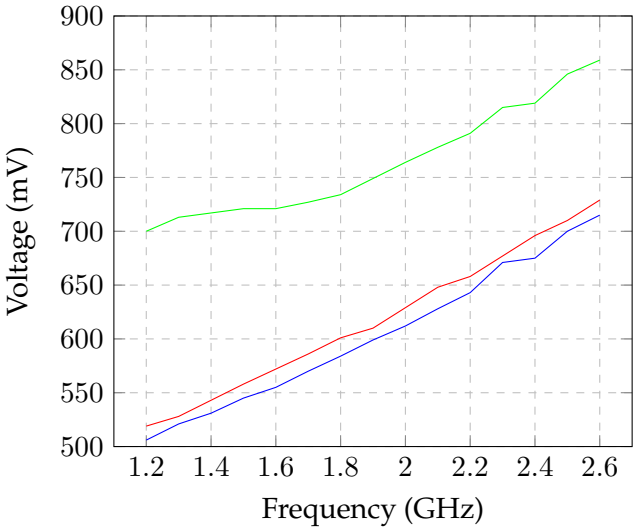

Figure 4.2: The red line shows the voltages at which a fault first was observed, while the blue line shows voltages at which a crash was observed for various frequencies, refer to Table 4.1 for values. The green line is the nominal voltage the system would normally run at.

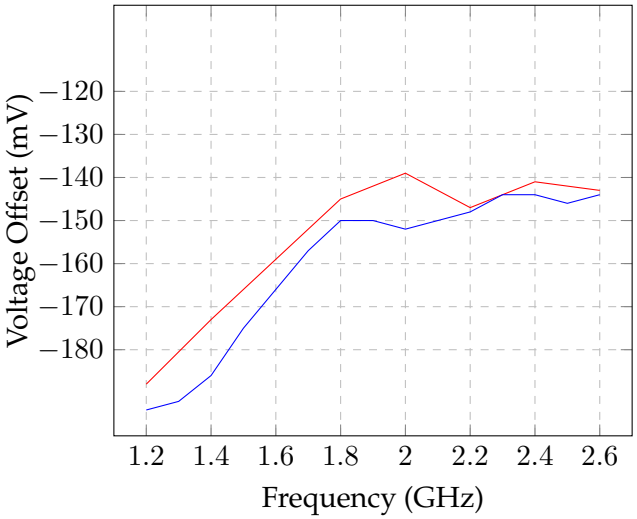

Figure 4.3: The red line shows the average voltage offset for first fault at different frequencies, while the blue line the voltage offset for a system crash, refer to Table 4.1 for values.

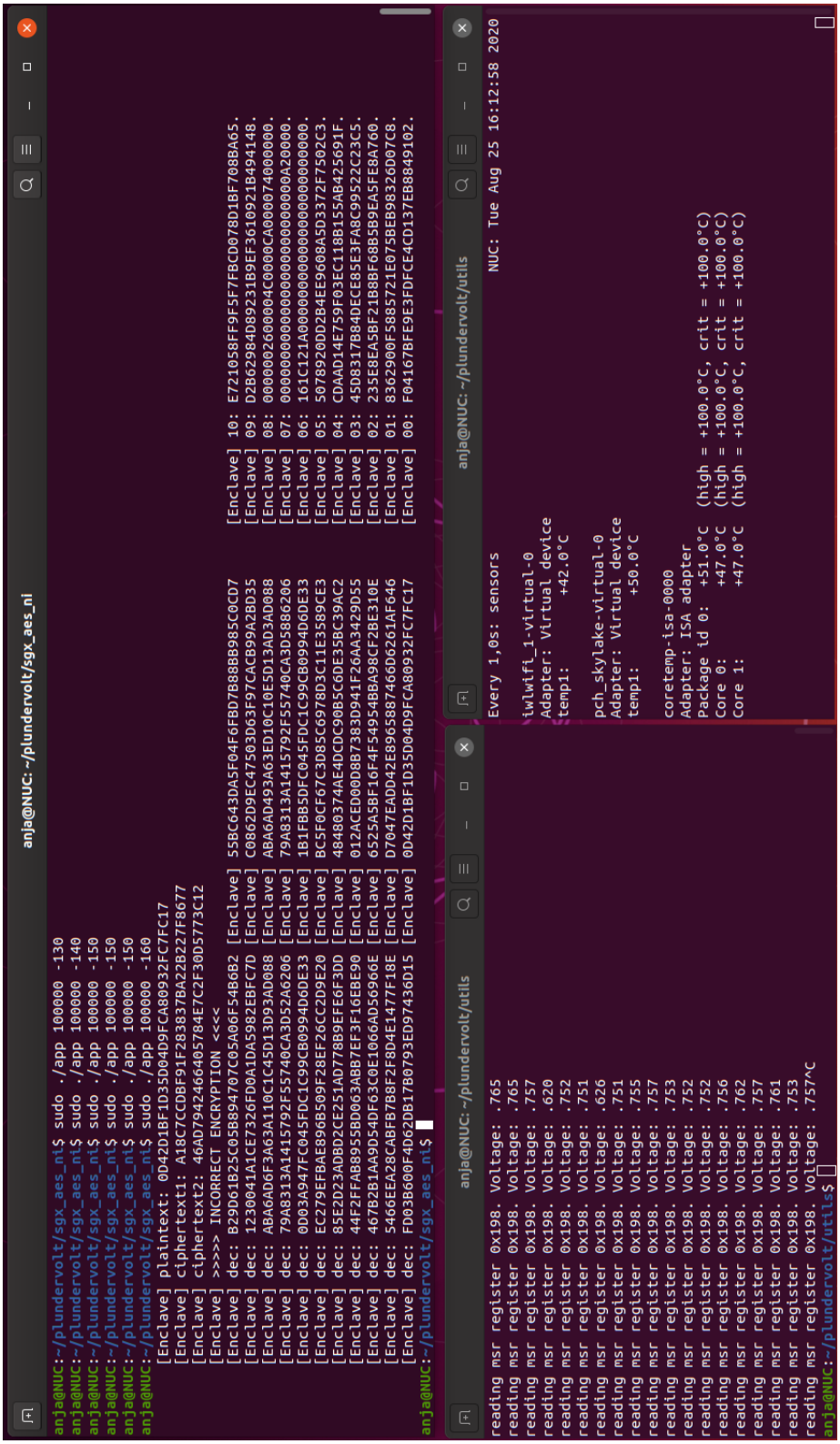

Figure 4.4: Erroneous decryption of AES-NI running inside an Enclave, with 0x000102030405060708090A0B0C0D0E0F. The fault was produced after 6 runs with 100000 iterations each at 2GHz. The maximum voltage offsets used per run can be seen in the screenshot.

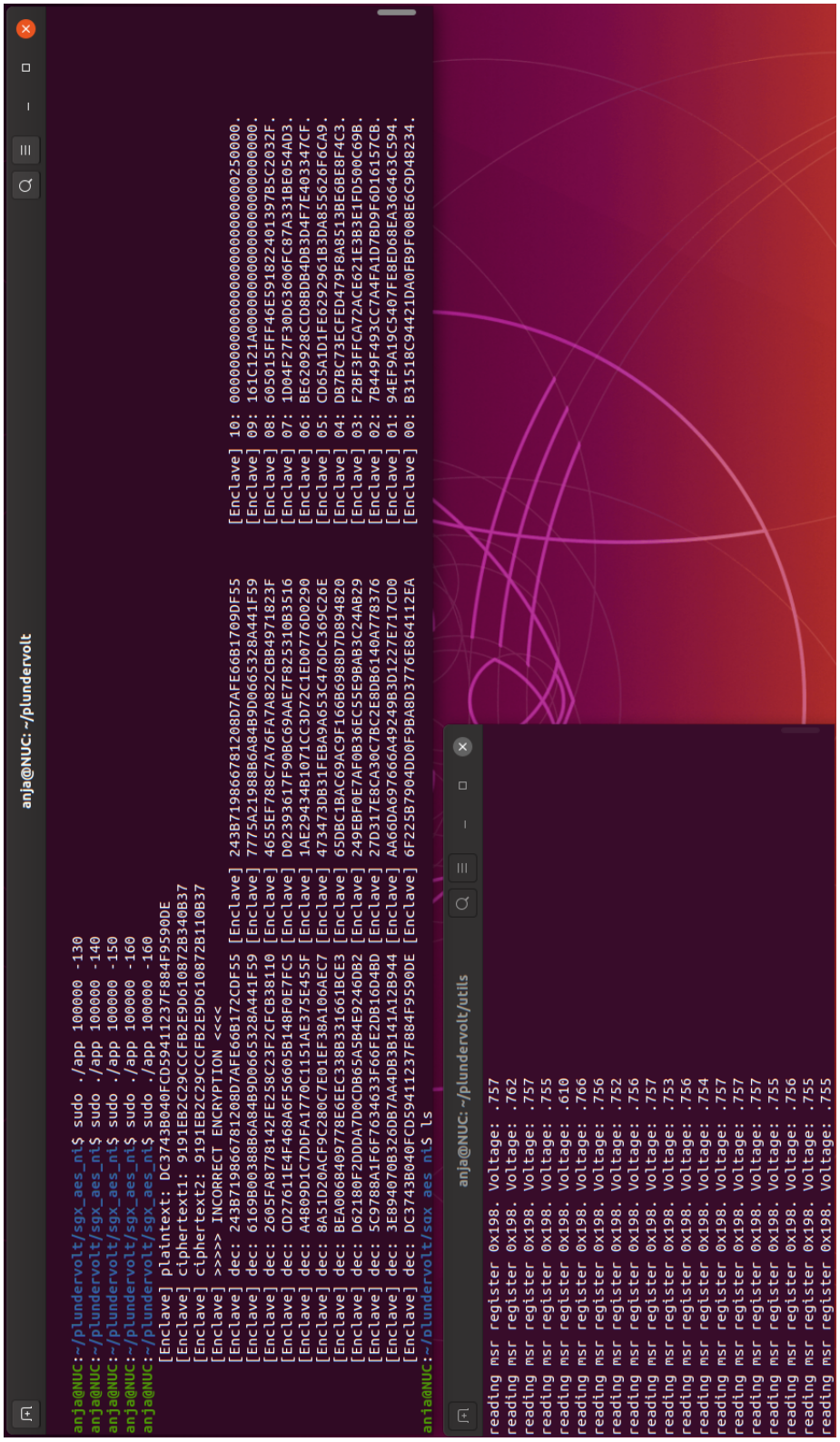

Figure 4.5: Another erroneous decryption of AES-NI running inside an Enclave, with 0x0D0E0A0D0B0E0E0F0C000D0E0C0A0F0E as the key. The fault was produced after 5 runs with 100000 iterations each at 2GHz. The maximum voltage offsets used per run can be seen in the screenshot.

5 Undervolting AMD Zen processors

5.1 Requirements

In order to replicate undervolting-based faults several conditions must be met:

- An interface for setting adversary specified voltage and frequencies must be present. Such functionality is provided by the SMU. Although P-States also leverage DVFS, they are not directly applicable in continuously decrementing voltage. Attempting to do so results in the set voltages within the designated MSR to be ignored.
- The adversary must have root permission on the system because access to hardware and privileged areas of the filesystem is required. This is required in order to read and write values in MSR as well as to PCI.

5.2 EPYC Server

A significant challenge is that overclocking is uncommon on server-grade hardware and most motherboards don't support it. Simply executing the SMU testbench produced no change in the voltage of the CPU. Whether the SMU responds to commands may depend on its firmware version. Attempting to loop over P-States did not work either, as the voltage values are not honored when they are set too low.

5.3 Undervolting Ryzen processors

Although Ryzen processors do not possess features providing confidentiality and integrity similar to SGX enclaves, attempting to produce faults in them is a good starting point before moving on to server hardware. It is important to note that setting the voltage and frequency using the SMU OC Mailbox requires BIOS and motherboard support. Testing showed that if overclocking is not supported, any commands sent are simply ignored.

5.3.1 Communication with the SMU

Reverse Engineering efforts of Rudolf Marek revealed how one can communicate with the SMU via PCI registers, which consist of an address register and a data register [Mar14, 13].

Reading a value from the SMU involves first writing the desired address to the address register, then reading the value that is placed in the data register. Writing to the SMU is done by first writing to the address register, then to the data register with the desired value [cem17][Gha16].

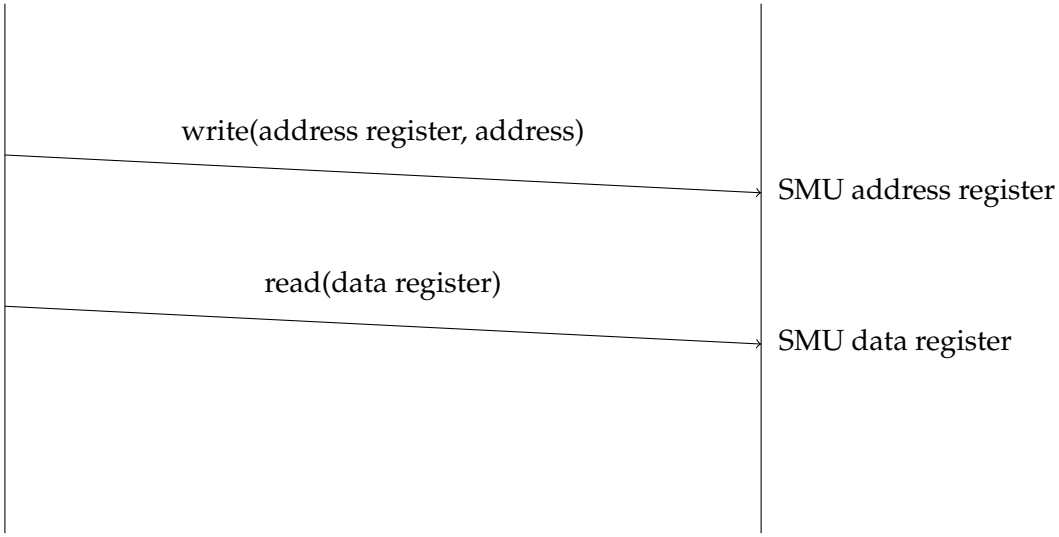

Figure 5.1: Reading from the SMU address space

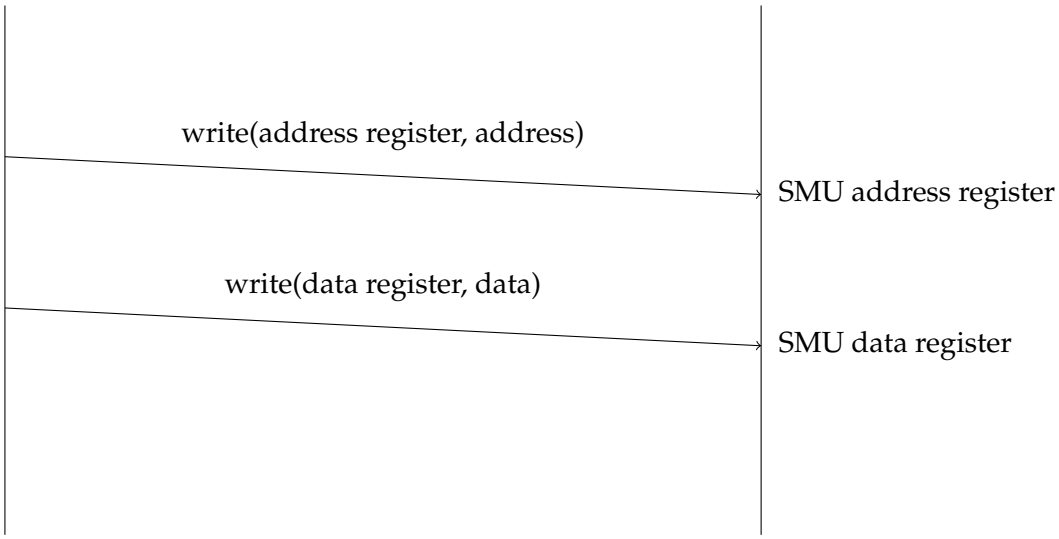

Figure 5.2: Writing to the SMU address space

Access to the SMU goes through the System Management Network (SMN), which has its own address space in PCI. These addresses vary across Zen generations and are not the same across Zen1 processors, but appear to have been unified in Zen2 [git19].

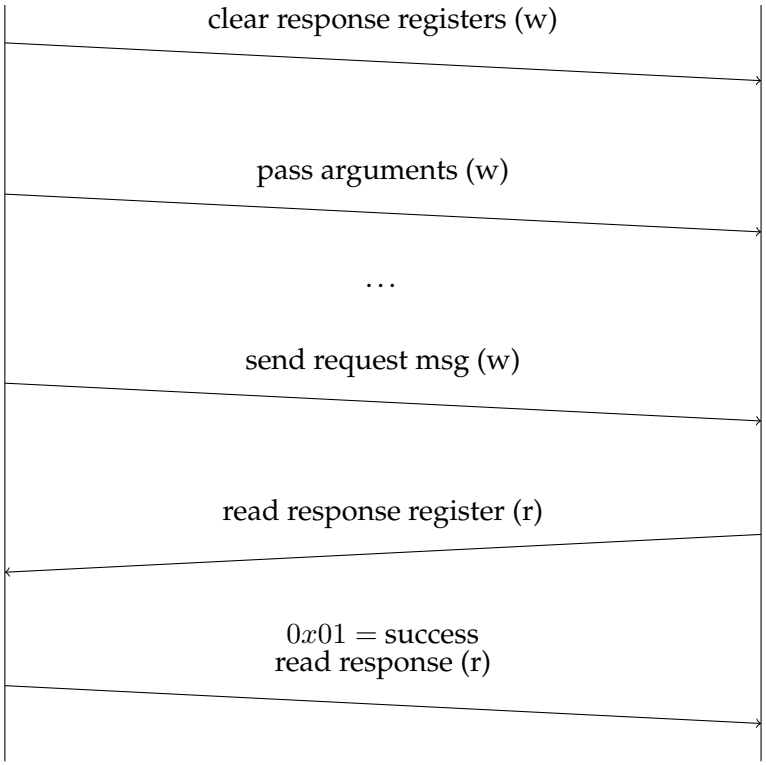

Figure 5.3: Sending a Request to the SMU

Using functions from the pci.h Header (from the libpci-dev package), they can be implemented as follows:

```
u32 smu_service_req(smu_t smu, u32 id, smu_service_args_t *args){
        u32 response = 0x0;

        /* Clear the response */
        smn_reg_write(smu->nb, smu->rep, 0x0);
        /* Pass arguments */
        smn_reg_write(smu->nb, C2PMSG_ARGx_ADDR(smu->arg_base, 0), args->arg0);
        smn_reg_write(smu->nb, C2PMSG_ARGx_ADDR(smu->arg_base, 1), args->arg1);
        /* Send message ID */
        smn_reg_write(smu->nb, smu->msg, id);
        /* Wait until reponse changed */
        while(response == 0x0) {
                response = smn_reg_read(smu->nb, smu->rep);
        }
        /* Read back arguments */
        args->arg0 = smn_reg_read(smu->nb, C2PMSG_ARGx_ADDR(smu->arg_base, 0));
        args->arg1 = smn_reg_read(smu->nb, C2PMSG_ARGx_ADDR(smu->arg_base, 1));

        return response;
}

u32 smn_reg_read(dev nb, u32 addr){
        u32 rep = 0;

        /* write the address of the register that will be read */
        pci_write_long(nb, NB_PCI_REG_ADDR_ADDR, (addr & (~0x3)));
        /* read the value in the data register */
        rep = pci_read_long(nb, NB_PCI_REG_DATA_ADDR);

        return rep;
}

void smn_reg_write(dev nb, u32 addr, u32 data){
        /* write the address of the register that will be written to */
        pci_write_long(nb, NB_PCI_REG_ADDR_ADDR, addr);
        /* write the value in to the data register */
        pci_write_long(nb, NB_PCI_REG_DATA_ADDR, data);
}
```

The 2nd argument is strictly not necessary in the SMU service request as all commands used in undervolting experiments only require one argument. Up to 6 arguments can be passed if required by the command sent.
For simplicity's sake, the initial setup used is similar to that of Plundervolt (Alg. 3 and Alg. 4 are run concurrently):

Algorithm 3: Multiplication thread

```
1 while (true) do
2     var = op1 * op2
3     while (var == op1 * op2) do
4         var = op1 * op2
5
6
7
8     end
9 end
```

Algorithm 4: Undervolt thread

```
1
2
3
4 enable_OC()
5 set_frequency()
6 set_voltage(volt)
7 volt−−
8
9
10 disable_OC()
```

Several differences ought to be noted: At the time of writing, there is no known way to directly set offsets to the CPU cores. Instead voltage values are directly set. Reverse engineering efforts from various individuals in the open-source community have revealed the location of the MSR for reading out the current voltage being supplied to the CPU[23]. This is used as the starting voltage. The desired offset is then subtracted from this starting value. Additionally, voltages cannot directly be written to the SMU, but are expressed as discrete values known as VIDs, with $0xFF$ being the minimum possible value, while $0x00$ stands for the highest possible value (1.55V). The scaling is as follows:

```
1 u32 voltage_to_vid(u32 voltage){
2         return (((1550 - voltage) * 100) / 625);
3 }
4
5 u32 vid_to_voltage(u32 vid){
6         return (1550 - ((vid * 625) / 100));
7 }
```

A single increment or decrement in the VID corresponds to a respective increment or decrement of 6.25mV. For comparison, Intel CPU VIDs are in steps of 5mV [MOG+20, 3].

[2] https://github.com/flygoat/ryzen_nb_smu
[3] https://github.com/irusanov/ZenStates-Linux

Unlike in Plundervolt, voltage doesn't have to be restored as disabling overclocking resets voltage and frequency to the current P-State.

5.3.2 Results

One drawback of using the manual OC functionality is that the SMU appears to adjust frequency and voltage by itself: during testing it was observed that if the voltage set was too low for the specified frequency, the set frequency would be ignored and the actual CPU frequency scaled downwards in order to accommodate the voltage (refer to Graph 5.4 for an example). The *cpupower* utility does not appear to support Zen at the moment, meaning that we cannot simply set the frequency to a desired value, additionally, it is unclear if this option wouldn't simply be overridden by the OC commands.

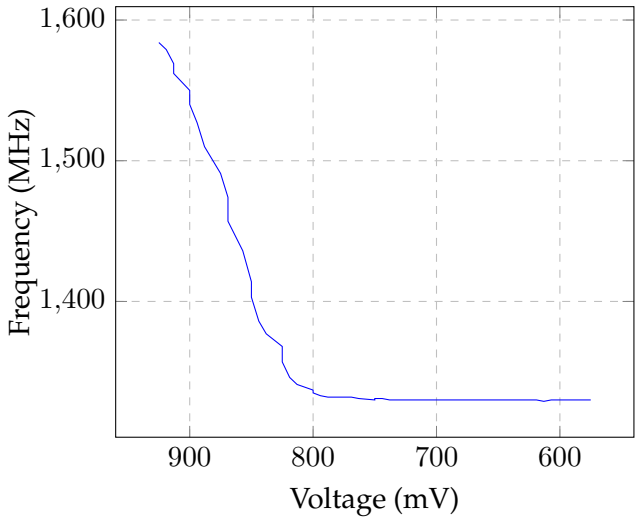

Figure 5.4: Observed frequency drop while decrementing the voltage and sending the same frequency of 1600MHz.

Experiment 1: recreating Plundervolt

Results for the initial setup (Alg. 3 and Alg. 4) can be seen in Fig. 5.6, which shows the observed voltage and frequencies while multiplying. For all graphs the maximum value of the 1st operand was 0xffffffff while the minimum value of the 2nd operand was 0xff on four multiplication threads with 1000000 iterations per multiplication.

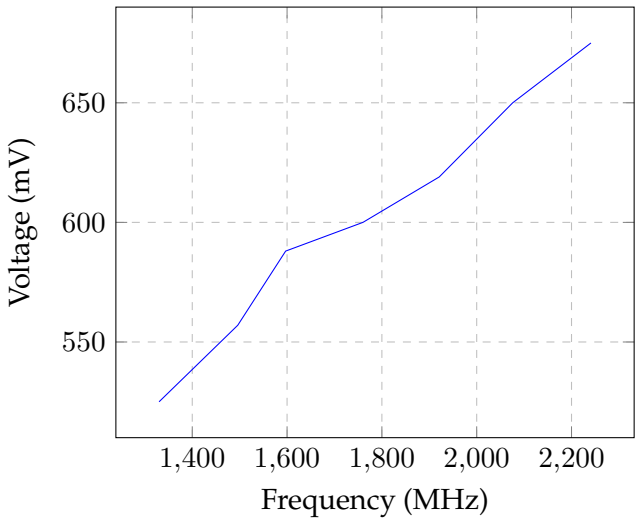

Figure 5.5: Crash point voltage and average frequency values that were observed. The set frequency range is from 1600MHz-2800MHz. The starting voltage was set to 925mV for a consistent voltage drop.

The same two behaviours that were observed on the NUC were also observed on here during testing:

- An abrupt crash of the entire system, where it would restart by itself, when the voltage is set too low for the system to continue normal operation.
- A system freeze, where still sufficient power remained for the system to not crash entirely, but further execution was halted on all fronts. A manual reset was required here.

Using this initial methodology, no erroneous result in multiplication was ever observed nor were MCE errors shown by *rasdaemon* (*mcelog* does not support AMD CPUs [mcea]). As one can see in table 5.1 the difference between a crash, a freeze or normal operation is minimal, for example the recorded crash point at a set frequency of 22000MHz had the same voltage as an instance where the testbench terminated properly, while also running at a lower frequency and recording a lower temperature. The same may be observed for higher set frequencies as well.

Set Freq. (MHz)	Crash (Volt, Freq., Temp.)	(Volt., Freq., Temp.)	(Volt., Freq., Temp.)
2000	(563, 1708, 38.2)	(569, 1783, 47.3),×	(575, 1780, 45.7),×
2200	(594, 1971, 40.8)	(594, 1979, 41.5),×	(600, 1951, 42.5),×
2400	(600, 2069, 44.3)	(594, 2138, 44.6),×	(600, 2162, 43.3),×
2600	(650, 2337, 47.0)	(650, 2280, 41.2),×	(657, 2343, 41.3),×
2800	(675, 2442, 41.0)	(675, 2468, 43.2),✓	(682, 2529, 44.0),×
3000	(707, 2631, 42.3)	(719, 2692, 44.0),✓	(719, 2663, 44.5),×

Table 5.1: Behaviour when the testbench is stopped at voltages, given as mV, slightly above the recorded crash voltage, average frequency and temperature (in the 2nd column) while multiplying. A tick is a freeze, a cross is the testbench terminating properly. For a consistent voltage drop, the starting voltage was 925mV for all tests.

Experiment 2: varying frequency along with voltage

Incrementing frequency In order to counter the above-mentioned effect of enabling overclocking, the set frequency is incremented in 25MHz intervals while the voltage is decremented (refer to Alg. 6 for the changed undervolting thread, but without line 11). While this did somewhat counteract the observed effect in Fig. 5.4, no fault was observed using purely this addition.

Low frequencies It was observed that down-scaling of frequency does not occur if the frequency set is sufficiently low (≤ 600), even if the voltage is reduced to the point of producing a crash. This was abandoned because despite the downwards scaling of frequency, the crash points as shown in Graph 5.6 appear fairly linear, implying that this effect does not inhibit the ability to produce crashes.

Experiment 3: further adjustments

As both the replicated setup from Plundervolt and the previous experiments did not produce any incorrect calculations, some inspiration was drawn from V0ltpwn:

- A stressor function was added based using an AVX instruction. The instruction $sqrtpd$ was chosen for its latency. Two random floats are generated per evocation.
- The undervolt thread and multiplication threads were divided up in separate cores by setting their thread affinities. The 0th core was reserved for the undervolt thread.
- Finally, minimizing the duration of the undervolt in order to keep the system as stable as possible such that calculation errors are produced, but the machine does not crash. This can be done by remembering the initial starting voltage and setting this value temporarily through the SMU, before decrementing the voltage again.

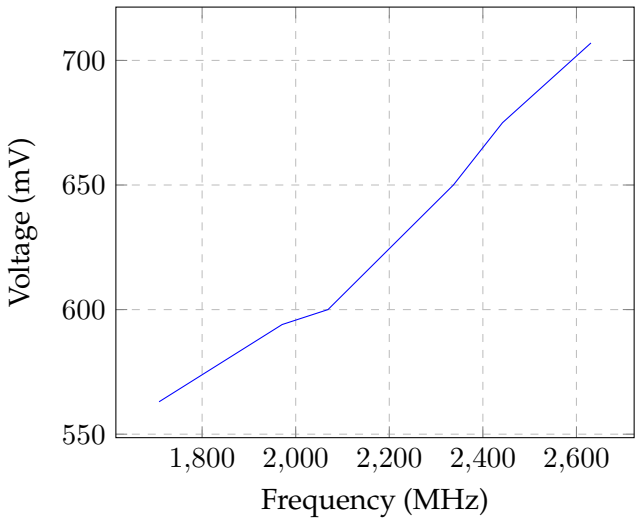

Figure 5.6: Crash point voltage and average frequencies recorded while multiplying (refer to Table 5.1 for values)

The final setup is as follows, with Algorithm 5 and Algorithm 6 running concurrently:

Algorithm 5: Variation of 3

```
1
2
3  while (true) do
4      var = op1 * op2
5      while (var == op1 * op2) do
6          var = op1 * op2
7
8
9
10
11         stressor()
12     end
13 end
```

Algorithm 6: Undervolt thread

```
1  int freq
2  int inc
3
4
5
6  enable_OC()
7  set_frequency(freq)
8  set_voltage(volt)
9  freq += inc
10 volt−−
11 temporary_voltage_restore()
12
13
14 disable_OC()
```

Set Freq. (MHz)	Crash (Volt, Freq., Temp.)	(Volt., Freq., Temp.)	(Volt., Freq., Temp.)
2000	(563, 1728, 39.6)	(575, 1670, 43.3),×	(582, 1673, 45.2),✓
2200	(575, 1894, 39.6)	(582, 1957, 38.7),×	(588, 1815, 39.5),×
2400	(600, 2118, 38.2)	(607, 2668, 38.2),×	(613, 2129, 38.5),×
2600	(644, 2325, 38.0)	(644, 2268, 38.8),×	(657, 2337, 46.7),×
2800	(675, 2486, 44.5)	(675, 2460, 41.1),✓	(682, 2518, 39.2),✓
3000	(707, 2700, 39.5)	(707, 2668, 38.6),✓	(707, 2709, 45.6),×

Table 5.2: Behaviour when the testbench is stopped at voltages, given as mV, slightly above the recorded crash voltage, average frequency and temperature (in the 2nd column) while multiplying with a stressor function in place. A tick is a freeze, a cross is the testbench terminating properly. For a consistent voltage drop, the starting voltage was 925mV for all tests.

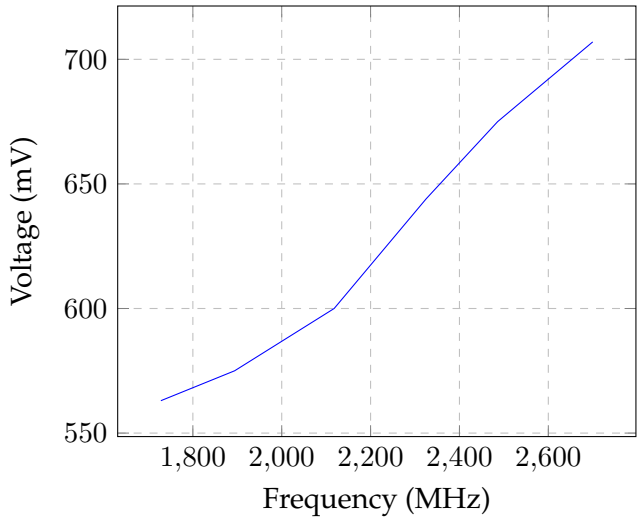

Figure 5.7: Crash point voltage (refer to Table 5.2 for values) and the average recorded frequency while multiplying with an additional stressor being run for every multiplication.

Experiment 4: Replacing multiplication with other operations

Under the assumption that the critical path of multiplications being insufficient to produce a bit-flip, two other operations were tested instead:

AES-NI The multiplication was replaced with AES-NI encryption, with the testbench described in Section 4 Algorithm 2 being used. Undervolting through the SMU instead of MSR as in Intel was modified accordingly.

sqrtpd The stressor instruction $sqrtpd$ was used instead of multiplication due to its longer critical path (a multiplication has a maximum latency value of 10, while the SSE2 instruction $sqrtpd$ has a maximum latency value of 28[4]. Latency "denotes the time from when the operands of the instruction are ready and the instruction can begin execution to when the results of the instruction are ready" [AR19]).

For both sub-variants, crashes and freezes were observed, but no faults.

[4]https://uops.info/table.html

6 Conclusions

6.1 Summary

This thesis dealt with DVFS-based hardware faults. First, results of a known vulnerability on Intel processors were reproduced, initially on multiplications running in userland, then on AES-NI running inside an enclave. The potential for reproducing such faults on AMD Zen processors was then explored. Initially, the testbench of Plundervolt was recreated as closely as possible and with the means known to the author. While crashes were produced, faults in operations were not. Hence, several modifications were introduced to the testbench, such as increasing the frequency while scaling the voltage down, trying to produce a fault under extremely low frequencies where the downward scaling behaviour of the frequency did not occur, as well as optimizations used by V0ltpwn, such as the integration of a stressor function. The modified experiments also did not produce faults. Due to the limitations of the power management interface used, it was not possible to test the EPYC lineup of processors.

6.2 Discussion and open problems

During testing on the Ryzen 5, not a single fault on the operation being run was ever observed. A possibility is that Zen CPUs possess larger capacitors, such that timing constraints are always met in Zen, allowing logic gates to always hold their correct values for the duration of their timing constraint. Another possibility is that Intel Hardware is more power optimized. An observation made while testing was that the idle state on the NUC was usually around 685mV, which is substantially less than the 900mV of the idle P-State. If the Intel core simply require less power to function in general, the threshold where a bit flip occurs can may reached without the entire system freezing, while this threshold is simply not reachable on Zen CPUs. Finally, control may not be fine-grained enough via the manual OC interface, as the power commands aren't directly sent to the CPU, but to the SMU which then updates the power supplied.
The greatest challenge may have been the limitations of the manual OC interface: one can only specify a discrete set of VIDs that can be run, making it a lot more difficult to hit the exploitable voltage window (Fig. 2.4). From Skylake onwards, Intel Chips possess an external voltage regulator that also takes a VID, but in 5mV steps, while the MSR 0x150

allows steps of $\frac{1}{1024}V$ [MOG+20, 3], meaning that one can set a far more precise value using the offset in the MSR than when directly using the VIDs. Whether an equivalent MSR or other interface exists for AMD Zen is an open question. Much about the functionality of the SMU is still unknown.
Finally it is unclear if the bitflips observed may not simply be a quirk of Intel's ALU. As their CPUs are extremely homogeneous, it is not surprising that the same behaviour is observed across different CPU types and Microarchitectures. As described in Section 4, a bitflip occurred at around the same position at every fault for sufficiently large operand values or the most significant bits would flip if the second operand was smaller than 0xff, in other words, it is possible to reliably produce an error at the same locations in a calculation. On the other hand, not a single fault was produced on the Ryzen. Control over voltage may simply not be fine-grained enough.
Since AMD Zen allows per core frequency setting, a similar setup to CLKScrew was also intended as an experiment in hope of improving the stability of the testbench. Here, a victim core hosting the multiplication thread is designated and its frequency is set to some high value and all other cores were set to a low value, such these would not be affected by undervolting as much (VIDs can only be set for the entire die). However, the scant documentation on the functionality of per core frequency settings proved to be incomplete and reversing proved to be far outside of the time available for a bachelor thesis. Much of the behaviour of the SMU is not known to be public and there is plenty of potential for reverse engineering.

References

[Ama20] Ron Amadeo. Heads roll at intel after 7nm delay. 07.08.2020. [Online; accessed 24.08.2020].

[AMD] AMD. Amd secure encrypted virtualization (sev). `https://developer.amd.com/sev/`. Accessed: 30.06.2020.

[AMD16] AMD. Bios and kernel developer's guide(bkdg) for amd family 16h models 30h-3fh processors. `https://www.amd.com/system/files/TechDocs/52740_16h_Models_30h-3Fh_BKDG.pdf`, 2016.

[AMD18] AMD. Bios and kernel developer's guide(bkdg) for amd family 15h models 70h-7fh processors. `https://www.amd.com/system/files/TechDocs/55072_AMD_Family_15h_Models_70h-7Fh_BKDG.pdf`, 2018.

[AR19] Andreas Abel and Jan Reineke. uops.info: Characterizing latency, throughput, and port usage of instructions on intel microarchitectures. In *ASPLOS*, ASPLOS '19, pages 673–686, New York, NY, USA, 2019. ACM.

[BMS+20] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 54–72. IEEE, 2020.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[cem17] cemeyer. Freebsd kernel commit. `https://github.com/freebsd/freebsd/commit/3bf35bb3c4db9d08e8a305c7ca9ac56edaa4fc10#diff-86d2e6e47ca32210871292624b4ea389`, 2017.

[CGG+19] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk

Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on meltdown-resistant cpus. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 769–784. ACM, 2019.

[do] david oswald. encl.cpp. `https://github.com/KitMurdock/plundervolt/blob/master/sgx_aes_ni/Enclave/encl.cpp`. [Online; accessed: 26.08.2020].

[Gha16] Yazen Ghannam. Linux kernel commit. `https://github.com/torvalds/linux/commit/ddfe43cdc0da3189feac4bb9f0f818bef6d6e56e`, 2016.

[git19] Different smu message ids for different platforms? `https://github.com/FlyGoat/ryzen_nb_smu/issues/3`, 2019.

[HH13] David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture*. Elsevier, 2nd edition, 2013.

[Int] Intel. Intel processors voltage settings modification advisory. `https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00289.html`. [Online; accessed 27.08.2020].

[Int15] Intel. What exactly is a p-state? (pt. 1). `https://software.intel.com/content/www/us/en/develop/blogs/what-exactly-is-a-p-state-pt-1.html`, 2015.

[Kap16] David Kaplan. Amd x86 memory encryption technologies. `https://en.wikichip.org/w/images/4/4e/amd_x86_memory_encryption_technology.pdf`, 2016. [Online; accessed 24.08.2020].

[KFG+20a] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0ltpwn: Attacking x86 processor integrity from software. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1445–1461. USENIX Association, 2020.

[KFG+20b] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0ltpwn: Attacking x86 processor integrity from software presentation video. `https://www.youtube.com/watch?v=TH709b_NJBU`, 2020. [Online; accessed 26.08.2020].

[KPW16] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, 21.04.2016. [Online; accessed 24.08.2020].

[Mar14] Rudolf Marek. Amd x86 smu firmware analysis. https://fahrplan.events.ccc.de/congress/2014/Fahrplan/system/attachments/2503/original/ccc-final.pdf, 2014.

[mcea] Mcelog faq. http://www.mcelog.org/faq.html. [Online; accessed 01.07.2020].

[mceb] Mcelog glossary. http://www.mcelog.org/glossary.html. [Online; accessed 01.07.2020].

[MOG+20] 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 *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1466–1482. IEEE, 2020.

[QWLQ19a] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 195–209. ACM, 2019.

[QWLQ19b] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. Voltjockey: Breaking SGX by software-controlled voltage-induced hardware faults. In *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2019, Xi'an, China, December 16-17, 2019*, pages 1–6. IEEE, 2019.

[SLM+19] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload: Cross-privilege-boundary data sampling. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 753–768. ACM, 2019.

[spe] speed47. Spectre & meltdown checker. https://github.com/speed47/spectre-meltdown-checker. [Online; accessed 30.06.2020].

[tru] Arm trustzone technology. `https://developer.arm.com/ip-products/security-ip/trustzone`. [Online; accessed 30. June 2020].

[TSS17] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. CLKSCREW: exposing the perils of security-oblivious energy management. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1057–1074. USENIX Association, 2017.