The Doubling Attack – *Why Upwards Is Better than Downwards*

Pierre-Alain Fouque and Frederic Valette

DCSSI Crypto Lab
51, Boulevard de Latour-Maubourg
75700 Paris 07 SP
France
Pierre-Alain.Fouque@ens.fr
Frederic.Valette@m4x.org

Abstract. The recent developments of side channel attacks have lead implementers to use more and more sophisticated countermeasures in critical operations such as modular exponentiation, or scalar multiplication in the elliptic curve setting. In this paper, we propose a new attack against a classical implementation of these operations that only requires two queries to the device. The complexity of this so-called "doubling attack" is much smaller than previously known ones. Furthermore, this approach defeats two of the three countermeasures proposed by Coron at CHES '99.

Keywords. SPA-based analysis, modular exponentiation, scalar multiplication, DPA countermeasures, multiple exponent single data attack.

1 Introduction

Modular exponentiation or scalar multiplication are the main parts of the most popular public key cryptosystems such as RSA [15] or DSA [13]. This very sensitive operation can be efficiently implemented in smart cards products. However, data manipulated during this computation should often be kept secret, so the implementation of such algorithms must be protected against side channel attacks. For example, during the generation of an RSA signature by a device, the secret exponent is used to transform a message related data into a digital signature via modular exponentiation.

Timings and power attacks, initially presented by Kocher [9,10] are now well studied and various countermeasures have been proposed. Those attacks represent a real threat when we consider operations that both involve secret data and require a long computation time. The consequence is that naive implementation of RSA based or discrete log based cryptosystems usually leak information about the secret key.

In this paper we present a new side channel attack, that we called "doubling attack", which allows to recover the secret scalar used in the binary scalar

C.D. Walter et al. (Eds.): CHES 2003, LNCS 2779, pp. 269–280, 2003.

multiplication or the secret exponent used in the binary exponentiation algorithm. It is worth to notice that contrary to previous attacks which work for the "Left-to-Right" and the "Right-to-Left" implementations of the binary modular exponentiation, the doubling attack only works for the "Left-to-Right" implementation. Furthermore, the "Left-to-Right" implementation is often used since it requires only one variable. The new attack enables to recover the secret key decryption of RSA [15] or the key decryption of ElGamal [4]. It can also be used to obtain the secret key of the Diffie-Hellman authentication system. We only focus on the decryption cases. In this attack we assume that the adversary mounts a chosen ciphertext attack. This is a valid assumption in a side channel scenario, since randomized paddings avoiding chosen ciphertext attack, such as OAEP [1], are checked after the running of the decryption process. As a consequence, the binary exponentiation or multiplication is always performed and side channel attacks can be mounted on these algorithms. The attack on the RSA cryptosystem is a direct application of the doubling attack. On discrete-log based cryptosystems, we describe the attack in the elliptic curve setting, since the doubling attack allows to defeat classical countermeasures which are mainly proposed to elliptic curve systems.

In this paper, we first remind classical binary scalar multiplication algorithms. Then, we shortly describe different types of side channel attacks such as simple power analysis and differential power analysis but also the attack of Messerges, Dabbish and Sloan [11] in order to motivate the most frequently used countermeasures.

Then, we present an improvement of Messerges *et al* attack that applies when so-called downward algorithms are used. It has a much smaller complexity since it only requires two queries to the device in order to recover all the secret data. This new attack is called "*doubling attack*" since it is based on the doubling operation in the elliptic curve setting. We also explain how to use this new attack to defeat Coron's countermeasures [3].

2 Binary Scalar Multiplication Algorithms

In classical cryptosystems based on the RSA or on the discrete logarithm problem, the main operation is modular exponentiation. In the elliptic curve setting, the corresponding operation is the scalar multiplication. From an algorithmic point of view, those two operations are very similar; the only difference is the underlying group structure. In this paper, we consider operations over a generic group, without using any additional property. The consequence is an immediate application to the elliptic curve setting but it should be clear that all what we state can be easily transposed to modular exponentiation.

Scalar multiplication is usually performed using the "double-and-add" method that computes $d \times P$ using the binary representation of the scalar $d = \sum_{i=0}^{n} d_i \times 2^i$:

$$d \times P = \sum_{i=0}^{n} d_i \times \left(2^i \times P\right)$$

Two versions of the double-and-add algorithm are usually considered, according to the order of the terms in the previous sum. The first routine starts from the most significant bit and works downward. This method is usually called "Left-to-Right" (see figure 1).

$S = 0$
for i from n down to 0
    $S = 2.S$
    if $d_i = 1$ then $S = S + P$
return S

Fig. 1. Downward "Left-to-Right" double-and-add(P,d)

The second routine starts from the least significant bit and works upward. This method is also known as "Right-to-Left" (see figure 2).

$S = 0$
$T = P$
for i from 0 to n
    if $d_i = 1$ then $S = S + T$
    $T = 2.T$
return S

Fig. 2. Upward "Right-to-Left"double-and-add(P,d)

The first implementation is the most frequently used since it requires less memory. Up to now, no distinction was made on the security of those routines since all proposed attacks can be adapted to both implementations. In the following sections, we focus on the downward implementations and we show that it may be much more easily attacked than upward versions.

3 Power Analysis Attacks

It is well known that naive double-and-add algorithms are subject to power attacks introduced by Kocher *et al* [10]. More precisely, they introduced two types of power attacks : Simple Power Analysis (SPA) and Differential Power Analysis (DPA) we now shortly remind.

3.1 Simple Power Analysis

The first type of attack consists in observing the power consumption in order to guess which instruction is executed. For example, in the previous algorithm, one

can easily recover the exponent $d = \sum_{i=0}^{n} d_i 2^i$ by distinguishing the doubling from the addition instruction. To avoid this attack, downward double-and-add algorithm is usually modified using so-called "dummy" instructions (see figure 3).

```
S[0] = 0
for i from n down to 0
    S[0] = 2.S[0]
    S[1] = S[0] + P
    S[0] = S[d_i]
return S[0]
```

Fig. 3. Downward double-and-add(P,d) resistant against SPA

Although this new algorithm is immune to SPA, a more sophisticated treatment of power consumption measures can still enable to recover the secret scalar d.

3.2 Differential Power Analysis

DPA uses power consumption to retrieve information on the operand of the instruction. More precisely, it no longer focuses on which instruction is executed but on the Hamming weight of the operands used by the instruction. Such an attack has been described in the elliptic curve setting in [3,14].

This technique can also be used in a different way. Messerges, Dabbish and Sloan introduced "Multiple Exponent Single Data" attack [11]. Note that, for our purpose, a better name would be "Multiple Scalar Single Data". We first assume that we have two identical equipments available with the same implementation of algorithm 3, one with an unknown scalar d and another one with a chosen scalar e. In order to discover the value of d, using correlation between power consumption and operand value, we can apply the following algorithm. We guess the bit d_n of d which is first used in the double-and-add algorithm and we set e_n to this guessed value. Then, we compare the power consumption of the two equipments doing the scalar multiplication of the same message. If the consumption is similar during the two first steps of the inner loop, it means that we have guessed the correct bit d_n. Otherwise, if the consumption differs in the second step, it means that the values are different and that we have guessed the wrong bit. So, after this measure, we know the most significant bit of d. Then, we can improve our knowledge on d by iterating this attack to find all bits as it is illustrated in the algorithm of figure 4.

This kind of attack is well known and some classical countermeasures are often implemented. For example, the Chaum's blinding technique [2] can be used to protect an RSA implementation since it prevents an attacker from knowing the data used in the exponentiation. This method cannot be applied directly for

```
for i from 0 to n
    e_i = 0
for i from 0 to n
    e_{n-i} = 1
    choose M randomly
    double-and-add(P,d) on equipment 1
    double-and-add(P,e) on equipment 2
    if no correlation at step (i + 1) e_{n-i} = 0
return e
```

Fig. 4. MESD attack to find secret scalar d

computations based on the Discrete Logarithm problem as there is no public exponent associated with the secret exponent, as in RSA.

4 Usual DPA Countermeasures for Double-and-Add Algorithm

The most well know countermeasures for scalar multiplication on elliptic curve have been published by Coron [3]. In this paper, the author describes three different countermeasures which are respectively based on the blinding of the scalar, on the blinding of the point or on the blinding of the multiplication. We now recall the description of the first two countermeasures. Then we explain, in section 5, how to defeat them.

4.1 Coron's First Countermeasure

During the computation of a scalar multiplication, this scalar can be blinded by adding a multiple of the number $\mathcal{E}$ of points of the curve. For this purpose, the algorithm needs a random value r which length is fixed to 20 bits in [3]. Then, the algorithm computes $(d + r\mathcal{E})P$ which is obviously equal to dP. This countermeasure, depicted in figure 5, is very efficient since the scalar value changes for each computation.

```
pick random value r
d' = d + rℰ
return double-and-add(P,d')
```

Fig. 5. Implementation 1 secure against DPA

4.2 Coron's Second Countermeasure

The second solution is based on the same idea as Chaum's blind RSA signature scheme. However, since we use a discrete log based problem, applying this method requires twice the time needed for a single scalar multiplication. To be more efficient, it is proposed in [3] to store a secret point R and the associated value $S' = dR$. The multiplication of P by d is performed by computing $d(P+R)$ and then subtracting S' to the result. The variability is obtained by doubling R and S' at each execution as shown in figure 6.

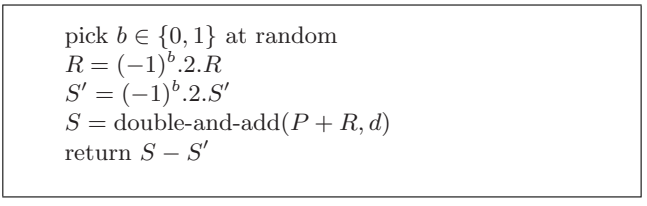

Fig. 6. Implementation 2 secure against DPA

In this paper we will not focus on the third countermeasure proposed by Coron since it has been partially broken by Goubin in [5]. Our attack does not work on this countermeasure and does not allow to enhance Goubin's attack.

These two countermeasures are well admitted to be efficient against power attacks. Other recently proposed countermeasures, such as randomized NAF [6] or [7,17,8] mainly focus on improving efficiency in terms of speed.

5 The New Attack

We introduce a new attack mainly based on two reasonable assumptions. This attack is able to recover the secret scalar with a few requests to the card. The adversary needs to send chosen messages directly to the double-and-add algorithm. Indeed, when considering decryption of the ElGamal cryptosystem or of the RSA cryptosystem for instance, the padding can only be verified at the end of the computation.

The idea of the attack is based on the fact that, even if an adversary is not able to tell which computation is done by the card, he can at least detect when the card does twice the same operation. More precisely, if the card computes $2.A$ and $2.B$, the attacker is not able to guess the value of A nor B but he is able to check if $A = B$. Such an assumption is reasonable since this kind of computation usually takes many clock cycles and depends greatly on the value of the operand. This assumption has been used in a stronger variant and validated by Schramm *et al.* in [16]. Indeed, they are able to distinguish collisions during one DES round computation which is much more difficult than distinguishing collisions during a doubling operation. If the noise is negligible, a simple comparison of the two

power consumption curves during the doubling will be sufficient to detect this equality.

If the noise is more important we propose two solutions to detect equalities. The first and easiest one is to compare the average of several consumptions curves with A and B. However asking twice the same computation may be impossible. In that case, a better solution is to use the tiny differences on many clock cycles since a point doubling usually takes a few thousand cycles. By summing the square of the differences between these curves on each clock cycle, we can decrease the influence of noise. This approach is precised in appendix A.

5.1 Description of the Doubling Attack

The so-called "*doubling attack*" is based on the fact that similar intermediate values may be manipulated when working with points P and $2P$. However this idea only works when using the downward routine.

Let us first consider an example. Let $d = 78 = 64 + 8 + 4 + 2$, i.e. $n = 6$ and

$$(d_0, d_1, d_2, d_3, d_4, d_5, d_6) = (0, 1, 1, 1, 0, 0, 1)$$

Then we compare the sequence of operations when the downward binary scalar multiplication algorithm of figure 3 is used to compute $d \times P$ (on the left) and $d \times (2P)$ (on the right) :

i	d_i	comput. of dP	comput. of $d(2P)$
6	1	2×0	2×0
		$0 + P$	$0 + 2P$
5	0	$2 \times P$	$2 \times 2P$
		$2P + P$	$4P + 2P$
4	0	$2 \times 2P$	$2 \times 4P$
		$4P + P$	$8P + 2P$
3	1	$2 \times 4P$	$2 \times 8P$
		$8P + P$	$16P + 2P$
2	1	$2 \times 9P$	$2 \times 18P$
		$18P + P$	$36P + 2P$
1	1	$2 \times 19P$	$2 \times 38P$
		$38P + P$	$76P + 2P$
0	0	$2 \times 39P$	$2 \times 78P$
		$78P + P$	$156P + 2P$
		return $78P$	return $156P$

If we focus on the doubling operations, we notice that some of them manipulate the same operand. More precisely, we observe that the doubling operation at rank i in the computation of dP is the same as the doubling operation at rank $i - 1$ in the computation of $d(2P)$ if and only if $d_{i-1} = 0$. Consequently, all the bits (except the least significant one) can be deduced from the SPA analysis of only two power consumption curves, the first one obtained during the computation of dP and the second one with $d(2P)$.

More formally, let us denote the partial sums $S_k(P) = \sum_{i=0}^{i=k} d_{n-i} 2^{k-i} \times P$. This value is the content of $S[0]$ in the algorithm 3 after $k+1$ iterations. Therefore we also refer to $S_k(P)$ as an intermediate result of the binary scalar multipliation algorithm. So this value is used in the next doubling operation in any case. Besides

$$\begin{aligned} S_k(P) &= \sum_{i=0}^{k} d_{n-i} 2^{k-i} \times P \\ &= \sum_{i=0}^{k-1} d_{n-i} 2^{k-1-i} \times (2P) + d_{n-k} \times P \\ &= S_{k-1}(2P) + d_{n-k} \times P \end{aligned}$$

Thus the intermediate result of the downward double-and-add algorithm with P at step k will be equal to the intermediate result with $2P$ at step $k-1$ if and only if d_{n-k} is null.

Using the same example as before, we obtain :

value of step k	0	1	2	3	4	5	6
value of d_{n-k}	1	**0**	**0**	1	1	1	**0**
value of $S_k(P)$	P	**2P**	**4P**	$9P$	$19P$	$39P$	**78P**
value of $S_k(2P)$	**2P**	**4P**	$8P$	$18P$	$38P$	**78P**	$156P$

In conclusion, we just need to compare the doubling computation at step $k+1$ for P and at step k for $2P$ to recover the bit d_{n-k}. If both computations are identical, d_{n-k} is equal to 0 otherwise d_{n-k} is equal to 1. This can also be observed by shifting the second measurement curve by one step to the right and comparing it to the first curve. Therefore, with only two requests to the card, it is possible to recover all the bits of the secret scalar.

Note that this attack also works with addition-subtraction chains such as Non Adjacent Form representation [12]. It allows to recover all the zeros in the NAF coding which represent roughly two third of the bits according to the paper of Morain and Olivos [12]. The missing information can be recovered by exhaustive search or by a more efficient method such as an adaptation of the baby step giant step algorithm for short Diffie-Hellman exponent. Indeed, if the prime group order is a 160-bit prime number, then only 54 bits remain to be discovered. Moreover, the Baby Step Giant Step algorithm can be used to reduce the complexity of the discovery of the discrete logarithm to 2^{27} in time and in memory.

5.2 Application of the Doubling Attack to Coron's Countermeasures

The first countermeasure of Coron uses a 20-bit random value to blind the secret scalar at each request to the card. This size of random value is sufficient to resist usual DPA attacks. However it is not enough to resist our new doubling

attack. Indeed, due to the birthday paradox, after 2^{10} requests with P and 2^{10} requests with $2P$, there should exist a common scalar with high probability. In order to recover this collision on the scalar, the attacker needs to compare each curve obtained with P to each curve obtained with $2P$. With the method described before, assuming the same scalar is used, it is possible to find the position of zeroes in this scalar. The right pair will be distinguished as it will give a scalar with enough zeroes. Indeed, if the corresponding scalars are different, it is unlikely that common intermediate values appear on both computations. Hence identifying the right pair is quite easy and requires only 2^{20} comparisons. In case several bits of the scalar cannot be clearly identified due to the noise, they can be recovered by exhaustive search. The attack is summarized in figure 7

```
while no correct pair is found
    request computation with P and store measurement C(P) in
    set A
    request computation with 2P and store measurement C(2P)
    in set B
    compare C(P) with all set B
    compare C(2P) with all set A
    if two measurements have many common intermediate squar-
    ing, a correct pair is found
exhaustive search for undefined bits of the scalar recovered with
the correct pair.
```

Fig. 7. Attack of Coron's first countermeasure

If the number of undefined bits is too large, one can notice that the number of correct pairs increases as the square of the number of extra requests. With about 2^{15} requests in each group, the number of correct pairs will be approximately 2^{10} which may help to decrease the work of the exhaustive search.

The second countermeasure is even more vulnerable to the doubling attack. Indeed, the random value which blinds P is itself doubled at each execution. The attack goes as follows : a point P is first sent as the first request. Then the card executes its routine with the value $P + R$. The adversary then requests the computation with the point $2P$. With probability $\frac{1}{2}$, the card will use the point $2P + 2R = 2(P + R)$. So the attacker is then able to compare the two measurements and to recover the secret scalar. If the noise is too important, the adversary can use a statistical approach. He can choose a random point Q, send Q and $2Q$ to the card and make the difference between the first curve and the second one shifted by one step to the right. By summing the square of those differences, we can recover all the bits of the secret scalar. Indeed, when the bit at step i is equal to 0, half of the differences at step $i + 1$ are null. So the curve representing the sum of the differences will be flatter at positions corresponding to a zero than at positions corresponding to a one.

6 Conclusion

A new powerful attack on scalar multiplication and modular exponentiation has been presented which takes advantage of some implementation choices that were not considered as a security concern up to now. As regards to this attack, it appears that the "bad" choice was the most commonly used, due to efficiency criteria. This vulnerability considerably weakens usual countermeasures used to defeat power attacks.

Since no attack as efficient as the doubling attack is known on the upward double-and-add algorithm (from the least to the most significant bit), we recommend to use this routine to compute scalar multiplication combined with the appropriate countermeasures.

It is an open problem to study whether our attack and Goubin's attack can be combined in order to defeat the combination of Coron countermeasures.

References

1. M. Bellare and P. Rogaway. Optimal Asymetric Encryption - How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92–111. Springer-Verlag, 1994.
2. D. Chaum. Blind Signatures for Untraceable Payments. In *Crypto '82*, pages 199–203. Plenum, NY, 1983.
3. J.S. Coron. Resistance against differential power analysis for elliptic curve. In *CHES '99*, LNCS 1717, pages 292–302. Springer-Verlag, 1999.
4. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *IEEE Transactions on Information Theory*, volume IT–31, no. 4, pages 469–472, july 1985.
5. L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *PKC '2003*, LNCS 2567, pages 199–210. Springer-Verlag, 2003.
6. J. Ha and S. Moon. Randomized signed-scalar Multiplication of ECC to resist Power Attacks. In *Pre-Proceeding CHES'02*, pages 553–565. Springer Verlag, 2002.
7. K. Itoh, J. Yajima, M. Takenaka, and N. Torii. DPA Countermeasures by Improving the Window Method. In *Pre-Proceeding CHES'02*, pages 304–319. Springer Verlag, 2002.
8. T. Izu, B. Moller and T. Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In *IndoCrypt '2002*, LNCS 2551, pages 296–313. Springer-Verlag, 2002.
9. P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Others Systems. In *Crypto '96*, LNCS 1109, pages 104–113. Springer-Verlag, 1996.
10. P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Crypto '99*, LNCS 1666, pages 388–397. Springer-Verlag, 1999.
11. T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power Analysis Attack of Modular Exponentiation in Smartcards. In *CHES '99*, LNCS 1717. Springer-Verlag, 1999.
12. F. Morain and J. Olivos. Speeding up the computation on an elliptic curve using addition-substraction chains. *Inform Theory Appl.*, 24:531–543, 1990.
13. NIST. Digital Signature Standard (DSS). Federal Information Processing Standards PUBlication 186–2, february 2000.

14. K. Okeya and K. Sakurai. A second-Order DPA Attack Breaks a Window-Method Based Countermeasure against Side Channel Attacks. In *ISC '2002*, LNCS 2433, pages 389–401. Springer-Verlag, 2002.
15. R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystem. *Communications of the ACM*, 21(2):120–126, 1978.
16. K. Schramm, T. Wollinger, and C. Paar. A New Class of Collision Attacks and its Application to DES. In *FSE '2003*, LNCS, pages –. Springer-Verlag, 2003.
17. E. Trichina and A. Bellezza. Implementation of Elliptic Curve Cryptography with Built-in Counter Measures against Side Channel Attacks. In *Pre-Proceeding CHES'02*, pages 98–113. Springer Verlag, 2002.

A Statistical Approach of Noise Reduction

Let c denotes the number of cycles of a point doubling operation. More precisely, we only consider the cycles that are data dependant, i.e., for which the power consumption differs when operands are changed. The power consumption (without noise) of the computation with A and B at cycle i are respectively named $C_A(i)$ and $C_B(i)$. The noise N can be modeled by random independent variables $N_A(i)$, $N_B(i)$ with mean μ and variance σ. The power consumption observed on cycle i are then equal to $C_A(i) + N_A(i)$ and $C_B(i) + N_B(i)$. The indicator I is defined as follow:

$$\begin{aligned} I &= \frac{1}{c}\sum_{i=1}^{c}(C_A(i) + N_A(i) - C_B(i) - N_B(i))^2 \\ &= \frac{1}{c}\sum_{i=1}^{c}(C_A(i) - C_B(i))^2 + \frac{1}{c}\sum_{i=1}^{c}(N_A(i) - N_B(i))^2 \\ &\quad + \frac{2}{c}\sum_{i=1}^{c}(N_A(i) - N_B(i))(C_A(i) - C_B(i)) \end{aligned}$$

If c is large enough, we can evaluate the mean of the indicator when $A = B$ and $A \neq B$. In the first case, $I = \frac{1}{c}\sum_{i=1}^{c}(N_A(i) - N_B(i))^2$, which is a sum of assumed independent variables $Y(i) = (N_A(i) - N_B(i))^2$. The mean of $Y(i)$ is

$$\begin{aligned} E(Y) &= E((N_A(i) - N_B(i))^2) \\ &= Var((N_A(i) - N_B(i)) - [E(N_A(i) - N_B(i))]^2 \\ &= Var(N_A(i)) + Var(N_B(i)) - 0 = 2\sigma^2 \end{aligned}$$

So the mean of the indicator, if $A = B$, is $E(I(A = B)) = 2\sigma^2$.

In the second case ($A \neq B$), assuming that

$$\forall i \leq c, \varepsilon_1 \leq |C_A(i) - C_B(i)| \leq \varepsilon_2$$

the mean of the indicator can be bounded as follow

$$2\sigma^2 + \varepsilon_1^2 \leq E(I(A \neq B)) \leq 2\sigma^2 + \varepsilon_2^2$$

With this bound in mind, it appears that the indicator can be used to distinguish if the manipulated data are equal or not. The confidence on this indicator relies on its variance and the number of clock cycles c to perform the operation.