Cryptanalysis of the Countermeasures Using Randomized Binary Signed Digits

Dong-Guk Han[1⋆], Katsuyuki Okeya[2], Tae Hyun Kim[1],
Yoon Sung Hwang[3], Young-Ho Park[4], and Souhwan Jung[5⋆⋆]

[1] Center for Information and Security Technologies(CIST),
Korea University, Seoul, KOREA
{christa,thkim}@korea.ac.kr
[2] Hitachi, Ltd., Systems Development Laboratory,
292, Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
ka-okeya@sdl.hitachi.co.jp
[3] Department of Mathematics, Korea University, Seoul, KOREA
yhwang@semi.korea.ac.kr
[4] Dept. of Information Security, Sejong Cyber Univ., Seoul, KOREA
youngho@cybersejong.ac.kr
[5] School of Electronic Engineering, Soongsil University, Seoul, KOREA
souhwanj@ssu.ac.kr

Abstract. Recently, side channel attacks (SCA) have been recognized as menaces to public key cryptosystems. In SCA, an attacker observes side channel information during cryptographic operations, and reveals the secret scalar using the side channel information. On the other hand, elliptic curve cryptosystems (ECC) are suitable for implementing on smartcards. Since a scalar multiplication is a dominant step in ECC, we need to design an algorithm to compute scalar multiplication with the immunity to SCA. For this purpose, several scalar multiplication methods that utilize randomized binary-signed-digit (BSD) representations were proposed. This type of countermeasures includes Ha-Moon's countermeasure, Ebeid-Hasan's one, and Agagliate's one. In this paper we propose a novel general attack against *"all"* the countermeasures of this type. The proposed attack lists the candidates for the secret scalar, however straight-forward approach requires huge memory, thus it is infeasible. The proposed attack divides the table into small tables, which reduces the memory requirement. For example, the computational cost and the memory requirement of the proposed attack for revealing the 163-bit secret key are $O(2^8)$ and $O(2^{23})$, respectively, using 20 observations on the scalar multiplication with Ha-Moon's countermeasure. The computational cost and the memory requirement are $O(2^{21})$ and $O(2^{12})$ for Ebeid-Hasan's one, and $O(2^{40})$ and $O(2^6)$ for Agagliate's one. If 40 observations are used, computational cost for Agagliate's one is reduced to $O(2^{33})$. Whenever we utilize a countermeasure of BSD type, we should

⋆ This research was supported by University IT Research Center Project.
⋆⋆ This work was supported by Korea Research Foundation Grant. (KRF-2001-042-E00045)

M. Jakobsson, M. Yung, J. Zhou (Eds.): ACNS 2004, LNCS 3089, pp. 398–413, 2004.

beware of the proposed attack. In other words, the security of BSD type is controversial.

Keywords: Elliptic Curve Cryptosystem, Side Channel Attacks, SPA, DPA, BSD Representation, Ha-Moon's Countermeasure, Ebeid-Hasan's Countermeasure, Agagliate's Countermeasure

1 Introduction

Recently, side channel attacks (SCA) have been recognized as menaces to public key cryptosystems. In SCA, an attacker observes side channel information such as computation timing, power consumption, and electro-magnetic radiation while a cryptographic device performs cryptographic operations, and utilizes such side channel information for revealing the secret information stored in the device [KJJ99]. On the other hand, elliptic curve cryptosystems (ECC) are suitable for implementing on a constrained device such as smartcard, since ECC achieves higher security using shorter key size. In ECC, a dominant computation is a scalar multiplication, which computes the scalar multiplied point $kP = \underbrace{P + \cdots + P}_{k \text{ times}}$ from an integer k and a point P. The integer is referred to as scalar, and is often secret. The attacker's goal is to detect the secret scalar during scalar multiplication. Thus constructing an efficient computation method of scalar multiplication which is secure against SCA and analyzing its security are important research topics.

For this purpose, many countermeasures against SCA were proposed. In particular, the countermeasure that utilizes randomized binary-signed-digit (BSD) representations of the secret scalar is a popular countermeasure. This type of countermeasure encodes the secret integer into BSD representation, then computes the scalar multiplied point using the representation. In addition, a different representation is used for each scalar multiplication. This thwarts the attacker's guess. This type of countermeasures is referred to as BSD type and it includes Ha-Moon's countermeasure [HaM02], Ebeid-Hasan's countermeasure [EH03], and the countermeasure of Agagliate et al [AGO03].

Recently, Okeya-Han [OH03] have proposed an attack algorithm to Ha-Moon [HaM02] method. However their analysis technique is ad-hoc in the sense that it tailored specifically to the target countermeasure, and it is not clear how to generalize it to analyze other countermeasures of BSD type. In other words, their attack is not applicable to every countermeasures of BSD type. On the other hand Karlof-Wagner [KW03] proposed the hidden Markov model cryptanalysis, which is a cryptanalytic framework for countermeasures that utilizes a probabilistic finite state machine. In fact, since some countermeasures of BSD type utilize it for encoding the secret scalar k into BSD representation, their attack is applicable. However, if the target countermeasure does not utilize it, the attack is not applicable. Ebeid-Hasan's countermeasure [EH03] is such an example. To sum up, no general attacks that are applicable to every countermeasures of BSD type have been proposed so far.

In this paper, we propose a novel attack against the countermeasures of BSD type. The proposed attack is applicable to *"all"* the countermeasures of BSD type; independent from the encoding method of the secret scalar. The use of several observations of side channel information reduces the search space for the secret scalar, however, this approach increases the memory requirement. The proposed attack reduces the memory requirement using the trick of a division into small tables, which is an analogy of the meet-in-the-middle attack against double DES. If the table is divided into u small tables, the required memory is reduced into the u-th root of the original in general, but this is dependent on the way to divide the table. We experimentally estimated the computational cost and the memory requirement according to the number of AD sequences for standard 163-bit secret scalars. Regarding Ha-Moon's countermeasure [HaM02], the proposed attack revealed the 163-bit secret key with $O(2^8)$ computational cost and $O(2^{23})$ memory requirement using 20 observations of the scalar multiplication. In the case of Ebeid-Hasan's countermeasure [EH03], the computational cost and the memory requirement are respectively $O(2^{21})$ and $O(2^{12})$ using 20 observations. In the case of Agagliate's countermeasure [AGO03], they are $O(2^{40})$ and $O(2^6)$ using 20 observations, and the use of 40 observations reduces the computational cost to $O(2^{33})$ with $O(2^8)$ memory requirement. Therefore, whenever we utilize a countermeasure of BSD type, we should beware of the proposed attack. In other words, the security of BSD type is controversial.

The paper is organized as follows: Section 2 surveys elliptic scalar multiplication and side channel attacks. Section 3 proposes the attack against the countermeasures of BSD type, and shows examples and implementation results for the proposed attack.

2 Elliptic Scalar Multiplication and Side Channel Attacks

In this section, we review elliptic scalar multiplication and side channel attacks. Some countermeasures against side channel attacks utilize randomized binary-signed-digit representations of the secret integer. The main topic of this paper is to construct an attack against such countermeasures. We review the countermeasures of this type in this section.

2.1 Elliptic Scalar Multiplication

Elliptic curve cryptosystem (ECC) is suitable for the implementation on constraint devices such as smart cards due to its short key size. In order to accelerate ECC, we need to optimize elliptic scalar multiplication, since elliptic scalar multiplication is usually a dominant operation in ECC. Note that the elliptic scalar multiplication is the operation that computes the scalar multiplied point kP from an elliptic point P and an integer k, and k is often secret in ECC.

Addition-Subtraction_Method is an efficient method to compute elliptic scalar multiplication, and the algorithm is as follows:

Addition-Subtraction_Method

INPUT A point P, and $k = \sum_{j=0}^{n} k_j 2^j$, $k_j \in \{-1, 0, 1\}$
OUTPUT $Q = kP$

1. $Q \leftarrow \mathcal{O}$
2. for $j = n$ downto 0
   2.1. $Q \leftarrow$ECDBL(Q)
   2.2. if $k_j = 1$ then $Q \leftarrow$ECADD(Q, P)
   2.3. if $k_j = -1$ then $Q \leftarrow$ECSUB(Q, P)
3. Return Q

Here, ECADD, ECDBL, and ECSUB stand for elliptic addition, elliptic doubling, and elliptic subtraction, respectively. $\mathcal{O}$ denotes the identity element of the elliptic addition, namely the point at infinity. Note that the cost of ECSUB is same to that of ECADD, since we can compute the inverse $-P$ from the point P without additional cost.

2.2 Side Channel Attacks

Side channel attacks (SCA) are a serious menace for embedded devices which are running cryptographic applications and leaking critical information through side channels, like power consumptions [KJJ99]. The attack aims at guessing the secret key (or some related information) using the correlation between some side channel information and the secret. For example, SCA retrieves some secret information while Addition-Subtraction_Method performs. The flow calculates ECADD or ECSUB if and only if the j-th bit k_j is non-zero. The standard implementation of ECADD is different from that of ECDBL [CMO98], however, ECADD and ECSUB are very similar, since ECSUB$(Q, P) =$ ECADD$(Q, -P)$. Although ECADD and ECSUB are ambiguous in the sense of SCA (e.g. power consumption), the attacker can distinguish these operations from ECDBL. Thus the attacker can recognize whether the bit is zero or not.

2.3 Countermeasures Using Randomized Representations of Binary Signed Digits

Many countermeasures against side channel attacks were proposed. The countermeasure that utilizes randomized binary-signed-digit (BSD) representations of the secret scalar for resisting against side channel attacks is a popular one. This type of the countermeasure is referred to as BSD type. Considering the binary representation of an integer k as one of its BSD representations, different BSD representations for k can be obtained by replacing 01 with $1\overline{1}$ and vice versa and by replacing $0\overline{1}$ with $\overline{1}1$ and vice versa. For example if $k = (13)_{10}$ is represented in 5 bits, i.e., $k = (01101)_2$, the different BSD representations for k are: 01101, $0111\overline{1}$, $10\overline{1}01$, $1\overline{1}101$, and so forth. The countermeasures of BSD type utilizes such different BSD representations for thwarting the attacker's guess. Several examples of BSD type are Ha-Moon's countermeasure [HaM02], Ebeid-Hasan's

countermeasure [EH03], and the countermeasure of Agagliate et al. [AGO03]. Ha-Moon [HaM02] proposed the randomized signed scalar multiplication method. Ebeid and Hasan [EH03] proposed a general version of the countermeasure of this type. In addition, Agagliate et al. [AGO03] proposed another countermeasure of this type.

On the other hand, Oswald-Aigner [OA01] proposed the randomized addition-subtraction chains method, which is similar to BSD type. It does not use a BSD representation but a similar one. Note that Oswald's countermeasure directly computes the scalar multiplication without the conversion into another representation of the integer, but it implicitly uses such a representation. Unfortunately, Oswald's countermeasure was broken. Okeya-Sakurai [OS02] proposed an attack against the basic version of Oswald's countermeasure. Later they extends their attack to the advanced version of Oswald's countermeasure [OS03]. Han et al. also proposed the attack against the advanced version [HCJ$^+$03]. Finally Walter [Wal03] broke the generalized version of Oswald's countermeasure.

Recently, Okeya-Han [OH03] have proposed an attack algorithm to Ha-Moon [HaM02] method. However their analysis technique is ad-hoc in the sense that it tailored specifically to the target countermeasure, and it is not clear how to generalize it to analyze other countermeasures of BSD type, for instance, Ebeid-Hasan's countermeasure [EH03] and Agagliate's countermeasure [AGO03]. In other words, their attack is not applicable to every countermeasures of BSD type.

While some countermeasures of BSD type utilize a probabilistic finite state machine for encoding the secret k into BSD representation, Karlof-Wagner [KW03] proposed the hidden Markov model cryptanalysis against such countermeasures. However, if the operational behavior of the target countermeasure does not modeled by a probabilistic finite state machine, their attack does not work. For example, Ebeid-Hasan's countermeasure [EH03] does not utilizes a probabilistic finite state machine, thus the hidden Markov model cryptanalysis is not applicable.

To sum up, no general attacks that are applicable to every countermeasures of BSD type have been proposed so far. In the next section we will propose such a general attack. Hence, the security of BSD type is controversial.

We give some notations which are used in the next section.

Notations: Let $k = \sum_{j=0}^{n-1} k_j 2^j$ with $k_j \in \{0, 1\}$ be the n-bit secret binary value and $d = \sum_{j=0}^{n} d_j 2^j$ with $d_j \in \{-1, 0, 1\}$ be the $(n+1)$-bit random recoded number generated from k by a random recoding method. We obviously have $k = d$. Let the i-th random recoded number of k be denoted as $d^{(i)} := \sum_{j=0}^{n} d_j^{(i)} 2^j$, where $d_j^{(i)} \in \{-1, 0, 1\}$. Let N and N' be the set $\{0, 1, ..., n\}$ and $\{1, 2, ..., n-1\}$, respectively. For $d^{(i)}$, the set of indices $J^{(i)} \subset N$ stands for $\{j \in N | d_j^{(i)} = 0\}$. Define $d^{(i)}(s, t) := \sum_{j=s}^{t} d_j^{(i)} 2^j$. Here, $0 \leq s \leq t \leq n$. $w(d^{(i)})$ denotes the number of non-zero digits in $d^{(i)}$, i.e., hamming weight of $d^{(i)}$. $|d_j^{(i)}|$ denotes the absolute value of $d_j^{(i)}$. For a set A, $\#A$ denotes the number of the elements in A.

3 Proposed Attack

In this section we propose a general side channel attack against the countermeasures with randomized BSD representations, and show the countermeasures are vulnerable to the proposed attack. Namely, the proposed attack algorithm does not depend on the method of randomizing BSD representation.[1] First, we introduce the main idea of the proposed attack, and propose a proposition and a theorem about the relations among BSD representations for the same integer. Then, we describe the attack algorithm with the proposed proposition and theorem, display an example of the proposed attack, and show experimental results.

3.1 Main Idea

We describe the main idea of our proposed attack.

Assumption 1: Assume that an attacker obtained m random recoded number $d^{(i)}$ generated from the secret value k and he knows the positions of zero digit, i.e., he can determine $d_j^{(i)}$ is zero or not. But he does not know whether $d_j^{(i)}$ is 1 or -1 when $d_j^{(i)} \neq 0$. In addition, he knows the plaintext-ciphertext pair (P, kP).

In general, the attacker can find the secret value k from the following exhaustive search method if his computing power is unlimited. The goal of the attacker is to determine whether $d_j^{(i)}$ is 1 or -1 when $d_j^{(i)} \neq 0$ because he knows the positions of zero digit. Note the non-zero most significant bit (MSB) is 1.

Exhaustive Search Method (ESM).

1. For $i = 1$ to m do
   - Make Table$_i$ which contains $2^{w(d^{(i)})-1}$ integers, which are $n+1$-bit integers, determined by $d_j^{(i)} \neq 0$ in $d^{(i)}$. The nonzero MSB is 1.
     Here, Table$_i = \{k' \mid k' = \sum_{j \in N - J^{(i)}} d_j^{(i)} 2^j$ with $d_j^{(i)} \in \{1, -1\}\}$.
2. Find the candidates for the secret key k.
   $\cap_{i=1}^{m}$Table$_i$ is the set of all possible keys, and $\#[\cap_{i=1}^{m}\text{Table}_i] = l > 0$, since $d^{(1)} = \cdots = d^{(m)} = k$.
3. Test l possible keys.
   For any $k' \in \cap_{i=1}^{m}\text{Table}_i$, compute $k'P$. If $k'P = kP$ then k' is the secret key.
4. Time complexity (TC) of ESM is $O(l)$ scalar multiplications and memory complexity (MC) of ESM is $O(y)$ memory for $n+1$-bit integers, where $y = \sum_{i=1}^{m} 2^{w(d^{(i)})-1}$.

The goals of this paper can be categorized as follows:

1. Reduce MC of ESM to the reasonable bound.

[1] Thus, in this paper, we do not describe the explicit algorithms of these countermeasures. For comprehensive descriptions, see [HaM02,EH03,AGO03].

2. Accurately determine TC of ESM only with obtained random recoded numbers $d^{(i)}$ without making tables.

The main idea of reducing the memory complexity is very simple. It is similar to meet-in-the-middle attack on double DES. Suppose $d^{(1)}$ and $d^{(2)}$ are $n+1$-bit random recoded numbers from the n-bit secret key k. In Step 1 (in ESM), the required storage is $O(y)$ for $n+1$-bit integers to make Table_1 and Table_2 to find all possible keys, where $y = \sum_{i=1}^{2} 2^{w(d^{(i)})-1}$.

Claim. Suppose that for some t $(1 \le t \le n-1)$ the following equations hold:

$$\sum_{j=0}^{t-1} d_j^{(1)} 2^j = \sum_{j=0}^{t-1} d_j^{(2)} 2^j \quad \textit{and} \quad \sum_{j=t}^{n} d_j^{(1)} 2^j = \sum_{j=t}^{n} d_j^{(2)} 2^j \tag{1}$$

Then required storage is reduced to $O(y')$ with $y' = \sum_{i=1}^{2} (2^{w(d^{(i)}(0,t-1))} + 2^{w(d^{(i)}(t,n))-1})$. In particular, if $w(d^{(1)}) \approx w(d^{(2)})$ and $w(d^{(i)}(0,t-1)) \approx w(d^{(i)}(t,n))$ for $i = 1, 2$, then $O(y') = O(\sqrt{y})$.

In addition, this claim can be enhanced as follows:

Claim. Suppose that for some $t_1, \cdots, t_u$ with $1 \le t_1 \le \cdots \le t_u \le n-1$ we have

$$\sum_{j=0}^{t_1-1} d_j^{(1)} 2^j = \cdots = \sum_{j=0}^{t_1-1} d_j^{(m)} 2^j, \sum_{j=t_1}^{t_2-1} d_j^{(1)} 2^j = \cdots = \sum_{j=t_1}^{t_2-1} d_j^{(m)} 2^j, \ldots, \sum_{j=t_u}^{n} d_j^{(1)} 2^j = \cdots = \sum_{j=t_u}^{n} d_j^{(m)} 2^j$$

and $w(d^{(1)}) \approx \cdots \approx w(d^{(m)})$, $w(d^{(i)}(0,t_1-1)) \approx \cdots \approx w(d^{(i)}(t_u,n))$ for $1 \le i \le m$, then the required storage is reduced to $O(y') = O((u+1) \cdot m \cdot \sqrt[u+1]{2y})$, where $y = \sum_{i=1}^{m} 2^{w(d^{(i)})-1}$. In other words, MC is approximately $(u+1)$-th root of the original.

In the next section, we propose a proposition and a theorem. The proposition shows that when the condition such as (1) is satisfied. Also, the theorem gives a novel formula which indicates the accurate TC of ESM only with obtained random recoded numbers.

3.2 Relations among BSD Representations for the Same Integer

We propose several important relations among BSD representations for the same integer which are used to reduce MC of ESM and to determine accurate TC of ESM. In this section, we keep assuming Assumption 1. Note that $w(d^{(i)}) = n + 1 - \#J^{(i)}$.

First, we propose a relation among BSD representations for the same integer, which is used to reduce MC of ESM.

Proposition 1. *Suppose* $d^{(1)} = d^{(2)}$. *For* $t \in N' - (J^{(1)} \cup J^{(2)})$, *we have*

$$d^{(1)}(0,t-1) = d^{(2)}(0,t-1) \textit{ and } d^{(1)}(t,n) = d^{(2)}(t,n).$$

Proof. $d^{(1)} = d^{(1)}(0, t-1) + d^{(1)}(t, n)$ and $d^{(2)} = d^{(2)}(0, t-1) + d^{(2)}(t, n)$. Without loss of generality (WLOG), assume $d^{(1)}(0, t-1) > d^{(2)}(0, t-1)$. As $d^{(1)} = d^{(2)}$,

$$d^{(1)}(0, t-1) - d^{(2)}(0, t-1) = d^{(2)}(t, n) - d^{(1)}(t, n) \tag{2}$$

Then the maximum of left hand side (LHS) of (2) is $2^{t+1} - 2$ and 2^{t+1} divides right hand side (RHS) of (2). It's a contradiction. Therefore, $d^{(1)}(0, t-1) = d^{(2)}(0, t-1)$ and $d^{(1)}(t, n) = d^{(2)}(t, n)$. □

From Proposition 1, the condition (1) described in Section 3.1 is satisfied when $d_t^{(1)}, d_t^{(2)} \neq 0$, i.e., $t \notin J^{(1)} \cup J^{(2)}$. Proposition 1 is generalized to the following Corollary 1.

Corollary 1. *Suppose* $d^{(1)} = \cdots = d^{(m)}$. *For* $t_1, t_2 \in N' - \cup_{i=1}^{m} J^{(i)}$ *with* $t_1 < t_2$, *we have*

$$\begin{aligned} d^{(1)}(0, t_1 - 1) &= \cdots = d^{(m)}(0, t_1 - 1), \\ d^{(1)}(t_1, t_2 - 1) &= \cdots = d^{(m)}(t_1, t_2 - 1), \text{ and} \\ d^{(1)}(t_2, n) &= \cdots = d^{(m)}(t_2, n). \end{aligned}$$

Note that as an integer $d^{(i)}$ can be determined by the non-zero digits, the number of all possible integers generated from $d^{(i)}$ is $2^{n+1-\#J^{(i)}} (= 2^{w(d^{(i)})})$. Namely, we could consider $d^{(i)}$ as a variable which has $2^{n+1-\#J^{(i)}}$ integers.

Now, we estimate the search space l for the secret key when $d^{(1)} = \cdots = d^{(m)}$ are utilized. Note that due to the space limitation we only give the sketch of proof for following theorem which is contained in Appendix A.

Theorem 1. *Suppose* $d^{(i)}$ *is the i-th random recoded number. Then the search space* l *is as follows:*

$$l = \#[\cap_{i=1}^{m} Table_i] = 2^{n+1-\#\{\cup_{i=1}^{m} J^{(i)}\}} = 2^x,$$

where $x = \#\{j \mid d_j^{(1)}, \ldots, d_j^{(m)} \neq 0,\ 0 \leq j \leq n\}$.

3.3 Proposed Generic Attack

In general, ECADD has a different power consumption pattern than ECDBL. Since ECADD and ECSUB only differ slightly, they can be implemented in such a way that they are indistinguishable for an attacker. Thus we assume that the attacker has the following capability:

Assumption 2: *ECADD and ECDBL are distinguishable by a single measurement of power consumption, whereas ECADD and ECSUB are indistinguishable.*

Assumption 3: Scalar multiplication dP is computed by using Addition-Subtraction_Method. Here, d is a random recoded number generated from k by a random recoding method. An attacker has the known plaintext-ciphertext pair (P, kP).

Remark 1. Assumption 1 described in Section 3.1 can be implied from above Assumption 2, 3. In Section 2.2, we described the practicality of Assumption 2, thus Assumption 1 is also practical.

On the other hand, we should note that Addition-Subtraction_Method in Section 2.1 has the following property.

Property 1. Suppose the variable Q is not the point at infinity. Then the recoded digit d_i is not zero if and only if ECADD or ECSUB is performed. That is to say, the recoded digit d_i is zero if and only if ECADD and ECSUB are not performed, only ECDBL is performed.

But we should not overlook the fact that such special cases of ECDBL and ECADD as $Q = 2 * \mathcal{O}$ or $Q = P + \mathcal{O}$ can be avoided in the implementation of scalar multiplication kP. In the ordinary implementation, instead of ECDBL or ECADD operation, the point duplication or assignment can be used.

Property 2. If ECDBL appears firstly in the AD sequence, then the previous bit is one.

For simplicity, ECADD and ECDBL are referred to as **A** and **D**, respectively. **A** and **D** are written with time-increasing from left to right.

Attack Algorithm (Advanced version of ESM). The concrete attack works as follows.

1. **AD sequence collection step:** The attacker inputs an elliptic curve point into a cryptographic device with a random recoding method, for instance Ha-Moon's method [HaM02] or Ebeid-Hasan's method [EH03] and so on, and obtains a sequence of **A** and **D** (AD sequence). He/she repeats this procedure m times and gathers m AD sequences.[2] Let $S^{(i)}$ be the i-th AD sequence $(1 \leq i \leq m)$.
2. **Data conversion and $J^{(i)}$ determination step:** As **A** and **D** are written with time-increasing from left to right, the attacker converts the obtained AD sequence $S^{(i)}$ into the signed-scalar number $d^{(i)}$ and determines $J^{(i)}$ as follows.
   - For $i = 1$ to m do

   2.1. Split the obtained AD sequence $S^{(i)}$ by symbol | between **D** and **DA** from right to left.
   2.2. Match **D** $\Leftrightarrow$ the random recoded digit $d_j^{(i)} = 0$.
   Add index j in $J^{(i)}$.
   2.3. Match **DA** $\Leftrightarrow$ the random recoded digit $d_j^{(i)} \neq 0$.
   2.4. If the last **D** (or **DA**) appears in the obtained AD sequence then the random recoded digits $d_j^{(i)} = 0$ (or $d_j^{(i)} \neq 0$) and $d_{j+1}^{(i)} = 1$.

[2] For some elliptic curve schemes, to gather plural AD sequences may be impossible, like the signature generation of ECDSA. However, some other schemes like ECDH are possible.

3. **All possible keys finding step:** The attacker finds all possible keys using Corollary 1.
   3.1 Find $\cup_{i=1}^{m} J^{(i)}$ ($\subset N$). Let $\mathcal{A} := N - \cup_{i=1}^{m} J^{(i)}$.
   3.2 By using Corollary 1 split obtained m signed-scalar numbers $(d^{(1)}, \ldots, d^{(m)})$ into a number of sub-signed-scalar numbers, such as $(d^{(1)}(t_v, t_{v+1}), \ldots, d^{(m)}(t_v, t_{v+1}))$.
   Here, (t_v, t_{v+1}) should satisfy following conditions.
   * For any $j \in \{t_v + 1, \ldots, t_{v+1}\}$, $j \notin \mathcal{A}$ and $t_v, t_{v+1}+1 \in \mathcal{A}$.
   * When $t_v = 0$, t_v may not be an element of $\mathcal{A}$.
   * When $t_{v+1} = n$, $t_{v+1} + 1$ need not be considered.

   Note that the sub-signed-scalar numbers $(d^{(1)}(t_v, t_{v+1}), \ldots, d^{(m)}(t_v, t_{v+1}))$ are all same integer, i.e., $d^{(1)}(t_v, t_{v+1}) = \cdots = d^{(m)}(t_v, t_{v+1})$ by Corollary 1.
   3.3 For $i = 1$ to m do
   For any (t_v, t_{v+1}) which satisfies above conditions, make $\mathrm{Table}_i^{(t_v, t_{v+1})}$.
   3.4 For any (t_v, t_{v+1}), find $\cap_{i=1}^{m} \mathrm{Table}_i^{(t_v, t_{v+1})}$.
4. **Key testing step:** Using the known pair of plaintext and ciphertext, the attacker checks all combinations of bit-pattern which are obtained from Step 3.4. Then he/she finds the secret key.

3.4 Example

We will give an example to illustrate the above-mentioned attack against Ebeid-Hasan's method [EH03].

Step 1. AD sequence collection: Assume that the attacker obtains the following AD sequences for a 15-bit secret key k ($= (23862)_{10}$):

$$S^{(1)} = DDADDDDADDADADDDADDAD$$
$$S^{(2)} = DADDDADDADDADDADDADAD$$
$$S^{(3)} = DADADDADADADDDADADDDADADAD$$

Step 2. Data conversion and $J^{(i)}$ determination: The attacker converts the obtained AD sequences $S^{(i)}$ into the signed-scalar number $d^{(i)}$ in accordance with Step 2 of the proposed attack algorithm:

$$d^{(1)} = (d_{15}^{(1)}, d_{14}^{(1)}, ..., d_0^{(1)}) = (1, 0, *, 0, 0, 0, *, 0, *, *, 0, 0, *, 0, *, 0),$$
$$d^{(2)} = (d_{14}^{(2)}, d_{13}^{(2)}, ..., d_0^{(2)}) = (1, *, 0, 0, *, 0, *, 0, *, 0, *, 0, *, *, 0),$$
$$d^{(3)} = (d_{15}^{(3)}, d_{14}^{(3)}, ..., d_0^{(3)}) = (1, *, *, 0, *, *, *, 0, *, *, 0, 0, *, *, *, 0),$$

where $*$ denotes a non-zero digit, namely 1 or -1.

$$J^{(1)} = \{0, 2, 4, 5, 8, 10, 11, 12, 14\}, \qquad J^{(2)} = \{0, 3, 5, 7, 9, 11, 12, 15\},$$
$$J^{(3)} = \{0, 4, 5, 8, 12\}.$$

Step 3. All possible keys finding:

– $\cup_{i=1}^{3} J^{(i)} = \{0, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14, 15\}$. Thus $\mathcal{A} = \{1, 6, 13\}$.
– Split obtained 3 signed-scalar numbers $(d^{(1)}, d^{(2)}, d^{(3)})$ into following three sub-signed-scalar numbers:
  ($d^{(1)}(1,5)$, $d^{(2)}(1,5)$, $d^{(3)}(1,5)$), ($d^{(1)}(6,12)$, $d^{(2)}(6,12)$, $d^{(3)}(6,12)$), ($d^{(1)}(13,15)$, $d^{(2)}(13,15)$, $d^{(3)}(13,15)$).
  Note that the sub-signed-scalar numbers ($d^{(1)}(0,0)$, $d^{(2)}(0,0)$, $d^{(3)}(0,0)$) is trivially determined as $0 \in \cap_{i=1}^{3} J^{(i)}$, i.e., $d_0^{(1)} = d_0^{(2)} = d_0^{(3)} = 0$.
– Find $\mathrm{Table}_i^{(t_v, t_{v+1})}$. Refer to Table 1.
– $\cap_{i=1}^{3} \mathrm{Table}_i^{(1,4)} = \{10, -10\}$, $\cap_{i=1}^{3} \mathrm{Table}_i^{(6,11)} = \{704, -704\}$, $\cap_{i=1}^{3} \mathrm{Table}_i^{(13,15)} = \{24576\}$.

Table 1. $\mathrm{Table}_i^{(t_v, t_{v+1})}$ Note that, for instance, $d^{(1)}(1,4)$ denotes $d_3^{(1)} \cdot 2^3 + d_1^{(1)} \cdot 2^1$ as $d_4^{(1)} = d_2^{(1)} = 0$.

$\mathrm{Table}_1^{(1,4)}$		$\mathrm{Table}_2^{(1,4)}$		$\mathrm{Table}_3^{(1,4)}$	
$(d_3^{(1)}, d_1^{(1)})$	$d^{(1)}(1,4)$	$(d_4^{(2)}, d_2^{(2)}, d_1^{(2)})$	$d^{(2)}(1,4)$	$(d_3^{(3)}, d_2^{(3)}, d_1^{(3)})$	$d^{(3)}(1,4)$
(1,1)	**10**	(1,-1,-1)	**10**	(1,1,-1)	**10**
(-1,-1)	**-10**	(-1,1,1)	**-10**	(-1,-1,1)	**-10**
$\mathrm{Table}_1^{(6,11)}$		$\mathrm{Table}_2^{(6,11)}$		$\mathrm{Table}_3^{(6,11)}$	
$(d_6^{(1)}, d_7^{(1)}, d_9^{(1)})$	$d^{(1)}(6,11)$	$(d_6^{(2)}, d_8^{(2)}, d_{10}^{(2)})$	$d^{(2)}(6,11)$	$(d_6^{(3)}, d_7^{(3)}, d_9^{(3)}, d_{10}^{(3)}, d_{11}^{(3)})$	$d^{(3)}(6,11)$
(1,1,1)	**704**	(1,-1,-1)	**704**	(1,-1,-1,1,1)	**704**
(-1,-1,-1)	**-704**	(-1,1,1)	**-704**	(-1,1,1,-1,-1)	**-704**
$\mathrm{Table}_1^{(13,15)}$		$\mathrm{Table}_2^{(13,15)}$		$\mathrm{Table}_3^{(13,15)}$	
$d_{13}^{(1)}$	$d^{(1)}(13,15)$	$d_{13}^{(2)}$	$d^{(2)}(13,15)$	$(d_{13}^{(3)}, d_{14}^{(3)})$	$d^{(3)}(13,15)$
-1	**24576**	1	**24576**	(-1,1)	**24576**

Table 2. Implementation results of TC and MC for standard 163-bit keys with m AD sequences (These results do not depend on the elliptic curve and the underlying field.)

# of AD seq. used	Ha-Moon's Algo. [HaM02]		Ebeid-Hasan's Algo. [EH03]		Agagliate's Algo. [AGO03]	
	TC	MC	TC	MC	TC	MC
5	$O(2^{27})$	$O(2^{7})$	$O(2^{50})$	$O(2^{5})$	$O(2^{54})$	$O(2^{2})$
10	$O(2^{15})$	$O(2^{14})$	$O(2^{33})$	$O(2^{8})$	$O(2^{50})$	$O(2^{3})$
15	$O(2^{10})$	$O(2^{19})$	$O(2^{25})$	$O(2^{10})$	$O(2^{46})$	$O(2^{4})$
20	$O(2^{8})$	$O(2^{23})$	$O(2^{21})$	$O(2^{12})$	$O(2^{40})$	$O(2^{6})$
30	$O(2^{6})$	$O(2^{30})$	$O(2^{16})$	$O(2^{15})$	$O(2^{33})$	$O(2^{8})$
40	$O(2^{4})$	$O(2^{37})$	$O(2^{13})$	$O(2^{17})$	$O(2^{33})$	$O(2^{8})$

Step 4. Key testing: There are 4 possible keys $(25290)_{10}$, $(25270)_{10}$, $(23882)_{10}$, and $(23862)_{10}$. Note that, for instance, $(25290)_{10} = (24576)_{10} + (704)_{10} + (10)_{10}$. The true secret key k can be easily checked by using the known pair of plaintext and ciphertext. In fact, the secret key k was $(23862)_{10}$.

Remark 2. As $\#\{j \mid d_j^{(1)}, d_j^{(2)}, d_j^{(3)} \neq 0\} = \#\{1, 6, 13\} = 3$ and MSB is 1, there exists 4 $(= 2^{3-1})$ possible keys by Theorem 1.

Table 3. The average time for detecting 163-bit keys with m AD sequences

	Number of AD sequences used	Performance (sec)
Ha-Moon's Algo. [HaM02]	15	239.52
Ebeid-Hasan's Algo. [EH03]	40	457.12

Remark 3. In this example, the proposed attack algorithm requires only 76 memory for 15-bit integers. However, if ESM is applied to this example, ESM requires 1216 ($= 2^6 + 2^7 + 2^{10}$) memory for 15-bit integers.

3.5 Implementation Result

We experimentally estimated the computational cost and the memory requirement according to the number of AD sequences for standard 163-bit secret values. Table 2 shows the time complexity and the memory complexity. Regarding Ha-Moon's countermeasure [HaM02], the proposed attack revealed the 163-bit secret key with $O(2^8)$ computational cost and $O(2^{23})$ memory requirement using 20 observations of the scalar multiplication. In the case of Ebeid-Hasan's countermeasure [EH03], the computational cost and the memory requirement are respectively $O(2^{21})$ and $O(2^{12})$ using 20 observations. In the case of Agagliate's countermeasure [AGO03], they are $O(2^{40})$ and $O(2^6)$ using 20 observations, and the use of 40 observations reduces the computational cost to $O(2^{33})$ with $O(2^8)$ memory requirement. The result of TC and MC depend on m, the number of AD sequences, and the target algorithm. If m is increased, however, TC is always decreasing and MC is always increasing, independent of the target algorithm. So, determining the reasonable m to implement the attack algorithm depends on the target algorithm. From our implementation result, 15 AD sequences are reasonable in Ha-Moon's algorithm [HaM02] and 40 AD sequences in Ebeid-Hasan's one [EH03]. The methods for finding TC and MC are contained in Appendix B.

We have implemented the proposed attack algorithm on typical microprocessors: Pentium IV/2GHz (32-bit μP; Windows 2000, MSVC). Table 3 indicates the average time for detecting a randomly chosen 163-bit keys with 15 AD sequences for Ha-Moon's algorithm [HaM02] and with 40 AD sequences for Ebeid-Hasan's one [EH03]. For instance, the average time for detecting 163-bit keys with 40 AD sequences for Ebeid-Hasan's algorithm was about 457.12 seconds. In the case of Agagliate's one [AGO03], we need to test about 2^{33} possible keys to detect 163-bit keys with 40 AD sequences. It may take approximately 3.8 years. This is not directly implemented result, but it is calculated from (required time per one scalar multiplication)$*$TC. However, these are quite fast for an attack algorithm because it takes thousands of years to find out a secret value by using a direct-computational attack against the elliptic curve discrete logarithm problem. Thus the results of Table 2, 3 show that essentially all BSD methods are vulnerable to the proposed attack which lies within practical bounds.

References

[AGO03] Agagliate, S., Guillot, P., Orcière, O., *A Randomized Efficient Algorithm for DPA Secure Implementation of Elliptic Curve Cryptosystems*, in the proceedings of Workshop on Coding and Cryptography 2003 (WCC 2003), (2003), 11-19.

[CMO98] Cohen, H., Miyaji, A., Ono, T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), 51-65.

[Cor99] Coron, J.S., *Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems*, Cryptographic Hardware and Embedded Systems (CHES'99), LNCS1717, (1999), 292-302.

[EH03] Ebeid, N., Hasan, A., *Analysis of DPA Countermeasures Based on Randomizing the Binary Algorithm*, Technical Report of the University of Waterloo, No. CORR 2003-14.
`http://www.cacr.math.uwaterloo.ca/techreports/2003/corr2003-14.ps`

[HaM02] Ha, J., and Moon, S., *Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks*, Workshop on Cryptographic Hardware and Embedded Systems 2002 (CHES 2002), LNCS 2523, (2002), 551-563.

[HCJ+03] Han, D.-G., Chang, N.S., Jung, S.W., Park, Y.-H., Kim, C.H., Ryu, H., *Cryptanalysis of the Full version Randomized Addition-Subtraction Chains*, The 8th Australasian Conference in Information Security and Privacy (ACISP 2003), LNCS2727, (2003), 67-78.

[KJJ99] Kocher, C., Jaffe, J., Jun, B., *Differential Power Analysis*, Advances in Cryptology - CRYPTO '99, LNCS1666, (1999), 388-397.

[KW03] Karlof, C., Wagner, D., *Hidden Markov Model Cryptanalysis*, Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS2779, (2003), 17-34.

[OA01] Oswald, E., Aigner, M., *Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks*, Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS2162, (2001), 39-50.

[OH03] Okeya, K., Han, D.-G., *Side Channel Attack on Ha-Moon's Countermeasure of Randomized Signed Scalar Multiplication*, INDOCRYPT 2003, LNCS2904, (2003), 334-348.

[OS02] Okeya, K., Sakurai, K., *On Insecurity of the Side Channel Attack Countermeasure using Addition-Subtraction Chains under Distinguishability between Addition and Doubling*, The 7th Australasian Conference in Information Security and Privacy, (ACISP 2002), LNCS2384, (2002), 420-435.

[OS03] Okeya, K., Sakurai, K., *A Multiple Power Analysis Breaks the Advanced Version of the Randomized Addition-Subtraction Chains Countermeasure against Side Channel Attacks*, in the proceedings of 2003 IEEE Information Theory Workshop (ITW 2003), (2003), 175-178.

[Wal03] Walter, C.D., *Security Constraints on the Oswald-Aigner Exponentiation Algorithm*, International Association for Cryptologic Research (IACR), Cryptology ePrint Archive 2003/013, (2003).
`http://eprint.iacr.org/2003/013/`

Appendix

A Proof of Theorem 1

Notations: Let S, S_1, and S_2 be the set $\{0,1,...,s\}$, $\{0,1,...,s-1\}$, and $\{0,1,...,s-2\}$, respectively, i.e., $S_2 \subset S_1 \subset S$. Let

$$Table'_i = \{k' \mid k' = \sum_{j \in S - J^{(i)'}} d_j^{(i)'} 2^j \ with\ d_j^{(i)'} \in \{1,-1\}\},$$
$$Table^*_i = \{k' \mid k' = \sum_{j \in S_1 - J^{(i)*}} d_j^{(i)^*} 2^j \ with\ d_j^{(i)^*} \in \{1,-1\}\}.$$

Proof of Theorem 1

Proof. We may assume $\cap_{i=1}^m J^{(i)} = \emptyset$ by Corollary 1. We argue by induction on n. When $n = 1$ is clear. Suppose when $n < s$, the assertion is true.

We must prove the assertion is true when $n = s$. That is, suppose $d^{(i)'}$ is the i-th random recoded number with $s+1$-bit length and $\cap_{i=1}^m J^{(i)'} = \emptyset$, where $J^{(i)'} \subset S$, for $i = 1, \ldots, m$ then

$$\#[\cap_{i=1}^m \mathrm{Table}'_i] = 2^{s+1-\#\{\cup_{i=1}^m J^{(i)'}\}} \cdots (*).$$

1. When $s \notin J^{(i)'}$ for every $1 \le i \le m$.
   Since $d_s^{(1)'} \cdot 2^s + d^{(1)'}(0, s-1) = \cdots = d_s^{(m)'} \cdot 2^s + d^{(m)'}(0, s-1)$,

   $$|d_s^{(i)'} - d_s^{(r)'}| \cdot 2^s = |d^{(i)'}(0, s-1) - d^{(r)'}(0, s-1)|$$

   for $1 \le i \ne r \le m$. As $d_s^{(i)'} = \pm 1$ for $1 \le i \le m$, LHS $= 0$ or 2^{s+1}, and $0 \le$ RHS $\le 2^{s+1} - 2$. So, $d_s^{(1)'} = \cdots = d_s^{(m)'} = \pm 1$, and $d^{(1)'}(0, s-1) = \cdots = d^{(m)'}(0, s-1)$. Therefore,

   $$LHS\ of\ (*) = 2 \cdot \#[\cap_{i=1}^m \mathrm{Table}_i'^{(0,s-1)}] = 2 \cdot 2^{s-\#\{\cup_{i=1}^m J^{(i)'}\}} = 2^{s+1-\#\{\cup_{i=1}^m J^{(i)'}\}}.$$
   (*By inductive assumption.*)

2. When $s \in J^{(i)'}$ for some $1 \le i \le m$, by re-indexing if necessary, we assume $d_s^{(i)'} = 0$ if $1 \le i \le r (< m)$, and $d_s^{(i)'} \ne 0$ if $r < i \le m$.
   (1) When $s - 1 \notin J^{(i)'}$ for any $1 \le i \le m$, i.e., $d_{s-1}^{(i)'} = \pm 1$.
   Then when $1 \le i \le r$, $J^{(i)'} = J^{(i)^*} \cup \{s\}$ for some $J^{(i)^*} \subset S_2$ and when $r+1 \le i \le m$, $J^{(i)'} \subset S_2$. Let $J^{(i)^*} = J^{(i)'}$ for $r+1 \le i \le m$.

   $$LHS\ of\ (*) = \#[\cap_{i=1}^m \mathrm{Table}'_i] = \#[\cap_{i=1}^m \mathrm{Table}^*_i]$$
   $$= 2^{s-\#\{(\cup_{i=1}^r J^{(i)^*}) \cup (\cup_{i=r+1}^m J^{(i)'})\}} = 2^{s+1-\#\{\cup_{i=1}^m J^{(i)'}\}}.$$

   (2) When $s - 1 \in J^{(i)'}$ for some $1 \le i \le m$.

(2.1) When $s-1 \in J^{(i)'}$ for some $r < i \le m$, by re-indexing if necessary, we may assume that $d_{s-1}^{(i)'} = 0$ if $r+1 \le i \le r_1 (\le m)$ for some r_1, and $d_{s-1}^{(i)'} \neq 0$ if $r_1 < i \le m$.Then when $1 \le i \le r$, $J^{(i)'} = J^{(i)^*} \cup \{s\}$ for some $J^{(i)^*} \subset S_2$, when $r+1 \le i \le r_1$, $J^{(i)'} \subset S_1$ with $s-1 \in J^{(i)'}$, and when $r_1+1 \le i \le m$, $J^{(i)'} \subset S_2$. Let $J^{(i)^*} = J^{(i)'}$ for $r+1 \le i \le m$.

$$LHS\ of\ (*) = \#[\cap_{i=1}^{m} \mathrm{Table}_i'] = \#[\cap_{i=1}^{m} \mathrm{Table}_i^*]$$
$$= 2^{s-\#\{(\cup_{i=1}^{r} J^{(i)^*}) \cup (\cup_{i=r+1}^{m} J^{(i)'})\}} = 2^{s+1-\#\{\cup_{i=1}^{m} J^{(i)'}\}}.$$

(2.2) When $s-1 \in J^{(i)'}$ for some $1 \le i \le r$, by re-indexing if necessary, we may assume that $d_{s-1}^{(i)'} = 0$ if $1 \le i \le r_1 (\le r)$ for some r_1, and $d_{s-1}^{(i)'} \neq 0$ if $r_1 < i \le r$.Then when $1 \le i \le r_1$, $J^{(i)'} = J^{(i)^*} \cup \{s\}$ for some $J^{(i)^*} \subset S_1$ with $s-1 \in J^{(i)^*}$, when $r_1+1 \le i \le r$, $J^{(i)'} = J^{(i)^*} \cup \{s\}$ for some $J^{(i)^*} \subset S_2$, and when $r+1 \le i \le m$, $J^{(i)'} \subset S_2$. Let $J^{(i)^*} = J^{(i)'}$ for $r+1 \le i \le m$.

$$LHS\ of\ (*) = \#[\cap_{i=1}^{m} \mathrm{Table}_i'] = \#[\cap_{i=1}^{m} \mathrm{Table}_i^*]$$
$$= 2^{s-\#\{(\cup_{i=1}^{r} J^{(i)^*}) \cup (\cup_{i=r+1}^{m} J^{(i)'})\}} = 2^{s+1-\#\{\cup_{i=1}^{m} J^{(i)'}\}}.$$

□

B Method for Finding TC and MC in Table 2

For instance, the time complexity and the memory complexity in the case of standard 163-bit key with 40 AD sequences in Table 2 for Ebeid-Hasan algorithm are obtained as follows:

- For $l = 1$ to 100000 do
  - Select a 163-bit string randomly.
  - Obtain 40 AD sequences using another program that outputs characters **A** and **D** depending on the elliptic curve operations it executes while computing a scalar multiplication using the Ebeid-Hasan's randomized addition chains method.
  - Convert AD sequence $S^{(i)}$ into the signed-scalar number $d^{(i)}$ where $1 \le i \le 40$.
  - Compute the testing number $TestNum_l$ which is needed to recover the secret key:
    $TestNum_l = 2^{\#\{\ j\ |\ d_j^{(i)} \neq 0\ for\ all\ 1 \le i \le 40,\ where\ 0 \le j \le 163\} - 1}$.
  - Compute the memory space $MemSpace_l$ which is needed to make $\mathrm{Table}_i^{(t_j, t_{j+1})}$ for $1 \le i \le 40$:
    $MemSpace_l = \sum_{i=1}^{40} \sum_{(t_j, t_{j+1})} 2^{w(d^{(i)}(t_j, t_{j+1}))}$.
    Note that t_j, t_{j+1} satisfy the conditions described in sub-step 3.2 in the attack algorithm.

– The average testing number of standard 163-bit key with 40 AD sequences $AveTestNum(163, 40)$ is computed as follows:

$$AveTestNum(163, 40) = \frac{\sum_{l=1}^{100000} TestNum_l}{100000} \approx O(2^{13}).$$

– The average memory space of standard 163-bit key with 40 AD sequences $AveMemSpace(163, 40)$ is computed as follows:

$$AveMemSpace(163, 40) = \frac{\sum_{l=1}^{100000} MemSpace_l}{100000} \approx O(2^{17}).$$