RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*

Daniel Genkin[1], Adi Shamir[2], and Eran Tromer[3]

[1] Technion and Tel Aviv University
danielg3@cs.technion.ac.il
[2] Weizmann Institute of Science
adi.shamir@weizmann.ac.il
[3] Tel Aviv University
tromer@cs.tau.ac.il

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: as we show in this paper, they can leak the key used in cryptographic operations. This is surprising, since the acoustic information has very low bandwidth (under 20 kHz using common microphones, and a few hundred kHz using ultrasound microphones), which is many orders of magnitude below the GHz-scale clock rates of the attacked computers. We describe a new *acoustic cryptanalysis* attack which can extract full 4096-bit RSA keys from the popular GnuPG software, 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.

1 Introduction

1.1 Overview

Cryptanalytic side-channel attacks target implementations of cryptographic algorithms which, while perhaps secure at the mathematical level, inadvertently leak secret information through indirect channels: variations in power consumption, electromagnetic emanations, timing variations, contention for CPU resources such as caches, and so forth (see [And08] for a survey). Acoustic emanations are another potential channel, but so far it was used only in order to eavesdrop to slow electromechanical components such as keyboards and printers [AA04, ZZT05, BDG$^+$10].

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

* The authors thank Lev Pachmanov for programming and experiments support.

draw according to the operations they execute, but in a very coarse way due to the low bandwidth, which does not enable the attacker to "hear" individual instructions executed on a multi-GHz computer.[4]

The first indication that acoustic emanation from electronic computers is of cryptanalytic interest was by Shamir and Tromer [ST04], 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 stated that while "acoustic effects have been suggested as possible side channel, the quality of the resulting measurements is likely to be low" [KJJR11].

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) [Gpg], which is a popular cross-platform open-source implementation of the OpenPGP standard. We first verified 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 computer, or by a sensitive microphone from a distance of 10 meters. In a nutshell, our attack relies on crafting special chosen RSA ciphertexts that cause numerical cancellations deep inside GnuPG's modular exponentiation algorithm. This causes the special value zero to appear frequently in the innermost loop of the algorithm, where it affects control flow. A single iteration of that loop is much too fast for direct acoustic observation. However, 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 not only causes key-dependent side-channel leakage in GnuPG's RSA implementation, but moreover utilizes the GnuPG's *own code* in order to amplify the aforementioned leakage.

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 or WEP. To break GnuPG, we used a new attack vector based on Enigmail [Eni], 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 and PGP/MIME standards. For "new e-mail" notifications, 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), and

[4] Above a few hundred kHz, sound propagation in the air has a very short range, due to non-linear attenuation and distortion effects (viscosity, relaxation and diffusion at the molecular level). Most microphones are limited to about 20 kHz.

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 acoustic attacks can be applied in a large variety of situations. For example, any electronic device which has an internal microphone can be used to spy on itself, using an unprivileged application which has access to the microphone listen to the sounds made by a privileged security application (even when the two applications run in two different virtual machines). On a cellular phone, the whole attack can be packaged into a simple software "app" which can close the adaptive chosen-ciphertext loop in real time, using the phone's signal processing capabilities and wireless data connectivity. The phone can then be used to spy on a nearby laptop computer, for example, during an hour-long face-to-face business meeting between two persons who place their gadgets on the same table. In another scenario, the attacker can place in advance a hidden acoustic bug near the likely location of the attacked laptop, e.g., in a lecture podium used by a visiting speaker, or in a hotel desk.

Physical countermeasures. Many of the physical side-channel countermeasures used in highly sensitive applications, such as air gaps, Faraday cages, and power supply filters, provide no protection against acoustic leakage. In particular, Faraday cages containing computers require ventilation, which is typically provided by means of vents covered with perforated sheet metal or metal honeycomb. These are very effective at attenuating compromising electromagnetic radiation ("TEMPEST"), but — as we empirically verified — are nearly transparent to acoustic emanations. This can make the acoustic attack one of the few remaining options when the target is heavily protected by expensive shielding that blocks all the standard sources of electronic emanations.

Current status. Our attacks can be applied to all recent versions of the GnuPG 1.x series (up to the latest, 1.4.15, released on 15 Oct. 2013), including the side-channel mitigation introduced in GnuPG 1.4.14 (which ironically helps our attack by amplifying the aforementioned effect of the zero value in the innermost loop). After disclosing our detailed attack to the GnuPG developers and main distributors, we suggested several suitable countermeasures, and verified that the new versions of GnuPG 1.x and of libgcrypt (which underlies GnuPG 2.x), released concurrently with this paper's first public posting, correctly implement our countermeasures and resist the current key-extraction attack.

1.2 Related work

Auditory eavesdropping on human conversations is a common practice, first published several millenia ago [Gen]. Analysis of sound emanations from mechanical devices is a newer affair, with precedents in military context such as identification of vehicles (e.g., submarines) via the sound signature of their engine or propeller. Wright [Wri87, pp. 103–107] provides an account of MI5 and GCHQ using a phone tap to eavesdrop on an electromechanical Hagelin cipher machine, by counting the clicks during the rotors' secret-setting procedure. Keystroke timing patterns (which can be acquired acoustically) are known to be a way to identify users, and more recently, also to leak information about the typed text (see [SWT01] and the references therein). Later work by Asonov and Agrawal [AA04], improved by Zhuang et al. [ZZT05], Berger et al. [BWY06], and by Halevi and Saxena [HS10], shows that keys can also be distinguished individually by their sound, due to minute differences in mechanical properties such as their position on a slightly vibrating printed circuit board. Backes et al. [BDG$^+$10] show that the sound produced by dot matrix printers can be used to recover printed English text.

NSA's partially-declassified "TEMPEST Fundamentals" [Nat82] mentions acoustic emanations, but defines them narrowly as "emanations in the form of free-space acoustical energy produced by the operation of a *[sic]* purely mechanical or electromechanical device equipment". Other official publications, such as the latest FIPS 140-3 draft [Nat09], describe a large variety of side channel attacks, but do not mention acoustic emanations. One is thus led to conclude that this attack vector was not believed to pose a threat to nonmechanical systems.

2 Observing acoustic leakage

In this section we show that it is possible, using acoustic emanations, to glean information about the CPU operations of various laptop computers. We show that it is possible for an attacker to learn the instructions executed by the target computer, solely by observing its acoustic emanations using a microphone. Moreover, we show a rudimentary cryptographic side channel, namely distinguishing GnuPG RSA keys (which will be further developed in the subsequent sections).

Lab-grade experimental setup. For the experiments in this section, meant to best characterize the emitted signal, we used carefully-optimized lab-grade equipment with very high sensitivity and frequency range. Specifically, we measured the acoustic emanations using a Brüel&Kjær 4190 and 4939 microphone capsules, connected to a Brüel&Kjær 2669 pre-amplifier, powered by a Brüel&Kjær 2804 microphone power supply. The signal is low-pass filtered at 1.9 MHz, amplified using a (customized) Mini-Circuits ZPUL-30P amplifier, and then high-pass filtered at 10 kHz. The result is digitized using an National Instruments PXIe 6356. For further details, see the extended version of this paper [GST13].

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 physical source of the relevant emanations is difficult to characterize precisely, since it varies between

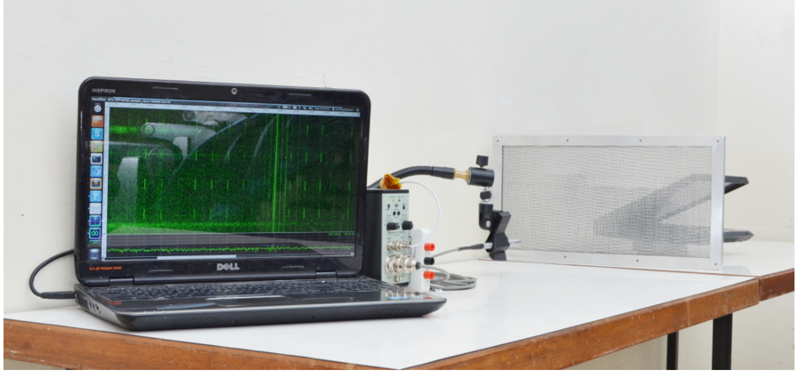


Fig. 1. Portable measurement setup recording a laptop through an EMI-shielded vent panel (1/2”-thick cross-cell double-honeycomb mesh, Holland Shielding Systems Honeycomb Ventilation 9500).

target machines, and is typically located in the hard-to-reach innards. Still, experimentation with the microphone placement invariably located the strongest useful signals in the vicinity of the on-board voltage regulation circuit supporting the CPU. Indeed, modern CPUs change their power consumption dramatically depending on software load, and we conjecture that this affects, and modulates, the dynamics of the pulse-width-modulation-based voltage regulator. More remote stages of the power supply (e.g., laptops’ external AC-DC “power brick” adapter) sometimes exhibit software-dependent acoustic leakage as well.

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 on-board power supply located on the laptop’s motherboard, but without intrusion or disassembly. Luckily, laptop computers have a substantial cooling system for heat dissipation, with a fan that requires large exhaust holes. In addition, there are numerous holes and gaps for ports such as USB, Express Card slot, and Ethernet port. Each of the above ports has proven useful on some computers.

Acoustic or EM? To ascertain that the obtained signal is truly acoustic rather than electromagnetic interference picked up by the microphone, we placed a sheet of non-conductive sound-absorbing material (e.g., cork or thick cloth) in front of the microphone. This always resulted in a severe attenuation of the recorded signals. Thus, we conclude that the measured signals are indeed acoustic.

EM shielding. As discussed in Section 1.1, standard TEMPEST electromagnetic shielding, such as metal meshes and perforated metal, can be nearly-transparent to acoustic emanations, as we verified on a professionally-produced EMI mesh shield (see Figure 1, and note that as before, covering the mesh with cardboard severely attenuated the signal).

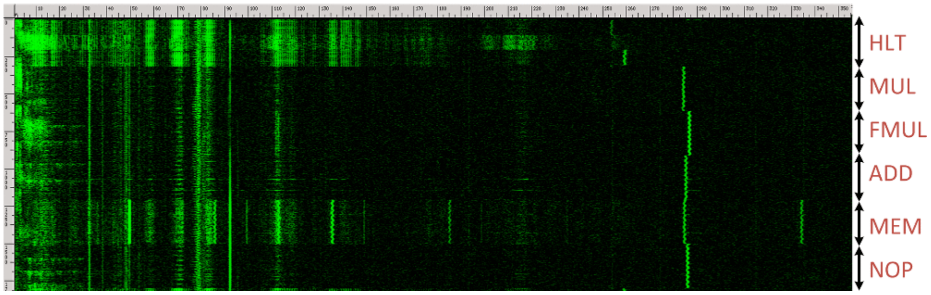


Fig. 2. Acoustic measurement frequency spectrogram of a recording of different CPU operations using the Brüel&Kjær 4939 microphone capsule. The horizontal axis is frequency (0–350 kHz), the vertical axis is time (1.7 sec), and intensity is proportional to the instantaneous energy in that frequency band.

2.1 Distinguishing various CPU operations

We begin our analysis of acoustic 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 to distinguish between some CPU operations on almost all the machines we tested, some machines have a particularly rich leakage spectrum. Figure 2 shows a recording of the Evo N200 laptop while executing our program using the Brüel&Kjær 4939 high frequency microphone capsule. As can be seen in Figure 2, the leakage of the Evo N200 is present all over the 0–350 kHz spectrum. Moreover, different types of operations can be easily distinguished. Similar types of leakage (although less prominent) was detected on numerous other machines as well.

2.2 GnuPG key distinguishability

The results in Section 2.1 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) [Gpg], a popular open source implementation of the OpenPGP standard available on all major operating systems.[5]

The sound of a single RSA secret key. Figure 3 depicts the spectrogram of five RSA signing operations in sequence, using the same message and five randomly generated 4096-bit keys. Each signing operation is preceded by a short delay, during which the CPU is in a sleep state. Figure 3 contains several interesting effects. The delays are manifested as bright horizontal strips. Between these strips, the five signing operations can be clearly distinguished. Halfway through each signing operation there is a transition at several frequency bands (marked by yellow arrows), corresponding to the transition between exponentiation modulo the secret p to exponentiation modulo the secret q, in the RSA decryption implementation of GnuPG, which uses the Chinese Remainder Theorem.

Distinguishing between RSA secret keys. Having observed that the acoustic signature of modular integer exponentiation depends on the modulus involved. Thus, one may expect different keys to cause different sounds. This is indeed the case, as demonstrated in Figure 3. It is readily observed that each signature (and in fact, each exponentiation using modulus p or q) has a unique spectral signature. This ability to distinguish keys is of interest in traffic-analysis applications.[6] It is likewise possible to distinguish between algorithms, between different implementations of an algorithm, and between different computers (even of the same model) running the same algorithm. Again, this effect is consistent and reproducible (in various frequency ranges) on various machines and manufacturers. Finally, for the case of ElGamal decryption, various secret keys can also be acoustically distinguished.

3 Overview of GnuPG RSA key extraction

In this section, we present our acoustic RSA key extraction attack, and discuss its performance (e.g., extracting a whole RSA key within about one hour using just the acoustic emanations from the target machine). For concreteness, in the following we consider GnuPG 1.4.14 and key size of 4096 bit (i.e., 2048 bit primes p, q), which should be secure beyond the year 2031 [BBB$^+$12].

3.1 GnuPG's modular exponentiation routine

GnuPG's mathematical library. Algebraic operations on large integers (which are much larger than the machine's word size) are implemented using

[5] 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 following the work of [YF13]). 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.

[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.

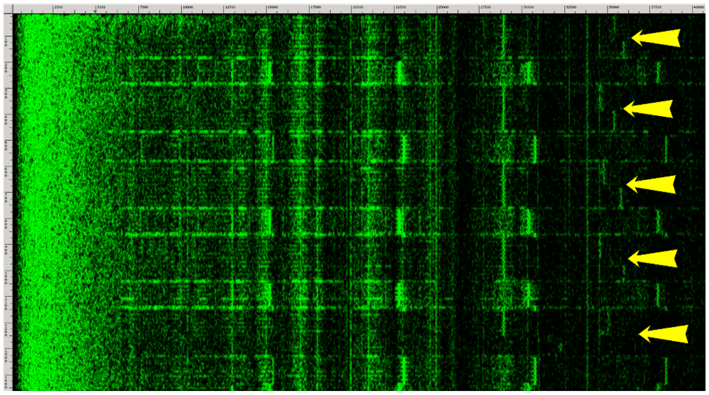


Fig. 3. Acoustic signature (1.4 sec, 0–40 kHz) of five GnuPG RSA signatures executed on a Lenovo ThinkPad T61. Recorded using the lab-grade setup and the Brüel&Kjær 4190 microphone capsule. The transitions between p and q are marked by yellow arrows.

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

We now review the modular exponentiation used in GnuPG's implementation of RSA (as introduced in GnuPG v1.4.14). 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 $\lceil t/32 \rceil$. Understanding this top-level exponentiation routine suffices for the high-level description of our attack. For details about GnuPG's underlying multiplication routines, necessary for understanding the attack's success, see Section 4.

Since GnuPG represents large numbers in arrays of 32 bit limbs, GnuPG optimizes the number of modulo reductions by always checking (at the end of every multiplication and squaring) whether the number of limbs in the partially computed result exceeds the number of limbs in the modulus. If so, a modular reduction operation is performed. If not, reduction will not decrease the limb count, and thus is not performed. This measurement of size in terms of limbs, as opposed to bits, slightly complicates our attack. Note that due to a recently introduced side-channel mitigation technique (following the work of [YF13]), this code always performs the multiplications, regardless of the bits of d.

3.2 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

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

```
Input: Three integers $c$, $d$ and $q$ in binary representation such that $d = d_n \cdots d_1$.
Output: $m = c^d \mod q$.
 1: procedure MODULAR_EXPONENTIATION($c, d, q$)
 2:     if SIZE_IN_LIMBS($c$) $>$ SIZE_IN_LIMBS($q$) then
 3:         $c \leftarrow c \mod q$
 4:     $m \leftarrow 1$
 5:     for $i \leftarrow n$ downto $1$ do
 6:         $m \leftarrow m^2$                          ▷ Karatsuba or grade-school squaring
 7:         if SIZE_IN_LIMBS($m$) $>$ SIZE_IN_LIMBS($q$) then
 8:             $m \leftarrow m \mod q$
 9:         $t \leftarrow m \cdot c$                 ▷ Karatsuba or grade-school multiplication
10:         if SIZE_IN_LIMBS($t$) $>$ SIZE_IN_LIMBS($q$) then
11:             $t \leftarrow t \mod q$
12:             if $d_i = 1$ then
13:                 $m \leftarrow t$
14:         return $m$
15:     end procedure
```

timing attack [BB05]). For each bit q_i of q, starting from the most significant bit position ($i = 2048$), we assume that key bits $q_{2048} \cdots q_{i+1}$ were correctly recovered, and check the two hypotheses about q_i. Eventually, we learn all of q and thus recover the factorization of n. Note that after recovering the top half the bits of q, it is possible to use Coppersmith's attack [Cop97] (following Rivest and Shamir [RS85]) to recover the remaining bits, or to continue extracting them using the side channel.[7]

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

GnuPG always generates RSA keys such that the most significant bit of q is set, i.e., $q_{2048} = 1$. Assume that we have already recovered the topmost $i - 1$ bits of q, and let $g^{i,1}$ be the 2048 bit ciphertext whose topmost $i - 1$ bits are the same as those recovered from q, whose i-th bit is 0, and whose remaining (low) bits are 1. Consider the RSA decryption of $g^{i,1}$. Two cases are possible, 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 of 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 routine in line 9.

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

- $\boldsymbol{q_i = 0}$**.** Then $q \leq g^{i,1}$. Thus, when $g^{i,1}$ is passed as the variable c to Algorithm 1, the modular reduction of c in line 3 changes the value of c. Since c and q share the same topmost $2048 - i$ bits, 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 line 9.

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

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

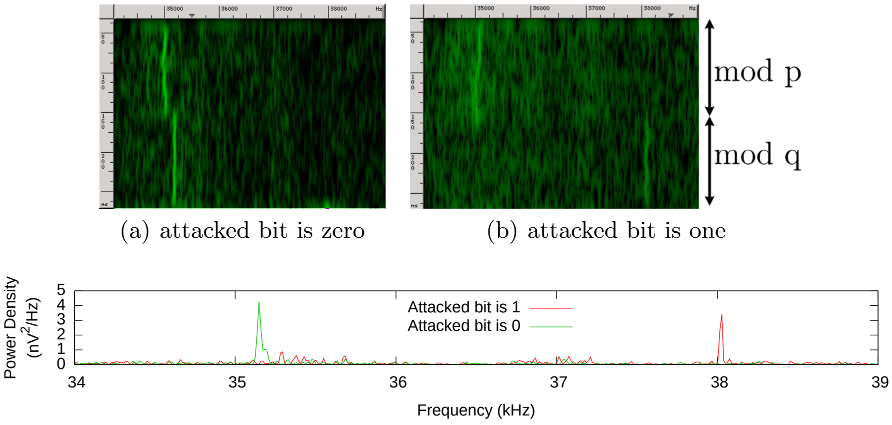
This can be solved in either of two ways. First, GnuPG's binary format parsing algorithm is willing to allocate space for leading zeros. Thus, one may just ask for a decryption of $g^{i,1}$ with additional limbs of leading zeros. GnuPG will pass on this suboptimal representation to the modular exponentiation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

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

3.3 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 Section 3.2, 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.

[8] 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.



(a) attacked bit is zero

(b) attacked bit is one

(c) Frequency spectra of the second modular exponentiation

Fig. 4. Acoustic emanations of RSA decryption for various values of the attacked bit ($q_{2039} = 1$ and $q_{2038} = 0$).

Figure 4(a) shows a typical recording of RSA decryption when the value of the attacked bit of q is 0 and Figure 4(b) shows a recording of RSA decryption when the value of the attacked bit of q is 1. Several effects are shown in the figures. Recall that GnuPG first performs modular exponentiation using the secret prime p and then performs another modular exponentiation using the secret prime q. As in figure 3, the transition between p and q is clearly visible in Figures 4(a) and 4(b). Note, then, that the acoustic signatures of the modular exponentiation using the prime q (the second exponentiation) are quite different in Figures 4(a) and 4(b). This is the effect utilized to extract the bits of q.

The spectral signatures in Figure 4(c) were computed from the acoustic signatures of the second modular exponentiation in Figures 4(a) and 4(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.

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 the extended version [GST13] for details).

3.4 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

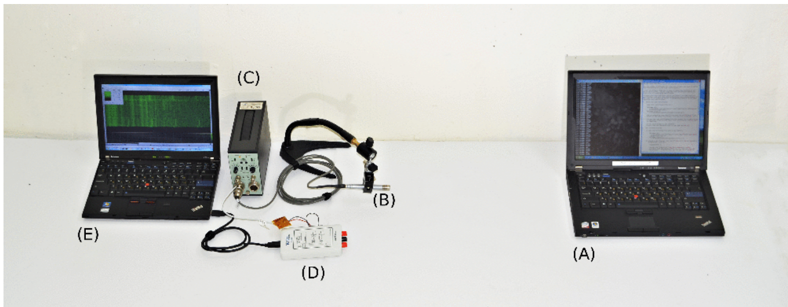
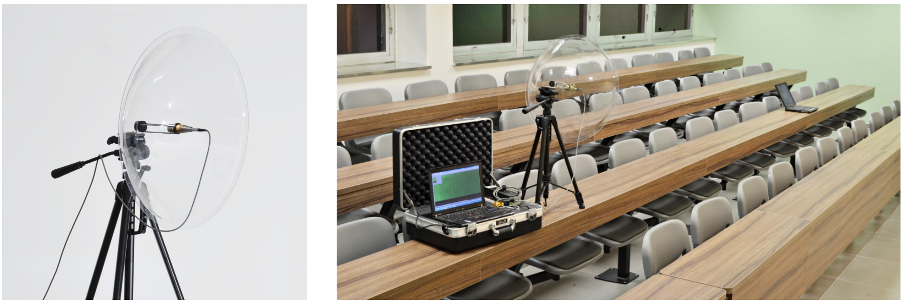


Fig. 5. Photograph of our portable setup. In this photograph (A) is a Lenovo ThinkPad T61 target, (B) is a Brüel&Kjær 4190 microphone capsule mounted on a Brüel&Kjær 2669 preamplifier held by a flexible arm, (C) is a Brüel&Kjær 5935 microphone power supply and amplifier, (D) is a National Instruments 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 this configuration, from a distance of 1 meter (see Section 3.4).

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 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 hour. Since this laptop's useful signal is at approximately 35 kHz, there is no need to use the full capabilities of the lab-grade setup (see Section 2), and instead we used a portable (briefcase-sized) setup consisting of a Brüel&Kjær 4190 microphone capsule connected (via a Brüel&Kjær 2669 pre-amplifier) to a Brüel&Kjær 5935 amplifier and microphone power supply, filtered, and digitized using a National Instruments MyDAQ device. Figure 5 depicts this portable setup.

Audible-frequency attack. Low frequency sound propagates in the air, and through obstacles, better than high frequency sound. Moreover, larger capsule diaphragms allow better sensitivity but reduce the frequency range. Indeed, some machines, such as the Lenovo ThinkPad X300 and ThinkPad T23, exhibit useful leakage at lower leakage frequency, 15–22 kHz (i.e., within audible range). This allows us to use the very sensitive Brüel&Kjær 4145 microphone capsule and extract the key from some machines at the range of around 1 meter. 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 Figure 6 for a similar setup.) With human-assisted signal processing, we extended the range up to 10 meters.



(a) Parabolic microphone (b) Distant attack

Fig. 6. (a) Brüel&Kjær 4145 microphone capsule and 2669 preamplifier, attached to a transparent parabolic reflector (56 cm diameter), on a tripod. (b) same, connected to the portable measurement setup, attacking a target laptop from a distance of 4 meters.

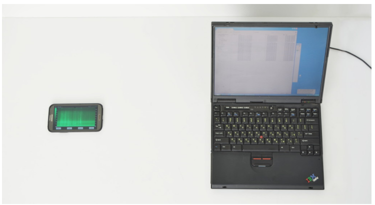


Fig. 7. Physical setup of a key recovery attack. A mobile phone (Samsung Note II) is placed 30 cm from a target laptop. The phone's internal microphone points towards the laptop's fan vents. Full key extraction is possible with this configuration and distance.

Mobile-phone attack. Lowering the leakage frequency also allows us to use lower quality microphones such as the ones in smartphones. We used several Android phones, with similar results: HTC Sensation, Samsung Galaxy S II and Samsung Galaxy Note II. Recorded using a custom Android app, accessing the internal microphone. 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 perform it on certain target computers, simply by placing the phone's microphone near to and directed towards the fan exhaust vent of the target while running the attack (see Figure 7). 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.

4 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 Section 3.3. We begin

Algorithm 2 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 $b_1 \leq 1$ then
 3:         if $b_1 = 1$ then
 4:             $p \leftarrow a$
 5:         else
 6:             $p \leftarrow 0$
 7:     else
 8:         $p \leftarrow$ MUL_BY_SINGLE_LIMB($a, b_1$)                      ▷ $p \leftarrow a \cdot b_1$
 9:     for $i \leftarrow 2$ to n do
10:         if $b_i \leq 1$ then
11:             if $b_i = 1$ then                          ▷ (and if $b_i = 0$ do nothing)
12:                 $p \leftarrow$ ADD_WITH_OFFSET($p, a, i$)              ▷ $p \leftarrow p + a \cdot 2^{32 \cdot i}$
13:         else
14:             $p \leftarrow$ MUL_AND_ADD_WITH_OFFSET($p, a, b_i, i$)     ▷ $p \leftarrow p + a \cdot b_i \cdot 2^{32 \cdot i}$
15:     return $p$
16: end procedure
```

by describing the multiplication algorithms used by GnuPG (Section 4.1) and then proceed (Sections 4.2 and 4.3) to describe the effects of our attack on the internal values computed during the execution of these algorithms.

4.1 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 [KO62]. The chosen combination of algorithms is based on the size of the operands, measured in whole limbs. 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 lets us leak q one bit at a time.

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 2).

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.

GnuPG's Karatsuba multiplication routine. The basic multiplication routine described above is invoked by both the modular exponentiation routine described in Section 3.1 and by the Karatsuba multiplication routine implementing a variant of the Karatsuba multiplication algorithm with some optimizations. GnuPG's variant of the Karatsuba multiplication algorithm relies on

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

where $u_{\mathsf{H}}, v_{\mathsf{H}}$ are the most significant halves of u and v respectively and $u_{\mathsf{L}}, v_{\mathsf{L}}$ are the least significant halves of u and v respectively. Note the subtraction $v_{\mathsf{L}} - v_{\mathsf{H}}$ in Equation 1. Recall that in Section 3 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 i-th bit is zero, and the rest of its bits are ones. Conversely, for the case where the $q_i = 0$, we have that c consists of $i - 1$ random-looking bits.

The code of GnuPG passes c (with some whole-limb truncations) directly to the Karatsuba multiplication routine as the second operand v. Thus, this structure of c has the property that the result of computing $v_{\mathsf{L}} - v_{\mathsf{H}}$ 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.

4.2 Attacking the most significant limb of q

In this section we analyze the effects of our attack on MUL_BASECASE (Algorithm 2). 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 c and m are of the same length and since the constant KARATSUBA_THRESHOLD is defined to be 16, the Karatsuba multiplication routine 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. Combining this observation with the case analysis in Section 3.2, 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.

Thus, during the first call to the Karatsuba multiplication routine, the result of $v_{\mathsf{L}} - v_{\mathsf{H}}$ contains mostly zero limbs, causing the second operand of all the calls to MUL_BASECASE resulting from the recursive call for computing $(u_{\mathsf{H}} - u_{\mathsf{L}}) \cdot (v_{\mathsf{L}} - v_{\mathsf{H}})$ to contain mostly zero limbs.

- If $q_i = 0$, then the second operand b of MUL consists of random-looking limbs. Thus, during the first call to the Karatsuba multiplication routine, result of $v_{\mathsf{L}} - v_{\mathsf{H}}$ contains very few (if any) zero limbs, causing the second operand of *all* the calls to MUL_BASECASE to consist of mostly non-zero limbs.

Next, recall that the control flow in MUL_BASECASE depends on the number of non-zero limbs present in its second operand. The drastic change in the number of zero limbs in the second 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.

Finally, 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 2) 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 leaf resulting from the first computation of $(u_{\mathsf{H}} - u_{\mathsf{L}}) \cdot (v_{\mathsf{L}} - v_{\mathsf{H}})$ for a total of 9 times in each of the 2048 multiplications in that loop. Thus, we get that leakage generated by line 11 in MUL_BASECASE is repeated $2048 \cdot 9 \cdot 7 = 129024$ times. This repetition is what allows the leakage generated by line 11 in MUL_BASECASE to be detected using only low bandwidth measurements.

4.3 The remaining bits of q

Unfortunately, the analysis in Section 4.2 does not precisely hold for the remaining bits of q. Recall that by our choice of ciphertexts, at the beginning of MODULAR_EXPONENTIATION both $c - n$ and q always agree on some prefix of their bits and this prefix becomes longer as the attack progresses and more bits of q are extracted. Thus, since $c - n < 2q$, the reduction of c modulo q always returns $c - q - n$ for the case of $c - n \geq q$ and $c - n$ otherwise.

In particular, after the first limb of q has been successfully extracted, for the case where $q_i = 0$, the value of c after the modular reduction in line 2 of MODULAR_EXPONENTIATION (Algorithm 1) is shorter than 64 limbs. Since in line 9, 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, might make several recursive calls to itself as well as several calls to the Karatsuba multiplication routine.

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. If $q_i = 1$, then c is a 2048 bit number such that the first $2048 - i$ bits are the same as q, the i-th bit is zero, and the rest of the bits are ones. 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 Section 4.2, the number of zero limbs in the second operand of MUL_BASECASE still allows us to extract the bits of q.

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 (discussed in the extended version of this paper [GST13]), we recover these problematic bits and continue our attack.

5 Conclusion

In this paper we developed a new side channel attack, exploiting low-bandwidth computation-dependent acoustic emanations which easily escape computers' chassis (and even expensive Faraday cages). We demonstrated extraction of full RSA keys within a reasonable amount of time using commonly available and easily concealed components. Some algorithmic countermeasures, such as ciphertext normalization and randomization, are effective against our key-extraction attack (and thus implemented in GnuPG consequentially to our results' disclosure), though not against acoustic key-distinguishing, and are discussed in the extended version of this paper [GST13].

Acknowledgments. Lev Pachmanov wrote much of the software setup used in our experiments, including custom signal acquisition programs. Avi Shtibel, Ezra Shaked and Oded Smikt assisted in constructing and configuring the experimental setup. Assa Naveh assisted in various experiments, and offered valuable suggestions. Sharon Kessler provided copious editorial advice. We 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; and to National Instruments Israel for donating PCI-6052E and MyDAQ hardware.

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 259426 ERC-CaC, by the 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".

References

[AA04] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy*, pages 3–11, 2004.

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

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

[BBB+12] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. NIST SP 800-57: Recommendation for key management — part 1: General, 2012.

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

[BWY06] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2006.

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

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

[Gen] Genesis 27:5.

[Gmp] GNU multiple precision arithmetic library. URL: `http://gmplib.org/`.

[Gpg] The GNU Privacy Guard. URL: `http://www.gnupg.org`.

[GST13] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis (extended version). *IACR Cryptology ePrint Archive*, 2013:857, 2013.

[HS10] Tzipora Halevi and Nitesh 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*, pages 97–108, 2010.

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

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

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

[Nat09] National Institute of Standards and Technology. FIPS 140-3: Draft security requirements for cryptographic modules (revised draft), 2009.

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

[ST04] Adi Shamir and Eran Tromer. Acoustic cryptanalysis: on nosy people and noisy machines, 2004. Eurocryt rump session. URL: `http://cs.tau.ac.il/~tromer/acoustic/ec04rump`.

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

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

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

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