Photonic Side Channel Attacks Against RSA

Elad Carmon*, Jean-Pierre Seifert†, Avishai Wool‡
*‡Tel-Aviv University, Tel-Aviv 69978, Israel
†Security in Telecommunications, Technische Universität Berlin, Germany

Email: *eladca@gmail.com, †Jean-Pierre.Seifert@telekom.de, ‡yash@eng.tau.ac.il

***Abstract*—This paper describes the first attack utilizing the photonic side channel against a public-key crypto-system. We evaluated three common implementations of RSA modular exponentiation, all using the Karatsuba multiplication method. We discovered that the key length had marginal impact on resilience to the attack: attacking a 2048-bit key required only 9% more decryption attempts than a 1024-bit key. We found that the most dominant parameter impacting the attacker's effort is the minimal block size at which the Karatsuba method reverts to naive multiplication: even for parameter values as low as 32 or 64 bits our attacks achieve 100% success rate with under 10,000 decryption operations. Somewhat surprisingly, we discovered that Montgomery's Ladder—commonly perceived as the most resilient of the three implementations to side-channel attacks—was actually the most susceptible: for 2048-bit keys, our attack reveals 100% of the secret key bits with as few as 4000 decryptions.**

I. Introduction

A. Background

While the phenomena of photonic emission from switching transistors in silicon is actually a very old one, cf. [9], [27], the role of photons in cryptography as a practical side channel source has just recently emerged as a novel research direction, cf. [33], [25], [24], [34], [7], [8]. Thus, it is important to include photonic side channels in future hardware evaluations of security ICs. In fact, a recent research tender from the BSI—the Federal Office for Information Security in Germany (cf. [3]) demands a Photonic Emission Analysis of a USB-based FIDO chip [1].

However, so far only the first steps within this direction have been successfully achieved. The work of [33], [25], [24], [34], [7], [8], showed that the required equipment to carry out successful SPEA (Simple Photonic Emission Analysis) or DPEA (Differential Photonic Emission Analysis) attacks against real world ICs is comparable in price to that of normal Power Analysis equipment and showed their validity in the AES case. The current paper continues the current state of the art of the Photonic Side Channel and utilizes it to attack the RSA encryption scheme for different (real-world) exponentiation implementations. In this respect the present paper is the first which considers the RSA case.

B. Related Work

Photonic emission analysis (PEA) in silicon is a known physical phenomena which has been studied since the 1950s [27]. In the failure analysis community, hot-carrier luminescence has primarily been used to characterize implementation and manufacturing faults and defects [11], [35]. Here, the technologies of choice to perform backside analysis are PICA (Picosecond Imaging Circuit Analysis) [4] and SSPDs (Superconducting Single Photon Detectors) [36]. Both technologies are able to capture photonic emissions with high performance in their respective field, but carry the downside of immense cost and complexity.

One of the first uses of PEA in CMOS in a cryptographic application was presented in 2008 [12]. However, the authors increased the voltage supply to 7V operating voltage, which is above the chips maximum limit for voltage. It took the authors 12 hours to recover a *single* potential key byte [12]. In 2011, an integrated PEA system and laser stimulation techniques were used to attack a DES implementation on an FPGA [10]. The authors proved that the optical side channel might be used for differential analysis. However, the analysis strongly relied on a specific implementation of DES in which registers were always zeroed before their use, and required the use of equipment valued at more than 2,000,000 Euros.

Nevertheless, recently, a real breakthrough was achieved by [33], [34]. This work presented a novel low-cost optoelectronic setup for time- and spatially-resolved analysis of photonic emissions. The authors also introduced a corresponding methodology, named Simple Photonic Emission Analysis. They successfully performed such analysis of an AES implementation and were able to recover AES-128 keys by monitoring memory accesses. This work was also extended to AES-192 and AES-256 [34]. The same research group also introduced Differential Photonic Emission Analysis and presented a successful attack against AES-128 [25]. They successfully revealed the entire secret key with their DPEA. In 2015 an enhanced simple photonic emission attack of AES was introduced by [8]. The authors designed a photonic emission simulator which they calibrated against the equipment of [33], and used signal processing and cryptographic post processing in order to apply an attack against AES using much less data. Bertoni et al. [7] also offered an improved Simple Photonic Emission Analysis, monitoring a different section of the SRAM logic, and also described an attack against a masked AES. However, they assumed an unrealistic SRAM structure with only a single byte in every row, their simulations do not model the physical environment but rather model a theoretical case in which the value of every bit can be identified. However the attack is unrealistic since it assumes and requires monitoring the photonic emission of a single experiment.

Using side channel attacks to break RSA is a well studied topic, cf. [23], [30], [31]. The attacks aim at the exponentiation step of the decryption process, utilizing power or timing side channel. Side channel analysis using memory access patterns is also reminiscent of the field of cache attacks and localized attacks EM radiation attacks [17]. For instance, the first "real world" cache-based chosen plaintext attacks on AES were carried out on OpenSSL implementations [5], [28]. A similar cache-based attack against RSA was discovered around the same time by Colin Percival [29].

C. Contributions

In this work we attack the RSA crypto-system using the photonic side channel, which is the first attack utilizing PEA against a public-key crypto-system in a "real-world" programming environment.

Namely, we consider three common implementations of the modular multiplication of RSA: the binary ("square and multiply") method, the fixed-window method, and the Montgomery Ladder, using the Karatsuba multiplication method, for various key sizes and several implementation variations. This is exactly where our real-world setting differentiates itself from the apparently similar work of [17] which utilizes localized attacks EM radiation attacks: We are able to handle in our PEA the intrinsic complications arising from the Karatsuba multiplication, whereas [17] attacks an ordinary full length modular multiplier, which is reasonable only against a custom FPGA using a 163-bit ECC crypto-system, but not with 2048-bit RSA on a standard CPU chip.

We first developed an auto-calibrating method to decode the photonic traces, eliminating the need to manually set thresholds. Our decoder has attributes of a "soft decoder": it returns "ambiguous" for bit values where the number of photonic emissions is to too close to the threshold, drastically reducing the post-decoding error-correcting effort.

Then we conducted an extensive evaluation of our attacks against the RSA implementations. We discovered that the key length had marginal impact on resilience to the attack: attacking a 2048-bit key required only 9% more decryption attempts than a 1024-bit key to reach an equivalent success rate. The parameter that has the most dominant impact on the attacker's effort turns out to be the minimal block size B at which the Karatsuba method switches to the naive multiplication: even for very small values ($B = 32$ or $B = 64$ bits), recommended for embedded 8-bit RSA implementations, our attacks achieve 100% success rate under 10,000 decryptions.

Somewhat surprisingly, we discovered that Montgomery's Ladder—commonly perceived as the most resilient of the three implementations to side-channel attacks—was actually the most susceptible: for 2048-bit keys, the attack reveals 100% of the secret key bits with as few as 4000 decryptions.

An additional contribution is to stress again the value of an Photonic Emission Simulator as put forward by [7] and [8]. Indeed, due to the lack of PEA equipment we believe that our simulator might be of great academic value, bringing scientific research to a better understanding of the photonic side-channel. Note that this is in line with the well-accepted SPICE simulator, which is an accepted de-facto standard in electrical engineering to check the integrity of circuit designs and to predict correct circuit behavior.

Algorithm 1 The Binary Method

```
1: Input: $c, d, n$
2: Output: $M = c^d \pmod{n}$
3: if $d_{k-1} == 1$ then
4:     $M = c$
5: else
6:     $M = 1$
7: end if
8: for $i = k - 2$ downto $0$ do
9:     $M = M \cdot M \pmod{n}$
10:    if $d_i == 1$ then
11:        $M = M * c \pmod{n}$
12:    end if
13: end for
14: return $M$
```

Algorithm 2 The m-ary

```
1: Input: $c, d, n$
2: Output: $M = c^d \pmod{n}$
3: Pre-Compute and store $c^\omega \pmod{n}$ for all $\omega = 2, 3, 4, \ldots, m - 1$.
4: Decompose $d$ into $r$-bit windows $F_i$ for $i = 0, 1, 2, \ldots, s - 1$.
5: $M = c^{F_{s-1}} \pmod{n}$
6: for $i = s - 2$ downto $0$ do
7:     $M = M^{2r} \pmod{n}$
8:     if $F_i \neq 0$ then
9:         $M = M \cdot c^{F_i} \pmod{n}$
10:    end if
11: end for
12: return $M$
```

Organization. The organization of the present paper is as follows. Section II introduces the modular exponentiation methods and the phototonic emissions side-channel. Section III describes how to attack RSA using the photonic side channel. Section IV explains how we decode the photonic traces. Section V describes our performance evaluation, and we conclude in Section VI.

II. Preliminaries and Background

A. RSA crypto-system

In the RSA public-key crypto-system, encryption of a message M is computed as $c = M^e \pmod{n}$ where $n = p \cdot q$ and p and q are two odd primes of approximate equal size, e is the public exponent, and decryption is done by computing $M = c^d \pmod{n}$ where d is the private key.

B. Exponentiation Methods

There are several popular methods to calculate modular exponentiation efficiently. Here we briefly describe the 3 methods we attack in this work.

1) The Binary Method: An efficient yet simple algorithm is the Binary method (also known as square and multiply) cf. [21]. This method (Algorithm 1) scans the bits of the exponent from the MSB to the LSB (an opposite variant of the method also exists). For each step a squaring is performed, and if the exponent bit is 1 a subsequent multiplication is performed.

Algorithm 3 The Montgomery's Ladder

```
1: Input: $c, d, n$
2: Output: $M = c^d \pmod n$
3: $M_1 = c$
4: $M_2 = c^2$
5: for $i = k - 2$ downto 0 do
6:     if $d_i == 0$ then
7:         $M_2 = M_1 * M_2 \pmod n$
8:         $M_1 = M_1^2 \pmod n$
9:     else
10:        $M_1 = M_1 * M_2 \pmod n$
11:        $M_2 = M_2^2 \pmod n$
12:    end if
13: end for
14: return $M_1$
```

2) The m-ary method (fixed window): The m-ary method, (Algorithm 2) c.f. [15], scans the bits of the exponent by $r = \log_2 m$-bits at a time. We call r the "window length". The exponent d is partitioned into s blocks of length r each for $s \cdot r = k$. If r does not divide k, the exponent is left-padded with at most r zeros. We define $F_i = (d_{ir+r-1}d_{ir+r-2} \dots d_{ir})$ to be the i-th key window. The m-ary method pre-computes $C^{\omega} \pmod n$ for $\omega = 2, 3, \dots, m-1$, in advance for a given exponentiation to the power of d. The bits of d are scanned r bits at a time and in each step the temporary result is raised to the 2^r power and then also multiplied by C^{F_i} for $F_i \neq 0$.

Note that there is variant of the fixed-window method, known as the sliding window method [22], which partitions the exponent into zeros and nonzero words of variable length. The sliding window method has some performance advantages if there are long zero-sequences in the exponent, since multiplications can be avoided.

3) Montgomery's Ladder: In order to provide immunity against timing-based and power-based side channel attacks, the so called Montgomery Ladder (Algorithm 3) can be used, c.f. [19]. The algorithm provides two symmetric branches with the same fixed sequence of operations regardless of the private key bit value. Its symmetry is the property that protects the algorithm against prior side channel attacks: It executes in a fixed time for all inputs of the same size. As we shall see, this is not enough to defend against side channels that are based on memory accesses such as the photonic side channel.

C. The Karatsuba Algorithm

All exponentiation methods need to perform many large number multiplications. The naive multiplication algorithm for two n bit numbers requires $\Omega(n^2)$ digit multiplications [21]. A more efficient multiplication algorithm is the Karatsuba algorithm [20] requiring at most $n^{\log_2 3}$ digit multiplications. In order to calculate $a \cdot b$ the algorithm splits the n-bit multiplicands a and b, into halves denoted a_0,a_1 and b_0,b_1 respectively. The product can be formed using only three recursive $n/2$-bit multiplications and some additions and subtractions:

$$a \cdot b = a_1 \cdot b_1 \cdot 2^n + ((a_1 + a_0) \cdot (b_1 + b_0) - a_1 \cdot b_1 - a_0 \cdot b_0) \cdot 2^{n/2} + a_0 \cdot b_0$$

The three $n/2$-bit products can be computed by recursive calls of the Karatsuba algorithm. The recursion is applied until the numbers are reduced to a size B where performing a naive multiplication is more efficient.

A crucial parameter of the Karatsuba algorithm's performance is the value of B. According to [14], the implementation in the GnuPG library [2] stops the recursion at a block size of $B = 512$ bits. Hutter and Schwabe [18] report that on the 8-bit ATMega architecture Karatsuba multiplication is faster than the naive method for surprisingly small inputs, starting at 48 or 64 bits, which in our terminology implies stopping the recursion at a block size of $B = 24$ or $B = 32$.

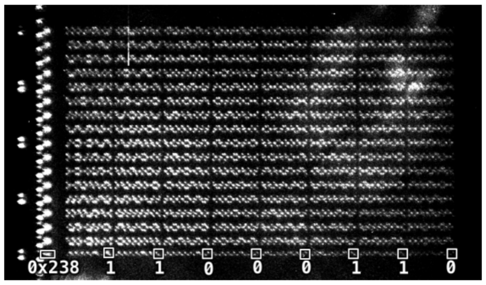

Fig. 1. The SRAM memory captured with a CCD by the courtesy of [33]. The row-access transistors appear to the left of the SRAM cells.

D. The Photonic Side Channel

1) The SRAM: SRAM is a common type of volatile memory found in many ICs. SRAM is built from memory cells arranged in rows and columns, and every memory cell can be approached using a row/column access logic. In particular, the access logic for each SRAM row includes a so called row-access transistor, which is activated whenever the IC needs to access any cell in that SRAM row. In order to enable an entire row, the row-access transistor is very strong. This means that the photonic emission of this transistor is by magnitudes larger than the individual SRAM cells by themselves. For a thorough introduction into SRAM and its physical implementation details see [37].

The number of bytes in an SRAM row, denoted by r, depends on the underlying SRAM architecture. In [33] the authors found that on an AT-Mega328P an SRAM row consists of $r = 8$ bytes, whereas an ATXMega128A1 stores $r = 16$ bytes in a row. Figure 1 shows a photo of the SRAM, with a row width of $r = 8$ bytes.

2) Simple Photonic Emission Analysis (SPEA): Monitoring the access patterns to the SRAM rows allows the SPEA as presented in [33]. The authors first used a CCD camera to map the IC's layout, locating the SRAM memory cf. [26]. Then they placed a NIR (Near Infra Red) photon detector offering time resolved measurements over the row access transistor of some SRAM row. The authors ran the encryption algorithm T times encrypting the same plaintext, where T is some large number, providing a sufficient SNR. By monitoring accesses to that SRAM row, they revealed sets of potential key candidates and used them to reveal the secret key.

III. Attacking RSA with photonic side channel

The arguments of the modular exponentiation are stored in an internal SRAM array of the IC performing the RSA decryption. In each step of the modular exponentiation, a value is read from the memory, some computations are made involving other SRAM locations, and the result is stored. By monitoring the row access transistor of a memory row containing bytes involved in the calculation, we can deduce the course of the algorithm and find the private exponent d.[1]

When using Karatsuba multiplication, the algorithm recursively splits the multiplicands until it reaches the minimal size B, where the algorithm performs a naive multiplication between partial parts of the multiplicands. For this naive multiplication the B-bit parts of the multiplicands need to be read from the memory, generating accesses to the relevant memory section. After conducting the naive multiplication, the results are not rewritten to the same SRAM addresses—instead they are kept in the stack, i.e., in locations not controlled by the monitored row transistor. So the only accesses to the monitored SRAM memory storing the multiplicands takes place during the naive multiplication.

A. Attacking the binary method

For the binary method algorithm we monitor the memory section storing c. During the loop operation we shall observe accesses for every private key bit $d_i = 1$. Thus whenever we detect accesses we can deduce that the current private key bit is 1, whereas for time periods lacking accesses— the private key bit is 0. This way we can recover the private key.

When the Karatsuba algorithm is used, the multiplication step ($M{*}c \pmod{n}$) is conducted recursively and the memory area storing c is only accessed in “limbs” of size B at a time. So, as stated above, when monitoring a single memory row, storing several bytes of c, the number of expected row accesses depends on the size of B. Note that the ATmega328P processor has an 8-bit data bus, hence every RAM byte that is read during the naive multiplication potentially generates an observable memory access.

An alternative approach for revealing the private key is monitoring the memory area containing the temporary value of M and differentiating between the cases of the “if” statement (line 10 in Algorithm 1). Whenever $d_i = 1$ an additional operation ($M = M * c \pmod{n}$) is conducted, generating accesses to the memory storing M. By differentiating between two different memory access patterns—based on the amount of memory accesses, we can deduce whether the condition of the “if” statement was True or not.

B. Attacking the m-ary method

For the m-ary method (Algorithm 2) we place our detector over the row access transistor of a row containing one of the precomputed variables c^{ω} (line 3), e.g., c^2 for the $m = 4$ case (2-bit windows). During the loop operation, for every window of private key bits equaling ‘10’, the operation $M{*}c^2 \pmod{n}$ (line 9) will take place, generating accesses to the memory section we are monitoring and revealing the corresponding 2 bits of the private key.

C. Attacking Montgomery’s Ladder

When attacking Montgomery’s Ladder (Algorithm 3), we can monitor either of M_1 or M_2. For example when we monitor M_1, for $d_i = 0$ we would have M_1 multiplied, squared and reassigned which will generate more accesses than when $d_i = 1$, in which case M_1 is only multiplied and reassigned (line 10). By differentiating between these two cases based on the amount of memory accesses, we can reveal the entire private key.

When monitoring M_2 instead of M_1 the “if” statement cases are reversed. Hence, if we do not know whether the monitored memory location stores M_1 or M_2, we only have two options for the key bits which are the bitwise complement of each other.

IV. Decoding the Photonic Traces

We activate the IC (or, in our case, the photonic emission simulator) T times to decrypt the ciphertext. For each activation we count the number of detected photons per time step, while the detector is fixed at some SRAM row as described above. The time step corresponds to the specific algorithm and matches the execution time of the operations inside the loop. We summarize the detection counts per time step, to obtain a “photonic trace” (an example of a trace can be found in Figure 2 (a)). Following [32], [33] we assume an IC instruction cycle of 800 ns. [Note that this clock frequency is a slow 1.25MHz. This clock frequency was calibrated to the real lab setup of [32], [33]] For multiplication of 2 n-bit multiplicands M_1, M_2 stored in SRAM memory with a row width of r bytes and minimal Karatsuba size of B bits, the naive multiplication executes $(B/8)^2$ 1-byte multiplications. Assuming a 1-byte memory bus, of the $(B/8)^2$ accesses, only an $8 \cdot r/B$ fraction is observable by a detector placed over one of the rows of one multiplicand, giving:

$$\#\text{observable-row-accesses} = B \cdot r/8 \qquad (1)$$

Note that (1) differs qualitatively from the effects in SPEA against AES [32], [33]: RSA multiplications access *all* the bytes in the row, multiple times, thus having a larger SRAM row width r makes the attack more efficient as it increases the number of observable accesses. In contrast, SPEA against AES with $r = 16$ (as in the ATXMega128A1 IC) requires much more effort than when $r = 8$ since the attacker must identify the specific byte that was accessed in the row.

We now need to decode the trace to distinguish between the two cases of the if statement in order to reveal the value of the private key bit d_i. A natural decoding rule is to use a threshold: if the number of detected activations during the time step exceeds the threshold, the if statement is True.

A. Threshold Calibration

A crucial task is calibrating the threshold to reliably distinguish between true detections and noise. Instead of a using

[1]Note that all known embedded RSA hardware-coprocessors also map their long internal registers to the internal RAM array, for easier software handling, and also that existing modern smartcards are still manufactured in 90nm where PEA is known to work.

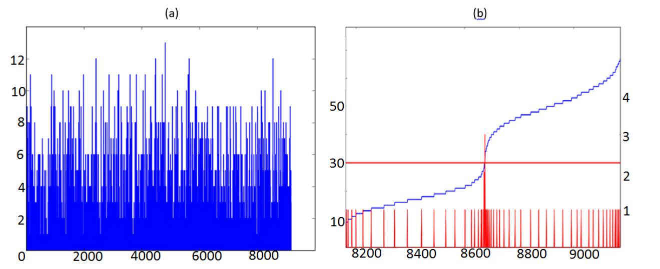

Fig. 2. (a) A photonic trace received from the simulator during RSA decryption for $T = 1,000$. Each point in the x axis corresponds to 220μsec. The y axis counts photonic detections per time step. (b) The sorted trace in blue for $T = 9,000$. The derivative plot is in red where we can spot a peak. The corresponding threshold is shown as a horizontal line.

a heuristic trial-and-error process, we calibrate the threshold automatically to an optimal value.

When monitoring accesses to an SRAM row containing a variable of the exponentiation algorithm, we have two cases: (i) "some-to-zero": distinguish between SRAM accesses versus no accesses. This is the needed distinguisher when attacking the binary method and monitoring c (Section III-A) or when attacking the m-ary method (Section III-B). (ii) "many-to-few": distinguish between many SRAM accesses versus few accesses: this is needed when attacking the binary method and monitoring M, or attacking Montgomery's Ladder (Section III-C).

For the first case, if we sort the "photonic trace" in ascending order, we expect to see a steep rise, a "jump", in the sorted trace (when the SNR is sufficient), between samples containing SRAM accesses and samples containing only noise. By looking at the derivative plot of the sorted photonic trace, we can detect this jump according to a distinct peak in the derivative plot (see Figure 2 (b)). The value of the threshold is set to be the value of the sorted photonic trace where the derivative plot has a peak (the mid-point of the "jump").

The pseudo-code for the "some-to-zero" threshold calibration is:

```
1) Sort the trace in ascending order.
2) Calculate the derivative of the sorted trace.
3) Find a peak in the derivative plot.
4) Set $x_{peak}$ to be the $x$ value for the peak.
5) Set the threshold to be the value of the sorted trace at $x_{peak}$.
```

We use this threshold in order to decode the photonic trace and decide for every bit of the private key d whether the If statement was carried out or not, revealing the secret key. For the second case ("many-to-few") we expect to find *two* steep rises in the sorted photonic trace since there will be one jump between samples containing only noise and samples containing the lesser amount of accesses, and another jump between samples containing the lesser amount of accesses and samples containing more SRAM accesses. For this case we take the threshold value to be the value of the sorted trace where the derivative trace has its second peak value.

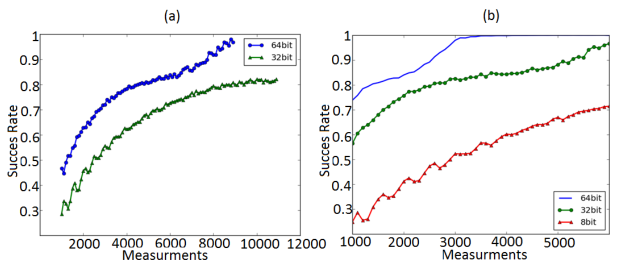

Fig. 3. (a) The success rate for the Binary method as a function of the number of decryptions for a key size of 1024 and a Karatsuba stopping size of $B = 32$ bits (bottom curve) and $B = 64$ bits (top curve). (b) The success rate for Montgomery's Ladder as a function of the number of decryptions for a key size of 1024 and a Karatsuba stopping size of $B = 8$ bits (bottom curve) $B = 32$ bits (middle curve) and $B = 64$ bits (top curve).

B. The Ambiguous Range and Handling Errors

When the number of measurements T is not high enough, the SNR drops and the threshold calibration method may introduce decoding errors. Even a single decoding error of one bit is critical, since we will have no knowledge of which is the incorrect bit. To enumerate over all possible 1-bit corrections would require n attempts—and in general $O(n^e)$ attempts for e 1-bit errors. When $n = 1024$ or more, this error correction can be prohibitive. The problem becomes much more tractable when the locations of the incorrect bits are known: then the correction of e errors only requires $O(2^e)$ attempts. We address this problem by introducing an ambiguous range close to the threshold. Trace samples in the ambiguous range are not assigned with a 0/1 value—instead they are declared as ambiguous bits, and we will have to enumerate over their values. The ambiguous range is set as follows:

1) Set a threshold T as presented in Section IV-A.
2) Set μ_x to be the mean value of samples greater than T.
3) Set $gap = \mu_x - T$, and set the ambiguous range to

$$\text{Ambiguous range} = [T - a \cdot gap, T + a \cdot gap] \quad (2)$$

In our experiments we set the parameter to be $a = 0.1$.

V. Practical Results

We used the photonic emissions simulator of [8] to simulate an ATmega328P running at 1.25MHz. The experiments were run on an Intel Core Duo T2450 2GHz, 2GB RAM PC running Windows Vista. We simulated the ATmega328P IC with SRAM row width of $r = 8$ and generated the RSA keys according to the PKCS standard.

In order to evaluate the performance of the attack we performed an extensive set of experiments. The experiments were done on the three modular exponentiation methods, with various key sizes, using various various stopping sizes B for the Karatsuba multiplication. We define the success rate as the percentage of private key bits we decode successfully.

A. Attacking the Binary method and Montgomery's Ladder

In Figure 3(a) we can see the success rate for the Binary method where the detector monitors the memory storing c, as a

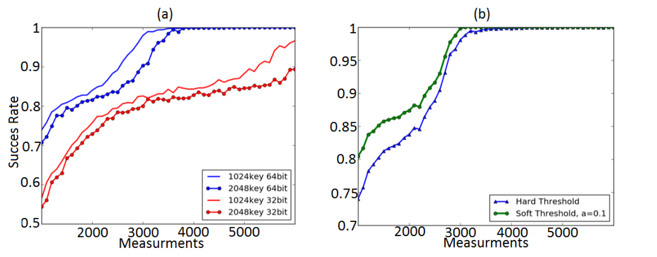

Fig. 4. (a) The success rate for Montgomery's Ladder as a function of the number of decryptions for a key size of 1024 and 2048 and a Karatsuba stopping size B of 32 and 64 bits. (b) The success rate for the Montgomery Ladder using the normal threshold (bottom curve) and the ambiguous range (top curve) as a function of the number of decryptions for a key size of 1024 and $B = 64$ bits.

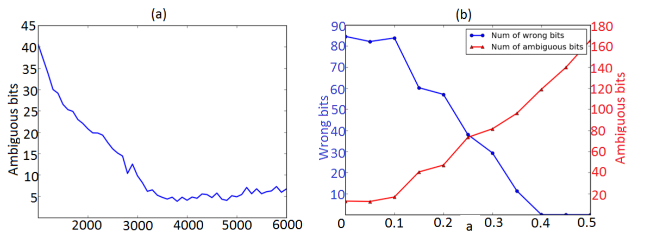

Fig. 5. (a) The number of ambiguous bits as a function of the number of decryptions for $a = 0.1$ a key size of 1024 and $B = 64$ bits. (b) The number of wrong and ambiguous bits for Montgomery's Ladder as a function of the parameter a for a key size of 1024, Karatsuba stopping size $B = 64$ bits, and 2500 measurements.

function of the number of decryptions performed. We can see that with $B = 64$ the success rate approaches 1 around $T = 10,000$ decryptions, while $B = 32$ has a lower success rate and requires twice as many decryptions for the same success rate—as expected by Equation 1.

Figure 3(b) shows the success rate in attacking Montgomery's ladder . Again we see the same effect—larger values of the Karatsuba minimal block size B help our attack. Further, comparing Figure 3(a) and (b) we see that our attack is more efficient against Montgomery's ladder, where more memory accesses are generated during the decryption process. This improves the SNR, and hence, allows fewer measurements to achieve the same success rate.

Figure 4(a) shows the effect of the key size n on the success rate. We see that for a larger key size more decryptions are needed to achieve the same success rate since there are more secret exponent bits to discover—but the difference is not dramatic. E.g., for a 99% success rate against Montgomery's ladder with $B = 64$ we need 3300 decryptions for 1024-bit keys, and 3600 for 2048-bit keys: 9% more.

Figures 4(b) shows the effect of the ambiguous range on Montgomery's Ladder. The figure shows that using the ambiguous range improves the success rate: The success rate is only measured over the non-ambiguous bits, thus the improvement in the success rate indicates that those bits marked as ambiguous would cause a significant number of decoding errors when using a hard threshold.

The importance of the ambiguous range is more evident

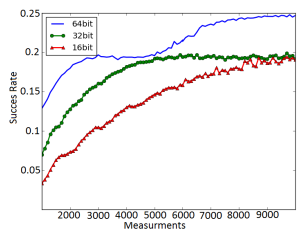

Fig. 6. The success rate for the m-ary when $m = 4$ as a function of the number of decryptions for a key size of 1024 and Karatsuba stopping size of $B = 16$ bits (bottom curve) $B = 32$ bits (middle curve) and $B = 64$ bits (top curve).

when considering absolute numbers. Figure 5 shows that the absolute number of ambiguous bits is very small: between 5–10 ambiguous bits beyond 4000 measurements. So few missing bits can easily be enumerated over with up to 2^{10} decryption tests—much better than correcting even 5 incorrectly decoded bits at unknown locations via $\binom{1024}{5} \cdot 2^5$ decryptions.

Figure 5(b) shows the numbers of wrong and ambiguous bits as a function of the parameter a for an intentionally-low number of tests ($T = 2500$). As expected, as a grows, we observe fewer wrong bits, but more ambiguous bits we have to enumerate on.

B. Attacking the m-ary method

For the m-ary method (Algorithm 2), when $m = 4$, using one detector we can monitor access to only one out of the four precomputed values stored in memory, e.g., the value c^2. So for every 2-bit window in the secret exponent d, for 1/4 of cases there is an observable memory access and we can learn that the bits' value is '10'. For 3/4 of cases there is no observable memory access, so we can learn that the bits' value is *not* '10'. Thus the success rate we can hope for when $m = 4$ is 25% (and $1/m$ in general). From an information-theory point of view we learn more: for $n = 1024$ we have 512 2-bit windows, and in 3/4 of these windows we can only have 3 possible values, so the remaining entropy is $512 \cdot 0.75 \cdot \log_2 3 = 608$ bits, i.e., we learn 40.6% of the secret exponent.

Figure 6 shows that the number of required measurements is as large as in the Binary method, since the number of SRAM accesses is low. Furthermore, we can see that, as expected, the success rate approaches 25% once sufficiently many decryption measurements become available.

Thus we see that the m-ary method is much more resilient to our attack than Montgomery's Ladder—in contrast to the resilience to timing or power-analysis side channels, against which Montgomery's Ladder is generally superior.

Note that despite learning only 25% of the secret exponent bits, the attack can perhaps be extended to reveal the missing bits. If the RSA implementation uses the CRT, then after our attack we know the public key n, e exactly, we discover 25% of the bits of d_p and d_q, and these bits are in arbitrary (but known) positions. This situation is close to that described by Heninger and Shacham [16], where the authors developed a solver that reveals the entire key with high probability from a partial key with only 24% randomly located bits. In their case they also assumed a partial knowledge of p and q, which we do not have. On the other hand, we know that the missing bits are constrained: every missing 2-bit window in d_p and d_q *cannot* equal, e.g., the value '10'. We leave this direction as an open question.

An alternative approach is to use multiple detectors: e.g., for $m = 4$, with 3 detectors, we could place them over the memory areas holding c^1, c^2, and c^3. By repeating our attack 3 times in parallel we would learn 3/4 of the bits (for all the windows where one of the detectors observes accesses), and reveal the remaining 25% by identifying windows where none of detectors observed memory accesses. Using more than one photonic detector was recently demonstrated in [13]. We leave the evaluation of the multi-detector attack for future work.

VI. Conclusions and Countermeasures

In this paper we demonstrated, for the first time, that the photonic side channel can be successfully used against the RSA public-key crypto-system. We discovered that the key length had marginal impact on resilience to the attack, while the minimal Karatsuba block size had a dominant effect. Somewhat surprisingly, we discovered that Montgomery's Ladder—commonly perceived as the most resilient of the three implementations we evaluated to side-channel attacks—was actually the most susceptible, while the m-ary method only allowed us to reveal a $1/m$ fraction of the secret exponent bits. This means that resilience to memory-access side channels in general, and the photonic emission side channel in particular, should be considered as important aspects of public-key crypto-system implementations.

While the symmetry of Montgomery's Ladder protects it from timing and power analysis side-channels, it is susceptible to the photonic side channel because its memory access pattern depends on the secret key bits. Thus, to protect RSA against the photonic side channel, we need to eliminate secret-key-dependent branches from the implementation—and there are ways to do so [6]. More general countermeasures against photonic side channel attacks include varying the memory locations of central variables used by the decryption process, adding dummy operations involving the same memory space access patterns.

References

[1] FIDO alliance. https://fidoalliance.org/about/overview/.

[2] Gnu privacy guard. http://https://www.gnupg.org/.

[3] Projekt 267: Analyse nicht-zertifizierter IT-Sicherheitselemente - Teilprojekt 1: Untersuchung eine JAVA-Karte mit FIDO U2F applet (German). BSI tender: https://www.evergabe-online.de/tenderdetails.html?0&id=122064.

[4] G. Bascoul, P. Perdu, A. Benigni, S. Dudit, G. Celi, and D. Lewis. Time resolved imaging: From logical states to events, a new and efficient pattern matching method for VLSI analysis. *Microelectronics Reliability*, 51(9):1640–1645, 2011.

[5] D. J. Bernstein. Cache-timing attacks on AES, 2004. Preprint, http://cr.yp.to/papers.

[6] D. J. Bernstein. Personal communication, 2016.

[7] Y. M. Bertoni, L. Grassi, and F. Melzani. Simulations of optical emissions for attacking AES and masked AES. In *Security, Privacy, and Applied Cryptography Engineering (SPACE), LNCS 9354*, pages 172–189. Springer Verlag, 2015.

[8] E. Carmon, J.-P. Seifert, and A. Wool. Simple photonic emission attack with reduced data complexity. In F.-X. Standaert and E. Oswald, editors, *7th Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 35–51, Graz, Austria, Apr. 2016. Springer.

[9] A. Chynoweth and K. McKay. Photon emission from avalanche breakdown in silicon. *Physical Review*, 102(2):369, 1956.

[10] J. Di-Battista, J.-C. Courrege, B. Rouzeyre, L. Torres, and P. Perdu. When failure analysis meets side-channel attacks. In *Cryptographic Hardware and Embedded Systems(CHES)*, pages 188–202. 2010.

[11] P. Egger, M. Grützner, C. Burmer, and F. Dudkiewicz. Application of time resolved emission techniques within the failure analysis flow. *Microelectronics Reliability*, 47(9):1545–1549, 2007.

[12] J. Ferrigno and M. Hlaváč. When AES blinks: introducing optical side channel. *Information Security*, 2(3):94–98, 2008.

[13] F. Ganji, J. Krämer, J.-P. Seifert, and S. Tajik. Lattice basis reduction attack against physically unclonable functions. In *Proc. 22Nd ACM Conference on Computer and Communications Security (CCS'15)*, pages 1070–1080, 2015.

[14] D. Genkin. personal communications, 2016.

[15] D. M. Gordon. A survey of fast exponentiation methods. *Journal of algorithms*, 27(1):129–146, 1998.

[16] N. Heninger and H. Shacham. Reconstructing RSA private keys from random key bits. In *Advances in Cryptology-CRYPTO 2009*, pages 1–17. Springer, 2009.

[17] J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl. Localized electromagnetic analysis of cryptographic implementations. In *Cryptographers Track at the RSA Conference*, pages 231–244. Springer, 2012.

[18] M. Hutter and P. Schwabe. Multiprecision multiplication on AVR revisited. *Journal of Cryptographic Engineering*, 5(3):201–214, 2015.

[19] M. Joye and S.-M. Yen. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 291–302. Springer, 2002.

[20] A. A. Karatsuba. The complexity of computations. *Proceedings of the Steklov Institute of Mathematics-Interperiodica Translation*, 211:169–183, 1995.

[21] D. E. Knuth. Seminumerical algorithms. 2007.

[22] C. K. Koc. High-speed RSA implementation. Technical report, RSA Laboratories, 1994.

[23] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *16th Advances in Cryptology — CRYPTO'96*, pages 104–113. Springer, 1996.

[24] J. Krämer, M. Kasper, and J.-P. Seifert. The role of photons in cryptanalysis. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 780–787. IEEE, 2014.

[25] J. Krämer, D. Nedospasov, A. Schlösser, and J.-P. Seifert. Differential photonic emission analysis. In *Constructive Side-Channel Analysis and Secure Design*, pages 1–16. Springer, 2013.

[26] D. Nedospasov, J.-P. Seifert, A. Schlosser, and S. Orlic. Functional integrated circuit analysis. In *IEEE Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 102–107, 2012.

[27] R. Newman. Visible light from a silicon pn junction. *Physical Review*, 100(2):700–703, 1955.

[28] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of AES. In *Topics in Cryptology–CT-RSA 2006*, pages 1–20. Springer, 2006.

[29] C. Percival. Cache missing for fun and profit. http://www.daemonology.net/papers/cachemissing.pdf, Oct. 2015.

[30] W. Schindler. A timing attack against RSA with the Chinese remainder theorem. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 109–124. Springer, 2000.

[31] W. Schindler. A combined timing and power attack. In D. Naccache and P. Paillier, editors, *Proc. 5th International Workshop on Practice and Theory in Public Key Cryptosystems, (PKC)*, pages 263–279, 2002.

[32] A. Schlösser. *Hot electron Luminescence in silicon structures as photonic side channel* (in German). PhD thesis, Faculty of Mathematics and Natural sciences, Berlin Institute of Technology, 2014.

[33] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert. Photonic emission analysis of AES. *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2012.

[34] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert. Simple photonic emission analysis of AES. *Journal of Cryptographic Engineering*, 3(1):3–15, 2013.

[35] L. Selmi, M. Mastrapasqua, D. M. Boulin, J. D. Bude, M. Pavesi, E. Sangiorgi, and M. R. Pinto. Verification of electron distributions in silicon by means of hot carrier luminescence measurements. *Electron Devices, IEEE Transactions on*, 45(4):802–808, 1998.

[36] P. Song, F. Stellari, B. Huott, O. Wagner, U. Srinivasan, Y. Chan, R. Rizzolo, H. Nam, J. Eckhardt, T. McNamara, et al. An advanced optical diagnostic technique of IBM z990 eserver microprocessor. In *Proceedings IEEE International Test Conference (ITC)*, pages 9–pp. IEEE, 2005.

[37] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits And Systems Perspective, 4/E*. Pearson Education, 2010.