A Major Vulnerability in RSA Implementations due to MicroArchitectural Analysis Threat (August 22, 2007)

Onur Acıiçmez[1] and Werner Schindler[2]

[1] Samsung Information Systems America, Samsung Electronics
95 West Plumeria Drive, San Jose, CA 95134, USA
onur.aciicmez@gmail.com
[2] Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185–189, 53175 Bonn, Germany
Werner.Schindler@bsi.bund.de

Abstract. Recently, Acıiçmez, Koç, and Seifert have introduced new side-channel analysis types, namely Branch Prediction Analysis (BPA) and Simple Branch Prediction Analysis (SBPA), which take advantage of branch mispredictions occur during the operations of cryptosystems [4, 5]. Even more recently, Acıiçmez has developed another attack type, I-cache analysis, which exploits the internal functionalities of instruction/trace caches [1]. These MicroArchitectural Analysis (MA) techniques, more specifically SBPA and I-cache Analysis, have the potential of disclosing the entire execution flow of a cryptosystem as stated in [4, 1]. Our focus of interest in this paper is that these attacks can reveal whether an extra reduction step is performed in each Montgomery multiplication operation. First Walter et al. and then Schindler developed attacks on RSA, which result in total break of the system if the occurrences of extra reduction steps can be determined with a reasonable error rate [39, 30, 29]. These attacks may be viewed as theoretical in the sense that neither Walter et. al. nor Schindler implemented actual attacks on real systems but instead they assumed that side-channel information obtained via power and timing analysis would reveal such occurrences of extra reduction step. In this paper we adjusted the attack from [30] to the current OpenSSL standard and put this attack into practice, proving its practicality via MA. The second part of the attack exploits the previously gathered information on the required extra reductions in an optimal way, using advanced stochastic methods as the formulation and analysis of stochastic processes.

Our results show the feasibility of compromising current RSA implementations such as OpenSSL. After we shared our result with OpenSSL development team, they included a patch into the stable branch ([45]), which allows users to compile an OpenSSL version that is resistent against our attack ([46]). In particular, this patch will affect the upcoming version of 0.9.8f. We also contacted the US CERT who informed software vendors. The US CERT assigned the vulnerability explained in this paper CVE name CVE-2007-3108 and CERT vulnerability number VU#724968, and they issued a vulnerability note ([47–49]). We point out that this publication appeared in accordance with the OpenSSL development team.

Several countermeasures have been developed and employed in widely used cryptographic libraries like OpenSSL to mitigate such side-channel analysis threats. However the current implementations still do not provide sufficient protection against MicroArchitectural Analysis, despite of all the sophisticated mitigation techniques employed in these implementations. In this paper, we will show that one can completely break the RSA implementation of the current OpenSSL version (v.0.9.8e) even if the most secure configuration, including all of the countermeasures against side-channel and MicroArchitectural analysis, is in place. We have only analyzed OpenSSL, thus we currently do not know the strength of other cryptographic libraries. Other libraries and software products need to be thoroughly analyzed and appropriately modified if it is necessary. At least, developers of the current software applications that rely on OpenSSL RSA implementation need to update their products based on the recent OpenSSL changes. Our results indicate that MicroArchitectural Analysis threatens at least 60% of the internet traffic worldwide and the current systems should be analyzed thoroughly to evaluate their overall strength against MicroArchitectural Analysis ([44]). We will eventually discuss appropriate countermeasures that must be employed in security systems.

Keywords: RSA, Montgomery Multiplication, MicroArchitectural Analysis, Instruction-Cache Attack, Branch Prediction Attack, Timing Analysis, Side Channel Analysis, Stochastic Process.

1 Introduction

MicroArchitectural Attacks (MA), which exploit the microarchitectural behavior of modern computer systems, form a new group of side-channel analysis. Cache, Branch Prediction, and Instruction Cache Analysis are three types of MA that are "publicly" known so far. [3] Branch Prediction Analysis (BPA) and its very powerful variant Simple Branch Prediction Analysis (SBPA) have been introduced by Acıiçmez et. al. [4, 5]. They showed that a carefully written spy-process running simultaneously with an RSA-process, is able to collect during one *single* RSA signing execution almost all of the secret key bits. They call such an attack, analyzing the CPU's Branch Predictor states through spying on a single quasi-parallel computation process, a *Simple Branch Prediction Analysis (SBPA)* attack — sharply differentiating it from those one relying on statistical methods and requiring many computation measurements under the same key. Following this interesting research vector, Acıiçmez has developed another MA attack based on the functionality of instruction cache (I-cache), which is another major processor component [1]. This new attack, called I-cache Analysis, is also aiming to reveal the instruction flow of cryptosystems just like SBPA.

The major cryptographic libraries, especially OpenSSL which is widely utilized in most of the security software today (according to an estimation from NTT made in November 2006, more than 60% of the web servers worldwide has OpenSSL toolkit installed [44]), have gone under several revisions to mitigate different MA attacks immediately after the announcements of their feasibilities. Unfortunately, despite of all these efforts spent to provide better protections against MA attacks, the current version of OpenSSL [4] still has a major vulnerability. One can completely break this RSA implementation even if all the currently implemented countermeasures are turned on[5]. The most secure configuration of the current OpenSSL version employs Fixed-window exponentiation[6], base blinding[7], and various other countermeasures for MA vulnerabilities. However, the most important algorithm for our purpose in this paper is Montgomery Multiplication (MM). The implementation of MM in OpenSSL still has the extra reduction step in version 0.9.8e, although it has been shown many times that this particular step cause serious security vulnerabilities, cf. [8, 12, 13, 33, 32, 39, 30, 29]. It also has been known for a long time that MM can be easily implemented without extra

[3] There is, in fact, a new paper describing a recently discovered MA type [2]. The details of it were not publicly available by the time we wrote this current paper. Therefore we omit this attack here and prefer not to disclose the details.

[4] Even the upcoming OpenSSL version (v.0.9.8f) which will contain several SBPA countermeasures would still be vulnerable to the attack we describe in this paper. After we shared our result with OpenSSL development team, they have prepared a patch which has been included in the stable version ([46]) and in particular will affect the upcoming version 0.9.8f.

[5] The side-channel related countermeasures in OpenSSL are optional. Some of them are turned on by default, while others are not. The systems built upon OpenSSL can choose which countermeasures to be active by setting the corresponding OpenSSL flags. It is possible for a system to activate all of these countermeasures, which naturally come with some performance loss, or totally ignore/deactivate all of them

[6] Fixed-window exponentiation was implemented as a mitigation technique against cache analysis presented in [27]. OpenSSL also handles the RSA structures in a special way to avoid cache based threats. These two techniques come in a bundle, i.e. used together to protect enhanced security against cache analysis

[7] OpenSSL uses a technique called base-blinding which prevents in particular pure timing attacks. This method was first implemented as an optional protection against chosen- and known-text attacks and later on became a default protection mechanism after the publication of Brumley and Boneh Attack [12], see also [8].

reduction step, cf. [40, 41, 14]. In this paper, we will show that even if we utilize all of these mitigation techniques in OpenSSL, we can still completely break the RSA implementation.

It is important to note that OpenSSL implementation has recently been modified to eliminate extra reduction step in MM after we showed the feasiblity of completely breaking even the most secure configuration of the library ([45]). These changes will in particular affect the upcoming OpenSSL version 0.9.8f. We also contacted the US CERT who informed software vendors. The US CERT assigned the vulnerability explained in this paper CVE name CVE-2007-3108 and CERT vulnerability number VU#724968, and they issued a vulnerability note ([47–49]).

Although SBPA and I-cache Analysis have the potential to reveal the entire execution flow of an RSA cipher, our focus in this paper is only on the extra reduction step of the Montgomery multiplication algorithm. We show that MA can be used to determine which Montgomery multiplication operations perform an extra reduction during the entire execution of RSA. The importance of this information is due to [39, 30, 29]. Walter et. al. developed an attack on RSA (for a fixed window size of 2 bits) which allow to extract the secret exponent if the occurrences of extra reduction steps for a sample of RSA decryption/signing under the same RSA key are known to the attacker [39]. Later, Schindler generalized and optimized this attack for arbitrary window sizes ([30]), and then Schindler and Walter extended this attack to a variant of Montgomery's multiplication algorithm ([29]). We need to mention that these attacks may be considered as "theoretical" because neither Walter et. al. nor Schindler implemented actual attacks on real systems but instead they assumed that side-channel information obtained via power and timing analysis would reveal such occurrences of extra reduction step.

In this paper, we combine the practicality of MicroArchitectural Analysis (Instruction Cache Analysis in particular) and the theory of [30] to show that the use of extra reduction step in Montgomery multiplication leads to a total break of current RSA software implementations. We also suggest several countermeasure that must be employed in software implementations of RSA.

2 MicroArchitectural Analysis: History, Fundamentals, and Potentials

MicroArchitectural Analysis (MA), which is a newly evolving area of side-channel cryptanalysis, studies the effects of common processor components and their functionalities on the security of software cryptosystems. As a natural consequence of strictly throughput, performance, and "performance per watt" oriented goals of modern processor designs and also highly "time-to-market" driven business philosophy, the resulting products, i.e. commodity processor architectures in the market, lack a thorough security analysis. The main element that gave birth to MicroArchitectural Analysis area is indeed this particular gap between the current processor architectures and the ideal secure computing environment.

The actual roots of MA goes a long way back to [16, 34]. Although the security risks of processor components like cache were implicitly pointed out in these publications, concrete and widely applicable security attacks based upon processor functionalities have recently been worked out and immediately attracted significant public interest. There are currently 3 different types of MA that had been identified so far: Cache, Branch Prediction, and I-cache Analysis.

The typical targets of side-channel analysis have been and still are smart cards. However, we have seen a significant increase in research efforts spent on the side-channel analysis of commodity PC platforms for the last few years. It was especially realized that the functionality of some microprocessor components like data and instruction cache and branch prediction units cause serious

side-channel leakage. These efforts led to the development of MicroArchitectural Analysis area. MA attacks exploit the microarchitectural components of a processor to reveal cryptographic keys.

Side-channel analysis can be defined as the study of data dependent variations in side-channel information such as execution time and power consumption. These variations either directly give the key value out during a single cipher execution or leak information which can be gathered during many executions and analyzed to compromise the system. The functionality of the aforementioned processor components generates such data dependent variations in execution time and power consumption, which are the subjects of MA.

A cache-based attack, abbreviated to "cache attack" or "cache analysis" from here on, exploits the cache behavior of a cryptosystem by obtaining the execution time and/or power consumption variations generated via cache hits and misses, cf. [25, 23, 27, 6, 7, 9, 20, 26, 10, 36, 11, 37, 38, 24]. The cache vulnerability of computer systems has been known for a long time, c.f. [16, 19, 17], however actual realistic and practical cache attacks were not developed until recent years. Cache analysis techniques enable an unprivileged process to attack another process, e.g., a cipher process, running in parallel on the same processor as done in [25, 23, 27]. Furthermore, some of the cache attacks can even be carried out remotely, i.e., over a local network [6].

The previous cache attacks, excluding instruction-cache attacks which are fundamentally different than data-cache attacks, are data-path attacks, i.e., exploit the data access patterns of a cipher. The memory accesses of S-box based ciphers are key-dependent. Cache attacks try to reveal these memory access patterns by analyzing the cache statistics of the cipher execution. These cache statistics include the number of cache hits and misses, the cache lines modified by the cipher, and such. An unprivileged malicious party cannot directly obtain these statistics [8], but can observe side-channel leakage to estimate these values. For example, the execution time of AES software implementations is directly related to the total number of cache hits and misses that occur during the encryption, cf. [35], and can be measured to determine these statistics.

The second type of MA we have seen is the Branch Prediction Analysis (BPA), which also includes a powerful variant called Simple Branch Prediction Analysis (SBPA) [4, 5]. It was proven that "a carefully written spy-process running simultaneously with an RSA-process, is able to collect during one *single* RSA signing execution almost all of the secret key bits". They could successfully apply an attack on the exponentiation phase of a simple RSA implementation as a case study. They ran an independent process to spy on RSA execution relying on the simultaneous-multithreading (SMT) capability of the platform. The concept introduced in [4] has recently been verified by Andre Seznec, who is a famous expert on branch prediction [43]. As stated in [3, 4], the actual power of SBPA is not limited to this basic application on RSA exponentiation. SBPA has a potential to determine the entire execution flow of a target process on *almost any* execution environment, i.e., with or without SMT. The reasons of this powerful claim, which are based on the results of previous studies on MA, were given in [3].

Following this interesting research vector, Acıiçmez has also introduced another MA type: I-cache Analysis [1]. Similar to BPA, I-cache analysis is used to reveal the execution flow of a target process. An adversary runs a spy process simultaneously or quasi-parallel with the cipher and detects the changes occur in the instruction cache.

[8] In fact, current processors have special registers inside the chip that count and store such statistics. These registers are intended to be used for performance monitoring purposes and fortunately require special privileges to be read. Without this requirement of high privilege, the potential power of a malicious party would be devastating. For further information on performance counters and performance monitoring events, see [42]

The spy process oriented MA attacks rely on the fact that the execution of cipher processes leaves persistent changes in the state of shared resources like cache and branch target buffer. In other words, the cipher execution leaves "footprints" on the observable state, i.e. the so-called metadata, of these resources and an unprivileged spy process can keep track of these footprints if it runs on the same processor in parallel with the cipher. Being able to spy on these states and especially the ability to detect the changes of these states as a function of time, the adversary can reveal the execution flow (and also the memory access patterns) of cryptosystems.

The most threatening feature of MA is its broad application range. These attacks can compromise security systems even in the presence of sophisticated protection mechanisms like sandboxing and virtualization because all of these attacks exploit deep processor functionalities which are below the trust architecture boundary of these security mechanisms. All of the aforementioned MA attacks are pure-software based and do not require any privilege on the system which makes them easy to deploy but hard to detect. Anyone with malicious intentions (and of course, the required skill set to implement such spy processes) can apply these attacks on multi-user systems, VPNs, virtual machines, and such systems that allow the parallel execution of processes of different parties. Due to the limited space, we cannot give further details of MA in this paper. We refer the interested readers to [5, 4, 3, 22, 9, 25, 6] for further information.

3 Actual Practical Implementations of the Attack on Extra Reduction via MA

Motivated from the attacks given in [39, 30, 29], we decided to realize these attacks via MicroArchitectural Analysis. It was already proven in [4, 1] that SBPA and I-cache Analysis can reveal the execution flow of RSA, which is exactly what we need in order to bring these theoretical extra reduction attacks into practice. We preferred to use OpenSSL as our target software RSA implementation and I-cache analysis as our tool to detect the occurrences of extra reduction steps. The results of our experiments are given in the following section. In this section, we outline the basics of I-cache analysis and describe our approach of using I-cache analysis on OpenSSL to gather er-vector values.

3.1 Overview of I-cache Analysis

I-cache analysis rely on the fact that instruction cache misses increase the execution time of code section. Instruction cache (a.k.a. I-cache) is a small buffer between the main memory and the processor core which provides the processor fast and easy access to the most frequently executed instructions. When the processor needs to read some instructions from the main memory, it first checks to see if they are already in I-cache. If they are already in I-cache (a.k.a. cache hit), the processor immediately uses these "cached" instructions instead of accessing the main memory, which has a significantly longer latency compared to an I-cache. Otherwise (a.k.a. cache miss), the instructions are read from the memory and a copy of them are stored in I-cache. Each I-cache miss mandates an access to a higher level memory, i.e., a higher level cache or main memory, and thus results in additional execution time delays.

In I-cache analysis, an adversary needs to execute a spy code, which keeps track of the changes in the state of I-cache, i.e., metadata, during the execution of a cipher process. A spy code / process can run simultaneously or quasi-parallel with the cipher process and determine which instructions are executed by the cipher. To give a concrete example, we can consider the exponentiation algorithm of OpenSSL. Reference [1] takes advantage of the fact that OpenSSL employs sliding window

exponentiation which generates a key dependent sequence of modular operations in RSA [9]. Furthermore, OpenSSL uses different functions to compute modular multiplications and square operations. [1] shows that if an adversary can run a spy routine and evict either one of these functions, he can easily determine the operation sequence of RSA.

In his attack scenario, a "protected" crypto process executes RSA signing/decryption operations and an adversary executes a spy process simultaneously or quasi-parallel with this cipher. The spy routine

1. continuously executes a number of dummy instructions, and
2. measures the overall execution time of all of these instructions

in such a way that these dummy instructions precisely maps to the same I-cache location with the instructions of multiplication function. In other words, the adversary creates a conflict between the instructions of the multiplication function and the spy routine. Because of this I-cache conflict, either the spy or multiplication instructions can be stored in I-cache at any time. Therefore, whenever the cipher process executes the multiplication function, the instructions of the spy routine have to be evicted from I-cache. This eviction can easily be detected by the spy routine because when it reexecutes its instructions the overall execution time will suffer from I-cache misses. Thus, the spy can determine when the multiplication function is executed. This information directly reveals the operation sequence of RSA if the square & multiply exponentiation algorithm is used. For further details of I-cache analysis and this particular attack, we refer the reader to [1].

3.2 Our Approach

Our approach in this paper is similar to the approach presented in [1]. If an adversary can determine the occurrences of modular multiplications by creating I-cache conflicts between the spy routine and multiplication function, he can also create conflicts with the instruction executed during extra reduction step and reveal the er-vectors [10].

OpenSSL library employs Montgomery multiplication algorithm as stated above. During a Montgomery operation, OpenSSL first calls either multiplication or square functions from BIGNUM library[11] and then reduces the result to the modulus via Montgomery reduction function[12]. At the end of the reduction after each Montgomery operation, whether it is a multiplication or square, the intermediate result is compared with the modulus to decide if an extra reduction is needed.

The extra reduction step consists of a multiprecision subtraction and OpenSSL realizes it as a call to BIGNUM library's unsigned multiprecision subtraction function[13]. This function is used/called only for the extra reduction step during the course of the RSA signing/decryption operation. In other words, there is not any other location in OpenSSL's code for RSA decryption

[9] OpenSSL implements both sliding window and fixed window exponentiations. Sliding window exponentiation is the default algorithm in OpenSSL. Fixed window exponentiation (which is slower than the sliding window) were implemented as an optional protection to cache attacks (and now it also provides protection against branch prediction attacks). The choice whether to turn the cache attack and/or branch prediction attack countermeasures on (including the fixed window exponentiation) is given to the user.

[10] An er-vector is an ordered tuple of 1 and 0 values, e.g., $(0, 0, 1, 0, 1, ..., 0)$, which shows the occurences of extra reduction steps. We denote the occurrence of an extra reduction step with a 1 in er-vectors and the value 0 indicates that extra reduction is not performed for that Montgomery operation. See further [30] and Section 4.2.

[11] BIGNUM is a software library integrated into OpenSSL and it is responsible for multiprecision operations.

[12] BN_from_montgomery() function

[13] BN_usub() function

that calls this function. Therefore, whenever a cipher process executes this function, it must be performing extra reduction step. Hence, it is sufficient to create a spy routine that has I-cache conflicts with this function in order to detect the occurrences of extra reduction steps[14].

Following this basic approach, we implemented such a spy function as described in [1] and also considering the above difference. Our spy function executes some dummy instructions and measures their overall running time. These dummy instructions have a conflict with the extra reduction routine of OpenSSL, which allows the spy to detect the occurrences of extra reduction steps, i.e., er-vector values. After enough er-vector vectors are gathered, it becomes feasible to break the cipher. One has to transfer the attack from [30], which has already been proven to be correct there, to the concrete situation and apply it on the gathered er-vectors.

However, the spy measurements are not perfectly clear and there is a noise factor to consider. It is already shown in [4] that a carefully-written spy process can get very clean results. But [4] also states that such clean results cannot always be collected and an adversary needs to make some trials to get clean results. Therefore, an adversary will have to deal with this noise factor, but the theoretical attacks explained above tolerates some error rates. Thus the problem for an adversary becomes how clearly he can detect the extra reduction steps, i.e., gathering er-vectors with a low enough error rate. A high error rate affects the necessary sample size of the attacks and may make it infeasible to compromise the cipher in some cases. However, we will show in the next section that the measurements collected by a spy function is clean enough to practically apply these attacks with a relatively small sample size.

4 Experimental Details, Mathematical Background and Empirical Results

We performed two different phases of experiments in this project. The objective of the first phase was to gather er-vectors via MA, i.e., I-cache Analysis in this case. The second phase consisted of determining the success rate of our attack for different sample sizes and different error rates.

4.1 Experimental Details

We compiled the RSA decryption function of OpenSSL (the latest version) with all the available countermeasures enabled. We disassembled the executable file to determine the logical addresses of BIGNUM multiprecision unsigned subtraction function instructions. GNU Project debugger (i.e. gdb) has two functions, "info line" and "disas", that we used for this task. Then we implemented our spy routine according to these logical addresses and also considered the parameters of the I-cache architecture on our platform.

Then we carried out the first experimental phase by letting the spy function run during the execution of RSA signing operation. To simplify our setup and reduce the necessary experimental efforts, we used a simple trick in this phase. Instead of using a stand-alone spy process and relying on Operating System functionalities as done in [23] or exploiting SMT-capability of processor architectures as done in [27, 25, 4], we called the spy routine inside the RSA process with a certain

[14] In fact, an adversary also needs to be sure that instructions of the spy routine do not get completely evicted by other functions such as BIGNUM's multiplication and square functions. This possible situation was not a problem in our case, because the executable code did not have such conflicts between these functions when we compiled OpenSSL using gcc compiler. A compiler is expected to remove possible internal cache conflicts to increase the performance of the code. Internal cache conflicts, in this context, indicate the conflicts occur between the instructions that belong to the same process.

frequency, i.e., after each exponentiation step. The other options would require special manual handling of each single measurement and would drastically increase the necessary efforts spent in this experiment, because we needed to analyze a large number of measurements to get an accurate estimation of the error rates. In more realistic attack scenarios that use stand-alone spy processes, an adversary may (and most likely will) encounter an error rate higher than our experimental results. However, according to an analysis on cache attacks from Neve which is given in [22], the theoretical and actual results taken from such a spy process are indeed very close to each other. Therefore, we expect the actual error rates in a spy process's measurements to be only slightly higher than our estimated error rates.

We must also mention that these error rates depend on several parameters including the actual platform, operating system and other software components of the system, the implementation of the spy process as well as the cipher. Higher error rates do not necessarily nullify the validity of these attacks since the optimal guessing strategy for correctly observed er-vectors (cf. Theorem 1) seems to tolerate error rates up to about 4 - 5%, and even guessing strategies exist that take classification errors explicitly into account (cf. Theorem 2). Thus, *we do not claim that our results perfectly reflect the performance of these attacks in every possible scenario. We only prove in this paper that er-vector attacks coupled with MA techniques create a valid and severe threat to software systems.*

We used the possibility of performing the same exact measurement many times and taking the average to decrease the measurement noise, i.e., error rate. OpenSSL uses the same blinding factor 32 times by default before updating it. Therefore, an adversary can force the system to perform RSA decryption for the same base and identical blinding values $t \leq 32$ many times and measure each of these t operations. For example, he can exploit SSL handshake protocol as done in [12, 8]. Using the average of more than 1 measurement decreases the noise and thus reduces the error rate as we will show in the next subsection.

The second phase of our experiments was to determine the success rate of er-vector attack for several sample sizes and to estimate the effect of different error rates.

We performed our experiments with the latest version of OpenSSL (v.0.9.8e), which uses Chinese Remainder Theorem (CRT), Montgomery's multiplication algorithm ([21], 14.36) and base blinding to compute the modular exponentiation $x \mapsto x^d (\mathrm{mod}\, n)$. We consider both of the exponentiation algorithms implemented in current OpenSSL version: fixed windows and sliding windows exponentiation ([21], 14.82 and 14.85). The fixed-window exponentiation algorithm (which is slower than the sliding window) were implemented as an optional protection to cache attacks (and now it also provides protection against branch prediction attacks). The choice whether to turn the cache attack and/or branch prediction attack countermeasures on (including the fixed window exponentiation) is given to the user. The size of the windows used in both exponentiation algorithms depends on the key size in OpenSSL. For common key sizes (1024 and 2048 bits), the windows are 5-bit long, and thus we will focus on this particular case in our paper.

4.2 Mathematical Background and Experimental Results

In this section we explain the mathematical background and present our experimental results.

Fixed Window Exponentiation Algorithm As usual, $n = p_1 p_2$ and R denotes the Montgomery constant while $\mathrm{MM}(a, b; n) := abR^{-1} (\mathrm{mod}\, n)$ stands for the Montgomery multiplication of a and b. The computation of $x^d (\mathrm{mod}\, n)$ is carried out in several phases:

1. Base blinding: $x_b := xA (\mathrm{mod} n)$ where A and $B := A^{-d} (\mathrm{mod}\, n)$ are the current blinding values.
2. Compute $x_b^d (\mathrm{mod}\, p_1)$
   a) group the binary representation of $d_{(1)} := d (\mathrm{mod}\, (p_1 - 1)) = (d'_{w-1}, \ldots, d'_0)_2$ into non-overlapping blocks of length $wsize$, starting from the least significant bit d'_0. This gives wsize-bit integers $D_{v-1}, \ldots, D_0$ with $v := \lceil w/wsize \rceil$.
   b) $x_{b,1} :\equiv x_b (\mathrm{mod}\, p_1)$
   c) Exponentiation algorithm **1**: Fixed windows

   u_0 := MM(1, R^2(mod p_1);p_1) (= $R\,(\mathrm{mod}\, p_1)$)
   u_1 := MM($x_{b;1}$, R^2(mod p_1);p_1) (= $x_{b;1} R\,(\mathrm{mod}\, p_1)$)
   for j := 2 to $2^{wsize} - 1$ do u_j:=MM(u_{j-1},u_1;p_1)
   $temp$:= u_0
   for i := $v - 1$ downto 0 do {
     for j := 1 to $wsize$ do $temp$:=MM($temp$,$temp$;p_1)
     $temp$:= MM($temp$,u_{D_i};p_1)}
   return MM($temp$,1) (= $x_{b;1}^d (\mathrm{mod}\, p_1) = x_b^d (\mathrm{mod}\, p_1)$)

   Note: $u_0, \ldots, u_{2^{wsize}-1}$ denote the table values.
3. Compute $x_b^d (\mathrm{mod}\, p_2)$ analogous to Step 2
4. Compute $x^d (\mathrm{mod}\, n)$
   a) CRT step: Compute $x_b^d (\mathrm{mod}\, n)$ from $x_b^d (\mathrm{mod}\, p_1)$ and $x_b^d (\mathrm{mod}\, p_2)$.
   b) "Remove" blinding: $y := x_b^d B (\mathrm{mod}\, n)$
5. (eventually) update A and B

Remark 1. We point out that fixed windows exponentiation algorithm from above can somewhat be speeded up. In fact, $\mathrm{MM}(temp, u_0; p_1) = temp$ has no computational effect, and also the beginning of the exponentiation phase can be implemented more efficiently. We treat a related variant of this exponentiation algorithm in the next subsubsection.

The adversary's goal is clearly to determine the secret exponent d. We first note that it is sufficient to determine $d_{(1)}$ since

$$y - x^{d_{(1)}} \equiv 0 (\mathrm{mod}\, p_1), \quad \text{i.e. } gcd(y - x^{d_{(1)}} (\mathrm{mod}\, n), n) = p_1 \tag{1}$$

for any known plaintext / ciphertext pair $(x, y = x^d (\mathrm{mod}\ n))$. The attacker has to recover the types of operation $T(1), T(2), \ldots, T(M)$ of the Montgomery operations during the exponentiation phase. To increase the readability of the paper we concentrate on $wsize = 5$, the case we are interested in. Of course, all assertions can immediately be transferred to any arbitrary window size. Essentially, it remains to substitute 31 by $2^{wsize} - 1$.

Clearly, $T(i) \in \Theta = \{\text{`}S\text{ '}, \text{`}M_0\text{'}, \text{`}M_1\text{'}, \ldots, \text{`}M_{31}\text{'}\}$ where 'S' says that the i^{th} Montgomery operation in the exponentiation phase is a squaring while 'M_k' stands for the multiplication with table entry u_k. The adversary grounds his decisions on the exponentiation of bases $x_1, \ldots, x_N$. For now, assume that the adversary knows which operations in the table initialization phase and in the exponentiation phase require extra reductions (ERs). More formally, $w'_{i(k)} = 1$, resp. $w'_{i(k)} = 0$ mean that the i^{th} operation in the table initialization phase (= computation of u_i for base x_k) requires an extra reduction, resp. requires no extra reduction. Similarly, $w_{i(k)} = 1$, resp. $w_{i(k)} = 0$

mean that the i^{th} operation in the exponentiation phase requires an extra reduction, resp. requires no extra reduction. In the following we outline the general procedure, explain the main steps.

As usually, we interpret the observations $w'_{i(k)}$ and $w_{i(k)}$ as realizations of suitably defined random variables $W'_{i(k)}$ and $W_{i(k)}$. As already shown in [33] and [31] we have

$$\mathrm{P}rob(W_{i(k)} = 1) = \begin{cases} \frac{1}{3}\frac{p_1}{R} & \text{if } T(i) = \text{`}S\text{'} \\ \frac{u_j}{2p_1}\frac{p_1}{R} & \text{if } T(i) = \text{`}M_j\text{'}. \end{cases} \tag{2}$$

We note that both probabilities depend on the ratio p_1/R which is yet unknown to the attacker. Recall that the positions of the $\#sq = 5\lceil \log_2(d_{(1)})/5 \rceil$ many squarings are well-known in this fixed windows variant since they do not depend on $d_{(1)}$, and hence the first line of (2) can be used to estimate p_1/R. (For all bases $x_1, \ldots, x_N$ count the ERs for all squarings in the sample and multiply this number by $3(N\#sq)^{-1}$.) Using also the second line of (2) the game was easy if the adversary knew the ratios $u_{j(1)}/p_1, \ldots, u_{j(N)}/p_1$ for each $0 \leq j \leq 31$. Due to base blinding (and the use of CRT) the adversary yet does not know these values.

The key is a formal treatment which analyzes the distribution of the random variables $W'_{1(k)}, \ldots, W'_{31(k)}$ and $W_{1(k)}, W_{2(k)}, \ldots$. We point out that $s'_{0(k)} := x_{b,1}/p_1$, $s'_{i(k)} := u_{i(k)}/p_1$ for $i \in \{1, \ldots, 31\}$ and $s_{i(k)} := temp_{i(k)}/p_1$ assume values in the unit interval $[0, 1)$ where $temp_{i(k)}$ stands for the i^{th} temp value in the exponentiation phase for the base x_k. In particular,$s_{0(k)} = u_{0(k)}/p_1$. We assume that the values $s'_{0(k)}, \ldots, s'_{31(k)}$ and $s_{1(k)}, s_{2(k)}, \ldots$ are taken on by $[0, 1)$-valued random variables $S'_{0(k)}, \ldots, S'_{31(k)}$ and $S_{1(k)}, \ldots, S_{M(k)}$. Lemma 1(iii) in [30] says that the random variables

$$S'_{0(k)}, \ldots, S'_{31(k)}, S_{1(k)}, \ldots, S_{M(k)} \text{ are independent and uniformly distributed on } [0, 1) \tag{3}$$

which matches with the intuition that the intermediate temp values 'spread' wildly over Z_{p_1}. It is even more interesting that the random variables $W'_{i(k)}$ and $W_{i(k)}$ can be expressed in terms of $S'_{i-1(k)}$ and $S'_{i(k)}$, resp in terms of $S_{i-1(k)}$ and $S_{i(k)}$. More precisely, Lemma 1(iii) in [30] implies

$$W'_{i(k)} := \begin{cases} 1_{S'_{1(k)} < S'_0(R^2 \,(\bmod\, p_1)/p_1)(p_1/R)} & \text{for } i = 1 \\ 1_{S'_{i(k)} < S'_{i-1(k)} S'_{1(k)} p_1/R} & \text{for } 2 \leq i \leq 31 \quad \textit{and} \end{cases} \tag{4}$$

$$W_{i(k)} := \begin{cases} 1_{S_{i(k)} < S^2_{i-1(k)} p_1/R} & \text{if } T(i) = \text{`}S\text{'} \\ 1_{S_{i(k)} < S_{i-1(k)} S'_{j(k)} p_1/R} & \text{if } T(i) = \text{`}M_j\text{'}. \end{cases} \tag{5}$$

Here $1_A(x)$ denotes the indicator function, i.e. $1_A(x) = 1$ iff $x \in A$ and $= 0$ otherwise. The stochastic process $W_{1(k)}, W_{2(k)}, \ldots, W_{M(k)}$ is non-stationary and dependent which clearly complicates its analysis. However, it allows an exact solution of our problem. The next goal is to compute the joint probabilities

$$p_\theta(w'_{1(k)}, \ldots, w'_{31(k)}, w_{i(k)}) := \mathrm{P}rob_\theta(W'_{1(k)} = w'_{1(k)}, \ldots, W'_{31(k)} = w'_{31(k)}, W_{i(k)} = w_{i(k)}) \tag{6}$$

for all possible types of operation $\theta \in \Theta = \{\text{`}S\text{'}, \text{`}M_0\text{'}, \text{`}M_1\text{'}, \ldots, \text{`}M_{31}\text{'}\}$. Due to (4) and (5) the values $w'_{1(k)}, \ldots, w'_{31(k)}$ and $w_{i(k)}$ can be characterized by conditions on $s'_{0(k)}, \ldots, s'_{31(k)}$ and $s_{i-1(k)}, s_{i(k)}$. For instance, $w'_{j(k)} = 1$ iff $s'_{j(k)} < s'_{j-1(k)} s'_{1(k)} p_1/R$, independent of θ while the impact of $w_{i(k)}$ on $s_{i-1(k)}$ and $s_{i(k)}$ clearly depends on θ (cf. (5)). Altogether, if $T(i) = \theta$, observing

$(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})$ is equivalent to $(s'_{0(k)},s'_{1(k)},\ldots,s'_{31(k)},s_{i-1(k)},s_{i(k)})\in A_\theta(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})$ for a well-defined subset $A_\theta(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})\subset[0,1)^{34}$ that only depends on $(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})$ and the hypothesis θ. The table entry u_0 plays an exceptional role as it does not depend on the basis x_k but is constant for all bases (see (9) below). Due to (3) the joint probability in (6) is given by the volume of the set $A_\theta(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})$.

$$p_\theta(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})=\int_0^1\int_{a'_1}^{b'_1}\cdots\int_{a'_{31}}^{b'_{31}}\int_0^1\int_{a_{\theta,i}}^{b_{\theta,i}}1\,ds_i ds_{i-1}ds'_{31}\cdots ds'_0 \tag{7}$$

with integration boundaries

$$(a'_1,b'_1)=\begin{cases}(0,s'_0(R^2(\bmod p_1))/R) & \text{if } w'_{1(k)}=1\\ (s'_0(R^2(\bmod p_1))/R,1) & \text{if } w'_{1(k)}=0.\end{cases} \tag{8}$$

$$i\in\{2,\ldots,31\}:\ (a'_i,b'_i)=\begin{cases}(0,s'_{i-1}s'_1p_1/R) & \text{if } w'_{i(k)}=1\\ (s'_{i-1}s'_1p_1/R,1) & \text{if } w'_{i(k)}=0.\end{cases}$$

$$(a_{\theta,i},b_{\theta,i})=\begin{cases}(0,s^2_{i-1}p_1/R) & \text{if } \theta=\text{`}S\text{'} \text{ and } w_{i(k)}=1\\ (s^2_{i-1}p_1/R,1) & \text{if } \theta=\text{`}S\text{'} \text{ and } w_{i(k)}=0\\ (0,s_{i-1}s'_jp_1/R) & \text{if } \theta=\text{`}M_j\text{'},\ j>0 \text{ and } w_{i(k)}=1\\ (s_{i-1}s'_jp_1/R,1) & \text{if } \theta=\text{`}M_j\text{'},\ j>0, \text{ and } w_{i(k)}=0.\end{cases}$$

We note that the hypothesis $\theta=' S'$ is only relevant for the exponentiation algorithms treated in the next subsections. Moreover, $p_{\prime M'_0}(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})$ does not depend on $w'_{1(k)},\ldots,w'_{31(k)}$. More precisely,

$$p_{\prime M'_0}(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})=\mathrm{P}rob(W'_{j(k)}=w'_{j(k)}\text{ for }1\le j\le 31)\cdot\mathrm{P}rob_{\prime M'_0}(W_{i(k)}=w_{i(k)}) \tag{9}$$

with

$$\mathrm{P}rob(W'_{j(k)}=w'_{j(k)}\text{ for }1\le j\le 31)=\int_0^1\int_{a'_1}^{b'_1}\cdots\int_{a'_{31}}^{b'_{31}}1\,ds'_{31}\cdots ds'_0\,, \tag{10}$$

and from (2) we obtain $\mathrm{P}rob_{\prime M'_0}(W_{i(k)}=1)=R(\bmod p_1)/2R=1(\bmod(p_1/R))/2$. We have already explained how to estimate the ratio p_1/R.

Since all table entries are equally likely, and since all guessing errors are equally harmful the optimal decision strategy is given by the maximum likelihood estimator (cf. next subsection where this is definitely not the case). Recall that for Exponentiation algorithm **1** the adversary knows where the squarings are performed.

Theorem 1. *[Exponentiation algorithm **1**, wsize=5] Let $\boldsymbol{w}_{i(k)}:=(w'_{1(k)},\ldots,w'_{31(k)},w_{i(k)})$. Assume that all observed vectors $\boldsymbol{w}_{i(1)},\ldots,\boldsymbol{w}_{i(N)}$ are correct and that in Montgomery operation i the temp value is multiplied by any table value u_j. The optimal strategy to guess $T(i)$ is to decide for that hypothesis $\theta'\in\Theta_0:=\Theta\setminus\{\text{`}S\text{'}\}$ that maximizes*

$$\prod_{k=1}^{N}p_\theta(\boldsymbol{w}_{i(k)}). \tag{11}$$

Recall that the er-vectors $ER_{(k)} = (w'_{1(k)}, \ldots, w'_{31(k)}, w_{1(k)}, \ldots, w_{M(k)})$ are the adversary's only information ($k = 1, \ldots, N$). Unlike in [30, 28], in our attack we are faced with erroneous observations, i.e. with flipped components of the er-vector. The terms $\mu(0 \mid 1)$ and $\mu(1 \mid 0)$ denote the probabilities to observe $w_{i(k)} = 0$ although an extra reduction is performed, resp. the probability to observe $w_{i(k)} = 1$ although no extra reduction is necessary. Practical experiments showed that the misclassification rate does not depend on the position of the respective Montgomery operation. The default setting in the current OpenSSL library v.0.9.8e keeps the blinding values A and B constant for consecutive 32 exponentiations. Consequently, for any base x_k the adversary may repeat measurements $t \leq 32$ times under identical blinding values, i.e. under identical conditions. The results shown in Table 1 are average error rates calculated based on the measurements collected during 1000 decryptions with random ciphertext under each of 10 different random 1024-bit RSA keys Increasing the value of t reduces the error rates, which become very close to 0 when t reaches 32.

t	$\mu(0 \mid 1)$	$\mu(1 \mid 0)$
1	0.1052	0.0021
2	0.0872	0.0010
4	0.0536	0.0582
8	0.0294	0.0239
16	0.0080	0.0063
32	0.0026	0.0062

Table 1. The average error rates for different values of t

With $t = 16$ we generated $N = 2000$ er-vectors (i.e., we perfomed $tN = 16 \cdot 20000 = 32000$ measurements), belonging to different bases $x_1, \ldots, x_{2000}$. We randomly selected subsets of size $N_1 < N$ and performed our attack on basis of the er-vectors that were contained in these subsets. Table 2 shows our results, underlining the practical feasibility of our attack even with only 10000 measurements ($N_1 = 600$). Of course, 1 guessing error (also 2 or maybe even 3) can be corrected by exhaustive search, checking the correctness of all guesses by (1). Note that if a false candidate θ^* maximizes (11) the corresponding correct hypothesis θ is usually also highly ranked.

N_1	guessing errors (average)	# attacks with ≤ 1 guessing errors
500	1.90	44/100
600	0.83	85/100
700	0.25	99/100
800	0.12	99/100
900	0.01	100/100
1000	0.02	100/100

Table 2. Practical experiments (Guessing strategy according to Theorem 1)

For window size 5 the adversary needs $|\Theta \setminus \{'S'\}| \cdot N \cdot 2 = 32 \cdot N \cdot 2$ joint probabilities $p_\theta(w'_{1(k)}, \ldots, w'_{31(k)}, j)$ (with $j \in \{0, 1\}$) Since $Prob_\theta(W_{i(k)} = 0 \mid \cdot) = 1 - Prob_\theta(W_{i(k)} = 1 \mid \cdot)$ this

essentially requires the computation of $32 \cdot N$ conditional probabilities $p_\theta(0 \mid w'_{1(k)}, \ldots, w'_{31(k)})$, i.e. of $32N$ many 34-dimensional integrals of type (7) and of N 32-dimensional integrals of type (10). For window size 5 these computations constitute the essential part of the workload. Principally, these computations are not difficult since (7) and (10) split into 34, resp. 32-dimensional, consecutive one-dimensional integrations of polynomials. Since each $w'_j = 0$ principally doubles the number of monomials for window size 5 the computations are yet memory- and time-consuming. In particular, identical monomials should be summarized regularly.

The simulation studies from the next subsection (cf. Table 3 and Table 4) indicate that the optimal decision strategy from Theorem 1 tolerates misclassification rates up to 4 to 5 per cent. We point out that the situation is even better than for Exponentiation algorithm **2** since the adversary need not guess the positions of the squarings. Surprisingly, if the adversary (roughly) knows the misclassification rates he can take them explicitly into account.

Theorem 2. *[Exponentiation algorithm **1**, wsize=5] Assume that in Montgomery operation i the temp value is multiplied by any table value u_j. Assume further that $\mu(0 \mid 1) = \mu_{01}, \mu(1 \mid 0) = \mu_{10} \geq 0$. For $\boldsymbol{a} = (a_1, \ldots, a_{32}) \in \{0,1\}^{32}$ define $C_0(\boldsymbol{a}) := \{j \leq 32 \mid a_j = 0\}$ and $C_1(\boldsymbol{a}) := \{j \leq 32 \mid a_j = 1\}$ The optimal strategy to guess $T(i)$ is to decide for that hypothesis $\theta' \in \Theta \setminus \{'S'\}$ that maximizes*

$$\prod_{k=1}^{N} \sum_{\boldsymbol{w}^* \in \{0,1\}^{32}} p_\theta(\boldsymbol{w}^*) \quad \times \tag{12}$$
$$\times \ \mu_{10}^{|C_0(\boldsymbol{w}_{i(k)}) \setminus C_0(\boldsymbol{w}^*)|} (1-\mu_{10})^{|C_0(\boldsymbol{w}_{i(k)}) \cap C_0(\boldsymbol{w}^*)|} \mu_{01}^{|C_1(\boldsymbol{w}_{i(k)}) \setminus C_1(\boldsymbol{w}^*)|} (1-\mu_{01})^{|C_1(\boldsymbol{w}_{i(k)}) \cap C_1(\boldsymbol{w}^*)|}.$$

Proof. Due to the properties of the stochastic process $S'_{1(k)}, \ldots, S_{M(k)}$, $k = 1, \ldots, N$, we may assume that er-vectors which belong to different bases or to different blinding values are independent. The optimal decision strategy maximizes

$$\prod_{k=1}^{N} \mathrm{P}rob_\theta(\boldsymbol{w}_{i(k)} \text{ observed}) \qquad \text{with}$$
$$\mathrm{P}rob_\theta(\boldsymbol{w}_{i(k)} \text{ observed}) = \sum_{\boldsymbol{w}^* \in \{0,1\}^{32}} \mathrm{P}rob_\theta(\boldsymbol{w}^* \text{ correct}) \mathrm{P}rob_\theta(\boldsymbol{w}_{i(k)} \text{ observed} \mid \boldsymbol{w}^* \text{ correct}).$$

Clearly, $\mathrm{P}rob_\theta(\boldsymbol{w}^* \text{ correct}) = p_\theta(\boldsymbol{w}^*)$ while the second term does not depend on θ but only on $\boldsymbol{w}_{i(k)}$, $\boldsymbol{w}^*$, μ_{01} and μ_{10}. Elementary considerations complete the proof of (12).

We first note that for $\mu_{01} = \mu_{10} = 0$ formulae (11) and (12) coincide. The drawback of the (12) is that it requires the computation of $32 \cdot 2^{32}$ probabilities $p_\theta(\boldsymbol{w}^*)$ which is gigantic. Recall that we concentrated on $wsize = 5$ to increase the readability of the document. Clearly, Theorem 1 and Theorem 2 can immediately be adjusted to arbitrary window size (31 corresponds to $2^{wsize} - 1$ in the general case). We point out that for window size 4, for instance, the situation is much better since it requires only $16 \cdot 2^{16}$ probabilities, and each probability can be calculated much faster. For window size 2 only $4 \cdot 2^4$ such probabilities are necessary. On the other hand, if the misclassification rates μ_{01}, μ_{10} are moderate those $\boldsymbol{w}^*$ with large Hamming distance to $\boldsymbol{w}_{i(k)}$ give only little contribution since $\mathrm{P}rob(\boldsymbol{w}_{i(k)} \text{ observed} \mid \boldsymbol{w}^* \text{ correct })$ is very small. Consequently, the adversary may only consider those $\boldsymbol{w}^*$ in (12) that have Hamming distance 1 or 2, giving an *approximately optimal decision strategy*.

Of course, in 'real-life' attacks the adversary does not know the values μ_{01} and μ_{10}. However, this does not constitute a serious problem. As already pointed out the computational bottleneck is the computation of the probabilities $p_\theta(\boldsymbol{w}^*)$. Once these probabilities have been computed the adversary can experiment with different values $\widetilde{\mu}_{01}$ and $\widetilde{\mu}_{10}$ since only the conditional probabilities $Prob(\boldsymbol{w}_{i(k)}$ is correct $\mid \boldsymbol{w}^*$ correct) have to re-calculated which is an easy task. (This concerns both (12) and the approximate decision strategy proposed in the previous paragraph.) The number of errors may serve as a simple quality measure for the suitability of the guessed values $\widetilde{\mu}_{01}$ and $\widetilde{\mu}_{10}$, i.e. whether these values are sufficiently close to μ_{01} and μ_{10}.

Fixed Window Exponentiation Algorithm (related variant) In this subsection we treat a related variant of the fixed windows algorithm which is used in many implementations. It differs from the fixed windows algorithm variant implemented in OpenSSL v.0.9.8e in Step 2.c).

2. Compute $x_b^d (\mathrm{mod}\, p_1)$

   c) Exponentiation Algorithm **2**: Fixed windows (related variant)

   u_1 := MM($x_{b;1}$, R^2 (mod p_1);p_1) (= $x_{b;1}R \,(\mathrm{mod}\, p_1)$)
   for j := 2 to $2^{wsize} - 1$ do u_j:=MM(u_{j-1},u_1;p_1)
   $temp$:= $u_{D_{v-1}}$
   for i := $v-2$ downto 0 do {
     for j := 1 to $wsize$ do $temp$:=MM($temp$,$temp$;p_1)
     if ($D_i > 0$) $temp$:= MM($temp$,u_{D_i};p_1)}
   return MM($temp$,1) (= $x_{b;1}^d (\mathrm{mod}\, p_1) = x_b^d (\mathrm{mod}\, p_1)$)

   Note: $u_1, \ldots, u_{2^{wsize}-1}$ denote the table values.

This exponentiation algorithm is closely related to the OpenSSL implementation. However, when performing an attack some important differences have to be considered. In the following we explain these differences. As in the previous section we concentrate on the case wsize=5.

For zero windows (00000) no multiplication with a table entry is performed. Consequently, the adversary does not know the positions of the squarings, i.e. he has to guess $T(i) \in \Theta_{(2)} = \{$'$S$ ',' M_1',...,' M_{31}'$\}$ *for all* $i \leq M$. Certainly, $T(1) = \cdots = T(5) = T(M-4) = \cdots = T(M-1) =$ 'S ' which gives an estimator for the ratio p_1/R. The equations (2), (3), (4), (5), (6), (7) and (10) also hold in this case. However, the maximum-likelihood estimator is not optimal in this case.

As already pointed out in [30] there are yet further criteria which should be taken into account. First of all, for randomly selected position i it is much more likely that $T(i)$ is a squaring than a multiplication with any table value u_j. Approximately, the probability for $T(i) = \theta$ is given by $\eta($'M_1'$) = \cdots = \eta($'M_{31}'$) = 1/191$ and $\eta($'S '$) = 160/191$ where η denotes the so-called *a priori distribution* (cf. [30], Sect. 5).

The adversary guesses the types of operations $T(1), T(2), \ldots$ independently. (Note that [30] allows the adversary also to guess several types of operations simultaneously, increasing the efficiency of the attack at cost of increasing computational workload (being too large for window size 5).) Then he has to correct eventual guessing errors. Clearly, between two multiplications with table entries must always be a number of squarings that is a multiple of 5. In other words: A guessing sequence ...,' S ',' S ',' M_6',' S ',' S ',' S ',' S ',' M_7',' M_{21}',' S ',' S ',' S ',' S ',' S ',' M_9',' S ',' S ',... cannot be correct. The guessing error can easily be detected and corrected (replace 'M_7' by 'S '). In [30] we distinguished between three types of guessing errors with regard to the workload which

is necessary to detect and correct them (type-a error: 'M_j' in place of 'S', type-b error: 'S ' in place of 'M_j', type-c error: 'M_j' in place of 'M_τ'). Of course, type-a and type-b errors can easily be detected just by checking the guessed 'pattern' of hypotheses unless these errors occur in bulks ($\rightarrow$ *local errors*). Moreover, type-a errors can easily be corrected while this not obvious for type-b errors (although it usually suffices to take the second likely hypothesis following 'S '). In contrast even the detection of type-c errors is not easy. Failed confirmations of the guessed hypotheses ($\rightarrow$ estimator $\widetilde{d}_{(1)}$, using (1)) point to type-c errors. The adversary will replace those decisions for $'M'_j$ that were 'close' ($\rightarrow$ *global errors*). The *loss function* $s(\theta, a)$ quantifies the efforts that are necessary to correct a wrong decision $a \in \Theta$ if θ is the true hypothesis (= true type of operation). Of course, $s(\theta, \theta) = 0$ (no loss for a correct guess). In our simulations below we used the function values $s(\text{'}S\text{ '}, \text{'}M_j\text{'}) = 1.0$, $s(\text{'}M_j\text{'}, \text{'}S\text{ '}) = 2.0$, $s(\text{'}M_j\text{'}, \text{'}M_\tau\text{'}) = 8.0$ for $j \neq \tau$.

Theorem 3. *[Exponentiation algorithm 2, wsize=5] (i) Assume that all observed vectors $\boldsymbol{w}_{i(1)}, \ldots, \boldsymbol{w}_{i(N)}$ are correct. The optimal strategy to guess $T(i)$ is to decide for that hypothesis $\theta' \in \Theta_{(2)}$ that minimizes*

$$\sum_{\theta \in \Theta_{(2)}} s(\theta, \theta')\eta(\theta) \prod_{k=1}^{N} p_\theta(\boldsymbol{w}_{i(k)}). \tag{13}$$

(ii) Assume that $\mu(0 \mid 1) = \mu_{01}, \mu(1 \mid 0) = \mu_{10} \geq 0$. For $\boldsymbol{a} = (a_1, \ldots, a_{32}) \in \{0,1\}^{32}$ define $C_0(\boldsymbol{a}) := \{j \leq 32 \mid a_j = 0\}$ and $C_1(\boldsymbol{a}) := \{j \leq 32 \mid a_j = 1\}$ The optimal strategy to guess $T(i)$ is to decide for that hypothesis $\theta' \in \Theta$ that minimizes

$$\begin{aligned} \sum_{\theta \in \Theta_{(2)}} s(\theta, \theta')\eta(\theta) \prod_{k=1}^{N} \sum_{\boldsymbol{w}^* \in \{0,1\}^{32}} p_\theta(\boldsymbol{w}^*) \;\times \\ \times\; \mu_{10}^{|C_0(\boldsymbol{w}_{i(k)}) \setminus C_0(\boldsymbol{w}^*)|} (1-\mu_{10})^{|C_0(\boldsymbol{w}_{i(k)}) \cap C_0(\boldsymbol{w}^*)|} \mu_{01}^{|C_1(\boldsymbol{w}_{i(k)}) \setminus C_1(\boldsymbol{w}^*)|} (1-\mu_{01})^{|C_1(\boldsymbol{w}_{i(k)}) \cap C_1(\boldsymbol{w}^*)|}. \end{aligned} \tag{14}$$

Remark 2. (i) The second part of Theorem 1 in [30] can be used to guess the initial value D_{v-1}.
(ii) Theorem 3 is the pendant to Theorem 1 and Theorem 2. Of course, we could handle the attack on the OpenSSL implementation in the same formal way. Setting $\eta(\text{'}M_j\text{'}) = 1/32$ and $s(\text{'}M_j\text{'}, \text{'}M_\tau\text{'}) = 1.0$ for $j \neq \tau$ we obtain equivalent estimators to (11) and (12).
(iii) Note that (13) differs from (15) in [30] by the fact that $p(\boldsymbol{w}_{i(k)})$ substitutes the conditional probability $Prob(W_{i(k)} = w_{i(k)} \mid W'_{j(k)} = w'_{j(k)}$ for $j \leq 31)$. Since the condition does not depend on θ it follows from Theorem 1(iii) in [28] that both terms give equivalent estimators.
(iv) The loss function and the a priori distribution have considerable impact on the effectiveness of the guessing strategy, at least for small sample size N. In [28] the success rates for 4-bit windows (with known ratio $modulus/R \approx 0.7$, no CRT) were compared for sample sizes $N = 550$ and $N = 450$. The optimal guessing strategy (11) in [28] yielded success rates of 94% and 67% which dropped down to 74% resp. to 12% for the maximum-likelihood estimator.

We performed extensive simulations with randomly selected bases and randomly selected ER classification errors according to assumed misclassification rates. (Note that the correct er-vector only depends on x_k, p_1, R but not on the concrete implementation. The success rates are shown in Table 3. The simulation studies underline that the optimal guessing strategy from Theorem 3 is to some extent tolerant against ER classification errors. Misclassification rates $\leq 3\%$ can apparently be compensated by increasing the sample size. We point out that the exact success probabilities

depend to some degree on the ratio p_1/R. In our simulation studies we considered $p_1 \approx 0.7 \cdot 2^{512}$ and $R = 2^{512}$. In Table 3 an attack was counted successful if the most likely correction of the local errors yields the correct position of the local errors, and if at most one global error occurred. For $\mu(0 \mid 1) = \mu(1 \mid 0) = 0.0$ and $N = 650$, for instance, we committed 2.2 type-a errors, 1.2 type-b errors, 0.2 type-c errors in average.

$\mu(0 \mid 1)$	$\mu(1 \mid 0)$	Sample Size	Success Rate
0.00	0.00	650	5/5
0.00	0.00	750	15/15
0.01	0.01	1000	3/3
0.01	0.01	1200	7/7
0.02	0.02	2000	4/4
0.03	0.03	3000	3/3

Table 3. Success rates for Exponentiation algorithm **2** with window size 5 (simulation studies)

In our experiments we also considered 4-bit fixed windows. Compared to 5-bit windows, the calculations required only about 3% of the computation time. For 4-bit windows the optimal guessing strategy compensates misclassification rates up to 5% (Table 4). Comparing the figures with $\mu(0 \mid 1) = \mu(1 \mid 0) = 0.03$ for window size 5 we may expect that also for window size 5 misclassification rates up to 4-5%, can be compensated, possibly requiring somewhat larger sample size N.

$\mu(0 \mid 1)$	$\mu(1 \mid 0)$	Sample Size	Success Rate
0.03	0.03	1000	5/15
0.03	0.03	2000	138/141
0.04	0.04	3000	7/11
0.05	0.05	5000	28/52
0.05	0.05	10000	27/30

Table 4. Success rates for Exponentiation algorithm **2** with window size 4 (simulation studies)

Sliding Windows Exponentiation In the following we treat the sliding windows exponentiation algorithm (cf. [21], 14.85), combined with CRT and Montgomery's multiplication algorithm. We note that the Steps 1, 4, and 5 are identical to fixed windows. Since Step 3 is analogous to Step 2 it remains to describe Step 2. As in the fixed windows case we concentrate on wsize=5.

2. Compute $x_b^d \pmod{p_1}$
   a) $x_{b,1} :\equiv x_b \pmod{p_1}$
   b) Exponentiation Algorithm **3**: Sliding windows (window size 5)
   u_1 := MM($x_{b;1}$,R^2(mod p_1);p_1) (= $x_{b;1}R \pmod{p_1}$)
   u_2 := MM(u_1,u_1;p_1)
   for i = 1 to 15 do u_{2i+1} := MM(u_{2i-1},u_2;p_1)

$temp$:= MM(1,R^2(mod p_1);p_1) $\quad (= R \pmod{p_1})$
k:=$w-1$
while $k >= 0$ do{
 if ($d_k = 0$) {$temp$:=MM($temp$,$temp$;p_1); $k--$;}
 else {
  select the smallest $u \in \{k, ..., k-4\}$ with $d_u = 1$
  for $j = k$ downto u do $temp$:=MM($temp$,$temp$;p_1)
  $temp$:=MM($temp$,$u_{(d_k,\ldots,d_u)_2}$;p_1)
  k:=$u-1$ } }
return MM($temp$,1) $\quad (= x_{b;1}^d \pmod{p_1} = x_b^d \pmod{p_1})$

Note: $u_1, u_3, u_5, \ldots, u_{31}$ denote the table values.

The central line is the same as for fixed windows. Hence we only point out the differences. In analogy to (3) we first conclude that the random variables $S'_{1(k)}, S'_{2(k)}, S'_{3(k)}, S'_{5(k)}, \ldots, S'_{31(k)}, S_{0(k)}, S_{1(k)}, \ldots, S_{M(k)}$ are independent and uniformly distributed on $[0,1)$. Further,

$$W'_{i(k)} := \begin{cases} 1_{S'_{1(k)} < S'_0 (R^2 \,(\bmod\, p_1)/p_1)(p_1/R)} & \text{for } i = 1 \\ 1_{S'_{2(k)} < S'^2_{1(k)} p_1/R} & \text{for } i = 2 \\ 1_{S'_{2i+1(k)} < S'_{2i-1(k)} S'_{2(k)} p_1/R} & \text{for } 1 \le i \le 15 \quad \textit{and} \end{cases} \tag{15}$$

$$W_{i(k)} := \begin{cases} 1_{S_{i(k)} < S^2_{i-1(k)} p_1/R} & \text{if } T(i) = \text{`}S\text{'} \\ 1_{S_{i(k)} < S_{i-1(k)} S'_{2j+1(k)} p_1/R} & \text{if } T(i) = \text{`}M'_{2j+1} \text{ for j=0,...,15.} \end{cases} \tag{16}$$

In particular, in the exponentiation phase we only have to distinguish between 17 alternatives $\Theta_{SW} = \{\text{`}S\text{'}, \text{`}M_1\text{'}, \text{`}M_3\text{'}, \ldots, \text{`}M_{31}\text{'}\}$ instead of 32. Similarly to (6) we define

$$\begin{aligned} &q_\theta(w'_{1(k)}, \ldots, w'_{31(k)}, w_{i(k)}) := \\ &Prob_\theta(W'_{2j+1(k)} = w'_{2j+1(k)} \text{ for } 0 \le j \le 15, W'_{2(k)} = w'_{2(k)}, W_{i(k)} = w_{i(k)}) \end{aligned} \tag{17}$$

for all $\theta \in \Theta_{SW}$. In analogy to (7) we have

$$q_\theta(w'_{1(k)}, \ldots, w'_{31(k)}, w_{i(k)}) = \int_0^1 \int_{a'_1}^{b'_1} \int_{a'_2}^{b'_2} \int_{a'_3}^{b'_3} \int_{a'_5}^{b'_5} \cdots \int_{a'_{31}}^{b'_{31}} \int_0^1 \int_{a_{\theta,i}}^{b_{\theta,i}} 1 \, ds_i ds_{i-1} ds'_{31} \cdots ds'_0 \tag{18}$$

defining an integral over the $(1 + 17 + 1 + 1) = 20$-dimensional unit cube. (Recall that (7) defines 34-dimensional integrals.) The integration boundaries are

$$(a'_1, b'_1) = \begin{cases} (0, s'_0(R^2 (\bmod p_1))/R) & \text{if } w'_{1(k)} = 1 \\ (s'_0(R^2 (\bmod p_1))/R, 1) & \text{if } w'_{1(k)} = 0. \end{cases} \tag{19}$$

$$(a'_2, b'_2) = \begin{cases} (0, {s'_1}^2 p_1/R) & \text{if } w'_{2(k)} = 1 \\ ({s'_1}^2 p_1/R, 1) & \text{if } w'_{2(k)} = 0. \end{cases}$$

$$i \in \{1, 2, \ldots, 15\}: \; (a'_{2i+1}, b'_{2i+1}) = \begin{cases} (0, s'_{2i-1} s'_2 p_1/R) & \text{if } w'_{2i+1(k)} = 1 \\ (s'_{2i-1} s'_2 p_1/R, 1) & \text{if } w'_{2i+1(k)} = 0. \end{cases}$$

$$(a_{\theta,i}, b_{\theta,i}) = \begin{cases} (0, s^2_{i-1} p_1/R) & \text{if } \theta = \text{`}S\text{'} \text{ and } w_{i(k)} = 1 \\ (s^2_{i-1} p_1/R, 1) & \text{if } \theta = \text{`}S\text{'} \text{ and } w_{i(k)} = 0 \\ (0, s_{i-1} s'_{2j+1} p_1/R) & \text{if } \theta = \text{`}M_{2j+1}\text{'} \text{ and } w_{i(k)} = 1 \\ (s_{i-1} s'_{2j+1} p_1/R, 1) & \text{if } \theta = \text{`}M_{2j+1}\text{'} \text{ and } w_{i(k)} = 0. \end{cases}$$

We may assume that in the sliding windows exponentiation algorithm both $d_k = 0$ and $d_k = 1$ occur with probability 0.5. In the second case all table values u_{2j+1} (determined by the following bits $(d_{k-1}, d_{k-2}, d_{k-3}, d_{k-4})$), are equally likely. The next window does not 'start' for any $k' \geq k-4$. Consequently, a multiplication with table value u_{2j+1} should occur about $\approx (512-4)/(6 \cdot 16) = 5.3$. We expect about $508 + 508/6 = 593$ Montgomery operations in the exponentiation phase of which 508 are squarings, and we use the a priori distribution $\eta_{SW}(\text{`}M_{2j+1}\text{'}) = (508/96)/(508 \cdot 7/6) = 1/112$ for each table value and $\eta_{SW}(\text{`}S\text{ '}) = 96/112$. Clearly, multiplications with table entries are isolated, and the lengths of two consecutive subsequences of squarings must sum up at to least window size 5. Error detection and correction are obviously less efficient than for fixed windows. On the positive side we only have to decide between 17 instead of 32 alternatives. A proposal for the loss function is $s_{SW}(\text{`}S\text{ '}, \text{`}M_j\text{'}) = 1.0$, $s_{SW}(\text{`}M_j\text{'}, \text{`}S\text{ '}) = 2.0$, $s_{SW}(\text{`}M_j\text{'}, \text{`}M_\tau\text{'}) = 2.0$ for $j \neq \tau$.

Theorem 3 remains true if we replace $(p_\theta, \eta, s(\cdot,\cdot))$ by $(q_\theta, \eta_{SW}, s_{SW}(\cdot,\cdot))$, and consider that $\boldsymbol{w}_{i(k)}, \boldsymbol{w}^* \in \{0,1\}^{18}$.

5 Possible Countermeasure Suggestions

There are several ways to achieve a "protected" RSA implementation against our MicroArchitectural Analysis presented in this paper. Instruction Cache Analysis and also Branch Prediction Analysis techniques exploit input-dependent execution flows of cryptosystems. Therefore, the implementations with fixed execution flows are intrinsically secure against these potential threats; hence they constitute a "risk-free" way of protection for these attacks.

[3] proposes a fixed execution flow implementation of RSA, which removes the extra reduction step completely. This method does not only provide an enhanced security, but also improves the performance of the RSA software compared to the current OpenSSL implementation. The idea of avoiding extra reduction steps is not novel to [3] and it was first given in [40, 41].

Walter proved in [40] that if the Montgomery parameters were chosen appropriately, we could avoid every extra reduction during an RSA exponentiation. He proved this property by analyzing the upper bounds on the values of intermediate Montgomery Multiplication results throughout the exponentiation. The parameters suggested by Walter necessitate to use more machine-words to hold the operands than necessary. In other words, his method increases the number of words in the operands and sets the Montgomery constant, which is usually denoted by R, accordingly and performs the multiplications in larger sizes. [15]

Later, Hachez & Quisquater improved the results of Walter [15]. They proved the same concept with tighter bounds and thus showed that the increase in the word size could be relatively smaller. Their method improves the speed of the Montgomery Multiplication and therefore the overall exponentiation compared to Walter's method.

Although we think it is a better practice to remove extra reduction step from the RSA computations, we need to mention that these attacks can also be avoided using alternative methods. For example, exponent blinding, which is a well-known countermeasure against side-channel analysis, is among these alternatives. This method adds a random multiple of the group-order, i.e., a random multiple of $\phi(n)$, where n is the public RSA modulus, to the secret exponent. This random multiple needs to be updated frequently, preferably for each RSA operation. The disadvantage of this method is the increased number of Montgomery operations due to the increase in the exponent size. We also need to mention that this method is covered by a patent, c.f. [18].

[15] the most significant word will initially be zero

6 Conclusions

We have identified a potential major weakness in the RSA implementations. We have developed an attack by leveraging MicroArchitectural analysis techniques and the early studies on extra-reduction based attacks. We have demonstrated this attack on OpenSSL and shown that the RSA implementation in the current version 0.9.8e can be completely broken. OpenSSL already implements several countermeasures against side-channel and MicroArchitectural analysis, some of which are operational by default while the others needs to be turned on by the user. However, even if we turn on all of these countermeasures that are available in v0.9.8e, the RSA implementation may still be completely broken with our attack strategy. We have conceptually and empirically proven this vulnerability in this paper. The OpenSSL development team already generated a patch, which has been included in the stable version and in particular will affect the upcoming version 0.9.8f. US CERT informed software vendors, assigned the vulnerability CVE name CVE-2007-3108 and CERT vulnerability number VU#724968, and issued a vulnerability note.

We have showed that the occurrences of extra reduction steps of Montgomery Multiplication during RSA operations could be extracted by malicious parties via the use of MicroArchitectural Analysis. In particular, we used Instruction Cache Analysis, which was introduced in [1], in our experimental setup and we could successfully construct the so-called er-vectors with reasonable error rates.

It was already proven in earlier studies ([39, 30, 29]) that the extraction of er-vectors with a small error rate let an adversary reveal the secret exponent. In this paper, we have lifted the theoretical works presented in [39, 30, 29] and put them into practice. Furthermore, we have adapted the theory of [30] to the specific implementation of OpenSSL v.0.9.8e and analyzed both fixed-window and sliding-window exponentiation methods used in this implementation.

We have also suggested several countermeasures that can be integrated into cryptographic libraries. We believe that the gravity of the vulnerability we identified here mandates revisions of the affected RSA implementations, which had already been done in OpenSSL. We have only focused on OpenSSL library in this paper due to its wide acceptance, however, other libraries and RSA implementations may also be vulnerable to our attack. Due to resource and time limitations, we could not conduct a comprehensive investigation on current security systems. We leave this task to other researchers.

References

1. O. Acıiçmez. Yet Another MicroArchitectural Attack: Exploiting I-cache. 14^{th} *ACM Conference on Computer and Communications Security (ACM CCS'07) — Computer Security Architecture Workshop*, 2007, to appear.
   Also available at: Cryptology ePrint Archive, Report 2007/164, May 2007.
2. O. Acıiçmez and J.-P. Seifert. Cheap Hardware Parallelism Implies Cheap Security. 4^{th} *Workshop on Fault Diagnosis and Tolerance in Cryptography — FDTC*, 2007, to appear.
3. O. Acıiçmez, S. Gueron, and J.-P. Seifert. New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures. 11^{th} *IMA International Conference on Cryptography and Coding*, 2007, to appear.
   Also available at: Cryptology ePrint Archive, Report 2007/039, February 2007.
4. O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. On The Power of Simple Branch Prediction Analysis. *2007 ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'07)*, R. Deng and P. Samarati, editors, pages 312-320, ACM Press, 2007.
   Also available at: Cryptology ePrint Archive, Report 2006/351, October 2006.

5. O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. Predicting Secret Keys via Branch Prediction. *Topics in Cryptology — CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007*, M. Abe, editor, pages 225-242, Springer-Verlag, LNCS 4377, 2007.
Also available at: Cryptology ePrint Archive, Report 2006/288, August 2006.
6. O. Acıiçmez, W. Schindler, and Ç. K. Koç. Cache Based Remote Timing Attack on the AES. *Topics in Cryptology — CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007*, M. Abe, editor, pages 271-286, Springer-Verlag, LNCS 4377, 2007.
7. O. Acıiçmez and Ç. K. Koç. Trace-Driven Cache Attacks on AES. Cryptology ePrint Archive, Report 2006/138, April 2006.
8. O. Acıiçmez, W. Schindler, Ç. K. Koç. Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations. *Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security*, C. Meadows and P. Syverson, editors, pages 139-146, ACM Press, 2005.
9. D. J. Bernstein. Cache-timing attacks on AES. Technical Report, 37 pages, April 2005. Available online at: `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`
10. G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. *International Symposium on Information Technology: Coding and Computing - ITCC 2005*, volume 1, pages 4-6, 2005.
11. J. Bonneau and I. Mironov. Cache-Collision Timing Attacks against AES. *Cryptographic Hardware and Embedded Systems — CHES 2006*, L. Goubin and M. Matsui, editors, pages 201-215, Springer-Verlag, LNCS 4249, 2006.
12. D. Brumley and D. Boneh. Remote Timing Attacks are Practical. *Proceedings of the* 12^{th} *Usenix Security Symposium*, pages 1-14, 2003.
13. J.-F. Dhem, F. Koeune, P.-A. Leroux, P.-A. Mestré, J.-J. Quisquater, J.-L. Willems. A Practical Implementation of the Timing Attack. *Smart Card – Research and Applications*, J.-J. Quisquater and B. Schneier, editors, pages 175-191, Springer-Verlag, LNCS 1820, 2000.
14. S. Gueron. Enhanced Montgomery Multiplication. *Cryptographic Hardware and Embedded Systems — CHES 2002*, B. S. Kaliski, Ç.K. Koç and C. Paar, editors, pages 46–56, Springer-Verlag, LNCS 2523, 2002.
15. G. Hachez and J.-J. Quisquater. Montgomery Exponentiation with no Final Subtractions: Improved Results. *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç.K. Koç and C. Paar, editors, pages 91–100, Springer-Verlag, LNCS 1965, 2000.
16. W. M. Hu. Lattice scheduling and covert channels. *Proceedings of IEEE Symposium on Security and Privacy*, IEEE Press, pages 52-61, 1992.
17. J. Kelsey, B. Schneier, D. Wagner, C. Hall. Side Channel Cryptanalysis of Product Ciphers. *Journal of Computer Security*, volume 8, pages 141-158, 2000.
18. P. C. Kocher and J. M Jaffe. Secure Modular Exponentiation with Leak Minimization for Smartcards and other Cryptosystems. United States Patent, Patent No.: US 6,298,442 B1, October 2001.
19. P. C. Kocher. Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems. *Advances in Cryptology - CRYPTO '96*, N. Koblitz, editor, pages 104-113, Springer-Verlag, LNCS 1109, 1996.
20. C. Lauradoux. Collision attacks on processors with cache and countermeasures. *Western European Workshop on Research in Cryptology — WEWoRC 2005*, C. Wolf, S. Lucks, and P.-W. Yau, editors, pages 76-85, 2005.
21. A. J. Menezes, P. C. van Oorschot, S. C. Vanstone. *Handbook of Applied Cryptography*. Boca Raton, CRC Press, New York, 1997.
22. M. Neve. Cache-based Vulnerabilities and SPAM Analysis. Ph.D. Thesis, Applied Science, UCL, July 2006
23. M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. *Selected Areas of Cryptography — SAC'06*, 2006.
24. M. Neve, J.-P. Seifert, Z. Wang. A refined look at Bernstein's AES side-channel analysis. *Proceedings of ACM Symposium on Information, Computer and Communications Security — ASIACCS'06*, Taipei, Taiwan, March 21-24, 2006.
25. D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. *Topics in Cryptology — CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006*, D. Pointcheval, editor, pages 1-20, Springer-Verlag, LNCS 3860, 2006.
26. D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002.
27. C. Percival. Cache missing for fun and profit. *BSDCan 2005*, Ottawa, 2005. Available online at: `http://www.daemonology.net/hyperthreading-considered-harmful/`.
28. W. Schindler. On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods. 8^{th} *International Workshop on Theory and Practice in Public Key Cryptography — PKC 2005*, S. Vaudenay, editor, pages 85-103, Springer-Verlag, LNCS 3386, 2005.

29. W. Schindler and C. D. Walter. More Detail for a Combined Timing and Power Attack against Implementations of RSA. *9th IMA International Conference on Cryptography and Coding*, K. G. Paterson, editor, pages 245-263, Springer-Verlag, LNCS 2898, 2003.
30. W. Schindler. A Combined Timing and Power Attack. *PKC 2002*, D. Naccache and P. Paillier, editors, LNCS 2274, pp. 263-279, 2002.
31. W. Schindler. Optimized Timing Attacks against Public Key Cryptosystems. *Statistics and Decisions*, volume 20, pages 191-210, 2002.
32. W. Schindler, F. Koeune, and J.-J. Quisquater. Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies. 8^{th} *International IMA Conference on Cryptography and Coding*, B. Honary, editor, pages 245-267, Springer-Verlag, LNCS 2260, 2001.
33. W. Schindler. A Timing Attack against RSA with the Chinese Remainder Theorem. *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç.K. Koç and C. Paar, editors, pages 110–125, Springer-Verlag, LNCS 1965, 2000.
34. O. Sibert, P. A. Porras, and R. Lindell. The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems. *IEEE Symposium on Security and Privacy*, pages 211-223, 1995.
35. K. Tiri, O. Acıiçmez, M. Neve, and F. Andersen. An Analytical Model for Time-Driven Cache Attacks. *Fast Software Encryption*, March 2007.
36. Y. Tsunoo, E. Tsujihara, M. Shigeri, H. Kubo, K. Minematsu. Improving cache attacks by considering cipher structure. *International Journal of Information Security*, volume 5, issue 3, pages 166-176, Springer-Verlag, 2006.
37. Y. Tsunoo, T.Saito, T. Suzaki, M. Shigeri, H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. *Cryptographic Hardware and Embedded Systems — CHES 2003*, C. D. Walter, Ç. K. Koç, and C. Paar, editors, pages 62-76, Springer-Verlag, LNCS 2779, 2003.
38. Y. Tsunoo, E. Tsujihara, K. Minematsu, H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. *ISITA 2002*, 2002.
39. C. D. Walter and S. Thompson. Distinguishing Exponent Digits by Observing Modular Subtractions. *Topics in Cryptology — CT-RSA 2001, The Cryptographers' Track at the RSA Conference 2001*, D. Naccache, editor, LNCS 2020, pp. 192-207, 2001.
40. C. D. Walter. Montgomery exponentiation needs no final subtractions. *IEE Electronics Letters*, volume 35, issue 21 pages 1831-1832, 1999.
41. C. D. Walter. Montgomery's Multiplication Technique: How to Make It Smaller and Faster. *Cryptographic Hardware and Embedded Systems — CHES 1999*, Ç.K. Koç and C. Paar, editors, pages 80–93, Springer-Verlag, LNCS 1717, 1999.
42. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide.
43. `http://www.irisa.fr/activity/new/007/branchpredictionattack004`.
44. `http://www.ntt.co.jp/news/news06e/0611/061108a.html`.
45. `http://cvs.openssl.org/chngview?cn=16275`
46. ftp://ftp.openssl.org/snapshot/
47. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3108`
48. `http://www.cert.org/`
49. US CERT vulnerability note. `http://www.kb.cert.org/vuls/id/724968`