RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*

Daniel Genkin
Technion and Tel Aviv University
danielg3@cs.technion.ac.il

Adi Shamir
Weizmann Institute of Science
adi.shamir@weizmann.ac.il

Eran Tromer
Tel Aviv University
tromer@cs.tau.ac.il

December 18, 2013

Abstract

Many computers emit a high-pitched noise during operation, due to vibration in some of their electronic components. These acoustic emanations are more than a nuisance: they can convey information about the software running on the computer, and in particular leak sensitive information about security-related computations. In a preliminary presentation (Eurocrypt'04 rump session), we have shown that different RSA keys induce different sound patterns, but it was not clear how to extract individual key bits. The main problem was that the acoustic side channel has a very low bandwidth (under 20 kHz using common microphones, and a few hundred kHz using ultrasound microphones), many orders of magnitude below the GHz-scale clock rates of the attacked computers.

In this paper we describe a new *acoustic cryptanalysis* key extraction attack, applicable to GnuPG's current implementation of RSA. The attack can extract full 4096-bit RSA decryption keys from laptop computers (of various models), within an hour, using the sound generated by the computer during the decryption of some chosen ciphertexts. We experimentally demonstrate that such attacks can be carried out, using either a plain mobile phone placed next to the computer, or a more sensitive microphone placed 4 meters away.

Beyond acoustics, we demonstrate that a similar low-bandwidth attack can be performed by measuring the electric potential of a computer chassis. A suitably-equipped attacker need merely touch the target computer with his bare hand, or get the required leakage information from the ground wires at the remote end of VGA, USB or Ethernet cables.

*The authors thank Lev Pachmanov for programming and experiment support during the course of this research.

Contents

1 Introduction

1.1 Overview

Cryptanalytic side-channel attacks target implementations of cryptographic algorithms which, while perhaps secure at the mathematical level, inadvertently leak secret information through indirect channels: variations in power consumption, electromagnetic emanations, timing variations, contention for CPU resources such as caches, and so forth (see [And08] for a survey). Acoustic emanations, i.e., the noise generated by computers, are one such potential channel. Eavesdropping on keyboards and printers has been well-demonstrated [AA04, ZZT05, BDG+10]. Mechanical noise from fans and storage devices such as hard disks are obviously indicative of system activity, but seem to carry very coarse information that is apparently of little use for cryptanalysis.

We focus on a different source of computer noise: vibration of electronic components in the computer, sometimes heard as a faint high-pitched tone or hiss (commonly called "coil whine", though often generated by capacitors). These acoustic emanations, typically caused by voltage regulation circuits, are correlated with system activity since CPUs drastically change their power draw according to the type of operations they perform. However, the bandwidth of these signals is very low: up to 20 kHz for audible signals and commodity microphones, and up to a few hundred kHz using ultrasound microphones. (Beyond these frequencies, air attenuation and reduced microphone sensitivity render the signals undetectable.)[1] Cryptanalytic side-channel attacks typically require measurements with temporal resolution similar to the time scale of the target operation, but here the target cryptographic computation is many orders of magnitude faster (at the GHz scale), so we have no hope of observing individual operations. Moreover, the acoustic signals of interest are very faint. Indeed, a recent survey on side channels surmised that while "acoustic effects have been suggested as possible side channels[2], the quality of the resulting measurements is likely to be low" [KJJR11].

Acoustic cryptanalysis. We show that, despite these difficulties, full key recovery via *acoustic cryptanalysis* is quite feasible on common software and hardware. As a study case, we focus on the GnuPG (GNU Privacy Guard) [Gnu], a popular, cross-platform, open-source implementation of the OpenPGP standard [CDF+07]. We observe that GnuPG's RSA signing (or decryption) operations are readily identified by their acoustic frequency spectrum. Moreover, the spectrum is often key-dependent, so that secret keys can be *distinguished* by the sound made when they are used. The same applies to ElGamal decryption. We devise and demonstrate a *key extraction* attack that can reveal 4096-bit RSA secret keys when used by GnuPG running on a laptop computer, within an hour, by analyzing the sound generated by the computer during decryption of chosen ciphertexts. We demonstrate the attack on various targets and by various methods, including the internal microphone of a plain mobile phone placed next to the computer, and using a sensitive microphone from a distance of 4 meters.

In a nutshell, the key extraction attack relies on crafting chosen ciphertexts that cause numerical cancellations deep inside GnuPG's modular exponentiation algorithm. This causes the special value zero to appear frequently in the innermost loop of the algorithm, where it affects control flow. A single iteration of that loop is much too fast for direct acoustic observation, but the effect is repeated and amplified over many thousands of iterations, resulting in a gross leakage effect that is discernible in the acoustic spectrum over hundreds of milliseconds.

Low-bandwidth power and voltage analysis. With this technique for inducing robust low-bandwidth signals, we revisit other channels and observe that similar leakage can often be induced, acquired, and exploited. For example, in many computers the "ground" potential fluctuates (even when connected

[1] Above a few hundred kHz, sound propagation in the air has a very short range, due to non-linear attenuation and distortion effects, such as viscosity, relaxation and diffusion at the molecular level. The exact attenuation rates depend on frequency, air pressure, temperature and humidity; see [EB72, BK75]. Moreover, the size (and thus sensitivity) of membrane-based microphone transducers is limited by the acoustic wavelength.

[2] Citing an announcement of our early findings [TS04], preceding most results reported in the present paper.

to a grounded power supply) and leaks the requisite signal; this can be measured by touching exposed metal on the computer's chassis, or at the remote end of VGA, USB and Ethernet cables. We can also acquire and exploit these low-frequency signals by power analysis, using measurement of the power supply current, even if the clockrate-scale high frequencies (needed for standard power analysis) have been filtered out.

Chosen-ciphertext channel by e-mail. The key extraction attack requires decryption of ciphertexts adaptively chosen by the attacker. Prior works requiring chosen plaintext or ciphertext typically attacked network protocols such as SSL/TLS [BB05] or WEP [BGW01, BHL06], or used direct access to a protected device's input. To apply the attack to GnuPG, we identified and exploited a new chosen-ciphertext attack vector: OpenPGP encrypted e-mail messages. We specifically targeted Enigmail, a popular plugin to the Thunderbird e-mail client that enables transparent signing and encryption of e-mail messages using GnuPG, following the OpenPGP [CDF+07] and PGP/MIME [ETLR01] standards. Enigmail automatically decrypts incoming e-mail (for notification purposes), as long as the GnuPG passphrase is cached or empty. In this case, an attacker can e-mail suitably-crafted messages to the victims, wait until they reach the target computer, and observe the acoustic signature of their decryption, thereby closing the adaptive attack loop.

Applicability. Our observations apply to many laptop computers, of various vendors and models, running various operating systems. Signal quality and effective attack distance vary greatly, and seem to be correlated with the computer's age.

The attacks apply to all recent versions of the GnuPG 1.x series (up to the latest, 1.4.15, released on 15 Oct. 2013), including the side-channel mitigation recently introduced in GnuPG 1.4.14. Ironically, this side-channel mitigation in GnuPG actually helps our attack, by increasing the aforementioned amplification of the innermost loop. Another natural mitigation, that of placing artificial load on another CPU core to mask the operation, likewise ends up helping the attack by reducing the signal frequencies where the useful leakage resides (and thus the requisite measurement bandwidth). We surmise that acoustic cryptanalysis requires carefully considered countermeasures, of which we discuss several.

For concreteness, our case study focused on a particular cryptographic implementation (that of GnuPG 1.x), chosen ciphertext channel (OpenPGP-encrypted e-mail messages processed by Enigmail), and class of computers (laptops). However, we expect that similar attacks will be feasible for other software, protocols and hardware.

Current status. We have disclosed our attack to GnuPG developers and main distributors as CVE-2013-4576 [MIT13], suggested suitable countermeasures, and worked with the developers to test them. New versions of GnuPG 1.x, GnuPG 2.x and libgcrypt, containing these countermeasures and resisting our current key-extraction attack, were released concurrently with this paper's first public posting. However, some of the effects presented in this paper (such as RSA key distinguishability) remain present.

1.2 Acoustic attack scenarios

To elucidate the possible ramifications of acoustic attacks, we enumerate a number of hypothetical scenarios where they pose a new threat. These exceed the settings of our case study (in software, protocol, hardware or signal analysis), but nonetheless seem quite plausible given the findings, and thus require due consideration and mitigation.

An acoustic attack app. Mobile phones are ubiquitous and contain internal microphones that, as we demonstrate, are of sufficient bandwidth and sensitivity for mounting key extraction attacks. Moreover, they have ample signal processing capabilities, and (using their wireless data connectivity) can close the adaptive chosen-ciphertext loop in real time. Thus, the whole attack could be packaged into a software "app" requiring no special hardware or knowledge. An attacker would install this software, reach physical proximity to the target computer under some pretext, and place the phone appropriately for the duration of the attack. For example, in a meeting, the attacker could innocuously place his phone on the desk

next to the target laptop (as in Figure 4), and obtain the key by the meeting's end. Similar observations apply to other mobile devices with built-in microphones, such as tablets and laptops.

Eavesdropping via compromised mobile device. In a similar vein, a mobile device could be remotely compromised, through any of the numerous known attacks, and the attack code installed. When the device's owner inadvertently places it in the vicinity of a target computer, the mobile device can autonomously attack the target and send the results to the attacker.

Self-eavesdropping. Taken to the extreme, a device containing (or connected to) a microphone may spy on itself. In this scenario, the attacker controls an unprivileged process with no more than microphone recording permissions and network connectivity (e.g., a web page using the HTML Media Capture features or a Flash app, as in existing web-based videoconferencing services). Using these, the attacker can record and analyze cryptographic operations running in a different process, or even a different virtual machine, on that same computer.

Eavesdropping bugs. Acoustic eavesdropping "bugs" are a staple of espionage. Matchbox-sized, battery-operated bugs, with built-in microphones and cellular network connectivity, are readily available for under $30. Traditionally used for eavesdropping on conversations, these may now find additional cryptanalytic use. Other traditional eavesdropping equipment, such as phone bugs, and laser microphones capable of listening through windows from afar, may likewise be repurposed.

Targeted bugs. For best signal acquisition, acoustic bugs can be hidden where they will be placed in close proximity or contact with the computer. For example, laptop computers are placed in a fairly predictable way when placed down on a charging station, presentation podium or a crammed table. An attacker may exploit this to place a hidden microphone, in advance, in close proximity and optimal orientation. Likewise, a cable or a Kensington lock, made conveniently available to visitors, may can contain hidden microphones that will be placed in perfect alignment when the plugged into the laptop.

Eavesdropping en masse. In a setting where multiple devices are placed in proximity, such as a server room, an attacker could compromise some device equipped with a microphone. The software would then record the neighboring devices, disambiguate their (typically distinct) acoustic signatures, and mount attacks on each of them. After transmitting the findings, the attack software would self-erase, leaving no anomalous evidence in hardware.

Faraday cages and air gaps. In sensitive applications, the dangers of network and side-channel attacks are recognized, and critical computers are protected by measures such as air gaps, Faraday cages, and power supply filters. Alas, none of these eliminate acoustic leakage. In particular, Faraday cages containing computers require ventilation, typically by means of vents covered with perforated sheet metal or metal honeycomb, which are very effective at attenuating compromising electromagnetic radiation ("TEMPEST"), yet are fairly transparent to acoustic emanations, for the sake of air flow. See Figure 1. Thus, even if all other communication and side channels are carefully controlled, acoustic emanations can still escape the enclosure and be acquired by nearby devices.

1.3 Related work

Auditory eavesdropping on human conversations is a common practice, first published several millenia ago [Gen]. Analysis of sound emanations from mechanical devices is a newer affair, with precedents in military context such as identification of vehicles (e.g., submarines) via the sound signature of their engine or propeller. There are anecdotal stories of computer programmers and system administrators monitoring the high-level behavior of their systems by listening to sound cues generated by mechanical system components, such as hard disk head seeks; however, these do not appear very useful in the presence of caching, delayed writes and multitasking.

Electromechanical ciphers. Wright [Wri87, pp. 103–107] provides an informal account of MI5 and GCHQ using a phone tap to eavesdrop on a Hagelin cipher machine in the Egyptian embassy in London,

Figure 1: Examples of electromagnetic shielding. The two computers meet the strict NSTISSAM TEMPEST/1-92 Level I requirements for electromagnetic side-channel emanations, yet have ample ventilation holes (catalog photos, left: Advanced Programs, Inc. model DWT-105; middle: EMCON Emanation Control Limited Core 2 Quad workstation). On the right are shielded honeycomb air vents (Parker Chomerics CHO-CELL panels of several sizes, "especially designed for high frequency environments and to meet TEMPEST specifications").

by counting the clicks during the rotors' secret-setting procedure.[3] Acoustic emanations from mechanical and electromechanical ciphers are also mentioned in historic US government documents (see below).

Keyboard acoustic emanations. Keystroke timing patterns are known to be a way to identify users, and more recently, also to leak information about the typed text (see [SWT01] and the references therein). Timing of keystrokes can, of course, be detected acoustically. Later work by Asonov and Agrawal [AA04], improved by Zhuang et al. [ZZT05], Berger et al. [BWY06], and by Halevi and Saxena [HS10], shows that keys can also be distinguished individually by their sound, due to minute differences in mechanical properties such as their position on a slightly vibrating printed circuit board. While these attacks are applicable to data that is entered manually (e.g., passwords), they are not applicable to large secret data, such as RSA keys, that is not manually typed.

Acoustic emanations from printers. The acoustic emanations produced by dot-matrix printers can also be used for recovering information. Backes et al. [BDG+10] show that the sound of needles inside the printing head of a dot matrix printer can be used to recover the printed information. While their dictionary-based approach indeed manages to correctly recover up to 72% of printed English words, it does not extend to high-entropy data such as random passwords.

Acoustic emanations from electronic components. Tromer and Shamir first observed in their presentation [TS04] that acoustic emanations from different motherboard components leak information about the instructions performed by the target's CPU. They also demonstrated that it is possible to acoustically distinguish between GnuPG RSA keys (Sections 3 and 4), but were not able to recover individual key bits. LeMay and Tan [LT06] showed how these instruction-dependent emanations can be scheduled to create an acoustic covert channel for transmitting information across an air gap. This requires attackers' code to run on the transmitting computer and have access to the desired information — a very strong assumption.

Power analysis. The power consumed by an electrical device varies according to the operations it currently performs. Thus, by measuring the voltage across a resistor placed in series with the device's power supply, at a very high sampling rate, an attacker might learn something about the individual operations performed by the device, and thus deduce information about its secret internal state. Indeed, it was shown that many common ciphers such as DES, AES, RSA, as well as many types of hardware, are vulnerable to this type of leakage (see [MOP07] and [KJJR11] and the references therein).

Power analysis via the USB port. Oren and Shamir [OS06] observed that the voltage on power lines

[3] Wright refers to this technique by the codename "ENGULF" [Wri87, p. 107].

of USB ports leaks information about the target computer's activity, which suffices for distinguishing idle time from OpenSSL cryptographic operations. This is possible even when the USB ports are disabled in software or by the overload protection hardware.

Power analysis via the electrical outlet. Clark et al. [CMR+13] observed that power analysis can also be performed by tapping the AC electrical outlet to which the target is connected. Concretely, they show that one can identify, with high probability, the web pages loaded by a web browser on the target machine. Unlike traditional power analysis techniques, which require bandwidth of several hundred MHz, their technique only requires a bandwidth of a few kHz.

Timing attacks. Timing attacks, introduced by Kocher [Koc96], exploit the fact that some sensitive computational operations vary in time depending on their secret inputs. By measuring the running time of the operation, an attacker can learn information about these inputs. Inherently, these attacks assume that the attacker gets some response from the target in order to measure its running time. Utilizing such responses from the target, the works of Kocher [Koc96], Brumley and Boneh [BB05], and Brumley and Tuveri [BT11] demonstrate attacks on many popular ciphers and encryption schemes such as DSS, RSA and ECDSA.

Government publications. NSA's partially-declassified NACSIM 5000 ("TEMPEST Fundamentals") [Nat82] mentions acoustic emanations, and defines them as "emanations in the form of free-space acoustical energy produced by the operation of a */sic/* purely mechanical or electromechanical device equipment". These are described as follows. "[S]ources of [Compromising Emanations] exist where mechanical operations occur and sound is produced. Keyboards, printers, relays — these produce sound, and consequently can be sources of compromise." The focus is clearly on intentionally-moving mechanical elements. No further details are given in the declassified portions, beyond referencing three other NSA documents: NACSEM 5103 ("Compromising Emanations Laboratory Test Standard, Acoustics"), NACSEM 5104 ("Technical Rationale" for aforementioned) and NACSEM 5105 ("Administrative Guidelines" for aforementioned). These three documents, all from Oct. 1970, remain classified, but have since been canceled by memorandums NSTISSC-052-95 and NSTISSC-002-98 (see [Com13]). A comprehensive list of past and present publications by the US government's Committee on National Security Systems [Com13], which lists the above and other extensive treatments of side channels, does not list any additional documents which (judging by title or declassified content) discuss acoustic emanations. One is thus led to conclude that acoustic emanations from historical *mechanical* and *electromechanical* devices are well studied, but acoustic emanations from modern *electronic* components are not addressed at all in the usual US government publication venues. Nor could pertinent public documents by other governments be found.

Cache attacks on GnuPG. Yarom and Falkner [YF13] recently presented a cache attack on GnuPG. Their cache contention side channel allows an effective sample rate of 1.5 MHz, which (unlike acoustic leakage) is high enough for a direct simple power analysis attack on the square-and-multiply algorithm used in GnuPG 1.4.13. This attack requires the target computer to execute malicious cache-probing code written by the attacker. Ironically, the mitigation of this attack in GnuPG 1.4.14 somewhat *assisted* our attack (see Section 9.1).

1.4 Paper outline

The paper is organized as follows. We start by introducing our experimental setup (Section 2). We then show that acoustic leakage is indeed correlated with the code running on the target computer, and discuss the various sources of this leakage (Section 3). We show that acoustic leakage easily allows distinguishing between different RSA keys (Section 4). We proceed to present our approach for extracting RSA keys from GnuPG as well as report on the empirical performance of our attack (Section 5). We then explain why our approach works by analyzing the relevant code of GnuPG (Section 6), and give a more detailed report of our empirical results (Sections 7 and 8). Next, we show that acoustic key distinguishability is

possible on other ciphers and that acoustic RSA key extraction is possible from other versions of GnuPG (Section 9). We empirically compare our attacks to timing attacks (Section 10). Finally, we discuss some countermeasures against acoustic cryptanalysis (Section 11).

2 Experimental setup

We consider three experimental setups representing various trade-offs between costs, portability and measuring capabilities. The first is a lab-grade setup, which offers the best-available measurement capabilities but is expensive and not easily portable. The second is a portable measurement setup, which offers somewhat lower measurement capabilities, but is battery-operated and fits in a briefcase. The third setup is readily available and fits in a pocket: plain mobile phone.

In many cases, we operate critical equipment (especially the microphones) well beyond its nominal operating range, where suitable specifications do not exist. Thus, the task of constructing a good setup often requires experimentally-tested educated guesses. In Appendix C we present a detailed discussion of our lab-grade setup and its rationale. We proceed to briefly describe the details of each of the experimental setups.

2.1 Lab-grade setup

This setup aims for the best possible acoustic acquisition quality (in terms of sensitivity, noise and frequency response) and high flexibility in measurement configuration. To this end, it uses professional lab equipment from Brüel&Kjær, National Instruments and Mini-Circuits, in conjunction with custom-built electronics. This setup is AC-operated and not easily portable. (As shown later, it can be miniaturized, at some cost in flexibility and signal quality.)

The beginning of the measurement chain, and the only component that is location-sensitive and must be placed in proximity to the target, is the *microphone*, i.e., the transducer which converts acoustic signals (changes in air pressure) to electric signals (voltage). Our lab-grade setup uses Brüel&Kjær condenser microphones, in which the transducer is embodied in a *microphone capsule*. We use 3 such capsules, with different frequency vs. sensitivity tradeoffs: Brüel&Kjær model 4939 (up to 350 kHz), 4190 (up to 40 kHz) and 4145 (up to 21 kHz). See Figure 2 for photos.

These capsules are operated by a Brüel&Kjær 2669 preamplifier, powered by a Brüel&Kjær 2804 microphone power supply. We amplify the signal using an (augmented) Mini-Circuits ZPUL-30P low-

Figure 2: Microphone capsules (from right to left): Brüel&Kjær 4145 (1" diameter), 4190 (1/2" diameter) and 4939 (1/4" diameter). A common pencil is included for size comparison.

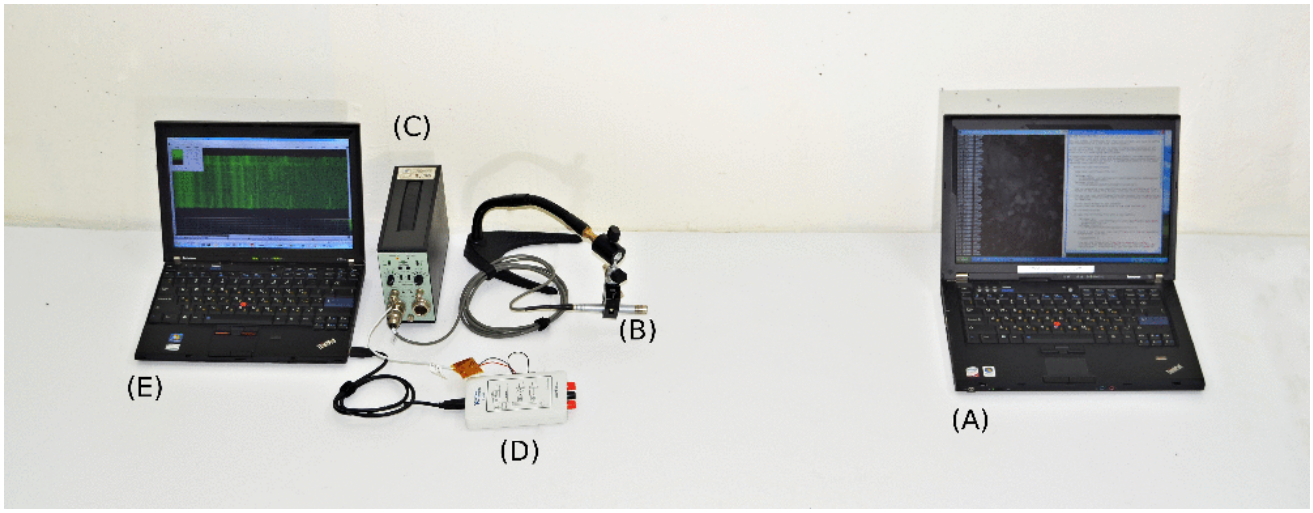


Figure 3: Photograph of our portable setup. In this photograph (A) is a Lenovo ThinkPad T61 target, (B) is a Brüel&Kjær 4190 microphone capsule mounted on a Brüel&Kjær 2669 preamplifier held by a flexible arm, (C) is a Brüel&Kjær 5935 microphone power supply and amplifier, (D) is a National Instruments MyDAQ device with a 10 kHz RC low-pass filter cascaded with a 150 kHz RC high-pass filter on its A2D input, and (E) is a laptop computer performing the attack. Full key extraction is possible in this configuration, from a distance of 1 meter (see Section 5.4).

noise amplifier, filter it, and digitize it (at up to 1.25 Msample/sec) using a National Instruments 6356 data acquisition device. See Appendix C for further details and discussion.

2.2 Portable setup

We built an alternative setup that is portable (fits in a briefcase) and battery-operated, yet maintains excellent sensitivity and noise up to 100 kHz. To do so, we kept the Brüel&Kjær capsules and preamplifier, but replaced the power, amplification and analog-to-digital components. See Figure 3.

Here, the microphone power, amplification and some filtering are done by an integrated, battery-operated Brüel&Kjær 5935 microphone power supply. After a self-built 10 kHz RC low-pass filter cascaded with a 150 kHz RC high-pass, analog-to-digital-conversion is done by the compact, USB-operated National Instruments MyDAQ device. The MyDAQ device captures at 200 Ksample/sec, and is thus Nyquist-limited to 100 kHz.[4] The Brüel&Kjær 5935 amplifier is likewise limited to a frequency of 100 kHz. We later show that this often suffices for the attack.

2.3 Mobile-phone setup

In this setup, we try to use the most ubiquitous compact hardware that is readily available to every potential attacker: a mobile phone (see Figure 4). The small size and low cost come at the cost of low quality microphones and amplifiers, that are not designed for the signal frequencies and amplitudes sought by the attack.

We used several Android phones, with similar results: HTC Sensation, Samsung Galaxy S II and Samsung Galaxy Note II. Recording was done by a custom Android app, accessing the internal mi-

[4] High-end sound cards reach a 192 Ksample/sec rate and can be used instead.

Figure 4: Physical setup of a key recovery attack. A mobile phone (Samsung Note II) is placed 30 cm from a target laptop. The phone's internal microphone points towards the laptop's fan vents. Full key extraction is possible in this configuration and distance (see Section 5.4).

crophone. The specifications of the microphone, amplification, filtering and A2D hardware were not available to us. We observe that sensitivity is lower, and noise is higher than in the above setups. Frequency response is also very limited: upper bounded by the 24 kHz Nyquist frequency (48 kS/s sample rate), and in practice much lower due to transducer and filter design (recall that cellular speech codecs filter out audio beyond 4 kHz).

2.4 Distant acquisition

Parabolic microphones. The range of our attack can be greatly enhanced by using a parabolic reflector, which focuses incoming planar sound waves into a single focal point. To this end, we placed the Brüel&Kjær 4145 microphone capsule at the focal point of a parabolic reflector (plastic, 56 cm diameter, 10 cm focal distance, $40 from eBay), held by a self-built holder (see Figure 5). As discussed in Section 5.4, this increases the effective range of our key extraction attacks from 1 meter to 4 meters (see Figure 6).

Laser vibrometers. We conjecture that laser microphones and laser vibrometers will greatly increase the effective range of attacks, given an optical line of sight to a reflecting surface on, or near, the target computer. This will be studied in future works.

Figure 5: Parabolic microphone: Brüel&Kjær 4145 microphone capsule and 2669 preamplifier, in front of a transparent parabolic reflector (56 cm diameter), using a self-built attachment, on a tripod.

Figure 6: Parabolic microphone (same as in Figure 5), attached to the portable measurement setup (in a padded briefcase), attacking a target laptop from a distance of 4 meters. Full key extraction is possible in this configuration and distance (see Section 5.4).

3 Observing acoustic leakage

3.1 Distinguishing various CPU operations

We begin our analysis of low-bandwidth leakage by attempting to distinguish various operations performed by the CPU of the target computer. For this purpose we wrote a simple program that executes (partially unrolled) loops containing one of the following x86 instructions: HLT (CPU sleep), MUL (integer multiplication), FMUL (floating-point multiplication), main memory access (forcing L1 and L2 cache misses), and REP NOP (short-term idle). While it was possible (especially when using the lab-grade setup) to distinguish between some CPU operations on almost all the machines we tested, some machines (such as Compaq Evo N200 or Sony Vaio VGN-T350P) have a particularly rich leakage spectrum. Figure 7 shows a recording of the Evo N200 laptop while executing our program using the Brüel&Kjær 4939 high frequency microphone capsule.

As can be seen in Figure 7, the leakage of the Evo N200 is present all over the 0–310 kHz spectrum. Moreover, different types of operations can be easily distinguished. While the Compaq Evo N200 laptop computer is quite antiquated, similar leakage (in lower frequencies ranges) was detected using the lab-grade setup and the Brüel&Kjær 4190 capsule on current models (Lenovo ThinkPad W530). See Figure 8 for comparison between three operations using various target computers.

3.2 Distinguishing various code lengths

Acoustic measurements can also be used to distinguish loops of different lengths executing the same instruction types. For example, the leakage produced by a program executing 20,000 ADD instructions in an infinite loop is different from a program executing 50,000 ADD instructions in an infinite loop. Figure 9 shows a recording of the Evo N200 executing 1 sec loops of ADD instructions. Each loop is repeated for 1 sec, but the size of the code inside the loop (and thus the number of repetitions) differs. As can be seen in Figure 9, the various code sizes can be easily distinguished.

3.3 Leakage source

Acoustic or EM? To ascertain that the obtained signal is truly acoustic rather than electromagnetic interference picked up by the microphone, we simply placed a sheet of non-conductive sound-absorbing material (e.g., cork, foam or thick cloth) in front of the microphone. Figure 10 shows an acoustic recording using the same setup as in Figure 7, except a sheet of cork is placed between the microphone and the laptop. As can be seen, the signal is severely attenuated. We conclude that the measured signals are indeed acoustic emanations.

Culprit components. The exploited acoustic emanations are clearly not caused by fan rotation, hard disk activity, or audio speakers — as readily verified by disabling these. Rather, it is caused by vibrations of electrical components in the power supply circuitry, familiar to many as the faint high-pitched whine produced by some devices and power adapters (commonly called "coil whine", though not always emanating from coils). The precise physical source of the relevant emanations is difficult to characterize precisely, since it varies between target machines, and is typically located in the hard-to-reach innards. Moreover, acoustic localization is difficult due to mechanical coupling of soldered components, and due to acoustic reflections. Nonetheless, our experimentation with the microphone placement invariably located the strongest useful signals in the vicinity of the on-board voltage regulation circuit supporting the CPU. Indeed, modern CPUs change their power consumption dramatically (by many watts, and by orders of magnitude) depending on software load, and we conjecture that this affects

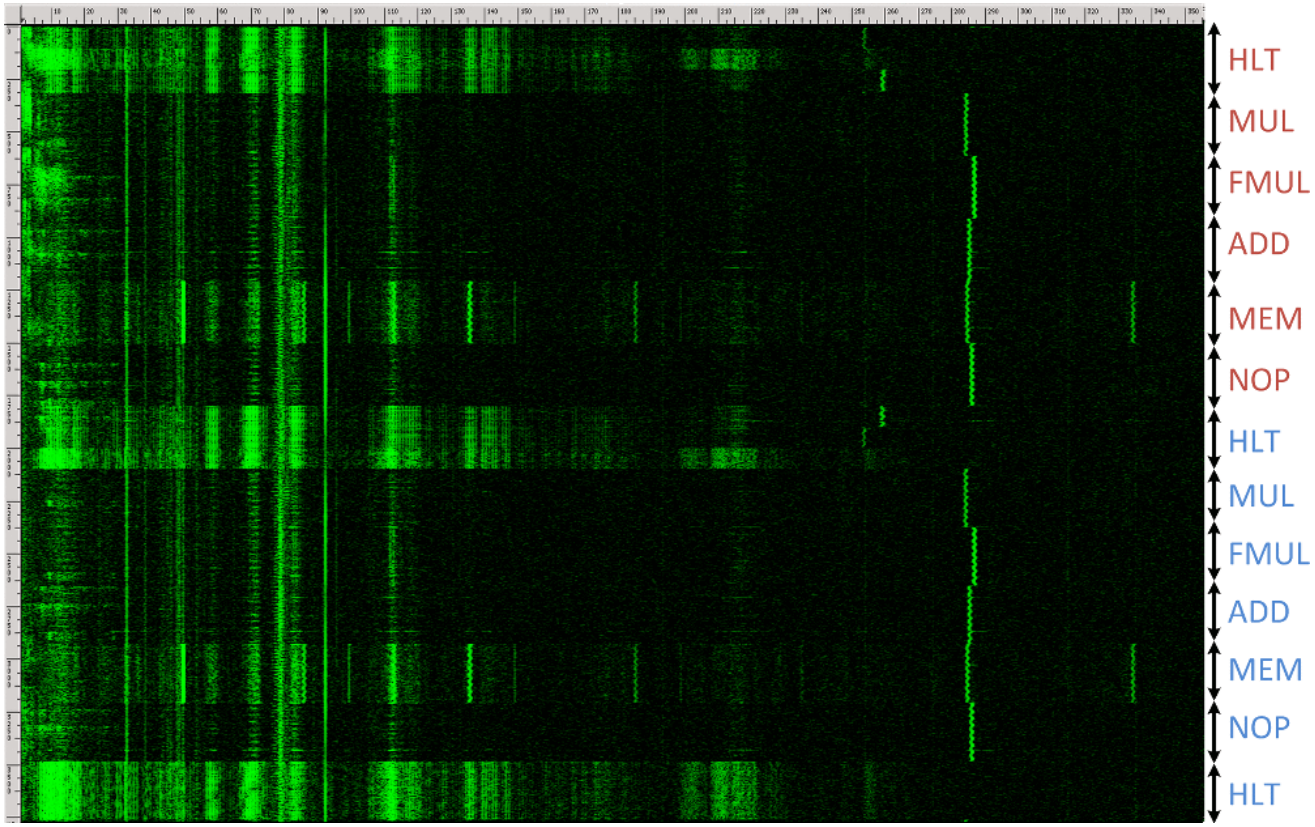
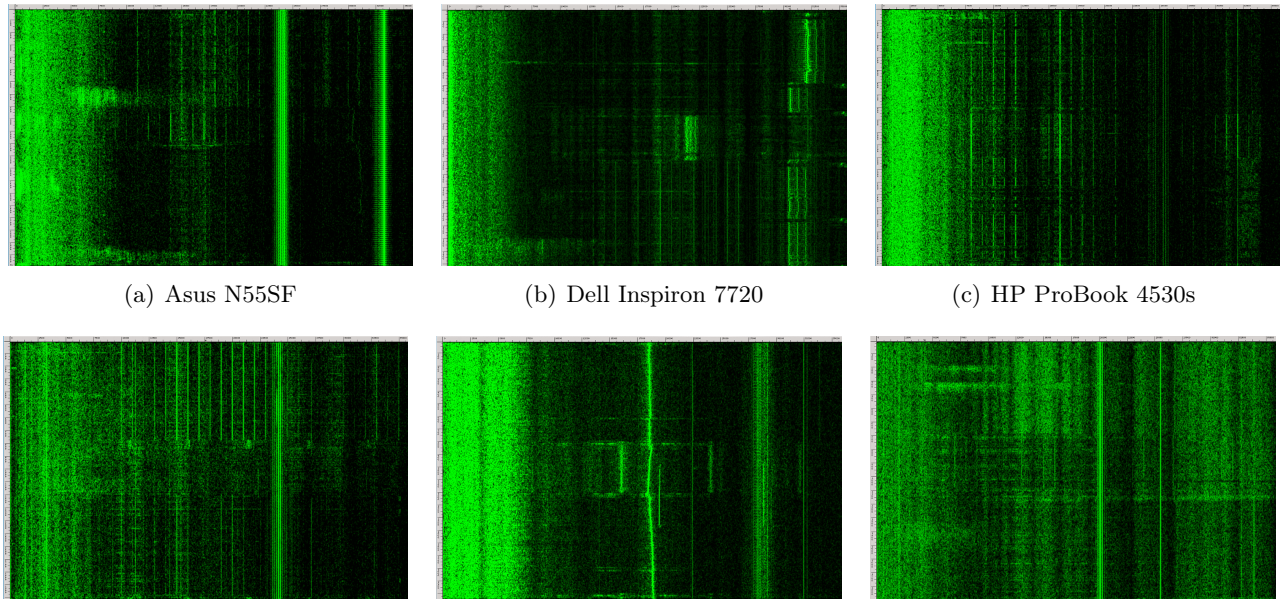


Figure 7: Acoustic measurement frequency spectrogram of a recording of different CPU operations using the Brüel&Kjær 4939 microphone capsule. The horizontal axis is frequency (0–310 kHz), the vertical axis is time (3.7 sec), and intensity is proportional to the instantaneous energy in that frequency band.



(a) Asus N55SF (b) Dell Inspiron 7720 (c) HP ProBook 4530s

(d) HP Pavilion Sleekbook 15-b005ej (e) Samsung NP300V5A (f) Lenovo ThinkPad W530

Figure 8: Acoustic emanations (2 sec, 0–35 kHz) of various target computers performing MUL, HLT and MEM in this order. Note that the three operations can be distinguished on all machines.

and modulates the dynamics of the pulse-width-modulation-based voltage regulator (see also the power analysis in Appendix A). At times, more remote stages of the power supply (e.g., laptops' external AC-DC "power brick" adapter) exhibited software-dependent acoustic leakage as well.

In one attempt at better localization, we opened a desktop PC and pointed the high-frequency Brüel&Kjær 4939 microphone capsule at components on an (antiquated) PC Chips M754LMR motherboard. The strongest acoustic signal was observed around a bank of 1500 μF electrolytic capacitors, part of the CPU's voltage regulator. We cooled down these capacitors, one at a time, by locally spraying them with Quik-Freeze (non-conductive, non-flammable, "will freeze small areas to $-48°$C"). For precisely one of these capacitors, cooling down dramatically affected the acoustic signal (see Figure 11). We conclude that this capacitor, or more generally the voltage regulation circuit of which it is part, is the likely source of the acoustic emanation.[5] This electrolytic capacitor had a bulging vent, indicating deterioration of its internal electrolyte.

Such failure of electrolytic capacitors is very common as computers age, and anecdotally seems correlated with prolific acoustic emanations (though we have also observed noisy computers without any obviously-faulty capacitors). More generally, we observed strong positive correlation between machine age, in terms of calendar time and usage, and the cryptanalytic usefulness of their acoustic emanations.

We also attempted localization by shifting a small piece of cork sheet (2 cm diameter) over the motherboard surface, keeping the microphone in a fixed position. The greatest signal attenuation was observed when the cork sheet occluded the voltage regulator's components (capacitors and coils).

[5]The temperature change affected the capacitor's mechanical properties, but also its electrical properties, which in turn changes the dynamics of the circuit and affects other components. It is unclear which effect is observed.

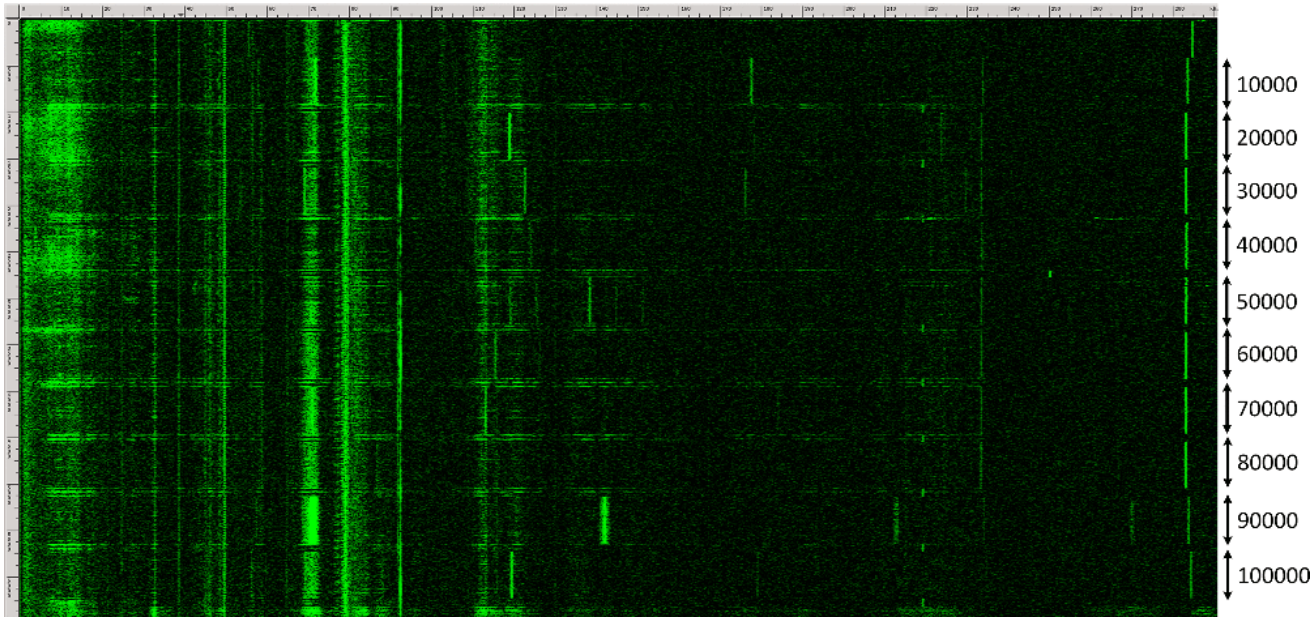


Figure 9: Acoustic signature (6 sec, 0–300 kHz) of loops in different lengths of ADD instructions executed on the Evo N200. The loop length is indicated next to its acoustic signature. The recording was performed using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule.

3.4 Microphone placement

The placement of the microphone relative to the laptop body has a large influence on the obtained signal. Ideally, we would like to measure acoustic emanations as close as possible to the CPU's on-board power supply located on the laptop's motherboard. Unfortunately, this requires major physical tampering with the laptop such as taking it completely apart or at least removing its keyboard.

Such blunt tampering will be immediately detected and is thus considered by us impractical. Luckily, modern laptop computers have a substantial cooling system for heat dissipation, with a fan that requires large air intake and exhaust holes. In addition, there are typically numerous holes and gaps for ports such as USB, Express Card slot, SD card reader, and Ethernet port. Each of the above ports has proven useful on some computers. For modern ThinkPad laptops, typically the best microphone placement is near the Ethernet port or the fan exhaust vent.

While the fan exhaust vent typically offers good acoustic access to the culprit power supply components, placing the microphone in the exhaust air flow creates strong wideband noise (which can overload the amplifier at high gain). Also, electromagnetic interference, caused by the fan motor and PWM driving circuit, can disrupt the acoustic leakage. This can be mitigated by placing the microphone at sufficient distance from the vent, placing the microphone at the edge of the vent outside the air stream, or (as we implemented) recognizing the fan disturbance in the signal and pausing cryptanalysis until the machine cools down and turns off its fan.

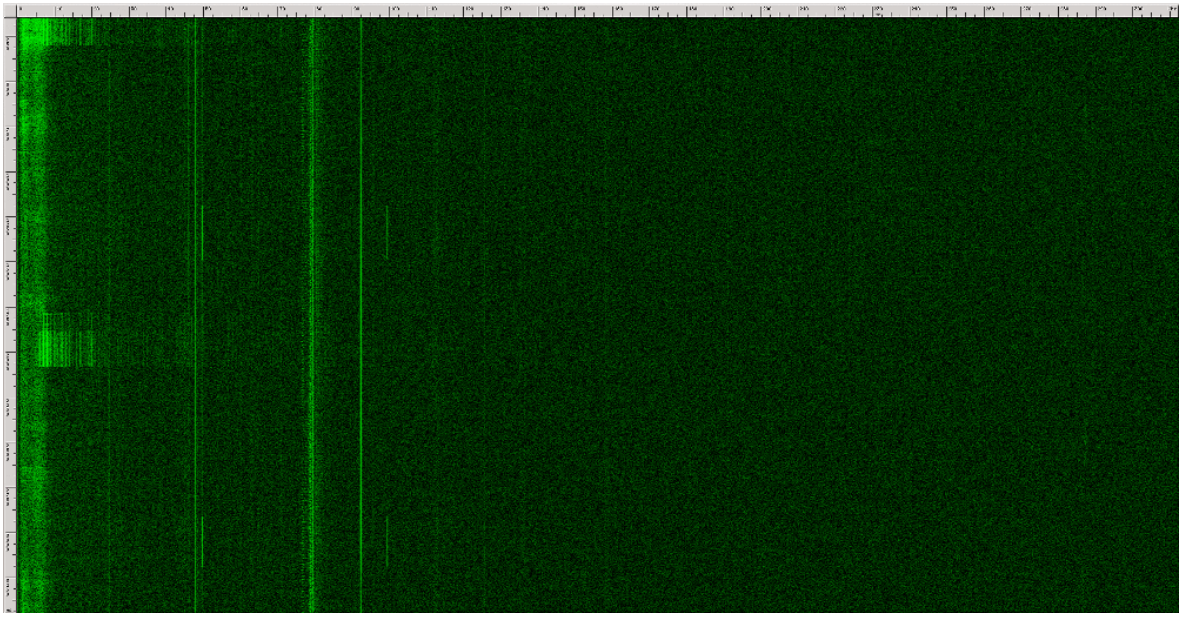


Figure 10: Acoustic measurement frequency spectrogram (3.7 sec, 0–310 kHz) of a recording of different CPU operations using the Brüel&Kjær 4939 microphone capsule in identical physical conditions and measurement setup as in Figure 7, except this time the capsule is acoustically shielded by a cork sheet. The signal is almost completely muffled.

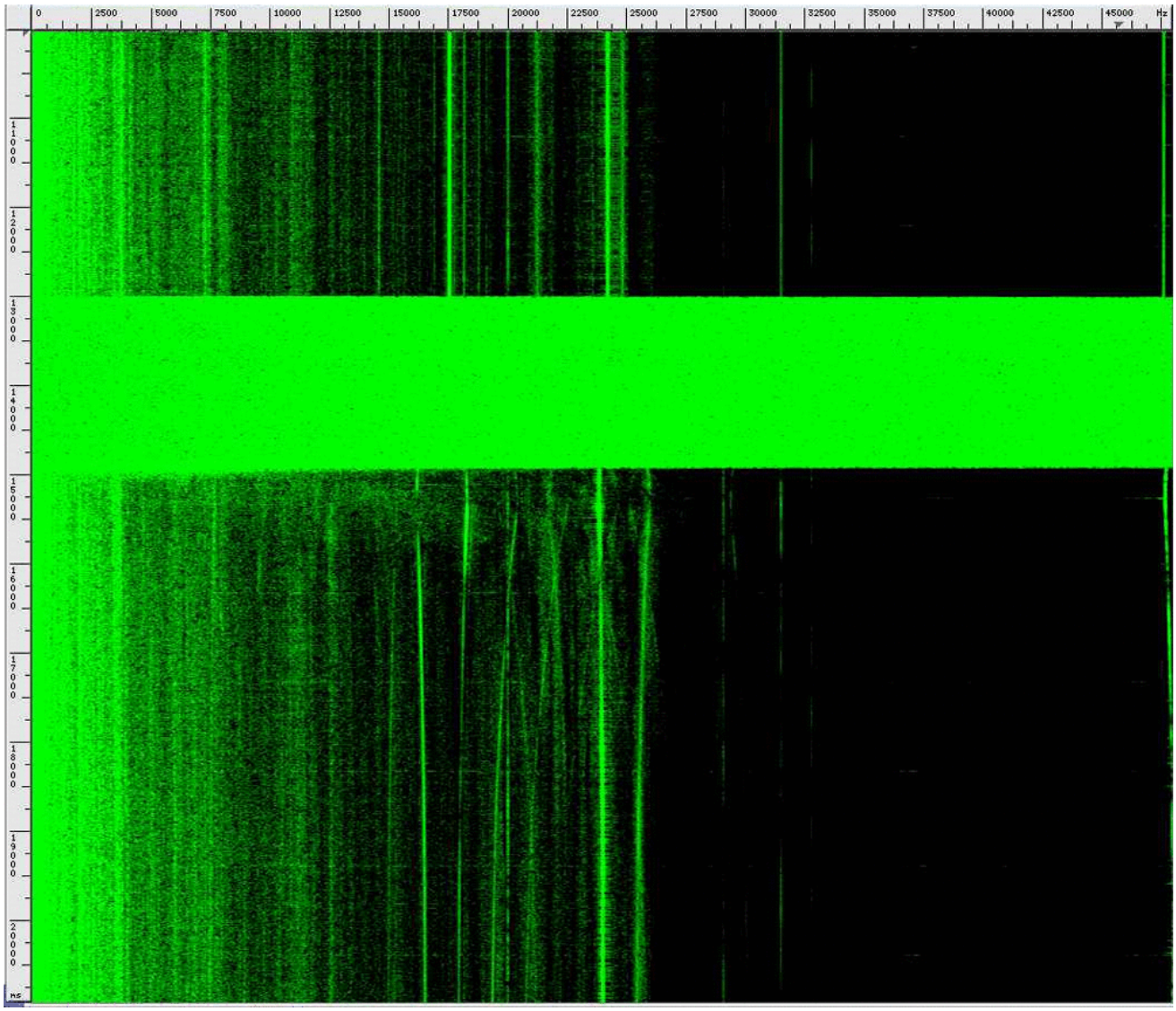


Figure 11: Acoustic measurement frequency spectrogram (2 sec, 0–45 kHz) of MUL operations while applying Quik-Freeze spray to a capacitor. After the spraying (which is loud enough to saturate the measurement), signal components shift by as much as 10Hz.

4 GnuPG RSA key distinguishability

The results in Section 3 demonstrate that it is possible, even when using a very low-bandwidth measurement of 35 kHz, to obtain information about the code executed by the target machine. While this is certainly some form of leakage, it is not clear how to use this information to form a real key extraction attack on the target machine. In this section, we show that some useful information about a secret key used by the target machine can be obtained from the acoustic information, even though it is still unclear how to use it to derive the full key. In particular, we demonstrate that the acoustic information obtained during a single RSA secret operation (such as ciphertext decryption or signature generation) suffices in order to determine which of several randomly generated keys was used by the target machine during that operation.

Throughout this paper, we target a standard and commonly used RSA implementation, GnuPG (GNU Privacy Guard) [Gnu], a popular open source implementation of the OpenPGP standard available on all major operating systems. We focus on the GnuPG 1.x series and its internal cryptographic library (including the side-channel countermeasures recently added in GnuPG 1.4.14; see Section 9.1). The effects presented in the paper were observed on a variety of operating systems (Windows 2000 through 7, Fedora Core 2 through 19), GnuPG versions 1.2.4 through 1.4.15, and target machines (mainly old ThinkPad models, due to their availability to us). While our attack ran on numerous combinations of these elements, the experimental data in this paper (unless stated otherwise) uses Windows XP and GnuPG version 1.4.14 compiled with the MinGW gcc version 4.6.2 compiler [Min] with its default options.

4.1 The sound of a single secret key

Figure 12 depicts the spectrogram of two identical RSA signing operations in sequence, using the same 4096 bit key and message. Each signing operation is preceded by a short delay during which the CPU is in a sleep state. Figure 12 contains several interesting effects. The delays, where the computer is idle, are manifested as bright horizontal strips. Between these strips, the two signing operations can be clearly distinguished. Halfway through each signing operation there is a transition at several frequency bands (marked with yellow arrows). This transition corresponds to a detail in the RSA implementation of GnuPG. For public key $n = pq$, the RSA signature $s = m^d \mod n$ is computed by evaluating $m^{d \mod (p-1)} \mod p$ and $m^{d \mod (q-1)} \mod q$, and combining these via the Chinese Remainder Theorem. The first half of the signing operation corresponds to the exponentiation modulo p, and the second to the exponentiation modulo q. Note that the transition between these secret modules is clearly visible. Moreover, this effect is consistent and reproducible (in various frequency ranges) not only on the Evo N200, but on many modern machines made by various manufacturers. For example, Figure 13 contains the recording of RSA signatures executed on a Lenovo ThinkPad T61 using the same key as in Figure 12. Note that while not as prominent as in the case of the Evo N200, the RSA operations are clearly visible at 34–36 kHz.

4.2 Distinguishing between RSA secret keys

In addition to measuring the duration and internal properties of individual RSA secret-key operations, we have investigated the effect of different keys. Having observed that the acoustic signature of modular integer exponentiation depends on the modulus involved, one may expect different keys to cause different sounds. This is indeed the case, as demonstrated in Figure 14. Here, we used GnuPG 1.4.14 to sign a fixed message using five different 4096 bit RSA keys randomly generated beforehand. Each signature is preceded by a short CPU sleep in order to visually separate these signatures. It is readily observed that

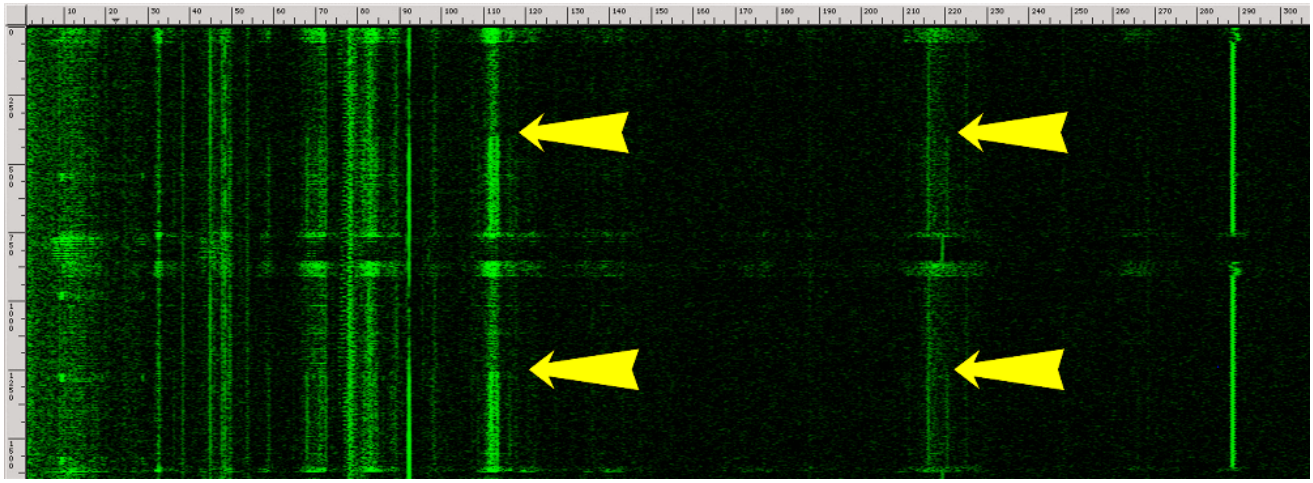


Figure 12: Acoustic signature (1.6 sec, 0–300 kHz) of two GnuPG RSA signatures executed on the Evo N200. The recoding was made using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule. The transitions between p and q are marked with yellow arrows.

each signature (and in fact, each exponentiation using modulus p or q) has a unique spectral signature. This ability to distinguish keys is of interest in traffic-analysis applications[6]. It is likewise possible to distinguish between algorithms, between different implementations of an algorithm, and between different computers (even of the same model) running the same algorithm. Again, this effect is consistent and reproducible (in various frequency ranges) not only on the Evo N200, but on various modern machines and manufacturers (see Figure 15 for a recording of RSA signatures, using the keys from Figure 14 executed on a Lenovo ThinkPad T61).

[6]For example, observing that an embassy has now decrypted a message using a rarely employed key, heard before only in specific diplomatic circumstances, can form valuable information.

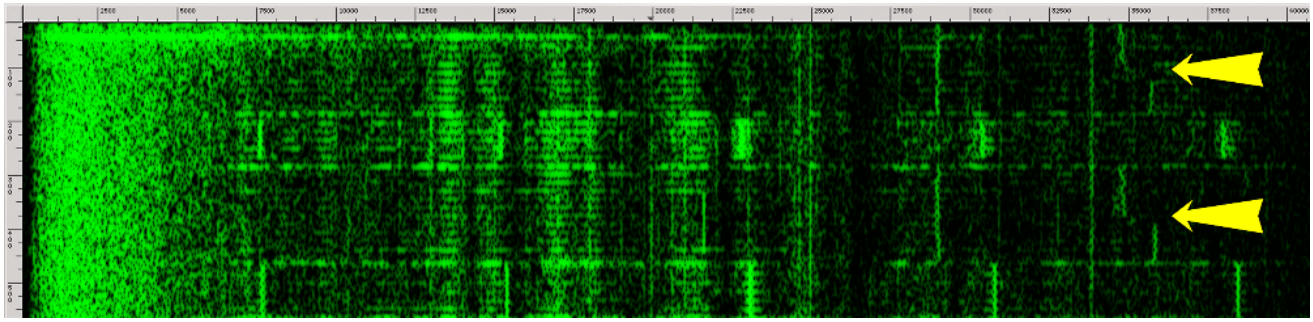


Figure 13: Acoustic signature (0.5 sec, 0–40 kHz) of two GnuPG RSA signatures executed on Lenovo ThinkPad T61. The recoding was made using the lab-grade setup and the Brüel&Kjær 4190 microphone capsule. The transitions between p and q are marked with yellow arrows.

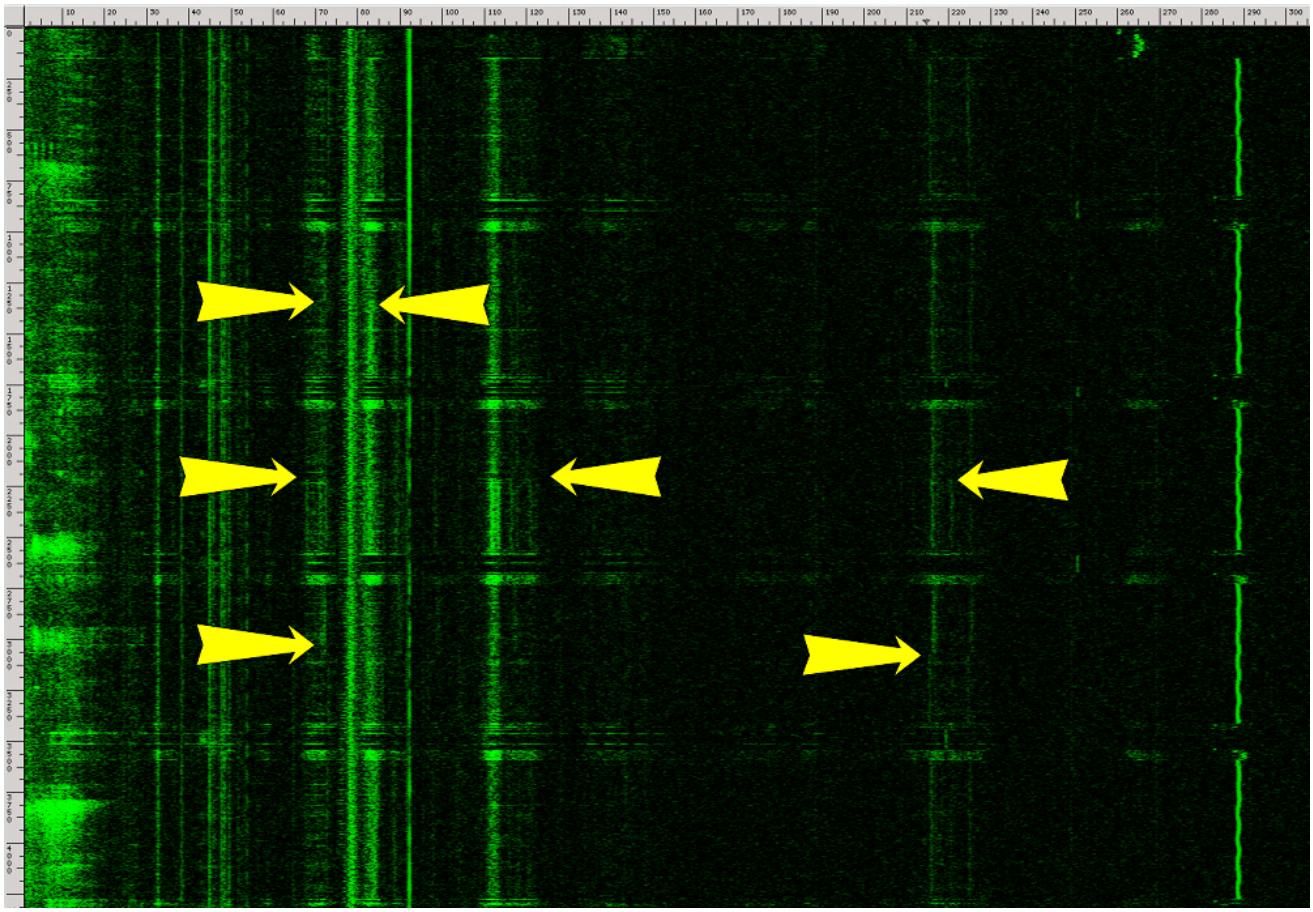


Figure 14: Acoustic signature (4 sec, 0–300 kHz) of five GnuPG RSA signatures executed on Evo N200. The recoding was made using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule. The transitions between p and q are marked with yellow arrows.

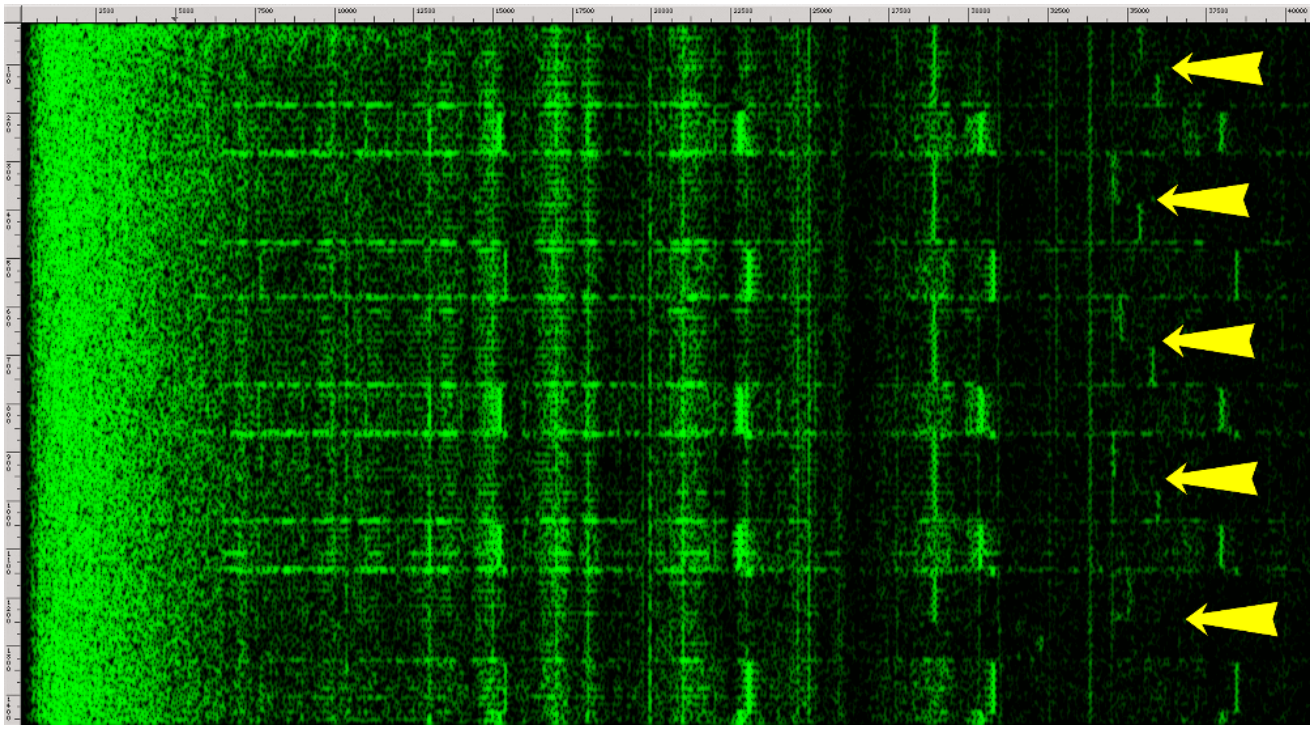


Figure 15: Acoustic signature (1.4 sec, 0–40 kHz) of five GnuPG RSA signatures executed on a Lenovo ThinkPad T61. The recoding was made using the lab-grade setup and the Brüel&Kjær 4190 microphone capsule. The transitions between p and q are marked with yellow arrows.

5 Overview of GnuPG RSA key extraction

Recall that in the RSA cryptosystem [RSA78], the key generation procedure selects two random primes p, q of prescribed size, a (fixed) public exponent e, and a secret exponent d such that $ed = 1 \mod \phi(n)$ where $n = pq$. The public key is $\mathsf{pk} = (n, e)$ and the secret key is $\mathsf{sk} = (d, p, q)$. RSA decryption of a ciphertext c starts by computing $c^d \mod n$. Modern RSA security standards mandate key sizes of at least 2048 bits (i.e., 1024 bit primes p, q) in order to achieve adequate levels of security [BBB+12]. For concreteness, in the following we consider even larger keys, of size 4096 bit (and 2048-bit primes), which should be secure beyond the year 2031 [BBB+12]. We show an attack that can extract whole 4096-bit RSA keys within about one hour using just the acoustic emanations from the target machine.

5.1 GnuPG's modular exponentiation routine

GnuPG's implementation of RSA. GnuPG uses an optimization of RSA that is based on the Chinese Remainder Theorem and called RSA-CRT. This optimization improves the efficiency of RSA decryption by a factor of about 4 by performing two exponentiations using 2048 bit numbers instead of one exponentiation using 4096 bit numbers. In order to compute $m = c^d \mod n$, GnuPG first computes two constants, d_p and d_q, and then computes $m_p = c^{d_p} \mod p$ and $m_p = c^{d_q} \mod q$. Finally, m_p and m_q are combined in order to obtain m. In particular, the ciphertext (potentially chosen by the adversary) is directly passed to the underlying modular exponentiation routine, along with the secret exponent and modulus.

GnuPG's mathematical library. Algebraic operations on large integers (which are much larger than the machine's word size) are implemented using software routines. GnuPG uses an internal mathematical library called MPI, which is based on the GMP library [GMP], to store and perform mathematical operations on large integers. MPI stores each large integer in an array of *limbs*, each consisting of a 32-bit word (on the x86 architecture used in our tests).

We now review the modular exponentiation used in GnuPG's implementation of RSA.[7] GnuPG uses a side-channel protected variant of the square-and-multiply modular exponentiation algorithm, processing the bits of the exponent d from most significant bit to the least significant one. Algorithm 1 is a pseudocode of the modular exponentiation algorithm used in GnuPG. The operation SIZE_IN_LIMBS(x) returns the number of limbs in the number $x = x_t \cdots, x_1$, namely $\lceil t/32 \rceil$. The routine SHIFT_LEFT(x) shifts x by one bit to the left (discarding the most significant bit). The constant KARATSUBA_THRESHOLD, defined to be 16, means that when the ciphertext c contains more than 15 limbs, the Karatsuba multiplication algorithm [KO62] is used; otherwise, the regular schoolbook multiplication algorithm is used.

Understanding this top-level exponentiation routine suffices for the high-level description of our attack. For details about GnuPG's underlying multiplication routines, necessary for understanding the attack's success, see Section 6.

Since GnuPG represents large numbers in arrays of 32 bit limbs, GnuPG optimizes the number of modulo reductions by always checking (at the end of every multiplication and squaring) whether the number of limbs in the partially computed result exceeds the number of limbs in the modulus. If so, a modular reduction operation is performed. If not, reduction will not decrease the limb count, and thus is not performed. This measurement of size in terms of limbs, as opposed to bits, slightly complicates our attack. Note that due to a recently introduced side-channel mitigation technique (see Section 9.1), this code always performs the multiplications, regardless of the bits of d.

Note that the ciphertext c is passed, unmodified, directly to the underlying multiplication routine. Thus, if c contains many limbs that have all their bits set to 1, this special structure will be passed

[7] We describe the implementation of the modular exponentiation (and its underlying mathematical library) introduced in GnuPG v1.4.14 (see Section 9.1).

Algorithm 1 GnuPG's modular exponentiation (see function mpi_powm in mpi/mpi-pow.c).

Input: Three integers c, d and q in binary representation such that $d = d_n \cdots d_1$.
Output: $m = c^d \mod q$.

```
1: procedure MODULAR_EXPONENTIATION(c, d, q)
2:     if SIZE_IN_LIMBS(c) > SIZE_IN_LIMBS(q) then
3:         c ← c mod q
4:     m ← 1
5:     for i ← n downto 1 do
6:         m ← m^2
7:         if SIZE_IN_LIMBS(m) > SIZE_IN_LIMBS(q) then
8:             m ← m mod q
9:         if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then          ▷ defined as 16
10:            t ← MUL_BASECASE(m, c)                  ▷ Compute t ← m · c using Algorithm 3
11:        else
12:            t ← MUL(m, c)                           ▷ Compute t ← m · c using Algorithm 5
13:        if SIZE_IN_LIMBS(t) > SIZE_IN_LIMBS(q) then
14:            t ← t mod q
15:        if d_i = 1 then
16:            m ← t
17:    return m
18: end procedure
```

by the modular exponentiation routine directly to the underlying multiplication routine without any modifications.

5.2 The attack algorithm

Our attack is an adaptive chosen-ciphertext attack, which exposes the secret factor q one bit at a time, from MSB to LSB (similarly to Boneh and Brumley's timing attack [BB05]; see Section 10). For each bit q_i of q, starting from the most significant bit position ($i = 2048$), we assume that key bits $q_{2048} \cdots q_{i+1}$ were correctly recovered, and check the two hypotheses about q_i. Eventually, we learn all of q and thus recover the factorization of n. Note that after recovering the top half the bits of q, it is possible to use Coppersmith's attack [Cop97] (following Rivest and Shamir [RS85]) to recover the remaining bits, or to continue extracting them using the side channel.

The same technique applies to p. However, on many machines we noticed that the second modular exponentiation (modulo q) exhibits a slightly better signal-to-noise ratio, possibly because the target's power circuitry has by then stabilized.

Ciphertext choice for modified GnuPG. Let us first consider a modified version of GnuPG's modular exponentiation routine (Algorithm 1), where the size comparisons done in line 2 are removed and line 3 is always executed.

GnuPG always generates RSA keys such that the most significant bit of q is set, i.e., $q_{2048} = 1$. Assume that we have already recovered the topmost $i-1$ bits of q, and let $g^{i,1}$ be the 2048 bit ciphertext whose topmost $i-1$ bits are the same as those recovered from q, whose i-th bit is 0, and whose remaining (low) bits are 1. Consider the invocation of RSA decryption on $g^{i,1}$. Two cases are possible, depending on q_i.

- $\boldsymbol{q_i = 1}$**.** Then $g^{i,1} \leq q$. The ciphertext $g^{i,1}$ is passed as the variable c to Algorithm 1, in which (with the modification introduced earlier) the modular reduction of c in line 3 returns c (since

$c = g^{i,1} < q$). Thus, the structure of c being a 2048 bit number whose $i-1$ lowest bits are set to 1 is preserved, and is passed as is to the multiplication routines in lines 10 and 12.

- $\boldsymbol{q_i = 0}$. Then $q < g^{i,1}$. Thus, when $g^{i,1}$ is passed as the variable c to Algorithm 1, the modular reduction of c in line 3 changes the value of c. Since c and q share the same topmost $2048 - i$ bits, we have that $g^{i,1} < 2q$, and thus the reduction amounts to computing $c \leftarrow c - q$, which is a random-looking $i-1$-bit number. This is then passed to the multiplication routine in lines 10 and 12.

Thus, depending on q_i, the second operand to the multiplication routine will be either full-size and repetitive or shorter and random-looking. We may hope that the multiplication routine's implementation will behave differently in these two cases, and thus result in key-dependent side-channel leakage. Note that the multiplication is performed 2048 times with that same second operand, which will hopefully amplify the difference and create a distinct leakage pattern that persists for the duration of the exponentiation. As we shall see, there is indeed a difference, which lasts for hundreds of milliseconds and can thus be detected even by very low bandwidth leakage channels such as our acoustic technique. [8]

Ciphertext choice for unmodified GnuPG. Unfortunately, line 2 in Algorithm 1 makes its reduction decision on the basis of the limb count of c. This poses a problem for the above attack, since even if $g^{i,1} \geq q$, both $g^{i,1}$ and q have the same number of limbs (64 limbs each). Thus, the branch in line 2 of Algorithm 1 is *never* taken, so c is never reduced modulo q, and the multiplication routine always gets a long and repetitive second operand.

This can be solved in either of two ways. First, GnuPG's binary format parsing algorithm is willing to allocate space for leading zeros. Thus, one may just ask for a decryption of $g^{i,1}$ with additional limbs of leading zeros. GnuPG will pass on this suboptimal representation to the modular exponentiation routine, causing the branch in line 2 of Algorithm 1 to be *always* taken and the reduction to always take place, allowing us to perform the attack.

While the above observation can be easily remedied by changing the parsing algorithm to not allocate leading zero limbs, there is another way (which is harder to fix) to ensure that the branch in line 2 of Algorithm 1 is always taken. Note that the attacker has access to the public RSA modulus $n = pq$, which is 128 limbs (4096 bits) long. Moreover, by definition it holds that $n = 0 \mod q$. Thus, by requesting the decryption of the 128 limb number $g^{i,1} + n$, the attacker can still ensure that the branch in line 2 of Algorithm 1 will be *always* taken and proceed with the attack.

Key extraction. We assume the use of a routine, DECRYPT_AND_ANALYZE_LEAKAGE_OF_Q(c), which distinguishes the cases $q_i = 0$ from $q_i = 1$. The routine triggers an RSA decryption of the ciphertext c with the unknown secret key (p, q) on the target machine, measures some side channel leakage during decryption, and applies some classifier to recognize the difference between the two possible leakage patterns created by the multiplication routine (as discussed above). We expect the routine to correctly recover q_i (after repeating the aforementioned process several times if necessary) when $c = g^{i,1} + n$ is chosen as above. A simplified pseudocode for the outer loop is given in Algorithm 2 (following Boneh and Brumley [BB05]). Its implementation, complications and robustness concerns are discussed below, in Section 8.

[8] Ironically, the fact that the latest GnuPG implementations use the side-channel mitigation technique of always multiplying the intermediate results by the input only helps our attack, since it doubles the number of multiplications and replaces their random timing with a highly repetitive pattern that can be picked up more easily by our microphones.

Algorithm 2 Top loop of the (simplified) attack on GnuPG's RSA decryption.

Input: An RSA public key $\mathsf{pk} = (n, e)$ such that $n = pq$ where n is an m bit number.
Output: The factorization p, q of n.

procedure SIMPLIFIEDATTACK(pk)
  $g \leftarrow 2^{(m/2)-1}$ ▷ g is a $m/2$ bit number of the form $g = 10\cdots0$
  for $i \leftarrow m/2 - 1$ **downto** 1 **do**
    $g^{i,1} \leftarrow g + 2^{i-1} - 1$ ▷ set all the bits of g starting from $i-1$-th bit to be 1
    $b \leftarrow$ DECRYPT_AND_ANALYZE_LEAKAGE_OF_Q($g^{i,1} + n$) ▷ obtain the i-th bit of q
    $g \leftarrow g + 2^{i-1} \cdot b$ ▷ update g with the newly obtained bit
  $q \leftarrow g$
  $p \leftarrow n/q$
  return (p, q)
end procedure

5.3 Acoustic leakage of the bits of q

In this section we present empirical results about acoustic leakage of the bits of q using our attack. As argued in Section 5.2, we expect that during the entire modular exponentiation operation using the prime q, the acoustic leakage will depends on the value of the single bit being attacked.

Figure 16(a) shows a typical recording of RSA decryption when the value of the attacked bit of q is 0 and Figure 16(b) shows a recording of RSA decryption when the value of the attacked bit of q is 1. Several effects are shown in the figures. Recall that GnuPG first performs modular exponentiation using the secret prime p and then performs another modular exponentiation using the secret prime q. As in figure 15, the transition between p and q is clearly visible in Figures 16(a) and 16(b).

Note, then, that the acoustic signatures of the modular exponentiation using the prime q (the second exponentiation) are quite different in Figures 16(a) and 16(b). This is the effect utilized in order to extract the bits of q. Also, note that within each figure, the acoustic signature of exponentiation modulo q does not change much over time as the modular exponentiation progresses.

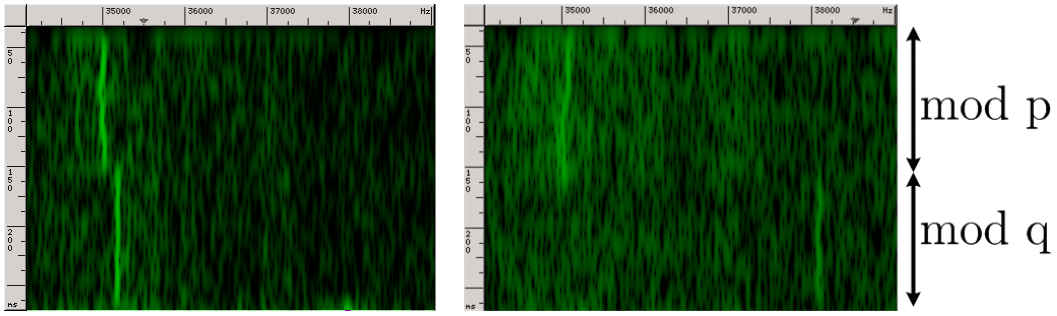
Figure 16(c) was computed from the acoustic signatures of the second modular exponentiation in Figures 16(a) and 16(b). For each signature, we computed the median frequency spectrum (i.e., the median value in each frequency bin over the sliding-window FFT spectra). Again, the differences in the frequency spectra between a zero bit and a one bit are clearly visible and can be used to extract the bits of q. Finally, note that the acoustic signatures of the modular exponentiation using the secret prime p (the first exponentiation) looks the same in Figures 16(a) and 16(b). This is because (in this case) both of the guesses generated to attack q are larger than p, and thus both guesses trigger a reduction modulo p, causing the second operand of the multiplication routine to be random-looking during the entire exponentiation operation modulo p.

Unfortunately, the differences in acoustic leakage between zero and one bits as presented in this section become less prominent as the attack progresses. Thus, in order to extract the entire 2048 bit prime q, additional analysis and improvements to the basic attack algorithm are needed (see Section 8).

5.4 Overall attack performance

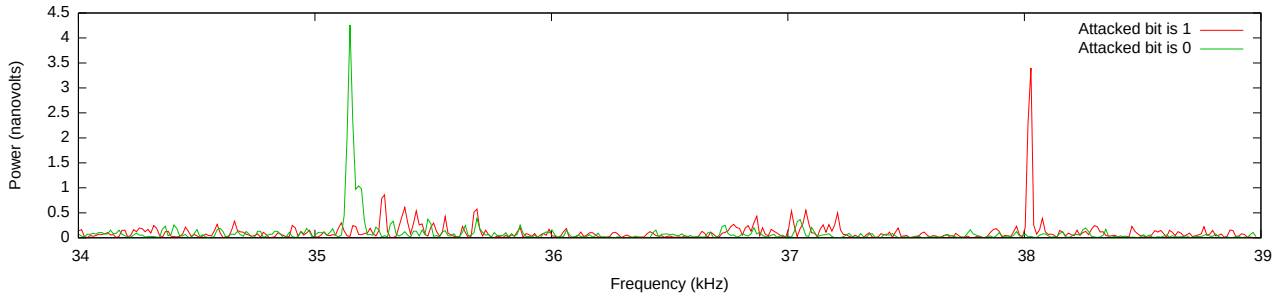
The attack's success, and its running time (due to repeated measurements and backtracking), depend on many physical parameter. This include the machine model and age, the signal acquisition hardware, the microphone positioning, ambient noise, room acoustics, and even ambient temperature (which affects fan activity). The following are some examples of successful attacks.

Ultrasound-frequency attack. Extracting the topmost 1024 bits of q (and thereby, the whole key)



(a) attacked bit is zero

(b) attacked bit is one



(c) Frequency spectra of the second modular exponentiation

Figure 16: Acoustic emanations of RSA decryption for various values of the attacked bit ($q_{2039} = 1$ and $q_{2038} = 0$).

from GnuPG 1.4.14, running on a Lenovo ThinkPad T61 laptop, in a typical office environment, takes approximately 1 hour. Since this laptop's useful signal is at approximately 35 kHz, there is no need to use the lab-grade setup, and we we can use instead a Brüel&Kjær 4190 microphone capsule connected to our portable setup (see Section 2.2 for a description).

Audible-frequency attack. Low frequency sound propagates in the air, and through obstacles, better than high frequency sound. Moreover, larger capsule diaphragms allow better sensitivity but reduce the frequency range.

Indeed, some machines, such as the Lenovo ThinkPad X300 and ThinkPad T23, exhibit useful leakage at lower leakage frequency, 15–22 kHz (i.e., within audible range). This allows us to use the very sensitive Brüel&Kjær 4145 microphone capsule and extract the key from some machines at the range of around 1 meter. By placing the Brüel&Kjær 4145 microphone in a parabolic reflector, the range is increased to 4 meters (see Figure 6 and Section 2.4).

Mobile-phone attack. Lowering the leakage frequency also allows us to use lower quality microphones such as the ones present in smartphones. As described in Section 2.3, we used mobile phones to perform the attack. Due to the lower signal-to-noise ratio and frequency response of the phone's internal microphone, our attack is limited in frequency (about 24 kHz) and in range (about 30 centimeters). However, it is still possible to perform it on certain target computers, simply by placing the phone's microphone near to and directed towards the fan exhaust vent of the target while running the attack (see Figure 4). Unlike previous setups, all that is required from the attacker in order to actually mount the attack is to download a suitable application to the phone, and place it appropriately near the target computer. No special equipment is required to perform the attack.

5.5 Chosen-ciphertext attack vector

The key extraction attack requires the target device to decrypt ciphertexts that are adaptively chosen by the attacker. The OpenPGP file format [CDF+07], and the GnuPG software [Gnu], are used in numerous communication, file transfer and backup applications, many of which indeed allow an attacker to cause decryption of chosen ciphertexts. As one example, we consider an attack via encrypted email.

Enigmail attack. Enigmail [Eni13] is a popular plugin to the Mozilla Thunderbird and SeaMonkey e-mail clients that enables transparent signing and encryption of e-mail messages, using the OpenPGP file format and PGP/MIME encoding [ETLR01]. Enigmail provides an integrated graphical user interface and handles e-mail encoding and user interaction; the actual cryptography is done by calls to an external GnuPG executable. Incoming e-mail messages are decrypted upon the user's request. In addition and by default, Enigmail automatically decrypts incoming e-mail messages, when Thunderbird displays its "new email" popup notification . This decryption happens when the requisite GnuPG passphrase is cached[9] or empty. Thus, an attacker can send a suitably-crafted e-mail message to the victim, containing a chosen ciphertext under PGP/MIME encoding. When this e-mail message is fetched by the target computer, the attacker observes the acoustic emanations during decryption, and learns a bit of the secret key. The attacker then proceeds to adaptively send additional e-mail messages, until all key bits are recovered. If the messages are backdated and/or crafted to look like spam, they may even go unnoticed.

6 Analyzing the code of GnuPG RSA

In this section we analyze how our attack effect the code of GnuPG's multiplication routine and causing the differences presented in Section 5.3. We begin by describing the multiplication algorithms used by GnuPG (Section 6.1) we then proceed (Sections 6.2 and 6.3) to describe the effects of our attack on their internal values.

6.1 GnuPG's multiplication routine

GnuPG's large-integer multiplication routine combines two multiplication algorithms: a basic schoolbook multiplication routine, and a variant of a recursive Karatsuba multiplication algorithm [KO62]. The chosen combination of algorithm is based on the size of the operands, measured in whole limbs (see Algorithm 5). Our attack (usually) utilizes the specific implementation of the Karatsuba multiplication routine in order to make an easily-observable connection between the control flow inside the schoolbook multiplication routine and the current bit of q. This change in the control flow is what allows us to leak the value of q one bit at a time.

We now proceed to describe GnuPG's implementation of schoolbook multiplication and Karatsuba multiplication. For clarity's sake, in the discussions below we slightly simplify the code of GnuPG while preserving its side channel weaknesses.

6.1.1 GnuPG's basic multiplication routine

The side-channel weakness we exploit in the code of GnuPG resides inside the basic multiplication routines of GnuPG. Both of the basic multiplication routines used by GnuPG are almost identical implementations of the naive quadratic schoolbook multiplication algorithm, with optimizations for multiplication by limbs equal to 0 or 1, the pseudocode of which is given in Algorithm 3.

[9] The passphrase caching period is user-configurable. In the latest version (Enigmail 1.6), caching relies on GnuPG's `gpg-agent`, which defaults to 10 minutes. In prior versions (e.g., Enigmail 1.5.2) caching is done internally, by default for 5 minutes.

Algorithm 3 GnuPG's basic multiplication code (see functions mul_n_basecase and mpihelp_mul in mpi/mpih-mul.c).

Input: Two numbers $a = a_k \cdots a_1$ and $b = b_n \cdots b_1$ of size k and n limbs respectively.
Output: $a \cdot b$.

```
 1: procedure MUL_BASECASE(a, b)
 2:     if b_1 ≤ 1 then
 3:         if b_1 = 1 then
 4:             p ← a
 5:         else
 6:             p ← 0
 7:     else
 8:         p ← MUL_BY_SINGLE_LIMB(a, b_1)                    ▷ p ← a · b_1
 9:     for i ← 2 to n do
10:         if b_i ≤ 1 then
11:             if b_i = 1 then                  ▷ (and if b_i = 0 do nothing)
12:                 p ← ADD_WITH_OFFSET(p, a, i)          ▷ p ← p + a · 2^(32·i)
13:         else
14:             p ← MUL_AND_ADD_WITH_OFFSET(p, a, b_i, i)  ▷ p ← p + a · b_i · 2^(32·i)
15:     return p
16: end procedure
```

Here, the function MUL_BY_SINGLE_LIMB(v, u) multiplies a multi-limb number v by a single-limb number u and returns the result. The function ADD_WITH_OFFSET(u, v, i) gets as input two multi-limb numbers u, v, shifts v to the left by i limbs, adds the shifted v to u, and returns the result $u+v\cdot 2^{32\cdot i}$. The function MUL_AND_ADD_WITH_OFFSET(u, v, w, i) gets as input two multi-limb numbers u, v, a single-limb number w and an integer i. It multiplies v by w, shifts the result by i limbs to the left, adds the result to u and returns the result $u + v \cdot w \cdot 2^{32\cdot i}$.

Notice how MUL_BASECASE handles zero limbs of b. In particular, when a zero limb of b is encountered, none of the operations MUL_BY_SINGLE_LIMB, ADD_WITH_OFFSET and MUL_AND_ADD_WITH_OFFSET are performed and the loop in line 8 continues to the next limb of c. This particular optimization is critical for our attack. Specifically, our chosen ciphertext will cause the private key bit q_i to affect the number of zero limbs of b given to MUL_BASECASE, thus the control flow in lines 3 and 11, and thereby the side-channel emanations.

6.1.2 GnuPG's Karatsuba multiplication routine

The basic multiplication routine described above is invoked by both the modular exponentiation routine described in Section 5.1 and by the Karatsuba multiplication routine implementing a variant of the Karatsuba multiplication algorithm [KO62] with some (non-asymptotic) optimizations.

The Karatsuba multiplication algorithm performs a multiplication of two n bit numbers u, v using $\Theta(n^{\log_2 3})$ basic operations. GnuPG's variant, given in Algorithm 4 (with slight modifications to the recursive calls), relies on the identity

$$uv = (2^{2n} + 2^n)u_{\mathsf{H}}v_{\mathsf{H}} + 2^n(u_{\mathsf{H}} - u_{\mathsf{L}})(v_{\mathsf{L}} - v_{\mathsf{H}}) + (2^n + 1)v_{\mathsf{L}}u_{\mathsf{L}} \tag{1}$$

where $u_{\mathsf{H}}, v_{\mathsf{H}}$ are the most significant halves of u and v respectively and $u_{\mathsf{L}}, v_{\mathsf{L}}$ are the least significant halves of u and v respectively. The subroutine MUL_AND_ADD_WITH_OFFSET(u, v, w, i) and constant KARATSUBA_THRESHOLD are as before.

Algorithm 4 GnuPG's Karatsuba multiplication code (see function mul_n in mpi/mpih-mul.c).

Input: Two n limb numbers $a = a_n \cdots a_1$ and $b = b_n \cdots b_1$.
Output: $a \cdot b$.

1: **procedure** KARATSUBA_MUL(a, b)
2: **if** $n <$ KARATSUBA_THRESHOLD **then** ▷ defined as 16
3: **return** MUL_BASECASE(a, b) ▷ multiply using Algorithm 3
4: **if** n is odd **then**
5: $p \leftarrow$ KARATSUBA_MUL$(a_{n-1} \cdots a_1, b_{n-1} \cdots b_1)$ ▷ $p \leftarrow (a_{n-1} \cdots a_1)\,(b_{n-1} \cdots b_1)$
6: $p \leftarrow$ MUL_AND_ADD_WITH_OFFSET$(p, a_{n-1} \cdots a_1, b_n, n)$ ▷ $p \leftarrow p + (a_{n-1} \cdots a_1) \cdot b_n \cdot 2^{32 \cdot n}$
7: $p \leftarrow$ MUL_AND_ADD_WITH_OFFSET(p, b, a_n, n) ▷ $p \leftarrow p + b \cdot a_n \cdot 2^{32 \cdot n}$
8: **else**
9: $h \leftarrow$ KARATSUBA_MUL$(a_n \cdots a_{n/2+1}, b_n \cdots b_{n/2+1})$
10: $t \leftarrow$ KARATSUBA_MUL$(a_n \cdots a_{n/2+1} - a_{n/2} \cdots a_1, b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1})$
11: $l \leftarrow$ KARATSUBA_MUL$(a_{n/2} \cdots a_1, b_{n/2} \cdots b_1)$
12: $p \leftarrow \left(2^{2 \cdot 32 \cdot n} + 2^{32 \cdot n}\right) \cdot h + 2^{32 \cdot n} \cdot t + (2^{32 \cdot n} + 1) \cdot l$
13: **return** p
14: **end procedure**

Notice the subtraction $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$ in line 10 in KARATSUBA_MUL(a, b). Recall that in Section 5 we created a connection between the bits of q and specific values of c. Concretely, for the case where $q_i = 1$ then c is a 2048 bit number such that its first $i - 1$ bits are the same as q, its i-th bit is zero and the rest of its bits are ones. Conversely, for the case where the $q_i = 0$, we have that c consists of $2048 - i$ random looking bits.

The code of GnuPG passes c (with some whole-limb truncations) directly to KARATSUBA_MUL as the second operand b. Thus, this structure of c has the property that result of computing $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$ will have almost all of its limbs equal to zero when the current bit of q is 1 and have all of its limbs be random when the current bit of q is 0. Thus, when the recursion eventuality reaches its base case, namely MUL_BASECASE, it will be the case that if the current bit of q is 1 the values of the second operand b supplied to MUL_BASECASE (in some branches of the recursion) will have almost all of it limbs equal to zero and when the current bit of q is 0 the values of the second operand b supplied to MUL_BASECASE in all branches of the recursion will be almost random. Next, by (indirectly) measuring the number of operations performed by MUL_BASECASE, we shall be able to deduce the number of zero limbs in c and thus whether the correct bit of q is 0 or 1.

The code presented in KARATSUBA_MUL does not handle the multiplication of numbers that have different length (in limbs). These cases are handled by the topmost multiplication routine of GnuPG that is presented (with slight changes to the invocation of the basic multiplication routine) in Algorithm 5. The subroutine ADD_WITH_OFFSET(u, v, i), and constant KARATSUBA_THRESHOLD are as before.

6.2 Attacking the most significant limbs of q

In this section we analyze the effects of our attack on MUL_BASECASE (Algorithm 3). Notice that in this case, the cipher text c used in the main loop of the modular exponentiation routine (Algorithm 1) always contains at least 2017 bits (64 limbs), meaning that MUL is used for multiplication.

Since in this case both operands (c and m) of MUL are typically of the same length, it calls KARATSUBA_MUL only once. Next, since the constant KARATSUBA_THRESHOLD is defined to be 16, KARATSUBA_MUL generates a depth-4 recursion tree where each node has 3 children before using the basic multiplication code (MUL_BASECASE) on 8-limb (256 bit) numbers located on the leaves of the

Algorithm 5 GnuPG's multiplication code (see function mpihelp_mul_karatsuba_case in mpi/mpih-mul.c).

Input: Two numbers $a = a_k \cdots a_1$ and $b = b_n \cdots b_1$ of size k and n limbs respectively.
Output: $a \cdot b$.

```
1:  procedure MUL(a, b)
2:      if n < KARATSUBA_THRESHOLD then                          ▷ defined as 16
3:          return MUL_BASECASE(a, b)                            ▷ multiply using Algorithm 3
4:      p ← 0
5:      i ← 1
6:      while i · n ≤ k do
7:          t ← KARATSUBA_MUL(a_{i·n} ··· a_{(i−1)·n+1}, b)      ▷ multiply n limb numbers using Algorithm 4
8:          p ← ADD_WITH_OFFSET(p, t, (i − 1) · n)               ▷ p ← p + t · 2^{32·(i−1)·n}
9:          i ← i + 1
10:     if i · n > k then
11:         t ← MUL(b, a_k ··· a_{(i−1)·n+1})                    ▷ multiply the remaining limbs of a using a recursive call
12:         p ← ADD_WITH_OFFSET(p, t, (i − 1) · n)               ▷ p ← p + t · 2^{32·(i−1)·n}
13:     return p
14: end procedure
```

tree. Figure 17 shows the structure of this recursion tree; the dashed leaves result from the recursive call made by line 10 of Algorithm 4. In particular the second operand of the recursive call in line 10 is the result of $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$.

Combining this observation with the case analysis of Section 5.2, we see that for each bit i of q one of the following holds.

1. If $q_i = 1$ then the second operand b of MUL mainly consists of limbs having all their bits set to 1. Thus, during the first call to KARATSUBA_MUL the value $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$ in line 10 contains mostly zero limbs, causing the second operand of all the calls to MUL_BASECASE resulting from the recursive call in line 10 (located on the dashed leaves of Figure 17) to contain mostly zero limbs.

2. If $q_i = 0$ then the second operand b of MUL consists of random-looking limbs. Thus, during the first call to KARATSUBA_MUL the value $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$ in line 10 contains mostly very few (if any) zero limbs causing the second operand of *all* the callas to MUL_BASECASE located on *all* the leaves of Figure 17 consists of mostly non-zero limbs.

Confirming this, Figure 18 presents the number of zero limbs in the second operand of MUL_BASECASE during an execution of the modular exponentiation routine with the secret prime q using our carefully chosen cipher texts and five randomly generated 4096 bit keys. The values of the first 100 bits of q are clearly visible (a large number of zero limbs implies that the value of the respective bit is 1).

Next, recall the effect of zero limbs in the second operand on the code of MUL_BASECASE. Notice that the control flow in MUL_BASECASE depends on the number of non-zero limbs present in its second operand. The drastic change in the number of zero limbs in the second operand of MUL_BASECASE is detectable by our side channel measurements. Thus, we are able to leak the bits of q, one bit at a time, by creating the connection between the current bit of q and number of zero limbs in the second operand of MUL_BASECASE using our carefully chosen cipher texts.

Finally, notice that the Karatsuba multiplication algorithm is (indirectly) called during the main loop of the modular exponentiation routine (Algorithm 1) once per every bit of d_q as computed by the

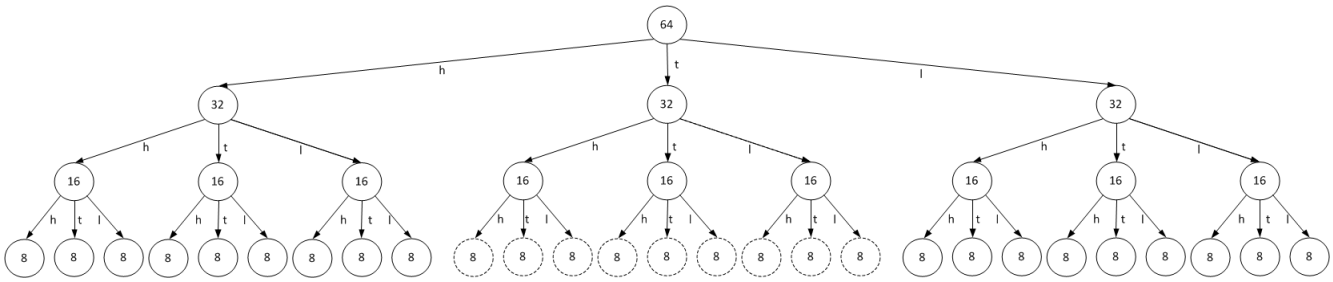


Figure 17: The recursion tree created by Algorithm 4 during the first 32 bits of the attack. The size (in limbs) of the numbers handled by each recursive call is indicated inside each node while the variable holding the value returned by the call is indicated on the edge leading to the node. The leaves of the tree are computed using MUL_BASECASE (Algorithm 3). The leaves influenced by our attack are dashed.

RSA decryption operation. Since d_q is a 2048 bit number we get that the leakage generated by line 11 in MUL_BASECASE is repeated once for every zero limb of b for a total of 7 times during an execution of MUL_BASECASE which is in turn repeated once for every dashed leaf of Figure 17 for a total of 12 times in each of the 2048 multiplications in that loop.

Thus, we get that leakage generated by line 11 in MUL_BASECASE is repeated 172032 times. This repetition is what allows the leakage generated by line 11 in MUL_BASECASE to be detected using only low bandwidth leakage.

6.3 The remaining bits of q

Unfortunately, the analysis of Section 6.2 does not precisely hold for the remaining bits of q. Recall that by our choice of ciphertexts, at the beginning of MODULAR_EXPONENTIATION both $c - n$ and q always agree on some prefix of their bits and this prefix becomes longer as the attack progresses and more bits of q are extracted. Thus, since $c - n < 2q$ the reduction of c modulo q always returns $c - q - n$ for the case of $c - n \geq q$ and $c - n$ otherwise.

In particular, after the first limb of q has been successfully extracted, for the case where $q_i = 0$, the value of c after the modular reduction in line 2 of MODULAR_EXPONENTIATION is shorter than 64 limbs. Since in lines 10 and 12 we have that c is passed as is to the multiplication routine while m remains a 64 limb number, the part of the multiplication routine responsible for handling operands of different sizes is used and, instead of a single call to the Karatsuba multiplication routine (Algorithm 4), might make several recursive calls to itself as well as several calls to Algorithm 4.

None the less, there is still an easily observed connection between secret bits of q and the structure of the value of c passed to multiplication routine by the modular exponentiation routine (Algorithm 1) as follows. For any $1 \leq i \leq 2048$ one of the following two cases holds.

1. If $q_i = 1$ then c is a 2048 bit number such that the first $i - 1$ bits the same as q, the i-th bit is zero and the rest of the bits are ones.

2. If $q_i = 0$ then c consists of $64 - \lfloor i/32 \rfloor$ random looking limbs.

While the analysis in this case is not as precise as in Section 6.2, the number of zero limbs in the second operand of MUL_BASECASE still allows us to extract the bits of q (see Figure 18).

Unfortunately, while still possible, distinguishing the above two cases using side channel leakage is particularly hard for bits in the rage of 1850–1750. This complication requires us to use additional tricks to recover these problematic bits and continue our attack (see Section 8.3 for details).

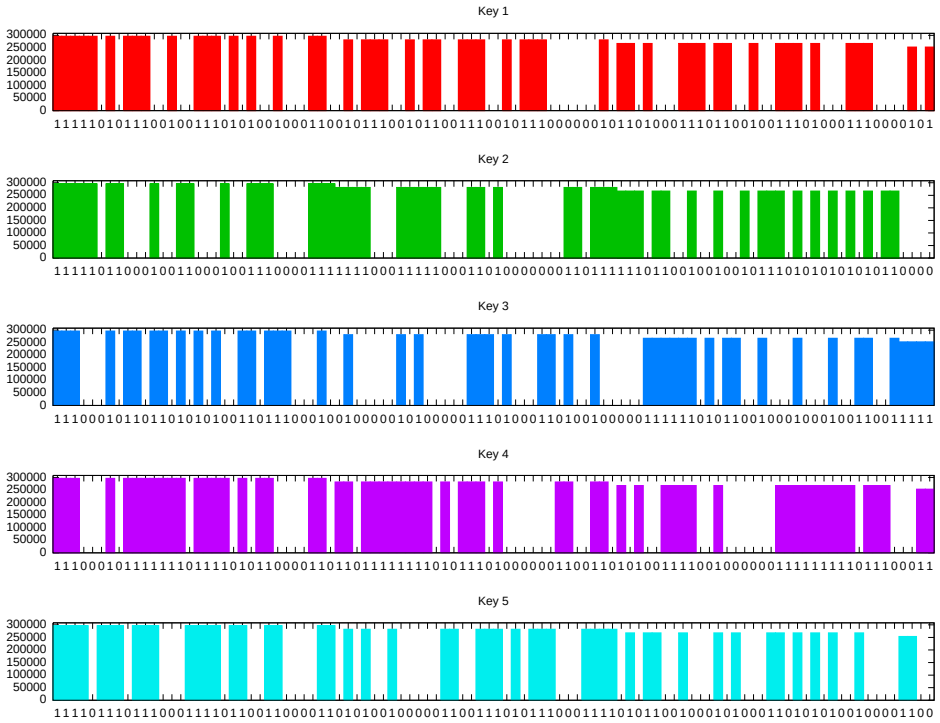


Figure 18: Number of zero limbs in the second operand of MUL_BASECASE(a, b) during an execution of the modular exponentiation routine with the secret prime q using our attack and randomly generated keys. The correct bit values are indicated on the horizontal axis of each key.

7 Analyzing the acoustic leakage of GnuPG

In this section we further analyze the acoustic leakage produced by GnuPG, supporting the code analysis of Section 6. Recall that Section 5 allows us to make a connection between q_i (assuming all the previous bits are known) and the second operand c of MUL (Algorithm 5) as follows.

1. If $q_i = 1$ then c is a 2048 bit number such that the first $i-1$ bits the same as q, the i-th bit is zero and the rest of the bits are ones.

2. If $q_i = 0$ then c consists of $64 - \lfloor i/32 \rfloor$ random-looking limbs.

The value of the second operand c of MUL (Algorithm 5) is affected by our attack in two separate ways. Concretely, our attack effects both the number of limbs in c, as well as the bits structure of c by making c to be random looking or having most of its bits fixed to 1. In this section we examine how both the number of limbs in c, as well as the bits structure of c effect the leakage produced by GnuPG. Unless otherwise indicated, in the remaining of this work, all the measurements were performed using a Lenovo ThinkPad T61, and the Brüel&Kjær 4190 microphone capsule.

7.1 Acoustic leakage of various ciphertext length

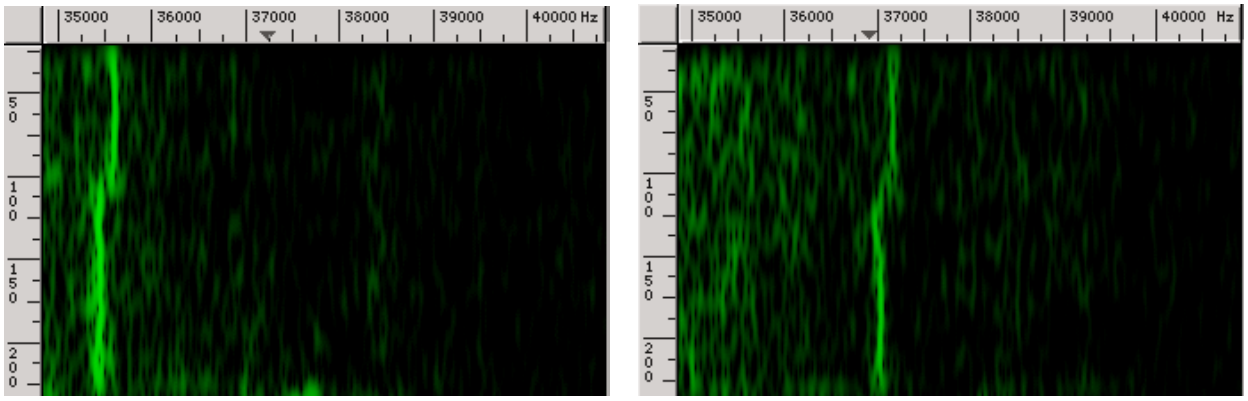
We begin by examining the effects of decrypting random ciphertexts of various length (in limbs). Notice that since the code of GnuPG does not modify the ciphertext till it reaches MUL, the distributions of values of the second operand of MUL in this case, is similar to the distribution created when attacking zero bits of q.

Concretely, consider c_0 to be a randomly generated 63 limb ciphertext and consider c_1 being randomly generated 57 limb ciphertext. Figure 19(a) shows a recording of RSA decryption of c_0 and Figure 19(b) shows a recording of RSA decryption of c_1. As in Figures 16(a) and 16(b), the two modular exoneration operations are clearly visible in Figures 19(a) and 19(b). Next, the two modular exponentiation operations during the decryption of c_0 produce a signal at lower frequency compared to the signal produced by the two modular exposition operations during the decryption of c_1. Thus, this indicates that it is possible for an attacker to learn the number of limbs in the second operand of MUL.

Exploring this effect more systematically, Figure 20 contains acoustic signatures of the second modular exponentiation on randomly generated ciphertexts of various length. Each row in Figure 20, is obtained by generating several random ciphertexts of appropriate length in limbs, decrypting each ciphertext and computing the frequency spectrum of the acoustic leakage of the second modular exponentiation (for the case where the ciphertext has the same length in limbs as q, we generate only ciphertext that are smaller in magnitude that q to avoid being reduced modulo q). Finally, for each length of ciphertext and for each frequency bin, we plot the median value across all ciphertext spectra. As can be seen in figure 20, the shorter the number (in limbs) the higher the frequency of the acoustic leakage. Moreover, notice that the signal strength appears to degrade with the increase of frequency. We conjecture that this is due to the non-linear frequency response of the Brüel&Kjær 4190 microphone capsule at these frequencies.[10] Thus we confirm our hypothesis that a potential attacker might lean the number of limbs in the second operand of MUL by observing the acoustic leakage of the second modular exponentiation.

Finally, notice that for each row i in Figure 20, the values of second operand of MUL have a similar distribution as when attacking the zero bits of q located in limb i. Thus, we expect the acoustic signatures of the second modular exponentiation when attacking zero bits of q located in the i-th limb, to be similar to the signatures obtained in row i in Figure 20.

[10] Recall that the Brüel&Kjær 4190 microphone capsule has a nominal range of up to 20 kHz while e focus on the $30-40$ kHz range.



(a) 63-limb ciphertext (b) 57-limb ciphertext

Figure 19: Typical acoustic emanations (0.2 sec, 35–40 kHz) of RSA decryption for randomly-chosen ciphertexts with different number of limbs.

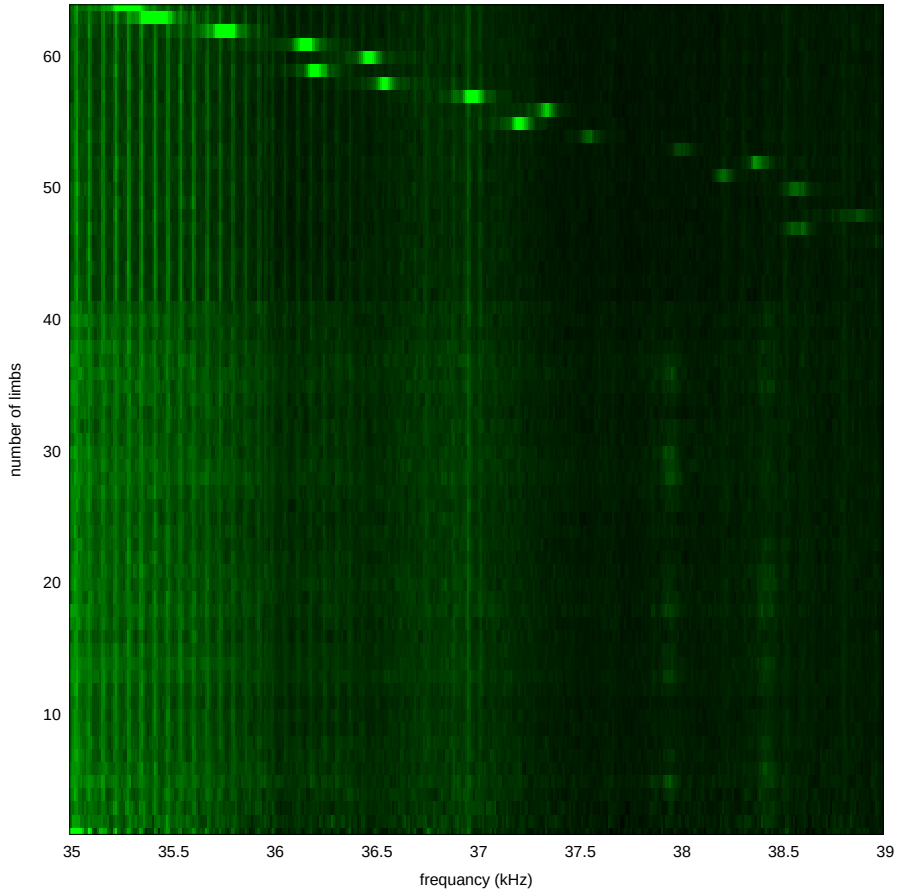
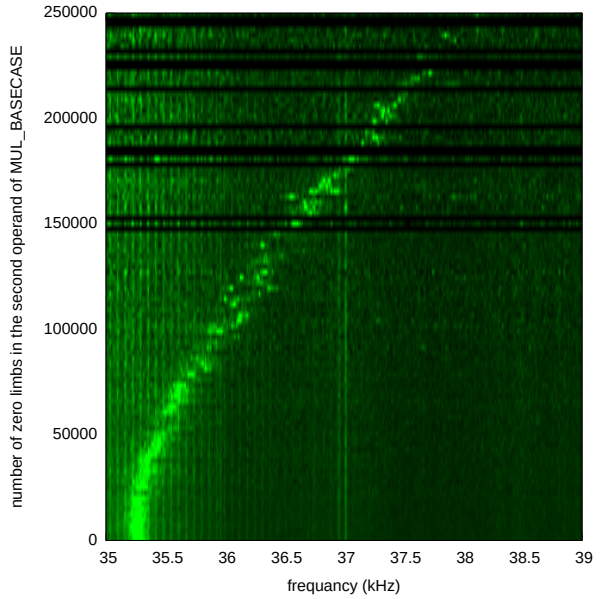
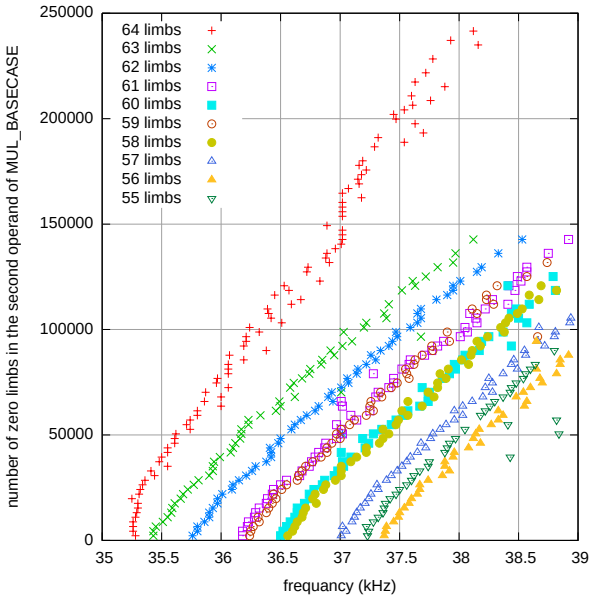


Figure 20: Frequency spectrum of acoustic emanations of the second modular exponentiations during an RSA decryption of ciphertexts of various length. The horizontal axis is frequency and the vertical axis is the number of limbs in the decrypted ciphertext.



(a) ciphertexts of size 64 limbs (black rows correspond to zero-limb counts that did not occur in any of the samples)



(b) frequency of the strongest signal (after filtering) during the second modular exponentiation of ciphertexts of various length

Figure 21: Acoustic emanations of the second modular exponentiations during an RSA decryption of ciphertexts. The horizontal axis is frequency and the vertical axis is the number of zero limbs on the second operand of MUL_BASECASE.

7.2 Acoustic leakage of zero limbs

We now proceed to examine the effects of decrypting ciphertexts of various length (in limbs) that cause the second operand of the basic multiplication routing to have mostly zero limbs. For the case where the second operand of MUL is 64 limbs long, the distribution of the second operand of MUL_BASECASE is similar to the distribution of the second operand of MUL_BASECASE created when attacking one bits of q. We achieve this by generating ciphertexts as follows. For any length of ciphertext $1 \leq i \leq 64$ (in limbs), and for any number of limbs $i \leq j \leq 64$, we generate a random limb ℓ and a random a random ciphertext c of length i. Next, we replace randomly chosen j limbs from c by the fixed limb ℓ. Finally, we repeat the above experiment 16 times and thereby obtaining a dataset of size 33280 ciphertexts.

Figure 21(a) contains acoustic signatures of the second modular exponentiation on ciphertexts generated from the above data set which are 64 limbs long. The signatures arranged by the number of zero limbs in the second operand of MUL_BASECASE. Each row in Figure 21(a) is generated by computing the frequency spectra of the second modular exponentiation during the decryption of all the 64 limb long ciphertexts which cause the number of zero limbs in the second operand of MUL_BASECASE to be as indicated in the row. Next, for each row, and for each frequency bin, we plot the median value across all frequency spectra corresponding to the row. Finally, the rows corresponding to values of zero limbs in the second operand of MUL_BASECASE that are not present in our dataset are filled with black. Each row in Figure 21(b) is obtained by performing the same process as in Figure 21(a) but on various ciphertext lengths (one length per row) and plotting the frequency of the bin containing the maximal power value (ignoring unrelated noise).

As can be seen from Figure 21(b), the frequency of the leakage produced by the second modular exponentiation depends on *both* the length of the second operand of MUL (Algorithm 5) as well as on

the number of zero limbs in the second operand of MUL_BASECASE (Algorithm 3) during the decryption. In particular, in case the attacker knows either the number of limbs in the second operand of MUL (Algorithm 5) or the number of zero limbs in the second operand of MUL_BASECASE (Algorithm 3) during the decryption, he can deduce the other by observing the frequency of the leakage of the second modular exponentiation. We now proceed to show how learning this information allows us to break the security of RSA.

8 Further details on GnuPG RSA key extraction

We now give a more detailed account on our acoustic RSA key extraction attack. While the attack presented in Section 5 indeed recovers the most significant limb of q, A more careful analysis of the acoustic leakage created by our attack is required for the remaining limbs. In this section we systematically analyze the acoustic leakage created by our attack and present the complete attack on GnuPG.

8.1 Extracting the most significant limb of q

Recall the graphs presented in Figures 16. Since in both cases, we are attacking the most significant limb of q, the second operand of MUL is 64 limbs long. Notice that the peaks in Figure 16(c) are exactly as predicted by Figure 21(a). This is since zero bits create very few zero limbs in the second operand of MUL_BASECASE while one bits make almost all limbs in the second operand of MUL_BASECASE equal to zero.

Thus, if the attacker were to have two spectrum *templates* describing the leakage of zero and one bits, he could classify an unknown signal by checking the similarity between it and the templates he has. Concretely, in our case a template is a vector of real numbers describing the signal power at each frequency bin. The classification is based on computing the correlation of the Fourier spectrum of the leakage with the two templates (see Section 8.4 for details).

This approach however, poses a problem since it requires the attacker to obtain an example of a leakage spectrum of zero and one bits. Recall that q is chosen to be a prime such that its most significant bit is always set to one. Moreover, this information is known to an attacker. Thus, obtaining an example of a leakage of a one bit can be done by measuring the leakage resulting from the decryption of $g^{2048,1}$. Obtaining an example of a leakage of a zero bit is more tricky. This is because the attacker does not know in advance the location of the first zero bit in q. However, this problem can be easily avoided. Consider any number ℓ such that $q < \ell < 2q$ (for example $\ell = 2^{2048} - 1$). Notice that the reduction of ℓ modulo q is equivalent to computing $\ell - q$ and will cause the bits of the result to be random thus achieving a similar spectrum as the sound of zero bits of q at the beginning of the attack.

8.2 Adapting to changes in the acoustic signature

Figure 22 presents frequency spectra of the second modular exponentiation during our attack, for 0-valued vs. 1-valued bits. Notice that the peaks in the spectra of 0-valued bits and 1-valued bits get closer as more and more bits are extracted. This is also to be expected given the data in Figures 20 and 21(a). This is because as the attack progresses, the following effects occur. For the case of zero bits, the second operand of MUL becomes shorter (in limbs) and for the case of one bits, the number of zero limbs in the second operand of MUL_BASECASE also decreases. Thus, the two types of "peaks" in the frequency spectrum of the second modular exponentiation move "toward each other"' on the frequency axis forcing us to adapt to these changes during our measurements.

Luckily, as the attack progress, the changes in the spectra corresponding to 0 and 1 bits are small (at most 0.5 kHz) and only significant over time. This means that we can utilize previous observations about the leakage spectrum in order to be able to correctly classify a currently unknown spectrum as

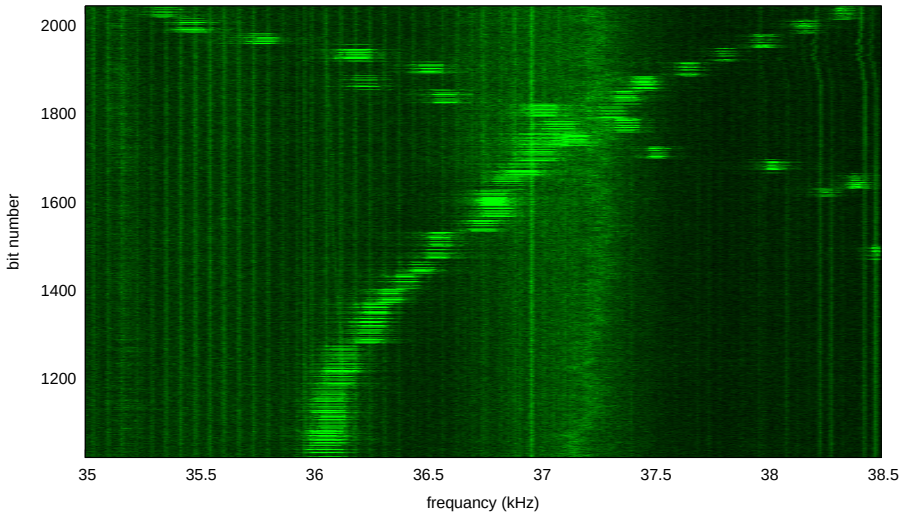


Figure 22: Acoustic measurement frequency spectra of the second modular exponentiation at different times in the attack (overlaid). The curve starting at top left corresponds to zero valued bits. The curve starting at top right corresponds to one values bits.

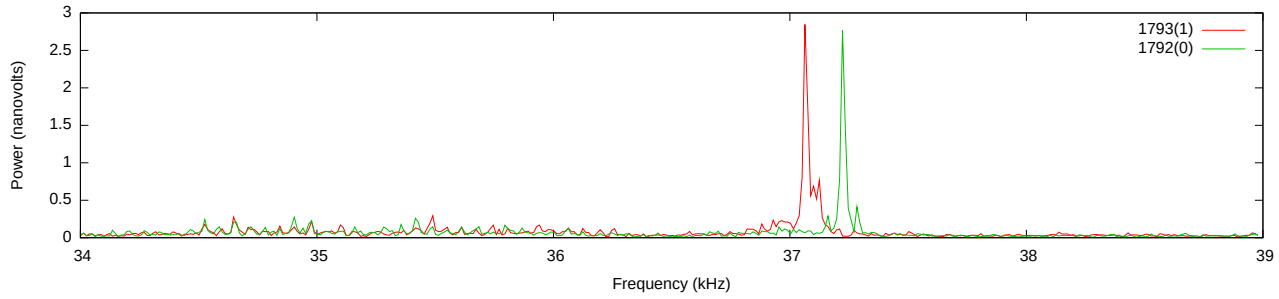
corresponding to a zero bit or a one bit. Later, this spectrum will be used (along with others) to correctly classify the next spectrum. Thus, our attack will have two stages. During the calibration stage, the attacker generates two ciphertexts corresponding to a leakage of zero and one bits of q and obtains multiple samples of their decryption. This allows the attacker to generate a template of the leakage caused by zero bit and a a template of the leakage caused by a one bit.

Next, the attack stage consists of two steps. First, during the classification step, a spectrum of an obtained leakage is classified using the templates as corresponding to zero bit or to a one bit. This step might be repeated a few times until a certain classification is achieved. Second, during the template update step, new templates for zero bits and one bits are generated using the last few spectra (we use 20) including the newly obtained ones. This is done by computing the median-spectrum for zero and one bits by taking the median power value, in each frequency band, across all available spectra for zero and one bits respectively. With these updated templates in hand, the attack proceeds to extract the next bit of q.

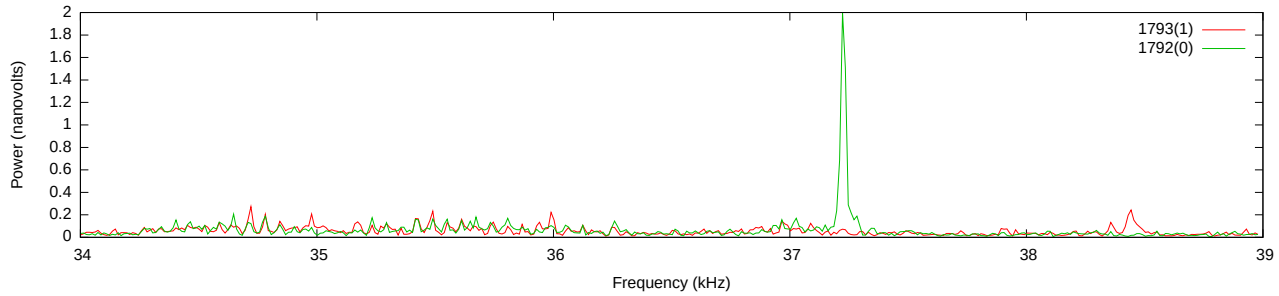
8.3 Overcoming the crossing point

The attack presented in Sections 5.3 and 8.2 is indeed able to recover about 200 key bits. While already causing a breach of security in RSA, it is not clear how this information can be used by a potential attacker. Concretely, as can be seen in Figure 22 the problem is that in bits 1850–1750 where the distance between a signal resulting from a leakage of a zero bit and the signal resulting from a leakage of a one bit is about 200 Hz (see Figure 23(a)). This is less than the frequency change between subsequent bits (around 400 Hz) and thus the bit value cannot be easily distinguished. (This behavior is qualitatively and quantitatively typical, across many target machines.) Recall that when attacking the i-th bit, we assume that all the bits prior to i have been successfully extracted. Thus, we cannot continue our attack without somehow extracting the bits in this problematic range.

Luckily, the specific bit index where this crossing point occurs depends on the specific values of the cipher text used. Concretely, let $g^{i,0}$ be 2048 bit number whose top $i-1$ bits are the same as q, its i-th bit is 1 and all the rest of its bits are 0. Repeating the analysis of Section 5 using $g^{i,0}$ we can obtain



(a) using $g^{i,1}$ as cipher text



(b) using $g^{i,0}$ as cipher text

Figure 23: Acoustic measurement frequency spectra of the second modular exponentiation when attacking the bits in the range of 1850–1750. The bit values are indicated in parenthesis in the legend.

similar connection between q_i and the second operand c of MUL as the connection obtained using $g^{i,1}$. However, as can be seen in Figure 23(b) using $g^{i,0}$ it is now possible to distinguish the bits in the range of 1850–1750 thus allowing our attack to proceed. Notice that the switch from $g^{i,1}$ to $g^{i,0}$ upon reaching the 1850-th bit and backwards upon reaching the 1750-th bit requires us to generate new templates in order to be able to correctly classify the remaining key bits. However, in this case obtaining the bits is quite simple since at this point all the previous key bits are known and can be used in order to generate the required templates. Concretely, for the case of switching between $g^{i,1}$ to $g^{i,0}$ upon reaching the 1850-th bit (the other case is handled similarly), let j be the smallest index of a one bit and let j' be the smallest index of a zero bit such that both j, j' are larger than 1850. Notice that j and j' are known to the attacker at this point since all the bits above the 1850-th bit have been extracted. The template for one bits is obtained by asking several decryptions of $g^{j,0}$. Similarly, the template for zero bits is obtained by asking several decryptions of $g^{j',0}$.

Extracting all bits. Algorithm 6 is a pseudocode of our attack on GnuPG which extracts all bits. As before, we assume the use of a routine, DECRYPT_AND_ANALYZE_LEAKAGE_OF_Q(c), which distinguishes the cases $q_i = 0$ from $q_i = 1$. The routine triggers an RSA decryption of the ciphertext c with the unknown secret key (p, q) on the target machine, measures the acoustic side channel leakage during decryption, and applies a template-based classifier to recognize the difference between the two possible leakage patterns created by MUL and MUL_BASECASE (as discussed above). We also assume that this routine generates new templates for zero and one bits upon switching from $g^{i,1}$ to $g^{i,0}$ and back.

Notice the branch in line 6. This branch is directly designed to deal with the problematic bits describes in Section 8.3. Notice that this switching of ciphertexts only depends on the number of the bit being attacked and does not depend on the secret key, GnuPG version or target machine. Thus, the switching is universal for all 4096 bit keys.

Algorithm 6 Extracting all bits from GnuPG's implementation of 4096-bit RSA-CRT.

Input: A an RSA public key $\mathsf{pk} = (n, e)$ such that $n = pq$ where n is an m bit number.
Output: The factorization p, q of n.

procedure AttackAllBits(pk)
  $g \leftarrow 2^{(m/2)-1}$ ▷ g is a $m/2$ bit number of the form $g = 10\cdots0$
  for $i \leftarrow m/2 - 1$ **downto** 1 **do**
    $g^{i,1} \leftarrow g + 2^{i-1} - 1$ ▷ set all the bits of g starting from $i-1$-th bit to be 1
    $g^{i,0} \leftarrow g + 2^{i-1}$ ▷ set the i-th bit of g to be 1
    if $1750 \leq i \leq 1850$ **then**
      $b \leftarrow \mathsf{decrypt\_and\_analyze\_leakage\_of\_q}(g^{i,0} + n)$ ▷ obtain the i-th bit of q using $g^{i,0}$
    else
      $b \leftarrow \mathsf{decrypt\_and\_analyze\_leakage\_of\_q}(g^{i,1} + n)$ ▷ obtain the i-th bit of q using $g^{i,1}$
    $g \leftarrow g + 2^{i-1} \cdot b$ ▷ update g with the newly obtained bit
  $q \leftarrow g$
  $p \leftarrow n/q$
  return (p, q)
end procedure

8.4 Signal classification

In this section we sketch our algorithm for classifying the acoustic leakage in each iteration of the attack, to deduce the current key bit. As discussed, our classification is based on the frequency spectra of the acoustic leakage during second modular exponentiation. For the classification, we look at the sliding-window Fourier transform, with a spectral resolution (FFT bin width) of about 10 Hz, truncated to the frequency range of interest. Our approach uses two templates T^0, T^1 corresponding to the acoustic leakage of zero and one bits. As described in Section 8.2, the templates are set dynamically and adaptively. When attacking the q_i, the template T^b is the median-spectrum across the last few spectra (we use 20) of the second modular exponentiation, that were obtained in previous iterations of the attack and classified as belonging to bit value b (in case data from previous iterations of the attack is not present, the templates are generated heuristically as described in Section 8.1). Given two templates T^0, T^1 describing the signal power in dB at each frequency bin, the classification of q_i as being zero or one consists of the following stages.

1st stage: obtaining the trace of the second modular exponentiation. We begin by obtaining the acoustic leakage of the second modular exponentiation. This is done by sending the appropriate ciphertext to the target for decryption and recording target's acoustic emanations into *trace*. The trace is recorded or truncated as to contain just the second modular exponentiation. In our experiments this was done by manually-set time offsets; it can also be done by detection of the very distinct signal of exponentiation (see the figures in Section 4).

2nd stage: computing the frequency spectrum. Next, we compute the sliding-window Fourier transform of the trace, yielding a sequence of spectra, and then aggregate these spectra by taking the median value of each bin. (The use of median effectively rejects temporally-local outliers, such as transient spikes.) The spectrum is truncate to the frequency range of interest (determined manually). We denote this aggregate spectrum by s. For example, Figure 24 presents two such spectra, one for a 0-valued bit of q and the other for a 1-valued bit.

3rd stage: peak smoothing. The templates T^0, T^1, and s, contain a lot of noise, often manifested as spikes in the spectrum. We filter out these peaks by convolving T^0, T^1 and s with a Gaussian.

4th stage: normalization. We would like to make the measurements independent of microphone

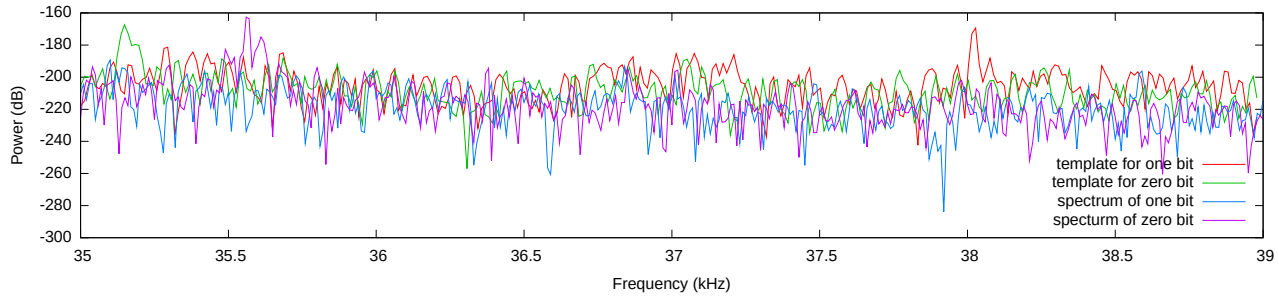


Figure 24: Acoustic frequency spectra of the second modular exponentiation, when attacking a 0-valued bit and a 1-valued bit of q, both in the most significant limb. The corresponding templates T^0 and T^1 are also shown. Notice that the frequency of the peak at 35.2 kHz in the template for zero bits (green) does not match the frequency of the peak at 35.6 kHz in the signal for the zero bit (purple). Similar mismatch (albeit harder to spot) exists between the peak at 38 kHz in the template for one bit (red) and the signal for one bit (blue). Thus, we need to align the template and signal spectra.

frequency response (especially since we push our microphones well beyond their flat-response range), and independent of signal-independent noise. Thus, for every frequency bin i in T^0, T^1 and s, we normalize the power of this bin by subtracting the signal-independent noise $\min(T^0_i, T^1_i)$.

5th stage: computing the distance. We compute the distance d_0 between s and T^0, and likewise the distance d_1 between s and T^1. We would like it to be the case that low values of d_b mean a good match between s and T^b. Later we shall classify the s as corresponding to zero bit in case that $d_0 < d_1$ and corresponding to one bit otherwise. In our case, for each template T^b and for each frequency bin i in T^b we compute the difference between s and T^b in the i-th bin as $(|T^b_i - s_i|)^3$ and then obtain d_b by summing all the differences.

Unfortunately, as showed in Section 8.2 the signal peaks corresponding to the leakage sometimes move on the frequency axis. Thus, we cannot simply compute d_b and must allow for some "shift" of the peaks in s relative to the peaks in T^b. We then take d_b to be the smallest obtained across all allowed shifts. Concretely, let δ be some shift parameter (for GnuPG 1.4.14 we use the equivalent of 400 Hz in FFT bins), and let m be the number of bins in s we compute d_0 and d_1 by

$$d_0 = \min_{-\delta/2 \le i \le \delta/2} \sum_{j=\delta/2}^{m-\delta/2} \left(|T^0_{j+i} - s_j|\right)^3 \text{ and } d_1 = \min_{-\delta/2 \le i \le \delta/2} \sum_{j=\delta/2}^{m-\delta/2} \left(|T^1_{j+i} - s_j|\right)^3 .$$

6th stage: classification. In case that s is sufficiently close to T^0 and sufficiently far from T^1 (meaning that d_0 is sufficiently small and d_1 is sufficiently large) we classify s as corresponding to a zero bit. Otherwise, in case that s is sufficiently close to T^1 and sufficiently far from T^0 (meaning that d_1 is sufficiently small and d_0 is sufficiently large) we classify s as corresponding to a one bit. Otherwise, we repeat the above stages (each time asking for a new decryption of the same ciphertext) until a suitable classification can be achieved.

7th stage: template update. Finally, new templates are generated. For $b = 0$ and $b = 1$, the new template T^b is computed by taking the most recent 20 spectra acquired for bits classified as b (including both the current bit and recent ones), and computing the median value of each frequency bin. These templates will be used for attacking the next bit of q.

8.5 Error detection and correction

In order to extract the q_i using our attack, we rely on knowing bits $q_{2048}, \cdots, q_{i+1}$ (in order to craft the guess $g^{i,1}$ and $g^{i,0}$). In this section, we examine the implications of misclassifying a zero bit as one and vice versa. Recall that the reduction of c modulo q in the modular exponentiation routine (Algorithm 1) is equivalent to computing $c - n$ if $c < q$ and computing $c \leftarrow c - q - n$ otherwise. Two types of mistakes are possible.

Misclassifying a one bit as a zero. If by mistake some bit $q_j = 1$ is misclassified as zero, this means that successive values of both $g^{i,1}$ and $g^{i,0}$ for all $i < j$ will be always smaller then q. Thus, it will be always the case that the value of c after a reduction modulo q will be a 64 limb number whose first $j-1$ bits are the same as q, its j-th bit is zero and the rest of its bits are ones. Notice that this value of c will have the same side channel leakage as if $q_i = 1$. Thus, our attack will always report subsequent bits of q as one regardless their actual value. Luckily, while this prevents any chance of continuing the attack, this situation is easily detectable since q is a random prime that is unlikely to contain long sequences of bits that are one. Thus, when such a sequence (of say 20 bits) is detected, it probably means that a mistake happened somewhere beforehand. Once this mistake is detected, the attacker can just backtrack some number of bits (say 50 bits) and try again (discarding the current templates and using the previous ones).

Misclassifying a zero bit as a one. If by mistake some bit $q_j = 0$ is misclassified as one, this means that successive values of both $g^{i,1}$ and $g^{i,0}$ for all $i < j$ will be always larger than q. Recall that a reduction of c modulo q in this case is equivalent to computing $c \leftarrow c - q - n$. Thus, it will be always the case that c after a reduction modulo q consists of $64 - \lfloor j/32 \rfloor$ random-looking limbs. Notice that this value of c will have the same side channel leakage as if $q_i = 0$. Thus, our attack will always report subsequent bits of q as one regardless their actual value. This situation is again easily detected by the attacker, allowing him to backtrack and retry (discarding the current templates and using the previous ones).

9 Other ciphers and other versions of GnuPG

In this section we investigate the applicability of low bandwidth attacks on other ciphers as well as on different implementations of RSA. Concretely, in Section 9.1 we investigate how a different implementation of the square and multiply algorithm affects our attacks and in Section 9.2 we investigate low bandwidth leakage on GnuPG implementation of ElGamal decryption [ElG85].

9.1 Other versions of GnuPG

Our attacks were first implemented against GnuPG version 1.4.13. However, during the course of this research two newer versions of GnuPG (version 1.4.14 and 1.4.15) where released. Version 1.4.14 introduced changes in modular exponentiation routine, which are reflected in the above analysis. The older modular exponentiation routine, as of version 1.4.13, is given in Algorithm 7. The main difference between the new (Algorithm 1) and old (Algorithm 7) algorithms is a change in the decision on when to perform the additional multiplication by c in lines 11 and 13 of Algorithm 7. While Algorithm 7 performs a multiplication by c when the current bit of d is 1. Algorithm 1 always performs this multiplication (and only performs an additional pointer update in the case where the current bit of d is 1). This is a countermeasure against cache side-channel attacks [YF13] which check the presence of multiplication code in the code cache in order to recover the bits of d.

Recall that our attack utilizes branches deep inside the multiplication code of GnuPG as well as line 3 of Algorithm 7 both of which remain unchanged between different versions, thus our attack is

Algorithm 7 GnuPG's old modular exponentiation (see function mpi_powm in mpi/mpi-pow.c in GnuPG 1.4.13).

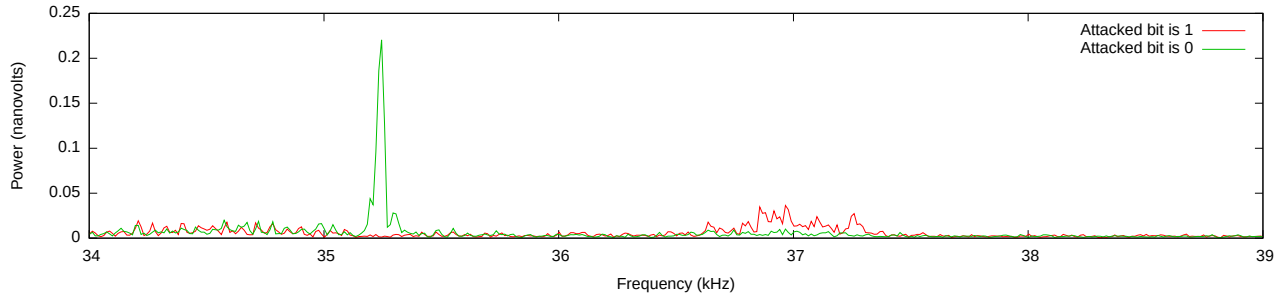
Input: Three integers c, d and q in binary representation such that $d = d_n \cdots d_1$.
Output: $m = c^d \mod q$.

```
1: procedure MODULAR_EXPONENTIATION(c, d, q)          ▷ called powm in GnuPG's code
2:     if SIZE_IN_LIMBS(c) > SIZE_IN_LIMBS(q) then
3:         c ← c mod q
4:     m ← 1
5:     for i ← n downto 1 do
6:         m ← m^2
7:         if SIZE_IN_LIMBS(m) > SIZE_IN_LIMBS(q) then
8:             m ← m mod q
9:         if d_i = 1 then
10:            if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then      ▷ defined as 16
11:                m ← MUL_BASECASE(m, c)          ▷ Compute m ← m · c using Algorithm 3
12:            else
13:                m ← MUL(m, c)                   ▷ Compute m ← m · c using Algorithm 5
14:            if SIZE_IN_LIMBS(m) > SIZE_IN_LIMBS(q) then
15:                m ← m mod q
16:    return m
17: end procedure
```

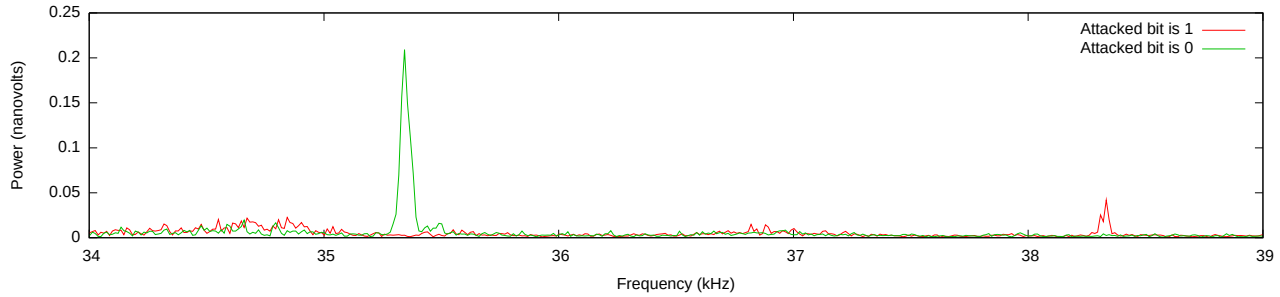
not effected by this change. However, as can be seen in Figure 25, this code change in GnuPG *does* effects the leakage spectra of GnuPG when attacking the bits of q that are set to 1. Notice that while this code modification changes the acoustic (and other) leakage, it is still possible to leak the bits of q using our attacks. Thus, our attacks are in some sense resilient to other code changes and require dedicated countermeasures in order to prevent them. See Section 11 for a discussion about effective and non-effective countermeasures.

9.2 ElGamal key distinguishability

Another popular public key encryption scheme is the ElGamal encryption [ElG85]. In ElGamal encryption, the key generation algorithm consists of generating a large prime p, a generator α of $\mathbb{Z}_p^*$ and a secret exponent $a \in \mathbb{Z}_p^*$. The public key is $\mathsf{pk} = (p, \alpha, \alpha^a)$ and the secret key is $\mathsf{sk} = a$. Encryption of a message m consists of choosing a random k and computing $\gamma = \alpha^k \mod p$ and $\delta = m \cdot (\alpha^a)^k \mod p$, the resulting ciphertext is $c = (\gamma, \delta)$. Decryption of the ciphertext $c = (\gamma, \delta)$ involves computing $h = \gamma^{-a} \mod p$ and $m = h \cdot \delta \mod p$. Figure 26 presents an FFT spectrogram of acoustic emanations recorded during execution of four ElGamal decryptions using the same message and randomly generated 3072 bit keys. Similarly to RSA, the four decryptions are clearly visible. Moreover, while the leakage is not a prominent as in the case of RSA, the four different keys can be easily distinguished.



(a) GnuPG version 1.4.13



(b) GnuPG version 1.4.14

Figure 25: Acoustic measurement frequency spectra of the second modular exponentiation of during our attack using different versions of GnuPG.

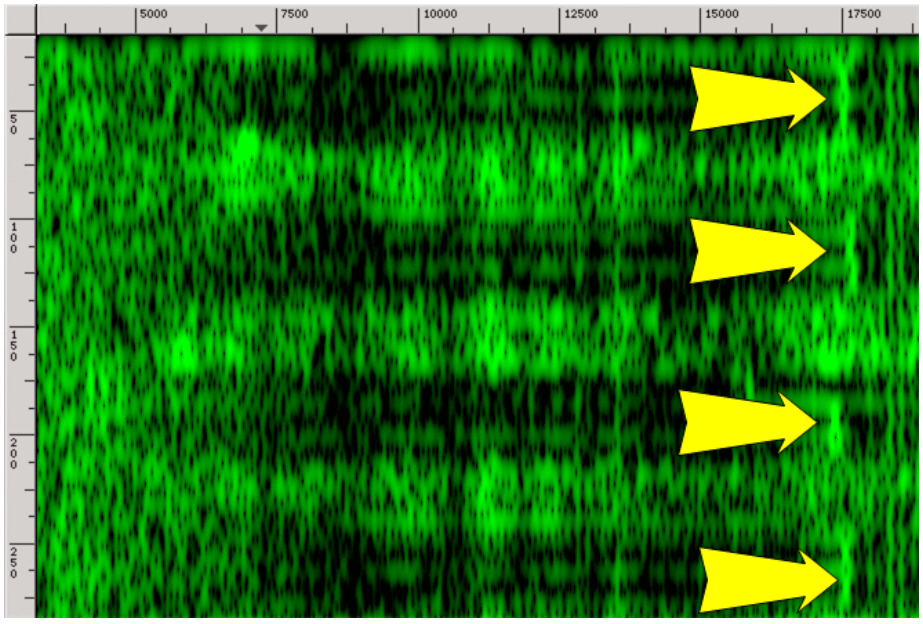


Figure 26: Acoustic signature (0.25 sec 0 − 17.5 kHz) of four ElGamal decryptions using the same message and randomly generated 3072 bit keys.

10 A comparison with timing attacks

In this section we compare our techniques with the timing attacks of [BB05]. Notice that, while our attack and the attacks of [BB05] are chosen-ciphertext-attacks that leak the secret one bit at a time, the effect on the RSA decryption code, as well as the underlying principals of the two attacks are completely different. Concretely, the work of [BB05] utilize OpenSSL's [Ope] use of Montgomery reduction [Mon85] and the OpenSSL's choice of using regular multiplication vs. Karatsuba multiplication. In our case, both of these effects are not relevant. First, GnuPG does not use the Montgomery form to represent its internal values; it uses regular schoolbook long division in order to perform the modular reduction operation. Second, GnuPG always uses Karatsuba multiplication (for values longer than KARATSUBA_THRESHOLD limbs, i.e. 512 bits). Current standards of security call for RSA key size of at least 2048–4096 bits [BBB+12] meaning Karatsuba multiplication will be always used in our case. Thus, we cannot use any of the effects used by [BB05] in our attacks.

In our case, instead of relying on the implementation of the modular reduction and on a high level choice between multiplication algorithms, we are forced to dig deeply into the multiplication code of GnuPG and heavily exploit its internal structure (See Sections 5 and 6 for details). Finally, notice that while it is relatively easy to defend against the timing attack of [BB05] by adding delay cycles at the end of the RSA decryption operation, however, this will not be effective in the case of our attacks. Instead, in order to protect against our attack, deeper changes in the code of GnuPG are required. See Section 11 for details about effective and ineffective countermeasures.

We now examine the possibility of extracting the secret key using our attacks under the assumption that the attacker somehow managed to obtain an accurate measurement of RSA decryption time. This is the best possible scenario for a potential attacker since he gets access to a very precise measurement of the execution time of an RSA decryption operation. To do this, we instrumented the code of GnuPG to measure execution time by using the Windows API call QueryPerformanceCounter on the target machine.[11] The QueryPerformanceCounter call relies on the high performance counter that has a typical update frequency of several megahertz.

Figure 27 presents a graph of the execution times of an RSA decryption operation using our attack on the first 110 bits of q measured on a Lenovo ThinkPad T61. Recall that when attacking the i-th bit, we assume that all the bits prior to i have been successfully extracted. We repeat the attack of each bit 20 times and take the median of the time measured (to reduce noise, and in particular the influence of abnormally long decryption due to bursts of background activity). The values of the first 110 key bits are clearly visible where zero bits require more execution time compared to one bits.

[11] Anther approach is to use the rdtsc instruction. However while working correctly in single core machines, the rdtsc instruction is problematic on multi-core x86 machines since the instruction counters are not necessarily synchronized between cores thus introducing noise into the measurement.

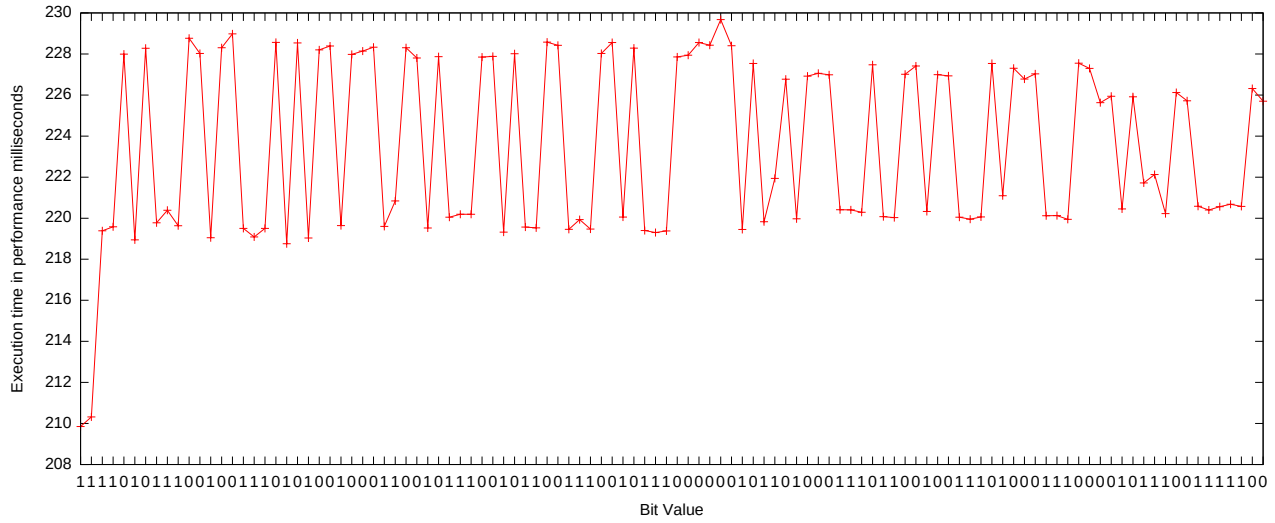


Figure 27: Execution time of RSA decryptions using our attack and a randomly-generated key measured on a Lenovo ThinkPad T61 using the windows performance counter. Each point in the graph is a median of 20 RSA decryptions times while attacking the same bit of q.

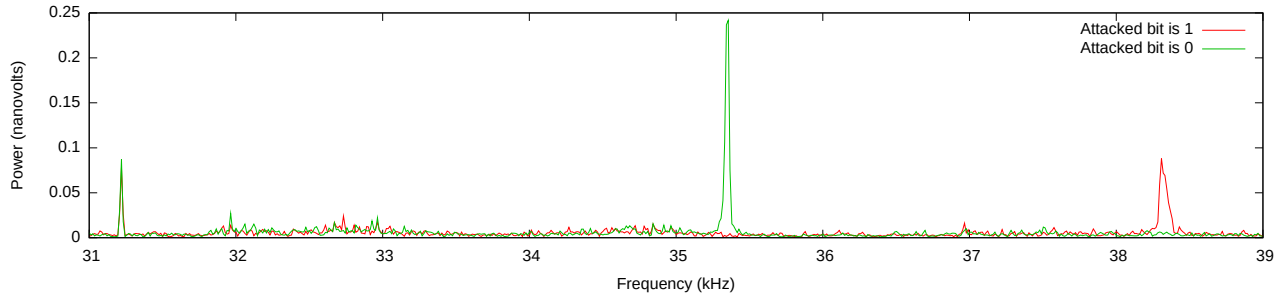
11 Mitigation

Acoustic shielding. Proper acoustic shielding, using acoustic absorbers and sound-proof enclosures, would attenuate the signals, thereby reducing the attacker's signal-to-noise ratio and thus making the attack's cost or time. However, this raises costs in design and in air circulation for cooling. In particular, laptop cooling fan vent holes are typically a prolific source of emanations, and cannot be easily blocked. We also observe that the external "power brick" power supplies of some laptops also seem to exhibit computation-dependent acoustic emanation, and thus likewise require engineering attention and mitigation.

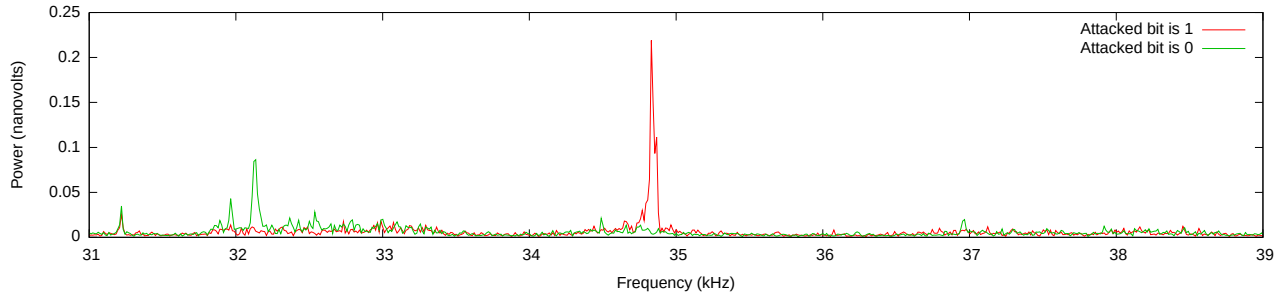
Noisy environment. One may expect that placing the machine in a noisy machine will foil the attack. However, the energy of noise generated in a typical noisy environment (such as outdoors or a noisy room) is typically concentrated at low frequencies, below 10 kHz. Since the acoustic leakage is usually present well above this rage, such noises can be easily filtered out during the data acquisition process using a suitable high-pass filter (as we did in our experiments). Also, the signal would be observe during any pauses in ambient noise (e.g., music). Note also that the attacker may, to some extent, spectrally shift the acoustic signal to a convenient notch in the noise profile, by inducing other load on the machine (see below). Thus, a carefully-designed acoustic noise generator would be required for masking the leakage.

Parallel software load. A natural candidate countermeasure is to induce key-independent load on the CPU, in hope that the other computation performed in parallel will somehow mask the leakage of the decryption operation. Figure 28 demonstrates the difference in the frequency spectra of the the acoustic signature of the second modular exponentiation during our attack resulting from applying a background load comprised of an infinite loop of ADD instructions being performed in parallel. As can be seen from Figure 28, background load on the the CPU core affects the leakage frequency by moving it from the 35–38 kHz range to the range of 32–35 kHz. In fact, this so called "countermeasure" actually might *help* the attacker since the lower the leakage frequency is the more sensitive microphone capsule can be used in order to perform the attack (see Section 5.4).

Cipher text randomization. One countermeasure that *is* effective in stopping our attack is RSA



(a) without background load



(b) with background load

Figure 28: Acoustic measurement frequency spectra of the second modular exponentiation with and without constant load.

ciphertext randomization. Given a cipher text c, instead of decrypting c immediately one can generate a 4096 bit random value r, compute r^e and then decrypt $r^e \cdot c$ and multiply the result by r^{-1}. Since $ed = 1 \mod \phi(n)$ we have that

$$(r^e \cdot c)^d \cdot r^{-1} \mod n = r^{ed} \cdot r^{-1} \cdot c^d \mod n = r \cdot r^{-1} \cdot c^d \mod n = m.$$

In this case, the value sent to the modular exponentiation routine is completely random, thus preventing our chosen cipher text attack. This countermeasure has a nontrivial cost, since an additional modular exponentiation is required (albeit with the exponent e, typically small). Also, it does not affect key distinguishability, which is independent of the ciphertext.

Modulus randomization. Another standard side-channel countermeasure, which may help against both key distinguishing and key extraction, is to randomize the modulus during each exponentiation. To compute $m_q = c^{d_q} \mod q$, first draw a random medium-sized positive integer t and compute $m'_q = c^{d_q} \mod tq$, and then reduce: $m_q = m'_q \mod q$. Better yet, the modulus can be changed to various multiples of q during the exponentiation loop.

Cipher text normalization. Recall that our attack requires chosen ciphertexts that are a slightly larger than q (but have the same limb count as q) to undergo reduction modulo q. However, in this case, GnuPG's modular exponentiation routine will not reduce this ciphertext modulo q. We force this reduction to take place (without changing the result) by padding the ciphertext with leading zeros or adding n to it. This observation yields an immediate countermeasure to our attack. Before decrypting a ciphertext c, one can strip c of any leading zeros and then compute $c' = c \mod n$. In this case, since the ciphertext given to the modular exponentiation routine will have the same limb count as q, the branch in line 2 of Algorithm 1 will be never taken, thereby making it impossible to use the modular reduction in line 3 of Algorithm 1 in order to create a leakage-observable connection between the bits of q and the

bits of the second operand of the multiplication routine. This too foils our key recovery attack (but not key distinguishing).

Acknowledgments

Lev Pachmanov wrote much of the software setup used in our experiments, including custom signal acquisition programs. Avi Shtibel, Ezra Shaked and Oded Smikt assisted in constructing and configuring the experimental setup. Assa Naveh assisted in various experiments, and offered valuable suggestions. Sharon Kessler provided copious editorial advice. We are indebted to Pankaj Rohatgi for inspiring the origin of this research, and to Nir Yaniv for use of the Nir Space Station recording studio and for valuable advice on audio recording. National Instruments Israel generously donated a National Instruments PCI-6052E DAQ card and a MyDAQ device. We thank numerous volunteers for access to test-target machines.

This work was sponsored by the Check Point Institute for Information Security; by the Israeli Ministry of Science and Technology; by the Israeli Centers of Research Excellence I-CORE program (center 4/11); and by NATO's Public Diplomacy Division in the Framework of "Science for Peace".

A Power analysis

The low-bandwidth attacks described in this paper are not limited to the acoustic channel and can be also carried out using various versions of power analysis.

Classical power analysis setup. In this attack scenario we monitor leakage present on the power supply port of the target machines. This is done by putting a voltage divider between the power adapter and target laptop such that the fluctuations in laptop's power consumption could be measured. Concretely we place a $0.5\,\Omega$ resistor in series with the laptop's power supply. Since most laptops use 16 to 20 volts and consume current about 2 to 5 amperes, this means that the voltage on the resistor is between 1 and 2 volts. We measure the voltage on the resistor (and thus the current consumed by the laptop) at a rate of (up to) several hundred kHz.

Non cryptographic leakage. Figure 29 is an analog of Figure 7 showing a recording of the Evo N200 target executing (partially unrolled) loops of various CPU instructions (see Section 3.1 for details), using the power analysis setup and measured using the National Instruments 6356 DAQ from the lab-grade setup. The acoustic and power channel often convey similar information at these frequencies (notice the resemblance between Figure 7 and Figure 29). Typically, the power channel presents higher signal-to-noise ratio, but the reverse also occurs: we have seen cases where the acoustic channel conveyed cryptanalytically-useful signals not discernible in the power channel (see Figure 30).

RSA key distinguishability and extraction. Using power analysis, it is possible to not only distinguish between various CPU operations but also between different RSA keys (See Figure 31). While usually the signal-to-noise ratio on this channel is much higher compared to the acoustic side channel there are exceptions where this is not the case and *no* leakage is present on the power analysis channel leaving only the acoustic channel. Figure 30 is an analog of Figure 14 using the power analysis setup. Notice that no RSA leakage is present in this case. Finally, our attack is not limited to the acoustic channel and can also be carried out using other side-channels such as power analysis. Moreover, the improved signal to noise ratio allows our attack to run much quicker since fewer repetitions are needed to distinguish between bit values. Figure 32 is an analog of Figure 16(c) using power analysis.

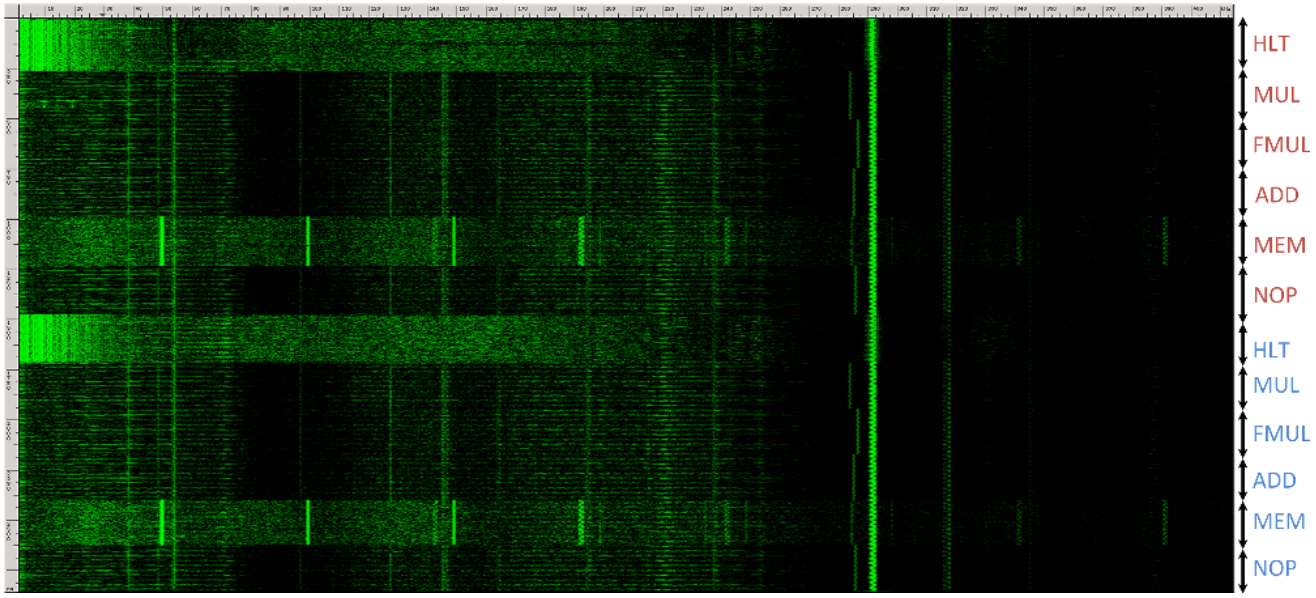


Figure 29: Power measurement FFT spectrogram of a recording (2.8 sec, 400 kHz) of different CPU operations.

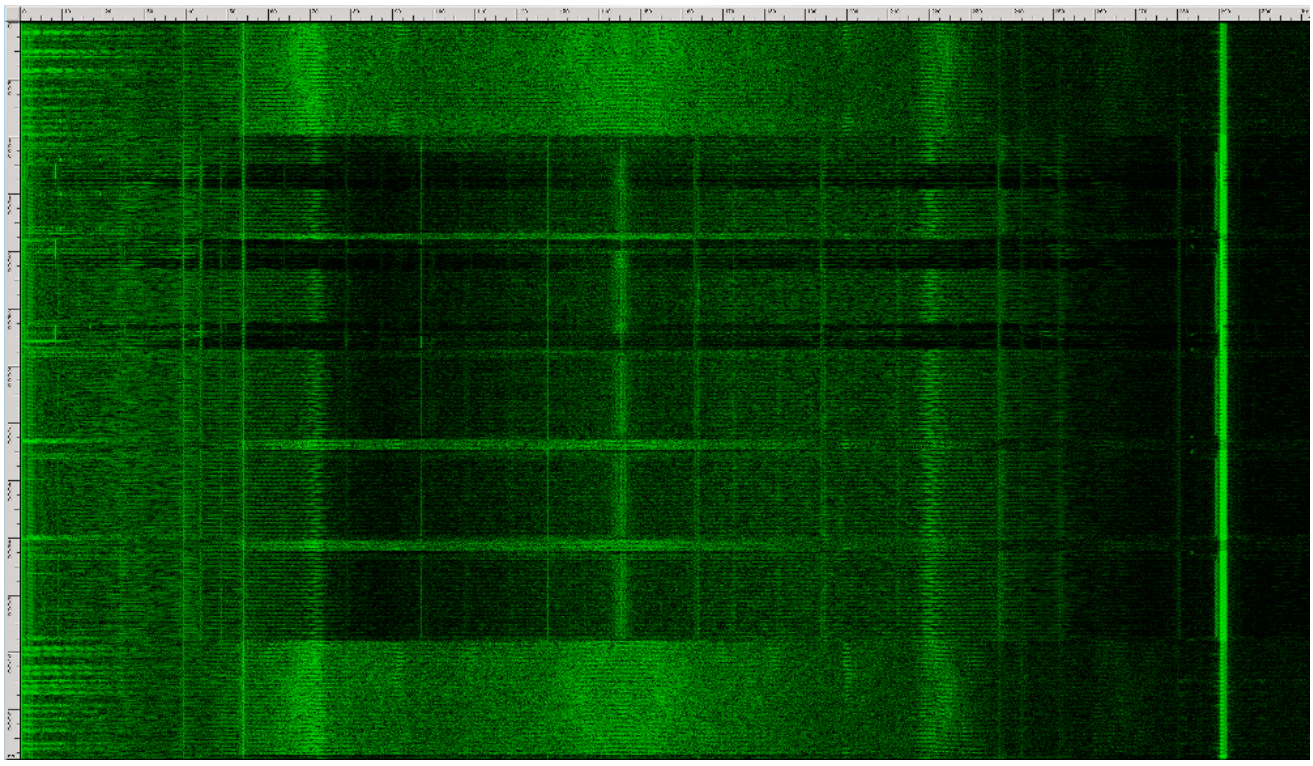


Figure 30: Power measurement FFT spectrogram (6 sec, 0–300 kHz) of a recording of five GnuPG RSA signatures executed on Evo N200 using power analysis.

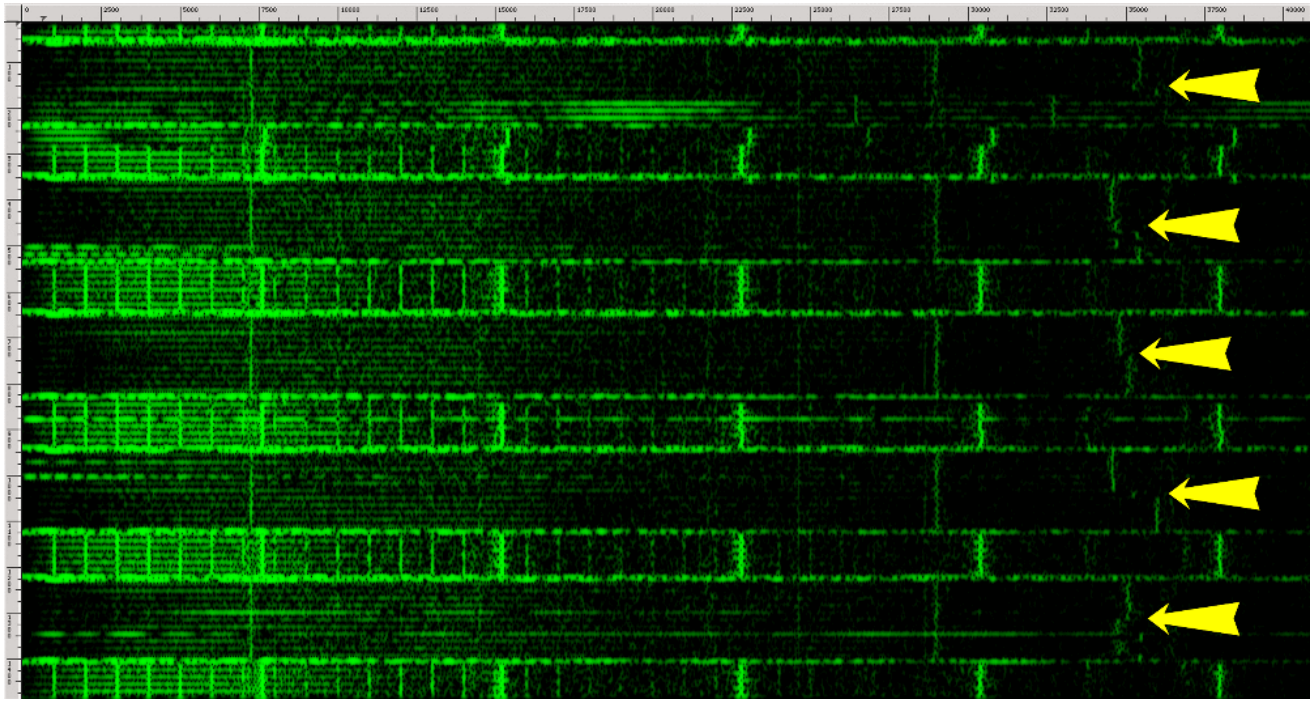


Figure 31: Power measurement FFT spectrogram of a recording (1.4 sec, 0–40 kHz) of five GnuPG RSA signatures executed on a Lenovo ThinkPad T61 using power analysis. The transitions between p and q are marked with yellow arrows.

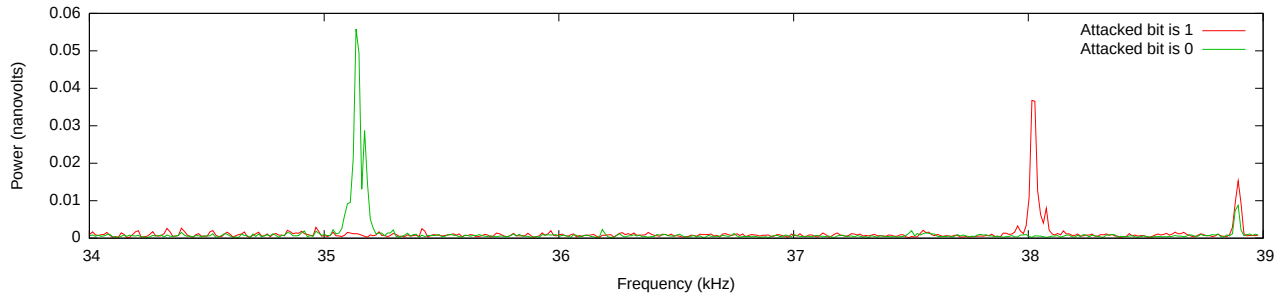


Figure 32: Power measurement frequency spectra of the second modular exponentiation.

B Chassis potential analysis

Another side channel is the electric potential of the laptop's chassis, relative to the room's ground. We observe that when a laptop computer is grounded (by common connections such as the AC power supply adapter or Ethernet cable), the voltage between room ground and the laptop's chassis often carries similar signals to the one we observed acoustically, and moreover with a better signal-to-noise ratio. Most of the experimental results presented in this paper, including the RSA key extraction results, can be conducted using chassis potential measurements.

A voltage probes may be placed in contact with the laptop chassis directly, or indirectly by means such as the following.

B.1 Far-end-of-cable attack

The laptop chassis is electrically connected to the shield of many IO ports such as USB, Ethernet, VGA, DisplayPort, HDMI, etc. When a plug is inserted, the port's shield typically makes contact with the plug shield's, which in turn is connected to a conductive cable shield running the length of the cable. Thus, one can measure the chassis potential from the *far* side of cables connected to the aforementioned ports.

For example, we demonstrated extracting a 4096-bit RSA key from a Lenovo ThinkPad T61 by observing the change in its chassis potential from the far side of a 10 meters long Ethernet cable, while it was connected (via an unshielded Ethernet switch) to the network. Notably, our only point of contact with the target machine was the shield on far side of the Ethernet cable (see Figure 33). This attack does not care about the data transmitted over the VGA or Ethernet cables, or whether the port is even enabled.[12]

While many users are careful about connecting suspicious devices (such as USB devices) to the physical ports of their machines, connecting VGA cables to laptops is often done fearlessly. Likewise, users often connect Ethernet cables to their laptop computers, when an adequate firewall is configured or when the environment appear trustworthy. However, a simple voltage measurement device, perhaps located in the cabinet or server room to which the cable leads, could be capturing the leakage. This is hard to check, since Ethernet wiring and projectors' VGA cables are often hidden out of sight and cannot be easily tracked by the user.

B.2 Magic-touch attack

On many laptops (e.g., most Lenovo ThinkPad models), the chassis potential can be easily reached by a human hand, through metal connectors and conductive coating on metal surfaces. Thus, an attacker can measure the chassis potential by merely *touching* the laptop chassis with his hand.[13] Surreptitiously, the attacker can simultaneously measure his own body potential relative to the room's ground potential, e.g., by having a concealed differential probe touching both his body and some nearby conductive grounded surface in the room. Perhaps surprisingly, even this circuitous measurement offers sufficient signal-to-noise ratio for the key extraction attack.

[12] Oren and Shamir [OS06] made the related observation that on USB ports, the 5V power line exhibits leakage even when the port is disabled.

[13] The attack is especially effective in hot weather, since sweaty fingers offer a lower electric resistance.

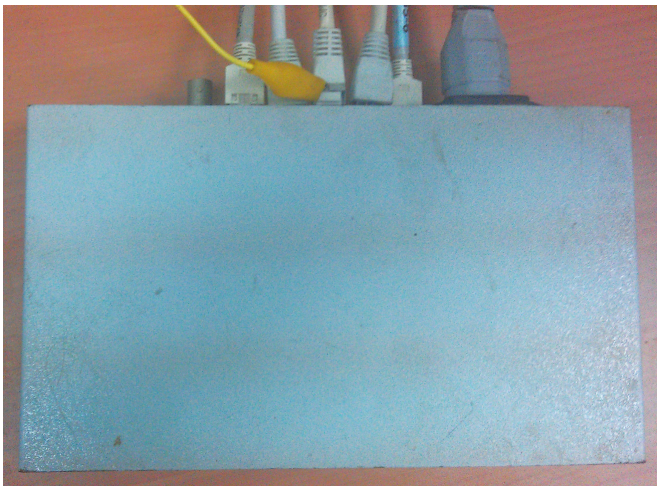


Figure 33: Chassis measurement from the far side of the Ethernet cable, while connected to an Ethernet switch. The potential is measured via the yellow clip connected to the shield of a Ethernet plug.

C The lab-grade setup

In this section we provide further details on our lab-grade setup, introduced in Section 2.1. Since the signals we measure are both weak in power and have relatively high frequency. the measurement setup must be carefully optimized to balance sensitivity and frequency range. In many cases, we use equipment well beyond its designed operating ranges where no official specifications are available. In these cases, we relied on estimations and empirical observations.

Microphone capsules. The most crucial part of the measurement setup, and the only one that must be placed in proximity to the target, is the transducer which converts acoustic signals (changes in air pressure) to electric signals (voltage). Our lab-grade setup uses various condenser microphones, all manufactured by Brüel&Kjær. In these microphones, the transducer is embedded in a *microphone capsule*. The capsule contains a conductive diaphragm, held under tension, in parallel to a firm conductive backplate, at a distance of 15–30 μM. Air pressure causes the diaphragm to vibrate, thereby changing the electric capacitance between the diaphragm and backplate. The change in capacitance is detected by placing a DC polarization voltage (200 V) between the diaphragm and backplate, and observing its AC fluctuations. The capsule is assembled on a *preamplifier* (discussed below).

After testing a variety of Brüel&Kjær microphone capsules, we identified 3 models that dominate the choice (i.e., no other tested model simultaneously provides better frequency response and signal-to-noise ratio): Brüel&Kjær 4939, 4190 and 4145 (see Figures 2 and 34). Details follow.

For very high frequencies, up to 350 kHz, we use a Brüel&Kjær 4939, 1/4" microphone capsule, connected using a UA0035 adapter to a Brüel&Kjær 2669 preamplifier. The Brüel&Kjær 4939 capsule has a sensitivity of 4 mV/Pa and has flat frequency response up to 100 kHz. However, while not officially supported by Brüel&Kjær, the 4939 is able to detect signals from target computers at frequencies as high as 350 kHz (with significant loss in sensitivity).

For lower frequencies, up to 40 kHz, we use a Brüel&Kjær 4190, 1/2" microphone capsule again connected to a Brüel&Kjær 2669 preamplifier. The 4190 capsule has a sensitivity of 50 mV/Pa and a flat frequency response up to 20 kHz. However, the 4190 is able to pick up some signals at frequencies up 40 kHz (with significant loss in sensitivity)[14].

[14]Brüel&Kjær also offers a 4191 microphone capsule that has a flat frequency response up to 40 kHz. However, while

Brüel&Kjær model	Frequency range		Diameter	Sensitivity	SNR
	Effective	Nominal			
4939	0–350 kHz	0–100 kHz	1/4”	4 mV/Pa	51.8 dB
4190	0–40 kHz	0–20 kHz	1/2”	50 mV/Pa	75.1 dB
4145	0–21 kHz	0–18 kHz	1”	50 mV/Pa	83.2 dB

Figure 34: Comparison of microphone capsules used. We specify the nominal range of flat frequency response (as given by Brüel&Kjær), and the larger range of frequencies where the capsule proved effective in our experiments. The signal to noise ratio (SNR) was estimated using the Brüel&Kjær specifications of the relevant capsule connected to a Brüel&Kjær 2669 preamplifier relative to a 1 Pa signal at 10 kHz.

Finally, for even lower frequencies, up to 20 kHz we use a Brüel&Kjær 4145, 1” microphone capsule connected to a 2669 preamplifier using a DB0375 adapter. The 4145 capsule has a sensitivity of 50 mV/Pa and a flat frequency response up to 18 kHz. However, the 4145 is able to pick up some signals at frequencies up to 21 kHz (with significant loss in sensitivity). While both the 4190 and 4145 capsules have a sensitivity of 50 mV/Pa, the 4145 has an internal noise that is lower than the 4190. Thus, the 4145 has a better signal to noise ratio compared to the 4190 at the expanse of the frequency range.

Microphone preamplifier. The aforementioned Brüel&Kjær capsules require a preamplifier, which provides impedance matching (using a unity-gain low-noise high-input-impedance operational amplifier), and also feeds in and filters out the DC polarization voltage to the capsule. We found the Brüel&Kjær 2669 preamplifier to work best with the above capsules.[15]

Microphone power supply. The preamplifier, in turn, requires a suitable low-ripple power supply, providing 28 V power and a 200 V polarization voltage. We use the dedicated Brüel&Kjær 2804 microphone power supply.

Amplification. The signal produced by the capsule, through the unity-gain preamplifier, is very weak (on the order of mV or less). To amplify it, we use a Mini-Circuits ZPUL-30P amplifier connected to the signal output of the Brüel&Kjær 2804 power supply (through a Mini-Circuits BLP–1.9+ 1.9 MHz low pass filter, and self-built DC-block and surge protection circuits). The ZPUL-30P amplifier, powered by a lab bench power supply, provides about 40 dB gain (empirically measured) in the range of 0–500 kHz.[16]

Analog-to-digital conversion. To digitize the analog signal produced by the amplifier, we use data acquisition device (DAQ). Concretely we use a National Instruments PXIe 6356 connected to a 10 kHz high-pass filter and housed in a National Instruments PXIe-1073 chassis. The 6356 is capable of up to 1.25 Msample/sec at a resolution of 16 bits.

not having a flat frequency response, the 4190 capsule still has better sensitivity than the 4191 at 40 kHz.

[15]The Brüel&Kjær 4939 1/4” capsule can also be connected to the 2670 1/4” preamplifier, eliminating the need for the UA0035 adapter. However, this preamplifier has a relatively high noise floor compared to the 2669 preamplifier resulting in a lower signal-to-noise ratio.

[16]Brüel&Kjær also offers the Nexus amplifiers that also combine a built in power supply. However, these amplifiers have a built in 100 kHz low-pass filter which prevents the measurement of signals in the 100–350 kHz range (recall that these signal are already particularly weak due to poor performance of the 4939 capsule in these frequencies). Moreover, Nexus amplifiers have noise density of 13.4 nV/$\sqrt{\text{Hz}}$ which is worse then the ZPUL-30P.

References

[AA04] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *SP '04: Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 3–11, Washington, DC, USA, 2004. IEEE Computer Society.

[And08] Ross J. Anderson. *Security engineering - a guide to building dependable distributed systems (2. ed.)*. Wiley, 2008.

[BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[BBB+12] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management part 1: General. In *NIST Special Publication 800-57*. National Institute of Standards and Technology, 2012. URL: `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`.

[BDG+10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *USENIX Security Symposium*, pages 307–322, 2010.

[BGW01] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001.

[BHL06] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in wep's coffin. In *IEEE Symposium on Security and Privacy*, pages 386–400, 2006.

[BK75] H. E. Bass and Roy G. Keeton. Ultrasonic absorption in air at elevated temperatures. *The Journal of the Acoustical Society of America*, 58(1):110–112, 1975.

[BT11] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *ESORICS*, pages 355–371, 2011.

[BWY06] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 245–254. ACM, 2006.

[CDF+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880, November 2007. URL: `http://www.ietf.org/rfc/rfc4880.txt`.

[CMR+13] Shane S. Clark, Hossen A. Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. Current events: Identifying webpages by tapping the electrical outlet. In *ESORICS*, pages 700–717, 2013.

[Com13] Committee on National Security Systems. Index of national security systems issuances, September 2013. URL: `https://www.cnss.gov/CNSS/issuances/Issuances.cfm`.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

[EB72] L.B. Evans and H.E. Bass. Tables of absorption and velocity of sound in still air at 68° F. In *report WR72-2*. Wyle Laboratories, 1972.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Eni13] The Enigmail Project. Enigmail: A simple interface for OpenPGP email security, 2013. URL: https://www.enigmail.net.

[ETLR01] M. Elkins, D. Del Torto, R. Levien, and T. Roessler. MIME Security with OpenPGP. RFC 3156, August 2001. URL: http://www.ietf.org/rfc/rfc3156.txt.

[Gen] Genesis 27:5.

[GMP] The GMP Project. GNU Multiple Precision Arithmetic Library. URL: http://gmplib.org/.

[Gnu] The GnuPG Project. The GNU Privacy Guard. URL: http://www.gnupg.org.

[HS10] Tzipora Halevi and Nitesh Saxena. On pairing constrained wireless devices based on secrecy of auxiliary channels: The case of acoustic eavesdropping. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 97–108, New York, NY, USA, 2010. ACM.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[KO62] A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Proceedings of the USSR Academy of Sciences*, 145:293–294, 1962.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.

[LT06] Michael LeMay and Jack Tan. Acoustic surveillance of physically unmodified PCs. In *SAM '06: Proceedings of the 2006 International Conference on Security and Management*, pages 328–334. CSREA Press, 2006.

[Min] MinGW. Minimalist GNU for Windows. URL: http://www.mingw.org.

[MIT13] MITRE. Common vulnerabilities and exposures list, entry CVE-2013-4576, 2013. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4576.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[Nat82] National Security Agency. TEMPEST fundamentals. In *NACSIM 5000*. February 1982. URL: http://cryptome.org/jya/nacsim-5000/nacsim-5000.htm.

[Ope] OpenSSL. OpenSSL. URL: http://www.openssl.org.

[OS06] Yossi Oren and Adi Shamir. How not to protect pcs from power analysis, 2006. CRYPTO 2006 rump session. URL: http://iss.oy.ne.ro/HowNotToProtectPCsFromPowerAnalysis.

[RS85] Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In *EUROCRYPT*, pages 31–34, 1985.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SWT01] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001, 2001.

[TS04] Eran Tromer and Adi Shamir. Acoustic cryptanalysis: on nosy people and noisy machines, 2004. Eurocryt 2004 rump session; see `http://cs.tau.ac.il/~tromer/acoustic/ec04rump`.

[Wri87] Peter Wright. *Spycatcher*. Viking Penguin, 1987.

[YF13] Yuval Yarom and Katrina E. Falkner. Flush+reload: a high resolution, low noise, L3 cache side-channel attack. *IACR Cryptology ePrint Archive*, 2013:448, 2013.

[ZZT05] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. In *ACM Conference on Computer and Communications Security*, pages 373–382, 2005.