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

Daniel Genkin[1,2], Lev Pachmanov[2], Itamar Pipman[2], and Eran Tromer[2(✉)]

[1] Technion, Haifa, Israel
danielg3@cs.technion.ac.il
[2] Tel Aviv University, Tel Aviv, Israel
{levp,itamarpi,tromer}@tau.ac.il

Abstract. We present new side-channel attacks on RSA and ElGamal implementations that use sliding-window or fixed-window (m-ary) modular exponentiation. The attacks 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 (unmodified ElGamal and non-blinded RSA), within 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.

Keywords: Side channel · Electromagnetic analysis · RSA · ElGamal

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 and have been exploited by attacks on many cryptographic implementations (see [8,25,27] for surveys). Most research on physical side-channel attacks has focused on small devices such as smartcards, FPGAs and other simple embedded hardware. On general-purpose PCs (laptop and desktop computers, servers, etc.), software-based side-channel attacks on PCs (e.g.,

T. Güneysu and H. Handschuh (Eds.): CHES 2015, LNCS 9293, pp. 207–228, 2015.
DOI: 10.1007/978-3-662-48324-4_11

exploiting timing and CPU cache contention) have been extensively studied. But physical side channels in PCs received less academic attention, and involve 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 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 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 attacks on PCs have been reported only at a low bandwidth leakage (less than a MHz). Emanations of interest have been shown at the USB port [30] and through the power outlet [12]. Recently, low-bandwidth physical side-channel *key-extraction* attacks on PCs were demonstrated [20,21], 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 / 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.

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.

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

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	[21]
RSA, ElGamal	Square and always multiply	Non-adaptive	1	seconds	2 MHz	Lab-grade	[20]
RSA	Sliding/fixed window	Non-adaptive	16	seconds	2 MHz, 100 kHz bandwidth	Common	This work
ElGamal			8				

Another limitation of [20,21] is that they target the square-and-multiply algorithm. These attacks do not work for sliding-window or fixed-window exponentiation, used in most RSA and ElGamal implementations nowadays.

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 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. **Speed.** 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 laptop or a small SoC board (see Figs. 8(a) and 7). Alternatively, in some cases all that is required is a common, consumer-grade radio, with its audio output recorded by a phone (see Fig. 8(b)). 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 EM probes.

Approach. Our attack utilizes the fact that, in the sliding-window or fixed-window exponentiation, the values inside the table of ciphertext powers can be partially predicted. Using a suitable ciphertext, we cause the value at a specific

table entry to have a certain structure. This structure, coupled with a subtle control flow difference deep inside GnuPG's multiplication code, causes a difference in the leakage whenever a multiplication by this structured value occurs. Such a ciphertext is crafted separately for each table index. During its decryption, we nonintrusively measure the EM leakage, focusing on a narrowband frequency-modulated signal around 1.5–2 MHz. After filtering, demodulation, distortion compensation and averaging, a clean aggregate trace is produced for that table index, revealing all the locations inside the secret exponent where the specific table entry is selected by the bit pattern in the window. We then recover the key by combining the (misaligned but partially-overlapping) aggregate traces.

1.3 Vulnerable Software and Hardware

Similarly to [20,21], this work targets commodity laptop computers. We have tested numerous laptops of various models and makes. In this paper our examples use Lenovo 3000 N200 laptops, which exhibit a particularly clear signal.

GnuPG. We focused on GnuPG 1.4.18 [3], which is the latest version at the time of writing this paper. We compiled GnuPG using the MinGW GCC 4.6.2 [5] and ran it on Windows XP.[1] 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 [20,21], GnuPG uses ciphertext randomization for RSA (but not for ElGamal; see Sect. 4). To test our attack on RSA with sliding-window exponentiation, we disabled that countermeasure, making GnuPG decrypt the ciphertext directly. The ElGamal attack applies to unmodified GnuPG.

Current Status. We worked with the authors of GnuPG to suggest several countermeasures and verify their effectiveness (see CVE-2014-3591 [29]). 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 external inputs, from numerous frontends, via emails, files, chat and web pages. The list of GnuPG frontends [4] contains dozens of such applications, each of them can be potentially exploited for our attack. Concretely, as observed in [20,21], Enigmail [17], a plugin for the Mozilla Thunderbird e-mail client, automatically decrypts incoming emails, passing them to GnuPG. Thus, it is possible to inject ciphertexts into GnuPG by sending them as a PGP/MIME-encoded e-mail [16].

1.4 Related Work

Side-channel attacks have been demonstrated on numerous cryptographic implementations, via various channels (see [8,25,27] and the references within).

[1] Similar effects where observed on other version of Windows as well as on Linux.

EM Side Channel. The electromagnetic (EM) side channel has been exploited for attacking smartcards and other small devices (e.g., [7,19,33]). On PCs, [39] observed EM leakage (but did not show cryptanalytic applications), and [20] demonstrated EM attacks on a side-channel protected PC implementation of the square-and-multiply exponentiation of RSA and ElGamal.

Attacks on Sliding Window Exponentiation. While most attacks on public key schemes focus on variants of the square-and-multiply algorithm, several focus on attacking sliding window exponentiation on small devices (sampling much faster than the target's clockrate). These either exploit high-bandwidth operand-dependent leakage of the multiplication routine [13,23,24,35] or utilize the fact that it is possible to distinguish between squarings and multiplications [6,18].

Neither of the above approaches fits our case. The first approach requires very high-bandwidth leakage the acquisition of which, even for small and slow embedded devices, requires expensive lab equipment. Non-intrusive acquisition of such signals for the PC class of devices (running multi-GHz CPUs) is especially difficult. The second approach is blocked by a countermeasure to the attack of [37]: 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 [30] or through the power outlet [12]. Key extraction attacks have been presented on PCs, utilizing timing differences [10] and cache access patterns [9,31,32]. Recently, low-bandwidth key-extraction attacks that utilize physical channels such as sound [21] and chassis potential [20] were demonstrated on GnuPG running on PCs.

Cache Attacks in GnuPG. Yarom and Falkner [37] 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 [38] an attack on sliding-window exponentiation by observing the access patterns to the table of ciphertext powers.

2 Cryptanalysis

2.1 GnuPG's Sliding-Window Exponentiation Routine

GnuPG uses an internal mathematical library called MPI (based on GMP [2]) in order to perform the big-integer operations occurring in ElGamal and RSA. In recent versions, exponentiation is performed using a sliding-window algorithm. MPI stores big integers as arrays of *limbs* (32-bit words, in our case). Algorithm 1 is a pseudocode of the exponentiation routine. The function SIZE_IN_LIMBS(x) returns the number of limbs in the t-bit number x, namely $\lceil t/32 \rceil$.

Consider lines 8–12. For a fixed value of w, these compute a table indexed by $1, 3, 5, \ldots, 2^w - 1$, mapping each odd w-bit integer u to the group element g^u. 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.

Algorithm 1. GnuPG's modular exponentiation (simplified).

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

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 multiplications performed in lines 18, 22 and 25 of Algorithm 1. Note that this sequence depends only on the exponent d. If an attacker were to learn the SM-sequence, and moreover obtain for each multiplication in line 22 the corresponding table index u used to index into the table, then the exponent could be recovered.

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 22, occur inside the SM-sequence. In what follows, for any given table index u, we shall refer to such locations as SM-locations. For 3072-bit ElGamal, GnuPG chooses a secret exponent of about 400 bits. Thus $w = 4$, so the table indices are odd 4-bit integers. 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 $x^u \equiv y \pmod{p}$. 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 multiplications by g_u from multiplications by $g_{u'}$ for all $u' \neq u$, during the main loop of $\text{MOD\_EXP}(x, d, p)$. Indeed, by the code of Algorithm 1 we have that $g_u = x^u \bmod p = y$, which contains many zero limbs. Conversely, for any $u' \neq u$, $g_{u'} = x^{u'} \bmod p$ which contains 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 [20,21]. Thus, it is possible to distinguish the multiplications by g_u in line 22 of Algorithm 1 from multiplications by $g_{u'}$ where $u' \neq u$.

Distinguishing Between Squarings and Multiplications. GnuPG implements the squaring in lines 18 and 25 using the same multiplication code used for line 22 (this is a countermeasure to the attack of [37]). In the case of squaring, the argument a supplied to the multiplication routine is an 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 $u' \neq u$. Thus the attacker can still determine the SM-locations of g_u.

Key Extraction. Applying this method across all ciphertexts, the attacker learns the SM-locations of multiplications by g_u performed in line 22 for all possible u. Since u is an odd 4-bit number, only 8 possible values of u exist. Any remaining locations must correspond to a squaring operation performed by lines 18 or 25. At this point, the attacker has learned the entire SM-sequence performed by Algorithm 1 and obtained the corresponding value of u for each multiplication performed by line 22, allowing him to recover the secret exponent.

Attacking the Fixed-Window Method. The fixed-window (m-ary) exponentiation method (see [28, 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 handled in turn by performing m squaring operations and a single multiplication by an appropriate value selected from a precomputed table using the current 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

Unlike ElGamal, the security of RSA already breaks down if the top half of the bits of any of the secret exponents (d_p, d_q) is leaked [15]. We now adapt the ElGamal attack presented in Sect. 2.2 to RSA, first for GnuPG's sliding window implementation, and then for the fixed window version.

Revealing the Location of Table Indices. For 4096-bit RSA since GnuPG's RSA uses the CRT, the exponents d_p and d_q 2048 bits long; hence, $w = 5$. Given an odd 5-bit table index u, the attacker wishes to learn the SM-locations of multiplications by g_u performed during the modular exponentiation routine. Unlike for ElGamal, the attacker does not know p and cannot select a number y with many zero limbs and compute x such that $x^u \equiv y \pmod{p}$. Neither can he 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. Instead, we relax the requirements so that, given a 5-bit odd integer u, the attacker learns 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 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 11 of Algorithm 1). Thus, given a 5-bit odd integer u, the attacker requests 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 Sect. 2.2, GnuPG implements the squaring in lines 18 and 25 using the same multiplication code used for line 22. In the case of squaring, the argument a supplied to the multiplication routine is a random-looking intermediate value. Thus, the squaring operations will produce similar leakage to case 2 above.

Next, as in the attack presented in Sect. 2.2, since the leakage produced by GnuPG's multiplication routine depends on the number of zero limbs in its

second operand, it is possible to distinguish between multiplications by $g_{u'}$ where $u' \leq u$ (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. Applying the above for every table index u (since u is an odd 5-bit integer, only 16 possible values of u exist). The attacker deduces the SM-locations of all multiplication performed by line 22. Moreover, for each multiplication, by finding the lowest u such that the leakage of the multiplication corresponds to case 1 above, the the index of its second operand is also deduced.

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 18 and 25 of Algorithm 1, and can thus be deduced from the SM-locations of the squaring operations in lines 18 and 25. These SM-locations are simply the remaining SM-locations after accounting for those of the multiplications in line 22, already identified above. The attacker has now learned the position and value of all bits in 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.

3 Experimental Results

3.1 SDR Experimental Setup

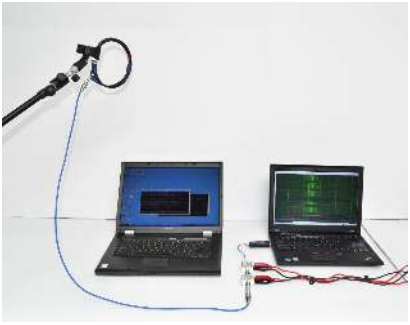
Our first setup uses Software Defined Radio to study EM emanations from laptop computers at frequencies of 1.5–2 MHz, as detailed below (see also Fig. 1(a)).

Probe. As a magnetic probe, we constructed a simple shielded loop antenna using a coaxial cable, wound into 3 turns of 15 cmdiameter, and with suitable conductor soldering and center shield gap [34]. The placement of the EM probe relative to the laptop influences the measured signal. We measured the EM emanations close to the CPU's voltage regular, located on the laptop's motherboard, yet without case intrusion. The voltage regulator is typically located in the rear left corner, and placing the probe there usually yields the best signal.

Receiver. We recorded the signal using a FUNcube Dongle Pro+ [1] SDR receiver. The FUNcube Pro+ is an inexpensive (GBP 125) USB dongle that contains a software-adjustable mixer and a 192 Ksample/secADC.

Amplification. In order to extend the attack range, we added a 50dB gain stage using a pair of inexpensive amplifiers (Mini-Circuits ZFL-500LN+ and ZFL-1000LN+ in series, USD 175 total) between the loop antenna and the SDR receiver. We also added a low-pass filter before the amplifiers. See Fig. 1(b).

(a) A loop antenna (handheld) attacking a target laptop (left). The antenna is connected to an SDR receiver dongle attached to the attacker's computer (right).



(b) The loop antenna is held 50cm above the target, and is connected to a low-pass filter, followed by a pair of amplifiers, leading to the SDR receiver dongle and the attacker's computer.

Fig. 1. The SDR-based setup attacking a Lenovo 3000 N200 target.

Attack Range. Using our loop antenna and SDR receiver we achieved key extraction from a range of about 20 cm(Figs. 1(a) and 8(a)). Using a cheap mini-circuits amplifiers we extended the attack range to half a meter (Fig. 1b).

3.2 Signal Analysis

Exponent-Dependent Leakage. Confirming the dependence of leakage on exponents, Fig. 2 shows how ElGamal decryptions using different exponents can be distinguishable by their EM leakage. The same holds for RSA.

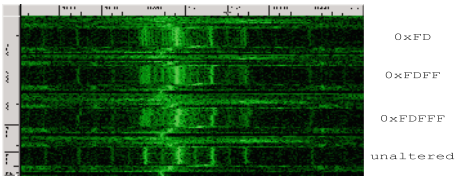
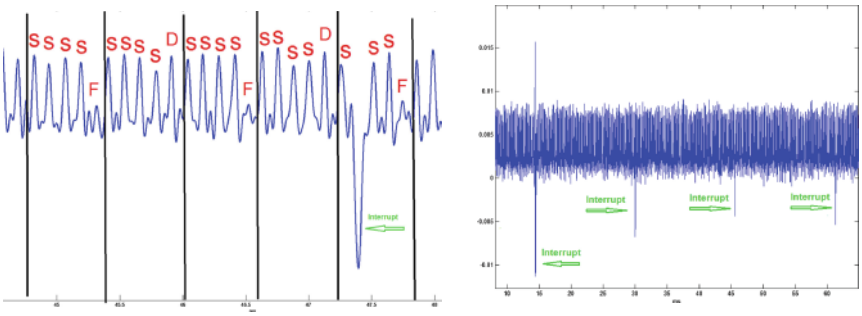


Fig. 2. EM measurement (0.5 sec, 1.49–1.57 MHz) of four GnuPG ElGamal decryptions executed on a Lenovo 3000 N200 laptop. The exponent is overridden to be the 3072-bit number obtained by repeating the bit pattern written to the right. In all cases, the modulus p is the same and the ciphertext c is set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

Demodulation. As can be seen in Fig. 2, when using periodic exponents the leakage signal takes the form of a central peak surrounded by distinguishable side



(a) A segment of the demodulated trace. Squaring is marked by S, and multiplication by the used table index u (here, 0xD or 0xF). When $u = 0xF$, dips occur. Algorithm 1 main-loop iterations are marked by vertical lines.

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

Fig. 3. Frequency demodulation of the first leakage signal from Fig. 2. 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 $c^{15} \equiv 2^{3071} \pmod{p}$.

lobes. This is a strong indication that the secret bit exponents are modulated by the carrier. As in [20], the carrier signal turned out to be frequency modulated.

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. We then sampled the signal using our SDR, and performed digital signal demodulation. After additional digital filtering, we received a demodulated trace as shown in Fig. 3(a).

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

Moreover, there are various distortions in the signal, making key extraction difficult. The signals are corrupted every 15 msec, by the timer interrupt on the target laptop. Each interrupt corrupts the trace for a duration of several bits, and may also create a time shift relative to other traces (see Figs. 3(b) and 4). Also, traces exhibit a gradual drift, increasing the duration between two adjacent peaks (relative to other traces), making signal alignment even more problematic.

The attack of [20], 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. Since, in their case, the baseband signal reflected a sequence of random-looking key bits, correlation proved sufficient aligning trace segments.

However, this approach is inadequate for our attack. Here, the demodulated traces are mostly periodic, consisting of similar peaks that change when the corresponding table index is used. Correlating such signals produces an ambiguity as to the actual shift compensation required for proper alignment. The problem is exacerbated by the low bandwidth of the attack: had we performed clockrate-scale measurements, consecutive peaks would likely have been distinguishable due to fine-grained data dependency, making alignment via correlation viable.

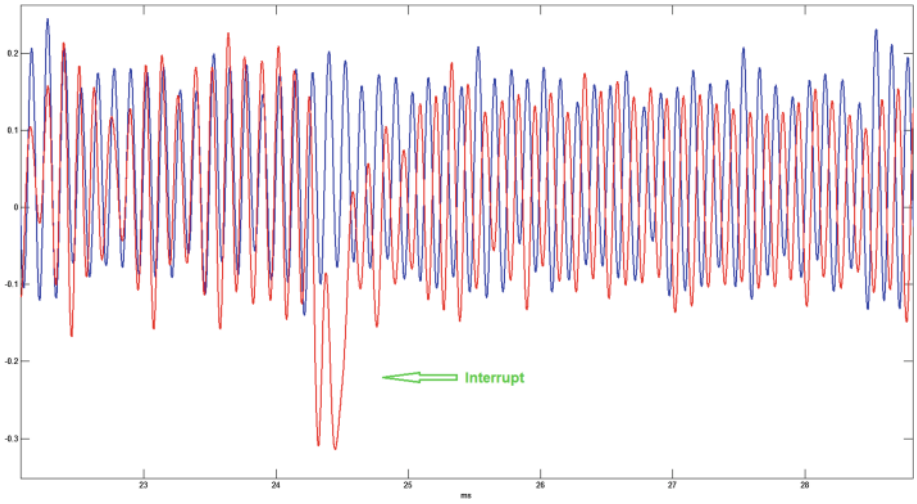


Fig. 4. FM demodulation of an EM measurement during two ElGamal decryptions of the same ciphertext and key. The red signal is shifted relative to the blue signal (Color figure online).

Aligning the Signals. As a first attempt to align the signals and correct distortions, we applied the "Elastic Alignment" [36] 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.

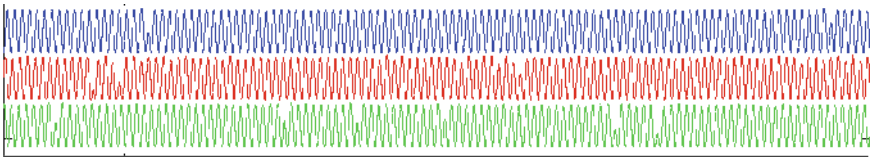
Initial Synchronization. We first aligned all traces belonging to decryptions of the same ciphertext. We used the fact that the computations performed just prior to the exponentiation produced a consistent pattern at the start of each trace. Correlation was used to align this pattern in all traces relative to a reference trace, chosen randomly from the trace set. Next, we independently compared each trace to the reference trace, correcting distortions as follows.

Handling Interrupts. In order to align the signals despite the interrupt-induced shifts, a search for interrupts was performed across both the current and reference trace, from beginning to end. Interrupts are easily detected, as they cause large frequency fluctuations. Whenever an interrupt was encountered, the

delay it induced was estimated by correlating a short segment immediately following the interrupt in both signals. The samples corresponding to the interrupt were then removed from the interrupted signal, to locally restore alignment. This process was repeated until 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 interrupts, we performed a periodic comparison (by direct correlation) and compensated for the drift by removing samples from the appropriate signal. In order to avoid ambiguity, the comparisons were made frequently so that the drift never created a delay longer than half a peak.

Aggregating Aligned Traces. The foregoing process outputs fully-aligned 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 remove the interrupt). In order to obtain a clean *aggregate trace*, the signals were combined via a mean-median filter. At each time point, the samples from different traces were sorted, and the highest and lowest several values discarded. The remaining values were averaged, resulting in an interrupt free trace. Finally, even after we combined several aligned traces, the peak amplitudes across each aggregate trace varied greatly. To facilitate peak detection, the peak amplitudes were equalized by extracting the signal envelope, followed by low-pass filtering and smoothing. See Fig. 5(a).



(a) After peak amplitudes equalization (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)

Fig. 5. Aggregate traces for table indices 1,3,5 obtained during our ElGamal attack.

3.3 ElGamal Key Extraction

When attacking ElGamal, 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 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 Fig. 5(b).

Revealing the ElGamal SM-sequence. Observing that dips occur during multiplication by operands having many zero limbs, coupled with the analysis of Sect. 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 22 of Algorithm 1, the corresponding value of the table index u. As explained in Sect. 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 three bits 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.

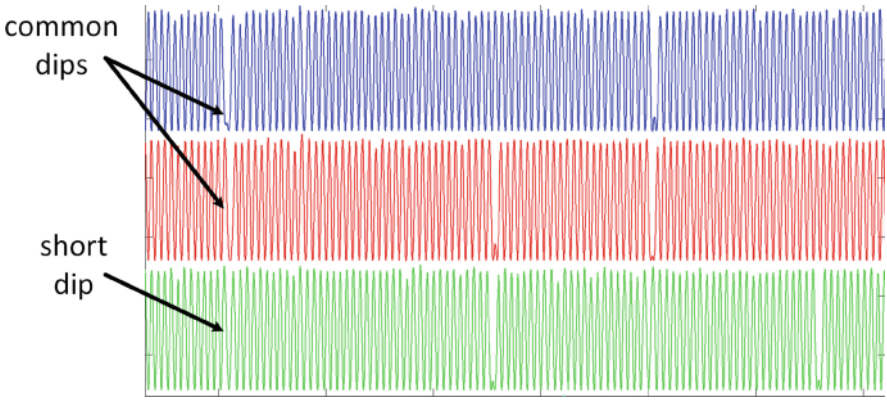
3.4 RSA Key Extraction

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

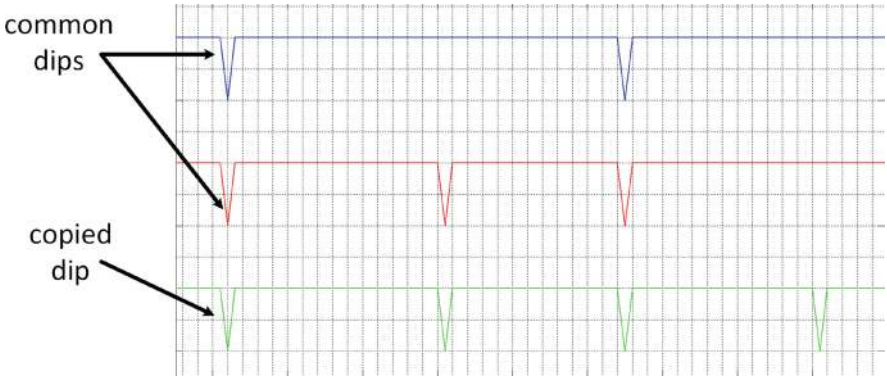
Peak Detection. As in the ElGamal case, for each aggregate trace corresponding to a table index u, we derived a vector of binary values representing the peaks and dips in this trace by detecting peaks above some amplitude threshold and comparing their distances to a time threshold. Figure 6(a) depicts some of the aggregated traces obtained during the RSA attack presented in Sect. 2.3. As predicted in Sect. 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 length of each dip gets progressively shorter and 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 dips become so short as to be 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 dips, we had to align all the aggregate vectors (corresponding to different table indices). Luckily, even though the dips get progressively shorter, in adjacent vectors there are many common dips to allow for alignment. Thus, the following iterative process was performed for every two adjacent vectors: First, the current vector



(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)

Fig. 6. Aggregate traces for table indices 3,5,7 obtained during our RSA attack.

was aligned to the previous one. Next, going from left to right, 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. See Fig. 6(b).

Revealing the RSA SM-sequence. Note that each multiplication corresponds to a dip in one of the vectors obtained in the previous stage. Thus, since in the above aggregation process dips are propagated across adjacent vectors, the last vector contains all the SM-locations, where each multiplication is marked with a dip and each squaring is marked with a peak. In order to recover the 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. Thus, as mentioned in Sect. 2.3, the attacker has all the data required to recover the 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. For each chosen ciphertext, we used traces obtained from 40 decryptions, each taking about 0.2 sec. We thus measured a total of $16 \cdot 40 = 640$ decryptions.

3.5 Untethered SDR Attack

The low bandwidth nature of our attack allows us to simplify and shrink the analog and analog-to-digital portion of the setup, compared to prior works. Our prototype, the Portable Instrument for Trace Acquisition (PITA), is built of readily-available electronics and food items (see Figs. 7 and 8(a)).

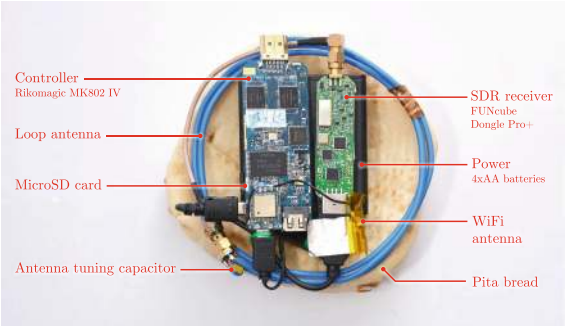


Fig. 7. Portable Instrument for Trace Acquisition (PITA), a compact untethered measurement device for low-bandwidth electromagnetic key-extraction attacks.

Functionality. The PITA can be operated in two modes. In *online mode*, it connects to a nearby 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 Fig. 8(a)). 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. The PITA uses an unshielded loop antenna made of copper wire, wound into 3 turns of diameter 13 cm, with a tuning capacitor chosen to maximize sensitivity at 1.7 MHz (see Fig. 7). These are connected to an SDR receiver

(a) Portable Instrument for Trace Acquisition (PITA), measuring the target and streaming the filtered signal over WiFi.

(b) Experimental setup using a consumer AM radio receiver, placed near the target and recorded by a smartphone.

Fig. 8. Two of our experimental setups for key extraction.

dongle. We controlled the SDR using a small embedded computer, the Rikomagic MK802 IV. This is an inexpensive (USD 68) Android TV dongle supporting USB host mode, WiFi and flash storage. We replaced the operating system with Linux in order to run our software, which operates the SDR receiver via USB and communicates via WiFi. Power was provided by 4 AA batteries.

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 before, we used a total of 320 decryptions, taking 0.1 seceach.

3.6 Consumer-Radio Attack

Despite its low cost and compact size, assembly of the PITA still requires the purchase of an SDR device. We now show how to improvise a side-channel attack setup for extracting ElGamal keys, using common household items.

As discussed, the leakage signal is *frequency* modulated (FM) around a carrier circa 1.7 MHz. While the signal processing can be performed in software, we could not find any household item able to digitize external signals at such frequencies. Since the bandwidth of the demodulated signal is only a few kHz, an alternative approach is to perform demodulation in hardware and then digitize the result. While most household radios can demodulate FM, the frequencies used in commercial FM broadcasting are 88–108 MHz. Even when using lab-grade equipment, we did not observe key-dependent leakage within the commercial FM band. Despite this, we managed to use a plain consumer-grade radio receiver to acquire the desired signal, as described below, replacing the magnetic probe and SDR. After appropriate tuning, all that remained was to record the radio's headphone jack output, and digitally process the signal. See Fig. 8(b).

Demodulation Principle. Most consumer radios are able to receive amplitude modulated (AM) broadcasts in addition to FM. AM radio broadcasts typically use parts of the Medium Wave band (0.5–1.7 MHz), in which our signal of interest resides. AM signals are routed through a different analog path than the FM signals, so the radio's 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, a tuning filter, and an AM demodulation block. In normal operation, the filter is set so that its center frequency exactly matches that of the incoming signal. An FM signal would pass through the tuning filter unchanged, but be completely suppressed by the AM block since its amplitude is essentially constant. But by setting the center frequency of the tuning filter to a few kHz away from the FM carrier, the slope of the filter effectively acts as an FM to AM converter, transforming the frequency changes of the signal into changes in amplitude. The AM demodulation block then extracts and amplifies these amplitude changes. See Fig. 9.

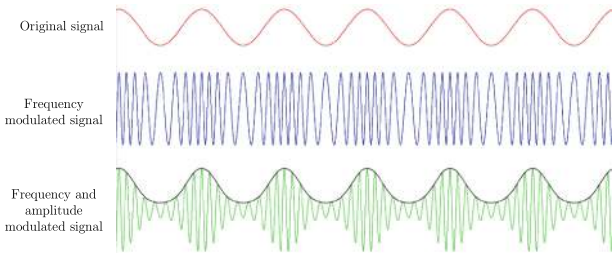


Fig. 9. FM to AM conversion using the AM tuning filter slope. The top signal is some periodic baseband signal. The middle signal is an FM modulation of the top signal. The bottom signal is obtained by filtering the FM-modulated signal through a slightly skewed bandpass filter; the resulting signal is both AM and FM modulated. The baseband signal can be reconstructed by extracting the envelope of the resulting signal. (For visual clarity, we compensate for the filter's time delay and attenuation).

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 phone, sampling at 48 Ksample/sec, through an adapter cable (see Fig. 8(b)). The radio replaced the magnetic probe, SDR and digital demodulation.

Further Digital Signal Processing. The radio's headphone jack produced a signal at 8 kHz, which is similar to the frequency of the peaks Fig. 4. After low-pass filtering at 16 kHz, traces similar to Figs. 3(a) and 4 were obtained. We then applied the remainder of the signal processing algorithms from Sect. 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. Similar results were obtained by directly connecting the radio's output to a computer's audio input, recording at 48 Ksample/sec.

4 Discussion

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, but detects the key-dependent use of specific table entries "poisoned" by a chosen ciphertext. Thus, the attack is 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 fix or randomize the table access patterns.

Future Work. In order to achieve key extraction, our attack requires traces obtained during 40 decryption operation for each chosen ciphertext. We leave the task of reducing the total number of required traces as an open problem.

Next, while public key encryption schemes were proven to vulnerable to physical low-bandwidth chosen-ciphertext key-extraction attacks on PCs, attacks on other public key primitives (such as digital signatures), as well as non-chosen ciphertext attacks were not yet demonstrated for the case of PCs.

Finally, physical key extraction attacks on symmetric key primitives were also never demonstrated for the PCs. Presumably since the frequencies of the externally-available leakage is too low given the high execution speed of such primitives. We leave the task of attacking such primitives as an open problem.

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. One such method is *multiplicative exponent randomization*, adding a random multiple of $p-1$ to the secret exponent (see [26]). Unfortunately, in the case of GnuPG's ElGamal implementation this incurs significant slowdown, since the secret exponent is selected to be short (about 400 bits), but adding a multiple of $p-1$ increases it to over 3072-bits, incurring a slowdown of $\times 3072/400 \approx 7$. A cheaper exponent randomization alternative is *additive exponent randomization*, in which the exponent is additively divided into two shares (see [11,14] and a related patent [22]). In GnuPG's ElGamal, this requires two exponentiations with 400 bit exponents, incurring a $\times 2$ slowdown.

Another countermeasure that generally blocks chosen ciphertext attacks is *ciphertext randomization (blinding)*, which randomizes the base of the exponentiation. For RSA decryption, this is a common countermeasure with low overhead: instead of decrypting a ciphertext c by directly computing $c^d \mod n$, one generates a random r, computes r^e (which is cheap since the encryption exponent e is small, typically 65537), decrypts $r^e \cdot c$ and divides by r to obtain c^d. Current versions of GnuPG already do this for RSA, preventing Sect. 2.3's attack.

For ElGamal, ciphertext randomization is more expensive. Instead of computing $\gamma^{-\chi} \cdot \delta \bmod p$ directly, one generates a random r and computes $y_1 = r^\chi \bmod p$, $y_2 = (\gamma \cdot r)^{-\chi} \bmod p$ and finally $y_1 \cdot y_2 \cdot \delta \bmod p$. For GnuPG's ElGamal, this requires two exponentiations with 400-bit exponents, plus an inversion, 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. and 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

1. FUNcube Dongle. http://www.funcubedongle.com
2. GNU multiple precision arithmetic library. http://gmplib.org/
3. GNU Privacy Guard. https://www.gnupg.org
4. GnuPG Frontends. https://www.gnupg.org/related_software/frontends.html
5. Minimalist GNU for Windows. http://www.mingw.org
6. SPA/SEMA vulnerabilities of popular RSA-CRT sliding window implementations. Presented During Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2012 Rump Session (2012). https://www.cosic.esat.kuleuven.be/ches2012/ches_rump/rs5.pdf
7. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side—channel(s). In: Kalisk, B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2002)
8. Anderson, R.J.: Security Engineering - A Guide to Building Dependable Distributed Systems, 2nd edn. Wiley, New York (2008)
9. Bernstein, D.J.: Cache-timing attacks on AES (2005). http://cr.yp.to/papers.html#cachetiming
10. Brumley, D., Boneh, D.: Remote timing attacks are practical. Comput. Netw. **48**(5), 701–716 (2005)
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)

12. Clark, S.S., Mustafa, H., Ransford, B., Sorber, J., Fu, K., Xu, W.: Current events: identifying webpages by tapping the electrical outlet. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 700–717. Springer, Heidelberg (2013)
13. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 46–61. Springer, Heidelberg (2010)
14. Clavier, C., Joye, M.: Universal exponentiation algorithm. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 300–308. Springer, Heidelberg (2001)
15. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J. Cryptol. **10**(4), 233–260 (1997)
16. Elkins, M., Torto, D.D., Levien, R., Roessler, T.: MIME security with OpenPGP. RFC 3156 (2001). http://www.ietf.org/rfc/rfc3156.txt
17. Enigmail Project, T.: Enigmail: A simple interface for OpenPGP email security. https://www.enigmail.net
18. Fouque, P.-A., Kunz-Jacques, S., Martinet, G., Muller, F., Valette, F.: Power attack on small RSA public exponent. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 339–353. Springer, Heidelberg (2006)
19. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
20. Genkin, D., Pipman, I., Tromer, E.: Get your hands off my laptop: physical side-channel key-extraction attacks on PCs. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 242–260. Springer, Heidelberg (2014)
21. Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 444–461. Springer, Heidelberg (2014)
22. Goubin, L.: Method for protecting an electronic system with modular exponentiation-based cryptography against attacks by physical analysis, US Patent 6,973,190 (2005)
23. Heyszl, J., Ibing, A., Mangard, S., De Santis, F., Sigl, G.: Clustering algorithms for non-profiled single-execution attacks on exponentiations. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 79–93. Springer, Heidelberg (2014)
24. Homma, N., Miyamoto, A., Aoki, T., Satoh, A., Shamir, A.: Collision-based power analysis of modular exponentiation using chosen-message pairs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 15–29. Springer, Heidelberg (2008)
25. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptograph. Eng. **1**(1), 5–27 (2011)
26. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
27. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, New york (2007)
28. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography, 1st edn. CRC Press Inc., Boca Raton (1996)
29. MITRE: Common vulnerabilities and exposures list, entry CVE-2014-3591 (2014). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3591

30. Oren, Y., Shamir, A.: How not to protect PCs from power analysis. Presented During CRYPTO 2006 Rump Session (2006). http://iss.oy.ne.ro/HowNotToProtectPCsFromPowerAnalysis
31. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
32. Percival, C.: Cache missing for fun and profit. Presented at BSDCan (2005). http://www.daemonology.net/hyperthreading-considered-harmful
33. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
34. Smith, D.: Signal and noise measurement techniques using magnetic field probes. In: IEEE International Symposium on Electromagnetic Compatibility (EMC 1999), vol. 1, pp. 559–563. IEEE (1999)
35. Walter, C.D.: Sliding windows succumbs to big mac attack. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 286–299. Springer, Heidelberg (2001)
36. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving differential power analysis by elastic alignment. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 104–119. Springer, Heidelberg (2011)
37. Yarom, Y., Falkner, K.: FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In: USENIX Security Symposium 2014, pp. 719–732. USENIX Association (2014)
38. Yarom, Y., Liu, F., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: IEEE Symposium on Security and Privacy (S&P). IEEE (2015)
39. Zajic, A., Prvulovic, M.: Experimental demonstration of electromagnetic information leakage from modern processor-memory systems. IEEE Trans. Electromagn. Compat. (EMC) **56**(4), 885–893 (2014)