The Carry Leakage on the Randomized Exponent Countermeasure

Pierre-Alain Fouque[1], Denis Réal[2,3], Frédéric Valette[2], and Mhamed Drissi[3]

[1] École normale supérieure/CNRS/INRIA, 75 Paris, France
Pierre-Alain.Fouque@ens.fr
[2] CELAR, 35 Bruz, France
{Denis.Real,Frederic.Valette}@dga.defense.gouv.fr
[3] INSA-IETR, 20 avenue des coesmes, 35043 Rennes, France
{Denis.Real,Mhamed.Drissi}@insa-rennes.fr

Abstract. In this paper, we describe a new attack against a classical differential power analysis resistant countermeasure in public key implementations. This countermeasure has been suggested by Coron since 1999 and is known as the *exponent randomization*.

Here, we show that even though the binary exponentiation, or the scalar product on elliptic curves implementation, does not leak information on the secret key, the computation of the randomized secret exponent, or scalar, can leak useful information for an attacker. Such part of the algorithm can be not well-protected since its goal is to avoid attack during the exponentiation. Consequently, our attack can be mounted against any kind of exponentiation, even very resistant as soon as the exponent randomization countermeasure is used. We target an ℓ-bit adder which adds ℓ-bit words of the secret exponent and of a random value. We show that if the carry leaks during the addition, then we can almost learn the high order bits of each word of the secret exponent. Finally, such information can be then used to recover the entire secret key of RSA or ECC based cryptosystems.

1 Introduction

Side channel attacks are very powerful attacks and today most embedded applications that require high level of security use countermeasures against such kind of attacks. Two of the most carefully studied algorithms are the square-and-multiply algorithm and its analog on Elliptic Curve, the double-and-add algorithm, since its wide usage. There exists a classical countermeasure to avoid simple power analysis (SPA) attack, that always performs the multiply or the add operation so that all the operations of the implementation are not key dependent. This countermeasure is very efficient in practice, so that most implementations use it. However, such implementations can be attacked by using differential power analysis (DPA [13]) techniques such as in [14] and a popular countermeasure consists in randomizing the secret exponent or secret scalar by a multiple of the order of the elements $\varphi(N)$ in the case of RSA modulus or of the order of the base point in the case of Elliptic Curve. Such countermeasure has been proposed by Coron in [7] since 1999. With this countermeasure, the secret exponent will never be the same and DPA attacks that recover the secret bit by bit cannot be mounted.

Related Work. This well-known countermeasure has been first attacked by Fouque and Valette in [11] using the Doubling Attack. However, in such attack the adversary is assumed to be able to send many times the same message and that no randomization of the message is performed before the exponentiation. Here, our attack avoids these two drawbacks since the attack does not need the knowledge of the message.

In [10], Fouque *et al.* show that if Coron's countermeasure is used with some windowing exponentiation algorithms and a small public key e, then a simple SPA followed by a very clever attack can recover the secret key d and $\varphi(N)$ in the same time. In [10], the implementation is not protected against SPA attacks since the classical SPA attack does not work on the windowing algorithms. In this work, the authors have to solve a problem similar of that which we try to solve here, namely, recovering the secret d in RSA, knowing some *non-consecutive* bits of d. Indeed, side channel technique allows Fouque *et al.* to learn some key bits of many randomized exponents of the form $d_j = d + \lambda_j \varphi(N)$, for many λ_j in a small set, the set of 20-bit or 32-bit integers in typical implementations.

Recovering secret RSA key knowing some bits of d is an old problem starting from the pionerring work of Boneh, Durfee and Frankel in [2] since 1998. However, the techniques used in Boneh *et al.*'s paper are based on Coppersmith's lattice algorithm [5,6] that works well when the bits are *consecutive*. Later, other attacks such as [9,1] have been proposed on RSA, but no one except [10] targets the case when bits are non consecutive.

In the Elliptic Curve case, the problem of recovering secret scalar when non-consecutive bits are known has also been studied. The Baby Step Giant Step algorithm can always be used, however reducing the memory requirement is not always possible as with Pollard algorithm or the lambda method, *a.k.a.* the kangoroo algorithm in [19,15]. However, Stinson describes an algorithm due to Coppersmith in [18] that can be used to reduce the memory requirement. A similar algorithm has been devised by Coron *et al.* in [8] for RSA modulus. However, the missing bits must not be too numerous since the method is based on the birthday paradox and memory and time requirements are almost in the square root or fourth root of the number of missing bits.

Our Results. In this paper, we show that the exponent randomization countermeasure can be attacked very efficiently and the whole secret key can be recovered. The main novelty of the attack is to target the computation of the randomization itself $d_j = d + \lambda_j \cdot \varphi(N)$ in case of an RSA modulus and not the exponentiation $x \mapsto x^{d_j} \bmod N$. In the addition of a random value with a fix and secret one, the targeted operation is the sum of the secret scalar with a random number, a random multiple of the order of the base point P. Seifert in [17] and Brier *et al.* in [3] have also studied attacks on other part of the algorithm, on some public information for example. Here, our attack is less invasive since we do not change parameters and we only record some electromagnetic radiations. Finally, this attack is very efficient since it works against very secure or even "provably-secure" exponentiation that uses the exponent randomization since the side channel leakage comes from the countermeasure and not from the exponentiation algorithm.

We show that when the secret exponent, or scalar, and the randomization are cut into ℓ-bit word, then the carries of the adder can leak and such information can be used to guess the high order bits of each ℓ-bit word of the secret with a good precision. Then to recover the whole secret key, either the number of missing bits is small enough so that a classical baby step giant step method could be used or other techniques are required to find the other bits. In the case of RSA keys or large ECC keys, the idea consists in recovering the randomized value λ_j using the known bits of the order. Once the λ_j's are known, the addition or the exponentiation are unprotected against classical DPA attacks such as address-bit DPA [12] or Correlation Power Analysis (CPA) attack [4].

Organization of the Paper. The principle of the attack is presented in section 2. Then, in section 3, we theoretically explain how the knowledge of the number of carries allows us to guess the high order bits of each word of the secret key. In section 4, we show that the internal carries of the full addition involved in the masking process can be observed by SCA. Finally, in section 5 we describe the attacks against classical implementations of RSA and ECC to retrieve the whole secret key.

2 The Attack Principle

The idea of the attack is to target the countermeasure operation and not the exponentiation or scalar product operation. The former operation is usually not well protected since it is used to protect the latter one. So, in the sequel, we assume that the exponentiation is protected against SPA by using the square-and-multiply always algorithm and against DPA attack by using randomization of the message even with unknown blinding and the randomization of the exponent.

2.1 The Secret Randomization Countermeasure

It is well-known that randomizing d with $d_j = d + \lambda_j \varphi(N)$ for RSA and $d_j = d + \lambda_j \#\mathcal{E}$ for ECC leads to the same results. Furthermore, if λ_j is different at each execution of the algorithm, classical DPA attacks which retrieve the secret bit by bit become ineffective. Such a countermeasure is known as the exponent randomization. Fig. 1 describes this technique for RSA and ECC.

```
– Inputs: a message $M$ for RSA (resp. a point $P$ of a curve $\mathcal{E}$ for
  ECC), a word size in bits $\mu$, an exponent $d$, a modulus $N$ (resp.
  $\#\mathcal{E}$, the cardinal of $\mathcal{E}$).
– Output: $M^d$ mod $N$ for RSA (resp. $d \cdot P$ for ECC)

1. Take a $\mu$-bit random integer $\lambda_j$
2. Compute $d_j = d + \lambda_j \varphi(N)$ (resp. $d_j = d + \lambda_j \#\mathcal{E}$)
3. Return SCA protected exponentiation $M^{d_j}$ mod $N$ (resp. $d_j \cdot P$)
```

Fig. 1. The Private Exponent Randomization for RSA (resp. ECC)

2.2 The Sketch of the Attack

If someone adds random integers R_i to a fixed integer S, the probability over the different values of R_i to observe a carry flag only depends on S. Indeed, on 8-bit integers, random addition with the fixed value `0xFF` is more likely to raise a carry flag than with the fixed integer `0x01`.

Integers are often too large to be added through a digital circuit. The operands are usually broken into ℓ-bit words and the full addition function is splitted into ℓ-bit additions. An ℓ-bit addition is the sum of two ℓ-bit integers. A carry flag is raised for a buffer overflow, *i.e.* when the ℓ-bit sum is larger or equal to 2^ℓ.

These carry flags raised during the full addition can be observed by side channel analysis. An attacker who observes a device for many secret randomizations can use the carry flag as a source of information to retrieve the secret RSA or ECC exponent. Our attack uses two stages: the side channel analysis to obtain information on the secret and the cryptographic attack which uses the information to recover the entire secret key.

2.3 The Exponent Randomization Ripple Carry Addition

This subsection describes the notations used in the rest of the paper. The attacker performs m exponent randomizations and j denotes the indice of the randomization from 0 to $m-1$.

The addition function used for the exponent randomization is assumed to be designed as a k-word ripple carry addition. The two operands of the addition are broken in k ℓ-bit words with $\ell = 8$, 16 or 32. The full addition is then performed word by word using a ℓ-bit adder which takes as input two ℓ-bit operands and a carry-in and outputs the sum and the carry-out. The ripple strategy consists in chaining the carry-out and the carry-in together. Let i be the word indice from 0 to $k-1$. The private exponent and the mask are denoted by d and $A^{(j)} = \lambda_j \varphi(N)$ for RSA and $A^{(j)} = \lambda_j \#\mathcal{E}$ for ECC. The carry flag raised during the i^{th} ℓ-bit addition for the j^{th} randomization is $c_i^{(j)}$ and C_i is the sum of the carry flags raised during the m exponent randomizations, $C_i = \sum_{j=0}^{m-1} c_i^{(j)}$. The principle of the ripple adder for the j^{th} exponent randomization is described in Fig. 2 and the notations are the following:

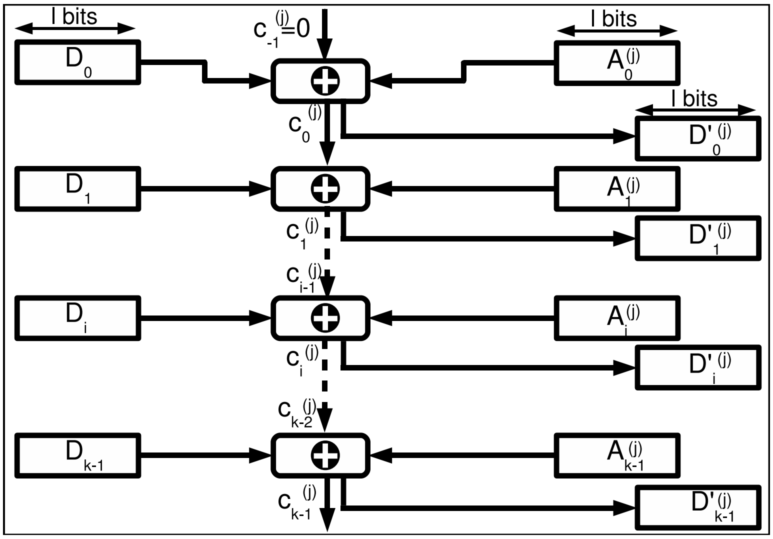

Fig. 2. j^{th} Exponent randomization

- ℓ: The atomic adder size
- k: The number of words.
- m: The number of exponent randomizations observed.
- d: The private exponent $d = \sum_{i=0}^{k-1} D_i 2^{\ell \cdot i}$.
- d': The randomized private exponent $d' = \sum_{i=0}^{k-1} D'_i 2^{\ell \cdot i}$.
- $A^{(j)}$: The j^{th} mask $A^{(j)} = \sum_{i=0}^{k-1} a_i^{(j)} 2^{\ell \cdot i}$.
- $c_i^{(j)}$: The carry involved in the addition of the i^{th} ℓ-bit word:
  - $c_{-1}^{(j)} = 0$ (no initial carry.)
  - $c_i^{(j)} = 1$ if $D_i + a_i^{(j)} + c_{i-1}^{(j)} \geq 2^\ell$ with $0 \leq i < k$ and $c_i^{(j)} = 0$ otherwise.
- C_i: The number of carries in the addition of the i^{th} ℓ-bit word: $C_i = \sum_{j=0}^{m-1} c_i^{(j)}$

3 The Exponent Randomization Attack

The exponent randomization consists in summing the private exponent with a mask. To do so, both exponent and mask are divided into k ℓ-bit words. In this section, we assumed that the attacker can observe or deduce the number C_i of carries involved on the i^{th} ℓ-bit addition in the m exponent randomizations. In the next section, we show that such information can be observed by using side channel attack.

In the following, we assume that the randomization $\lambda_j \varphi(N)$ (or $\lambda_j \#\mathcal{E}$) are uniformly distributed values. Even though such an assumption is not correct, we can assume that it is locally correct. For each word of $\lambda_j \varphi(N)$, we can assume this property since the number of curves needed is less than the 2^{32} values of the λ_j's and the multiplication has the property to quickly spread the random values of the λ_js into all words of $\lambda_j\varphi(N)$ except maybe the first and last words.

Probability of Guessing a Word given the Number of Carries. The attacker has to guess the i^{th} word of the secret exponent knowing the number of carries involved during the m randomizations. Theorem 1 gives us the probability of a correct guess of the probability distribution of guessing the i^{th} word of the secret knowing the number C_i of carry flags involved in its making is given by Eq. 1.

Theorem 1. *The probability distribution of guessing the i^{th} word of the secret knowing C_i the number of carries flags involved in the m randomizations is*

$$\Pr(D_i = n|C_i = q) = \frac{(n/2^\ell)^q(1-n/2^\ell)^{m-q}}{\sum_{\alpha=0}^{2^\ell-1}(\alpha/2^\ell)^q(1-\alpha/2^\ell)^{m-q}} \tag{1}$$

Proof. First, we compute the probability distribution of the first ℓ-bit word D_0 of the secret exponent given the number of carries C_0 involved with a ℓ-bit adder implementation during m randomizations, *i.e.* we prove the above formula for $i = 0$. Then, we use an induction on i to prove the theorem for all values i.

During a *single* randomization, the probability $\Pr(C_0 = 1|D_0 = n)$ of observing one carry for the first word is $n/2^\ell$. Indeed, let a given mask A^j, a given secret d, and their first ℓ-bit words are respectively a_0^j and D_0. These words can take 2^ℓ different values with the same probability. The value D_0 is fixed while a_0^j is purely random, thus: $\Pr(C_0 = 1|D_0 = n) = \Pr(n + a_0^j > 2^\ell - 1) = \Pr(a_0^j > 2^\ell - n - 1)$. Then a carry is observed when a_0^j takes one of the n values larger than or equal to $2^\ell - n$ and smaller than or equal to $2^\ell - 1$. Therefore:

$$\Pr(C_0 = 1|D_0 = n) = n/2^\ell \tag{2}$$

Now, we compute the probability distribution $\Pr(D_0 = n \cap C_0 = q)$ using the definition of the conditional probability: $\Pr(D_0 = n \cap C_0 = q) = \Pr(C_0 = q|D_0 = n) \cdot \Pr(D_0 = n)$. Since there exist $\binom{m}{q}$ possible cases where q carries are observed during m randomizations. Therefore:

$$\Pr(D_0 = n \cap C_0 = q) = \left[\binom{m}{q} \Pr(C_0 = 1|D_0 = n)^q(1 - \Pr(C_0 = 1|D_0 = n))^{m-q}\right] \cdot (1/2^\ell) \tag{3}$$

Then, we need to compute the probability distribution of the event $C_0 = q$. Since, the secret D_0 can take 2^ℓ different values, we can thus compute the probability by summing on all value of D_0 as follows: $\Pr(C_0 = q) = \sum_{\alpha=0}^{2^\ell-1} \Pr(C_0 = q \cap D_0 = \alpha)$ and using (3), we get:

$$\Pr(C_0 = q) = \frac{1}{2^\ell} \cdot \binom{m}{q} \sum_{\alpha=0}^{2^\ell-1} (\alpha/2^\ell)^q(1-\alpha/2^\ell)^{m-q} \tag{4}$$

Finally, we compute the probability distribution $\Pr(D_0 = n|C_0 = q)$ by using (3) and (4):

$$\Pr(D_0 = n|C_0 = q) = \frac{(n/2^\ell)^q(1 - n/2^\ell)^{m-q}}{\sum_{\alpha=0}^{2^\ell-1}(\alpha/2^\ell)^q(1 - \alpha/2^\ell)^{m-q}} \tag{5}$$

Now, we prove theorem (1) for $i > 0$. For the j^{th} randomization, the $(i+1)^{th}$ addition carry c_{i+1}^j does not only depend on the value of $D_{i+1} + a_{i+1}^j$ but also on the i^{th} addition carry c_i^j. More precisely, the $(i+1)^{th}$ addition carry does not depend on the i^{th} addition carry except if $D_{i+1}^j + a_{i+1}^j = 2^\ell - 1$. Then, as D_{i+1} is fixed, c_{i+1}^j depends on c_i^j one time out of 2^ℓ. If we omit this fact, then equation (5) can be generalized to:

$$\Pr(D_{i+1} = n|C_{i+1} = q) = \frac{(n/2^\ell)^q(1 - n/2^\ell)^{m-q}}{\sum_{\alpha=0}^{2^\ell-1}(\alpha/2^\ell)^q(1 - \alpha/2^\ell)^{m-q}} \tag{6}$$

$\square$

Even if this function is discrete, the probability distribution of the random variable $D_i/2^\ell$ knowing C_i can be approximated as the Beta distribution $\beta(q + 1, m - q + 1)$. This approximation is detailed in Appendix B and Fig. 3 represents the evolution of the probability distribution according to the number m of experiments.

The probability distribution shape tends to zero except on a lobe which is maximal for $\lfloor q \cdot (2^\ell + 1)/m \rfloor$ or $\lceil q \cdot (2^\ell + 1)/m \rceil$. The attacker can then take a decision. The most probable of these two words is defined as the secret estimate $\hat{D}_i$. The attacker's probability to take the right decision, *i.e.* the probability of $\hat{D}_i = n$, increases with m. The worst case, *i.e.* when the probability of $\hat{D}_i = n$ is the lowest, is for $m = 2q$ leading to $\hat{D}_i = 2^{\ell-1}$.

Furthermore, instead of choosing one single word, the attacker can select the most probable words that could match to the secret. He owns then not anymore one estimate but a set of estimates. He can then accumulate the different probabilities, meaning he tries to guess part of the secret instead of the whole secret itself. This strategy can be very efficient. Indeed, just a few words achieve a non negligible probability, the other ones having a probability close to 0. This strategy consists then in using cumulative properties instead of the density properties.

This gain can be illustrated through an example: an attacker observes $10,000$ exponent randomizations and observes $1,250$ carries on a 8-bit adder. What can he deduce? The probability that `0x20` is the secret word is 0.47. But the probability that `0x1F` or `0x20` is the secret word increases to 0.7. Cumulating 4 words (`0x1E`,`0x1F`,`0x20` `0x21`) leads to a probability of success higher than 0.99. In the worst case, $m = 2q$, the variance of $\beta(m/2+1, m/2+1)$ is $\sigma^2 = 1/4(m+3)$ [16]. Then, the number of estimates to accumulate for reaching a success probability of at least 0.99 is proportional with $2^\ell/\sqrt{m}$ by using Chebyshev bound. We verified experimentally this result: for $10,000$ exponent randomizations, 4 estimates are needed for getting a probability of 0.99 when $q = 5,000$ and $\ell = 8$.

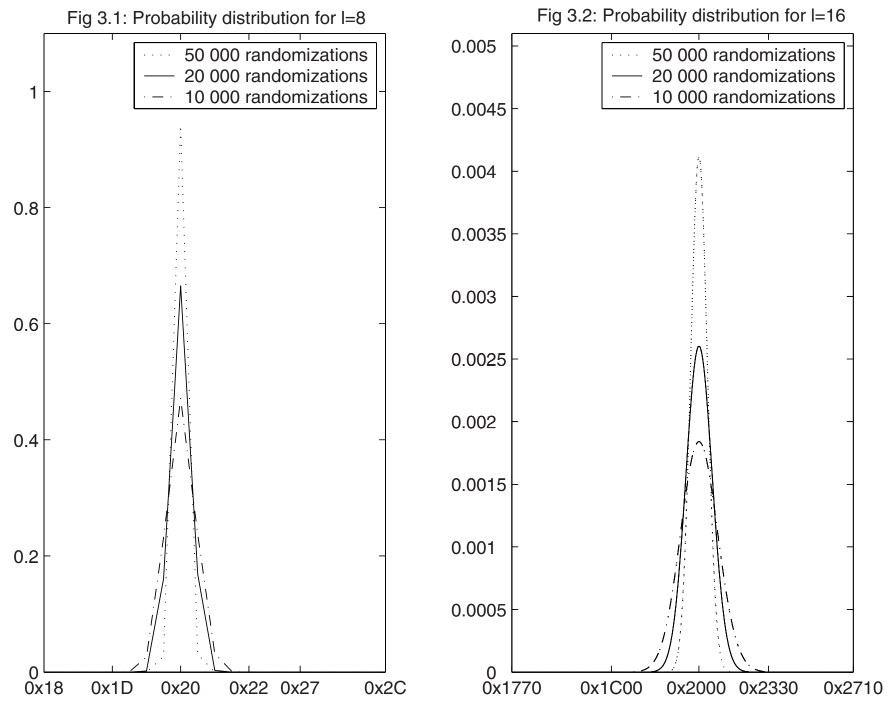

Fig. 3. Probability Distribution of $D|C$ According to the Number of Experiments with $\ell = 8$ and $\ell = 16$

4 The Exponent Randomization SCA

In this section, we show that the value C_i can be learned by the adversary. The target of our side channel attack is the carry-out of the atomic adder. We have tested its feasibility by simulating a 160-bit masking on the ProASIC 3/E starter kit from Actel which is a FPGA development kit. We have designed a full ripple addition function with a 32-bit adder. In appendix A, we give some information concerning addition design.

4.1 The Location and Profiling Stages

The SCA feasibility is demonstrated with EMA techniques, studying the electromagnetic side channel. Radiation is measured in the near field zone using a small loop probe sensitive to the horizontal magnetic field. The used test bench is represented on Fig. 4. The two operands are randomly chosen to localise in space the adder on the chip and time slot where the addition is performed during the implementation. The carry flag can then be localised more sharply by using a DPA attack.

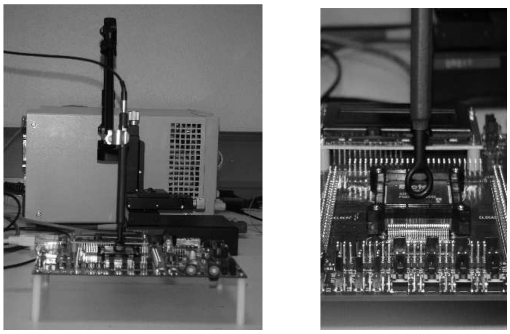

Fig. 4. EM Test Bench

In order to build the j^{th} 160-bit mask used for the j^{th} exponent randomization, the random generator of the FPGA is used. The 32-bit addition is performed in two stages: the loading stage (the new operands of the adder are loaded) and the addition stage (the add instruction is executed).

4.2 The Attacking Stage

The 160-bit secret d is split in 5 32-bit words. Then, it is randomized m times and the average EMA trace Γ_m is computed. From the profiling stage, we can locate on Γ_m the carry contribution for each word D_i. This contribution is noise free. Indeed, the noise is assumed to be zero-mean. It is close to zero with m large enough. For each word D_i, the corresponding carry contribution is expected to be proportional with the carry probability. The number of carry flags raised during the m masking operations can be then deduced according to the previous section.

The previous statements are illustrated on a concrete case. We performed 1000 masking operations. The least significant bits (LSB) of each word D_i are chosen randomly, the probability to have a carry depends then only on the most significant bits (MSB) of D_i. Thus, we build d such as:

– D_0=`0x00FC3478`: the expected carry probability is around 0
– D_1=`0x40FE56AC`: the expected carry probability is around 63/256
– D_2=`0x804890BD`: the expected carry probability is around 127/256
– D_3=`0xC0C2A4C8`: the expected carry probability is around 200/256
– D_4=`0xFF98ACBF`: the expected carry probability is around 255/256

Fig. 5 shows Γ_{1000} where the contribution of the masking of D_0 is subtracted. To do so, an extra loading is made with D_0 parameters but the addition is not performed: this yields the characteristic of the unrelated instructions.

For a given word of d, the expected carry probability and the carry radiation are proportional as it is shown in Tab. 1. The relative amplitude difference between two consecutive maskings is 5μV.

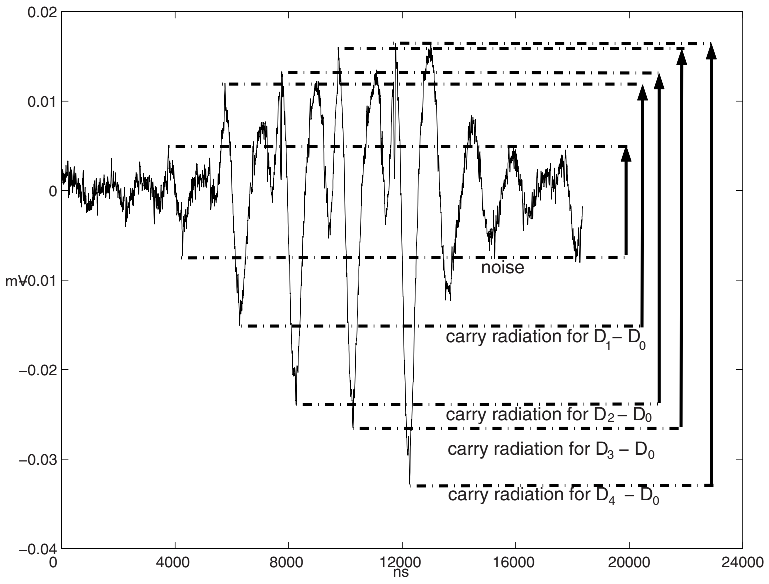

Fig. 5. Average Trace Γ_{1000} where the contribution of the masking of D_0 is subtracted

Table 1. Absolute and relative contributions of the carry on Γ_{1000}

Masking	**Absolute Amplitude**	**Relative Amplitude**
$D_0 - D_0$	0.012mV	0mV
$D_1 - D_0$	0.031mV	0.019mV
$D_2 - D_0$	0.036mV	0.024mV
$D_3 - D_0$	0.043mV	0.031mV
$D_4 - D_0$	0.049mV	0.037mV

4.3 Results and Conclusion

For a ripple carry addition, the attacker can have access to the information C_i even in the presence of noise. If the addition function has been designed another way, we claim that the attacker has access to the same amount of information. Indeed, the computational cost of the carry-out of a ℓ-bit adder depends on the way it is built. The more the carry-out is complex to obtain, the more its computation costs power and the more it leaks with the side channel. The ripple carry adder is the adder whose carry-out is the lowest side channel available. Indeed, it needs 2 OR and 3 AND while the carry-out of a 4-bit look-ahead adder costs 10 OR and 4 AND as it is stated in Appendix A.

Furthermore, independently of the addition design, it takes into account word adder whose operands are a word of the private exponent and the corresponding word of the mask: the unique difference is the carry-in treatment. However this difference is negligible: as D_i is fixed, the carry-out of the word adder depends one time out 2^ℓ on the carry-in. Then, irrespective of the addition function used, we assume that the multiple bits adder takes into account a carry-in equals to zero.

5 Recovering the Entire Secret Keys

In this section, we present two ways to use the information extracted by the side channel measurements. The first technique consists in finding enough bits with the carry leakage to be able to realize a kind of exhaustive search of the secret by using the baby step-giant step method. The second technique consists in combining two side channel attacks to retrieve the entire secret key. Both attacks are complementary as their efficiency depends on the size of the key and on the size of the registers. Some examples are discussed in the last subsection.

5.1 A Kind of Exhaustive Search

We assume that the attacker performs m measurements of the exponent randomization of the secret d, stored in k ℓ-bit words. In the previous analysis, he is able to reduce the number of possible values in each word of d. For each word, a fraction $2^\ell/\sqrt{m}$ of the corresponding key word is possible (the probability the secret is in this set is then higher than 0.99) so the number of possible values for d will be $(2^\ell/\sqrt{m})^k$. If the attacker can reduce the set of possible values for d to a subset of size lower than 2^{128}, we consider that he can find the whole secret exponent d with classical baby-step giant-step methods for a computational cost lower than 2^{64}. We can note that this attack will be more efficient on shorter keys and smaller register such as elliptic curve implementations on 8-bit or 16-bit registers. So the computational cost of the attack is $(2^\ell/\sqrt{m})^{k/2}$.

5.2 The Combined Attack

The other solution uses the carry leakage information to find partial information on d which will be used to find for each masking operation $d_j = d + \lambda_j \times \varphi(N)$ (or $d_j = d + \lambda_j \times \#\mathcal{E}$) the random value λ_j. Once sufficiently many λ_j's are known, a classical DPA attack can be mounted either on the masking operation or directly on the exponentiation to retrieve the missing bits of d. In fact, the knowledge of λ_j will unprotect the exponentiation against classical attacks such as an address bit DPA which does not need to know the value of the message. We will see in the following that the success of this attack depends more on the size of key and on the size of λ than on the number of possible measurements.

Sketch of the Attack. The attack can be divided into three steps:

- with m measurements, the attacker approximates the value D_i of each register with a precision of $\sqrt{m}$,
- with this approximation, he can try all possible values for λ and compute for the known bits of the order $Ord = \varphi(N)$ or $\#\mathcal{E}$ all the possible values for $\lambda \times Ord$. In case of RSA, only half of the bits are known as the most significant bits

of $\varphi(N)$ are equal to those of N, but in the discrete logarithm case, the order of the group is known so all the bits of Ord are known. With the approximation of d, the attacker can compute for each value λ the value of the carry of the i^{th} register. The carry at register i will be perfectly defined excepted when it comes from the unknown bits of D_i which can happen with probability $1/\sqrt{m}$. If the number of carries information is sufficient, each curve can be associated with a single value of λ. This will happen when the number of registers where the carry is known, is larger than the size of λ.

– with the m measurements and their associated value of λ, an address-bit DPA or CPA attack can be mounted to retrieve the value of d. If the attacker targets the masking operation or the address during the exponentiation, he will have to guess recursively the unknown bits of d and eventually, the unknown bits of $\varphi(N)$ in case of RSA.

The number of measurements m is defined by the number of curves needed to complete an address bit DPA attack on the masking operation or on the exponentiation without the exponent masking protection. Usually, $10,000$ curves are sufficient to mount such an attack but this depends on the noise level. With such a number of curves, the approximation of the value D_i of each register has a precision of 2^6. If λ is a 32-bit long random value, the attacker needs the secret key to be stored on more than 32 registers in case of discrete logarithm problem or more than 64 in case of RSA as only the most significant bits of $\varphi(N)$ are known.

5.3 Results on RSA and ECC

In this section, we will present some applications of the previous attacks. The complexity in terms of measurement and computation is evaluated according to the considered attack with a λ of 32 bits.

Table 2. Attack complexity on some examples. "ES" stands for exhaustive search, "CA" for combined attacks, and "NP" for Not Practical.

Cryptographic implementation	**attack**	**Measurements**	**computational cost**
RSA 1024 **on a** 8-**bit adder cell**	ES	2^{16}	1
RSA 1024 **on a** 8-**bit adder cell**	CA	$10,000$	2^{32}
RSA 1024 **on a** 16-**bit adder cell**	CA	$10,000$	2^{32}
RSA 1024 **on a** 32-**bit adder cell**	NP		
RSA 2048 **on a** 32-**bit adder cell**	CA	$10,000$	2^{32}
ECC 160 **on a** 16-**bit adder cell**	ES	2^{16}	$\approx (2^{16}/\sqrt{2^{16}})^{10/2} = 2^{40}$
ECC 160 **on a** 32-**bit adder cell**	ES	2^{20}	$\approx (2^{32}/\sqrt{2^{20}})^{5/2} = 2^{55}$

6 Conclusion

In this article, we show that the addition performed during an exponent randomization is a risky operation. Indeed, the internal carries due to local buffer overflows during this operation are a side channel available and secret dependent so that the whole private exponent can be recovered for some public key implementations. The SCA feasibility has been demonstrated using near field techniques for gaining the electromagnetic radiations of a FPGA summing two 32-bit words: the presence of a carry has been detected.

This new attack is interesting since it targets the countermeasure and not the algorithm that it has to protect. Usually this operation is not well-protected and so side channel leakage can be observed. Finally, the attack can be performed on any exponentiation algorithm except the final phase which is needed only for RSA based cryptosystem. The carry leakage is in general sufficient to attack ECC based cryptosystem since the secret keys are smaller.

References

1. Blömer, J., May, A.: New Partial Key Exposure Attacks on RSA. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 27–43. Springer, Heidelberg (2003)
2. 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)
3. Brier, E., Chevallier-Mames, B., Ciet, M., Clavier, C.: Why One Should Also Secure RSA Public Key Elements. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 324–338. Springer, Heidelberg (2006)
4. 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)
5. Coppersmith, D.: Finding a Small Root of a Bivariate Integer Equation; Factoring with High bits Known. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 155–165. Springer, Heidelberg (1996)
6. Coppersmith, D.: Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. J. Cryptology 10(4), 233–260 (1997)
7. 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)
8. Coron, J.-S., Lefranc, D., Poupard, G.: A New Baby-Step Giant-Step Algorithm and Some Applications to Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 47–60. Springer, Heidelberg (2005)
9. Ernst, M., Jochemsz, E., May, A., de Weger, B.: Partial Key Exposure Attacks on RSA up to Full Size Exponents. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 371–386. Springer, Heidelberg (2005)
10. 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)
11. 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)
12. Itoh, K., Izu, T., Takenaka, M.: Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 129–143. Springer, Heidelberg (2003)

13. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
14. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power Analysis Attacks of Modular Exponentiation in Smartcards. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 144–157. Springer, Heidelberg (1999)
15. Pollard, J.M.: Kangaroos, Monopoly and Discrete Logarithms. J. Cryptology 13(4), 437–447 (2000)
16. Rade, L., Westergren, B.: Mathematics Handbook for Science and Engineering, Sudentlitteratur, Lund (1998)
17. Seifert, J.-P.: On authenticated computing and RSA-based authentication. In: Atluri, V., Meadows, C., Juels, A. (eds.) ACM Conference on Computer and Communications Security, pp. 122–127. ACM, New York (2005)
18. Stinson, D.R.: Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. Math. Comput. 71(237), 379–391 (2002)
19. van Oorschot, P.C., Wiener, M.J.: Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 229–236. Springer, Heidelberg (1996)

A The Addition Strategy

The addition problems start when adding 2 single bits and finishes when able to add 2 words of arbitrary length.

A.1 The Single Bit Adder

The single bit adder is the most elementary logical circuit of a device. Two kinds of single bit adder exist: the half adder and the full adder. The Half Single Bit Adder (HA) has two inputs labelled a and b and two outputs: the sum s and the carry-out c_{out}. The value s is the 1-bit sum of a and b while c_{out} is the carry flag raised in case of overflow. Sum and carry-out are computed as follow : $s = a \oplus b$ and $c_{out} = a.b$ The Full Single Bit Adder (FA) is a half adder that takes into account the carry-in bit c_{in}. The different relations become $s = a \oplus b \oplus c_{in}$ and $c_{out} = (a.b) + (b.c_{in}) + (c_{in}.b)$.

A.2 The Word Adder

An ℓ-bit adder is an element used for the addition of two words of ℓ bits each, typically, $\ell = 8$, 16 or 32. Let $A = \sum_{i=0}^{\ell-1} a_i 2^i$ and $B = \sum_{i=0}^{\ell-1} b_i 2^i$ be the two ℓ-bit operands, C_{in} be the carry-in, $S = \sum_{i=0}^{\ell-1} s_i 2^i$ be the sum and C_{out} be the carry-out. The value C_{out} is the object of the side channel analysis. There is not just one way of building a word adder. Indeed, different strategies exist for dealing with internal carries. Then, the way C_{out} is computed depends on the word adder design.

The Ripple Carry Adder. This is the most straightforward implementation of a final stage ℓ-bit adder. Carry-ins and carry-outs are chained together requiring

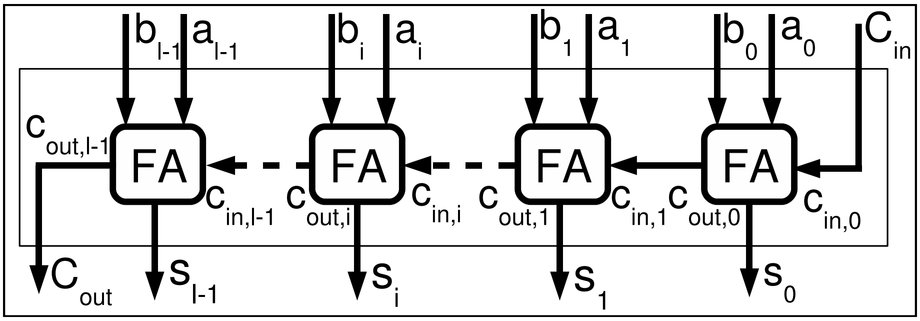

Fig. 6. The Ripple Carry Adder

ℓ FAs. Fig. 6 describes this design. Let $c_{out,i}$ and $c_{in,i}$ be respectively the carry-out and the carry-in of the i^{th} FA.

Chaining carries together leads to the following relations: $c_{in,0} \leftarrow C_{in}$
for $0 \leq i < \ell \quad c_{in,i+1} \leftarrow c_{out,i}$
$C_{out} \leftarrow c_{out,\ell-1}$.
Then C_{out} is connected to the carry-out of the last FA.

The Carry Look-Ahead Adder. This adder aims to generate all carry-ins in parallel for not waiting until the carry propagates from the stage of the FA it has been generated. The carry propagation signal $\{P_i\}$ and the carry generation signal $\{G_i\}$ are introduced using the previous notations: $P_i = a_i \oplus b_i$, $G_i = a_i \cdot b_i$ and then $c_{in,i+1} = G_i + c_{in,i} \cdot P_i$. These expressions can be computed in parallel for all the carries. As, an example, for a 4-bit adder, we have:

$c_{in,0} = C_{in}$

$c_{in,1} = G_0 + c_{in,0} \cdot P_0 = G_0 + C_{in} \cdot P_0$

$c_{in,2} = G_1 + c_{in,1} \cdot P_1 = G_1 + G_0 \cdot P_1 + C_{in} \cdot P_0 \cdot P_1$

$c_{in,3} = G_1 + c_{in,2} \cdot P_2 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_{in} \cdot P_0 \cdot P_1 \cdot P_2$

$c_{in,4} = G_3 + c_{in,3} \cdot P_3 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_{in} \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3$

$C_{out} = c_{in,4}$

B The Beta Distribution

The last probability distribution of the secret estimate knowing the carry function given by formula (6) can be approximated by a discrete beta distribution. Indeed: the beta distribution is defined as

$$\beta(q+1, m-q+1) = \int_0^1 t^q (1-t)^{m-q} dt$$

and using Riemann sums, we obtain:

$$\beta(q+1, m-q+1) = \lim_{n\to\infty} \frac{1}{n} \sum_{\alpha=1}^{n} \frac{\alpha}{n}^{q} \left(1 - \frac{\alpha}{n}\right)^{m-q} .$$

Finally, if we assume that 2^ℓ is large enough, then

$$2^\ell \cdot \beta(q+1, m-q+1) \approx \sum_{\alpha=0}^{2^\ell - 1} \frac{\alpha^q}{2^\ell} \left(1 - \frac{\alpha}{2^\ell}\right)^{m-q} .$$