Defeating Countermeasures Based on Randomized BSD Representations

Pierre-Alain Fouque[1], Frédéric Muller[2], Guillaume Poupard[2], and Frédéric Valette[2]

[1] École normale supérieure, Département d'informatique
45 rue d'Ulm 75230 Paris Cedex 05 France
Pierre-Alain.Fouque@ens.fr

[2] DCSSI Crypto Lab 51, Boulevard de Latour-Maubourg
75700 Paris 07 SP France
{Frederic.Muller,Guillaume.Poupard,Frederic.Valette}@sgdn.pm.gouv.fr

Abstract. The recent development of side channel attacks has lead implementers to use increasingly sophisticated countermeasures in critical operations such as modular exponentiation, or scalar multiplication on elliptic curves. A new class of countermeasures is based on inserting random decisions when choosing one representation of the secret scalar out of a large set of representations of the same value. For instance, this is the case of countermeasures proposed by Oswald and Aigner, or Ha and Moon, both based on randomized Binary Signed Digit (BSD) representations. Their advantage is to offer excellent speed performances. However, the first countermeasure and a simplified version of the second one were already broken using Markov chain analysis.
In this paper, we take a different approach to break the full version of Ha-Moon's countermeasure using a novel technique based on detecting local collisions in the intermediate states of computation. We also show that randomized BSD representations present some fundamental problems and thus recommend not to use them as a protection against side-channel attacks.

1 Introduction

Modular exponentiation or scalar multiplication are used by most popular public key cryptosystems like RSA [22] or DSA [4]. However, data manipulated during these computations should generally be kept secret, since any leakage of information (even only a few bits of secret information) might be useful to an attacker. For example, during the generation of an RSA signature by a cryptographic device, the secret exponent is used to transform an input related to the message into a digital signature via modular exponentiation.

Timings and power attacks, first introduced by Kocher [11, 12] are now well studied and various countermeasures have been proposed. These attacks represent a real threat when considering operations that involve secret data and require a long computation time. In general, naive implementations leak information about the secret key.

In [2], Coron has shown that several countermeasures are possible to prevent this type of leakage. In the context of Elliptic Curve Cryptosystems (ECC), he proposed different techniques based on blinding the critical data manipulated. An alternative approach is to randomize the number and the sequence of steps in the multiplication algorithm itself. In this type of countermeasure the usual scalar multiplication algorithm on ECC is replaced by a randomized addition-subtraction chain. From a general perspective, the idea is to allow additional symbols in the secret scalar representation. When using the set of digits $\{0, 1, -1\}$, we generally speak of Binary Signed Digit (BSD) representation. Then the multiplication algorithm picks at random a valid representation of the secret scalar. Actually several algorithms of this class have been proposed in the recent years [7, 13, 20], many of which have been broken quickly [16, 26, 27]. In this paper, we present a new side channel attack against randomized exponentiation countermeasures. We believe this result enlightens fundamental defects in these constructions.

Basically our attack scenario is that an attacker has physical access to a cryptographic device and tries to find the private key used by the device. He first obtains different encryptions of a fixed message. Since the scalar representation is randomized, the cryptographic device performs a different computation each time. However, we will show that collisions occur frequently at each step of computation. They can be detected using power consumption curves and reveal critical information concerning the private key. Our attack does not depend much on which public key encryption scheme is actually used, so we focus on the case of ECCs. Furthermore, the Ha-Moon's countermeasure [7], proposed at CHES'02, was designed originally for ECCs. It is straightforward to apply our ideas to RSA-based encryption schemes and even to signature schemes based on RSA with a deterministic padding (i.e. without randomization) such as PKCS#1 v1.5 [23]. Indeed all we need is the ability to send the same input several times to the cryptographic device.

In this paper, we first recall the classical binary scalar multiplication on elliptic curves. Then, we briefly describe different types of side channel attacks such as Simple Power Analysis (SPA) and Differential Power Analysis (DPA) but also the attack of Messerges, Dabbish and Sloan [14] in order to motivate common countermeasures. Next, we describe the principles of countermeasures using a randomized BSD representation through the example of [7]. In the last two sections, we expose some major weaknesses in this family of algorithms and describe a new collision-based attack against the full Ha-Moon's countermeasure.

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 with additive notations. We do not use additional properties of this group.

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 $k \times P$ from a point P, using the binary representation of the scalar $k = \sum_{i=0}^{n-1} k_i 2^i$:

$$k \times P = \sum_{i=0}^{n-1} k_i \times \left(2^i \times P\right)$$

This method is obviously analog to the "square-and-multiply" method for modular exponentiation. The resulting algorithm is described in Figure 1, where $\mathcal{O}$ is the point at infinity.

Input: a point P, an n-bit integer $k = \sum_{i=0}^{n-1} k_i 2^i$
Output: $k \times P$
$Q = \mathcal{O}$
for i from $n-1$ down to 0
    $Q = 2Q$
    if ($k_i == 1$) then $Q = Q + P$
return Q

Fig. 1. Naive "double-and-add" scalar multiplication algorithm

3 Power Analysis Attacks

It is well known that the naive double-and-add algorithm is subject to the power attacks introduced by Kocher *et al* [12]. More precisely, they introduced two types of power attacks : Simple Power Analysis (SPA) and Differential Power Analysis (DPA).

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 $k = \sum_{i=0}^{n-1} k_i 2^i$, provided the doubling instruction can be distinguished from the point addition. To avoid this attack, the basic "double-and-add always" algorithm is usually modified using so-called "dummy" instructions (see Figure 2).

Although this new algorithm is immune to SPA, a more sophisticated treatment of power consumption measures still enables the recovery of the secret scalar k.

Input: a point P, an n-bit integer $k = \sum_{i=0}^{n-1} k_i 2^i$
Output: $k \times P$
$Q[0] = \mathcal{O}$
for i from $n-1$ down to 0
  $Q[0] = 2Q[0]$
  $Q[1] = Q[0] + P$
  $Q[0] = Q[k_i]$
return $Q[0]$

Fig. 2. Double-and-add always algorithm resistant against SPA

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 attacks have been described, in the elliptic curve setting, in [2, 17].

This technique can also be used in a different way. Messerges, Dabbish and Sloan introduced "Multiple Exponent Single Data" attack [14]. Note that, for our purpose, a better name would be "Multiple Scalar Single Data". We first assume that we have two identical devices available with the same implementation of the algorithm of Figure 2, one with an unknown scalar k and another one for which the scalar e can be chosen and modified. In order to discover the value of k, using correlation between power consumption and operand value, we can apply the following algorithm. We guess the bit k_{n-1} of k which is first used in the double-and-add algorithm and we set e_{n-1} to this guessed value. Then, we compare the power consumption of the two devices doing the scalar multiplication of the same message. If the consumption is similar during the first two steps of the inner loop, it means that we have guessed the correct bit k_{n-1}. 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 k. Then, we can improve our knowledge on k by iterating this attack to find all bits as it is illustrated in the algorithm of Figure 3.

This kind of attack is well known and some classical countermeasures are often implemented (see [2, 9]).

4 Countermeasures Using a Randomized Scalar Representation

In the case of scalar multiplication on ECC, the most popular countermeasures against DPA are those proposed by Coron [2]. They include randomizing the secret scalar, blinding the point and using randomized projective coordinates. New directions for attacking these countermeasures have recently been proposed [5, 6] but none of them works when all protections proposed by Coron are applied

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

Fig. 3. MESD attack to find secret scalar k

simultaneously. It remains to be investigated if Coron's countermeasures can be defeated in the general case.

An alternative to Coron's countermeasures is to randomize the multiplication algorithm itself by introducing some random decisions. Two recent propositions [7, 20] use a randomized addition-subtraction chain, which is equivalent to represent the scalar with the alternative set of digits $\{0, 1, -1\}$. Both of them claim excellent performances in terms of speed, so they appear to be very attractive.

However, these countermeasures alone do not protect against SPA. This was illustrated recently by several new attacks [16, 18, 19]. They result from the assumption that distinguishing between the point addition (or subtraction) and the doubling instruction is possible. At CHES'03, a unified framework for such attacks, called the Hidden Markov Model attacks was proposed by Karlof and Wagner [10]. Hence randomized representation techniques are useful to counteract DPA attack but need to be strengthened in order to resist SPA.

A possible enhancement is to use special elliptic curves where the point addition (or subtraction) and the doubling instruction require exactly the same field operations [1, 8, 13]. Another approach is to transform these algorithms into "double-and-add-always" algorithms. Basically, this corresponds to the SPA countermeasure proposed by Coron [2]. The Ha and Moon's paper [7] actually proposes a SPA-immune algorithm using this technique. This strengthened version still remains to be broken. In the next sections, we focus on this algorithm and show how to break it using a completely different approach to the Markov model. More generally we expose some important defects in this class of countermeasures.

4.1 The Ha-Moon countermeasure

It is well known that any positive integer k can be represented as a finite sum of the form $k = \sum d_i \times 2^i$ where d_i is in a suitable fixed set of digits. When $d_i \in \{0, 1\}$, we obtain the usual binary representation of k. Another possible choice is to use the set of digits $\{0, 1, -1\}$ then we speak of Binary Signed Digit (BSD) representation. Such a representation of k is clearly no longer unique.

However, there exists a unique representation where no two consecutive d_i's are non-zero, i.e. $d_i d_{i+1} = 0$ for all $i \geq 0$. This representation is called the "Non Adjacent Form" (NAF).

The Ha-Moon countermeasure [7] uses concepts from the NAF recoding algorithm to pick at random a representation from a scalar of initial length n bits. Actually there exist many ways to build a representation of the form

$$k = \sum_{i=0}^{n-1} d_i \times 2^i$$

where $d_i \in \{0, 1, -1\}$. Indeed this system includes the binary representation, thus all positive integers $k \leq 2^n - 1$ are included along with their opposites. But there are 3^n possible combinations, so the representation is clearly redundant. The proposed countermeasure picks one of these representations using an auxiliary random source. This randomization is described in Section 4.2. During the process, it may increase the number of digits of k from n to $n + 1$.

Once a new representation of k has been chosen, the new digits are used in the multiplication algorithm. With the usual methods, only doubling and addition are mixed. Now, subtractions are also mixed into the algorithm. This idea of mixing addition and subtraction in elliptic curve computations has been known since a long time [15]. The full SPA-immune countermeasure with "double-and-add always" algorithm becomes :

```
Input: a point P, an integer k = Σ_{i=0}^{n} d_i 2^i
Output: k × P
Q[0] = O
P[1] = P, P[−1] = −P and P[0] = P
for i from n down to 0
    Q[0] = 2Q[0]
    Q[1] = Q[0] + P[d_i]
    Q[−1] = Q[1]
    Q[0] = Q[d_i]
return Q[0]
```

Fig. 4. NAF based Multiplication Algorithm

4.2 The randomization algorithm

The technique used to generate digits $d_i \in \{0, 1, -1\}$ is very efficient since it uses a simple table and increases the length of the scalar by at most one digit. It is very similar to the technique used to transform a binary representation into a NAF representation, referred to as NAF recoding algorithm. An analysis of this algorithm is given in [3]. For the purpose of our attack, we will describe it using the following notations :

A random value called $R = \sum_{i=0}^{i=n-1} r_i \times 2^i$ is generated and auxiliary carry bits, called c_i, are used (c_0 is set to 0). The digits d_i are then computed using the following table (taken from [7]).

Input				Output	
k_{i+1}	k_i	c_i	r_i	c_{i+1}	d_i
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	1	-1
0	1	0	0	0	1
0	1	0	1	1	-1
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	1	-1
1	0	1	1	0	1
1	1	0	0	1	-1
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	0

For instance, for a scalar of length $n = 9$ bits, $k = \mathtt{111011110}_2 = 478$ and with $r = \mathtt{110101001}_2$ we obtain, $c = \mathtt{1110111100}_2$ and $d = \mathtt{100\bar{1}1000\bar{1}0}_2$ where $\bar{1} = -1$ by definition. We call a the sum of k and $\frac{k}{2} \oplus r$, and $\sum_{i=0}^{i=n} a_i \times 2^i$ its binary representation. Consequently,

$$k + (\frac{k}{2} \oplus r) = a = \sum_{i=0}^{n} a_i \times 2^i$$

The c_i's can be seen as the carry bits in this addition. Then, by definition of the bits d_i's in the previous table,

$$d_i = a_i - (k_{i+1} \oplus r_i)$$

for $0 \leq i \leq n$. Therefore, the following relation holds

$$\sum_{i=0}^{n} d_i \times 2^i = a - (k/2 \oplus r) = k$$

which shows that we actually compute an alternative representation of k.

5 Weaknesses in Randomized Representations

In this section, we describe how to attack the Ha-Moon full countermeasure. Our attack takes advantage of inherent weaknesses in the randomized BSD representation. Moreover, it might also be applied to any countermeasure based on a similar principle.

5.1 Collision Attacks

Side channel attacks based on collision detection have been recently applied with success to block ciphers [24] and to public key cryptosystems [5]. Although it might be difficult in practice to detect which computation is done by the cryptographic device, it is usually much easier to detect internal data collisions. For instance, by analyzing the power consumption of a smart card, an adversary may be able to tell when the same sequence of instructions is executed by the card. More precisely, if the card computes $2 \times A$ and $2 \times B$, the attacker is not able to tell the value of A nor B, but he is able to check if $A = B$. Such an assumption is reasonable as long as the computation takes many clock cycles and depends greatly on the value of the operand. A stronger variant of this assumption has been validated by Schramm *et al.* in [24]. Indeed, they are able to detect collisions during one DES round computation which is arguably more difficult than detecting collisions during a doubling operation. If the noise is negligible, a simple comparison of the power consumption curves is sufficient to detect a collision.

We now focus on the randomized BSD representation and show how to obtain internal data collisions using the randomization algorithm described previously.

5.2 Intermediate states in the multiplication algorithm

The randomization algorithm proposed in [7] apparently generates a large number of alternative representations of the number k. Since n bits of randomness are used, we may indeed expect to obtain up to 2^n different sequences of digits d_i for each k. However, as we have seen before, there are only 3^{n+1} representations with $(n+1)$ digits d_i which must correspond to 2^n possible values of k. Consequently, there are on average $\simeq \left(\frac{3}{2}\right)^n$ representations per value of k, and not 2^n randomized representations. Moreover, at each step of the multiplication algorithm, the internal state may only take a reduced number of values. For sake of simplicity, we suppose in the following that computations are made upwards, while the algorithm initially proposed is downwards (see Figure 4). It should be clear that both directions yield similar properties. In the upward direction, after t steps, the value of $Q[0]$ corresponds to

$$Q[0] = \left(\sum_{i=0}^{t-1} d_i \times 2^i \right) \times P = D_t \times P$$

where $D_t = \sum_{i=0}^{i=t-1} d_i \times 2^i$ denotes the partial sum of the digits d_i and it is clear that $D_{n+1} = k$. At each step, the internal value D_t must also be compatible with $(k \bmod 2^t)$. Indeed, when we reach the most significant bit, we obtain the right value of k, except for a term of correction of the form $d_n \times 2^n$. More generally, it is easy to verify that

$$D_t = k \bmod 2^t - \varepsilon_t \times 2^t$$

where $\varepsilon_t = 0$ or 1. This can be directly seen from the relations of Section 4.2. Indeed, at step number t,

$$D_t = ((k + (k/2 \oplus r)) \bmod 2^t) - ((k/2 \oplus r) \bmod 2^t)$$

Therefore, after t steps of computation, although there are 2^t possible sequences of random bits, only 2 intermediate values - say V_t and V'_t - are possible, depending on ε_t. This is true independently of the direction of the computation . If we do it upwards, these values are $V_t = (k \bmod 2^t) \times P$ and $V'_t = ((k \bmod 2^t) - 2^t) \times P$ respectively. Furthermore, in Figure 5, we have represented the full computation of a NAF representation on a small integer using a customized scale. The two curves correspond to the two sequences of possible states V_t and V'_t for $0 \leq t \leq n$. At each step, the arrows represent the possible transitions. Horizontal segments represent the case $d_i = 0$, while upwards and downwards segments respectively represent the cases $d_i = 1$ and $d_i = -1$. Here, we use the value $k =\mathtt{10011110101}_2$. The upper and lower curve respectively correspond to the representations $\mathtt{010011110101}_2$ and $\mathtt{10\bar{1}\bar{1}0000\bar{1}0\bar{1}\bar{1}}_2$.

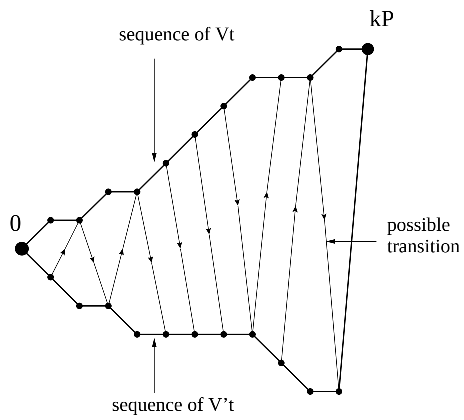

Fig. 5. A full NAF computation

However only a few state transitions are possible at each step. Indeed,

$$\begin{aligned} d_t \times 2^t &= D_{t+1} - D_t \\ &= k_t \times 2^t + \varepsilon_t \times 2^t - \varepsilon_{t+1} \times 2^{t+1} \end{aligned}$$

Hence, $d_t = k_t + \varepsilon_t - 2 \times \varepsilon_{t+1}$. It is easy to see that when k_t is fixed, there is a unique solution for each value of d_t

k_t	d_t	ε_t	ε_{t+1}
0	0	0	0
0	1	1	0
0	-1	1	1
1	0	1	0
1	1	0	0
1	-1	0	1

Thus, to each value of d_t corresponds a unique transition from one internal state to another at the corresponding step of computation. For instance, in the case of the upwards algorithm, this table of transitions is given in figure 6. A similar property holds in the downwards direction.

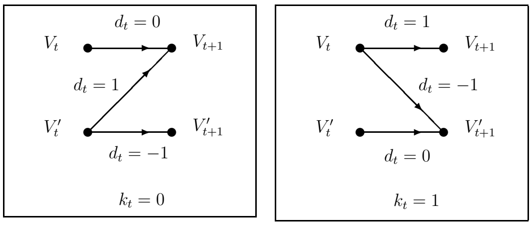

Fig. 6. State transitions

5.3 Fundamental Weaknesses in the BSD representation

Generally, although the number of valid randomized BSD representation for any given scalar is huge, only 3 situations are possible locally, at each step of the multiplication. If we analyze things more carefully, we notice that the arguments given previously hold independently of the randomization algorithm used to build an alternative representation of k. Indeed, after step t, the difference from the current intermediate value to the "real" intermediate value (the one that should be obtained with the usual multiplication algorithm) has to be $\pm 2^t$, otherwise it is impossible to correct this error at the following stages of computation (indeed they correspond to powers of 2 greater than 2^t). Thus, independently of the randomization algorithm, any given input scalar k yields a small limited number of local behaviors of the multiplication algorithm.

As we argued previously, it is possible to detect when collisions occur in the intermediate steps of computations by looking at the power consumption curves. Over a set of measurements, 3 groups $(d_t = 0, d_t = 1, d_t = -1)$ will be distinguished at each step t according to collisions on these power consumption curves. Each group corresponds to a value of d_t, but an attacker cannot tell which group corresponds to which value.

6 The Attack

In the last section, weaknesses of the BSD representation have been investigated. We suppose that no additional countermeasure is added, thus an attacker can repeat the same computation with a different randomization each time. This provides him with a set of measurements. Depending on local behaviors, groups can be built at each step of computation. In fact, when considering only one step, the transitions do not provide any useful information on the secret scalar. However, when considering two consecutive state transitions, we show that the case $k_t = k_{t+1}$ and the case $k_t \neq k_{t+1}$ can be distinguished.

6.1 Two cases : consecutive key bits that are equal and different

First, let us consider the case where $k_t = k_{t+1} = 0$ (similar observations hold when $k_t = k_{t+1} = 1$). Let $p_t(x)$ denote the probability that $d_t = x$ for $x = 0, 1, -1$

$$p_t(x) = Prob\,[d_t = x]$$

The transition function is represented in Figure 6. In addition, it is easy to see from the randomization table of Section 4.2 that when two state transitions are possible (corresponding to $d_t = \pm 1$), both occur with probability $\frac{1}{2}$. For example, in the case $k_t = k_{t+1} = 0$, when considering two consecutive state transitions, it is possible to derive the following relations :

$$p_{t+1}(0) = p_t(0) + p_t(1)\,, \quad p_{t+1}(1) = \frac{1}{2} \times p_t(-1)\,, \quad p_{t+1}(-1) = \frac{1}{2} \times p_t(-1)$$

Therefore, the cardinality of the group of collisions corresponding to $d_t = 0$ will grow very quickly when consecutive key bits are equal to 0. Actually, exactly the same property holds when they are equal to 1. More precisely, when we start from any probabilities at step t and two consecutive bits of the secret key are the same, we can even guarantee that

$$p_{t+1}(0) \geq \frac{1}{2}$$

Indeed, we have seen that $p_t(1) = p_t(-1)$. Besides, $p_t(1) + p_t(-1) + p_t(0) = 1$, thus

$$p_t(-1) \leq \frac{1}{2}$$

and

$$p_{t+1}(0) = 1 - p_{t+1}(1) - p_{t+1}(-1) = 1 - p_t(-1) \geq \frac{1}{2}$$

On the other hand, when two consecutive bits of secret key are different, the probabilities tend to average. If we suppose $k_t = 0$ and $k_{t+1} = 1$, then

$$\begin{aligned} p_{t+1}(0) &= p_t(-1) \\ p_{t+1}(1) &= \frac{1}{2} \times (p_t(0) + p_t(1)) \\ p_{t+1}(-1) &= \frac{1}{2} \times (p_t(0) + p_t(1)) \end{aligned}$$

Using similar arguments as in the previous case, it is straightforward to verify that all 3 probabilities fulfill

$$p_{t+1}(x) \leq \frac{1}{2}$$

for $x = 0, 1, -1$. To summarize, we have two very distinct situations. When two consecutive key bits are equal, one of the transitions will be by far over-represented inside our group of measurements. It is guaranteed that this will happen with probability $\geq 50\%$. In contrary, when two consecutive bits differ, the probability of the 3 transitions tend to average and none can be > 50%. Thus, we have two cases which are very easy to distinguish : $k_t = k_{t+1}$ and $k_t \neq k_{t+1}$.

The limit case is when the distribution of d_t is of the form $(\frac{1}{2}, \frac{1}{2}, 0)$. In this particular situation, it is difficult to distinguish $k_t = k_{t+1}$ from $k_t \neq k_{t+1}$ since both will yield distributions of the form $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ at the next step.

6.2 An efficient key recovery technique

As we have seen, it is easy in the general case to determine whether $k_t = k_{t+1}$ or $k_t \neq k_{t+1}$ by observing the distributions of d_t and d_{t+1}. Roughly, using 100 measurement curves appears to be sufficient to recover this information. However, we have a problem when the distribution at step i is close to $(\frac{1}{2}, \frac{1}{2}, 0)$. Typically, this happens just after a long run of 0's or 1's. Then, the distribution at the previous step was of the form $(1, 0, 0)$. When the long run ends, we can detect it easily because the distribution changes to $(\frac{1}{2}, \frac{1}{2}, 0)$. But the next step is very tricky, since distributions will be the same whatever the next secret key bit may be.

If the run is not too long - say t consecutive bits - there is a small bias at the tricky step, say $\varepsilon \simeq 2^{-t}$ between both distributions. Thus they can be distinguished if the number of available curves is about $M \simeq 2^{2t}$. In practice, if $n = 160$ bits, there will be few runs of more than $t = 5$ consecutive equal bits. We picked randomly 10^6 values of k and we obtained an average of 4.91 such long runs. Therefore, it is not a problem to guess the "tricky" bits in these cases. For shorter runs (of length $\leq t$), we can recover secret key bits after about $2^{2t} \simeq 1000$ requests to the cryptographic device.

Moreover, our algorithm is quite resistant to errors of measurement. Indeed, if this probability of error is not too high, we can basically apply the same statistical arguments, using an increased number of message. In situations were the bias ε is smaller than the probability of error in the measurements, it may become impossible to distinguish the two cases $k_t = k_{t+1}$ and $k_t \neq k_{t+1}$. However, as we argued previously, this happens quite rarely and it is not a problem to guess a few additional bits of secret key.

6.3 Practical Simulation

We have implemented a simulation of our attack. Using numbers of length $n = 160$ bits, we have fixed threshold values corresponding to the previous analysis. For different values of the number of messages M, we have applied our technique. Results are summarized in the following table (sequences have been truncated to 40 bits in order to fit). The terminology of symbols is the following

- `=` represents the case $k_t = k_{t+1}$
- `#` represents the case $k_t \neq k_{t+1}$
- `!` represents the case when our algorithm has made an error
- `?` represents the case when our algorithm has been unable to make a decision

M	n	Secret key	Errors	Unknowns
		`0010100100011101011111101111010011001011 ...`		
10	160	`=#!#?=#?==#????##=====#!===#?#=#??=###? ...`	8	46
		Average over 1000 keys	10.7	43.6
100	160	`=#?#?=##==#?=#?#?=====#?===#?#=#?#=##?? ...`	0	33
		Average over 1000 keys	1.5	32.4
1000	160	`=#?##=##==#==####=====#?===###=#=#=###= ...`	0	7
		Average over 1000 keys	0.5	8.8

Then we repeated the same experience, but we introduced errors of measurement. Supposing a 10% rate of errors, we obtain the following table

M	n	Secret key	Errors	Unknowns
		`0110010011001110101001100111010001000001 ...`		
10	160	`#=#=#??#??=#==##????#??=#=!#?#?=##====# ...`	14	47
		Average over 1000 keys	20.9	48.0
100	160	`#?#?##=#?#?#?=###?#=#?#=#?=###?=#?====# ...`	5	38
		Average over 1000 keys	5.6	35.7
1000	160	`#!#=##=#?#=#==#####=#=#=#==###==##====# ...`	1	10
		Average over 1000 keys	2.6	10.9

One sees that our algorithm is quite efficient. In practice, 100 queries are sufficient to reduce the entropy of the secret key to about 40 bits. Using 1000 messages, only 10 bits remain unknown. Errors of decision are not problematic, since we can usually detect which positions are likely to cause such errors. Recovering the secret bits that remain unknown can be done in different ways. The simpler technique is an exhaustive search, which is not a problem if the entropy is of 40 bits. An improved strategy would be the "baby step-giant step" technique, which would lower the complexity to 2^{20}, but would also require a memory of size 2^{20}. It is still an open problem to avoid this memory complexity, for instance using a lambda-method [21, 25] when the missing bits are disseminated.

6.4 Link with Markov model cryptanalysis

Recently, a new framework for attacking randomized multiplication algorithm has been proposed [10]. This technique is called Hidden Markov Model Cryptanalysis (HMMC). The idea is that randomized computations realized by a cryptographic device can be represented by a probabilistic finite state machine (which might also support inputs), with known probabilities of transition. The attacker has no direct access to the state of the machine, however each state corresponds to an output to which the attacker has access. This output can be seen as the side channel information. The cryptanalysis problem is to infer the most likely secret key from this partial information about the sequence of state. Applications have been proposed to variants of the "double-and-add" algorithm using randomization, like [20]. In this case, the partial information is the sequence of doubling instructions and additions performed during the computation.

However, this technique does not work on the full randomized BSD countermeasure proposed in [7], since it uses a "double-and-add-always" algorithm.

Therefore, the usual attack based on distinguishing additions from doubling instructions does not work here. Although there exists a clear state transition function, no obvious state-related output can be observed through side channel.

Our contribution here is a new attack that can still be viewed in the framework of HMMC. Indeed, in our case, the probabilistic finite state machine contains the 3 probabilities corresponding to $d_i = 0$, 1, and -1 in an unknown order. What we observe corresponds to an experimental distribution resulting from multiple measurements. We have shown that it was possible to infer the sequence of states and the bits of secret key from this observation.

7 Conclusion

A new powerful attack against algorithms using a randomized BSD representation has been presented. It is based on detecting and exploiting internal collisions. We have taken the example of Ha-Moon countermeasure and demonstrated how to break their full SPA-immune algorithm, by comparing power consumption curves for the same message and different randomizations. This is the first attack against the full Ha-Moon countermeasure proposed at CHES'02. However it works only in situations where the messages are not randomized.

More generally we have pointed out the lack of entropy in these BSD representations. Indeed, at each step of computation, only a small number of states are possible which results in collisions on intermediate values. Any reasonable countermeasure based on randomizing the multiplication algorithm should guarantee locally a large number of possible internal states and a large number of possible transitions from each state. Randomized BSD representations do not satisfy this constraint, which is a fundamental weakness.

References

1. E. Brier and M. Joye. Weierstrass Elliptic Curves and Side-Channel Attacks. In *Public Key Cryptography – 2002*, LNCS 2274, pages 335–345. Springer, 2002.
2. J-S. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Cryptographic Hardware and Embedded Systems (CHES) – 1999*, LNCS 1717, pages 292–302. Springer, 1999.
3. N. Ebeid and A. Hasan. Analysis of DPA Countermeasures Based on Randomizing the Binary Algorithm. Technical Report CORR 2003-14, 2003. `http://www.cacr.math.uwaterloo.ca/techreports/2003/corr2003-14.ps`.
4. FIPS PUB 186-2. Digital Signature Standard (DSS), 2000.
5. P-A. Fouque and F. Valette. The Doubling Attack – Why Upwards is Better than Downwards. In *Cryptographic Hardware and Embedded Systems (CHES) – 2003*, LNCS 2779, pages 269–280. Springer, 2003.
6. L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *Public Key Cryptography – 2003*, LNCS 2567, pages 199–210. Springer, 2003.
7. J. Ha and S. Moon. Randomized signed-scalar Multiplication of ECC to resist Power Attacks. In *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, LNCS 2523, pages 551–563. Springer, 2002.
8. M. Joye and J.-J. Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. In *Cryptographic Hardware and Embedded Systems (CHES) – 2001*, LNCS 2162, pages 402–410. Springer, 2001.

9. M. Joye and C. Tymen. Compact Encoding of Non-adjacent Forms with Applications to Elliptic Curve Cryptography. In *Public Key Cryptography*, LNCS 1992, pages 353–364. Springer, 2001.
10. C. Karlof and D. Wagner. Hidden Markov Model Cryptanalysis. In *Cryptographic Hardware and Embedded Systems (CHES) – 2003*, LNCS 2779, pages 17–34. Springer, 2003.
11. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Others Systems. In *Advances in Cryptology – Crypto'96*, LNCS 1109, pages 104–113. Springer, 1996.
12. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology – Crypto'99*, LNCS 1666, pages 388–397. Springer, 1999.
13. P.-Y. Liardet and N. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In *Cryptographic Hardware and Embedded Systems (CHES) – 2001*, LNCS 2162, pages 391–401. Springer, 2001.
14. T. Messerges, E. Dabbish, and R. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. In *Cryptographic Hardware and Embedded Systems (CHES) – 1999*, LNCS 1717, pages 144–157. Springer, 1999.
15. F. Morain and J. Olivos. Speeding up the Computation on an Elliptic Curve using Addition-Substraction Chains. In *Inform. Theory Appl.*, 24:531–543, 1990.
16. K. Okeya and D.-G. Han. Side Channel Attack on Ha-Moon's Countermeasure of Randomized Signed Scalar Multiplication. In *Advances in Cryptology – INDOCRYPT'03*. To appear.
17. K. Okeya and K. Sakurai. Power Analysis Attacks and Algorithmic Approaches to their Countermeasures for Koblitz curve Cryptosystems. In *Advances in Cryptology – INDOCRYPT'00*, LNCS 1965, pages 93–108. Springer, 2000.
18. K. Okeya and K. Sakurai. On Insecurity of the Side Channel Attack Countermeasure using Addition-Substraction Chains under Distinguishability Between Addition and Doubling. In *Australasian Conference on Information Security and Privacy - ACISP'02*, LNCS 2384, pages 420–435. Springer, 2002.
19. E. Oswald. Enhancing Simple Power-Analysis Attacks on Elliptic Curves Cryptosystems. In *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, LNCS 2523, pages 83–97. Springer, 2002.
20. E. Oswald and K. Aigner. Randomized Addition-substraction Chains as a Countermeasure against Power Attacks. In *Cryptographic Hardware and Embedded Systems (CHES) – 2001*, LNCS 2162, pages 39–50. Springer, 2001.
21. J. M. Pollard. Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, 32(143):918–924, July 1978.
22. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM*, 21(2):120–126, 1978.
23. RSA Laboratories. PKCS #1 v1.5 : RSA Encryption Standard, 1993. Available at `http://www.rsalabs.com/pkcs/pkcs-1`.
24. K. Schramm, T. Wollinger, and C. Paar. A New Class of Collision Attacks and its Application to DES. In *Fast Software Encryption – 2003*, LNCS 2887. Springer, 2003.
25. P. C. van Oorschot and M. J. Wiener. On Diffie-Hellman Key Agreement with Short Exponents. In *Eurocrypt '96*, LNCS 1070, pages 332–343. Springer, 1996.
26. C. Walter. Breaking the Liardet-Smart Randomized Exponentiation Algorithm. In *CARDIS 2002*, 2002. Available at `http://www.usenix.org/`.
27. C. Walter. Issues of Security with the Oswald-Aigner Exponentiation Algorithm. In *CT-RSA 2004*, LNCS 2964, pages 208–221. Springer, 2004.