Correlation Analysis against Protected SFM Implementations of RSA

Aurélie Bauer and Éliane Jaulmes

ANSSI
51, boulevard de La Tour-Maubourg
75700 PARIS-07 SP
{Aurelie.Bauer,Eliane.Jaulmes}@ssi.gouv.fr

Abstract. Since Kocher's first attacks in 1996, the field of side-channel analysis has widely developed, and new statistical tools have competed against new countermeasures to threaten cryptosystems. Among existing algorithms, RSA has always been a privileged target. It seems generally admitted that a combination of SPA protection such as regular exponentiation associated with blinding techniques such as randomization of the exponent and of the input message offers in practice sufficient protection against all known side-channel attacks. Indeed, known attacks either require building statistical information over several executions of the algorithm, which is countered by exponent randomization, or rely on partial SPA leakage, which implies an incorrect implementation of known countermeasures, or require specific internal knowledge of the implementation and hard-to-obtain experimental conditions, as for the recent horizontal correlation analysis of Clavier *et al.* [10]. In this paper, we show that it is possible to attack a state-of-the-art implementation of Straightforward Method (SFM) RSA. Our attack requires a small public exponent (no greater than $2^{16} + 1$) and a reasonable exponent blinding factor (no greater than 32 bits). It does not require additional internal knowledge of the implementation, neither does it have special experimental requirements. From a practical point of view, it thus compares with classical correlation analysis. We provide simulations of our attack demonstrating its efficiency, even in noisy scenarios. This shows that SFM implementations of RSA may be much more difficult to protect against side-channel attacks than CRT implementations.

Keywords: Side-Channel Attacks, Correlation Power Analysis, Collision Correlation Power Analysis, RSA scheme, Exponent Blinding, Message Blinding.

1 Introduction

Physical components included in embedded systems may leak information on data manipulated throughout cryptographic computations. *Side-channel analysis*, which was first introduced by Kocher *et al.* [17] in 1996, exploits such leakages and retrieves information on the secret parameters. This field covers nowadays a large range of statistical and cryptanalytic techniques such as timing attacks [17], simple power analysis (SPA) [17], differential power analysis (DPA) [18], correlation power analysis (CPA) [7], mutual information analysis (MIA) [3, 14] and many others.

G. Paul and S. Vaudenay (Eds.): INDOCRYPT 2013, LNCS 8250, pp. 98–115, 2013.

To counter these threats, research has been focusing on devising implementations that are resistant to side-channel analysis. While there exists generic countermeasures designed such that they can protect any given implementation (for instance clock jitter), most of them stay dedicated to specific algorithms or particular operations. One set of countermeasures still widely used up to now in the public key setting, is that introduced by Coron [11] in 1999.

Focusing on the well-known RSA scheme, it can be seen that the modular exponentiation phase, which consists in raising a given message to a secret exponent modulo a public integer, is a promising target for side-channel attackers. An unprotected RSA implementation can easily be threatened by a simple power analysis (SPA) where only one power curve is sufficient to recover the whole secret key. Thus, implementing RSA while avoiding classical side-channel attacks requires a set of well-chosen countermeasures. A first idea is to use a regular exponentiation, for example the "Square-and-Multiply Always" algorithm [11], the Montgomery ladder [16], the Joye ladder [15] or the atomic Square-and-Multiply exponentiation [8]. These techniques allow to proceed the same operations independently from the value of the key bits. Among all these solutions, the later one is often preferred due to its efficiency. Obviously these countermeasures offer protection against SPA, but are not sufficient to protect the scheme against more advanced attacks such as DPA or CPA [7, 18]. Those methods exploit leakage information from several executions of the algorithm and use statistical tools to extract the secret information. As a consequence, protecting the scheme against those kinds of attacks requires to execute the algorithm differently from one call to the next. Random values should thus be added to the computation, either in the secret exponent, which refers to the technique called "*Exponent Blinding*", or in the original message, namely using "*Message Blinding*".

In this work, we focus on SPA-resistant RSA implementations that use both exponent blinding and message blinding. As far as we are aware, this scenario represents the academic state of the art of secure RSA implementations. In fact, there are few side-channel attacks that threaten this scheme [9, 10, 12, 13, 19–23] and they apply only in particular settings. Indeed, the attack performed by Walter [22] on RSA with small public exponent, the one proposed by Fouque *et al.* [12] or the recent attack of Schindler and Itoh [19], all require partial SPA leakage exploitable on a single power curve. In Walter's Big-Mac attack [20] and Clavier *et al.* [9, 10] horizontal correlation analysis the authors exploit leakage information coming from each of the elementary operations involved in the Long Integer Multiplication. These attacks are very powerful but may be difficult to apply in practice. Indeed, they require to obtain additional internal information such as detailed knowledge of the implementation of the modular multiplication. Moreover, the adversary should also work with experimental tools of high quality as this attack requires a huge memory on the acquisition device and a precise timing to separately observe the leakage corresponding to each register operation.

In This Paper. We propose a side-channel attack against SPA-resistant SFM-RSA[1] scheme, implemented using both message blinding and exponent blinding. This attack can be seen as an alternative to Clavier *et al.* ROSETTA attack [9], where no specific information concerning the implementation of the modular multiplication is required. In fact, it exploits only general leakage information

[1] As mentioned before, SFM stands for Straightforward Method, *i.e.* a non-CRT implementation of RSA.

from each modular operation and does not assume high end metrology. Moreover, no additional SPA leakage is needed that could allow the attacker to distinguish multiplications from squaring operations on a unique power curve. Our attack is described on the Square-and-Multiply always algorithm and on the atomic Square-and-Multiply implementation of the exponentiation, but it also works for other choices of algorithms. It applies when the exponent masking technique suggested by Coron [11] is used and for any kind of message blinding. It works for small public exponents. The masking factor used for exponent randomization should either be less than 32-bit long or reduced in this range through fault injection.[2]

The paper is organised as follows. Section 2 recalls some basic notions related to the RSA scheme with exponent blinding and message blinding, and gives statistical definitions required for correlation power analysis. Section 3 presents existing side-channel attacks on RSA implementations using various countermeasures. In Section 4, we describe our attack against protected RSA implementations for a public exponent e equal to 3 and known inputs. In Section 5, we explain how to adapt this attack when the input is blinded by using the collision correlation technique [23]. Then Section 6 provides simulation results for various signal-to-noise ratios. Section 7 concludes the analysis.

2 Preliminaries

2.1 State-of-the-Art Implementation of RSA

Description. Let N be an n-bit RSA modulus, defined as the product of two large primes p and q. In the sequel, we focus on the *balanced* case meaning that the prime factors p and q are equal-sized. The public exponent e is chosen to be coprime to the Euler's totient function $\phi(N) = (p-1)(q-1)$. The corresponding private key d satisfies the well-known RSA equation $ed \equiv 1 \bmod \phi(N)$. In other words, there exists an integer $k \in \mathbb{Z}$ such that:

$$ed = 1 + k\phi(N) \ . \tag{1}$$

By definition, the private key satisfies $0 < d < \phi(N)$. Its binary representation is expressed as $d = (d[0] \dots d[n-1])$, where the least significant bit is referred as $d[0]$ and the most significant one as $d[n-1]$. Note that the variable k verifies $0 < k < e$ since otherwise it would imply $d > \phi(N)$, which is not possible [6].

Implementation. In this paper, we focus on *StraightForward Method* (SFM) implementations of RSA. It means that the decryption of a ciphertext C using the private key d is computed as $C^d \bmod N$. In order to resist "side-channel attacks", several steps must be taken into account to protect the sensitive operations where secrets bits are manipulated. In the particular case of RSA, this concerns the modular exponentiation $C^d \bmod N$, where secrets bits are processed sequentially and combined to the known value C.[3]

[2] Even if a 32-bit long masking factor is not recommended, this scenario still corresponds to some implementation designs of RSA on embedded devices.

[3] The attack is presented on a RSA decryption but could also apply on a RSA signature.

The first threat to address is the *Simple Power Analysis* (SPA). In order to resist this attack, the algorithm should behave similarly when the bits of the secret exponent are equal to 0 or 1. Regular algorithms have been designed to address this issue: the well-known *Square-and-Multiply Always* technique, provided in Fig. 1, Algorithm 1 and the *Atomic Square-and-Multiply* method [8], provided in Fig. 1, Algorithm 2, which is one of the most efficient technique. In Section 5, we focus on these two implementations. Other implementations choices are studied in Appendix A.

Algorithm 1. "Square-and-Multiply-Always" (*from left to right*)

$R_0 \leftarrow 1$
for $i = n - 1$ **to** 0 **do**
  $R_0 \leftarrow R_0^2$
  if $d[j] = 1$ **then**
    $R_0 \leftarrow R_0 \cdot C$
  else
    $R_1 \leftarrow R_0 \cdot C$
return R_0

Algorithm 2. "Atomic Square-and-Multiply" (*from left to right*)

$R_0 \leftarrow 1$; $R_1 \leftarrow C$
$k \leftarrow 0$, $i \leftarrow n - 1$
while $i \geq 0$ **do**
  $R_0 \leftarrow R_0 \cdot R_k$
  $k \leftarrow k \oplus d[i]$
  $i \leftarrow i - \neg k$
return R_0

Fig. 1. Two well-known regular SFM-RSA implementations

Additionally to that SPA-protection, the RSA scheme is also assumed to be implemented using exponent masking. This countermeasure prevents an attacker to gain information on the secret exponent d by studying several power consumption curves corresponding to computations of $C^d \bmod N$, for different values of C. In that case, the adversary could apply a *Differential Power Analysis* (DPA) or use an improved version called *Correlation Power Analysis* (CPA) (further details on this attack are provided in Section 2.2). To prevent such scenarios, the key should be masked before its use inside the modular exponentiation. A suitable idea, originally described by Coron [11], is to blind the exponent d using a random value λ. Thus a new secret key $d^{(i)} = d + \lambda^{(i)}\phi(N)$ is generated, each time the modular exponentiation has to be performed on $C^{(i)}$, with $\lambda^{(i)}$ a random l-bit integer. That way, the decryption (similarly the signature) process is performed as $\left(C^{(i)}\right)^{d^{(i)}} \bmod N$. In our attack, we require l to be no greater than 32. In real implementations, this could be the case for low end devices where generating random bits have a non negligible cost or it could be the result of a fault injection during the generation of $\lambda^{(i)}$ or during the computation of $d^{(i)}$.

Finally, in order to make the scheme fully secure, we also assume that it has been implemented using message blinding. Several techniques have been proposed in the literature, but we will not detail them here, since our attack applies independently from the chosen method.

2.2 Correlation Power Analysis

The attack proposed in this paper against secure SFM-RSA schemes, implemented as described in the previous section, makes use of the *Correlation Power Analysis* technique, introduced in [7]. For the sake of completeness, we remind the basic principle of this method, which can be seen as an extension of the Differential Power Analysis of Kocher *et al.* [18].

As all side-channel attacks, correlation power analysis works by first registering leakage information from the power consumption or electromagnetic emanation of the device during the computation. Such leakage can come from several instants of a single execution, in the case of horizontal power analysis [9, 10], or from a single instant of several executions, in the case of classical power analysis [7]. More generally, any combination of the two above is also possible, see for instance a unified description of CPA in both contexts in [4]. In the leakage traces, the attacker needs to identify the *points of interest*, namely the points in the traces where the leakage corresponds to the manipulation of the targeted sensitive information. For example, in vertical attacks, there is only one such point on each execution trace. In the case of horizontal analysis, several points of interest must be identified on the unique trace used to perform the attack. Let us denote as $(\ell_i)_{i\in I}$ the leakage values at the identified points of interest extracted from the power consumption curve(s).

Then, the adversary makes an hypothesis on a sub-part of the secret. Using his hypothesis and following the algorithm, the attacker is able to predict the operations $\mathcal{O}_i$ that took place during the execution(s) at each identified point of interest. He determines the contributions of these predicted operations to the global leakage. Therefore he chooses a leakage model $\mathcal{M}$ and computes the quantities $\mathfrak{m}_i = \mathcal{M}(\mathcal{O}_i)$, all related to the hypothesis on the secret parameter. The choice of a given leakage model should of course be based on the knowledge of the attacked device architecture. A common choice for $\mathcal{M}$ is to take the Hamming Weight of register size values manipulated during the operation $\mathcal{O}_i$. For instance, in the case of a multiplication on a 32-bit architecture, one could choose the Hamming Weight of the 32 least significant bits of the result.

Finally, the adversary validates or invalidates his hypothesis by computing the so-called *correlation coefficient* between the modelization values and the leakages. If L denotes the random variable corresponding to the observed leakages $(\ell_i)_{i\in I}$ and M the one corresponding to the modelized predictions $(\mathfrak{m}_i)_{i\in I}$, then this coefficient can be expressed as:

$$\rho = \rho(\mathsf{L}, \mathsf{M}) = \frac{cov(\mathsf{L}, \mathsf{M})}{\sigma_{\mathsf{L}}\sigma_{\mathsf{M}}},$$

where "*cov*" is the covariance function and "σ" the standard deviation. For the sample $(\mathfrak{m}_1, \mathfrak{m}_2 \ldots \mathfrak{m}_{|I|})$ of predicted leakages and $(\ell_1, \ell_2, \ldots \ell_{|I|})$ of registered power consumption values, an approximation $\tilde{\rho}(\mathsf{L}, \mathsf{M})$ of $\rho(\mathsf{L}, \mathsf{M})$ is given by the Pearson coefficient:

$$\tilde{\rho}(\mathsf{L},\mathsf{M}) = \frac{\sum_i (\ell_i - \ell)(\mathfrak{m}_i - \mathfrak{m})}{\sqrt{\sum_i (\ell_i - \ell)^2 \sum_i (\mathfrak{m}_i - \mathfrak{m})^2}},$$

where $\mathfrak{m} = \frac{1}{|I|}\sum_i \mathfrak{m}_i$ and $\ell = \frac{1}{|I|}\sum_i \ell_i$, with a sum taken over $i \in I$.

When the value of this correlation coefficient is high, it means that the random variables L and M are related, implying that the hypothesis on the sub-part of the secret was correct. In the other case, it means that the initial guess was wrong. In practice, the good hypothesis is often determined as that giving the highest correlation value among all possible hypotheses.

Remark: In practice, the identification of the points of interest is in fact done *a posteriori* by running the same attack on all points of the traces (or in a selected interval). The points of interest are then the points for which one of the key hypotheses produced a correlation peak.

3 Previous Attacks on RSA Implementations

In this section, we recall some existing side-channel attacks on RSA implementations. In particular, we study their applicability to the implementation we attack in this paper.

3.1 Statistical Analyses on Several Consumption Traces

Most existing side-channel attacks, such as Differential Power Analysis, Correlation Power Analysis or Mutual Information Analysis, require a high number – at least several – consumption traces to be efficient. Indeed, statistical analyses are performed on the collected curves, all related to the same sub-part of the secret, allowing to validate or invalidate hypotheses on the secret key. Such attacks target SPA-protected implementations, where the observation of a single power consumption trace does not provide enough information on the key.

Among such attacks, one can cite, for instance the work of Amiel *et al.* [2] describing a classical correlation power analysis on RSA or the work in [1] where the study of the Hamming weight distributions allows to distinguish multiplications from squaring operations. In [23], the collision correlation technique can be used to distinguish products with a common operand (key bit equal to 0) and products with independent operands (key bit equal to 1). Finally the well-known *doubling attack* of Fouque *et al.* [13] observes common intermediate values between the exponentiation of C and that of C^2.

In our context. Clearly these techniques are powerful on SPA-protected implementations. However they are successful only when sufficiently many traces, relied to the same sub-part of the secret, are available. As a consequence, the use of exponent blinding, which consists in changing the secret exponent at each new modular exponentiation performed by the device, make such attacks ineffective.

3.2 Attacks that Exploit an SPA Leak

Another approach is to assume that the SPA protection is not perfect and that some information can be extracted from a single execution of the secret computation. Indeed, in particular configurations, such as, for instance, when using sliding windows or implementing special types of multiplication algorithms such as MMM the "Montgomery Modular Multiplication", leakages might be obtained revealing partial information about the secret key. For a practical example, see Fouque *et al.* 's attack on RSA when e is small [12] and the improvement proposed by Walter in 2007 [22]. In 2011 [19], Schindler and Itoh generalized this technique by showing that any partial SPA information can be used to reconstruct the secret exponent, even when exponent blinding is used. They have no limitation on the size of the public key.

In our context. These techniques are efficient, since they are successful even against exponent blinding in the RSA exponentiation. However, they also rely on a strong hypothesis, which is the presence of SPA leaks. In this paper, we focus on implementations, that do not leak any exploitable SPA information. All the attacks mentioned in this section become ineffective in this context.

3.3 Horizontal Attacks

Up to now, the only attacks successful against protected implementations using exponent blinding, and which do not require any SPA leak, are horizontal correlation analyses. Indeed, these attacks use a unique power consumption curve. Their main idea can be explained as follows: when considering the leakage from the result of a modular multiplication or squaring, the attacker only gets a single information. However, modular operations in the case of RSA consist in multiplying 1024-bit or 2048-bit long numbers. In an embedded device, this is done by splitting the numbers into several smaller registers. The attacker could then consider the leakage coming from each register operation and thus gain much more information from a single modular operation. This idea was first studied by Walter in [20,21] in the so-called Big-Mac attack. It consists in cutting the power consumption trace – obtained from one or many execution of the algorithm on a single input – in many sub-traces, each of them containing information on a single internal operation such as, for instance, an elementary multiplication inside a LIM. This attack has then been extended in the work of Clavier *et al.* [9, 10] giving the Horizontal correlation analysis. A unified version of these two approaches can be found in [4].

In our context. This kind of attack is very efficient and provides strong results even for message blinding and exponent blinding implementations. However, it relies on two assumptions. First, the attacker needs a precise knowledge of the internal modular operations implementation (*e.g.* performed using LIM or MMM, parallelized or not, *etc.*). Secondly, the power consumption curve must be sufficiently precise to obtain several points of interest from one modular multiplication[4]. Thus correct time synchronisation and patterns identification will

[4] By comparison, a classical CPA on RSA like [2] will use one point of interest for each modular multiplication. In [9] the authors exploit $(\ell^2-\ell)/2$ points per multiplication, where $\ell = n/w$ for a modulus of size n on a w-bit architecture.

become a critical factor. As a matter of fact, in some experimental settings, horizontal analysis might not be applicable. In such configurations, our attack could be an interesting alternative, since it uses the same metrology as classical CPA on RSA schemes.

3.4 Attack Proposed in this Paper

Our attack applies against protected implementations of RSA using a regular exponentiation algorithm, with exponent masking of the form $d^{(i)} = d + \lambda^{(i)}\phi(N)$ and any kind of message blinding. We do not assume any partial SPA leakage nor require a precise knowledge of the internal implementation of modular operations. Our attack uses only one point of interest for each modular operation, as for vertical CPA attacks [2], and thus have less constraints on the metrology than horizontal attacks. For this reason, it represents an interesting alternative in noisy or black box scenarios.

4 The Attack on Protected RSA with Known Inputs

This section provides a description of our attack in a simplified setting, when the inputs[5] are known and for a public-key e equal to 3. The generalization for implementations using messages blinding will be discussed later, see Section 5. Other implementation choices are discussed in Appendix A. Additional considerations required for a practical implementation of the attack are provided in Appendix B.

4.1 Special Properties of the RSA Scheme

Let us first recall some well-known facts about the RSA cryptosystem, that will be useful in the following. For a more complete description of RSA properties, see [5].

Known Bits on $\phi(N)$. Since the factorization of the modulus N is a private information, the totient Euler's function is unknown to the attacker. However, half of its bits can easily be recovered. Indeed, knowing that $\phi(N) = (p-1)(q-1) = N - p - q + 1$ and assuming that p and q are balanced (*e.g.* $p, q \simeq \sqrt{N}$), the half[6] most significant bits of $\phi(N)$ is known and is equal to the half most ones of N.

Relation between d and $\phi(N)$. In the relation $ed = 1 + k\phi(N)$, see RSA Equation (1), the parameters d, k and $\phi(N)$ refer to unknown values. However when $e = 3$, the relation becomes:

$$3d = 1 + 2\phi(N). \tag{2}$$

The knowledge of the half most significant bits of $\phi(N)$ allows to deduce the half most significant ones of the secret key d. This result remains valid if more significant bits are known or guessed on $\phi(N)$. Conversely, guessing additional bits on d automaticaly implies recovering the corresponding ones on $\phi(N)$.

[5] Known inputs can either represent known messages for RSA signature or known ciphertexts for RSA decryption.

[6] Of course, there could be some carries issues, but this point will be discussed in Appendix B.

4.2 Description of the Attack

Let us now explain the details of our correlation analysis on protected RSA implementations. We recall that the secret exponent d is masked before each modular exponentiation as $d^{(i)} = d + \lambda^{(i)}\phi(N)$, see implementation details in Section 2.1, and that we first focus here on a simplified setting where each input data $C^{(i)}$ is known to the attacker.

General idea. Let $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(L)}$ be L power consumption curves, registered by the adversary during the computation of $\left(C^{(1)}\right)^{d^{(1)}} \bmod N, \ldots, \left(C^{(L)}\right)^{d^{(L)}} \bmod N$. In these expressions, the notation $d^{(i)}$ refers to the i-th masked value of d. The random masking factor $\lambda^{(i)}$ belongs to $[0, 2^{32}-1]$. The attack, which mainly consists in two steps, works as follows. First, the goal consists in guessing, for each i, the value of the masking factor $\lambda^{(i)}$ associated to the curve $\mathcal{T}^{(i)}$. To do so, several separate correlation analyses are made on each curve, using the first half of the exponentiation. In a second step, the adversary tries to guess the unknown bits of d by small increments using a classical correlation analysis, similar to the one described in [2]. For this step, the adversary uses the information gathered during the first part, namely the value of the masking factors $\lambda^{(i)}$. Thus a guess on d can easily be transformed into as many guesses on the masked values $d^{(i)}$.

Step 1: Learning Masking Factors with Correlation Attacks. First, we know that the half most significant bits of d can easily be recovered using Equation (2) of Section 4.1. From this information, the attacker wants to learn the value of the masking factors $\lambda^{(i)}$ that have been used to blind the exponent d as $d^{(i)} = d + \lambda^{(i)} \cdot \phi(N)$ for each exponentiation and thus for each curve $\mathcal{T}^{(i)}$. To do so, the attacker should perform the following operations:

1. Try all possible values for $\lambda^{(i)}$. Since each masking factor has been chosen as a 32-bit random value, this step requires 2^{32} operations.
2. For each possible $\lambda^{(i)}$, deduce the $n/2$ most significant bits of $d^{(i)}$. Indeed, observe that once $\lambda^{(i)}$ is known, we can use the knowledge on the most significant bits of d and $\phi(N)$ (see Section 4.1) to deduce the most significant bits of $d^{(i)}$ thanks to the relation $d^{(i)} = d+\lambda^{(i)}\phi(N)$. Note that we may know a few less bits than that, due to carries coming from the unknown parts of $\phi(N)$ and d that cannot be predicted.
3. From that point, since the adversary knows the input value $C^{(i)}$ together with the half most significant bits of $d^{(i)}$, it can predict the first half of the intermediate operations that have been performed during the modular exponentiation $(C^{(i)})^{d^{(i)}} \bmod N$. Depending on the exponentiation algorithm, the number η of intermediate operations the adversary is able to predict will vary from $n/2$ to n. In what follows, we denote as $\mathcal{O}_j^{(i)}$ such operations. Then, the attacker chooses a leakage model function $\mathcal{M}$ and computes some predicted values $\mathfrak{m}_j^{(i)} = \mathcal{M}(\mathcal{O}_j^{(i)})$ for $j \in \{1, \eta\}$. These values are then stored in a vector $\mathsf{M}^{(\mathrm{i})} = (\mathfrak{m}_j^{(i)})_{1 \le j \le \eta}$.
4. Perform a correlation analysis between the values $\mathsf{M}^{(\mathrm{i})} = (\mathfrak{m}_j^{(i)})_{1 \le j \le \eta}$, that have been predicted, and the leakages $\mathsf{L}^{(\mathrm{i})} = (\ell_j^{(i)})_{1 \le j \le \eta}$ coming from the

trace $\mathcal{T}^{(i)}$ at the identified points of interest. To do so, the adversary computes the Pearson correlation coefficient $\tilde{\rho}(\mathsf{L}^{(i)}, \mathsf{M}^{(i)})$, as described in Section 2.2. (See Figure 2 for an illustration.)

5. Eventually keep the mask $\lambda^{(i)}$ that gives the best correlation value.

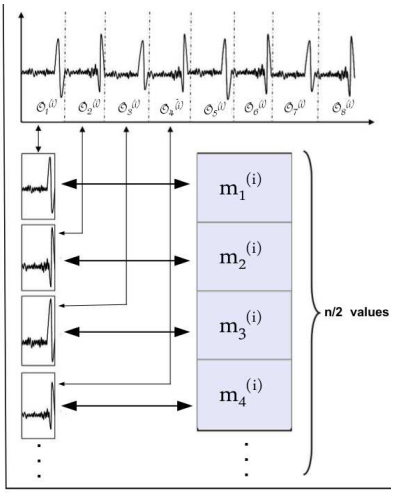

Fig. 2. Step 1: Correlation between predicted values $\mathfrak{m}_\ell^{(i)}$ and points on the curve $\mathcal{T}^{(i)}$

At the end of the process, the attacker is able to link each curve $\mathcal{T}^{(i)}$ to its corresponding masked factor $\lambda^{(i)}$.

Step 2: Recovering *d*. Assume now that we have guessed sufficiently many λ's corresponding to given power consumption curves, say L for instance. In this case, the final step consists in recovering the whole secret key d, namely guessing its $n/2$ least significant bits. To do so, the adversary performs a correlation power analysis as described in [2]. The idea is to make an hypothesis on a few bits on d, say w at a time, from the most significant bits to the least significant ones, then to (in)validate it using a CPA on the obtained consumption curves. So until we have guessed the whole secret key d or until the remaining bits can simply be recovered by a final exhaustive search, we repeat the following operations[7]:

1. Try all possible values for w bits of d. In practice, w may be taken quite small, as an example 8 bits could be a good choice, the idea being to make this exhaustive search step as fast as possible. From that hypothesis, we get a truncated value named $\overline{d}$, corresponding to the known most significant bits of d, concatenated with our guess and padded with 0's on the least significant positions (except for the last bit which we know is 1 since d is odd).

[7] The following process starts from the most significant unknown bits of d after execution of step 1 of the attack.

2. For each obtained value $\overline{d}$, compute $\overline{\phi(N)}$ as $\frac{3\overline{d}-1}{2}$, see Equation (2).
3. For each curve $\mathcal{T}^{(i)}$, deduce an approximation of $d^{(i)}$ by computing $\overline{d^{(i)}} = \overline{d} + \lambda^{(i)}\overline{\phi(N)}$. From these guessed bits on $d^{(i)}$, the adversary is able to determine the corresponding intermediate operations $\mathcal{O}_j^{(i)}$ during the modular exponentiation $\left(C^{(i)}\right)^{d^{(i)}} \bmod N$. One can notice that, depending on the size of $\lambda^{(i)}$, the place of the w bits in $\overline{d^{(i)}}$ will slightly vary From that point, the adversary uses a model function $\mathcal{M}$ and computes the predicted values $\mathfrak{m}_j^{(i)} = \mathcal{M}(\mathcal{O}_j^{(i)})$ corresponding to the intermediate operations.
4. Extract from the curves $(\mathcal{T}^{(i)})_i$ the leakages $\ell_j^{(i)}$ corresponding to the same operations. Use the correlation analysis described in Section 2.2 and compute the correlation coefficient $\tilde{\rho}\left((\ell_j^{(i)})_{i,j}, (\mathfrak{m}_j^{(i)})_{i,j}\right)$. (See Figure 3 to illustrate the process.)

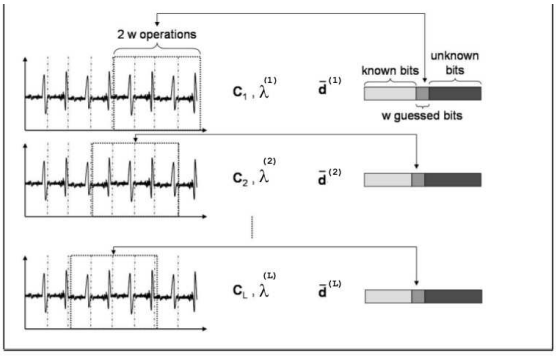

Fig. 3. Correlation Analysis between L curves $\mathcal{T}^{(i)}$ and the corresponding guessed bits on $\overline{d^{(i)}}$

5. Validate the w bits on d that give the best correlation coefficient.

5 The Attack on Protected RSA with Masked Inputs

In the previous attack, since we use a correlation analysis, we need to know the input in order to evaluate our predictions. When the input is unknown, the attack can be modified by performing a *Collision Correlation Analysis*, as described in [23]. Indeed, even if the input is unknown, we can still predict, from our guess on the exponent, the sequence of operations that should occur, according to the implemented exponentiation algorithm. In the following, we especially focus on two implementation cases.

5.1 The Atomic Square-and-Multiply Exponentiation Algorithm

This exponentiation technique has been proposed in [8] and is provided by Algorithm 2, see Figure 1. Here, from the guess on the exponent, we are able to deduce the sequence of operations. That is, we know when a squaring written as a multiplication is followed by a multiplication by $R_1 = C$ (case $d[j] = 1$) and when it is followed by another squaring (case $d[j] = 0$). In the first case, the result R_0 of the first squaring is then multiplied by $R_1 = C$, which value is independent from R_0. In the second case, this result is multiplied by R_0 itself. Thus, in this second case we expect a correlation between the result of the first squaring and the second operand of the next operation.

Following this exponentiation algorithm, a bit $d^{(i)}[j]$ of the exponent corresponds to 1 or 2 modular operations. Let us denote as $\mathcal{O}^{(i)}_{f(j)}$ the first squaring that always occurs. The next modular operation $\mathcal{O}^{(i)}_{f(j)+1}$ is then either a multiplication by C or the squaring associated to $d^{(i)}[j-1]$. When we know the most significant bits of $d^{(i)}$ down to j, we are able to determine the corresponding values of $f(j)$. In the following, we denote as $(\ell_{\mathtt{in}})^{(i)}_{f(j)}$ the leakage associated to the loading of the second operand of $\mathcal{O}^{(i)}_{f(j)}$ and $(\ell_{\mathtt{out}})^{(i)}_{f(j)}$ the one related to the output of $\mathcal{O}^{(i)}_{f(j)}$.

First Part of the Attack. From a guess on $\lambda^{(i)}$, we obtain the half most significant bits of $d^{(i)}$. Let L_1 and L_2 be two sets initially empty. For all known bits of $d^{(i)}$ such that $d^{(i)}[j] = 0$, we add $(\ell_{\mathtt{out}})^{(i)}_{f(j)}$ to L_1 and $(\ell_{\mathtt{in}})^{(i)}_{f(j)+1}$ to L_2. Thus the correlation coefficient $\tilde{\rho}(\mathsf{L}_1, \mathsf{L}_2)$ is maximal for the good hypothesis, which gives us the correct value for $\lambda^{(i)}$.

Second Part of the Attack. We proceed as above, except that one guess on the bits of the private exponent d corresponds to different values for the corresponding bits of the masked exponents $d^{(i)}$. We use the same sets L_1 and L_2 initially empty. For all i and all new known bits of $d^{(i)}$ such that $d^{(i)}[j] = 0$, we add $(\ell_{\mathtt{out}})^{(i)}_{f(j)}$ to L_1 and $(\ell_{\mathtt{in}})^{(i)}_{f(j)+1}$ to L_2. Again, the correlation coefficient $\tilde{\rho}(\mathsf{L}_1, \mathsf{L}_2)$ is maximal for the good hypothesis, which gives us the correct value for the targeted bits of d.

5.2 The Square-and-Multiply-Always Exponentiation Algorithm

This exponentiation is given by Algorithm 1, see Figure 1. Here, from a guess on the private exponent, we can predict when two consecutive multiplications will have a common operand. Indeed when $d[j] = 0$, the Square-and-Multiply-Always algorithm computes $R_1 \leftarrow R_0 \cdot C$ followed by $R_0 \leftarrow R_0 \cdot R_0$. The R_0 input value is the same for both operations. When $d[j] = 1$, the algorithm computes $R_0 \leftarrow R_0 \cdot C$ followed by $R_0 \leftarrow R_0 \cdot R_0$. The R_0 value is updated after the first multiplication and thus the two operations have no common operand.

Following this exponentiation algorithm, a bit $d^{(i)}[j]$ of the exponent always corresponds to 2 modular operations. We denote $\mathcal{O}^{(i)}_{f(j)}$ the first squaring. The

next modular operation $\mathcal{O}^{(i)}_{f(j)+1}$ is the multiplication by C and $\mathcal{O}^{(i)}_{f(j)+2}$ corresponds to the squaring associated with $d^{(i)}[j-1]$. Let $\ell^{(i)}_{f(j)}$ be the leakage associated to the loading of the first operand of $\mathcal{O}^{(i)}_{f(j)}$.

First Part of the Attack. From a guess on $\lambda^{(i)}$, we obtain the half most significant bits of $d^{(i)}$. Let L_1 and L_2 be two sets initially empty. For all known bits of $d^{(i)}$ such that $d^{(i)}[j] = 0$, we add $\ell^{(i)}_{f(j)+1}$ to L_1 and $\ell^{(i)}_{f(j)+2}$ to L_2. The correlation coefficient $\tilde{\rho}(\mathsf{L}_1, \mathsf{L}_2)$ is maximal for the good hypothesis, which gives us the correct value for $\lambda^{(i)}$.

Second Part of the Attack. We proceed as above, except that one guess on the bits of the private exponent d corresponds to different values for the corresponding bits of the masked exponents $d^{(i)}$. We use the same sets L_1 and L_2 initially empty. For all i and all new known bits of $d^{(i)}$ such that $d^{(i)}[j] = 0$, we add $\ell^{(i)}_{f(j)+1}$ to L_1 and $\ell^{(i)}_{f(j)+2}$ to L_2. That way, the correlation coefficient $\tilde{\rho}(\mathsf{L}_1, \mathsf{L}_2)$ is maximal for the good hypothesis, which gives us the correct value for the targeted bits of d.

6 Simulation Results

In order to check the validity of our attack, we performed many simulations on RSA implementations, using a public key e equal to 3, a modulus of size 1024 and 2048 bits and exponent blinding with 8-bit and 16-bit random masked factors[8]. For each scenarios, we performed a hundred simulations, each of them following the same process: simulating and storing the leakage of the implementation at each loop iteration, then performing our attack using either correlation power analyses or collision correlation attacks depending whether the inputs are known or not. More precisely, in order to simulate the leakage of the operation $Z = X.Y \bmod N$, we produce the three consecutive leakages $(\mathrm{HW}_{32}(X), \mathrm{HW}_{32}(Y), \mathrm{HW}_{32}(Z))$, where $\mathrm{HW}_{32}(X)$ denotes the Hamming weight of the 32 least significant bits of X. Then, we add to this value a random noise of zero mean and standard deviation σ. Finally, each experiment has been performed for different number of traces. Obviously, when this number increases, the Pearson coefficient better estimates the value of the correlation. Thus, the attack works better. However, it also takes longer to compute since the random blinding factor must first be guessed for each trace.

Figure 4 presents the success rate observed over the hundred experiments for the two steps of the known input attack. The first number represents the success rate for guessing the correct value of $\lambda^{(i)}$. The second number represents the success rate for guessing the correct value of d. Note that success consists in recovering the whole secret exponent. Thus we observe that when the noise becomes too high the success rate drops quickly since we need to accumulate 128 correct guesses ($w = 8$) in step 2 to obtain the whole secret key, for a 1024-bit modulus.

[8] The exponent blinding factor was chosen quite small in order to be able to launch hundreds of attacks for comparison. However, simulations indicate that when this factor increases it mainly impacts the computation time.

On the same figure, we show two variations on the main experiments. In the first one, we used moduli of size 2048 bits instead of 1024 bits and kept $\lambda^{(i)}$ of size 8 bits with 500 traces. We observe in this case that the success rate for the whole attack drops quicker since a success requires more correct guesses. On the other hand, even in high noise the guesses on $\lambda^{(i)}$ stay quite good. Further experiments suggest that, at a given noise level, decreasing the success rate of the first step has a negligible impact on the success of the second step. In the second scenario, we used $\lambda^{(i)}$ of size 16 bits and kept moduli of size 1024 bits with 500 traces. We observe in this case that the success rate for the first step of the attack drops when the noise is high. However, it has no impact when the noise stays reasonable and the overall success rate is unchanged. This observation was confirmed on our simulations for moduli of size 256 to 2048 bits.

On Figure 5, we show the attack on unknown inputs with the Square-and-Multiply-Always algorithm and with the Atomic Square-and-Multiply algorithm. In these last two cases, the success rate of the attack is more impacted by noise. Indeed here noise impacts the two sets we try to correlate instead of one. This observation shows that our attack is similar to second order attacks classically applied on symmetric algorithm. We also observe in the "no noise" scenario that some wrong guesses have the same correlation than the correct one. These wrong branches could be detected by a more elaborating backtracking algorithm since the correlation coefficient then drops for the next guesses. This would also improve the results in the noisy case.

			Deviation σ											
N	λ	Traces	0		1		5		10		15		20	
1024	8	500	100	100	100	100	100	96	100	71	99.8	0	93.9	0
1024	8	1000	100	100	100	100	100	100	100	100	99.8	61	93.9	0
1024	8	2000	100	100	100	100	100	100	100	100	99.8	96	93.9	55
2048	8	500	100	100	100	100	100	95	100	29	100	0	99.9	0
1024	16	500	100	100	100	100	100	96	100	71	94	0	58	0

Fig. 4. Percentage of success in the known input attack

	Square-and-Multiply Always								***Atomic Square-and-Multiply***							
Traces	$\sigma = 0$		$\sigma = 1$		$\sigma = 3$		$\sigma = 5$		$\sigma = 0$		$\sigma = 1$		$\sigma = 3$		$\sigma = 5$	
1000	100	97	100	97	99.8	74	53.9	0	100	99	100	98	99.7	83	21.9	0
2000	100	99	100	98	99.8	76	51.4	4	100	97	100	96	98.8	84	30.6	0

Fig. 5. Percentage of success for regular algorithms, with input blinding ($N = 1024$ bits and $\lambda = 8$ bits)

7 Conclusion

We have described an attack against an SFM implementation of RSA protected against SPA by a regular exponentiation algorithm and against DPA by exponent and message blinding. Our attack does not require any assumption concerning the details of the modular multiplication. It works in two steps, combining the results of several correlation analyses. The attack only applies when the exponent blinding factor allows an exhaustive search. This could limit the applicability of our attack. However, in a scenario where the blinding factor may be reduced in a small interval through fault injection, our attack may find a renewed interest.

One can observe that RSA-CRT implementations do not suffer from this attack since the private exponents d_p and d_q corresponding to the prime factors p and q are completely unknown from the attacker. Indeed, even if e is small, we cannot deduce useful information about the first bits of d_p and d_q.

References

1. Amiel, F., Feix, B., Tunstall, M., Whelan, C., Marnane, W.P.: Distinguishing Multiplications from Squaring Operations. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 346–360. Springer, Heidelberg (2009)
2. Amiel, F., Feix, B., Villegas, K.: Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 110–125. Springer, Heidelberg (2007)
3. Bastina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F.-X., Veyrat-Charvillon, N.: Mutual Information Analysis: A Comprehensive Study. Journal of Cryptology 24(2), 269–291 (2011)
4. Bauer, A., Jaulmes, E., Prouff, E., Wild, J.: Horizontal and vertical side-channel attacks against secure RSA implementations. In: Dawson, E. (ed.) RSA 2013. LNCS, vol. 7779, pp. 1–17. Springer, Heidelberg (2013)
5. Boneh, D.: Twenty years of attacks on the RSA cryptosystem. Notices of the American Mathematical Society, AMS (1999)
6. Boneh, D., Durfee, G., Frankel, Y.: An Attack on RSA Given a Small Fraction of the Private Key Bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998)
7. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
8. Chevallier-Mames, B., Ciet, M., Joye, M.: Lowcost Solutions for Preventing Simple Side-Channel Cryptanalysis: Side-Channel Atomicity. IEEE Transactions on Computers 53(6), 760–768 (2004)
9. Clavier, C., Feix, B., Gagnerot, G., Giraud, C., Roussellet, M., Verneuil, V.: ROSETTA for Single Trace Analysis – Recovery of Secret Exponent by Triangular Trace Analysis. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 140–155. Springer, Heidelberg (2012)
10. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal Correlation Analysis on Exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 46–61. Springer, Heidelberg (2010)
11. Coron, J.-S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
12. Fouque, P.-A., Kunz-Jacques, S., Martinet, G., Muller, F., Valette, F.: Power Attack on Small RSA Public Exponent. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 339–353. Springer, Heidelberg (2006)

13. Fouque, P.-A., Valette, F.: The Doubling Attack – Why Upwards is Better than Downwards. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 269–280. Springer, Heidelberg (2003)
14. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
15. Joye, M.: Highly regular m-ary powering ladders. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 350–363. Springer, Heidelberg (2009)
16. Joye, M., Yen, S.-M.: The Montgomery Powering Ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
17. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
18. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
19. Schindler, W., Itoh, K.: Exponent Blinding Does Not Always Lift (Partial) SPA resistance to Higher-Level Security. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 73–90. Springer, Heidelberg (2011)
20. Walter, C.D.: Sliding Windows Succumbs to Big Mac Attack. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 286–299. Springer, Heidelberg (2001)
21. Walter, C.D.: Longer Keys May Facilitate Side Channel Attacks. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 42–57. Springer, Heidelberg (2004)
22. Walter, C.D.: Longer Randomely Blinded RSA Keys May Be Weaker Than Shorter Ones. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 303–316. Springer, Heidelberg (2008)
23. Witteman, M.F., van Woudenberg, J.G.J., Menarini, F.: Defeating RSA Multiply-Always and Message Blinding Countermeasures. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 77–88. Springer, Heidelberg (2011)

A Our Attack in Other Implementation Cases

A.1 Right-to-Left Implementation

The description can be adapted to the right-to-left case by going backward from the result of the computation. Indeed if we look at a right-to-left "Square-and-Multiply Always" implementation (see Algorithm 3), we see that knowing the last t bits of d, we can deduce the last t squares and multiplies.

Algorithm 3. "Square-and-Multiply-Always" (*from right to left*)

$R_0 \leftarrow 1$; $R_1 \leftarrow C$
for $j = 0$ **to** $n - 1$ **do**
  if $d[j] = 1$ **then**
    $R_0 \leftarrow R_0 \cdot R_1$
  else
    $t \leftarrow R_0 \cdot R_1$
  $R_1 \leftarrow R_1^2$
return R_0

A.2 Montgomery

Our attack in the known input case can be adapted to the improved version of the "Square-and-Multiply Always" algorithm based on Montgomery [16] (see Algorithm 4).

Algorithm 4. "Montgomery ladder"

```
R0 ← 1 ; R1 ← C
for j = n − 1 to 0 do
    if d[j] = 0 then
        R1 ← R1 · R0
        R0 ← R0 · R0
    else
        R0 ← R0 · R1
        R1 ← R1 · R1
return R0
```

In the masked input case, we can distinguish a transition from '0' to '1' or '1' to '0' from no transition. Indeed when a transition occurs, the output of the square is used as the first operand of the next multiplication. When there is no transition the output of the squaring is used as the second operand.

A.3 Larger Public Key

When e is greater than 3, the coefficient k is no longer known for certain (see Equation (1)). Thus it must be guessed together with the $\lambda^{(i)}$ in order to apply the first part of the attack. Since k verifies $0 < k < e$, this means that the exhaustive search factor is multiplied by the value of the public key. In most RSA implementations, the RSA public exponent does not exceed $2^{16} + 1$. This means that a 2^{16} factor should be added to the exhaustive search complexity.

B Attack Implementation: Carries and Wrong Guesses

There is an issue when performing "Step 2" of the attack: how to deal with the carries and their potential wrong guesses implication. Indeed, in this part of the algorithm, we are trying to guess the bits of d by small increments, setting the lower unknown bits to 0. When the intermediate value $\overline{d}$ is used to compute $\overline{\phi(N)}$ and $\overline{d^{(i)}}$, we may be confronted to the fact that the bits we are guessing are wrong because of unknown carries coming up to this point in the real values. More precisely, we may be wrong on two points:

1. **When We Compute $\overline{d^{(i)}}$ from $\overline{d}$ and $\overline{\phi(N)}$.** In this case, an error means that the prediction given by the corresponding curve will be incorrect: this adds some noise in the correlation coefficient computation. If we are not wrong too often, this will not change the overall decision about $\overline{d}$.
2. **When We Compute $\overline{\phi(N)}$ from $\overline{d}$.** In this configuration, an error means that we will be wrong for all curves. Indeed, if $\overline{\phi(N)}$ is incorrect, then all $\overline{d^{(i)}}$ will also be incorrect. But let us have a look at the kind of error it implies. Assume that we predict some value a for the w bits of d we are

considering at this point. Thus, we can write $\overline{d}$ as $[d]_{(j+1)w} + a2^{jw} + 1$, where $[d]_{(j+1)w} = d - (d \bmod 2^{(j+1)w})$. In that case, we compute $\overline{\phi(N)}$ as:

$$\begin{aligned}\overline{\phi(N)} &= \frac{3\overline{d} - 1}{2} = \frac{3[d]_{(j+1)w} + a2^{jw} + a2^{jw+1} + 2}{2} \\ &= \frac{3[d]_{(j+1)w}}{2} + a2^{jw} + a2^{jw-1} + 1.\end{aligned}$$

Let b be the value of the corresponding bits of $\overline{\phi(N)}$, that is:

$$b = \left(\frac{3[d]_{(j+1)w}}{2^{jw+1}} + a + \lfloor a/2 \rfloor \right) \bmod 2^w$$

If we assume that b is incorrect, this means that a carry should have appeared in the computation of b and the correct value is $b + 1$. In this case, the predictions of $d^{(i)}$ for the guess a will all be wrong. On the other hand, the predictions for $a + 1$ will give:

$$\overline{\phi(N)} = \frac{3[d]_{(j+1)w}}{2} + (a+1)2^{jw} + a2^{jw-1} + 2^{jw-1} + 1.$$

As a consequence, if b' denotes the value of the corresponding bits of $\overline{\phi(N)}$, we obtain:

$$b' = \left(\frac{3[d]_{(j+1)w}}{2^{jw+1}} + a + 1 + \lfloor a/2 \rfloor \right) \bmod 2^w = b + 1.$$

This time, we reach the correct value for $\overline{\phi(N)}$. Since $\overline{d^{(i)}} = \overline{d} + \lambda^{(i)}\overline{\phi(N)}$, with high probability, we also have the correct value for the bits of $\overline{d^{(i)}}$ we are considering. The guess $a+1$ will have a better correlation coefficient than the guess a. At the next step, however, the correlation will drop suddenly. When this happens, we know that the previous guess $a + 1$ was incorrect and we just have to back up one step, subtract 1 to the guess, and go on. The simulations we have done confirm that this approach is correct.