NTNU – Trondheim
Norwegian University of
Science and Technology

Analysis of Acoustic Emanations From Computer Systems

Martin Kirkholt Melhus & Haakon Garseg Mørk

Submission date: December 2014
Responsible professor: Stig F. Mjølsnes, ITEM
Co-supervisor: Markku-Juhani O. Saarinen, ITEM

Norwegian University of Science and Technology
Department of Telematics

Abstract

Computers allegedly emit low-bandwidth acoustic signals, i.e. acoustic emanations from the audible range of the human ear and up to several hundred kHz. This phenomenon exposes computers to acoustic side-channel attacks, as adversaries can obtain information about internal operations on the CPU, which can possibly be used to extract cryptographic keys.

In this paper we build a portable and a lab-grade setup, and use them to analyze the information in the suggested side-channel for several computers.

Our empirical analysis of frequency spectrograms resulting from our experiments enhance the suspicion that there exist information leakage through an acoustic side-channel in the frequency range 0-100kHz.

We are able to distinguish between different CPU operations and variations in CPU workload by evaluating frequency spectrograms. We also demonstrate that the acoustic signatures of different hardware configurations vary significantly.

Acknowledgments

We would like to thank Stig Frode Mjølsnes for guidance and valuable suggestions throughout the process. Markku-Juhani O. Saarinen helped in suggesting ways to improve Signal-to-Noise Ratio (SNR) in frequency spectrograms, as well as suggesting experimentation on ARM devices.

We would also like to thank the NTNU Department of Telematics (ITEM) for economical support which enabled us to obtain the necessary equipment for our experimental setups. Tim Cato Netland from the NTNU Department of Electronics and Telecommunications (IET) suppled invaluable advice and recommendations of equipment for portable setup and also helped us assemble the lab grade setup, and consulted us in the use of the anechoic chamber. Magne Hallstein Johnsen helped with guidance regarding signal processing implementations.

Tarjei Husøy provided us with the Lenovo T60p laptop, and together with Karoline Harstad Jensen contributed with editorial advice and suggestions.

Contents

List of Figures

Listings

List of Tables

List of Acronyms

AC Alternating Current.

CPU Central Processing Unit.

DoD U.S. Department of Defense.

DPA Differential Power Analysis.

IET NTNU Department of Electronics and Telecommunications.

ITEM NTNU Department of Telematics.

NI National Instruments.

NSM Norwegian National Security Authority.

PCM Pulse-Code Modulation.

PGP Pretty Good Privacy.

RAM Random-Access Memory.

SNR Signal-to-Noise Ratio.

SPA Simple Power Analysis.

WAV Waveform Audio File Format.

Chapter 1
Introduction

1.1 Motivation

In 2014, Genkin et al. presented their work on exploiting low-bandwidth acoustic emanations to extract a full 4096-bit RSA key [1]. This novel approach opens for a whole new attack vector that has unexplored potential, as the emanations are allegedly able to leak information about what is being executed on the CPU. In this paper we will verify some of the claims presented by Genkin et al., by setting up and performing the experiments that are outlined, and possibly reproduce the results presented.

1.2 Problem and Scope

We build a similar setup to what was presented in [1], and use it to record computers running in different states. Then we analyze the recorded acoustic emanations, and attempt to relate the CPU operations to the acoustic emanations.

We diverge from the original project description in that we do not emphasize on applications with regard to cryptanalysis. Rather we explore the extremes of CPU operations, to be able to verify the initial results of Genkin et al. We rely strictly on empiric evaluation of our results during analysis, and use no statistical correlation models. This is done despite the fact that it introduces some limitations to our exploration of the side channel, as the proposed attack on the vulnerable RSA implementation would require such a model. However, the approach is deemed viable due to the results presented in [1], and it requires a simpler setup. Thus RSA key extraction is beyond the scope of this paper.

Using this setup we will show that we are indeed successful in reproducing some of the results presented by Genkin et al., by distinguishing between different CPU activities based on the acoustic fingerprint and computer assisted empirical evaluation.

1.3 Methodology

We record the acoustic emanations of known execution patterns. Then, we represent the acoustic signatures visually in frequency spectrograms, and empirically try to correlate the nature of the spectrograms with the execution patterns.

1.4 Outline

In this paper we will first present related work, and give a short presentation of the background of our work. Then we will present our experimental setup, first in the form of the instrumentation; then in the form of the code that we execute on the computers during our experiments. After this has been described in detail we will present our results, and discuss our findings.

Chapter 2
Background

In this chapter we will look at how side-channels have been exploited earlier, and look at current research on the field of low-bandwidth acoustic cryptanalysis.

2.1 Side-Channels

Computer systems can in theory achieve security, as code can be secure from a mathematical perspective. However, these systems are realized in a physical form which introduces unintended channels where information is leaked. These channels include electricity consumption, time taken to compute, and electromagnetic emanations. All of these have to be taken into consideration when transforming a system from a theoretical model to a physical realization, as even computer code needs to be executed on a physical circuit. Failing to consider this information leakage can enable an attacker to break the system, e.g. by extracting cryptographic keys.

2.1.1 Acoustic Cryptanalysis of Mechanical Systems

The acoustic side channel has been known as a potential attack vector for mechanical systems since the second world war, where phone taps were used to find the keys of mechanical cipher machines. The codes were renewed every day, producing a mechanical click sound which enabled an eavesdropper to derive yesterdays key [2, pp. 103-107].

In 2005, Asonov et al. were able to reconstruct what was typed on a keyboard by analyzing the acoustic emanations, using a neural network [3]. They also showed that phones and ATMs with mechanical keyboards can be vulnerable to this kind of acoustic side-channel attacks.

2.1.2 Timing Attacks

The devastating result of not taking side channels into consideration in cryptographic implementations was shown by Paul Kocher in 1996. He presented his findings in how measuring the amount of time required by an operation enabled an attacker to break cryptosystems such as RSA and Diffie-Hellman [4].

In 2011, Brumley et al. published a paper describing a timing attack vulnerability in OpenSSL allowing an attacker to ultimately extract private keys [5]. This is a good example of how hard known side-channels can be to silence.

2.1.3 Power Analysis

Electronic devices can leak information through their power consumption, and this was proven by Paul Kocher in 1999 [6]. The power traces used in power analysis contain much more information than the timing information utilized in timing attacks, and Kocher demonstrated that a single power trace can be enough to read out a full RSA key, with a naive implementation of the cryptographic algorithm using Simple Power Analysis (SPA). These kinds of attacks can be empowered by using statistical models, using Differential Power Analysis (DPA).

2.1.4 Electromagnetic Emanation Analysis

An adversary can, intercept electromagnetic signals emitted by all electronic devices, given that he is within range of the emanations. It is not possible to detect this attack, and also hard to prevent. One known attack extracts the electromagnetic signals from a computer monitor to reproduce the screen output. These signals originate from the electron gun that manipulates each pixel [7]. The stronger a pixel is, the stronger the signal. Pretty Good Privacy (PGP) include a secure viewer that prevents these exact high frequency signals by making softened edges in the fonts. However, it is said to be very difficult, thus expensive to extract the signals emitted by a computer monitor.

2.2 Related Work

The acoustic side channel is the basis for this paper, and the research done in the area of acoustic emanations from a computer is limited.

2.2.1 Acoustic emanation

In a rump session at the Eurocrypt 2004, Eran Tromer and Adi Shamir presented [8] a new concept of listening to CPU's searching for acoustic emanation. They managed to detect HLT instructions, the only one detectable at that time, on a certain computer

and used this result for further research on extracting information used in acoustic cryptanalysis [9].

2.2.2 Low-Bandwidth Acoustic Cryptanalysis

We base our project on the work of Daniel Genkin, Adi Shamir and Eran Tromer. The research they have done is the RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis [1], based on their earlier work [8, 9]. Their claim is that computers produce low frequency acoustic emanations from their electronic components, possibly caused by vibrations in transistors or electronic components. The frequency range of the allegedly viable emanations lie in the sub 100kHz range. Genkin et al. were able to extract a 4096-bit RSA key using this side channel and exploiting a vulnerability in an implementation of RSA. The claim that the acoustic emanations in this frequency range from a computer system carries information about the state of the computer, is new. Earlier, acoustic cryptanalysis has been limited to mechanical systems, hence the discovery of this attack vector may open for a whole new range of attacks.

Chapter 3
Experimental Setup

In this chapter we will look at the equipment we use in our recordings. Further we will go into detail about the limitations and possibilities introduced by our experimental setup.

3.1 Recording and Sampling

3.1.1 Sampling and Encoding

We want to analyze the acoustic emanations from a CPU in the frequency domain, and look at power spectra representing a period of time. The sampling frequency has a big impact on the range of frequencies that can be analyzed, as it is limited by the Nyquist frequency given in Equation 3.1. Genkin et al. presented results that suggest that one should be able to observe phenomenons related to acoustic emanations from computer systems for frequencies below 100kHz; our setup relies on National Instruments (NI) myDAQ that is capable of sampling at a rate F_s of 200kHz [1]. Thus by only looking at Equation 3.1, we are able to analyze frequencies up to 100kHz.

$$F_{nc} = \frac{F_s}{2} \tag{3.1}$$

The captured signals are stored in Waveform Audio File Format (WAV) files where the audio is encoded as Pulse-Code Modulation (PCM). Hence we do not perform real-time analysis, as we work on stored recordings during our analysis. Implications are discussed in detail in chapter 4.

[1] Our initial experiments were conducted using a Zoom H4n recorder, with a 96kHz sampling rate. However, we did not possess a remote control for the recorder, hence we had to physically press buttons on the recorder to start and stop recordings. This introduced a lot of noise in the recordings, and made our results inconclusive.

3.1.2 Portable Recording Setup

For our first round of experiments we use the Knowles Ultrasonic SPU0410LR5H microphone. The microphone has a flat frequency response up to 80kHz, with a -4dB sensitivity [10]. To power the microphone we use a 4.5V battery. NI myDAQ is used for sampling, connecting the microphone directly into the Analog Input 0 on the myDAQ device.

The fact that the sampled signal is not bandlimited, i.e there are no low-pass filter in this setup, the sampling theorem suggest that we should observe distortion due to aliasing [11, pp. 384-394].

3.1.3 Lab Grade Recording Setup

With our second setup we are aiming to come close to what Genkin et al. used for their lab-grade setup. We use a Brüel&Kjær 4939 microphone that is able to do precise measurements up to frequencies around 100kHz [12]. The microphone is then connected to a Norsonic 1201 preamplifier [13] which again is connected to a Norsonic type 336 microphone amplifier and power supply. The Norsonic 336 provides several options regarding the gain [14]. We use 40 dB gain in all of our recordings. The output signal from the Norsonic is led into a Krohn-Hite Corporation 3945 filter [15] which provides a 1kHz high-pass filter as well as a 80kHz low-pass filter. The filtered signal is sent to the NI myDAQ, and lastly encoded in a WAV file using a laptop computer.

$$B \leq \frac{F_s}{2} \tag{3.2}$$

The sampling theorem, given in Equation 3.2, gives that the ability to uniquely recover frequencies B from a sampled signal is bound by the sampling frequency, given a bandlimited continuous-time signal [11, pp. 384-394]. Since the inclusion of a band-pass filter gives such a signal for this setup, and we have $B = 80kHz$ and $F_s = 200kHz$, the condition of the sampling theorem holds. Thus, we should experience far less distortion resulting from aliasing using this setup.

3.2 Processing and Signal Extraction

The captured sound is stored in a WAV file. This file is processed in using a piece of self written software, which utilizes libraries such as `FFTW`[2] and `libsndfile`[3] to ensure correct processing of the sound files.

[2] `FFTW` is a library for computing the Fast Fourier Transform, available at http://www.fftw.org/

[3] `libsndfile` is a library for working with different sound file formats, available at http://www.mega-nerd.com/libsndfile/

The WAV files produced by the setups explained in this chapter contain a mono signal. In the WAV format, frames representing each sample is stored subsequently, such that the sample $s_{f,c}$ represents the PCM response for frame f channel c. For a stereo signal this means that $f \in [1, 2]$, thus the samples are ordered $s_{0,0}, s_{0,1}, s_{1,0}, s_{1,1}, ..., s_{n,0}, s_{n,1}$. Since we want to work on a mono signal, we simply ignore all frames where $f \neq 0$, which with our setup means that we use all frames in the stored file.

The captured signal is Fourier-transformed using a window size of 4096 frames, and the hamming window function. The power spectra results are studied, with emphasis on how they are evolving in the time domain, and how they correlate to the expected effects of the software described in chapter 4. Source code for these steps are included in Appendix B.

3.3 Positioning Relative to Source of Emanations

The theory is that the acoustic emanations of interest are emanating from within the CPU, therefore we are positioning the microphones as close to the CPU as possible. When experimenting with a Lenovo T60p, we unmounted the keyboard letting us position the microphone on top of the cooling block for the CPU as illustrated in Figure 3.1.

For the other target computers we use in the research, we strive to position the microphones as close to the CPU as possible, without breaking or dismantling the chassis. A full list of all target computers and devices is given in Table 3.1.

Computer	Operating System	Processor
Lenovo ThinkPad T60p	64-bit Linux Mint 17	Intel Core 2 T7400 @ 2.16GHz
Dell Latitude D430	64-bit Linux Mint 17	Intel Core 2 U7600 @ 1.20GHz
Raspberry Pi Model B	Raspbian Wheezy	ARM1176JZF-S @ 700MHz

Table 3.1: Computers used during experiments.

Figure 3.1: A Knowles Ultrasonic SPU0410LR5H microphone positioned close to the CPU of a Lenovo T60p.

Chapter 4
Predictable Execution Patterns

This chapter will describe how we force predictable CPU activity during experiments, and also how we tailor the software run on different architectures to achieve comparable results. We will also outline our expectations regarding analysis of the acoustic emanations from the experiments.

4.1 Selection of Test Cases

A prerequisite to analyze a acoustic recording is to possess an idea of what has been recorded. This allows us to look at the correlation between different actions, and their effect on the recording. In our case this means that we need to know what is happening in the CPU for the duration of our recordings. With this knowledge, we should be able to perform analysis on the correlation between what is observed in the acoustic emanations and distinct CPU activity.

We use several test cases, where the aim is to gradually record increasingly subtle variations in CPU activity, and the recordings of these will make the basis for analysis. Each of these cases is represented by a utility program that is specifically tailored to force the desired CPU activity during the experiment. We will start by oscillating between abnormally high CPU loads and an idle system, and eventually we will try to distinguish between the execution of different distinct low-level CPU operations.

The following sections will in detail describe the different utilities we use to inflict this CPU behavior, and how we wrap the different tokens of predictable executions in a deterministic repeated pattern to be used in later analysis, hence relating the CPU behavior to the time domain.

4.2 CPU Load

Our first choice of distinguishable CPU activity is in the form of a CPU burn utility. The idea is that when no programs are running, save the operating system, the

activity level of the CPU is low. However, if the user executes a program that makes all cores on the CPU work at close to their full capacity, the internal activity of the CPU will change drastically. Additionally, the temperature of the CPU will increase, hence the name.

All these desired properties are available in the `cpuburn` collection which can be installed using the debian package manager[1]. To be able to relate this to the time domain, we execute the CPU burn tool in the pattern described in Listing 4.1.

```
1 for i in {1..3}
2 do
3   burn all cores for 2 seconds
4   sleep 1
5 done
```

Listing 4.1: Mapping execution to the time domain: CPU Burn Utility

In the loop, the CPU will operate at close to full capacity for two seconds, then sleep for a second, representing the heavy load state and the idle state. By recording emanations during execution of this utility, we hope to be able to distinguish between the two states, heavy load and idle, thus be able to observe a repeating pattern with a period of 3 seconds, representing the loop.

For the Raspberry Pi, we use the `sysbench`[2] benchmarking tool to generate CPU load.

4.3 CPU Operations

The CPU operations utility program is used to force reliable execution of different CPU operations at the microinstruction level.

4.3.1 Mapping to Time Domain

Genkin et al. suggest that looping through a set of low-level CPU operations that are repeated over an observable period of time, should result in a distinguishable acoustic fingerprint for each CPU operation. We want a utility that lets us do a similar experiment, and possibly achieve similar results.

Since we are using sampling frequencies in the kHz-range and processors operate on frequencies in the GHz-range, it is futile to try to capture the fingerprint of a single clock cycle representing the execution of a specific microinstruction. This limitation

[1] `cpuburn` can be installed using `apt-get install cpuload`. See http://www.hecticgeek.com/2012/03/cpuburn-cpu-stress-test-ubuntu-linux/ for further details.

[2] `sysbench` is a benchmarking tool available on Raspbian Wheezy. See http://wiki.gentoo.org/wiki/Sysbench for further details.

can be overcome by repeating the same instruction for a longer period of time, such that every Δt seconds, a new instruction will be chosen, and this instruction will be repeated in a loop until it is replaced after another Δt seconds. With n different instructions, one pass through all instructions will take $T = n \times \Delta t$ seconds. We want to repeat the loop more than once, so that we can look not only for a change from one lastingly stable signal every Δt seconds, but also a repeating pattern every T seconds, representing the loop over all instructions.

The repetition pattern used in this utility is given in Listing 4.2

```
1 for i in {1..3}
2 do
3   for j in {1..n}
4   do
5     run CPU operation j for 0.33 seconds
6   done
7 done
```

Listing 4.2: Mapping execution to the time domain: CPU operation loop utility.

The CPU operations we will observe are the `MUL`, `ADD` and `NOP` microinstructions as well as memory access with forced L1 and L2 cache miss. The following subsections will go into further detail on how timely and predictable execution is achieved in all four cases for both ARM and x86 architectures.

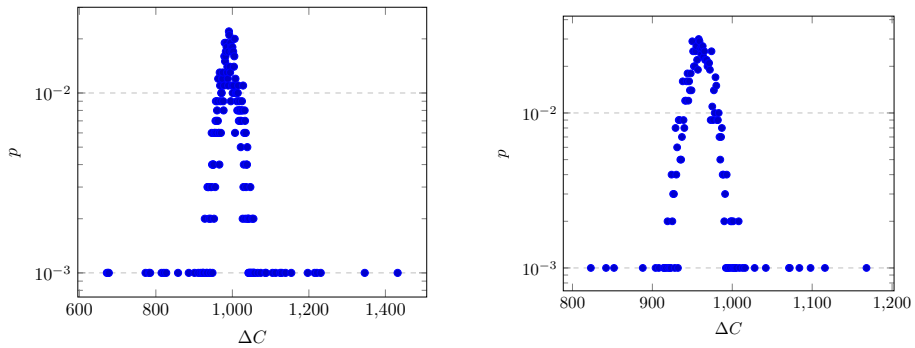
4.3.2 Memory Access and Predictable Cache Miss

The memory dereference (`MEM`) operation's performance with regard to speed depends heavily on if the target is cached or not. If a cache hit occurs in the L1 or L2 cache, the lookup time is much lower than if the value is read from Random-Access Memory (RAM). We want to achieve an execution sequence where the CPU constantly has to go all the way to RAM to fetch values.

As it turns out, list access loads and stride access loads are a good generator of cache misses [16]. However, modern processors are putting great effort into predicting memory accesses; constant stride load patterns can easily be detected in hardware [17], and the data can be prefetched causing few to zero cache misses. For this reason, we cannot rely on simple mechanisms such as sequential memory access using a stride bigger than the size of the cache. Luckily, the existence of hardware performance counters made available in the Linux operating system allow us to evaluate our approach to the `MEM` operation. Performance counters are a set of hardware registers that can be used to count events such as cache misses during the execution of a program, without impacting the execution [18]. Therefore, we are able to benchmark our approach, and verify if we successfully bypass prediction and prefetching mechanisms.

We make two programs; A, which randomly resolves $n = 1000$ indexes in an array that is several orders of magnitude bigger than the L2 cache of the targeted CPU; and B, which is identical to A, save the fact that it deterministically resolves subsequent indexes. During the course of execution, A should to suffer approximately $k + n$ cache misses, while B should suffer only k, due to successful stride prediction. Here k represents the baseline cache misses suffered for running the parts in some section of the program required to set up the loop, and n represents the amount of cache misses suffered from resolving indexes in the array, due to having to perform the memory dereference in RAM. If this condition holds, we can argue that the program A should cause n more cache misses than B and thus reliably executes the `MEM` operation.

We ran A and B 1000 times, measuring the number of cache misses c_A and c_B suffered by A and B. Then we look at the difference in cache misses $\Delta C_i = c_{A_i} - c_{B_i}$ for every run $i \in [1, 1000]$. The results are given in Figure 4.1, and represent the results for our two target laptops; a Lenovo T60p laptop with a Intel Centrino Duo CPU; and a Dell Latitude D430 laptop also with a Intel Centrino Duo CPU.



(a) Dell Latitude D430, $\bar{\Delta C} = 960, s = 21$ (b) Lenovo Thinkpad T60p, $\bar{\Delta C} = 992, s = 43$

Figure 4.1: Sample distribution of suffered cache misses - probability of observing ΔC cache misses from sample size $n = 1000$ obtained by running the `MEM` benchmark described in Appendix C.

Figure 4.1 clearly show that the probability of observing the expected ΔC close to 1000 prevalent for both of the computers. Note that the results contain values that are both significantly bigger and smaller than the expected value of 1000, and this is caused by the fact that the value k is subject to a big variance. This is good enough to satisfy our condition, as the amount of cache misses suffered also strictly higher in A as compared to B. Our `MEM` operation is successfully bypassing the CPUs best efforts of hardware prefetching and stride prediction. For code-level details as to how the benchmark was conducted, see Appendix C.

4.3.3 MUL, ADD and NOP Execution on Different Architectures

The three native microinstructions that we look at are MUL, ADD, and NOP. To achieve persistent execution of these instructions over time, we repeat the assembly code for each instruction 1000 times in a loop. This ensures that the targeted instructions are prevalent, and reduce the impact of the required instructions to perform the eventual loop, and tracking the overall duration. Note that we could have extended the length of each assembly to take the frequency of the repeating runs to the kHz range that is observable with our setup. Genkin et al. have experimented and with various lengths of such loops, finding differences in the acoustic signatures resulting from different lengths [19, Section 3.2].

```
1  mov $0, %eax;
2  mov $1, $ebx;
3  mul %eax;
```

(a) MUL

```
1  mov $0, %eax;
2  mov $1, $ebx;
3  add %ebx, %eax;
```

(b) ADD

Figure 4.2: The setup code for the MUL and ADD instructions. The instruction on line no. 3 is repeated 1000 times.

The NOP operation is represented by `rep nop` repeated 1000 times. For the MEM and ADD instructions, see Figure 4.2.

4.3.4 Expectations regarding acoustic fingerprints

The utility is combining the MEM operation, as described in subsection 4.3.2 and the MUL, ADD and NOP-loops which are described in subsection 4.3.3. Executing the four operations in the pattern given in Listing 4.2, the goal is to observe repeating patterns in the acoustic emanations of the CPUs during execution of the utility. More precisely, we will look for periodic patterns with a duration in time t of 0.33 seconds, representing individual operations. If we can see such patterns with a period of $4t = 1.33$ seconds, the same as that of our outer loop, it would suggest that the emanations are be caused by the CPU performing the operations forced by our utility. Thus we can argue that we are in fact able to distinguish between such low level operations based purely on the acoustic fingerprint.

4.4 Decryption

Genkin et al. suggest that it is possible to distinguish between the different steps involved in decryption with a 4096-bit RSA key. For our decryption utility, we use the GnuPG 1.4.15 library as it contains the vulnerability suggested in [1, Sec. 9.1].

The cipher text being decrypted is the result of encrypting a 4.2MB WAV-format sound file, using a 4096-bit RSA key with the vulnerable version of GnuPG. Essentially

we are using the same encryption and decryption method, key-size and key generation as Genkin et al.

To be able to relate the decryption to the time domain, we execute the decryption according to the pattern given in Listing 4.3

```
1 for i in {1..3}
2 do
3     sleep 1
4     decrypt
5 done
```

Listing 4.3: Mapping execution to the time domain: Decryption loop utility.

The pause between each decryption may resolve in a pattern that can be observed during analysis of the acoustic emanations, allowing us to relate what is happening to the time domain, since we know the duration of each sleep. The duration of a single decryption can also be measured, although it will differ on different computers. Therefore we are left with a clear idea of what to search for during analysis.

Chapter 5
Results

The following chapter will give an overview of the significant results obtained during empirical evaluation of the recordings obtained during our experiments.

5.1 Understanding the Figures

All results are obtained by recording one of the computers mentioned in Table 3.1. While recording, the target computers were running the utilities described in chapter 4, and the computation patterns are allowing us to relate the patterns in the acoustic fingerprints to the different states of execution.

The recordings obtained using the lab grade setup, explained in detail in subsection 3.1.3 were collected following the experiment plan given in Appendix A. The all results are derived from one of the following experiment setups.

CPU load forced high CPU activity as described in section 4.2

CPU operations repeating `NOP`, `MEM`, `MUL` and `ADD` operations as described in section 4.3

Decryption running a RSA decryption as described in section 4.4.

We also give some results from our portable setup, described in subsection 3.1.2.

5.2 Recordings Using the Portable Setup

The results presented in this section are all captured using the portable setup with the Knowles microphone, as explained in subsection 3.1.2. Recordings were taken using a try-and-fail approach, over a period of several weeks, as a step in evaluating our experimental plan and selecting experiments for further recording with the lab

grade setup. All recordings are done in an office environment, with no means taken to reduce background noise. Additionally, all of the results in this section are derived from recordings of the Lenovo T60p laptop.

5.2.1 CPU load

In in Figure 5.1 we see the effects of alternating between idle states and high CPU loads, using the CPU load experiment setup. The result serves as a baseline for distinguishing between effects simply caused by a higher load, and anomalies in the acoustic signature caused by specific operations.

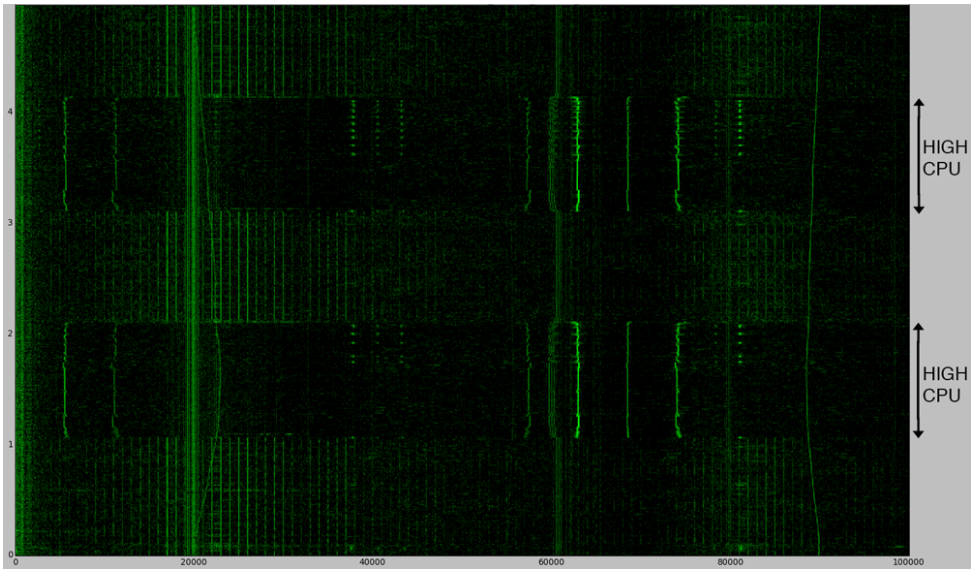


Figure 5.1: Acoustic signature of a battery powered Lenovo T60p, alternating between a full CPU load and an idle state. The vertical axis is time (5 sec) while the horizontal axis is frequency (0-100kHz). The intensity is proportional to the energy in that frequency band at that time. Recorded using the portable setup.

The difference in acoustic signature of the idle state and the high CPU load state is clearly visible in the plot, and the pattern corresponds to the execution pattern of the CPU load utility. Note that at the time of the recording, our CPU load utility had the burn duration set to a single second, resulting in this pattern.

The signal processing resulting in the power spectra making up this plot differs slightly from what is described in section 3.2. Since we are not interested in transforming the frequency spectra back to the time domain, we are able to work with overlapping windows. We compute the Fourier transform using a sliding window with a size W of 4096 frames, while moving the sliding window $W/2$ frames at the time.

This results in a better granularity on the time axis of the frequency spectrogram. Additionally, the z-axis represents the base 10 logarithm of the frequency responses, where only values between the median value (min) and 0 (max) are indicated with different intensities.

5.2.2 Decryption

In Figure 5.2 we can see the resulting acoustic signature from running the decryption utility using the decryption experiment setup.

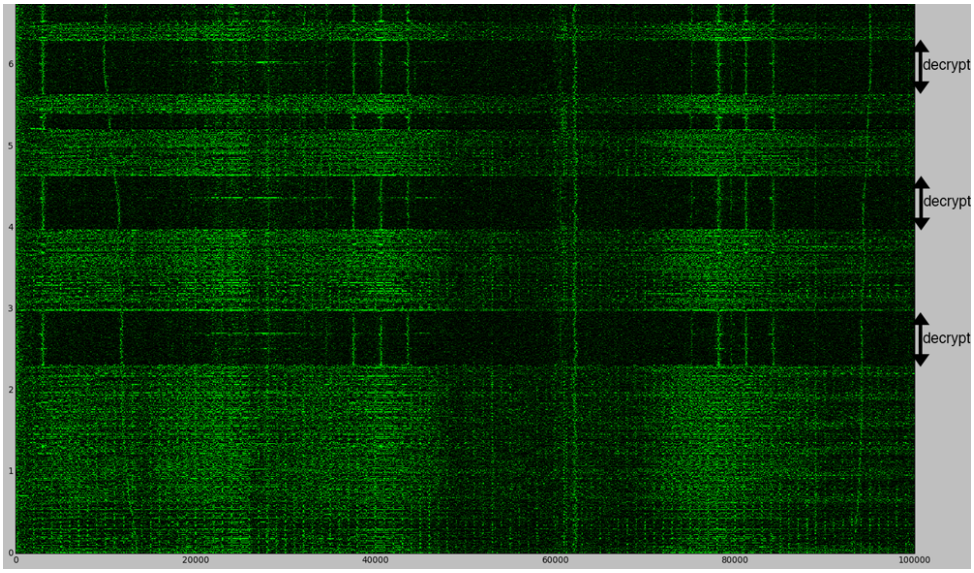


Figure 5.2: Acoustic signature (7 sec, 0-100kHz) of the decryption utility using running on a battery powered Lenovo T60p. Recorded using the portable setup.

The pattern represents portions of time when the decryption was executed. We see that the duration of each decryption, as well as the duration of the pause between each decryption, is according to the execution pattern of the utility.

5.2.3 CPU Operations

The acoustic signature given in Figure 5.3 has been produced when running the CPU operations experiment.

The data processing behind this plot is the same as what is described in subsection 5.2.1. Clearly, there is a noticeable anomaly in the time period representing the `MEM` operation at several frequencies. Additionally, it is easy to see the change between different operations, as it results in repeating patterns in the plot. Note that

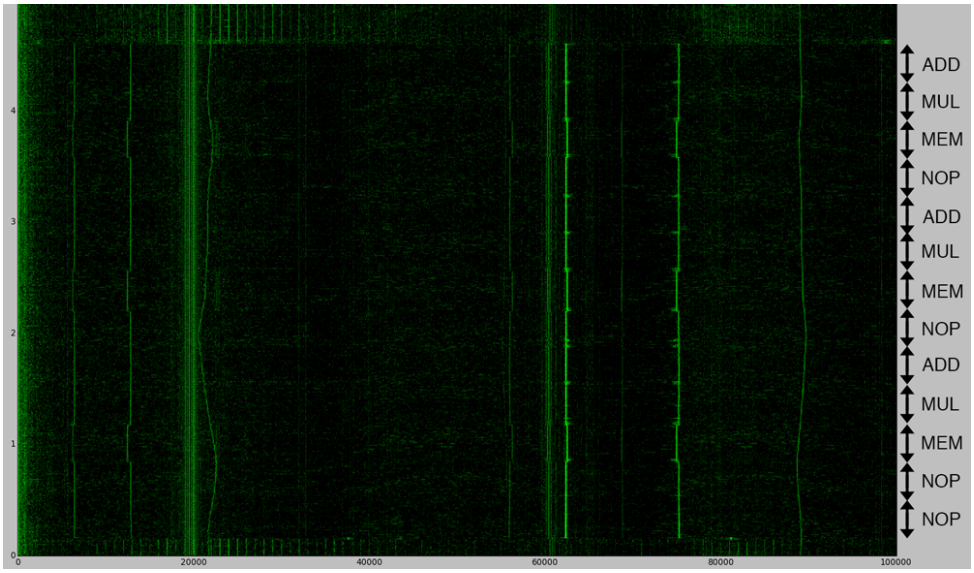


Figure 5.3: Acoustic signature (5 sec, 0-100kHz) of running the CPU operations utility on a battery-powered Lenovo T60p. Recorded using the portable setup.

the CPU operations utility at the time of this recording stared with `NOP` operations, instead of a idle period.

5.3 Recordings Using the Lab-Grade Setup

In this section we present the results from the recordings obtained with our lab-grade setup. The experiences from using the portable setup resulted in the final structure of our utilities, as well as an experimental plan that was carefully followed to obtain these recordings. The experimental plan can be found in Appendix A. Experiments were conducted on all the devices listed in Table 3.1, and recordings were done both in an office environment as well as in an anechoic chamber, to eliminate background noise.

5.3.1 CPU Load

In the anechoic chamber we performed the CPU load experiment on all of the target computers. The resulting frequency spectrograms are given for the Lenovo T60p and the Dell D430 in Figure 5.4 and Figure 5.6 accordingly. Both of these recordings were done with the computers connected to AC power supplies. We also ran the experiment on the Lenovo T60p running on battery power; the frequency spectrogram for this recording is given in Figure 5.5. For more related results recorded in an office environment, see Appendix D.

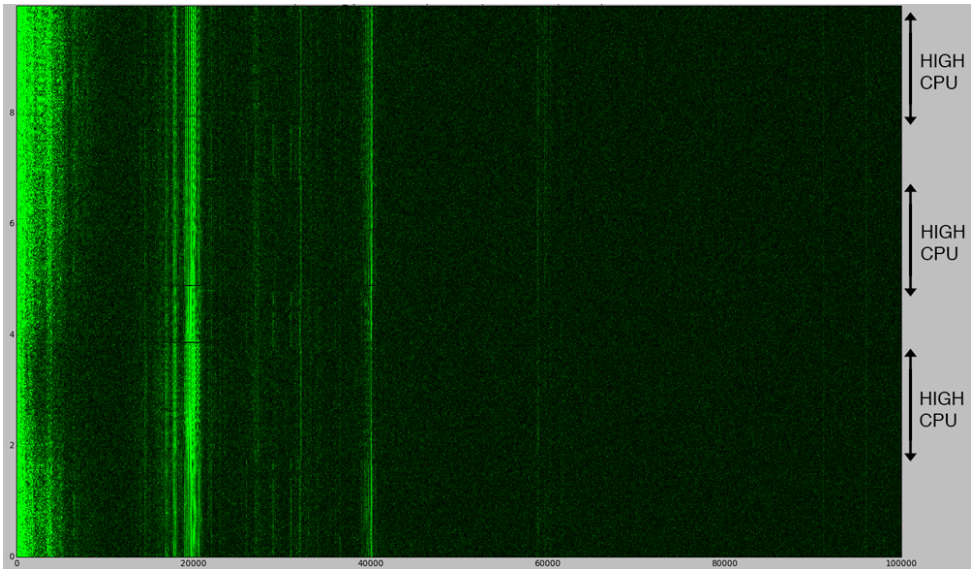


Figure 5.4: Acoustic signature (10 sec, 0-100kHz) of the Lenovo T60p alternating between high CPU loads and idle state while connected to an AC power supply. Recorded in an anechoic chamber using the lab-grade setup.

The differences between Figure 5.4 and Figure 5.5 show that whether the computer is connected to an AC power supply or not has a big impact on the acoustic signature. We are able to distinguish between the different states due to the execution pattern, where the CPU load state lasts for 2 seconds, while the idle state has a duration of 1 second.

5.3.2 Decryption

In Figure 5.7 we can see the resulting acoustic signatures from running the decryption utility with the lab-grade setup. Figure 5.8 gives the resulting frequency spectrogram for the Dell D430. Both recordings are done in an anechoic chamber.

Figure 5.8 is the result from the running decryption on the Dell 430 in the anechoic chamber.

5.3.3 CPU Operations

In Figure 5.9 we see the resulting frequency spectrogram of different CPU operations using the lab-grade setup on the Lenovo T60p laptop connected to an AC power supply. The recording was done in an anechoic chamber.

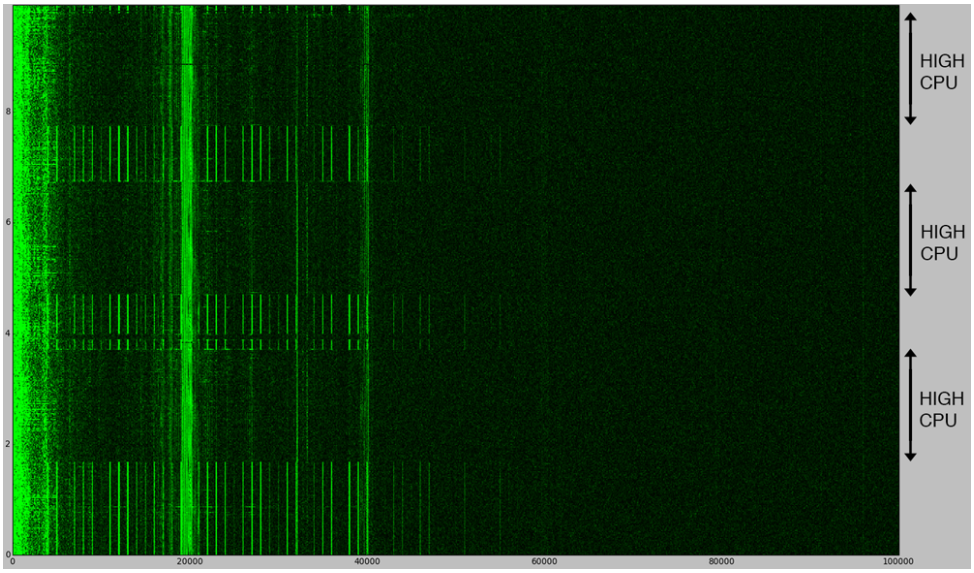


Figure 5.5: Acoustic signature (10 sec, 0-100kHz) of the Lenovo T60p alternating between high CPU loads and idle state while running on battery power. Recorded in an anechoic chamber using the lab-grade setup.

The anomaly at a little over 20kHz is also present in Figure 5.3, which suggests that this period in time corresponds to the `MEM` operation. Also, the duration of the anomalies, as well as the period between each time they occur, match the pattern of execution of the CPU operations utility.

5.3.4 Results from Raspberry PI

All our results from the Raspberry PI are inconclusive. We are not able to empirically correlate any of the frequency spectrograms to the experiments, the way we did the two laptop computers. These inclusive results are explained in section D.1.

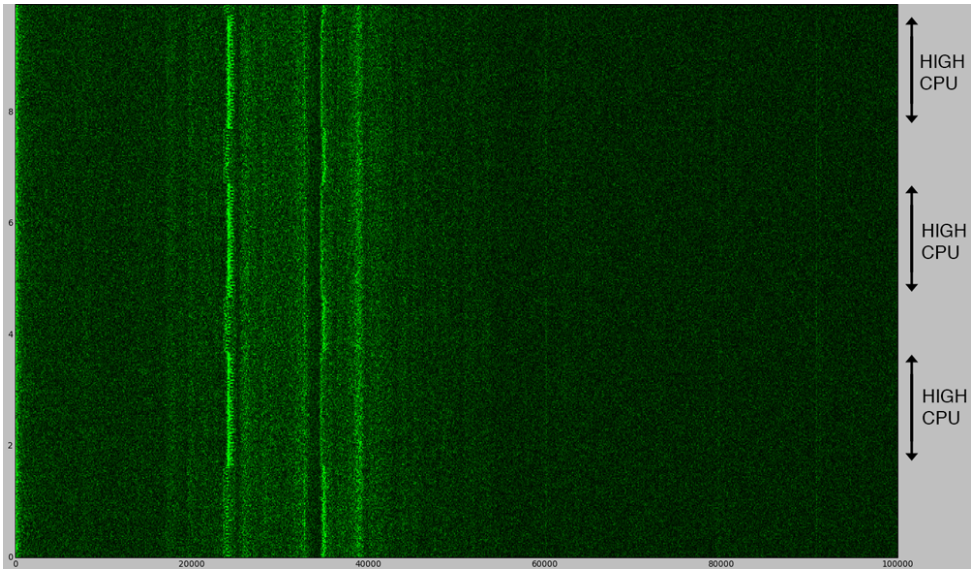


Figure 5.6: Acoustic signature (10 sec, 0-100kHz) of the Dell D430 alternating between high CPU loads and idle state while connected to an AC power supply Recorded in an anechoic chamber using the lab-grade setup.

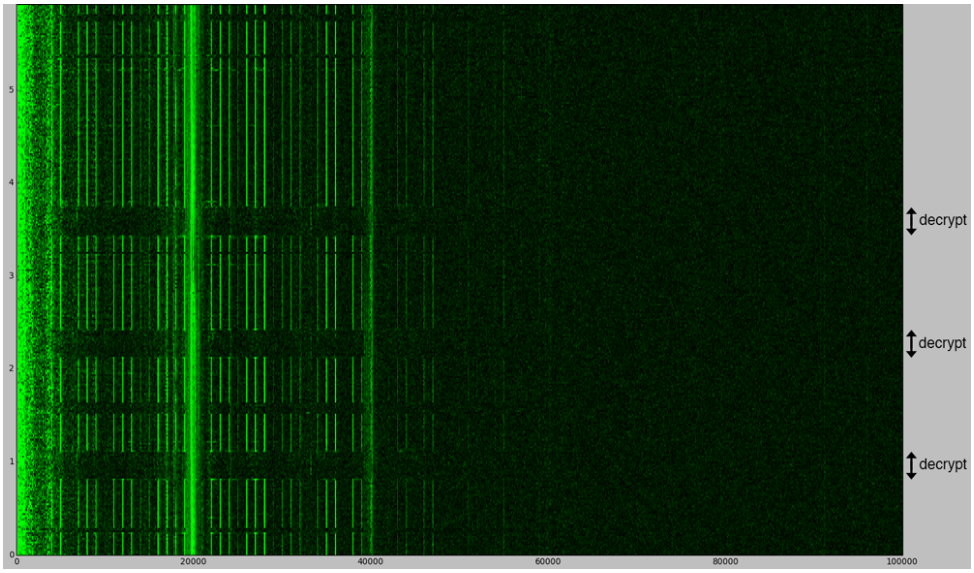


Figure 5.7: Acoustic signature (6 sec, 0-100kHz) of the Lenovo T60p running the decryption experiment while powered by the internal battery. Recorded in an anechoic chamber using the lab-grade setup.

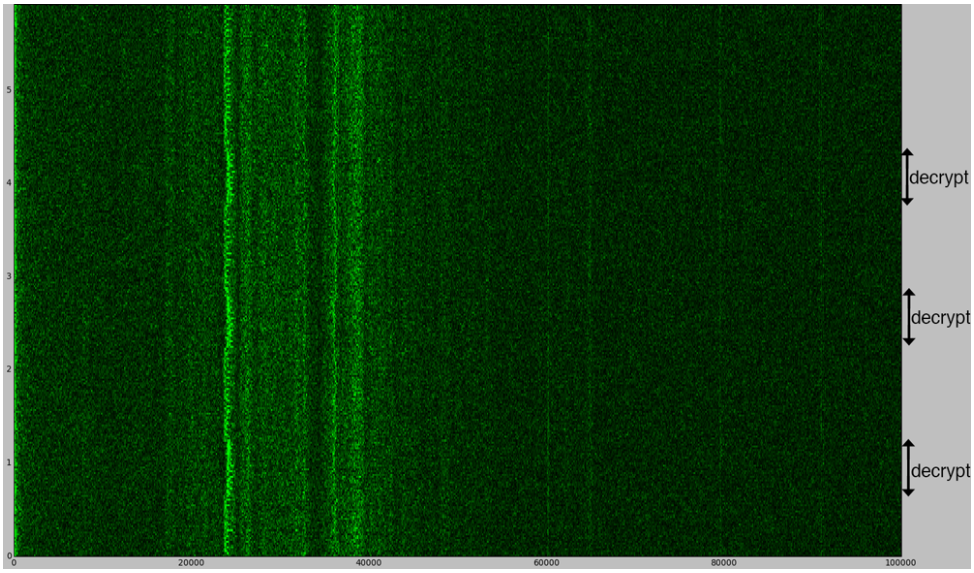


Figure 5.8: Acoustic signature (6 sec, 0-100kHz) of the Dell D430 connected to an AC power supply running the decryption experiment. Recorded in an anechoic chamber using the lab-grade setup.

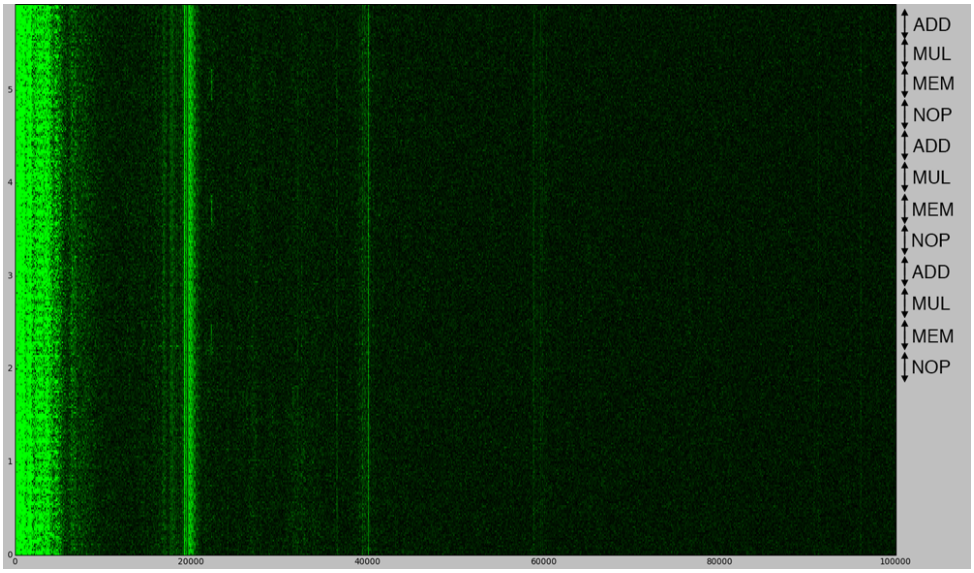


Figure 5.9: Acoustic signature (6 sec, 0-100kHz) of the Lenovo T60p when running different CPU operations. Recorded using the lab-grade setup in an anechoic chamber, while the computer was connected to an AC power supply.

Chapter 6
Discussion

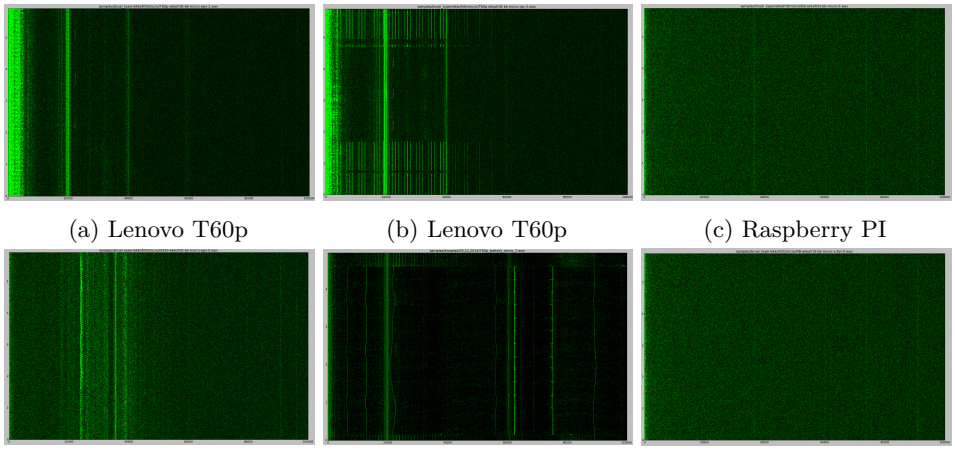
In this chapter we will discuss the viability of the low frequency acoustic emanations. We will also look at the cases where the analysis of these emanations can be useful and how current guidelines for shielding information systems are helping against this threat.

6.1 Significance of Results

We did not do any experiments to verify whether the captured signatures are in fact acoustic, or if they are caused by electromagnetic emanations. However, we rely on the work by Genkin et al., who discuss this question in detail [19, Section 3.3]. Using the acoustic side channel, we are able to distinguish between the `MEM` operation and the other microinstruction loops. Figure 5.3 is a display of how this is difference is visible for many frequencies in the frequency range. The results given in [1, Fig. 2] suggest that it is possible to observe a difference, even between the different microinstructions `ADD`, `MUL` and `NOP`. However, the most visible difference they present is in a frequency range much higher than the hard limit imposed on us by the sampling rate of our setup. Result presented in chapter 5 are therefore inconclusive when it comes to distinguishing between the `NOP`, `MUL` and `ADD` operations.

We have conducted experiments using different configurations of hardware and observe big variations in acoustic fingerprints for the different computers, as seen in Figure 6.1. An interesting side note is how the acoustic emanations of the Lenovo T60p laptop is changing depending on if the computer is running on battery power (Figure 6.1b) or with a power adapter (Figure 6.1a). We believe that the noise introduced by the AC power adapter is not emanating from the power adapter itself, as we have experimented with the positioning of the adapter, without seeing any impact on the resulting acoustic signatures.

We have conducted experiments both inside an anechoic chamber and in an office environment. The results show that eliminating background noise is not required to



(a) Lenovo T60p (b) Lenovo T60p (c) Raspberry PI

(d) Dell 430 (e) Lenovo T60p (f) Raspberry PI

Figure 6.1: Acoustic signatures when running the CPU operations experiment on following devices: (a) T60p with AC power adopter (lab-grade setup). (b) T60p on battery power (lab-grade setup). (c) Raspberry PI recording the CPU (lab-grade setup). (d) D430 on battery power (lab-grade setup). (e) T60p on battery power (portable setup). (f) Raspberry PI recording the CPU (lab-grade setup). All recordings, save (e), is done in an anechoic chamber.

observe the phenomenons. Our portable setup was able to make the same distinctions between the `MEM` operation and the other operations in the CPU operations experiment, under conditions of significant noise, i.e. in an office environment. Figure 5.1 and Figure 5.9 show how the difference is visible in both scenarios.

6.2 Localization of Source

We have not obtained a basis for drawing a conclusion as to whereas the emanations as presented in the previous chapters are in fact caused by the CPU. However, we have tried to capture the effect at different positions on our Lenovo T60p laptop. For this specific computer we experienced that it is far easier to find the patterns we are looking for in the resulting power spectra when the recordings are done right above the CPU. We also found that slight disposition of the microphone of a couple of centimeters, while still hovering the CPU cooling block, notably impact the signal strength.

6.3 Analysis Beyond Extremes

In our experiments, we were able to distinguish between different extremes in CPU activity based visual representation of the acoustic leakage. We do not use any

mathematical models, but rely on empirical analysis of the power spectra. This is a limiting factor in the sense that we are only able to draw conclusions based on clear patterns in the resulting data that are visible to the human eye, when presented in the way done here and by Genkin et al. None of the experiments we performed included real-time analysis of the leakage, hence we were limited to look for patterns resulting from static programs being executed to be able to relate the information captured to the time domain in our offline analysis. Real-time analysis, together with mathematical models for correlation, would allow a more statistical approach to distinguishing between distinct fingerprints, and could allow for attacks along the line of what is proposed in [6], only using acoustic emanations rather than power traces for analysis.

6.4 ARM Processors and Possible Attack Vectors

We chose to conduct experiments on a Lenovo laptop computer similar to what is used by Genkin et al., and also on an even older Dell laptop, as it was suggested that older processors had more explicit acoustic fingerprints. In addition, we conducted the same experiments for a Raspberry Pi, which is equipped with an ARM CPU.

A motivation for looking at ARM processors is that a lot of electronic equipment today is running on the ARM architecture. The potential of analyzing network traffic by looking at routers is just one of several interesting approaches to exploit the potential of low frequency acoustic emanations. Unfortunately, we found it much harder to distinguish between operations on the Raspberry Pi, than for the laptops running on x86/x64 Intel processors. It is still worth noting that the findings of Genkin et al. suggest that distinguishing between different RSA keys used for decryption can be done using this side channel; in the case of a router, this means that the approach could be a viable way to analyze ongoing IPSec sessions, where the router performs encryption or decryption, without being connected to the network at all.

6.5 Possible Attack Scenarios

There are interesting implications of the portable setup (subsection 3.1.2); the microphone itself is very small as well as being very cheap, costing only a few dollars. Hence, this is a very portable setup, that can be hidden and used in covert operations against unsuspecting targets, for instance in a side-channel attack against a vulnerable RSA implementation, as suggested in [1]. The low cost of the parts used, less the myDAQ, also makes it disposable. A small microphone also enables the possibility to hide the whole setup inside the casing of a desktop computer or a server, thus possibly making side channel attacks easier because of a better signal-to-noise ratio due to reducing the distance between the source and the microphone.

Manufacturers with malicious intentions can mount a small ultrasound microphone on a selection of devices from a batch, which in many cases will not be discovered.

Genkin et al. introduces some possible attack scenarios that could exploit the acoustic emanations in [19, Section 1.2 and Appendix B], like an acoustic attack app and self-eavesdropping.

6.6 Countermeasures

In the guidance paper about protection against eavesdropping published by the Norwegian National Security Authority (NSM), it is recommended that one should follow the TEMPEST guidance when installing an information system [20, Section 9.8, page 7]. However, the shielding requirement for a TEMPEST certified equipment is made with respect to electromagnetic signals. Thus TEMPEST requirements for shielding (e.g. a Faraday cage) might not protect against acoustic emanations.

The Red/Black Engineering-Installation Guidelines [21, Section 30.1, page 91] created by the U.S. Department of Defense (DoD), states that unauthorized (BLACK) and authorized (RED) equipment should at least be separated with 3 feet (roughly 0.9 m). Other electronic devices such as mobile phones and personal laptops should not be permitted to areas where RED equipment is installed. Genkin et al. were able to recover a 4096-bit RSA key from a distance of 10 meters, thus an air gap of 0.9 meters will not suffice in canceling the acoustic side-channel itself.

The frequency signatures presented in this paper are leaking information in the audible range of the human ear. Yet, our target computers are relatively old laptops, and our results add to the suspicion of Genkin et al. that the quality of information in the side-channel seems correlated to the computer's age. Hence, it might be harder to extract information using the acoustic side channel on newer computers. We have also added to this theory that ARM processors arguably are not viable targets, as our results show no visible correlation between CPU operations and acoustic signature.

Chapter 7
Conclusion and Future Work

7.1 Conclusion

An assumption taken in our work is that the emanations observed during the experiments are in fact acoustic. This question is discussed by Genkin et al., and they have conducted experiments to support their claim stating that this is true. All our research rely on the validity of this claim.

In this paper we have conducted independent experiments with an aim on verifying the claims of Genkin et al. The experiments have touched a part of the results presented in [1], as they have aimed to verify the bare existence of the acoustic side channel as a viable medium to extract information leakage.

Our research show that we are able to support the claims that the acoustic side channel conveys enough information to distinguish between different low-level CPU operations. We display that the SNR in some cases is strong enough to easily spot such differences using simple plots of the frequency spectra.

The most significant result obtained is the clear display of difference in the acoustic fingerprint of the `MEM` operation and looping other microinstructions. Differences can even be observed at frequencies in the audible range, which supports the claim made by Genkin et al., that a mobile phone's microphone can be used to extract viable information from the side channel. Additionally, this suggests that low-cost equipment can be sufficient for information extraction over this side-channel, something which is supported by our results using our portable recording setup as described in subsection 3.1.2.

Results presented in this paper support the claims by Genkin et al.; low-bandwidth acoustic emanations from computers carry information about the internal state of the CPU. Moreover, extracting this information can be done with a relatively cheap setup, under sub-optimal conditions. Our experiments show that recording in an

anechoic chamber is in no way a necessity, and that the SNR while recording in a room full of other people and different computer equipment is sufficient to clearly see differences in the acoustic fingerprint of distinct CPU operations.

7.2 Future Work

We have not reproduced any of the experiments to verify the claim that the emanations observed are in fact acoustic. Future work should aim to support this theory. Understanding the phenomenons causing the emanations, and identifying the source should also be subject to future research.

In this paper we have have merely looked at information leakage from CPUs operating under very specific extreme conditions. Building statistical correlation models to look at more subtle changes of acoustic signatures should also be a goal for future work, as the work presented in this paper is limited by the methodology chosen. This approach, combined with real-time analysis, would allow for experiments with suggested applications in cryptanalysis, such as the proposed side-channel attack on the vulnerable RSA implementation.

References

[1] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pp. 444–461, 2014.

[2] P. Wright and P. Greengrass, *Spycatcher: The candid autobiography of a senior intelligence officer*. Dell, 1988.

[3] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, pp. 3–11, 2004.

[4] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pp. 104–113, 1996.

[5] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, pp. 355–371, 2011.

[6] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pp. 388–397, 1999.

[7] "Sans institue. an introduction to tempest." Available online at http://www.sans.org/reading-room/whitepapers/privacy/introduction-tempest-981.

[8] E. Tromer, "Acoustic cryptanalysis: on nosy people and noisy machines," *Eurocrypt2004 Rump Session, May*, 2004.

[9] E. Tromer, "Hardware-based cryptanalysis," *Weizmann Institute of Science, Tese de Doutorado*, pp. 161–169, 2007.

[10] "Knowles ultrasonic spu0410lr5h specification." http://www.knowles.com/eng/content/download/4015/50958/version/4/file/SPU0410LR5H.pdf. Accessed 2014-11-10.

[11] J. Proakis, *Digital signal processing*. Upper Saddle River, N.J: Pearson Prentice Hall, 2007.

[12] "Brüel&kjær 4939 specification." http://www.bksv.com/products/transducers/acoustic/microphones/microphone-cartridges/4939. Accessed 2014-12-09.

[13] "Norsonic 1201 specification." http://www.norsonic.com/en/products/microphones+preamplifiers/preamplifiers/nor1201_preamplifier/. Accessed 2014-12-09.

[14] "Nor336 power supply specification." http://www.gracey.co.uk/specifications/nor_336-s1.htm. Accessed 2014-12-09.

[15] "Krohn-hite model 3945 specification." http://www.krohn-hite.com/htm/filters/PDF/3945Data.pdf. Accessed 2014-12-09.

[16] T. Ozawa, Y. Kimura, and S. Nishizaki, "Cache miss heuristics and preloading techniques for general-purpose programs," in *Proceedings of the 28th Annual International Symposium on Microarchitecture, Ann Arbor, Michigan, USA, November 29 - December 1, 1995*, pp. 243–248, 1995.

[17] J. Lee, H. Kim, and R. W. Vuduc, "When prefetching works, when it doesn't, and why," *TACO*, vol. 9, no. 1, p. 2, 2012.

[18] "Perf wiki." https://perf.wiki.kernel.org/index.php/Main_Page. Accessed 2014-11-28.

[19] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," *IACR Cryptology ePrint Archive*, vol. 2013, p. 857, 2013.

[20] N. Sikkerhetsmyndigheter, "Sikring mot avlytting (protection against eavesdropping)." https://www.nsm.stat.no/globalassets/dokumenter/veiledninger/veiledning-i-sikring-mot-avlytting.pdf, 2011. Accessed 2014-11-10.

[21] U. D. of Defense (DoD), "U.s. red/black engineering-installation guidelines." https://www.wbdg.org/ccb/FEDMIL/hdbk232a.pdf, 1988. Validated by DoD oct 2000 and aug 2014, accessed 2014-12-11.

Appendix A
Experiment Plan

This appendix will present our experiment plan, and the planning done prior to the execution of the recording experiments.

A.1 Requirements

We will perform recordings on the following target computers, prepared with programs tailored for this experiment[1].

Lenovo Thinkpad T60p The casing will be partially removed, to allow for recordings closer to the CPU. We will experiment with distances 0.5 cm and 30 cm from the CPU, and we will record while the computer is running both with a power adapter, and on battery power. All permutations of these two parameters add up to four configurations for us to experiment with.

Dell Latitude D430 For this computer we will record through the CPU fan exhaust, at an approximate distance from the CPU of 2 cm.

Raspberry Pi model B We will record with emphasis on three different positions: the CPU, the $3.3V$ regulator, and the area around the $1.8V$ and $2.8V$ regulators[2].

Further we will use the following equipment:

Microphone Brüel & Kjær 4939

Preamplifier Norsonic 1201

[1] The computers are running all our utilities, as described in chapter 4

[2] The Raspberry Pi model B layout is described in this figure http://upload.wikimedia.org/wikipedia/commons/a/af/Raspberrypi_pcb_overview_v04.svg (Accessed 17-Nov 2014)

Amplifier Norsonic type 336 (S/N 20626)

Filter Krohn-Hite Corporation Model 3945 (S/N 005133)

Configuration Ribbon Tweeter Banddiskant D8C, Kenwood FG-275 Function Generator.

Miscellaneous Power chords, camera for documentation, notebook

All recordings will be done twice: in an office environment, with unpredictable sources of background noise; and in an anechoic chamber, with minimal levels of background noise.

A.2 Experiments

The following experiments will be performed on all of the target computers, for all configurations, as given in the previous section. Everything will take place first in the office environment, and then repeated in the anechoic chamber. The following cases are recorded in this order for the three computers, with all the configurations listed[3].

1. A reference recording of a sinusoid produced by the Ribbon Tweeter and the Function Generator, to verify that our setup is in fact recording and working as intended.
2. Reference recordings of the target computer in idle state for 5 seconds.
3. Recordings of the CPU load, running the code described in section 4.2.
4. Recordings of different microinstructions, running the microinstruction loop described in section 4.3.
5. Recordings of the computer doing decryption, as described in section 4.4.

[3]All recordings are repeated five times to help distinguish between phenomenons caused by chance, and actual phenomenons caused by the activity we force on the CPU.

Appendix B
Code Used in Analysis

B.1 Processing Sound Files

The following program written in C was used to process the sound files recorded in our experiments. The reason for writing the program this way is because we were more comfortable with writing in C than using i.e. Matlab. Additionally, we think this code is very adoptable, and can easily be reworked to process real-time recordings as well as being used in combination with automated statistical analysis in future work.

B.2 Usage

The output from Listing B.1 is piped to Listing B.2, to produce a plot representing the frequency response for the duration of the recording.

B.3 Signal Processing in C

The signal processing is done with Listing B.1, a routine written in C. The frequency responses resulting from each Fourier transformed window is written to standard output, as well as some metadata.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdint.h>
4
5 #include <string.h>
6 #include <math.h>
7
8 #include <complex.h>
9 #include <fftw3.h>
10 #include <sndfile.h>
11
12 #define WINDOW_SIZE (4096)
13 #define WINDOW_OVERLAP (1)
```

```
14
15 int process_sndfile(char * sndfilepath);
16 static void dump_spectrum(float *arr, int N);
17 static void dump_metadata(char* filename, int W, int Fs);
18 static void wf_hamming(float * in, int N);
19 static float* power_spectrum(float *in, int N);
20
21
22
23 int main(int argc, char *argv[])
24 {
25     if (argc != 2)
26     {
27         printf("Usage: %s <soundfile.wav>\n", argv[0]);
28         return 1;
29     }
30     else if (argc == 2)
31     {
32         process_sndfile(argv[1]);
33     }
34     return 0;
35 }
36
37
38 int process_sndfile (char * sndfilepath)
39 {
40     SNDFILE *sndfile;
41     SF_INFO sndinfo;
42     float *window, **spectrum;
43     long i, j, w, num_w, ctr;
44     int o, num_c;
45
46     memset (&sndinfo, 0, sizeof (sndinfo)) ;
47
48     sndfile = sf_open (sndfilepath, SFM_READ, &sndinfo) ;
49     if (sndfile == NULL)
50     {
51         fprintf (stderr, "Error : failed to open file '%s' : \n%s\n",
               sndfilepath, sf_strerror (NULL)) ;
52         return 1;
53     }
54
55     /* Accept format 24bit PCM WAV */
56     if ( !( sndinfo.format == (SF_FORMAT_WAVEX | SF_FORMAT_PCM_24)   ||
           // myDAQ
57             sndinfo.format == (SF_FORMAT_WAV   | SF_FORMAT_PCM_24) ) )
                  // Zoom H4n
58     {
59         fprintf(stderr, "Input should be 24bit WAV\n");
60         sf_close(sndfile);
61         return 1;
62     }
```

```
63
64      /* Check channels - stereo */
65      if (sndinfo.channels > 2) {
66          fprintf(stderr, "Wrong number of channels; got more than two.\n
                ");
67          sf_close(sndfile);
68          return 2;
69      }
70
71      /* Print relevant sndfile metadata to stdout */
72      dump_metadata(sndfilepath, WINDOW_SIZE, sndinfo.samplerate);
73
74      float * buffer = malloc(sndinfo.channels*sndinfo.frames * sizeof(
            float));
75      if (buffer == NULL)
76      {
77          fprintf(stderr, "Could not allocate memory for data \n");
78          sf_close(sndfile);
79          return 1;
80      }
81
82      long num_frames = sf_readf_float(sndfile, buffer, sndinfo.frames);
83      if (num_frames != sndinfo.frames)
84      {
85          fprintf(stderr, "Did not read enough frames for source\n");
86          sf_close(sndfile);
87          free(buffer);
88          return 1;
89      }
90      sf_close(sndfile);
91
92      /************************************
93       ***** Start processing sndfile ******
94       ************************************/
95
96
97      /* Number of c_i in frequency response spectrum */
98      num_c = WINDOW_SIZE / 2 + 1;
99      /* Number of windows */
100     num_w = sndinfo.frames / WINDOW_SIZE;
101     window = malloc( WINDOW_SIZE * sizeof(float) );
102     spectrum = malloc( num_w * WINDOW_OVERLAP * sizeof(float*) );
103
104     ctr = 0;
105     /* For every whole window in the samples */
106     for (w = 0; w < num_w; w++)
107     {
108         /* For every partial window within the window (compute with
                overlap) */
109         for (o = 0; o < WINDOW_OVERLAP; o++)
110         {
111             j = 0;
```

```
112             /* For the samples making up this window */
113             for (i = 0; i < sndinfo.channels*WINDOW_SIZE; i += sndinfo.
                    channels)
114             {
115                 /* Assign sample to correct position in window */
116                 window[j++] = buffer[i + (w*WINDOW_SIZE) + o*(
                        WINDOW_SIZE/WINDOW_OVERLAP)];
117             }
118             spectrum[ctr++] = power_spectrum(window, WINDOW_SIZE);
119         }
120     }
121
122     for (i = 0; i < ctr; i++)
123     {
124         /* Print the frequency response components for this window */
125         dump_spectrum(spectrum[i], num_c);
126     }
127
128     free(spectrum);
129     free(window);
130     free(buffer);
131     return 0;
132
133 }
134
135
136 static float* power_spectrum(float *in, int N)
137 {
138     /* 1. Collect N samples where N is a power of 2 */
139     int i, nc;
140     fftwf_complex *out;
141     fftwf_plan plan_forward;
142     nc = N/2 + 1;
143
144     float *result = malloc( sizeof (float) * nc );
145
146     /* 2. Apply hamming window function to the samples */
147     wf_hamming(in, N);
148
149     /* 3. Pass windowed samples to FFTW */
150     out = fftwf_malloc( sizeof( fftwf_complex ) * nc );
151     plan_forward = fftwf_plan_dft_r2c_1d(N, in, out, FFTW_ESTIMATE);
152     fftwf_execute(plan_forward);
153
154     /* 4. Calculate squared magnitude of output bins */
155     for (i = 0; i < nc; i++)
156     {
157         result[i] = (float) sqrt( (float)creal(out[i])*creal(out[i]) +
                cimag(out[i])*cimag(out[i]) );
158     }
159
160     fftwf_destroy_plan(plan_forward);
```

```
161     fftwf_free(out);
162
163     return result;
164 }
165
166
167 static void dump_metadata(char* filename, int W, int Fs)
168 {
169     printf("%s\n", filename);
170     printf("%d\n", WINDOW_OVERLAP);
171     printf("%d\n", W);
172     printf("%d\n", Fs);
173 }
174
175
176 static void dump_spectrum(float *arr, int N)
177 {
178     int i;
179     for (i = 0; i < N; i++){
180         printf("%f ", arr[i]);
181     }
182     printf("\n");
183     /* Flush output buffer to remove buffer delay in pipe */
184     fflush(stdout);
185 }
186
187 static void wf_hamming(float * in, int N)
188 {
189     int n;
190     float a, b, w_n;
191     a = 0.54;
192     b = 0.46;
193     for (n = 0; n < N; n++){
194         w_n = (float) ( a - b * cos( (2*M_PI*n)/(N-1) ) );
195         in[n] *= w_n;
196     }
197 }
```

Listing B.1: main.c - Compute Power Spectrum of a Sound File

B.4 Plotting in Python

Plotting is done using the Python script given in Listing B.2. The data is read from standard input, i.e. the output from the signal processing routine. Simple operations, such as applying a logarithmic scale to the spectra, and setting thresholds for the plot, is done by tuning parameters in this script.

```
1 #!/usr/bin/python
2 import sys
3 import matplotlib.pyplot as plt
4 import numpy as np
```

```
5 from pylab import *
6
7 def main(argv):
8
9     title   = sys.stdin.readline().strip()          # filepath to
          sndfile
10    overlap = float(sys.stdin.readline().rstrip())  # window overlap
11    W       = float(sys.stdin.readline().rstrip())  # window size
12    Fs      = float(sys.stdin.readline().rstrip())  # sampling
          frequency
13    lines   = [line.rstrip() for line in sys.stdin]
14    Z       = [ [ (float(z)) for z in line.split(" ")] for line in
          lines[2:] ]
15
16    Z = np.log10(Z)
17
18    dt = W/(overlap*Fs) # timestep --> only progress part of a window
          at the time
19    df = Fs/W           # frequency step
20
21    Y, X = np.mgrid[slice( 0, len(Z) * dt, dt ),
22                    slice( 0, len(Z[0]) * df, df )]
23
24    print "min:\t%f"    % np.min(Z)
25    print "max:\t%f"    % np.max(Z)
26    print "mean:\t%f"   % np.mean(Z)
27    print "median:\t%f" % np.median(Z)
28    print "std:\t%f"    % np.std(Z)
29
30    # vary these parameters to enhance subtle changes in plot
31    z_min = np.median(Z)
32    z_max = np.median(Z) + 2*np.std(Z)
33
34    plt.title(title)
35    plt.pcolor(np.array(X), np.array(Y), np.array(Z), cmap=colormap(),
          vmin=z_min, vmax=z_max)
36    plt.axis([X.min(), X.max(), Y.min(), Y.max()])
37    plt.show()
38
39 def colormap():
40    import matplotlib as mpl
41    return mpl.colors.ListedColormap([(0.0, x, 0.0, 1) for x in np.
          linspace(0, 1, 255)])
42
43 if __name__ == '__main__':
44    main(sys.argv[1:])
```

Listing B.2: plot.py - Plot frequency spectra on time axis

Appendix C

MEM Operation Benchmark

C.1 Forcing Cache Misses

The cache misses making up the MEM operation, as explained in subsection 4.3.2, were generated by running the code given in Listing C.1 in a time-bound loop. Benchmarking the code, to verify the ability to forced increased amount of suffered cache misses, was done by comparing the results of running this code with the results from the code given in Listing C.2. These two pieces of code only differ on one single line, namely line number 22, where in Listing C.1 the random number is used for lookup, while in Listing C.2 the deterministic iteration counter is used.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define L1_SIZE (32768)
5 #define L2_SIZE (262144)
6 #define COUNT (1000)
7
8 int main()
9 {
10      srand(4321);
11      char *data, tmp;
12      int i, r, L;
13
14      /* Allocate memory 1000 times bigger than the L2 cache */
15      L = 1000*L2_SIZE*sizeof(char);
16      data = malloc( L );
17
18      /* Do 1000 random lookups in the array */
19      for ( i = 0; i < COUNT; ++i)
20      {
21          r = rand();
22          tmp = data[r % L];
23      }
24      free(data);
```

```
25 }
```

Listing C.1: 1k_cache_miss.c - Force 1000 cache misses

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define L1_SIZE (32768)
5 #define L2_SIZE (262144)
6 #define COUNT (1000)
7
8 int main()
9 {
10     srand(4321);
11     char *data, tmp;
12     int i, r, L;
13
14     /* Allocate memory 1000 times bigger than the L2 cache */
15     L = 1000*L2_SIZE*sizeof(char);
16     data = malloc( L );
17
18     /* Do 1000 random lookups in the array */
19     for ( i = 0; i < COUNT; ++i)
20     {
21         r = rand();
22         tmp = data[i % L];
23     }
24     free(data);
25 }
```

Listing C.2: 1k_cache_hit.c - Cache miss reference

C.2 Collecting Results

The data behind Figure 4.1 were generated by running the Python script given in Listing C.3 on both the target laptops.

```
1 import subprocess
2
3 def main():
4     values = []
5     for i in range(100):
6         c_A = count_misses_in_A();
7         c_B = count_misses_in_B();
8         values.append(c_A - c_B)
9     print sum(values) / len(values)
10
11
12 def count_misses_in_A():
13     cmd = 'perf stat -e cache-misses:u bin/1k_cache_miss'.split()
```

```
14      p = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess
            .PIPE)
15
16      err, out = p.communicate()
17
18      line = out.split("\n")[3].strip()
19      return int(line[:line.index(' ')].replace("\xc2\xa0", ""))
20
21  def count_misses_in_B():
22      cmd = 'perf stat -e cache-misses:u bin/1k_cache_hit'.split()
23      p = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess
            .PIPE)
24
25      err, out = p.communicate()
26
27      line = out.split("\n")[3].strip()
28      return int(line[:line.index(' ')].replace("\xc2\xa0", ""))
29
30  main()
```

Listing C.3: benchmark_cache_misses.py - Collect data from 1000 runs of the forced cache miss and reference benchmark utilities.

Appendix D
Additional Results

This appendix will present additional results that are not included in the paper. All experiments referred to in this section are described in chapter 4 unless else specified. Generally we have three different experiments; CPU operations described in section 4.3, CPU load described in section 4.2 and decryption described in section 4.4. We are working with two different experimental setups; the portable setup and the lab grade setup, which is explained more in detail in subsection 3.1.2 and subsection 3.1.3 respectively.

D.1 Raspberry PI

All of our recordings of the Raspberry PI is more or less equal and inconclusive, empirically speaking. Figure D.1 represents the frequency spectrogram from capturing the CPU operations experiment.

D.2 Office Environment Recordings

Figure D.2a and Figure D.2b is the results from the running CPU load on the Lenovo T60p in an office environment.

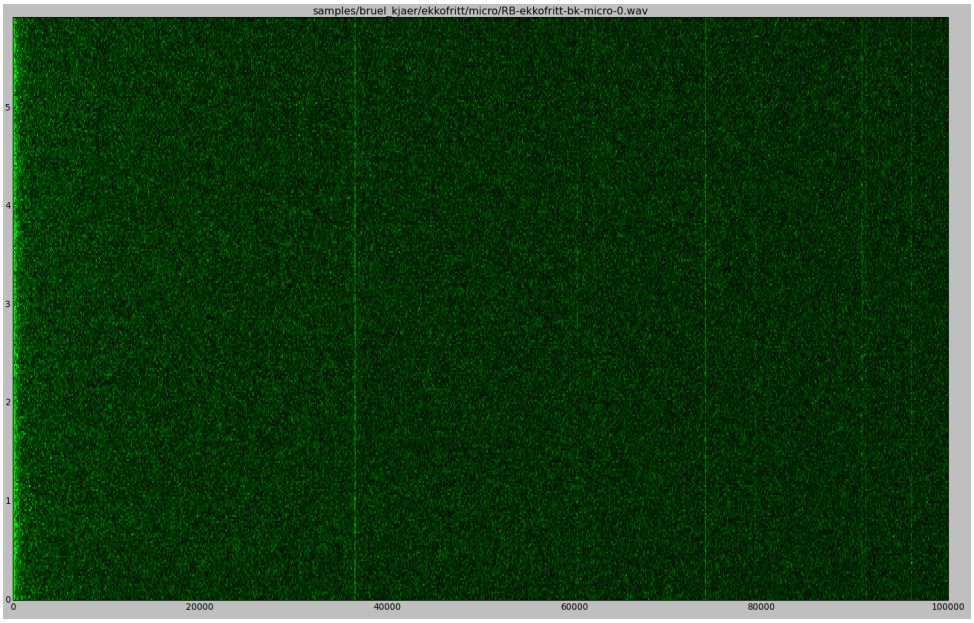
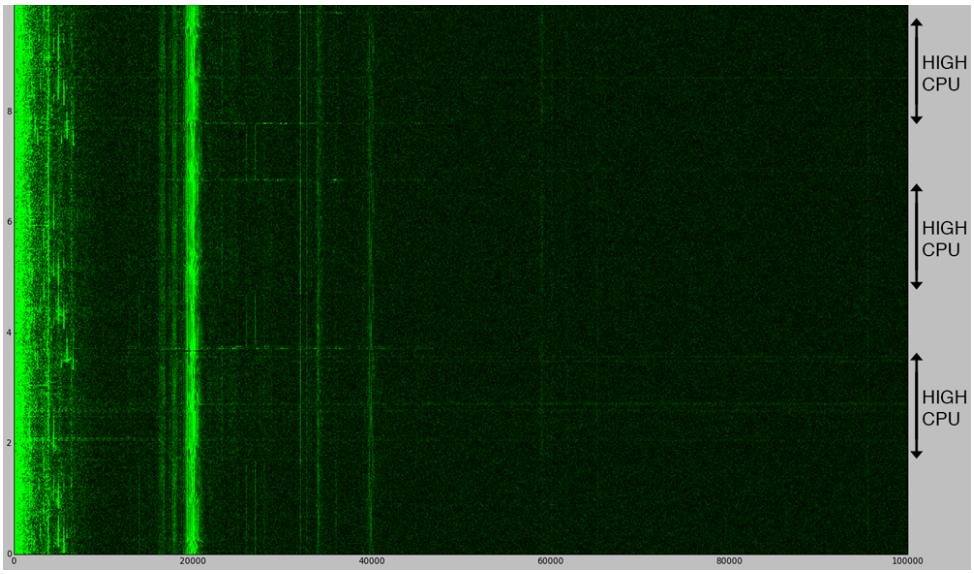
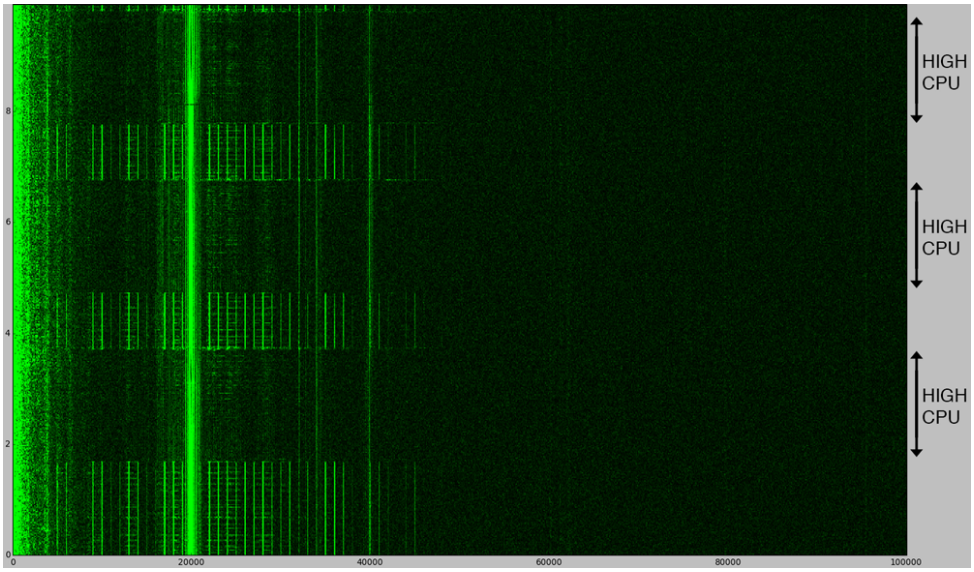


Figure D.1: Acoustic signature (6 sec, 0-100kHz) of the Raspberry PI running the CPU operations. Recorded in an anechoic chamber using the lab-grade setup.



(a) With AC power adopter



(b) Running on battery power

Figure D.2: Acoustic signature (10 sec, 0-100kHz) of the Lenovo T60p when running a high CPU load described in section 4.2. Recorded in an anechoic chamber using the lab-grade setup.