J Cryptol (2017) 30:392–443
DOI: 10.1007/s00145-015-9224-2

Journal of
CRYPTOLOGY

Acoustic Cryptanalysis*

Daniel Genkin
Technion, Haifa, Israel

Tel Aviv University, Tel Aviv, Israel
danielg3@cs.technion.ac.il

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

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

Communicated by François-Xavier Standaert.

Received 23 November 2014 / Revised 18 September 2015
Online publication 8 February 2016

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 the very low bandwidth of the acoustic side channel (under 20 kHz using common microphones, and a few hundred kHz using ultrasound microphones), and several 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 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 such attacks, using a plain mobile phone placed next to the computer, or a more sensitive microphone placed 10 meters away.

Keywords. Side channel attacks, Acoustic emanations, RSA, Cryptanalysis.

* This is an extended and updated version of results that appeared in CRYPTO 2014 [31]. The authors thank Lev Pachmanov for programming and experiments support during the course of this research.

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 [3] for a survey). Acoustic emanations are another potential channel, but so far it was used only in order to eavesdrop to slow electromechanical components such as keyboards and printers [1,6,52]. 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.

In this paper, 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 operations they execute. 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.[1]

Cryptanalytic side channel attacks typically require measurements with temporal resolution similar to the timescale of the target operations, but here the clock rate of the target cryptographic computation is several orders of magnitude faster (at the GHz scale), so observing individual operations seems to be hopeless. The first indication that acoustic emanation from electronic computers is of cryptanalytic interest was by Shamir and Tromer [48], observing that different RSA keys have distinguishable acoustic fingerprints. However, no approach has been proposed to extract actual key bits from the faint, noisy, and low-bandwidth acoustic information. In fact, a recent survey on side channels stated that while "acoustic effects have been suggested as possible side channel, the quality of the resulting measurements is likely to be low" [34].

Acoustic Cryptanalysis In this paper, we show that despite this skepticism, full key recovery via pure *acoustic cryptanalysis* is feasible on common software and hardware. As a typical case study, we focused on GnuPG (GNU Privacy Guard) [27], which is a popular cross-platform open-source implementation of the OpenPGP standard [13]. We first verified that the location of GnuPG's RSA signing (or decryption) operations can be easily identified by their unique acoustic fingerprint in the frequency domain, and that different secret keys can be *distinguished* by the spectrum of the sound made when they are used. We then developed a new *key extraction* attack which can find the full 4096-bit RSA secret keys used by GnuPG running on a laptop computer, within an hour, by analyzing only the sound picked up by either a plain cellular phone placed next to the

[1] Above a few hundred kHz, sound propagation in the air has a very short range, due to nonlinear 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 [9,20]. Moreover, the size (and thus sensitivity) of membrane-based microphone transducers is limited by the acoustic wavelength.

computer, or by a sensitive microphone from a distance of 10 m. In a nutshell, our attack extracts each of the key bits, in sequence, by crafting a special chosen RSA ciphertext that cause numerical cancellations deep inside GnuPG's modular exponentiation algorithm. This causes all-zero words to appear frequently in the innermost loop of the algorithm, thereby causing a specific branch to be taken there, but only if the attacked key bit is 1. A single iteration of that loop is much too fast for direct acoustic observation, but in our attack the effect is repeated and amplified over many thousands of iterations, resulting in a gross leakage effect that is discernible for hundreds of milliseconds and distinguishable in the acoustic spectrum. Thus, our attack causes key-dependent side channel leakage in GnuPG's RSA implementation and moreover utilizes GnuPG's *own code* in order to amplify the aforementioned leakage. This allows us to create a clearly observable low-bandwidth leakage which reveals the attacked key bit.

Chosen Ciphertexts by E-Mail Our key extraction technique requires the decryption of multiple ciphertexts which are adaptively chosen by the attacker. Prior works which used chosen plaintexts or ciphertexts required direct access to the input of the protected device, or attacked network protocols such as SSL/TLS [4] or WEP [7,8]. To break GnuPG, we used a new attack vector based on Enigmail [22], which is a popular plugin to the Thunderbird e-mail client that enables transparent signing and encryption of e-mail messages using GnuPG, following the OpenPGP [13] and PGP/MIME [23] standards. To enable meaningful user notification about incoming e-mails, Enigmail automatically decrypts each e-mail as soon as it is received, provided that the GnuPG passphrase is cached or empty. In this case, an attacker can e-mail suitably crafted messages to the victims (backdated, so they go unnoticed), await their arrival at the target computer, and observe the acoustic signature of their automatic decryption, thereby closing the adaptive attack loop without manual intervention by the recipient.

Applicability Our observations apply to many laptop computers made by various vendors and running various operating systems. Signal quality and effective attack distance vary greatly and seem to be correlated with the computer's age (i.e., older computers tend to emit stronger and more informative sounds).

Our attacks are applicable to all recent versions of the GnuPG 1.x series at the time of the initial publication of our results [31], i.e., up to GnuPG 1.4.15 (released on 15 Oct. 2013). The "square-and-*always*-multiply" side channel mitigation introduced in GnuPG 1.4.14, ironically, helps our attack by amplifying the aforementioned effect of the zero value in 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 frequencies where the useful leakage signal resides (and thus the requisite measurement bandwidth). Our specific key extraction technique can be mitigated by countermeasures against chosen-ciphertext attacks, 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 Prior to publication, we disclosed our detailed attack to GnuPG developers and main distributors CVE-2013-4576 [39] , suggested suitable countermeasures, and worked with the developers to test them. New versions of GnuPG 1.x and of libgcrypt (which underlies GnuPG 2.x), containing ciphertext randomization and normalization countermeasures and resistant to 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.

Follow-up works demonstrate faster physical side channel attacks on GnuPG's RSA and ElGamal square-and-always-multiply implementations [29], as well as on the windowed exponentiation in later GnuPG versions [28]. GnuPG 1.4.19 and Libgcrypt 1.6.3 mitigate those attacks as well.

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 Fig. 3) 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 contain hidden microphones that will be placed in perfect alignment when 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 (see Figs. 27, 28). While these are very effective at attenuating compromising electromagnetic radiation ("TEMPEST"), as we empirically verified, they are nearly transparent to acoustic emanations. 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 [24]. Analysis of sound emanations from mechanical devices is a newer affair, with precedents in military context such as identification of vehicles (e.g., aircraft and submarines) via the sound signature of their engine or propeller [38]. 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 [50, pp. 103–107] gives an informal account of the "ENGULF'' operation by MI5 and GCHQ, which used a phone tap to eavesdrop on

a Hagelin cipher machine in the Egyptian embassy in London by counting the clicks during the rotors' secret-setting procedure. 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 [49] and the references therein). Timing of keystrokes can, of course, be detected acoustically. Later work by Asonov and Agrawal [1], improved by Zhuang et al. [52], Berger et al. [12], and by Halevi and Saxena [32], 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 are 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. [6] 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 [48] 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 (Sects. 3, 4), but were not able to recover individual key bits. LeMay and Tan [37] 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 placing a resistor in series with the device's power supply and measuring the voltage across it, at a sampling rate which is on the same order of magnitude as the target's clock rate, an attacker may learn information about the individual operations performed by the device. From this information, the attacker can often deduce information about its secret internal state. Using suitable cryptanalytic techniques, this has been used to extract secret keys from many ciphers (including DES, AES, and RSA), on many types of hardware. See [34,41] and the references therein. Non-cryptographic power measurements on PCs was demonstrated by [15,19].

EM Analysis Electromagnetic radiation emitted by devices can likewise be exploited for key recovery. This has been demonstrated for smartcards and FPGAs (in [2,25,45] and many subsequent works), and recently also for PCs [28,29].

Chassis Potential Analysis Follow-up work [29] observes and exploits another side channel. The electrical fluctuations on the chassis of laptop computers, in reference to the mains earth ground, can also be used for side channel key extraction attacks. Such a measurement can be taken directly by contact with the laptop's chassis, the shield of any cable connected to the laptop, or even through the body of a human touching the laptop.

Timing Attacks Timing attacks, introduced by Kocher [36], 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 [36], Brumley and Boneh [4], and Brumley and Tuveri [11] 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") [42] 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 October 1970, remain classified, but have since been canceled by memorandums NSTISSC-052-95 and NSTISSC-002-98 (see [16]). A comprehensive list of all the past and present publications by the US government's Committee on National Security Systems [16], 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. The latest FIPS 140-3 draft [43], whose Appendix F specifically addresses noninvasive attack (defined as "an attack that can be performed on a cryptographic module without direct physical contact with components within the cryptographic boundary of the module"), likewise mentions power, electromagnetic, and timing analysis, but not acoustics.

One is thus led to conclude that acoustic emanations from historic *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 [51] 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 Sect. 9.1).

1.4. *Paper Outline*

The paper is organized as follows. We start by introducing our experimental setup (Sect. 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 (Sect. 3). We show that acoustic leakage easily allows distinguishing between different RSA keys (Sect. 4). We proceed to present our approach for extracting RSA keys from GnuPG as well as report on the empirical performance of our attack (Sect. 5). We then explain why our approach works by analyzing the relevant code of GnuPG (Sect. 6) and give a more detailed report of our empirical results (Sects. 7, 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 (Sect. 9). We empirically compare our attacks to timing attacks (Sect. 10). Finally, we discuss some countermeasures against acoustic cryptanalysis (Sect. 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: a 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 the "Appendix," 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 laboratory 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

Fig. 1. 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.

transducer is embodied in a *microphone capsule*. We use three such capsules, with different frequency versus sensitivity trade-offs: Brüel&Kjær model 4939 (up to 350 kHz), 4190 (up to 40 kHz), and 4145 (up to 21 kHz). See Fig. 1 for photographs.

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-noise amplifier, filter it, and digitize it (at up to 1.25 Msample/sec) using a National Instruments 6356 data acquisition device. See the "Appendix" 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 Fig. 2.

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 high-pass filter cascaded with a 150 kHz RC low-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.[2] 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 use ubiquitous compact hardware that is readily available to every potential attacker: a mobile phone (see Fig. 3). The small size and low cost come at the

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

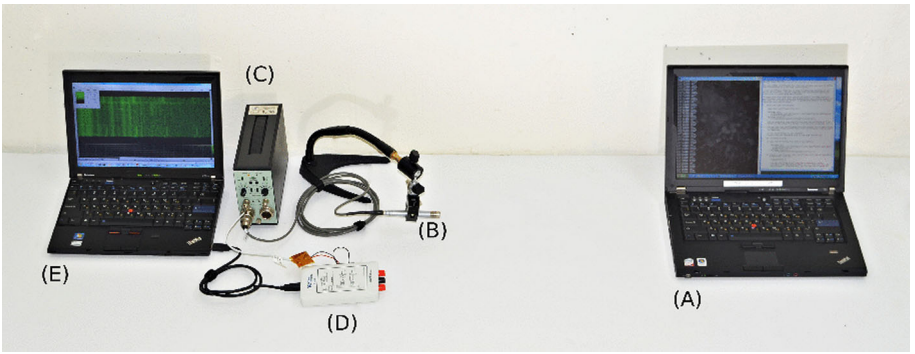


Fig. 2. Photograph of our portable setup. In this photograph, *a* is a Lenovo ThinkPad T61 target, *b* is a Brüel&Kjær 4190 microphone capsule mounted on a Brüel&Kjær 2669 preamplifier held by a flexible arm, *c* is a Brüel&Kjær 5935 microphone power supply and amplifier, *d* is a National Instruments MyDAQ device with a 10 kHz RC high-pass filter cascaded with a 150 kHz RC low-pass filter on its A2D input, and *e* is a laptop computer performing the attack. Full key extraction is possible in a similar configuration from a distance of 1 m (see Sect. 5.4).

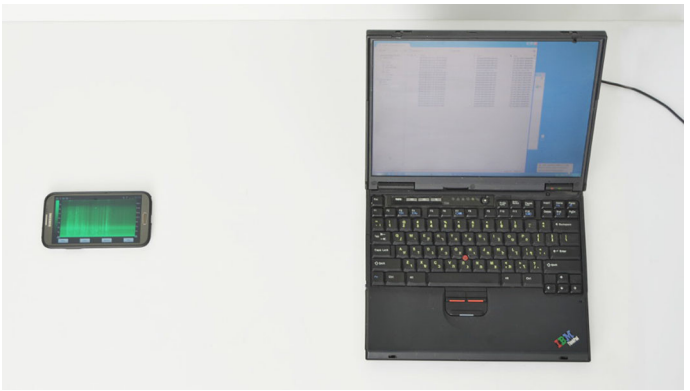
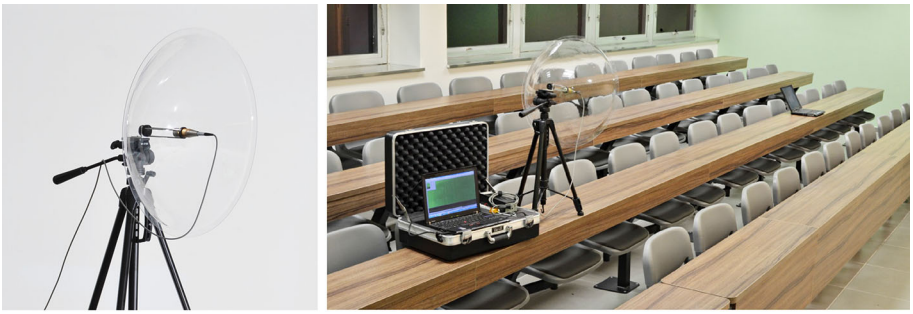


Fig. 3. Physical setup of a key recovery attack. A mobile phone (Samsung Note II) is placed 30 cm from a target laptop. The phone's internal microphone points toward the laptop's fan vents. Full key extraction is possible with this configuration and distance (see Sect. 5.4).

price of low-quality microphones and amplifiers, not designed for the sound frequencies and amplitudes sought in 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 microphone and transmitting the recorded signal to a workstation for further analysis. The specifications of the microphone, amplification, filtering, and A2D hardware were not available to us. We observe that sensitivity is lower and noise 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).



(a) Parabolic microphone **(b)** Distant attack

Fig. 4. **a** Brüel&Kjær 4145 microphone capsule and 2669 preamplifier, attached to a transparent parabolic reflector (56 cm diameter), on a tripod and **b** same, connected to the portable measurement setup (in a padded briefcase), attacking a target laptop (far *right*) from a distance of 4 m.

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 Fig. 4a). As discussed in Sect. 5.4, this increases the effective range of our key extraction attacks from 1 m to 10 m (see Fig. 4b).

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.

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 5 shows a recording of the Evo N200 laptop while executing our program using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule.

As shown in Fig. 5, the leakage of the Evo N200 is present all over the 0–350 kHz spectrum. Moreover, different types of operations can be easily distinguished. While

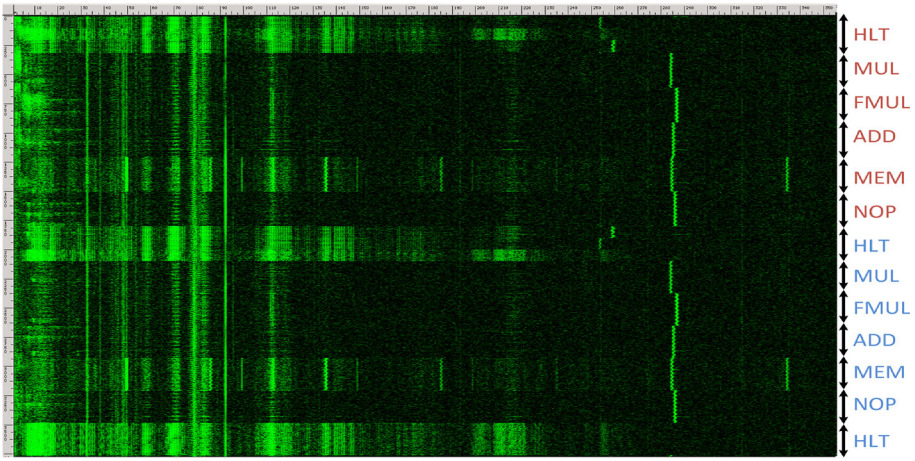


Fig. 5. Acoustic measurement frequency spectrogram of a recording of different CPU operations executed on the Compaq Evo N200 target using the lab-grade setup and Brüel&Kjær 4939 microphone capsule. The horizontal axis is frequency (0–350 kHz), the vertical axis is time (3.7 s), and intensity is proportional to the instantaneous energy in that frequency band.

the Compaq Evo N200 laptop computer is quite antiquated, similar leakage (in lower-frequency ranges) was detected using the lab-grade setup and the Brüel&Kjær 4190 capsule on current models as well (Lenovo ThinkPad W530). See Fig. 6 for comparison between three operations using various target computers.

3.2. *Distinguishing Various Loop 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 7 shows a recording of the Evo N200 executing 1 s loops of ADD instructions. Each loop is repeated for 1 s, but the size of the code inside the loop (and thus the number of repetitions) differs. As shown in Fig. 7, the code sizes can be easily distinguished. Note how some signal components vary, in both frequency and amplitude, depending on the loop length. The dependence is a complicated, nonlinear, and non-monotone function, mediated by the (undocumented) dynamics of the switch mode power supply whose components vibrate, and (as shown above) also affected by the type of instructions executed in the loop

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 8 shows an acoustic recording using the same setup as in Fig. 5, except that a sheet of cork is placed between the microphone and the laptop. As

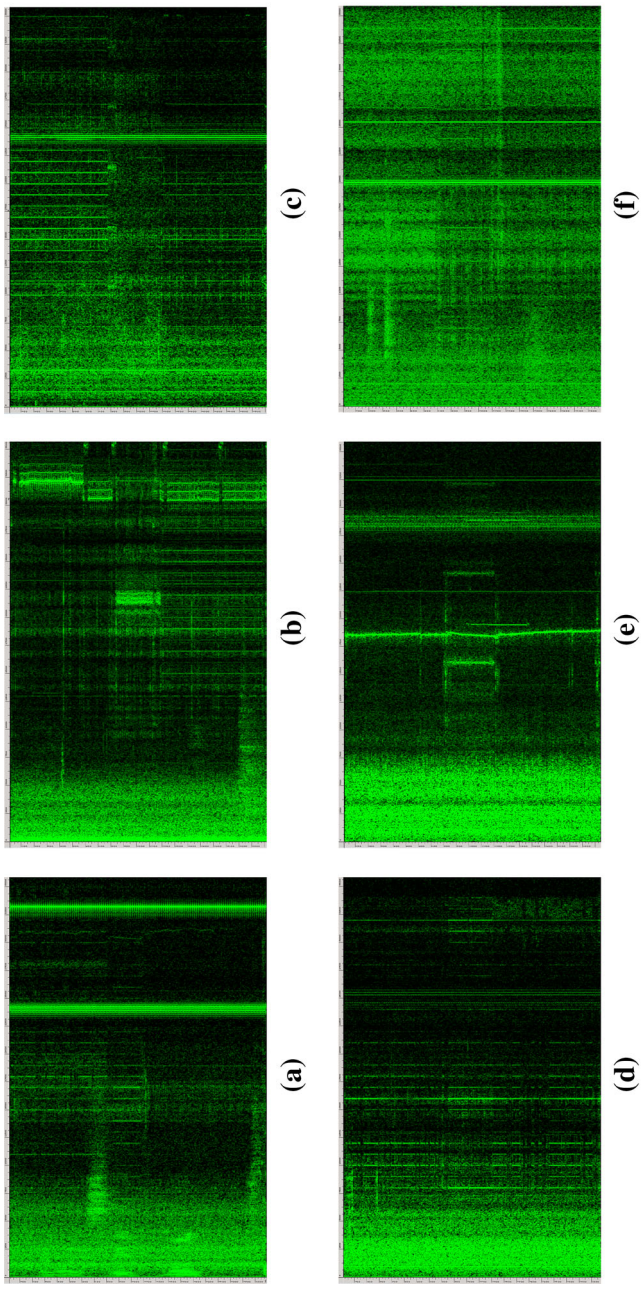


Fig. 6. Acoustic emanations (2 s, 0–35 kHz) of various target computers performing MUL, HLT, and MEM in this order. Note that the three operations can be distinguished on all machines. **a** Asus N55SF, **b** Dell Inspiron 7720, **c** HP Pavilion Sleekbook 15-b005ej, **d** HP ProBook 4530s, **e** Samsung NP300V5A, and **f** Lenovo ThinkPad W530.

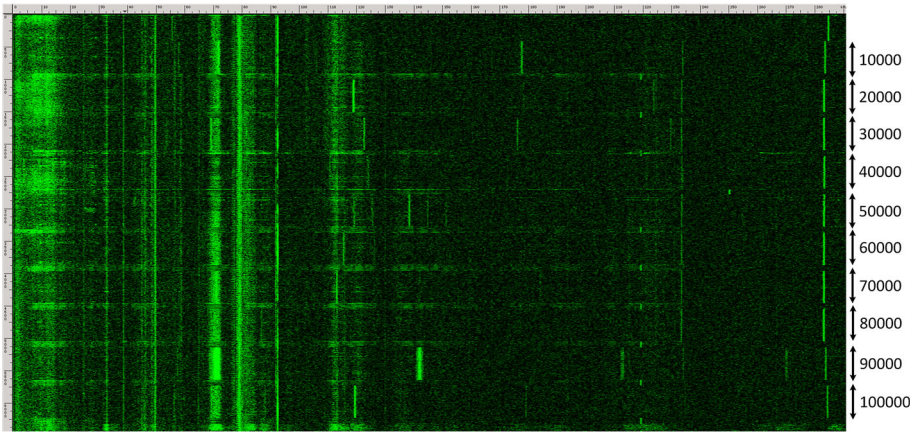


Fig. 7. Acoustic signature (6 s, 0–290 kHz) of loops in different lengths of ADD instructions executed on the Evo N200. The loop length is indicated next to its acoustic signature. Recorded using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule.

can be seen, the signal is severely attenuated. Similar results were obtained by placing a grounded metal mesh between the microphone and target, which did not significantly affect the signal, and then covering it with cardboard, which severely attenuated the signal (see Fig. 28). Lastly, the microphones and cables we use are fully shielded (except for the fine vents in front of the capsule; see Fig. 1); grounding these shields had no effect on the signal. We conclude that the measured signals are indeed acoustic emanations.[3]

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, they are 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 onboard 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 and modulates the dynamics of the pulse-width-modulation-based voltage regulator. 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.

[3] Follow-up research [28,29] observed similar signals from PCs on other channels: ground, power, and electromagnetic. The somewhat higher bandwidth of those channels allows for faster attacks.

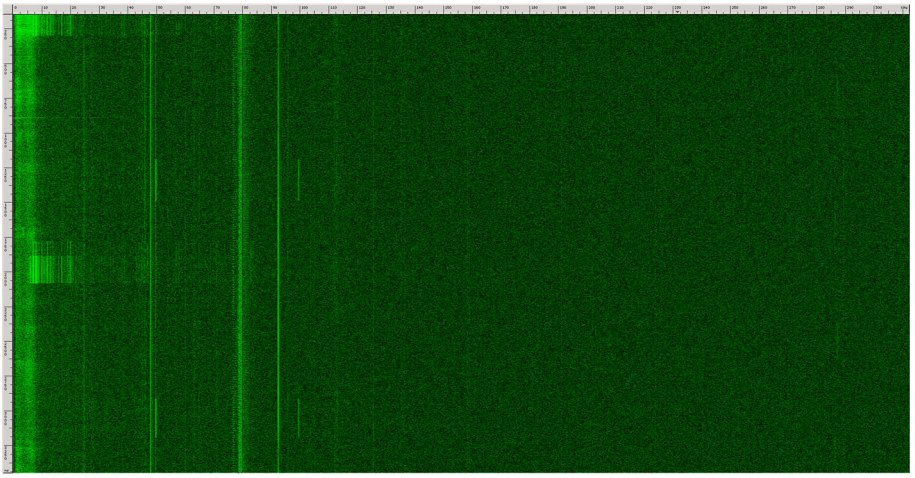


Fig. 8. Acoustic measurement frequency spectrogram (3.7 s, 0–310 kHz) of a recording of different CPU operations using the Brüel&Kjær 4939 microphone capsule in identical physical conditions and measurement setup as in Fig. 5, except this time the capsule is acoustically muffled by a cork sheet.

In one attempt at better localization, we opened a desktop PC and pointed the high-frequency Brüel&Kjær 4939 microphone capsule at components on a 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 significantly affected the acoustic signal (see Fig. 9). 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.[4] 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).

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

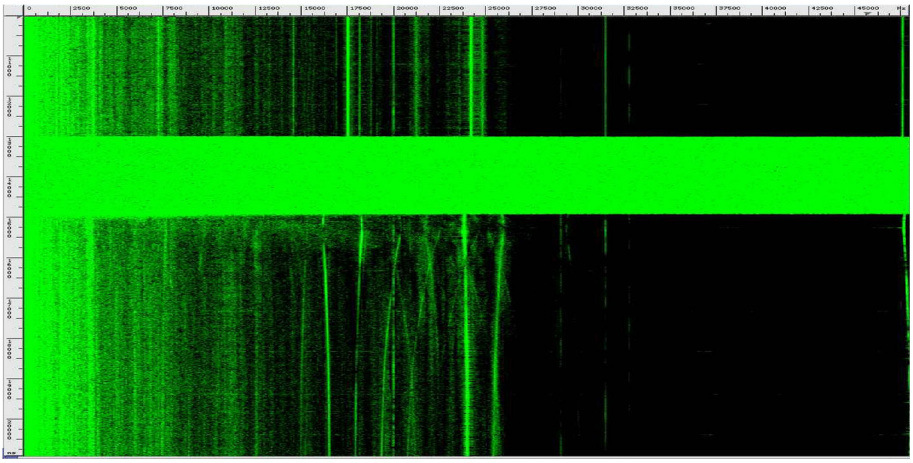


Fig. 9. Acoustic measurement frequency spectrogram (2 s, 0–45 kHz) of MUL operations while applying Quik-Freeze spray to a capacitor. After the spraying (which is loud enough to saturate the measurement), signal components shift by as much as 1 kHz. (PC Chips M754LMR motherboard recorded by a Røde NT3 condenser microphone, Alto S-6 mixer and a Creative Labs Audigy 2 sound card.).

3.4. *Microphone Placement*

The placement of the microphone relative to the laptop body has a great influence on the obtained signal. Ideally, we would like to measure acoustic emanations as close as possible to the CPU's onboard 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. Orientation is significant, especially at high frequencies, and is usually optimal when the microphone points toward the hole through which the sound emanates.[5]

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 wide-band noise (which can overload the amplifier at high gain). Also, electromagnetic inter-

[5] For characterization of a new target computer, we suggest the following protocol. Run a simple, controlled computation pattern on the target (see Sect. 3.1). Try several sensitive microphones, of different frequency responses (see Sect. 2 and the "Appendix"). Observe a real-time spectrogram while placing the microphone in various positions around the target, focusing on vents and other holes in the chassis, pointing at the hole and in close proximity (e.g., 1 cm). Identify positions and frequency bands exhibiting distinct computation-dependent signals. Then, choose the microphone providing the best signal-to-noise ratio at these frequencies, and maximize the range subject to the constraints of the signal analysis and cryptanalytic processing.

ference, caused by the fan motor and pulse-width modulation (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.

4. GnuPG RSA Key Distinguishability

The results in Sect. 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) [27], 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 Sect. 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 many 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) used Windows XP (with its default background maintenance tasks running) and GnuPG version 1.4.14 without any source code modifications. We compiled GnuPG with the MinGW GCC version 4.6.2 compiler and default options.

4.1. *The Sound of a Single Secret Key*

Figure 10 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. This figure shows 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 by 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 moduli is clearly visible.

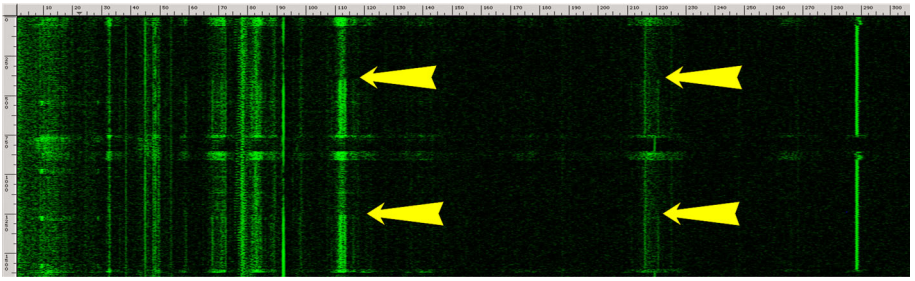


Fig. 10. Acoustic signature (1.6 s, 0–300 kHz) of two GnuPG RSA signatures executed on the Evo N200. Recorded using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule. The transitions between p and q are marked by *yellow arrows*.

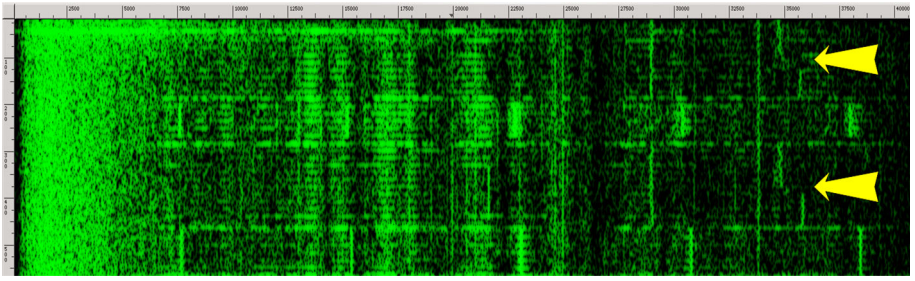


Fig. 11. Acoustic signature (0.5 s, 0–40 kHz) of two GnuPG RSA signatures executed on Lenovo ThinkPad T61. Recorded using the lab-grade setup and the Brüel&Kjær 4190 microphone capsule. The transitions between p and q are marked by *yellow arrows*.

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, Fig. 11 contains the recording of RSA signatures executed on a Lenovo ThinkPad T61 using the same key as in Fig. 10. 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 Fig. 12. Here, we used GnuPG 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 each RSA signature (and in fact, each exponentiation using modulus p or q) appearing in Fig. 12 has a unique spectral signature. While across a large sets of keys, these spectral signatures do not appear to be unique or (by themselves) reveal the bits of the key, the ability to distinguish between various keys taken from a small key set is still of interest in

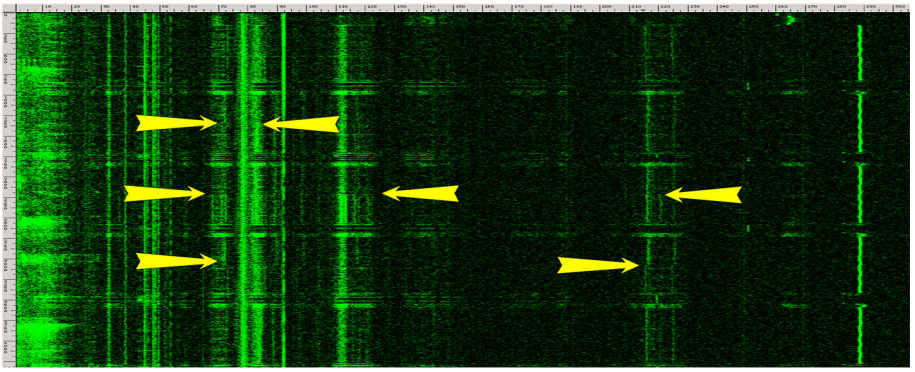


Fig. 12. Acoustic signature (4 s, 0–300 kHz) of five GnuPG RSA signatures executed on Evo N200. Recorded using the lab-grade setup and the Brüel&Kjær 4939 high-frequency microphone capsule. The transitions between p and q are marked by *yellow arrows*.

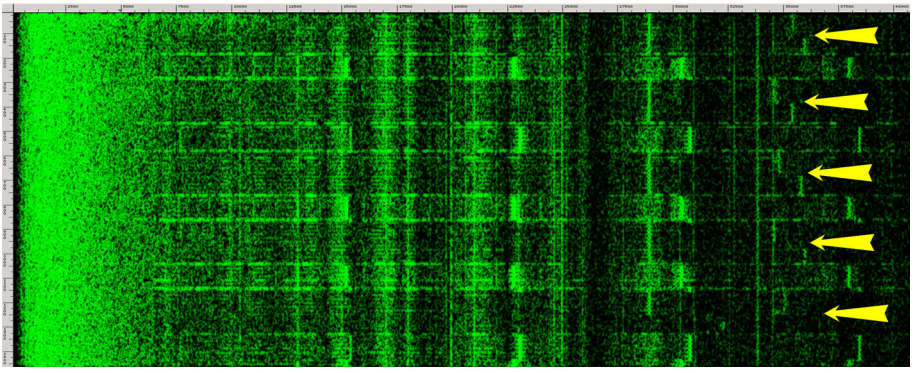


Fig. 13. Acoustic signature (1.4 s, 0–40 kHz) of five GnuPG RSA signatures executed on a Lenovo ThinkPad T61 and recorded using the lab-grade setup and the Brüel&Kjær 4190 microphone capsule. The transitions between p and q are marked by *yellow arrows*.

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 Fig. 13 for a recording of RSA signatures, using the keys from Fig. 12 executed on a Lenovo ThinkPad T61).

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

5. Overview of GnuPG RSA Key Extraction

Recall that in the RSA cryptosystem [47], 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 security. 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 [5]. In this section, we show an attack that can extract whole 4096-bit RSA keys within about 1 h 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 $m_q = c^{d_q} \mod q$. 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 [26], 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 (as introduced in GnuPG v1.4.14—see Sect. 9.1). 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 t-bit number x, namely $\lceil t/32 \rceil$. The constant KARATSUBA_THRESHOLD, defined to be 16, means that when the ciphertext c contains more than 15 limbs, the Karatsuba multiplication algorithm [35] is used; otherwise, the grade-school 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 Sect. 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

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, where $d_k \cdots d_1$ are the bits of d.
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 ← k downto 1 do
6:          m ← m^2                                  ▷ Karatsuba or grade-school squaring
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
```

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 Sect. 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 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 [4]; see Sect. 10). For each bit q_i of q, starting from the most significant bit position ($i = 2048$), we assume that key bits $q_{2048} \ldots q_{i+1}$ 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 [17] (following Rivest and Shamir [46]) 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 ith bit is 0, and whose remaining (low) bits are 1. Consider the invocation of RSA decryption on $g^{i,1}$. There are two cases, depending on q_i.

- $\boldsymbol{q_i = 1}$. Then, $g^{i,1} < q$. The ciphertext $g^{i,1}$ is passed as the variable c to Algorithm 1, in which (with the modification introduced earlier) the modular reduction in c in line 3 returns c (since $c = g^{i,1} < q$). Thus, the structure of c (a 2048 bit number whose $i-1$ lowest bits are set to 1) is preserved, and it is passed to the multiplication routines in lines 10 and 12.
- $\boldsymbol{q_i = 0}$. Then, $q \leq g^{i,1}$. Thus, when $g^{i,1}$ is passed as the variable c to Algorithm 1, the modular reduction in 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 number of size $i-1$ bits. 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 with its low bits set to 1 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.[7]

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 second operand with its low bits set to 1.

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.

[7] Ironically, the latest GnuPG implementations use the side channel mitigation technique of always multiplying the intermediate results by the input, but this only helps our attack, since it doubles the number of multiplications and replaces their random timing with a repetitive pattern that is easier to record and analyze.

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 and satisfying $n = pq = 0 \mod q$. Next, notice that $g^{i,1} + n \bmod q = g^{i,1} \bmod q + n \bmod q = g^{i,1} \bmod q$. 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 [4]). Its implementation, complications, and robustness concerns are discussed below, in Sect. 8.

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.

```
1:  procedure SimplifiedAttack(pk)
2:      g ← 2^((m/2)−1)                          ▷ g is a m/2 bit number of the form g = 10···0
3:      for i ← m/2 − 1 downto 1 do
4:          g^{i,1} ← g + 2^{i−1} − 1            ▷ set all the bits of g starting from i − 1-th bit to be 1
5:          b ← DECRYPT_AND_ANALYZE_LEAKAGE_OF_Q(g^{i,1} + n)   ▷ obtain the i-th bit of q
6:          g ← g + 2^{i−1} · b                  ▷ update g with the newly obtained bit
7:      q ← g
8:      p ← n/q
9:      return (p, q)
10: end procedure
```

5.3. *Acoustic Leakage of the Bits of q*

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

Figure 14a shows a typical recording of RSA decryption when the value of the attacked bit of q is 0, and Fig. 14b shows the same for an adjacent bit of q which is 1.[8] Several effects are shown in the figures. Recall that GnuPG first performs modular exponentiation

[8] In this example, we kept the key fixed, to avoid key-dependent changes in the acoustic signature (see Sect. 4). The two bits shown are both in the most significant limb, and are thus handled similarly by the code and induce similar value-dependent leakage, as shown in Fig. 20.

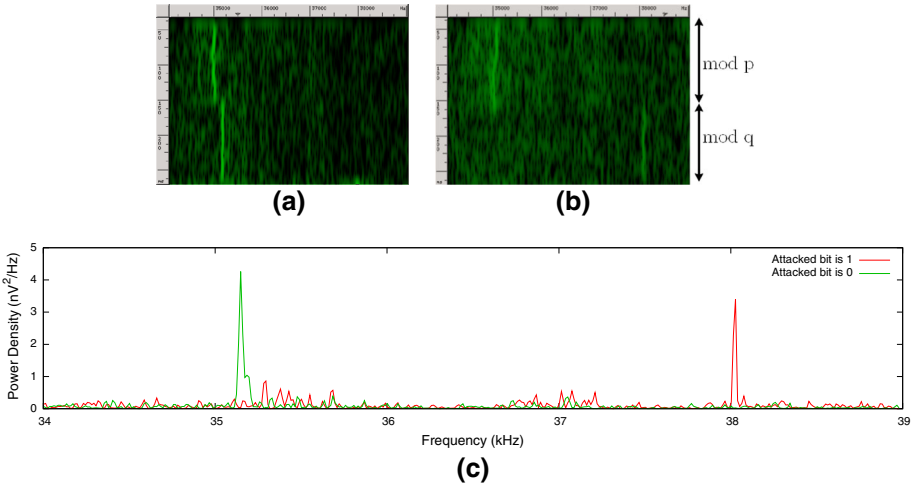


Fig. 14. Acoustic emanations of RSA decryption executed on a Lenovo ThinkPad T61 target for different values of the attacked bit ($q_{2039} = 1$ and $q_{2038} = 0$, both in the most significant limb) in the same key. The recording was done using the lab-grade setup and the Brüel&Kjær 4190 microphone. **a** Attacked bit is zero, **b** attacked bit is one, and **c** frequency spectra of the second modular exponentiation.

using the secret prime p and then performs another modular exponentiation using the secret prime q. The transition between p and q is clearly visible here (as it was in Fig. 13).

Note, next, that the acoustic signatures of the modular exponentiation using the prime q (the second exponentiation) are quite different between Fig. 14a, 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.

The spectral signatures in Fig. 14c were computed from the acoustic signatures of the second modular exponentiation in Fig. 14a, 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 0-valued bit and a 1-valued 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) look the same in Figs. 14a, 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 0-valued bits and 1-valued 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 Sect. 8).

5.4. *Overall Attack Performance*

We conducted our attack in a variety of measurement setups, on various target machines and software configurations. The attack's success and its running time (due to repeated

measurements and backtracking) depend on many physical parameters. These include the machine model and age, the signal acquisition hardware, the microphone positioning, ambient noise, room acoustics, and even ambient temperature (affecting fan activity). The following are examples of successful key extraction experiments.

Ultrasound Frequency Attack Extracting the topmost 1024 bits of q (and thereby, the whole key) from GnuPG 1.4.14, running on a Lenovo ThinkPad T61 laptop, in a typical office environment, takes approximately 1 h. Since this laptop's useful signal is at approximately 35 kHz, there is no need to use the lab-grade setup, and we can use instead a Brüel&Kjær 4190 microphone capsule connected to our portable setup (see Sect. 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 m. Moreover, by placing the Brüel&Kjær 4190 microphone in a parabolic reflector, we were able to extract all the bits automatically from the range of 4 meters (see Fig. 4b; Sect. 2.4). With human-assisted signal processing, we extended the range up to 10 m.

Mobile Phone Attack Lowering the leakage frequency also allows us to use lower-quality microphones such as the ones in smartphones. As described in Sect. 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 cm). However, it is still possible to achieve key extraction on certain target computers, simply by placing the phone's microphone near to and directed toward the fan exhaust vent of the target while running the attack (see Fig. 3). 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 [13] and the GnuPG software [27] 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 e-mail.

Enigmail Attack Enigmail [22] 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 [23]. 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 e-mail" popup notification. This decryption occurs 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, or crafted to look like spam (that is filed away without notification), they may even go unnoticed while the attack is ongoing.

6. Analyzing the Code of GnuPG RSA

In this section, we analyze how our attack affects the code of GnuPG's multiplication routine and causes the differences presented in Sect. 5.3. We begin by describing the multiplication algorithms used by GnuPG (Sect. 6.1) and then proceed (Sects. 6.2, 6.3) to describe the effects of our attack on the internal values computed during the execution of these algorithms.

6.1. *GnuPG's Multiplication Routine*

GnuPG's large-integer multiplication routine combines two multiplication algorithms: a basic grade-school multiplication routine, and a variant of a recursive Karatsuba multiplication algorithm [35]. The chosen combination of algorithms 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 grade-school multiplication routine and the current bit of q. This change in the control flow lets us leak q one bit at a time.

We now proceed to describe GnuPG's implementation of grade-school 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 GnuPG's code resides inside the basic multiplication routines. Both of the basic multiplication routines used by GnuPG are almost identical implementations of the simple, quadratic-complexity grade-school multiplication algorithm, with optimizations for multiplication by limbs equal to 0 or 1 (see 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 min. Prior versions (e.g., Enigmail 1.5.2) cached the passphrase internally, by default for 5 min.

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 $u + v \cdot w \cdot 2^{32 \cdot i}$. Note 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 9 continues to the next limb of b. 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 affecting 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 Sect. 5.1 and by the Karatsuba multiplication routine implementing a variant of the Karatsuba multiplication algorithm [35] 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_{H}, v_{H} are the most significant halves of u and v, respectively, and u_{L}, v_{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**

Note 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 Sect. 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 $2048 - i$ bits are the same as q, its ith 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 $i-1$ 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 the 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-looking when the current bit of q is 0.

Thus, when the recursion eventually reaches its base case, 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 its 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 random-looking. 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 lengths (in limbs). These cases are handled by the topmost multiplication routine of GnuPG, 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.

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
```

6.2. *Attacking the Most Significant Limb of q*

In this section, we analyze the effects of our attack on MUL_BASECASE (Algorithm 3). Note 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 tree. Figure 15 shows the structure of this recursion tree; the gray 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 in Sect. 5.2, we see that for each bit i of q:

- If $q_i = 1$, then the second operand b of MUL mainly consists of limbs having all their bits set to 1. Next, calling KARATSUBA_MUL with such an operand results in the value $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$ in line 10 containing 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 gray leaves of Fig. 15) to contain mostly zero limbs.

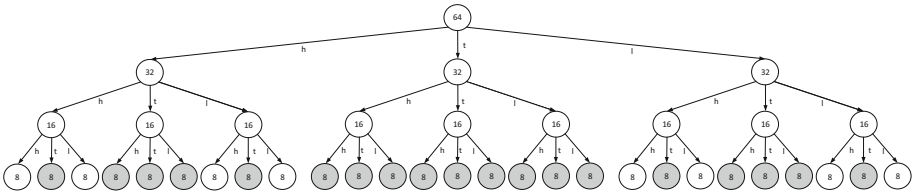


Fig. 15. Recursion tree created by Algorithm 4 during the first 32 bits of the attack. The size (in limbs) of the numbers handled by each recursive call is indicated inside each node while the variable holding the value returned by the call is indicated on the edge leading to the node. The leaves of the tree are computed using MUL_BASECASE (Algorithm 3). The leaves influenced by our attack are *gray* .

- If $q_i = 0$, then the second operand b of MUL consists of random-looking limbs. Thus, during the call to KARATSUBA_MUL, the value $b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$ in line 10 contains very few (if any) zero limbs, causing the second operand of *all* the calls to MUL_BASECASE located on *all* the leaves of Fig. 15 to consist of mostly nonzero limbs.

Recall the effect of zero limbs in the second operand on the code of MUL_BASECASE. Note that the control flow in MUL_BASECASE depends on the number of nonzero 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 the number of zero limbs in the second operand of MUL_BASECASE using our carefully chosen cipher texts.

Confirming the above analysis, Fig. 16 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 several 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).

Note that the Karatsuba multiplication algorithm is (indirectly) called during the main loop of the modular exponentiation routine (Algorithm 1) once per bit of d_q as computed by the RSA decryption operation. Since d_q is a 2048 bit number, we get that the leakage generated by line 11 in MUL_BASECASE (Algorithm 3) 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 gray leaf of Fig. 15 for a total of 19 times in each of the 2048 multiplications in that loop.

Thus, we get that leakage generated by line 11 in MUL_BASECASE is repeated approximately $2048 \times 19 \times 7 = 272{,}384$ times. This repetition is what allows the leakage generated by line 11 in MUL_BASECASE to be detected using only low bandwidth measurements.

6.3. *The Remaining Bits of q*

Unfortunately, the analysis in Sect. 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

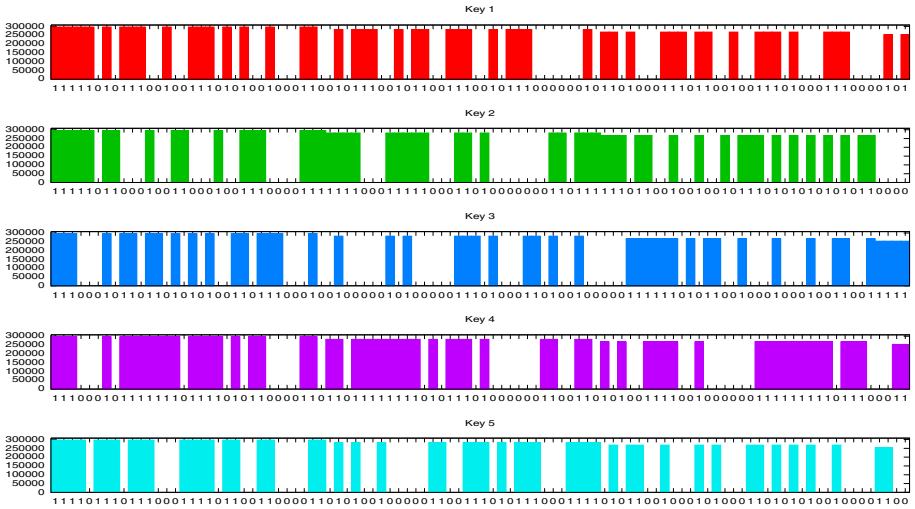


Fig. 16. Number of zero limbs in the second operand of MUL_BASECASE(a, b) during an execution of the modular exponentiation routine with the secret prime q using our attack and randomly generated keys. The correct bit values are indicated on the horizontal axis of each key.

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 (Algorithm 1) is shorter than 64 limbs. Since in lines 10 and 12 of MODULAR_EXPONENTIATION 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. Thus, instead of a single call to the Karatsuba multiplication routine (Algorithm 4), GnuPG's multiplication routine might make several recursive calls to itself as well as several calls to Algorithm 4.

Nonetheless, there is still a connection between the 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 $2048 - i$ bits are the same as q, the ith bit is zero, and the rest of the bits are ones.
2. If $q_i = 0$, then c consists of $i - 1$ random-looking bits.

While the analysis in this case is not as precise as in Sect. 6.2, the number of zero limbs in the second operand of MUL_BASECASE still allows us to extract the bits of q (see Fig. 16).

The acoustic distinguishability of the two cases does vary with bit index and in particular is harder for the range $q_{1850}, \ldots, q_{1750}$. Using additional techniques, we recover these problematic bits and continue our attack.

7. Analyzing the Acoustic Leakage of GnuPG

In this section, we further analyze the acoustic leakage produced by GnuPG, supporting the code analysis in Sect. 6. Recall that Sect. 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), which is as follows:

1. If $q_i = 1$, then c is a 2048 bit number such that the first $2048 - i$ bits are the same as q, the ith bit is zero, and the rest of the bits are ones.
2. If $q_i = 0$, then c consists of $i - 1$ random-looking bits.

The value of the second operand c of MUL (Algorithm 5) is affected by our attack in two separate ways. Concretely, our attack affects both the number of limbs in c and the bit structure of c by making c random-looking or by making it have most of its bits fixed to 1. In this section, we examine how both the number of limbs in c and the bit structure of c affect the leakage produced by GnuPG.

Setup In the remainder of this work, all the measurements (unless indicated otherwise) were performed on a Lenovo ThinkPad T61, using the portable setup and a 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 lengths (in limbs). Note that since the code of GnuPG does not modify the ciphertext until it reaches MUL, the distribution of values of the second operand of MUL in this case is similar to the distribution created when attacking 0-valued bits of q.

For example, let c_0 be a randomly generated 63-limb ciphertext, and let c_1 be a randomly generated 57-limb ciphertext. Figure 17a shows a recording of RSA decryption of c_0, and Fig. 17b shows a recording of RSA decryption of c_1. As in Figs. 14a, b, the two modular exponentiation operations are clearly visible in Figs. 17a, b. Next, the two modular exponentiation operations during the decryption of c_0 produce a signal at lower

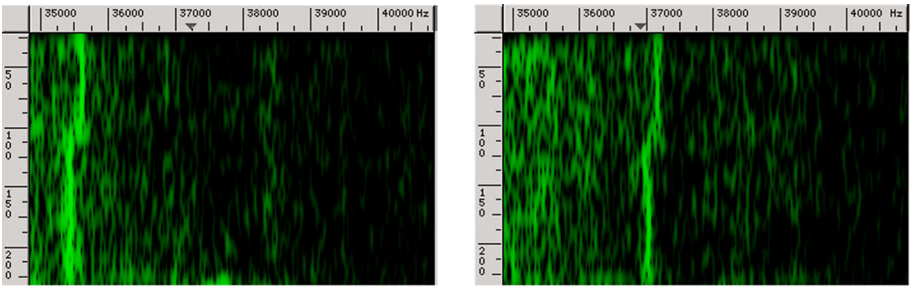


Fig. 17. Typical acoustic emanations (0.2 s, 35–40 kHz) of RSA decryption for randomly chosen ciphertexts with different numbers of limbs. (Lenovo ThinkPad T61 recorded by a Brüel&Kjær 4190 microphone.). **a** 63-limb ciphertext and **b** 57-limb ciphertext.

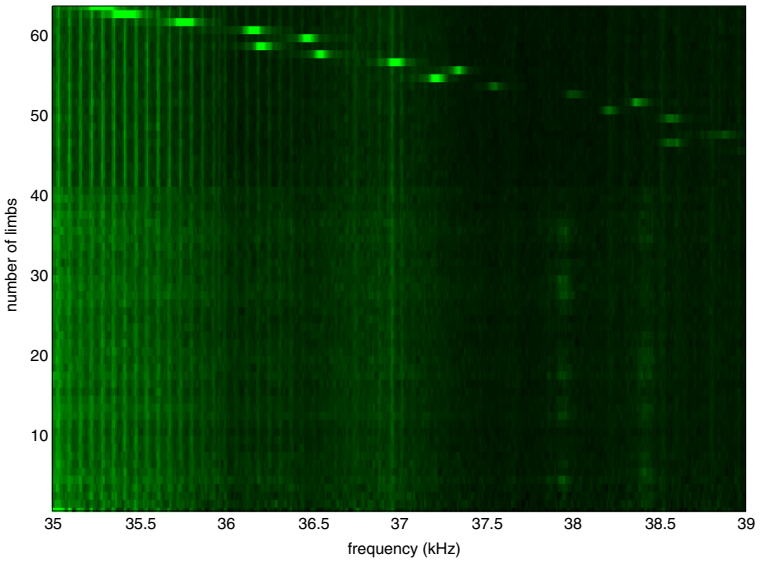


Fig. 18. Frequency spectrum of acoustic emanations of the second modular exponentiation during an RSA decryption of ciphertexts of various lengths. The horizontal axis is frequency, and the vertical axis is the number of limbs in the decrypted ciphertext .

frequency than during the decryption of c_1. Thus, this indicates that an attacker can in fact learn the number of limbs in the second operand of MUL.

A more systematic illustration of this effect can be seen in Fig. 18, which contains acoustic signatures of the second modular exponentiation on randomly generated ciphertexts of various lengths. Each row in Fig. 18 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 ciphertexts that are smaller in magnitude than q to avoid them 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 Fig. 18, the shorter the number (in limbs), the higher the frequency of the acoustic leakage. Moreover, note that the signal strength appears to degrade with the increase in frequency. We conjecture that this is due to the nonlinear frequency response of the Brüel&Kjær 4190 microphone capsule at these frequencies.[10] Thus, we confirm our hypothesis that a potential attacker might learn the number of limbs in the second operand of MUL by observing the acoustic leakage of the second modular exponentiation.

Finally, note that for each row i in Fig. 18, the second operand to MUL calls during the measured decryption has the same length (and similar distribution) as the second operand to MUL calls when attacking a bit in limb i of q, if that bit is 0. Thus, we expect

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

the acoustic signatures of the second modular exponentiation, when attacking 0-valued bits of q located in the ith limb, to be similar to the signatures obtained in row i in Fig. 18.

7.2. *Acoustic Leakage of Zero Limbs*

We now proceed to examine the effects of decrypting ciphertexts of various lengths (in limbs) that cause the second operand of the basic multiplication routine 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 1-valued 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 ciphertext c of length i. Next, we replace randomly chosen j limbs from c with the fixed limb ℓ. Finally, we repeat the above experiment 16 times, thereby obtaining a dataset of size 33,280 ciphertexts.

Figure 19a contains acoustic signatures of the second modular exponentiation on 64-limb ciphertexts generated from the above dataset. The signatures are arranged by the number of zero limbs in the second operand of MUL_BASECASE. Each row in Fig. 19a is generated by computing the frequency spectrum of the second modular exponentiation during the decryption of 64-limb ciphertexts in which the number of zero limbs in the second operand of MUL_BASECASE is as indicated in the row. Next, for each row, and for each frequency bin, we plot the median value across all frequency spectra for ciphertexts (in our dataset) corresponding to the row. Finally, the rows corresponding to values of

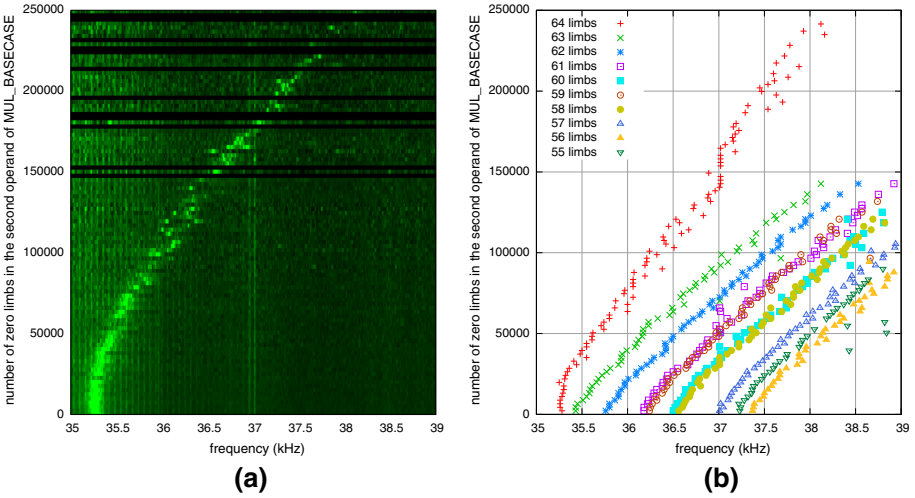


Fig. 19. Acoustic emanations of the second modular exponentiations during an RSA decryption of ciphertexts. The horizontal axis is frequency, and the vertical axis is the number of zero limbs on the second operand of MUL_BASECASE. **a** Ciphertexts of size 64 limbs (*black rows* correspond to zero limb counts that did not occur in any of the samples) and **b** frequency of the strongest signal (after filtering) during the second modular exponentiation of ciphertexts of various length.

zero limbs in the second operand of MUL_BASECASE that are not present in our dataset are filled with black. Each row in Fig. 19b is obtained by performing the same process as in Fig. 19a 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 shown in Fig. 19b, the frequency of the leakage produced by the second modular exponentiation depends on *both* the length of the second operand of MUL (Algorithm 5) and on the number of zero limbs in the second operand of MUL_BASECASE (Algorithm 3) during the decryption. In particular, if 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 GnuPG's RSA implementation.

8. Further Details on GnuPG RSA Key Extraction

We now give a more detailed account of our acoustic RSA key extraction attack. While the attack presented in Sect. 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 Fig. 14. Since in both cases we are attacking the most significant limb of q, the second operand of MUL is 64 limbs long. Note that the peaks in Fig. 14c are exactly as predicted by Fig. 19a. This is because 0-valued bits create very few zero limbs in the second operand of MUL_BASECASE, while 1-valued 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 0-valued and 1-valued 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 Sect. 8.4 for details).

This approach, however, poses a problem since it requires the attacker to obtain examples of leakage spectra corresponding to both 0-valued and 1-valued bits. Recall that q is chosen to be a prime whose most significant bit is always set to one. Moreover, this information is known to an attacker. Thus, an example of a leakage spectrum of 1-valued bit can be obtained by measuring the leakage resulting from the decryption of $g^{2048,1}$. Obtaining an example of a leakage spectrum of a 0-valued bit is trickier. This is because the attacker does not know in advance the location of the first 0-valued bit in q. However, this problem can be easily avoided. Consider any number ℓ such that $q < \ell < 2q$ (e.g., $\ell = 2^{2048} - 1$). Note that the reduction of ℓ modulo q is equivalent to computing $\ell - q$

and will cause the bits of the result to be random-looking, thus obtaining a spectrum similar to that of the sound of 0-valued bits of q at the beginning of the attack.

8.2. *Adapting to Changes in the Acoustic Signature*

Figure 20 presents frequency spectra of the second modular exponentiation during our attack, for 0-valued versus 1-valued bits. Note 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 Figs. 18 and 19a. This is because, as the attack progresses, the following two things occur: For the 0-valued bit case, the second operand of MUL becomes shorter (in limbs), and for the 1-valued bit case, 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 progresses, the changes in the spectra corresponding to 0-valued and 1-valued bits are small (at most 0.5 kHz) and only significant over time. This means that we can utilize previous observations about the leakage spectra in order to be able to correctly classify a currently unknown spectrum as corresponding to a 0-valued bit or a 1-valued 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 0-valued and 1-valued bits of q and obtains multiple samples of their decryption. This allows the attacker to generate a template of the leakage caused by 0-valued bit and a template of the leakage caused by a 1-valued bit.

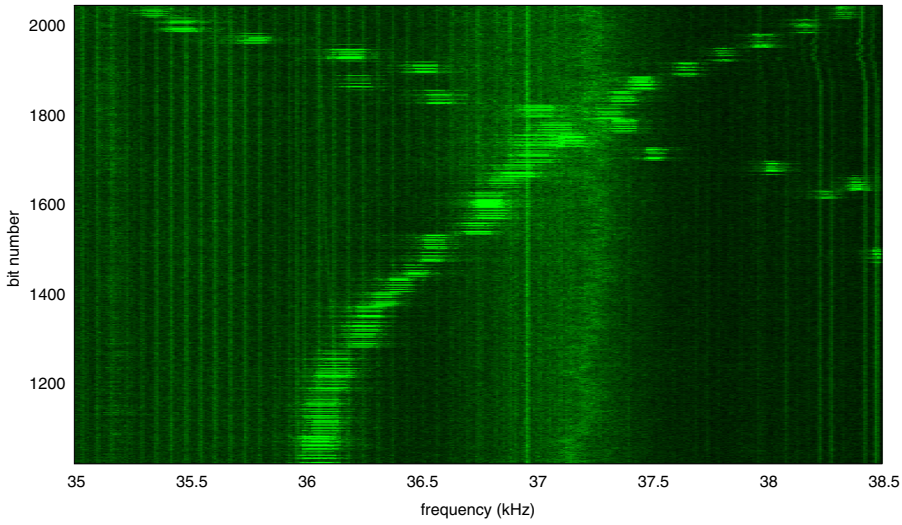


Fig. 20. Acoustic measurement frequency spectra (*horizontally*) of the second modular exponentiation, as the attack progresses (*vertically*). The jagged curve starting at *top left* corresponds to 0-valued bits. The curve starting at *top right* corresponds to 1-valued bits.

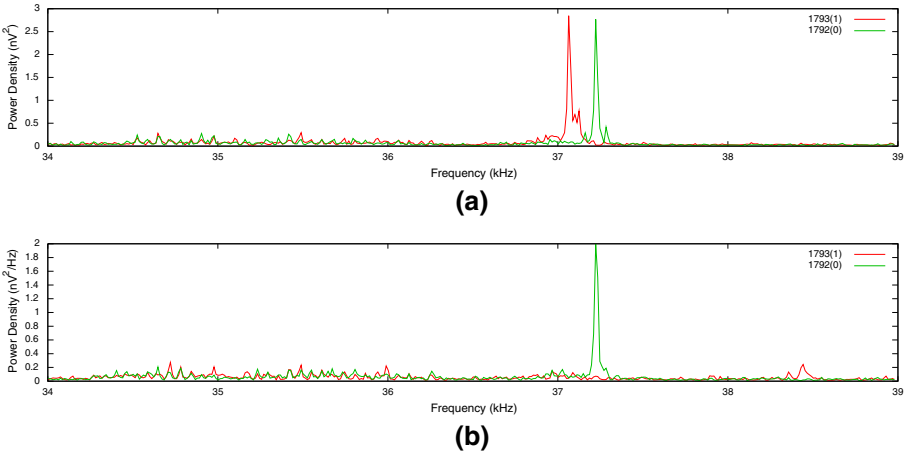


Fig. 21. Acoustic measurement frequency spectra of the second modular exponentiation when attacking the bits in the range of 1850–1750. The bit values are indicated in parentheses in the legend. **a** Using $g^{i,1}$ as cipher text and **b** using $g^{i,0}$ as cipher text.

The next 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 0-valued bit or to a 1-valued bit. This step might be repeated a few times until a certain classification is achieved. Second, during the template update step, new templates for 0-valued bits and 1-valued 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 0-valued and 1-valued bits by taking the median power value, in each frequency band, across all available spectra for 0-valued and 1-valued 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 Sects. 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. As shown in Fig. 20, the problem is that in bits 1850–1750, the distance between the signals resulting from a leakage of 0-valued bits and the signals resulting from a leakage of 1-valued bits is about 200 Hz (see Fig. 21a). This is less than the frequency change between subsequent bits (around 400 Hz), and thus, the bit values cannot be easily distinguished. (This behavior is qualitatively and quantitatively typical, across many target machines.) Recall that when attacking the ith 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 ciphertext used. Let $g^{i,0}$ be a 2048 bit number whose top $i-1$ bits are the same as q, whose ith bit is 1, and whose remaining bits are 0. Repeating the analysis in Sect. 5 using $g^{i,0}$, we can obtain a connection between q_i and the second operand c of

MUL, similar to the connection obtained using $g^{i,1}$. However, as shown in Fig. 21b, using $g^{i,0}$, it is now possible to distinguish the bits in the range of 1850–1750, thus allowing our attack to proceed. Note that the switch from $g^{i,1}$ to $g^{i,0}$ upon reaching the 1850th bit and backwards upon reaching the 1750th bit requires us to generate new templates in order to be able to correctly classify the remaining key bits. However, obtaining the templates is actually quite simple since at this point, all the previous key bits are known and can be used to generate the required templates. For example, when switching from $g^{i,1}$ to $g^{i,0}$ upon reaching the 1850th bit (the other case is handled similarly), let j be the smallest index of a 1-valued bit, and let j' be the smallest index of a 0-valued bit such that both j, j' are larger than 1850. Note that j and j' are already known to the attacker since all the bits above the 1850th bit have been extracted. The template for 1-valued bits is obtained by requesting several decryptions of $g^{j,0}$. Similarly, the template for 0-valued bits is obtained by requesting several decryptions of $g^{j',0}$.

Extracting All Bits Algorithm 6 is a pseudocode of our attack on GnuPG that 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 0-valued and 1-valued bits upon switching from $g^{i,1}$ to $g^{i,0}$ and back.

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.

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

Note the branch in line 6. This branch is specifically designed to deal with the problematic bits described in Sect. 8.3. In all of our experiments, this switching of ciphertexts depended only on the key size and the index of the bit being attacked, and not on the secret key, GnuPG version, or target machine.

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 the 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 0-valued and 1-valued bits. As described in Sect. 8.2, the templates are set dynamically and adaptively. When attacking q_i, the template T^b is the median spectrum across the last few spectra (we use 20) of the second modular exponentiation, obtained in previous iterations of the attack and classified as belonging to bit value b. (If data from previous iterations of the attack is not present, the templates are generated heuristically as described in Sect. 8.1.) Given two templates T^0, T^1 describing the signal power in dB at each frequency bin, q_i is classified as zero or one in the following steps.[11]

First 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 the target's acoustic emanations into *trace*. The trace is recorded or truncated so 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 Sect. 4).

Second 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.[12] The spectrum is truncated to the frequency range of interest (determined manually). We denote this aggregate spectrum by s. For example, Fig. 22 presents two such spectra, one for a 0-valued bit of q and the other for a 1-valued bit.

Third 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.

Fourth Stage: Normalization We would like to make the measurements independent of microphone 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 $\min(T_i^0, T_i^1)$.

[11] Our heuristic approach sufficed for achieving reliable key extraction. Improvements may be possible using the algorithmic approach of template attacks [14,18].

[12] A simpler approach is to take a single Fourier transform over the recording of the whole decryption period, but this is too sensitive to the transient loud noises in an typical office environment—due to sheer magnitude, they can contribute more to the result than the faint signal of interest. The median, taken across many smaller time windows, rejects temporally local outliers and proved much more robust.

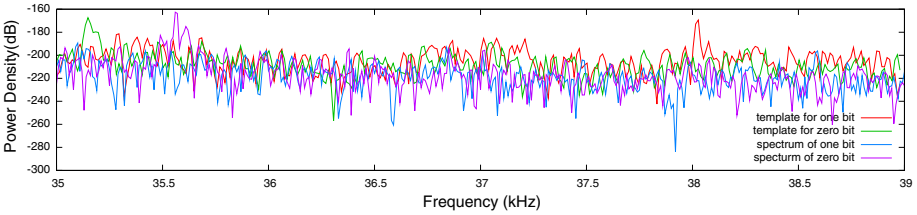


Fig. 22. Acoustic frequency spectra of the second modular exponentiation, when attacking a 0-valued bit and a 1-valued bit of q, both in the most significant limb. The corresponding templates T^0 and T^1 are also shown. Note that the frequency of the peak at 35.2 kHz in the template for 0-valued bits (*green*) does not match the frequency of the peak at 35.6 kHz in the signal for 0-valued bit (*purple*). Similar mismatch (albeit harder to spot) exists between the peak at 38 kHz in the template for 1-valued bits (*red*) and the signal for 1-valued bit (*blue*). Thus, we need to align the template and signal spectra.

Fifth 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 low values of d_b to mean a good match between s and T^b. Later, we shall classify s as corresponding to a 0-valued bit if $d_0 < d_1$ and corresponding to a 1-valued 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 ith bin as $|T^b_i - s_i|^3$ and then obtain d_b by summing all the differences (the exponent 3 was chosen empirically).

Unfortunately, as shown in Sect. 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

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

Sixth Stage: Classification If 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 0-valued bit. Otherwise, if 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 1-valued bit. Finally, in case that s is close to both T^0 and T^1, we repeat the above stages (each time asking for a new decryption of the same ciphertext) until a suitable classification can be achieved.

Seventh 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 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 0-valued bit as a 1-valued bit 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 1-Valued Bit as a 0-Valued Bit 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 than q. Thus, the value of c after a reduction modulo q will always be a 64 limb number whose first $j - 1$ bits are the same as q, whose jth bit is zero, and whose remaining bits are ones. Note that this value of c will have the same side channel leakage as if $q_i = 1$. Thus, our attack will report all subsequent bits of q as being 1-valued regardless their actual value. This situation is easily detectable, since q is a random prime that is unlikely to contain long sequences of 1-valued bits. Thus, when such a sequence (of say 20 bits) is detected, it probably means that a mistake was made 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 0-Valued Bit as a 1-Valued Bit 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, after a reduction modulo q, c will always consist of $64 - \lfloor j/32 \rfloor$ random-looking limbs. Note 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 being 0-valued regardless of 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. In Sect. 9.1, we investigate how a different implementation of the square-and-multiply algorithm affects our attacks, and in Sect. 9.2, we investigate low-bandwidth leakage in the GnuPG implementation of ElGamal decryption [21].

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 (versions 1.4.14 and 1.4.15) were released. Version 1.4.14 introduced changes in the 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 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 [51] that check for the presence of multiplication code in the CPU's code cache in order to recover the bits of d.

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
```

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, and thus, our attack is not affected by this change. However, as can be seen in Fig. 23, this code change in GnuPG *does* affect the leakage spectra of GnuPG when attacking the bits of q that are set to 1. Note 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 Sect. 11 for a discussion about effective and non-effective countermeasures.

9.2. *ElGamal key Distinguishability*

Another popular public key encryption scheme is ElGamal encryption [21]. 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 $\mathsf{pk} = (p, \alpha, \alpha^a)$,

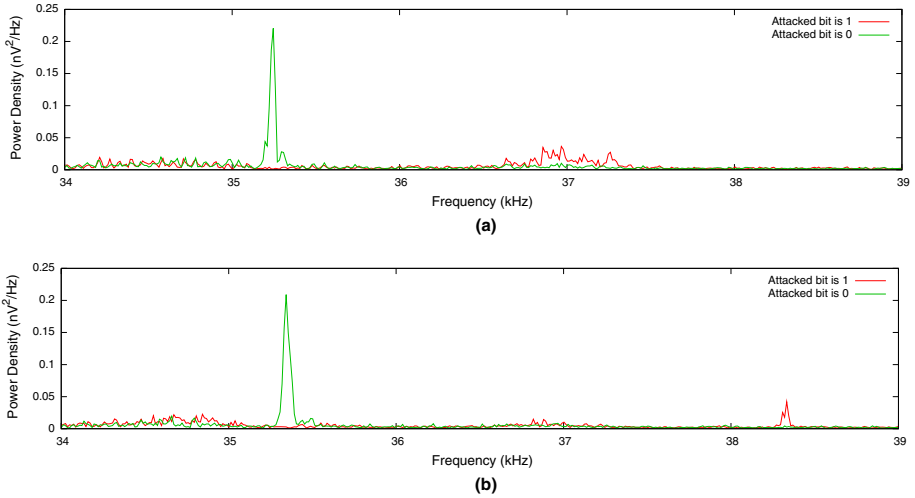


Fig. 23. Acoustic measurement frequency spectra of the second modular exponentiation during our attack using different versions of GnuPG. **a** GnuPG version 1.4.13 and **b** GnuPG version 1.4.14.

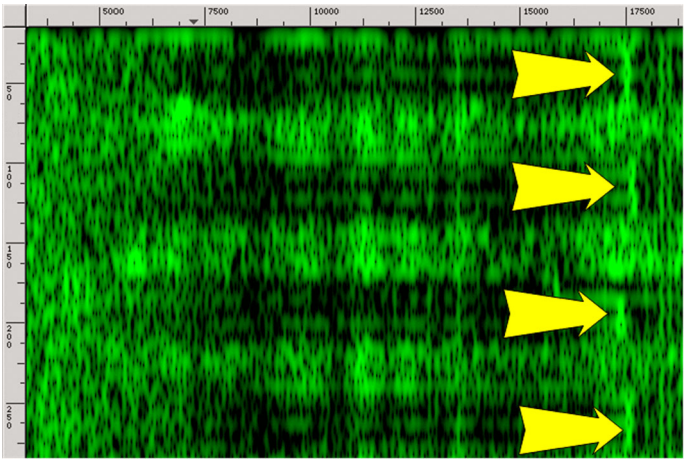


Fig. 24. Acoustic signature (0.25 s 2.5−17.5 kHz) of four ElGamal decryptions using the same message and randomly generated 3072 bit keys.

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 24 presents an FFT spectrogram of acoustic emanations recorded during execution of four ElGamal decryptions, using the same message and randomly generated 3072 bit keys. As in the RSA case, the four decryptions are clearly visible. Moreover, while the leakage is not as prominent as in the RSA case, the four different keys can be easily distinguished.

10. Comparison with Timing Attacks

In this section, we compare our techniques with the timing attacks of Boneh and Brumley [4].

Cryptanalytic Approach While both our attack and the attacks of [4] are chosen-ciphertext attacks that leak the secret key one bit at a time, their effect on the RSA decryption code, as well as their underlying principles, is fundamentally different. The work of [4] utilizes OpenSSL's [44] use of Montgomery reduction [40], and its high-level choice of using regular versus Karatsuba multiplication, in order to leak the current secret key bit. But neither of the effects exploited by [4] is applicable to GnuPG. First, GnuPG does not use the Montgomery form to represent its internal values; it uses grade-school long division to perform the modular reduction operation. Second, GnuPG always uses Karatsuba multiplication (for values longer than KARATSUBA_THRESHOLD limbs, i.e., 512 bits, whereas current standards of security call for RSA key size of at least 2048–4096 bits [5]). Thus, GnuPG's high-level code does not exhibit exploitable behavior. Instead, we dig into the low-level multiplication code of GnuPG and heavily exploit its internal structure, as detailed in the preceding sections.

Attack Scenarios In many cases, such as encrypted email, timing information cannot be observed by nominal communication channels, since there is no immediate response sent to the attacker; yet, the acoustic attack is still potentially applicable, as discussed in Sect. 1.2.

Timing Attacks via the Acoustic Channel One may try to conduct a hybrid attack, using the acoustic channel to measure just the total running time of the decryption. The problem is that, with this approach, the attacker is forced to identify the "end of decryption" event with high temporal accuracy, which forces him to consider few points in the measured time series. But recall that changes in the ongoing computation are acoustically discernible mainly as subtle frequency shifts, which (by the uncertainty principle) require many sample points to identify. Moreover, reliance on few points makes the result very sensitive to noise. By contrast, the acoustic attack we present uses the measurements acquired throughout the decryption, which yields excellent frequency resolution (hence applicability to more targets) and robustness against noise (hence greater range).

Countermeasures It is easy to defend against the timing attack of [4] by adding "padding" delay cycles at the end of the RSA decryption operation, to hide the decryption timing. This is not effective against our attacks, which can observe the decryption in real time and discard the padding; thus, the decryption algorithm itself should be changed (see Sect. 11 for further discussion).

Timing in GnuPG We now examine the possibility of extracting the secret key using our cryptanalytic approach, under the assumption that the attacker somehow managed to obtain an accurate measurement of RSA decryption time. In order to obtain the execution time measurements, we instrumented the code of GnuPG to measure execution time by

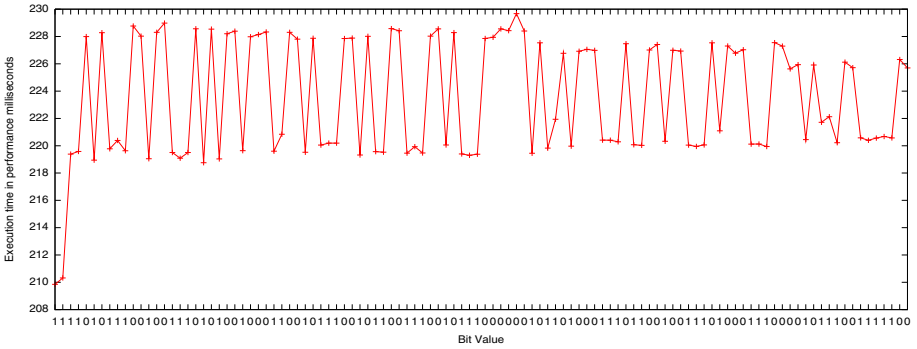


Fig. 25. Execution time of RSA decryptions using our attack and a randomly generated key measured on a Lenovo ThinkPad T61 using the Windows performance counter. Each point in the graph is the duration of a (single) RSA decryption while attacking the given bit of q.

using the Windows API call QueryPerformanceCounter on the target machine, which has a 1ms resolution, and averaged over 20 samples.[13]

Figure 25 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 ith 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 0-valued bits require more execution time compared to 1-valued bits.

We observe a strong correlation between the inverse of decryption time and some of the strong components in the leakage spectrum. We thus conjecture that repetitive operations in the exponentiation (e.g., the frequency of its outermost loop) are directly modulating the observed leakage.

11. Mitigation

Acoustic Shielding Proper acoustic shielding, using acoustic absorbers and soundproof enclosures, would attenuate the signals, thereby requiring the attacker to use more expensive equipment, more demanding microphone positioning, or longer attack time. However, such shielding raises design and manufacturing costs, especially due to the necessity of air circulation for cooling. In particular, laptop cooling fan vent holes are typically a prolific source of emanations and cannot be easily blocked. We further 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.

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

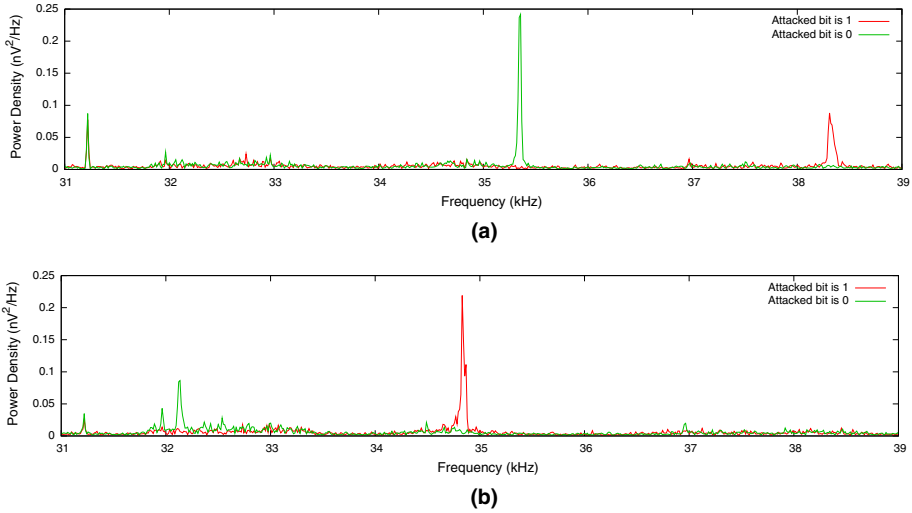


Fig. 26. Acoustic measurement frequency spectra of the second modular exponentiation with and without constant load. **a** Without background load and **b** with background load.

Noisy Environment One may expect that placing the machine in a noisy environment 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 easily be filtered out during the data acquisition process using a suitable high-pass filter (as we did in our experiments). Also, the signal would be observed 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 loads on the machine (see below). Thus, a carefully designed acoustic noise generator would be required to mask the leakage.

Parallel Software Load A natural candidate countermeasure is to induce key-independent load on the CPU, in the hope that the other computations performed in parallel will somehow mask the leakage of the decryption operation. Figure 26 demonstrates the difference in the frequency spectra of the acoustic signature of the second modular exponentiation during our attack that result from applying a background load comprised of an infinite loop of ADD instructions being performed in parallel. As can be seen from Fig. 26, background load on 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 the microphone capsule that can be used to perform the attack (see Sect. 5.4).

EM Shielding Conductive, grounded shields, ideally Faraday cages, are used to attenuate electromagnetic radiation for TEMPEST security (see Sect. 1.3) and moreover for electromagnetic compatibility (EMC). One may wonder about their effect on the acoustic channel. To facilitate air flow for convective cooling, such shields typically contain vents,

Fig. 27. Examples of electromagnetic shielding with perforations and meshes. 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").

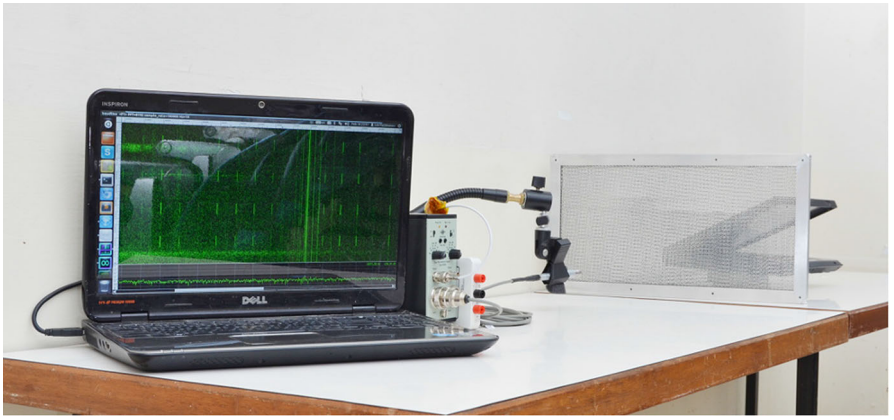


Fig. 28. Portable measurement setup recording a laptop through a grounded EMI-shielded vent panel (1/2"-thick cross-cell double-honeycomb mesh, Holland Shielding Systems Honeycomb Ventilation 9500). Removing the vent does not significantly affect the acoustic signal, but covering it with cardboard severely attenuates the signal.

built of meshes or dense perforations (see Fig. 27). These can be nearly transparent to acoustic emanations, as we verified on a high-grade EM mesh shield (see Fig. 28, and note that covering the mesh with cardboard severely attenuated the signal). Conveniently for an attacker, these vents are typically located close to the power regulation circuitry, which emits the acoustic signal.

Cipher Text Randomization One countermeasure that *is* effective in stopping our attack is RSA ciphertext randomization [36]. 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

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

In this case, the value sent to the modular exponentiation routine is completely random, thus preventing our chosen-ciphertext attack. This countermeasure has a non-trivial cost,

since an additional modular exponentiation is required (albeit with the exponent e, typically small). Nor does it affect key distinguishability, which (we observed empirically) is independent of the ciphertext.

Modulus Randomization Another standard side channel countermeasure, which may help prevent both key distinguishing and key extraction, is to randomize the modulus during each exponentiation [33]. 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 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 still allows key distinguishing).

Acknowledgements

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 thank Werner Koch, lead developer of GnuPG, for the prompt response to our disclosure and the productive collaboration in adding suitable countermeasures. We are indebted to Pankaj Rohatgi for inspiring the origin of this research; to Nir Yaniv for audio recording advice and use of the Nir Space Station studio; to National Instruments Israel for donating PCI-6052E and MyDAQ hardware; and to the anonymous reviewers for their valuable suggestions. This work was sponsored by the Check Point Institute for Information Security; by European Union's Tenth Framework Programme (FP10/2010–2016) under Grant Agreement No. 259426 ERC-CaC, by the Leona M. & Harry B. Helmsley Charitable Trust; 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."

Appendix: The Lab-Grade setup

In this section, we provide further details on our lab-grade setup, introduced in Sect. 2.1. Since the signals we measure are both weak in power and have relatively high frequency,

Table 1. Comparison of microphone capsules used.

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

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

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 (see [10]). The capsule is assembled on a *preamplifier* (discussed below).

After testing a variety of Brüel&Kjær microphone capsules, we identified three 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 Fig. 1; Table 1). 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 not having a flat frequency response, the 4190 capsule still has better sensitivity than the 4191 at 40 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 internal noise of the 4145 is lower than that of the 4190. Thus, the 4145 has a better signal-to-noise ratio than the 4190, at the expense 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. The Brüel&Kjær 2804 power supply also outputs the microphone's signal, which we then amplified (as described next).

Amplification The signal produced by the capsule, through the unity-gain preamplifier, is very weak (in 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 laboratory 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 a data acquisition device (DAQ): 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.

References

[1] D. Asonov, R. Agrawal, Keyboard acoustic emanations, in *IEEE Symposium on Security and Privacy 2004* (IEEE Computer Society, 2004), pp. 3–11

[2] D. Agrawal, B. Archambeault, J.R. Rao, P. Rohatgi, The EM side-channel(s), in *Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2002* (Springer, 2002), pp. 29–45

[15] The Brüel&Kjær 49391/4" capsule can also be connected to the 26701/4" 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, which also combine a built-in power supply. However, these amplifiers have a built-in 100 kHz low-pass filter that prevents the measurement of signals in the 100–350 kHz range (recall that these signals are already particularly weak due to poor performance of the 4939 capsule in these frequencies). Moreover, Nexus amplifiers have noise density of $13.4\,\mathrm{nV}/\sqrt{\mathrm{Hz}}$, which is worse then the ZPUL-30P.

[3] R.J. Anderson, *Security Engineering—A Guide to Building Dependable Distributed Systems (2nd ed.)* (Wiley, 2008)
[4] D. Brumley, D. Boneh. Remote timing attacks are practical. *Comput. Netw.***48**(5), 701–716 (2005)
[5] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, *NIST SP 800-57: Recommendation for Key Management—Part 1: General* (2012)
[6] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, C. Sporleder. Acoustic side-channel attacks on printers, in *USENIX Security Symposium 2010* (USENIX Association, 2010), pp. 307–322.
[7] N. Borisov, I. Goldberg, D. Wagner, Intercepting mobile communications: the insecurity of 802.11, in *International Conference on Mobile computing and Networking MOBICOM 2011* (2001), pp. 180–189
[8] A. Bittau, M. Handley, J. Lackey, The final nail in WEP's coffin, in *IEEE Symposium on Security and Privacy 2006* (IEEE Computer Society, 2006), pp. 386–400.
[9] H.E. Bass, R.G. Keeton, Ultrasonic absorption in air at elevated temperatures. *J. Acoust. Soc. Am.***58**(1), 110–112 (1975)
[10] Brüel & Kjær, *Technical Documentation—Microphone Handbook*, vol. 1 (1996)
[11] B.B. Brumley, N. Tuveri, Remote timing attacks are still practical, in *ESORICS 2011* (Springer, 2011), pp. 355–371.
[12] Y. Berger, A. Wool, A. Yeredor, Dictionary attacks using keyboard acoustic emanations, in *ACM Conference on Computer and Communications Security* (ACM, 2006), pp. 245–254
[13] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, R. Thayer, *OpenPGP message format*. RFC 4880 (November 2007).
[14] O. Choudary, M.G. Kuhn, Efficient template attacks, in *Smart Card Research and Advanced Applications (CARDIS) 2013* (Springer, 2013), pp. 253–270
[15] S.S. Clark, H.A. Mustafa, B. Ransford, J. Sorber, K. Fu, W. Xu, Current events: identifying webpages by tapping the electrical outlet, in *ESORICS 2013* (Springer, 2013), pp. 700–717.
[16] Committee on National Security Systems, Index of national security systems issuances. https://www.cnss.gov/CNSS/issuances/Issuances.cfm (September 2013)
[17] D. Coppersmith, Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptol.***10**(4), 233–260 (1997)
[18] S. Chari, J.R. Rao, P. Rohatgi, Template attacks, in *Cryptographic Hardware and Embedded Systems (CHES) 2002* (Springer, 2002), pp. 13–28
[19] S.S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, W. Xu, K. Fu, WattsUpDoc: power side channels to nonintrusively discover untargeted malware on embedded medical devices, in *USENIX Workshop on Health Information Technologies (HealthTech) 2013* (USENIX Association, 2013)
[20] L.B. Evans, H.E. Bass, Tables of absorption and velocity of sound in still air at 68° F, in *Report WR72-2* (Wyle Laboratories, 1972)
[21] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory***31**(4), 469–472 (1985)
[22] The Enigmail Project, *Enigmail: A simple interface for OpenPGP email security*. https://www.enigmail.net
[23] M. Elkins, D. Del Torto, R. Levien, T. Roessler. *MIME security with OpenPGP* (RFC 3156, 2001). http://www.ietf.org/rfc/rfc3156.txt
[24] Genesis 27:5
[25] K. Gandolfi, C. Mourtel, F. Olivier. Electromagnetic analysis: concrete results, in *Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2001* (Springer, 2001), pp. 251–261
[26] GNU multiple precision arithmetic library. http://gmplib.org/
[27] GNU Privacy Guard. https://www.gnupg.org
[28] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, Stealing keys from PCs using a radio: cheap electromagnetic attacks on windowed exponentiation, in *Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2015*. To appear. Extended version: Cryptology ePrint Archive, Report 2015/170 (2015), pp. 207–228.
[29] D. Genkin, I. Pipman, E. Tromer, Get your hands off my laptop: physical side-channel key-extraction attacks on PCs, in *Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2014*. See [30] for an extended version (Springer, 2014), pp. 242–260
[30] D. Genkin, I. Pipman, E. Tromer, Get your hands off my laptop: physical side-channel key-extraction attacks on PCs (extended version). *J. Cryptogr. Eng.***5**(2), 95–112 (2015). Extended version of [29]

[31] D. Genkin, A. Shamir, E. Tromer, RSA key extraction via low-bandwidth acoustic cryptanalysis, in *CRYPTO 2014*, vol. 1 (Springer, 2014), pp. 444–461
[32] T. Halevi, N. Saxena, On pairing constrained wireless devices based on secrecy of auxiliary channels: the case of acoustic eavesdropping, in *ACM Conference on Computer and Communications Security CCS 2010* (ACM, 2010), pp. 97–108
[33] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *CRYPTO 1999* (Springer, 1999), pp. 388–397
[34] P. Kocher, J. Jaffe, B. Jun, P. Rohatgi, Introduction to differential power analysis. *J. Cryptogr. Eng.***1**(1), 5–27 (2011)
[35] A. Karatsuba, Y. Ofman, Multiplication of many-digital numbers by automatic computers. *Proc. USSR Acad. Sci.***145**, 293–294 (1962)
[36] P.C. Kocher, Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems, in *CRYPTO 1996* (Springer, 1996), pp. 104–113
[37] M. LeMay, J. Tan, Acoustic surveillance of physically unmodified PCs, in *Security and Management 2006* (CSREA Press, 2006), pp. 328–334
[38] X. Lurton, *An Introduction to Underwater Acoustics: Principles and Applications*. Geophysical Sciences Series (Springer, 2002)
[39] MITRE. Common vulnerabilities and exposures list, entry CVE-2013-4576, 2013. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4576
[40] P.L. Montgomery, Modular multiplication without trial division. *Math. Comput.***44**(170), 519–521 (1985)
[41] S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks—Revealing the Secrets of Smart Cards* (Springer, 2007)
[42] National Security Agency, *NACSIM 5000: TEMPEST Fundamentals, February 1982*. http://cryptome.org/jya/nacsim-5000/nacsim-5000.htm
[43] National Institute of Standards and Technology, *FIPS 140-3: Draft Security Requirements for Cryptographic Modules (Revised Draft)* (2009)
[44] OpenSSL. http://www.openssl.org
[45] J.-J. Quisquater, D. Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards, in *E-smart 2001* (2001), pp. 200–210
[46] R.L. Rivest, A. Shamir, Efficient factoring based on partial information, in *Eurocrypt 1985* (Springer, 1985), pp. 31–34
[47] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM***21**(2), 120–126 (1978)
[48] A. Shamir, E. Tromer, Acoustic cryptanalysis: on nosy people and noisy machines, 2004. Eurocrypt rump session. http://cs.tau.ac.il/~tromer/acoustic/ec04rump
[49] D.X. Song, D. Wagner, X. Tian, Timing analysis of keystrokes and timing attacks on SSH, in *USENIX Security Symposium 2001* (USENIX Association, 2001)
[50] P. Wright. *Spycatcher* (Viking Penguin, 1987)
[51] Y. Yarom, K. Falkner, FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack, in *USENIX Security Symposium 2014* (USENIX Association, 2014), pp. 719–732
[52] L. Zhuang, F. Zhou, J.D. Tygar, Keyboard acoustic emanations revisited, in *ACM Conference on Computer and Communications Security* (ACM, 2005), pp. 373–382