Stealing Keys from PCs using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation

(extended version)

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

Lev Pachmanov
Tel Aviv University
levp@post.tau.ac.il

Itamar Pipman
Tel Aviv University
itamarpi@tau.ac.il

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

Posted February 27, 2015
Updated March 3, 2015

Abstract

We present new side-channel attacks on RSA and ElGamal implementations that use the popular sliding-window or fixed-window (m-ary) modular exponentiation algorithms. The attacks can extract decryption keys using a very low measurement bandwidth (a frequency band of less than 100 kHz around a carrier under 2 MHz) even when attacking multi-GHz CPUs.

We demonstrate the attacks' feasibility by extracting keys from GnuPG, in a few seconds, using a nonintrusive measurement of electromagnetic emanations from laptop computers. The measurement equipment is cheap and compact, uses readily-available components (a Software Defined Radio USB dongle or a consumer-grade radio receiver), and can operate untethered while concealed, e.g., inside pita bread.

The attacks use a few non-adaptive chosen ciphertexts, crafted so that whenever the decryption routine encounters particular bit patterns in the secret key, intermediate values occur with a special structure that causes observable fluctuations in the electromagnetic field. Through suitable signal processing and cryptanalysis, the bit patterns and eventually the whole secret key are recovered.

Contents

1 Introduction

1.1 Overview

Even when a cryptographic scheme is mathematically secure and sound, its implementations may be vulnerable to side-channel attacks that exploit physical emanations. Such emanations can leak information about secret values inside the computation, directly or indirectly, and have been exploited by attacks on many cryptographic implementations (see [And08, MOP07, KJJR11] for surveys). Traditionally, most of the research attention on physical side-channel attacks has focused on small devices such as smartcards, FPGAs, RFID tags, and other simple embedded hardware. General-purpose PCs (laptop and desktop computers, servers, etc.) have received less academic attention. While software-based side-channel attacks on PCs (e.g., exploiting timing and CPU cache contention) have been studied, studying physical side channels in PCs requires that we overcome several difficulties:

1. **Complexity.** As opposed to small devices, which often contain a single main chip and some auxiliary components, PCs are highly complex systems containing multiple large chips, numerous electric components, asynchronous mechanisms, and a complicated software stack.
2. **Acquisition Bandwidth.** Typical side-channel approaches require the analog leakage signals to be acquired at a bandwidth greater than the device's clockrate. For the case of PCs running a GHz-scale CPU, recording such high-bandwidth signals requires expensive, cumbersome, and delicate-to-operate lab equipment, and a lot of storage and processing power.
3. **Signal Integrity.** Multi-GHz bandwidths are also hard to acquire with high fidelity, especially non-intrusively, since such high frequencies are usually filtered close to their source using cheap and compact components (such as bypass capacitors) and are often subject to rapid attenuation, reflections, and so forth. Quantization noise is also a concern, due to limited ADC dynamic range at such frequencies (typically under 8 bits, as opposed to 16 or more effective bits at low frequencies).
4. **Attack Scenario.** Traditional side-channel attacks often require that the attacker have undeterred access to the target device. These scenarios often make sense for devices such as smartcards, which are easily pilfered or even handed out to potential attackers (e.g., cable-TV subscription cards). Yet when attacking other people's PCs, the physical access is often limited to brief, nonintrusive access that can go unobserved.

Physical side-channel attack from PCs have been reported only at a low bandwidth leakage (less than a MHz). Emanations of interest have been shown at the USB port [OS06] and through the power outlet [CMR+13]. Recently, low-bandwidth physical side-channel *key-extraction* attacks on PCs were demonstrated [GST14, GPT14], utilizing various physical channels. These last two works presented two different low-bandwidth attacks, with different equipment and attack time requirements:

- **Fast, Non-Adaptive MF Attack.** A non-adaptive chosen-ciphertext attack exploiting signals circa 2 MHz (Medium Frequency band), obtained during several decryptions of a single ciphertext. While both ElGamal and RSA keys can be extracted using this attack in just a few seconds of measurements, the attack used expensive low-noise lab-grade signal acquisition hardware.
- **Slow, Adaptive VLF/LF Attack.** Adaptive chosen-ciphertext attack exploiting signals of about 15–40 kHz (Very Low Frequency / Low Frequency bands) obtained during several decryptions of every ciphertext. Extraction of 4096-bit RSA keys takes approximately one hour, using common equipment such as a sound card or a smartphone.

Scheme	Algorithm	Ciphertext choice	Number of Ciphertexts	Time	Frequency	Equipment	Ref.
RSA	square and multiply	Adaptive	$\frac{\text{key bits}}{4}$	1 hour	50 kHz	common	[GST14]
RSA, ElGamal	square and always multiply	Non-adaptive	1	seconds	2 MHz	lab-grade	[GPT14]
RSA	sliding/fixed window	Non-adaptive	16	seconds	2 MHz, 100 kHz bandwidth	common	This work
ElGamal			8				

Table 1: Comparison of previous physical key extraction attacks on PCs. #ciphertexts counts the number of distinct ciphertexts; measurements may be repeated to handle noise.

This leaves a practicality gap: the attacks require either expensive lab-grade equipment (in the non-adaptive case), or thousands of adaptively-chosen ciphertexts decrypted over an hour (in the adaptive case). See Table 1 for a comparison.

Another limitation of [GST14, GPT14] is that they target decryption algorithm implementations that use a slow exponentiation algorithm: square-and-multiply, which handles the secret exponent's bits one at a time. These attacks do not work for sliding-window or fixed-window exponentiation, used in most RSA and ElGamal implementations nowadays, which preprocess the ciphertext and then handle the key in chunks of multiple bits.

1.2 Our Contribution

In this work we make progress on all fronts outlined above. We present and experimentally demonstrate a new physical side-channel key-extraction attack, which is the first to achieve the following:

1. **Windowed Exponentiation on PCs.** The attack is effective against RSA and ElGamal implementations that use sliding-window or fixed-window (m-ary) exponentiation, as in most modern cryptographic libraries, and running on PCs.

Moreover, the attack *concurrently* achieves all of the following properties (each of which was achieved by some prior work on PCs, but never in combination with the other properties, and not for sliding-window exponentiation):

2. **Short Attack Time.** This attack uses as few as 8 (non-adaptively) chosen ciphertexts and is able to extract the secret key in just several seconds of measurements.
3. **Low Frequency and Bandwidth.** The attack measures signals at a frequency of merely 2 MHz, and moreover at a low bandwidth (less than 100 kHz around the carrier). This makes signal acquisition robust and inexpensive.
4. **Small, Cheap and Readily-Available Setup.** Our attack can be mounted using simple and readily available equipment, such as a cheap Software Defined Radio USB dongle attached to a loop of cable and controlled by a regular laptop or a small SoC board (see Figures 10–11 and 10). Alternatively, in some cases all that is required is a common, consumer-grade radio, with its audio output recorded by a phone (see Figure 13). In both cases, we avoid the expensive equipment used in prior attacks, such as low-noise amplifiers, high-speed digitizers, sensitive ultrasound microphones, and professional electromagnetic probes.

Cryptanalytic Approach. Our attack utilizes the fact that, in the sliding-window or fixed-window exponentiation routine, the values inside the table of ciphertext powers can be partially predicted. By crafting a suitable ciphertext, the attacker can cause the value at a specific table entry to have a specific structure. This structure, coupled with a subtle control flow difference deep inside GnuPG's basic multiplication routine, will cause a noticeable difference in the leakage whenever a multiplication by this structured value has occurred. This allows the attacker to learn all the locations inside the secret exponent where the specific table entry is selected by the bit pattern in the sliding window. Repeating this process across all table indices reveals the key.

Signal Acquisition and Analysis Approach. The attack is demonstrated, via the electromagnetic channel, on the latest version of GnuPG, for ElGamal and RSA decryption using a regular unaltered laptop without any intrusion or disassembling. For each table index, we craft a suitable ciphertext and trigger its decryption. The exploited leakage appears as frequency-modulation on carrier waves, most clearly observed at 1.5–2 MHz. During decryption, we measure a narrow frequency band, typically 100 kHz, around the carrier. After filtering, demodulation, distortion compensation and averaging, a clean aggregate trace is produced for each table index. We then recover the key by deductively combining the (misaligned but partially-overlapping) information contained in the aggregate traces.

1.3 Vulnerable Software and Hardware

Targeted Hardware. Similarly to [GST14, GPT14], this work targets commodity laptop computers. We have tested numerous laptop computers of various models and makes.[1] For concreteness, in the remainder of this paper our examples use Lenovo 3000 N200 laptops, which exhibit a particularly clear signal.

GnuPG. We focused on GnuPG version 1.4.18 [Gpga], which is the latest version at the time of writing this paper. We compiled GnuPG using the MinGW GCC version 4.6.2 [Min] and ran it on Windows XP. GnuPG 2.1 (developed in parallel to GnuPG 1.x), as well as its underlying cryptographic library, libgcrypt (version 1.6.2), utilize very similar cryptographic codes and thus may also be vulnerable to our attack.

Following past attacks [GST14, GPT14], GnuPG now uses ciphertext randomization for RSA decryption (but not for ElGamal; see Section 4). To test our attack on RSA with sliding-window exponentiation, we disabled that countermeasure, making GnuPG decrypt the ciphertext directly as in prior versions. The ElGamal attack applies to unmodified GnuPG.

Current Status. Following the practice of responsible disclosure, we worked with the authors of GnuPG to suggest several countermeasures and verify their effectiveness against our attacks (see CVE-2014-3591 [MIT14]). GnuPG 1.4.19 and Libgcrypt 1.6.3, resilient to these attacks, were released concurrently with the public announcement of the results presented in this paper.

Chosen Ciphertext Injection. GnuPG is often invoked to decrypt externally-controlled inputs, fed into it by numerous frontends, via emails, files, chat and web pages. The list of GnuPG frontends [Gpgb] contains dozens of such applications, each of them can be potentially used in order to make the target decrypt the chosen ciphertexts required by our attack. Morover, since our attack is non-adaptive (the choice of ciphertext is fixed for all secret keys), such ciphertexts can be quickly injected into the target using just a single communication round. Concretely, as observed in [GST14, GPT14], Enigmail [Eni], a plugin for the Mozilla Thunderbird e-mail client,

[1] Signal quality varied dramatically with the target computer model and probe position. Computers of the same model exhibited consistent optimal probe position, but slight differences in the emitted signals (which can be used to distinguish between them).

automatically decrypts incoming emails by passing them directly to GnuPG. Thus, it is possible to remotely inject such ciphertexts into GnuPG by sending them as a PGP/MIME-encoded e-mail [ETLR01]. We have empirically verified that such an injection method does not have any noticeable effect on the leakage signal produced by the target laptop. GnuPG's Outlook plugin, GpgOL, also did not seem to alter the target's leakage signal.

1.4 Related Work

Side-channel attacks have been demonstrated on numerous cryptographic implementations and via various leakage channels (see [And08, MOP07, KJJR11] and the references within).

The EM Side Channel. The electromagnetic side channel, specifically, has been exploited for attacking smartcards, FPGA and other small devices (e.g., [QS01, GMO01, AARR02]). On PCs, [ZP14] observed electromagnetic leakage from laptop and desktop computers (but did not show cryptanalytic applications), and [GPT14] demonstrated successful EM attacks on a side-channel protected PC implementation of the square-and-multiply modular exponentiation algorithm, achieving RSA and ElGamal key extraction.

Attacks on Sliding Window Modular Exponentiation. While most side-channel attacks on public key schemes focus on variants of the square-and-multiply modular exponentiation algorithm, several focus on attacking sliding window modular exponentiation on small devices (sampling much faster than the target device's clockrate). These attacks either exploit high-bandwidth operand-dependent leakage of the multiplication routine [Wal01, CFG+10, HMA+08, HIM+13] or utilize the fact that it is possible to distinguish between squarings and multiplications [FKM+06, CRI12].

Neither of the above approaches fits our case. The first requires very high-bandwidth leakage, while the second is blocked by a recently-introduced countermeasure to the attack of [YF14]: GnuPG uses the same code for squaring and multiplications (and the resulting EM leakage indeed appears indistinguishable at low bandwidth).

Side-channel Attacks on PCs. Physical side-channel attacks of PCs were demonstrated by observing leakage through the USB port [OS06] or through the power outlet [CMR+13]. Key extraction side-channel attacks have been presented on PC computers, utilizing the timing differences [BB05] and cache access patterns [Ber05, Per05, OST06]. Recently, low-bandwidth key-extraction attacks that utilize physical channels such as sound [GST14] and chassis potential [GPT14] were demonstrated on GnuPG running on PCs.

Cache Attacks in GnuPG. Yarom and Falkner [YF14] presented an L3 cache attack on the square-and-multiply algorithm, achieving key extraction by directly observing the sequence of squarings and multiplications performed. In a concurrent work, Yarom et al. presented [YLG+15] an attack on sliding-window exponentiation by observing the access patterns to the table of ciphertext powers.

1.5 Preliminaries

ElGamal Encryption. Recalling notation, in the ElGamal encryption [ElG85] key generation consists of generating a large prime p, a generator g of $\mathbb{Z}_p^*$, and a secret exponent χ. The public key is (p, g, g^χ) and the secret key is χ. Encryption of a message m outputs a pair (γ, δ), where $\gamma = g^s \bmod p$ and $\delta = m \cdot (g^\chi)^s \bmod p$, and $s \in \mathbb{Z}_p^*$ is random. A ciphertext (γ, δ) is decrypted by computing the exponentiation $\gamma^{-\chi}$, and then multiplying by δ modulo p. GnuPG selects the prime p to be a random *safe prime*, meaning that $p = 2p' + 1$ for some prime p'.

RSA Encryption. In RSA encryption [RSA78] the key generation consists of selecting two large randomly chosen primes p, q, a (fixed) public exponent e, and a secret exponent d such that $ed \equiv 1 \pmod{\phi(n)}$ where $n = pq$. The public key is (n, e) and the secret key is (d, p, q). RSA encryption of a message m is performed by computing $m^e \mod n$. RSA decryption of a ciphertext c is performed by computing $c^d \mod n$. GnuPG uses an optimization of RSA that is based on the Chinese Remainder Theorem (CRT). That is, in order to compute $m = c^d \mod n$, GnuPG first computes the two exponentiations $m_p = c^{d_p} \mod p$ and $m_q = c^{d_q} \mod q$ (where d_p and d_q are derived from the secret key) and then combines m_p and m_q into m using the CRT.

2 Cryptanalysis

This section describes our cryptanalytic attack techniques (the applicability of which we demonstrate in Section 3). We begin by reviewing the GnuPG modular exponentiation algorithm (Section 2.1) and proceed by describing our attack on ElGamal decryption (Section 2.2) and on RSA decryption (Section 2.3).

2.1 GnuPG's Sliding-Window Exponentiation Routine

GnuPG uses an internal mathematical library called MPI (based on GMP [Gmp]) in order to perform the large integer operations occurring in ElGamal decryption. In recent versions (starting with GnuPG v1.4.16), this exponentiation is performed using a sliding-window algorithm, as follows.

MPI stores large integers as arrays of *limbs*, which are 32-bit words (on the x86 architecture used in our tests). Algorithm 1 is a pseudocode of the modular exponentiation routine, which operates on such limb arrays (simplified for the case of 32-bit limb size). The function SIZE_IN_LIMBS(x) returns the number of limbs in the t-bit number x, namely $\lceil t/32 \rceil$. The functions COUNT_LEADING_ZEROS(x) and COUNT_TRAILING_ZEROS(x) count the number of leading and trailing zero bits in x respectively. Finally, SHIFT_LEFT(x,y) and SHIFT_RIGHT(x,y) respectively shift x to the left or to the right y bits.

For 3072-bit ElGamal keys, GnuPG chooses the key so that the exponent $d = -\chi$ (using the notation of Section 1.5) is about 400 bits, and thus $w = 4$. For 4096-bit RSA keys, the size of both p and q is 2048 bits. Since GnuPG's RSA uses the CRT, the exponents d_p and d_q are also 2048 bits long; hence, $w = 5$.

Consider the computation in lines 14–18. For a fixed value of w, this computes a table indexed by 1,3,5,...,$2^w - 1$ (i.e., 2^{w-1} entries in total), mapping each such odd w-bit integer u to the group element g^u. Moreover, note that this computation only depends on the ciphertext and the modulus and not on the secret exponent. We will show how to exploit this table to create exponent-dependent leakage during the main loop of Algorithm 1, leading to full key extraction.

2.2 ElGamal Attack Algorithm

We start by describing the attack algorithm on GnuPG's ElGamal implementation, which uses sliding-window exponentiation. At the end of the section we discuss the fixed-window version.

Let *SM-sequence* denote the sequence of squaring and multiplication operations in lines 25, 29 and 32 of Algorithm 1. Note that this sequence depends only on the exponent d, and not on the value of g or p. If an attacker were to learn the SM-sequence, and moreover obtain for each multiplication performed by line 29 the corresponding table index u used to index the table computed in lines 14-18, then the secret exponent could be easily recovered as follows. Start from a partial exponent set to 1. Then, going over the SM-sequence from start to end, for every squaring

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

Input: Three integers g, d and p where $d_1 \cdots d_n$ is the binary representation of d.
Output: $a \equiv g^d \pmod{p}$.

```
1: procedure MOD_EXP(g, d, p)
2:     if SIZE_IN_LIMBS(g) > SIZE_IN_LIMBS(p) then
3:         g ← g mod p
4:     if SIZE_IN_LIMBS(d) > 16 then                    ▷ compute w, the window size
5:         w ← 5
6:     else if SIZE_IN_LIMBS(d) > 8 then
7:         w ← 4
8:     else if SIZE_IN_LIMBS(d) > 4 then
9:         w ← 3
10:    else if SIZE_IN_LIMBS(d) > 2 then
11:        w ← 2
12:    else
13:        w ← 1
14:    g_0 ← 1, g_1 ← g, g_2 ← g^2
15:    for i ← 1 to 2^{w-1} − 1 do                      ▷ precompute table of small powers of g
16:        g_{2i+1} ← g_{2i-1} · g_2
17:        if SIZE_IN_LIMBS(g_{2i+1}) > SIZE_IN_LIMBS(p) then
18:            g_{2i+1} ← g_{2i+1} mod p
19:    a ← 1
20:    j ← 0
21:    while d ≠ 0 do                                   ▷ main loop for computing g^d mod p
22:        j ← j + COUNT_LEADING_ZEROS(d)
23:        d ← SHIFT_LEFT(d, j)
24:        for i ← 1 to j + w do
25:            a ← a · a mod p                          ▷ using multiplication, not squaring
26:        t ← d_1 ··· d_w
27:        j ← COUNT_TRAILING_ZEROS(t)
28:        u ← SHIFT_RIGHT(t, j)
29:        a ← a · g_u mod p
30:        d ← SHIFT_LEFT(d, w)
31:    for i ← 1 to j do
32:        a ← a · a mod p                              ▷ using multiplication, not squaring
33:    return a
34: end procedure
```

operation in the SM-sequence append a zero bit to the partially-known exponent. Whenever a multiplication operation is encountered in the SM-sequence, replace the least significant bits of the exponent with the table index u corresponding to the multiplication.

Revealing the Locations of a Table Index. We now discuss how, for any given table index u, the attacker can learn the locations where multiplications by g_u, as performed by line 29, occur inside the SM-sequence. In what follows, for any given table index u, we shall refer to such locations as SM-locations. Recall that for 3072-bit ElGamal, GnuPG sets $w = 4$ inside Algorithm 1, so the

table indices are odd 4-bit integers.[2] Thus, given an odd 4-bit integer u, the attacker chooses the ciphertext so that multiplications by g_u produce different side-channel leakage compared to multiplications by $g_{u'}$ for all $u' \neq u$.

First, the attacker selects a number $y \in \mathbb{Z}_p^*$ containing many zero limbs and computes its u-th root, i.e., x, such that $x^u \equiv y \pmod{p}$.[3] It is likely that for all other odd 4-bit integer $u' \neq u$, there are few zero limbs in $x^{u'} \bmod p$ (otherwise, the attacker selects a different y and retries). Finally, the attacker requests the decryption of (x, δ) for some arbitrary value δ and measures the side channel leakage produced during the computation of $\text{MOD\_EXP}(x, d, p)$.

Distinguishing Between Multiplications. The above process of selecting x given an odd 4-bit integer u allows an attacker to distinguish between multiplications by g_u and multiplications by $g_{u'}$ for all $u' \neq u$ during the main loop of $\text{MOD\_EXP}(x, d, p)$. Indeed, note that by the code of Algorithm 1 we have that $g_u = x^u \bmod p = y$, which is a number containing many zero limbs. Conversely, for any $u' \neq u$ we have that $g_{u'} = x^{u'} \bmod p$ is a number containing few (if any) zero limbs. The number of zero limbs in the second operand of the multiplication can be detected via side channels, as observed by [GST14, GPT14] and summarized in Section 2.4. Thus, by observing the leakage of the multiplication routine, it is possible to distinguish the multiplications by g_u in line 29 of Algorithm 1 from multiplications by $g_{u'}$ where $u' \neq u$.

The above allows the attacker to distinguish between multiplications by g_u and multiplications by $g_{u'}$ for all $u' \neq u$ during $\text{MOD\_EXP}(x, d, p)$. In order to determine the SM-locations of g_u, it remains for the attacker to distinguish the multiplications by g_u from the squarings performed in lines 25 and 32. GnuPG implements the squaring in lines 25 and 32 using the same multiplication code used for line 29 (this is a countermeasure to the attack of [YF14]). Thus, the attacker cannot immediately distinguish between the leakage produced by the squaring operations and the leakage produced by multiplication operations where the second operand is $g_{u'}$ for some $u' \neq u$. However, in the case of squaring, the argument a supplied to the multiplication routine is a random-looking intermediate value, which is unlikely to contain any zero limbs. Thus, the squaring operations will produce similar leakage to that produced by multiplications by $g_{u'}$ for some $u' \neq u$. Although the attacker cannot yet distinguish between multiplications by $g_{u'}$ ($u' \neq u$) and squaring operations, he can nonetheless determine the SM-locations of g_u.

Key Extraction. Applying this method to every possible table index u (since u is an odd 4-bit number, only 8 possible values of u exist), the attacker learns the SM-locations of multiplications performed in line 29. Moreover, since each ciphertext corresponds to only a single table index u, the attacker also learns the value of u used during these multiplications. All that remains for the attacker to learn in order to recover the secret exponent is the SM-locations of the squaring operations performed by lines 25 and 32 of Algorithm 1. However, since at this point the SM-locations of all multiplication operations have been identified, any remaining location corresponds to a squaring operation performed by lines 25 and 32. Therefore, the attacker has learned the entire SM-sequence performed by Algorithm 1 and moreover obtained, for each multiplication performed by line 29, the corresponding value of the table index u, thus allowing him to recover the secret exponent.

Figure 1 presents the number of zero limbs in the second operand of the multiplication routine during the modular exponentiation routine, for each of our chosen ciphertexts and a randomly-

[2] To reduce the table size, GnuPG's code actually maps an odd 4-bit table index $u = 2u' + 3$ to a 3-bit index u', to store the table in a continuous array. For simplicity of exposition, we describe a direct mapping; this does not affect our attack.

[3] The attacker computes the u-th root of y modulo p, as follows. Since in GnuPG p is a large safe prime, for any odd 4-bit integer u it holds that $\gcd(u, p-1) = 1$, and therefore v can be computed such that $uv \equiv 1 \pmod{p-1}$, and then the u-th root is $x = y^v \bmod p$.

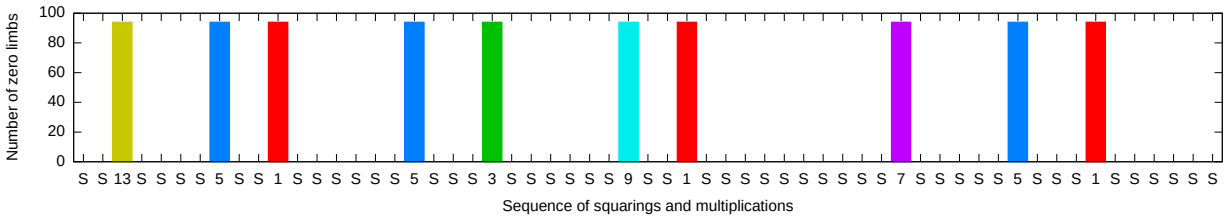


Figure 1: Number of zero limbs in the operand of the multiplication routine during an execution of the modular exponentiation routine, using our ElGamal attack and a randomly-generated key. Each squaring is marked by S and each multiplication is marked by the corresponding table index u of its second operand. For each chosen ciphertext, when its corresponding table index is used, there are 94 zero limbs in the second operand of the multiplication routine.

generated key. Note that for each ciphertext, the number of zero limbs increases only when its corresponding table index, u, is used. Across all ciphertexts, it is possible to deduce the exact SM-sequence and moreover to obtain, for each multiplication performed by line 29, the corresponding value of the table index u, and then (as discussed above) deduce the exponent.

Attacking the Fixed-Window Method. The fixed-window (m-ary) exponentiation method (see [MVO96, Algorithm 14.109]) avoids the key-dependent shifting of the window, thus reducing side-channel leakage. The exponent is split into contiguous, fixed-size m-bit words. Each word is then handled in turn by performing m squaring operations and a single multiplication by an appropriate value selected from a precomputed table using the current exponent word as the table index.

In attacking fixed-window ElGamal, each table index u may be targeted similarly as in the sliding window case by having the attacker select a number $y \in Z_p^*$ containing many zero limbs and compute the u-th root of y, x, such that $x^u \equiv y$. Like in the sliding window case, for any other m-bit word $u' \neq u$, it is likely that $x^{u'} \bmod p$ will contain few (if any) zero limbs. The remainder of the attack—leakage analysis and key extraction—is the same as for sliding window.

2.3 RSA Attack Algorithm

As in the case of ElGamal, the security of RSA also breaks down if the secret exponent d leaks. Moreover, even leakage of the top half of the bits of d_p (or of d_q) allows for a complete key recovery [Cop97]. In this section, we show how to adapt the ElGamal attack presented in Section 2.2 to RSA. As before, we first describe the attack algorithm on GnuPG's RSA implementation, which uses sliding-window exponentiation, and at the end of the section we discuss the fixed-window version.

Revealing the Location of Table indices. In the case of 4096-bit RSA, GnuPG uses a size of $w = 5$ bits. Given an odd 5-bit table index u, the attacker would like to learn the SM-locations of multiplications by g_u performed during the modular exponentiation routine. However, unlike for ElGamal, in this case the attacker does not know p and thus cannot select a number y containing many zero limbs and compute x such that $x^u \equiv y \pmod{p}$. Neither can the attacker compute u-th roots modulo N to compute $x^u \equiv y \pmod{N}$, as this would contradict the security of RSA with public exponent u.

Approximating the Location of Table indices. However, locating the precise locations is not,

in fact, necessary: the requirements can be relaxed so that, given a 5-bit odd integer u, the attacker will learn all the SM-locations of multiplication by $g_{u'}$ for some $u' \leq u$. To this end, the attacker no longer relies on solving modular equations over composite-order groups, but rather on the fact that, during the table computation phase inside GnuPG's modular exponentiation routine, as soon as the number of limbs of some table value g_u exceeds the number of limbs in the prime p, the table value g_u is reduced modulo p (see line 17 of Algorithm 1). Thus, given a 5-bit odd integer u, the attacker will request the decryption of a number t such that t contains many zero limbs and that $t^u \leq 2^{2048} < t^{u+1}$. The two above requirements are instantiated by computing the largest integer k such that $k \cdot u \leq 2048$ and requesting the decryption of 2^k. Finally, the side-channel leakage produced during the computation of $\text{MOD\_EXP}(2^k, d_p, p)$ is recorded.

Distinguishing Between Multiplication. Fix an odd 5-bit integer u and let k be the largest integer such that $(2^k)^u \leq 2^{2048}$. The SM-sequence resulting from the computation of $\text{MOD\_EXP}(2^k, d_p, p)$ contains three types of multiplication operations, creating two types of side-channel leakage.

1. **Multiplication by $g_{u'}$ where $u' \leq u$.** In this case $(2^k)^{u'} \leq (2^k)^u \leq 2^{2048}$ and therefore $g_{u'} = 2^{k \cdot u'} \bmod p$ does not undergo a reduction modulo p. Thus $g_{u'} = 2^{k \cdot u'}$, which is a number containing many zero limbs.

2. **Multiplication by $g_{u'}$ where $u' > u$.** In this case $2^{2048} < (2^k)^{u'}$ and therefore $g_{u'} = 2^{k \cdot u'} \bmod p$ undergoes a reduction modulo p, making it a random-looking number that will contain very few (if any) zero limbs.

3. **Multiplication resulting from squaring operations.** As mentioned in Section 2.2, GnuPG implements the squaring in lines 25 and 32 using the same multiplication code used for line 29. In the case of squaring, the argument a supplied to the multiplication routine is a random-looking intermediate value, which is unlikely to contain any zero limbs. Thus, the squaring operations will produce similar leakage to case 2 above.

Next, as in the attack presented in Section 2.2, since the leakage produced by GnuPG's multipication routine depends on the number of zero limbs in its second operand (See [GST14, GPT14] and Appendix 2.4 for an extended discussion), it is possible to distinguish between multiplications by $g_{u'}$ for some $u' \leq u$ (case 1 above) and all other multiplications (cases 2 and 3 above). Thus, the attacker learns the SM-locations of all multiplications by $g_{u'}$ where $u' \leq u$.

Key Extraction. The attacker applies the above method for every possible table index u (since u is an odd 5-bit integer, only 16 possible values of u exist). He can thus deduce the SM-locations of every multiplication performed by line 29. Moreover, for each multiplication, by finding the lowest u such that the leakage of the multiplication corresponds to case 1 above, the attacker deduces the table index u of its second operand.

The attacker has now learned the sequence of table indices (i.e., odd 5-bit values) that occur as the sliding window moves down the secret exponent d_p. To recover the secret exponent, the attacker need only discover the amounts by which the window slides between these values (due to runs of zero bits in d_p). This sliding is realized by the loops in lines 25 and 32 of Algorithm 1, and can thus be deduced from the SM-locations of the squaring operations in lines 25 and 32. These SM-locations are simply the remaining SM-locations after accounting for those of the multiplications in line 29, already identified above. The attacker has now learned the position and value of all bits in d_p.

Empirically illustrating the deduction (using a randomly generated secret-key), Figure 2 presents the number of zero limbs in the second operand of GnuPG's multiplication routine during the modular exponentiation routine for each of our chosen ciphertexts. Note that for a ciphertext corresponding to table index u, the number of zero limbs is greater than zero only in SM-locations

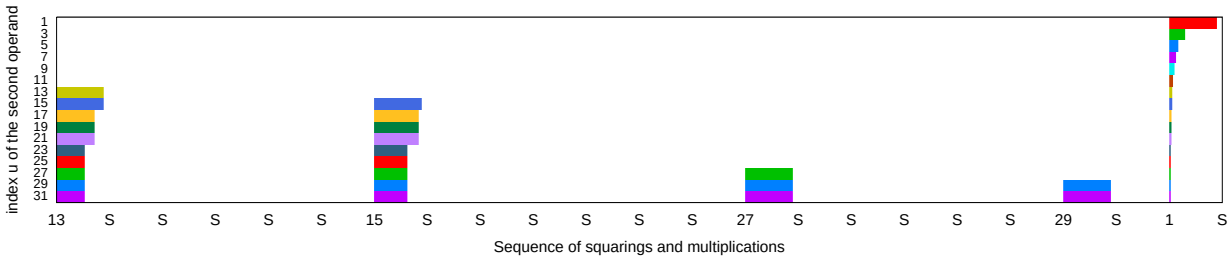


Figure 2: Number of zero limbs in the operand of the multiplication routine during an execution of the modular exponentiation routine, using our RSA attack and a randomly-generated key. Each squaring is marked with an S and each multiplication is marked with the corresponding table index u of the second operand. For each multiplication, and for each ciphertext corresponding to some table index u, the width of the corresponding bar is proportional to the number of zero limbs in the second operand of the multiplication routine. For each chosen ciphertext corresponding to some table index u, the number of zero limbs is greater than zero only when a table index $u' \leq u$ is used.

where there is a multiplication by $g_{u'}$, where $u' \leq u$. From this data, it is possible to deduce the exact SM-sequence and recover the exponent d_p.

Attacking the Fixed-Window Method. As for ElGamal case, this attack can also be applied to the fixed-window (m-ary) exponentiation case. This is done by modifying the attack above to approximate the location of all m-bit table indexes (as opposed to only odd m-bit indexes). The remainder of the attack—leakage analysis and key extraction—is the same as for the sliding window case.

2.4 Leakage from GnuPG's Multiplication Routine

As mentioned in Section 2.2 and 2.3, the leakage produced by GnuPG's multiplication routine varies according to the number of zero limbs in its second operand. The root cause of this data-dependent leakage, as exploited by our attacks, is located deep inside the code of GnuPG's multiplication routines. These routines were extensively analyzed in [GST14, GPT14]; the following is a simplified analysis.

GnuPG's large integer multiplication code uses two different multiplication algorithms: a variant of a recursive Karatsuba multiplication algorithm [KO62], and a simple grade-school "long multiplication" algorithm. The chosen combination of algorithms is based on the size (in limbs) of the second operand.

Basic Multiplication Routine. GnuPG's basic multiplication routine is presented in Algorithm 2. Note the optimizations for the case where a limb that equals 0 or 1 is encountered inside the second operand b. In particular, if a zero limb 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 makes the control flow (and thus side-channel leakage) depend on the operands of the multiplication routine.

This basic multiplication routine is used either directly from the modular exponentiation routine (when the second operand is small), or serves as the base case for the following recursive Karatsuba multiplication routine.

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 ≤ 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
```

Karatsuba Multiplication Routine. Given two numbers, a and b, denote by $a_\mathsf{H}, b_\mathsf{H}$ the most significant halves of a and b respectively. Similarly, denote by $a_\mathsf{L}, b_\mathsf{L}$ the least significant halves of a and b respectively. GnuPG's Karatsuba multiplication routine relies on the following identity:

$$ab = (2^{2n} + 2^n)a_\mathsf{H}b_\mathsf{H} + 2^n(a_\mathsf{H} - b_\mathsf{L})(b_\mathsf{L} - a_\mathsf{H}) + (2^n + 1)a_\mathsf{L}b_\mathsf{L} \ .$$

Operand-Dependent Leakage. Both the ElGamal and RSA attacks utilize the same side-channel weakness in GnuPG's basic multiplication routine. This weakness allows the attacker to distinguish multiplications where the second operand contains many zero limbs from multiplications where it does not. Moreover, if the second operand does contain many zero limbs, then the second operand in all three recursive calls for computing $a_\mathsf{H}b_\mathsf{H}$, $(a_\mathsf{H} - b_\mathsf{L})(b_\mathsf{L} - a_\mathsf{H})$ and $a_\mathsf{L}b_\mathsf{L}$ performed by GnuPG's variant of the Karatsuba multiplication algorithm will also contain many zero limbs. Then, as the recursion eventually reaches its base case, most of the calls to MUL_BY_SINGLE_LIMB, ADD_WITH_OFFSET and MUL_AND_ADD_WITH_OFFSET inside the basic multiplication routine will be skipped. Conversely, if the second operand of the multiplication routine is random looking, so will be the second operand in all three recursive calls during GnuPG's Karatsuba multiplication routine. This in turn will cause the basic multiplication routine to execute most of the calls to MUL_BY_SINGLE_LIMB, ADD_WITH_OFFSET and MUL_AND_ADD_WITH_OFFSET. Finally, since the basic multiplication routine is executed many times during the modular exponentiation routine, this drastic change inside its control flow creates side-channel leakage observable by even low-bandwidth means.

3 Experimental Results

This section presents experimental key extraction using the above cryptanalytic attack, via the electromagnetic side channel, using inexpensive Software Defined Radio (SDR) receivers and consumer radios.

3.1 SDR Experimental Setup

Our first experimental setup uses Software Defined Radio to study EM emanations from laptop computers at frequencies of 1.5–2 MHz, as detailed below (see also Figure 3).

Probe. As a magnetic probe, we constructed a simple shielded loop antenna using a coaxial cable, wound into 3 turns of 15 cm diameter, and with suitable conductor soldering and center shield gap [Smi99]. (For the compact untethered setup of Section 3.6 we used a different antenna, described there.)

Receiver. We recorded the signal produced by the probe using a FUNcube Dongle Pro+ [Fun] SDR receiver. The FUNcube Pro+ is an inexpensive (GBP 125) USB dongle that contains a software-adjustable mixer and a 192 Ksample/sec ADC, accessed via USB as a soundcard audio interface. We used the GNU Radio software [Gnu] to interface with this receiver. Numerous cheaper alternatives exist, including "rtl-sdr" USB receivers based on the Realtek RTL2832U chip (originally intended for DVB-T television receivers) with a suitable tuner and upconverter; the Soft66RTL2 dongle [RTL] (USD 50) is one such example.

Probe Placement. The placement of the EM probe relative to the laptop greatly influences the measured signal and noise. We wished to measure EM emanations close to the CPU's voltage regular, located on the laptop's motherboard, yet without mechanical intrusion. In most modern laptops, the voltage regulator is located in the rear left corner, and indeed placing the probe close to this corner usually yields the best signal. With our loop antennas, the best location is parallel to the laptop's keyboard for close distances (up to approximately 20 cm), and perpendicular to the keyboard for larger distances; see Figures 3, 9 and 11 for examples.

Exponent-Dependent Leakage. To confirm the existence of leakage that depends on the (secret) exponent, we first show how different exponents cause different leakage. Figure 4 demonstrates ElGamal decryption operations using different secret exponents, easily distinguishable by their electromagnetic leakage. Similar results were obtained for RSA.

3.2 Signal Analysis

Demodulation. As can be seen in Figure 4, when using periodic exponents the leakage signal takes the form of a central peak surrounded by distinguishable side lobes. This is a strong indication that the secret bit exponents are modulated by the carrier. As in [GPT14], the carrier signal turned out to be frequency modulated, though in our case the baseband signal does not directly represent the key bits.

Different targets produce such FM-modulated signals at different, and often multiple, frequencies. In each experiment, we chose one such carrier and applied a band-pass filter around it: first via coarse analog RC and LC filters (some built into the SDR receiver), and then via a high-order digital band-pass filter. We then demodulated the filtered signal using a discrete Hilbert transform and applied a low-pass filter, yieldinga demodulated trace as shown in Figure 5(a).

Figure 3: A shielded loop antenna (handheld) connected to the the attacker's computer through an SDR receiver (right), attacking a Lenovo 3000 N200 target (left).

Signal Distortions. In principle, only a single demodulated trace is needed per chosen ciphertext, if measurment is sufficiently robust. However, the signals obtained with our setup (especially those recorded from afar) have insufficient signal-to-noise ratio for reliable information extraction.

Moreover, there are various distortions afflicting the signal, making straightforward key extraction difficult. The signals are corrupted every 15 msec, by the 64 Hz timer interrupt on the target laptop. Each interrupt event corrupts the trace for a duration of several exponent bits, and may also create a time shift relative to other traces (see Figures 5(b) and 6(a)). In addition, many traces exhibit a gradual drift, increasing the time duration between two adjacent peaks (relative to other traces), making signal alignment even more problematic (see Figure 6(b)).

The attack of [GPT14], targeting square-and-always-multiply exponentiation, overcame interrupts and time drift using the fact that every given stage in the decryption appears in non-corrupted form in most of the traces. They broke the signal down into several time segments and aligned them using correlation, thereby resolving shift issues. Noise was suppressed by averaging the aligned segments across all signals. Since, in their case, the baseband signal reflected a sequence of random-looking key bits, correlation proved sufficient aligning trace segments.

However, such correlation-based alignment and averaging is inadequate for sliding-window exponentiation. Here, the demodulated traces are mostly periodic, consisting of a train of similar peaks that change only when the corresponding table index is used for multiplication. Correlating nearly-periodic signals produces an ambiguity as to the actual shift compensation required for proper alignment; it is also not very robust to noise.

The problem is exacerbated by the low bandwidth of the attack: had we (expensively) performed clockrate-scale measurments, consecutive peaks would likely have been distinguishable due to fine-grained data dependency, making alignment via segment correlation viable.

Aligning the Signals. As a first attempt to align the signals and correct distortions, we applied the "Elastic Alignment" [vWWB11] algorithm to the demodulated traces; however, for our signals the results were very unreliable. For more robust key extraction, we used a more problem-specific algorithm.

Initial Synchronization. We started by aligning all traces belonging to identical decryption operations using a short segment in the very beginning of each trace, before the start of the modular exponentiation operation. This segment is identical in all traces, making it suitable as a trigger for the alignment. All traces were aligned via correlation relative to a reference trace,

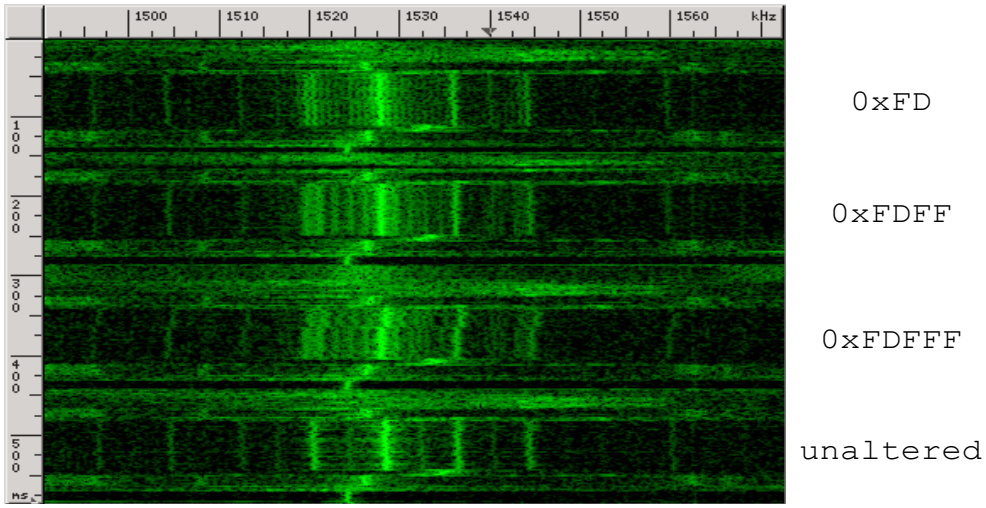


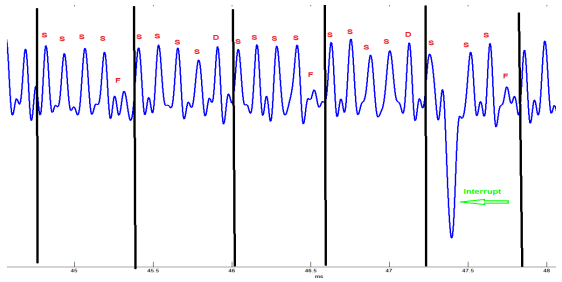
Figure 4: EM measurement (0.5 sec, 1.49–1.57 MHz) of four GnuPG ElGamal decryptions executed on a Lenovo 3000 N200 laptop. In the first 3 cases, the exponent is overridden to be the 3072-bit number obtained by repeating the bit pattern written to the right. In the last case, the exponent is unaltered. In all cases, the modulus p is the same and the ciphertext c is set to be such that $c^{15} \equiv 2^{3071} \pmod{p}$. Note the subtly different side lobes around the 1527 kHz carrier.

chosen randomly from within the trace set. If by some chance this initial segment was distorted in the chosen reference trace, the reference trace was discarded and a new one chosen. Traces that did not align well with the reference trace were also discarded. This process required a few dozen decryption traces per window (taking a few seconds total) in order to produce enough valid traces for reliable key extraction. We then independently compared each trace against the reference trace, correcting any distortion as soon as it manifested by changing the signal accordingly, as described next. This is possible because not all interrupts occur at the exact same time, and no two drifts are the same.

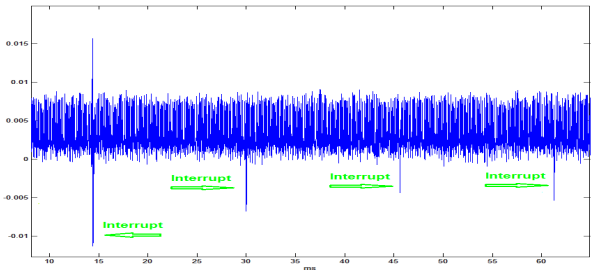
Handling Interrupts. In order to align the signals despite the interrupt-induced shifts, a search for interrupts was performed simultaneously across both the current signal and the reference signal, from beginning to end. Interrupts are easily detected, as they cause large frequency fluctuations. Whenever an interrupt was encountered in one of the signals, the relative delay it induced was estimated by correlating a short segment immediately following the interrupt in both signals. The samples corresponding to the interrupt duration were then removed from the interrupted signal, to restore alignment. The process was repeated until no more interrupts were detected in either signal and the signals were fully aligned. Note that the delay created by the interrupts was usually shorter than the peaks in the demodulated trace, so there was no ambiguity in the correlation and resulting delay estimate.

Handling Drifts. The slow drifts were handled by adding another step to the above process. Between each pair of detected interrupts, we performed a periodic comparison (again, by direct correlation) and compensated for the drift by removing samples from the appropriate signal (as done for interrupts). In order to avoid ambiguity in the correlation, the comparisons were made frequently enough so that the slow drift never created a delay longer than half a peak.

Aggregating Aligned Traces. The foregoing process outputs several fully-aligned but noisy traces that still contain occasional interrupts (since the interrupt duration is usually several peaks long but creates a delay of no more than one peak, the compensation process does not completely

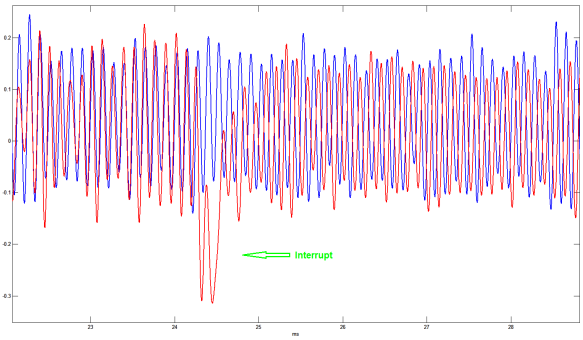


(a) A segment of the demodulated trace. Squaring is marked by S and multiplication is marked by the corresponding table index u (here, 0xD or 0xF). Note that multiplications where $u = 0$xF cause dips. Iterations of the main loop of Algorithm 1 are marked by vertical black lines.

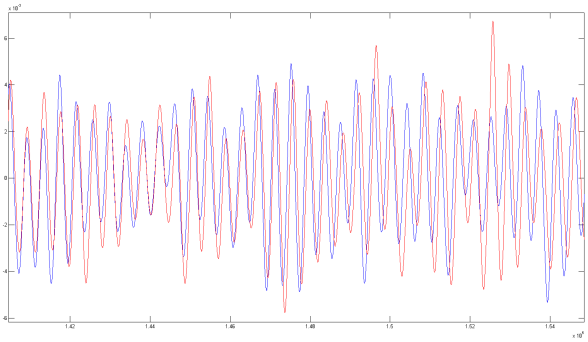


(b) Demodulation of the signal obtained during the entire decryption. The interrupts, occurring every 15 ms, are marked by green arrows.

Figure 5: Frequency demodulation of the first leakage signal from Figure 4. The exponent is overridden to be the 3072-bit number obtained by repeating the bit pattern 0xFD, and the ciphertext c is set to be such that $c^{15} \equiv 2^{3071} \pmod{p}$.



(a) Red signal shifted due to the interrupt

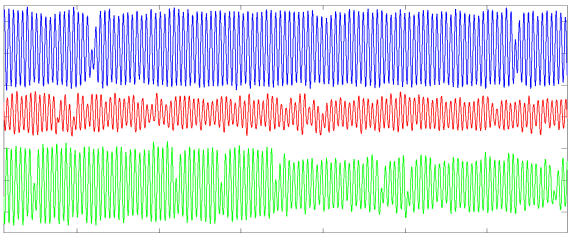


(b) Red signal drifted relative to blue signal

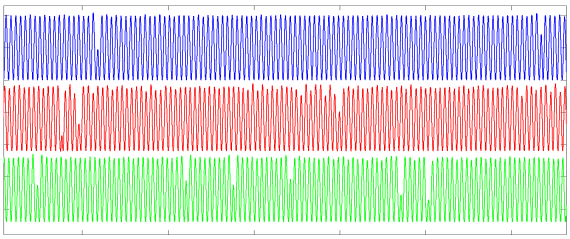
Figure 6: FM demodulation of an EM measurement around a carrier of 1.5 MHz during two ElGamal decryptions of the same ciphertext and same (randomly-generated) key.

remove the interrupt itself). In order to obtain a clean and disruption-free *aggregate trace*, the signals were combined and filtered via a mean-median filter hybrid. At each time point, the samples from all different traces were sorted, and then the highest and lowest several values were discarded. The remaining values were consequently averaged, resulting in an interrupt free trace (see Figure 7(a)).

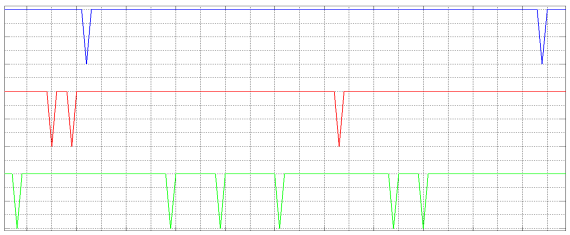
Note that even after we combined several aligned traces, the peak amplitudes across each aggregate trace varied greatly. To facilitate peak detection and thresholding, the peak amplitudes were equalized using the following procedure. First, an AM demodulation of the aggregate trace was performed by taking the absolute value of its Discrete Hilbert Transform. The result was then low-pass-filtered and smoothed using a Gaussian window, resulting in an outline of the envelope. The trace was then divided by its envelope to complete equalization. See Figure 7(b).



(a) Before peak amplitudes equalization (the horizontal axis is given in miliseconds)



(b) After peak amplitudes equalization (the horizontal axis is given in miliseconds)



(c) After peak detection (the horizontal axis is the peak/dip number and the vertical axis is "high" for peaks and "low" for dips)

Figure 7: Three aggregate traces, corresponding to table indices $u = 1,3,5$ obtained during our ElGamal attack using a randomly-generated key.

3.3 ElGamal Key Extraction

When attacking ElGamal following the method of Section 2.2, we first iterated over the 8 table indices, and for each measured and aggregated multiple traces of decryptions of that ciphertext. This resulted in 8 aggregate traces, which were further processed as follows.

Peak Detection. For each aggregate trace corresponding to a table index u, we derived a vector of binary values representing the peaks and dips in this trace. This was done by first detecting, in the aggregate trace, all local maxima exceeding some threshold amplitude. The binary vector then contains a bit for every consecutive pair of peaks, set to 1 if the peaks are close (below some time threshold), and set to 0 if they are further apart, meaning there is a dip between them; see Figure 7(c).

Revealing the ElGamal SM-sequence. Observing that dips occur during multiplication by operands having many zero limbs, coupled with the analysis of Section 2.2, we expect the 0 value to appear in this vector only at points corresponding to times when multiplication by g_u is performed.

Across all ciphertexts, these binary vectors allow the attacker to deduce the exact SM-sequence and, moreover, to obtain, for each multiplication performed by line 29 of Algorithm 1, the corresponding value of the table index u. As explained in Section 2.2, the key is then easily deduced.

Overall Attack Performance. Applying our attack to a randomly-generated 3072-bit ElGamal key by measuring the EM emanations from a Lenovo 3000 N200 laptop, we extracted all but the first bit of the secret exponent. For each chosen ciphertext, we used traces obtained from 40 decryption operations, each taking about 0.1 sec. We thus measured a total of $8 \cdot 40 = 320$ decryptions.

3.4 RSA Key Extraction

Analogously to the above, when attacking RSA following the method of Section 2.3, we first obtained 16 aggregate traces, one for each table index and its corresponding chosen ciphertext.

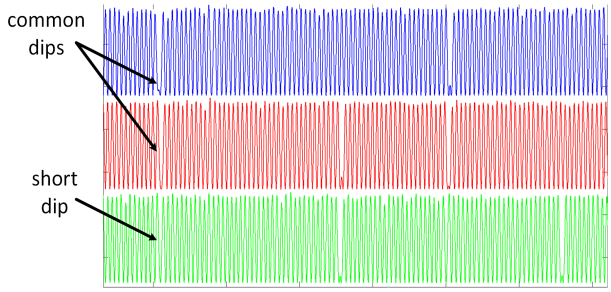
Peak Detection. As in the ElGamal case, for each aggregate trace corresponding to a table index u, we derived a vector of binary values representing the peaks and dips in this trace by detecting peaks above some amplitude threshold and comparing their distances to a time threshold. Figure 8(a) depicts some of the aggregated traces obtained during the RSA attack presented in Section 2.3. As predicted in Section 2.3, any dip first appearing in some trace corresponding to some table index u also appears in traces corresponding to table indices $u' > u$.

However, note that in each subsequent trace the distance between the two peaks defining the dip gets progressively shorter and therefore harder to observe. This is because the larger the value $u' - u$ is, the shorter the value stored in the u-th table index during the decryption of the ciphertext targeting the u'-th table index (and in particular this value contains less zero limbs). Eventually the distance between the two peaks defining a dip becomes indistinguishable from the regular distance between two peaks (with no dip in between), making the dip impossible to observe. Thus, the extracted vectors inevitably contain missing dips, requiring corrections as described next.

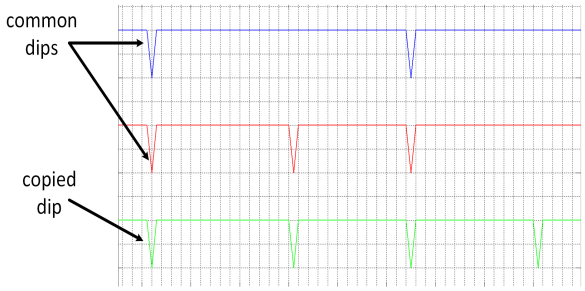
Inter-Window Dip Aggregation. In order to recover the undetected short dips, we had to align all the aggregate vectors (corresponding to the different table indices). Luckily, even though the dips get progressively shorter, in adjacent vectors (corresponding to table indices u and $u + 2$) there are sufficiently many common dips remaining to allow for alignment. Thus, the following iterative process was performed between every two adjacent vectors. First, the current vector was aligned to the previous one. Next, all missing dips were copied from the previous vector to the current one, as follows: going over the vectors from start to end, as soon as a dip was located in the previous vector that was missing from the current vector, it was copied to the current vector (shifting all other vector elements one coordinate to the right). The current vector was used for the next iteration. See Figure 8(b).

Revealing the RSA SM-Sequence. Note that each multiplication performed by Algorithm 1 corresponds to a dip in one of the binary vectors obtained in the previous stage. Thus, since in the above aggregation process dips are propagated across adjacent vectors, the last vector corresponding to table index 31 obtained after the aggregation process outlined above actually contains all the SM-locations, where each multiplication is marked with a dip and each squaring operation is marked with a peak. Thus, in order to recover the secret key, it remains for the attacker to learn the table index corresponding to every multiplication in the SM-sequence. Since each vector contains all the dips of all previous vectors, for each multiplication, the corresponding table index is the index of the vector where the dip appeared for the first time.

At this point the attacker has learned the exact SM-sequence and obtained, for each multiplication performed by line 29 of Algorithm 1, the corresponding value of the table index u. As mentioned in Section 2.3, it is possible to recover the secret key from this data.



(a) Before peak detection (the horizontal axis is given in miliseconds)



(b) After peak detection (the horizontal axis is the peak/dip number and the vertical axis is "high" for peaks and "low" for dips)

Figure 8: Three aggregate traces corresponding to table indices $u = 3,5,7$ obtained during our RSA attack using a randomly-generated key.

Overall Attack Performance. Applying our attack to a randomly generated 4096-bit RSA key by measuring the EM emanations from a Lenovo 3000 N200 laptop, we extracted the most-significant 1250-bits for d_p except for the first 5 bits.[4] For each chosen ciphertext, we used traces obtained from 40 decryption operations, each taking about 0.2 sec. We thus measured a total of $16 \cdot 40 = 640$ decryptions.

3.5 Long-Range Attack

Experimental Setup. We also attempted to expand the range of our electromagnetic attack. For simplicity, the experimental setup described in Section 3.1 does not contain an amplifier to amplify the probe signals before digitizing them using the FUNcube Dongle Pro+ SDR receiver. In order to extend the attack range, we added a 50dB gain stage using a pair of inexpensive low-noise amplifiers (Mini-Circuits ZFL-500LN+ and ZFL-1000LN+ in series, USD 175 total). We also added a low-pass filter before the amplifiers. See Figure 9.

Overall Attack Performance. Key extraction is possible with the antenna at a distance of half a meter from the target (the attacker's computer can be placed many meters away, connected by a coaxial cable). Recording the EM emanations from a Lenovo 3000 N200 laptop from this distance, our attack extracts the secret exponent of a randomly-generated 3072-bit ElGamal key (except for the first 3 bits, which are readily guessed). As in Section 3.3, we use a total of 320 decryptions, each taking about 0.1 sec.

[4]The first few bits of d_p are harder to measure, due to stabilization time.

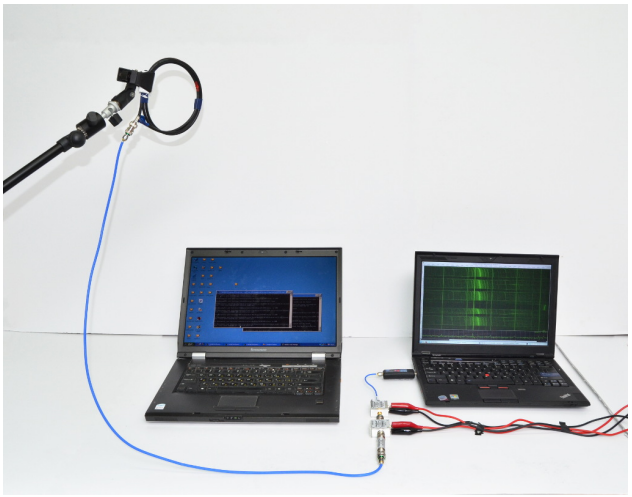


Figure 9: Long-range setup. The loop antenna is held half a meter above the target computer, Lenovo 3000 N200 (left). The antenna is connected via a coaxial cable (blue) to a low-pass filter, followed by a pair of amplifiers powered by a 15V DC voltage (red and black wires), leading to the SDR receiver dongle attached to the attacker's computer (right).

3.6 Untethered SDR Attack

The realization that the signal of interest is FM-modulated on a narrow bandwidth allowed us to greatly simplify and shrink the analog and analog-to-digital portion of the measurement setup, compared to prior works. One may thus wonder how small and cheap the whole setup can become. This section shows how the measurements can be fully acquired by a compact device, untethered to any wires. Our prototype, the Portable Instrument for Trace Acquisition (PITA), is built of readily-available electronics and food items (see Figure 10).

Functionality. The PITA can be operated in two modes. In *online mode*, it connects wirelessly to a nearby observation station via WiFi and provides real-time streaming of the digitized signal. The live stream helps optimize probe placement and allows adaptive recalibration of the carrier frequency and SDR gain adjustments (see Figure 11). In *autonomous mode*, the PITA is configured to continuously measure the electromagnetic field around a designated carrier frequency; it records the digitized signal into an internal microSD card for later retrieval, by physical access or via WiFi. In both cases, signal analysis is done offline, on a workstation.

Hardware. For compactness and simplicity, the PITA uses an unshielded loop antenna made of plain copper wire, wound into 3 turns of diameter 13 cm , with a tuning capacitor chosen to maximize sensitivity at 1.7 MHz (see Figure 10). These are connected to the aforementioned SDR receiver (FUNcube Dongle Pro+).

We controlled the SDR receiver using a small embedded computer, the Rikomagic MK802 IV. This is an inexpensive (USD 68) Android TV dongle based on the Rockchip RK3188 ARM SoC. It supports USB host mode, WiFi and flash storage. We replaced the operating system with Debian Linux in order to run our software, which operates the SDR receiver via USB and communicates via WiFi. Power was provided by 4 NiMH AA batteries, which suffice for several hours of operation.[5]

[5]The batteries take up most of the weight and volume. The apparatus can be made lighter and thinner by using a compact Li-Ion battery (e.g., a 700 mAh RCR-123 battery suffices for 1 hour), or a low-power embedded computer.

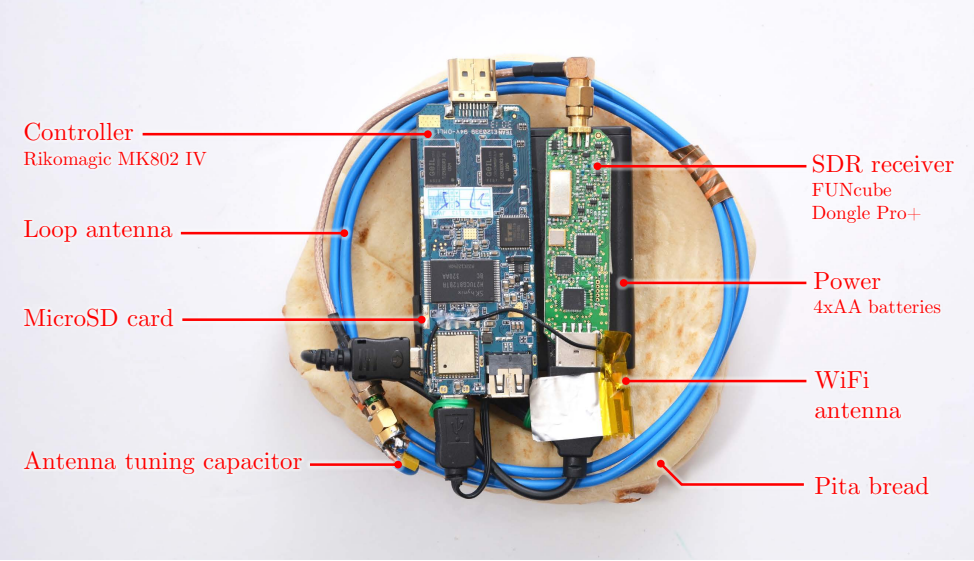


Figure 10: Portable Instrument for Trace Acquisition (Pita), a compact untethered measurement device for low-bandwidth electromagnetic key-extraction attacks.

Overall Attack Performance. Applying our attack to a randomly generated 3072-bit ElGamal key, we extracted all the bits of the secret exponent, except the most significant bit and the three least significant bits, from a Lenovo 3000 N200 laptop. As in Section 3.3, we used a total of 320 decryptions, taking 0.1 sec each.

3.7 Consumer-Radio Attack

Despite its low cost and compact size, assembly of the Pita device still requires the purchase of an SDR device. In this section, we show how to improvise a side-channel attack setup that extracts ElGamal keys from GnuPG, using common household items.

As discussed, the leakage signal is *frequency* modulated (FM) around a carrier (1.5–2 MHz) in the Medium Frequency band. While the required signal processing (frequency demodulation, filtering, etc.) can be easily performed in software, we could not find any household item able to digitize external signals at such frequencies. Since the frequency of the demodulated signal is only a few kHz, an alternative approach is to attempt to perform the FM demodulation in hardware and then digitize the resulting signal. While most household radio devices are capable of performing FM demodulation, the frequency range used in commercial FM broadcasting is 88–108 MHz, which is far outside the desired range. Within the commercial FM broadcasting band we did not observe key-dependant leakage even using lab-grade equipment. Despite this frequency range problem, we managed to use a plain consumer-grade radio receiver to acquire the desired signal, as decribed below, replacing the magnetic probe and SDR receiver. After appropriate tuning, all that remained was to record the radio's headphone jack output, and digitally process the signal. See Figure 13.

Demodulation Principle. Most consumer radios are able to receive amplitude modulated (AM) broadcasts in addition to the more popular FM. Commercial AM broadcasting typically uses parts of the Medium Wave band (0.5–1.7 MHz), in which our signal of interest resides. AM signals are received and routed through a completely different analog path than the FM signals, so the radio's internal FM demodulator cannot be used in these ranges. It *is* possible, however, to use the AM analog chain to perform unconventional FM demodulation. The AM path consists of an antenna,

Figure 11: Untethered measurement device in online mode. The PITA (handheld) measures the target computer (left) at a specific frequency band and streams the digitized signal over WiFi, in real time, to the attacker's computer (right). The attacker's computer can be many meters away when using a direct WiFi connection, or (if the PITA is configured to use a suitable WiFi access point) anywhere on the Internet.

a tuning filter, and an AM demodulation block. During normal operation, the tuning filter is set so that its center frequency exactly matches that of the incoming signal, in order to maximize reception quality. An FM signal received in this fashion would pass through the tuning filter unchanged but be completely suppressed by the AM block since the amplitude of an FM signal is essentially constant. By setting the center frequency of the tuning filter to be slightly (a few kHz) off the frequency of the incoming signal, the slope of the filter effectively acts as an FM to AM converter, transforming the frequency changes of the incoming signal into corresponding changes in amplitude. The amplitude demodulation circuits then extract and amplify these amplitude changes (while suppressing the still-present frequency deviations). See Figure 12.

Experimental Setup. This setup requires an AM radio receiver and an audio recorder (such as a smartphone microphone input or a computer's audio input). We used a plain hand-held radio receiver ("Road Master" brand) and recorded its headphone output by connecting it to the microphone input of an HTC EVO 4G smartphone, sampling at 48 Ksample/sec, through an adapter cable (see Figure 13).[6] The radio served as a front-end replacing the magnetic probe, SDR and digital demodulation.

Further Digital Signal Processing. The output of the radio's headphone jack produced a strong signal at around 8 kHz, which is similar to the frequency of the peaks Figure 6. After low-pass filtering it at 16 kHz, traces similar to Figure 6 were obtained (see Figure 14). We then applied the remainder of the signal processing algorithms discussed in Section 3.2.

Overall Attack Performance. Applying our attack to a randomly generated 3072-bit ElGamal key by measuring the EM emanations from a Lenovo 3000 N200 laptop, we extracted all but the first bit of the secret exponent. For each chosen ciphertext, we used traces obtained from 40 decryption operations, each taking about 0.1 sec. We thus measured a total of $8 \cdot 40 = 320$ decryptions. Similar results were obtained by directly connecting the radio's output to a computer's audio input, recording at 48 Ksample/sec.

[6] This adapter cable activates the microphone input of the phone, by presenting a 4.7kΩ DC resistance between the ground and microphone connectors in the phone's TRRS jack [Mic]. Dedicated line-in inputs of PCs and sound cards do not require an adapter.

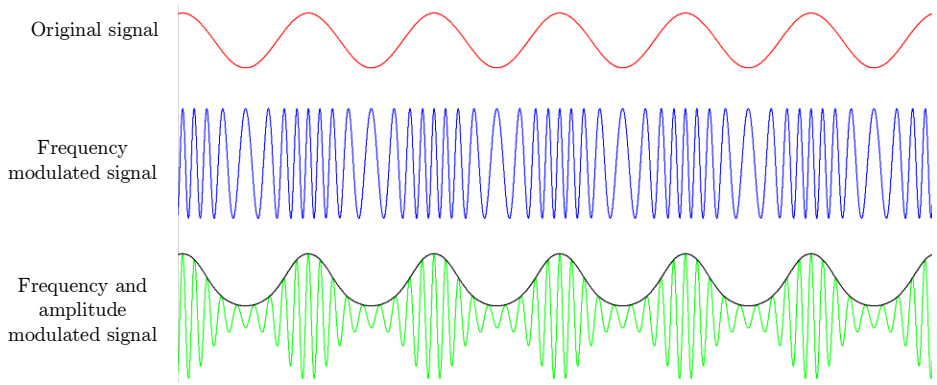


Figure 12: Illustrating FM to AM conversion using the AM tuning filter slope. The top (red) signal is some periodic baseband signal. The middle (blue) signal is an FM modulation of the red signal around some carrier f_c. The bottom (green) signal is obtained by filtering the FM-modulated signal through a slightly skewed bandpass filter, such that f_c falls on the filter's positive slope; the resulting signal is modulated in both amplitude and frequency. Note that the original baseband signal can now be reconstructed by extracting the envelope of the resulting signal (black). (For visual clarity, we compensate for the filter's time delay and attenuation.)

Figure 13: The radio-based experimental setup attacking the Lenovo 3000 N200 target. The radio receiver is placed near the target and tuned to approximately 1.5 MHz. The radio's output is connected, through the adapter cable, to the input of an HTC EVO 4G smartphone recording at 48 Ksample/sec.

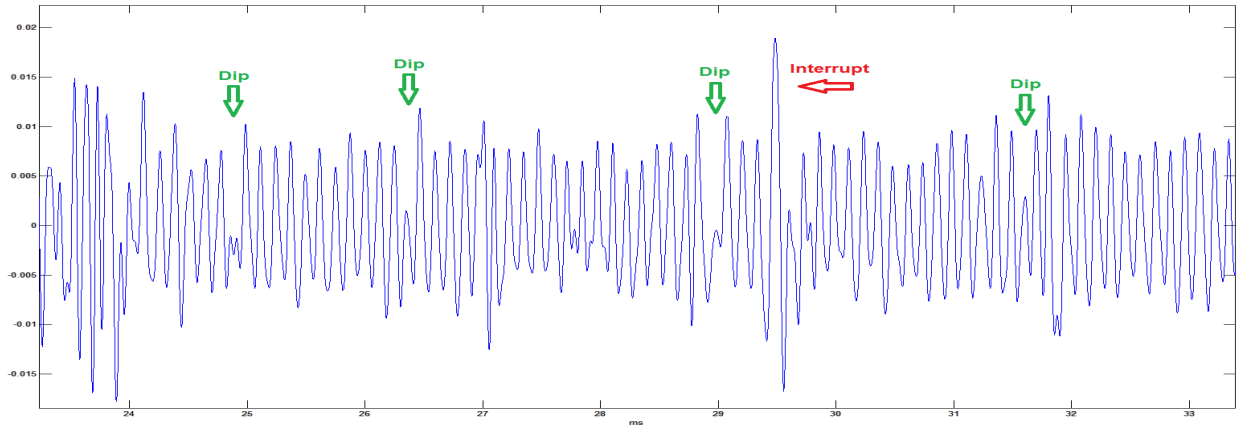


Figure 14: A recording of the radio's output, after 16 kHz low-pass filtering, during an ElGamal decryption using an HTC EVO 4G smartphone and an adapter cable.

4 Discussion

In this paper we presented and experimentally demonstrated new side-channel attacks on cryptographic implementations using sliding-window and fixed-window exponentiation. Our techniques simultaneously achieve low analog bandwidth and high attack speed, and the measurement setup is compact, cheap and unintrusive.

The attack does not rely on detecting movement of the sliding window (as in [FKM+06, CRI12]), but rather detects the key-dependent use of specific table entries "poisoned" by chosen ciphertext. Thus, the attack is also applicable to exponentiation algorithms that have a fixed schedule of squarings and multiplications, such as the fixed-window method. Likewise, it is oblivious to cache-attack mitigations that read the whole table, or that place all cache entries in addresses mapped to the same cache set.

Software Countermeasures. Our attack chooses ciphertexts that target specific table indices. For a targeted table index, the attacker learns the locations of the index in the sequence of squarings and multiplications. Since the sequence of squarings and multiplications only depends on the secret exponent, the attacker is able to reconstruct the secret exponent after recovering the location of all table indices in the sequence of squarings and multiplications. One class of countermeasures is an *exponent randomization*, which alters the sequence of squarings and multiplications between invocations of the modular exponentiation.

Multiplicative Exponent Randomization. Given a base x and a secret exponent d, instead of directly computing $y = x^d \bmod p$, one can generate a random number r and compute $y = x^{d+r(p-1)} \bmod p$. Notice that since $\left(x^{r(p-1)}\right) \equiv 1 \pmod{p}$, it holds that $y \equiv x^d \pmod{p}$. Since r is generated afresh for each exponentiation, the attacker cannot combine different executions in order to recover the exponent d. See [Koc96] for a complete description.

Unfortunately, this countermeasure would incur significant performance overheads in the case of GnuPG's ElGamal implementation. While p is a 3072-bit number, the secret exponent d (i.e., $-\chi$ in the notation of Section 1.5) is chosen to be approximately 400 bits long. Thus, instead of computing $y = x^d \bmod p$ directly using a 400-bit exponent, the computation of $x^{d+r(p-1)}$ will use an exponent of size 3072 bits, incurring a multiplicative slowdown of about $\times 3072/400 \approx 7$.

Additive Exponent Randomization. A cheaper exponent randomization alternative is *additive exponent randomization* (see [CJRR99, CJ01] and a related patent [Gou05]). The exponent is additively divided into two shares and the modular exponentiation is performed separately on each of them and eventually combined. Given a base x, instead of directly computing $y = x^d \bmod p$, one can generate a random number r and compute $y_1 = x^{(-d-r)} \bmod p$ and $y_2 = x^r \bmod p$. Finally, y is recovered by computing $y = y_1 \cdot y_2 \bmod p$. In GnuPG's ElGamal implementation, this requires two modular exponentiations with 400 bit exponents, incurring a $\times 2$ slowdown.

Ciphertext Randomization. Another countermeasure that generally blocks chosen ciphertext attacks such as ours is *ciphertext randomization*, which randomizes the base of the modular exponentiation and then cancels out the randomization.

For RSA decryption, this is a common countermeasure with low overhead: instead of decrypting a given ciphertext c by directly computing $c^d \bmod n$, one generates a random $r \in \mathbb{Z}_n$, computes r^e (which is cheap since the RSA encryption exponent e is small, typically 65537), decrypts $r^e \cdot c$ and finally divides by r to obtain c^d. Current versions of GnuPG already employ this countermeasure for RSA decryption, preventing the attack described in Section 2.3.

For ElGamal decryption, ciphertext randomization is more expensive. Given a ciphertext (γ, δ), a secret exponent χ and a prime p, instead of computing $\gamma^{-\chi} \cdot \delta \bmod p$ directly, one generates a random $r \in \mathbb{Z}_p^*$ and computes $y_1 = r^\chi \bmod p$, $y_2 = (\gamma \cdot r)^{-\chi} \bmod p$ and finally $y_1 \cdot y_2 \cdot \delta \bmod p$. In the case of GnuPG's ElGamal, this requires two modular exponentiations with 400-bit exponents, plus an inversion operation, incurring again a $\times 2$ slowdown. A new version of GnuPG, implementing this countermeasure, was released concurrently with the public announcement of our results.

Acknowledgments

We thank Werner Koch, lead developer of GnuPG, for the prompt response to our disclosure and the productive collaboration in adding suitable countermeasures. We thank Sharon Kessler for editorial advice.

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".

References

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In *CHES*, pages 29–45, 2002.

[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.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES. 2005. URL: `http://cr.yp.to/papers.html#cachetiming`.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *Information and Communications Security - ICICS 2010*, pages 46–61. Springer, 2010.

[CJ01] Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In *CHES*, pages 300–308. Springer, 2001.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, pages 398–412. Springer, 1999.

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

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

[CRI12] SPA/SEMA vulnerabilities of popular rsa-crt sliding window implementations, 2012. CHES rump session. URL: `https://www.cosic.esat.kuleuven.be/ches2012/ches_rump/rs5.pdf`.

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

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

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

[FKM+06] Pierre-Alain Fouque, Sébastien Kunz-Jacques, Gwenaëlle Martinet, Frédéric Muller, and Frédéric Valette. Power attack on small RSA public exponent. In *CHES*, pages 339–353, 2006.

[Fun] FUNcube Dongle. URL: `http://www.funcubedongle.com`.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: concrete results. In *CHES*, pages 251–261, 2001.

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

[Gnu] GNU Radio. URL: `http://gnuradio.org`.

[Gou05] L. Goubin. Method for protecting an electronic system with modular exponentiation-based cryptography against attacks by physical analysis, 2005. US Patent 6,973,190.

[Gpga] GNU Privacy Guard. URL: `https://www.gnupg.org`.

[Gpgb] GnuPG Frontends. URL: `https://www.gnupg.org/related_software/frontends.html`.

[GPT14] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs. In *CHES*, pages 242–260, 2014.

[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. Extended version of [GST14].

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO*, pages 444–461 (vol. 1), 2014. See [GST13] for extended version.

[HIM+13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering algorithms for non-profiled single-execution attacks on exponentiations. In *Smart Card Research and Advanced Applications - CARDIS 2013*, pages 79–93, 2013.

[HMA+08] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-based power analysis of modular exponentiation using chosen-message pairs. In *CHES*, pages 15–29, 2008.

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

[KO62] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of the USSR Academy of Sciences*, 145:293–294, 1962.

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

[Mic] Making your own smartphone cable. URL: `http://en.wiki.backyardbrains.com/index.php?title=Making_your_own_Smartphone_Cable`.

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

[MIT14] MITRE. Common vulnerabilities and exposures list, entry CVE-2014-3591, 2014. URL: `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3591`.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks — Revealing the Secrets of Smart Cards*. Springer, 2007.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

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

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA*, pages 1–20, 2006.

[Per05] Colin Percival. Cache missing for fun and profit. Presented at BSDCan, 2005. URL: `http://www.daemonology.net/hyperthreading-considered-harmful`.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *E-smart'01*, pages 200–210, 2001.

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

[RTL] Soft66RTL. URL: `http://zao.jp/radio/soft66rtl`.

[Smi99] D.C. Smith. Signal and noise measurement techniques using magnetic field probes. In *IEEE International Symposium on Electromagnetic Compatibility*, volume 1, pages 559–563, 1999.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *CT-RSA*, pages 104–119. Springer, 2011.

[Wal01] Colin D. Walter. Sliding windows succumbs to Big Mac attack. In *CHES*, pages 286–299, 2001.

[YF14] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732, 2014.

[YLG+15] Yuval Yarom, Fangfei Liu, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[ZP14] A. Zajic and M. Prvulovic. Experimental demonstration of electromagnetic information leakage from modern processor-memory systems. *IEEE Transactions on Electromagnetic Compatibility*, 56(4):885–893, 2014.