On Binary Signed Digit Representations of Integers

Nevine Ebeid and M. Anwar Hasan

Department of Electrical and Computer Engineering

and Centre for Applied Cryptographic Research,

University of Waterloo, Ontario, Canada

Abstract

Applications of signed digit representations of an integer include computer arithmetic, cryptography, and digital signal processing. An integer of length n bits can have several binary signed digit (BSD) representations and their number depends on its value and varies with its length. In this paper, we present an algorithm that calculates the exact number of BSD representations of an integer of a certain length. We formulate the integer that has the maximum number of BSD representations among all integers of the same length. We also present an algorithm to generate a random BSD representation for an integer starting from the most significant end and its modified version which generates all possible BSD representations. We show how the number of BSD representations of k increases as we prepend 0s to its binary representation.

1 Introduction

A *binary signed digit* representation of an integer $k \in [0, 2^n - 1]$ is a base-2 representation denoted by $(\kappa_n, \kappa_{n-1}, \ldots, \kappa_0)_{\text{BSD}}$ where $\kappa_i \in \{-1, 0, 1\}$. We will call the κ_is *signed bits*, or *sbits* for short, and -1 will be written as $\bar{1}$. An integer can have several BSD representations. For example, $k = (9)_{10}$ can be written as $(01001)_{\mathrm{BSD}} = (0101\bar{1})_{\mathrm{BSD}} = (1\bar{1}001)_{\mathrm{BSD}}$ among other possibilities. Among the possible BSD representations of an integer there are two unique representations: one is conventional the binary representation where there are no $\bar{1}$s, and the other is the *non-adjacent form* (NAF). The NAF of k is characterized by having no two adjacent non-zero sbits, and hence, having the number of 0s about $\frac{2}{3}n$. This representation was of particular importance in speeding up the scalar multiplication operation in elliptic-curve cryptosystems [8, 12]. Especially that in those cryptosystems, the use of negative bits does not incur any noticeable extra computations.

The NAF of an integer can be generated using different methods [8, 10, 12]. Recently, algorithms that generate a random BSD representation of an integer have been proposed [2, 5, 9]. Each of the latter algorithms is based on one of the former ones by inserting a random decision in the algorithm iterations. The original purpose of the algorithms of [5, 9] has been to provide protection against differential side-channel attacks by randomly changing the BSD representation of the secret key of elliptic curve cryptosystems. Subsequent work [4, 11] has however shown that randomly changing BSD representation of the secret key alone is not sufficient.

In this paper, we first consider a number of relevant questions concerning random BSD representations of an integer. These questions, which are not necessarily restricted to cryptographic applications of BSD representations and can be fundamentally important from the mathematical point of view, are: For an integer $k \in [0, 2^n - 1]$, what is the average number of BSD representations that are of length n or $n+1$ sbits and what is the exact number of representations? For integers in this range, which one has the maximum number of representations? The answers to these questions are presented in Section 2. Then in Section 3, an algorithm that calculates the exact number of representations of k in $\mathcal{O}(n)$ is presented.

In Section 4, we present an algorithm that generates a random BSD representation for an integer k by scanning its bits starting from the most significant end in $\mathcal{O}(n)$. Also, we provide modifications to this algorithm to obtain a BSD generation algorithm that can produce all

BSD representations of an integer k in $\mathcal{O}(3^{\lfloor \frac{n}{2} \rfloor})$ in the worst case. The latter algorithm helps us formulate the maximum number of BSD representations of an integer among all integers of length n and prove that it grows exponentially with n. We also demonstrate the effect of prepending 0s to k on the number of its BSD representations.

2 Number of Binary Signed Digit Representations

Considering the binary representation of k as one of its BSD representations, different BSD representations for k can be obtained by replacing 01 with $1\bar{1}$ and vice versa and by replacing $0\bar{1}$ with $\bar{1}1$ and vice versa [10, equations 8.4.4- and 8.4.5-]. For example if $k = (11)_{10}$ is represented in $n = 5$ bits, *i.e.*, $k = (01011)_2$, the different BSD representations for k are: 01011, $011\bar{1}1$, $0110\bar{1}$, $1\bar{1}011$, $1\bar{1}1\bar{1}1$, $1\bar{1}10\bar{1}$, $10\bar{1}\bar{1}1$, $10\bar{1}0\bar{1}$. The second representation can be obtained from the first one by replacing the second occurrence of 01 with $1\bar{1}$. The third representation can then be obtained from the second one by replacing the $\bar{1}1$ with $0\bar{1}$, and so forth. Those replacements are done exhaustively until all possible BSD representations for k are obtained.

The binary representation of k, must include at least one 0 that precedes a 1, so that starting from it we can obtain other BSD representations.

2.1 Useful Lemmas

In the following we present some lemmas related to the number of BSD representations of an integer k. These lemmas will be used to derive the main results of this paper.

Let $\lambda(k, n)$ be the number of BSD representations of $k \in [0, 2^n - 1]$ that are n sbits long. Then the following lemmas hold.

Lemma 1

(i) $\lambda(0, n) = 1$,

(ii) $\lambda(1, n) = n$,

(iii) $\lambda(2^i, n) = n - i$.

Proof.

(i) This is obvious since in two's complement representation, the value 0 is represented with n consecutive zeros. Therefore, there exists no choices for alternative representations.

Let us assume that there is some other BSD representation for the integer $k = 0 = \sum_{i=0}^{n-1} \kappa_i 2^i$ where $\kappa \in \{\bar{1}, 0, 1\}$. Then this representation must contain one or more sbits of the value 1 or $\bar{1}$. For example, let us assume that there is a representation where $\kappa_j = 1$ for some $0 \le j \le n-1$, then the summation of the remaining sbits with their appropriate weights should be -2^j. The largest absolute value that the sbits $(\kappa_{j-1}, \ldots, \kappa_0)_2$ can take is $\sum_{i=0}^{j-1} 2^i = 2^j - 1$. The smallest absolute value that is greater than 0 that the sbits $(\kappa_{n-1}, \ldots, \kappa_{j+1})_2$ can take is 2^{j+1}. The difference between these two values is $2^{j+1} - 2^j + 1 = 2^j + 1$. That is there is no possible assignment for the remaining sbits resulting in a value of -2^j.

(ii) The possible BSD representations for 1 in n sbits are $0^{n-1}1$, $0^{n-2}1\bar{1}$, $0^{n-3}1\bar{1}\bar{1}$, $\ldots$, $1\bar{1}^{n-1}$. Their total number is n.

This is true since, for any $t \in [0, n]$

$$2^t - \sum_{i=0}^{t-1} 2^i = 2^t - \frac{2^t - 1}{2 - 1} = 1.$$

(iii) The possible BSD representations for 2 in n sbits are $0^{n-2}10$, $0^{n-3}1\bar{1}0$, $0^{n-4}1\bar{1}\bar{1}0$, $\ldots$, $1\bar{1}^{n-2}0$. Note that these are the same representations for 1 when its binary representation is $(n-1)$ bits long with an added 0 as the least significant sbit. Their total number is $n - 1$.

That is,

$$\lambda(2, n) = \lambda(1, n-1) = n - 1.$$

Hence,

$$\lambda(4, n) = \lambda(2, n-1) = \lambda(1, n-2) = n - 2.$$

By induction,

$$\lambda(2^i, n) = n - i. \qquad \square$$

Lemma 2 *For* $2^{n-1} \leq k \leq 2^n - 1$,

$$\lambda(k, n) = \lambda(k - 2^{n-1}, n - 1).$$

Proof. An integer k in this range would have the binary form $(1, k_{n-2}, \ldots, k_0)_2$ and its value is $2^{n-1} + d$, where $d = (k_{n-2} \ \ldots \ k_0)_2$, and is $(n-1)$ bits long. The BSD representations for d in $n-1$ sbits are of the form $(\kappa_{n-2}, \ldots \kappa_0)_{\text{BSD}}$ with $\kappa_{n-2} \neq \overline{1}$, otherwise, d would be negative. The BSD representations of k are then of the form $(1, \kappa_{n-2}, \ldots, \kappa_0)_{\text{BSD}}$. The two most significant sbits are either 10 or 11. In both cases, no new BSD representations can be generated. Thus, k will have the same BSD representations as for d with an added 1 as the most significant sbit. $\square$

Lemma 3 *For k even,*

$$\lambda(k, n) = \lambda(\frac{k}{2}, n - 1).$$

Proof. In this case, the integer k is of the form $(k_{n-1} \ \ldots \ k_1 \ 0)_2 = 2d$ where $d = (k_{n-1} \ \ldots \ k_1)_2$, and is $(n-1)$ bits long. The BSD representations for d in $n-1$ sbits are of the form $(\kappa_{n-1}, \ldots, \kappa_1)_{\text{BSD}}$. When d is multiplied by 2 to obtain k, it is shifted left by one and a 0 is added to the least significant position. The same is done to each of its BSD representations. In all of them, the least two significant sbits will be either $\overline{1}0$, 00 or 10. In all three cases, no new BSD representations can be generated. Thus, k will have the same BSD representations as for d with an added 0 as the least significant sbit. $\square$

Lemma 4 *For k odd,*

$$\lambda(k, n) = \lambda(k - 1, n) + \lambda(k + 1, n), \tag{a}$$

or

$$\lambda(k, n) = \lambda\left(\frac{k-1}{2}, n - 1\right) + \lambda\left(\frac{k+1}{2}, n - 1\right). \tag{b}$$

Proof. There are two cases to consider.

Case 1: $k \equiv 1 \pmod 4$, that is k ends with 01, $k-1$ ends with 00 and $k+1$ ends with 10. From Lemma 3, the BSD representations for $k+1$ in n sbits are the same as those for $\frac{k+1}{2}$ with a 0 added as the least significant sbit. If this 0 is replaced by a $\overline{1}$, those representations will be possible representations for k. If we think we should start replacing the rightmost $1\overline{1}$ with 01 to generate new representations, we will find out that all representations that end with a 1 will be accounted for by considering those of $k-1$.

Also from Lemma 3, the BSD representations for $k-1$ in n sbits are the same as those for $\frac{k-1}{4}$ in $n-2$ sbits with 00 added as the least significant sbits. If the rightmost 0 is replaced with a 1, those representations will be possible representations for k. If we think we should start replacing the rightmost 01 with $1\overline{1}$ to generate new representations, we will find out that all representations that end with a $\overline{1}$ have been accounted for by considering those of $k+1$ as mentioned before.

Case 2: $k \equiv 3 \pmod 4$, that is k ends with 11, $k-1$ ends with 10 and $k+1$ ends with 00. The same argument holds as in case 1. The BSD representations of $k-1$ can be possible representations for k by replacing the least significant 0 with 1. Also the BSD representations for $k+1$ can be possible representations for k by replacing the least significant 0 with $\overline{1}$.

There are no other possible representations for k. This is obvious from the fact that any BSD representation for k should have the rightmost sbit either $\overline{1}$ or 1, it can not be 0. Otherwise, $k \equiv 0 \pmod 2$ which is not the case since k is odd. So, if the rightmost 1 or $\overline{1}$ in any representation of k is replaced with 0 then this will be one of the representations of the even number preceding or following k respectively as shown in (a). Using Lemma 3, (b) is obtained. $\square$

2.2 Number of BSD Representations of Length n

Here we will investigate the total number of BSD representations for all integers in the range $[0, 2^n - 1]$ that are n sbits long. We will denote that total number by $\sigma(n)$. Table 1 gives an example of $\lambda(k,n)$ and $\sigma(n)$ for small n.

Table 1: $\lambda(k,n)$ and $\sigma(n)$ for small n.

$n=1$		$n=2$		$n=3$		$n=4$		$n=5$		$n=5$ *cont'd*	
k	$\lambda(k,n)$	k	$\lambda(k,n)$	k	$\lambda(k,n)$	k	$\lambda(k,n)$	k	$\lambda(k,n)$	k	$\lambda(k,n)$
0	1	0	1	0	1	0	1	0	1	16	1
1	1	1	2	1	3	1	4	1	5	17	4
		2	1	2	2	2	3	2	4	18	3
		3	1	3	3	3	5	3	7	19	5
				4	1	4	2	4	3	20	2
				5	2	5	5	5	8	21	5
				6	1	6	3	6	5	22	3
				7	1	7	4	7	7	23	4
						8	1	8	2	24	1
						9	3	9	7	25	3
						10	2	10	5	26	2
						11	3	11	8	27	3
						12	1	12	3	28	1
						13	2	13	7	29	2
						14	1	14	4	30	1
						15	1	15	5	31	1
$\sigma(n)=2$		$\sigma(n)=5$		$\sigma(n)=14$		$\sigma(n)=41$				$\sigma(n)=122$	

We can intuitively find an expression for $\sigma(n)$ as follows. In BSD system, the integers k, represented in n sbits, would be in the range $-2^n < k < 2^n$. There are 3^n different combinations for k. We consider in this case non-negative integers k, *i.e.*, $0 \le k < 2^n$. In fact, $-k$ has the same BSD representations as k with the 1s replaced with $\overline{1}$s and vice versa. Thus, the total number of non-negative combinations is $\frac{3^n+1}{2}$. We will now use the previous lemmas to prove that

$$\sum_{k=0}^{2^n-1} \lambda(k,n) = \sigma(n) = \frac{3^n+1}{2}.$$

Let

$$\varepsilon(n) = \sum_{k=1}^{2^{n-1}-1} \lambda(k,n). \tag{1}$$

From Lemma 1(i) and Lemma 2, we have

$$\sigma(n) = \sigma(n-1) + \varepsilon(n) + 1. \tag{2}$$

If we substitute for $\sigma(n)$ recursively in this equation we obtain

$$\sigma(n) = \sigma(1) + \varepsilon(2) + \cdots + \varepsilon(n) + n - 1. \tag{3}$$

From Lemma 3 we have

$$\begin{aligned}\sum_{k=2,\ k \text{ even}}^{2^{n-1}-1} \lambda(k,n) &= \sum_{k=2,\ k \text{ even}}^{2^{n-1}-2} \lambda\left(\frac{k}{2}, n-1\right) = \sum_{i=1}^{2^{n-2}-1} \lambda(i, n-1) \\ &= \varepsilon(n-1). \end{aligned} \tag{4}$$

From Lemma 4 we have

$$\begin{aligned}\sum_{k=1,\ k \text{ odd}}^{2^{n-1}-1} \lambda(k,n) &= \sum_{k=1,\ k \text{ odd}}^{2^{n-1}-1} \left(\lambda\left(\frac{k-1}{2}, n-1\right) + \lambda\left(\frac{k+1}{2}, n-1\right)\right) \\ &= \sum_{i=0}^{2^{n-2}-1} \lambda(i, n-1) + \sum_{i=1}^{2^{n-2}} \lambda(i, n-1).\end{aligned}$$

From Lemma 1(iii), $\lambda(2^{n-1}, n) = 1$, and also from Lemma 1(i) we have

$$\sum_{k=1,\ k \text{ odd}}^{2^{n-1}-1} \lambda(k,n) = 2 + 2\,\varepsilon(n-1) \tag{5}$$

where the last equality follows from (4). Substituting from (4) and (5) into (1), we have

$$\begin{aligned}\varepsilon(n) &= \varepsilon(n-1) + 2 + 2\,\varepsilon(n-1) \\ &= 3\,\varepsilon(n-1) + 2.\end{aligned} \tag{6}$$

With recursive substitution for $\varepsilon(n)$ and the fact that $\varepsilon(1) = 0$, we obtain

$$\begin{aligned}\varepsilon(n) &= 3^{n-1}\varepsilon(1) + 2 + 2\cdot 3 + 2 \cdot 3^2 + \cdots + 2\cdot 3^{n-2} \\ &= 2 \cdot \frac{3^{n-1}-1}{3-1} \\ &= 3^{n-1} - 1.\end{aligned} \tag{7}$$

Finally we use (7) and the fact that $\sigma(1) = 2$ to evaluate (3)

$$\begin{aligned}\sigma(n) &= 2 + (3-1) + \cdots + (3^{n-1}-1) + n - 1 \\ &= 2 + (n-1) - (n-1) + 3 \cdot \frac{3^{n-1}-1}{3-1} \\ &= \frac{3^n + 1}{2}.\end{aligned} \tag{8}$$

2.3 Number of BSD Representations of Length $n+1$

The NAF of an integer may be one sbit longer than its binary representation [10, 12]. As mentioned before, the algorithms that generate a random BSD representation of an integer are each based on a NAF-generating algorithm [2, 5]. Therefore, we are interested in knowing the number of BSD representations of an n-bit integer k that are $n+1$ sbits long. Moreover for those integers that are in the range $[2^{n-1}, 2^n - 1]$, we will see the effect of having a 0 as the most significant bit in their binary representation on the number of their BSD representations and on the distribution of the number of representations among all n-bit integers as opposed to the previous section. The effect of prepending more 0s to the binary representation of integers on the number of their BSD representation is studied in Section 4.3.

Lemma 5 *Let $\delta(k, n)$ be the number of BSD representations of $k \in [0, 2^n - 1]$ in $n+1$ sbits. Then, we have*

$$\delta(k, n) = \lambda(k, n) + \lambda(2^n - k, n).$$

Proof. A part of the BSD representations of k that are $n+1$-sbit long are those that have a 0 as the most significant sbit and their number is $\lambda(k, n)$, as was defined in the previous section. If we change the most significant 0 in these representations to a 1, *i.e.*, add 2^n to the value of k, we should add $-(2^n - k)$ in the remaining n sbits, that is the negative of the two's complement of k. $2^n - k$ has $\lambda(2^n - k, n)$ BSD representations of length n. The negative of these representations is obtained by replacing the 1s with $\bar{1}$s and vice versa. $\square$

The same argument applies to the 2's complement of k

$$\begin{aligned} \delta(2^n - k, n) &= \lambda(2^n - k, n) + \lambda(k, n) \\ &= \delta(k, n). \end{aligned} \tag{9}$$

For $k = 0$, $\delta(0, n) = \lambda(0, n) = 1$. From (9), we conclude that, for $k \in [0, 2^n - 1]$ in the defined range, the distribution of $\delta(k, n)$ is symmetric around $k = 2^{n-1}$.

Let $\varsigma(n)$ be the total number of BSD representations of length $n+1$ for all integers $k \in [0, 2^n - 1]$, we have

$$\begin{aligned}\varsigma(n) &= \sum_{k=0}^{2^n-1} \delta(k,n) = 1 + \sum_{k=1}^{2^n-1} \delta(k,n) \\ &= 1 + \sum_{k=1}^{2^n-1} \big(\lambda(k,n) + \lambda(2^n - k, n)\big) = 1 + 2\sum_{k=1}^{2^n-1} \lambda(k,n) \\ &= 1 + 2\left(\frac{3^n+1}{2} - 1\right) \\ &= 3^n. \end{aligned} \tag{10}$$

Remark 1 *We conclude that, for any n-bit integer, the average number of its—$(n+1)$ sbits long—BSD representations is roughly* $\left(\frac{3}{2}\right)^n$

Table 2: $\delta(k,n)$ and $\varsigma(n)$ for small n.

$n=1$		$n=2$		$n=3$		$n=4$		$n=5$		$n=5$ *cont'd*	
k	$\delta(k,n)$	k	$\delta(k,n)$	k	$\delta(k,n)$	k	$\delta(k,n)$	k	$\delta(k,n)$	k	$\delta(k,n)$
0	1	0	1	0	1	0	1	0	1	16	2
1	2	1	3	1	4	1	5	1	6	17	9
		2	2	2	3	2	4	2	5	18	7
		3	3	3	5	3	7	3	9	19	12
				4	2	4	3	4	4	20	5
				5	5	5	8	5	11	21	13
				6	3	6	5	6	7	22	8
				7	4	7	7	7	10	23	11
						8	2	8	3	24	3
						9	7	9	11	25	10
						10	5	10	8	26	7
						11	8	11	13	27	11
						12	3	12	5	28	4
						13	7	13	12	29	9
						14	4	14	7	30	5
						15	5	15	9	31	6
$\varsigma(n)=3$		$\varsigma(n)=9$		$\varsigma(n)=27$		$\varsigma(n)=81$				$\varsigma(n)=243$	

Table 2 gives an example of $\delta(k,n)$ and $\varsigma(n)$ for small n. It is clear from Table 1 and Table 2 and from the definitions of $\lambda(k,n)$ and $\delta(k,n)$ that, for $0 \le k < 2^{n-1}$,

$$\lambda(k,n) = \delta(k, n-1) \tag{11}$$

since in this range, the binary representation of k has a 0 as the leftmost bit. In other words, the algorithm that computes $\lambda(k, n)$ that we present in Section 3 can be used to compute $\delta(k, n)$ as

$$\delta(k, n) = \lambda(k, n + 1), \tag{12}$$

for $0 \leq k < 2^{n-1}$.

2.4 Integer with Maximum Number of BSD Representations

It is desirable to know which integer $k \in [0, 2^n - 1]$ has the maximum number of BSD representations of length n or $n+1$ sbits. We note from (12) that the integer with the largest $\delta(k, n)$ is the same one with the largest $\lambda(k, n+1)$. Also from the symmetry of $\delta(k, n)$ around 2^{n-1} (Eq. (9)), we note that there are two values of k for which $\delta(k, n)$ is the maximum value. We will denote them as $k_{max_1,n}$ and $k_{max_2,n}$. From (9) we have $k_{max_2,n} = 2^n - k_{max_1,n}$. In the following lemma we will consider only $k_{max_1,n}$ and drop the suffix 1.

Lemma 6

$$\delta(k_{max,n}, n) = \delta(k_{max,n-1}, n - 1) + \delta(k_{max,n-2}, n - 2)$$

Proof. We will prove this lemma by induction. From Table 2, we see that the lemma is true for $n = 3$. Now we assume that it is true up to an arbitrary $n = i - 1$. That is, using (12) we can write

$$\lambda(k_{max,i-1}, i) = \lambda(k_{max,i-2}, i - 1) + \lambda(k_{max,i-3}, i - 2) \tag{13}$$

Also, from Lemma 4, we know that

$$\lambda(k_{max,i-1}, i) = \lambda\left(\frac{k_{max,i-1} - 1}{2}, i - 1\right) + \lambda\left(\frac{k_{max,i-1} + 1}{2}, i - 1\right). \tag{14}$$

From Lemma 3 and Lemma 4, $k_{max,i}$ must be an odd integer. Let k be an i-bit odd integer, then $k = \left(\frac{k-1}{2}\right) + \left(\frac{k+1}{2}\right)$. Obviously, one of these two terms is an odd integer and the other is the preceding or following even integer. We will denote those two integers as k_o and k_e, respectively. From Lemma 4, we have

$$\lambda(k, i + 1) = \lambda(k_o, i) + \lambda(k_e, i).$$

For k to be equal $k_{max,i}$, one or both of the following conditions must be true:

- $k_o = k_{max,i-1}$, and $k_e \equiv 2 \pmod 4$ (Lemma 3 and 4).
- k_e has the maximum number of representations among all even integers. From Lemma 3, this is equivalent to saying that $k_e = 2 \cdot k_{max,i-2}$.

We have four cases for the value of an odd k modulo 8. In each case, if the first condition is verified, we will prove that the second condition is also verified, which will prove the lemma.

Case 1: $k \equiv 1 \pmod 8$

$\Rightarrow k_e = \frac{k-1}{2} \equiv 0 \pmod 4$.

This violates the first condition.

Case 2: $k \equiv 3 \pmod 8$

$\Rightarrow k_o = \frac{k-1}{2} \equiv 1 \pmod 4 \Rightarrow k_e = \frac{k+1}{2} \equiv 2 \pmod 4$.

Assume $k_o = k_{max,i-1}$

$\Rightarrow \frac{k_{max,i-1}-1}{2} \equiv 0 \pmod 2$ (even).

Hence, from (13) and (14), we have

$$k_{max,i-2} = \frac{k_{max,i-1}+1}{2},$$

$$2 \cdot k_{max,i-2} = k_{max,i-1} + 1 = k_e. \tag{15}$$

That is, the second condition is verified.

Case 3: $k \equiv 5 \pmod 8$

$\Rightarrow k_e = \frac{k-1}{2} \equiv 2 \pmod 4 \Rightarrow k_o = \frac{k+1}{2} \equiv 3 \pmod 4$.

Assume $k_o = k_{max,i-1}$

$\Rightarrow \frac{k_{max,i-1}-1}{2} \equiv 1 \pmod 2$ (odd).

Hence, from (13) and (14), we have

$$k_{max,i-2} = \frac{k_{max,i-1}-1}{2},$$

$$2 \cdot k_{max,i-2} = k_{max,i-1} - 1 = k_e. \tag{16}$$

That is, the second condition is verified.

Case 4: $k \equiv 7 \pmod 8$

$\Rightarrow \frac{k-1}{2} \equiv 3 \pmod 4$ (odd) $\Rightarrow k_e = \frac{k+1}{2} \equiv 0 \pmod 4$.

This violates the first condition. $\square$

From the proof of Lemma 6, we can deduce the following corollary.

Corollary 1

$$k_{max,n} = k_{max,n-1} + 2 \cdot k_{max,n-2}.$$

Now, we will derive a formula for $k_{max_1,n}$ and $k_{max_2,n}$. From the proof of Lemma 6, we see that when $k_{max_1,n} \equiv 3 \pmod 4$ (Case 2), $k_{max_1,n-1} \equiv 1 \pmod 4$ and

$$k_{max_1,n} = 2 \cdot k_{max_1,n-1} + 1, \tag{17}$$

and that when $k_{max_1,n} \equiv 1 \pmod 4$ (Case 3), $k_{max_1,n-1} \equiv 3 \pmod 4$ and

$$k_{max_1,n} = 2 \cdot k_{max_1,n-1} - 1. \tag{18}$$

Thus, we see that with every increment of n ($n > 2$), cases 2 and 3 alternate. For $n = 3$, from Table 2 we have $k_{max_1,3} = 3 \equiv 3 \pmod 4$. Hence, Case 2 occurs when n is odd and Case 3 occurs when n is even.

For n even, we can substitute from (17) into (18) to obtain

$$\begin{aligned} k_{max_1,n} &= 2\ k_{max_1,n-1} - 1 \\ &= 2\ (2\ k_{max_1,n-2} + 1) - 1 \\ &= 4\ k_{max_1,n-2} + 1. \end{aligned} \tag{19}$$

Using recursive substitution,

$$\begin{aligned} k_{max_1,n} &= 4 \cdot 4 \cdot 4 \cdots k_{max_1,2} + 1 + 4 + 16 + \cdots \\ &= (2^2)^{\frac{n-2}{2}} \cdot 1 + \frac{(2^2)^{\frac{n-2}{2}} - 1}{2^2 - 1} \\ &= \frac{1}{3}(2^n - 1). \end{aligned} \tag{20}$$

For n odd, we can follow the same derivation procedure to obtain

$$k_{max_1,n} = \frac{1}{3}(2^n + 1). \tag{21}$$

As for $k_{max_2,n}$, for n even we have

$$\begin{aligned} k_{max_2,n} &= 2^n - \frac{1}{3}(2^n - 1) \\ &= \frac{1}{3}(2^{n+1} + 1) \\ &= k_{max_1,n+1} \end{aligned} \tag{22}$$

where the last equality follows from (21).

Similarly, for n odd, we have

$$\begin{aligned} k_{max_2,n} &= \frac{1}{3}(2^{n+1} - 1) \\ &= k_{max_1,n+1}. \end{aligned} \tag{23}$$

The sbit pattern of $k_{max_1,n}$ for even n and for odd n can be deduced from the previous discussion. Let $\mathcal{S}$ be a string. The notation $\langle\mathcal{S}\rangle^d$ denotes $\mathcal{S}$ repeated d times. For example, $(\langle 0\ 1\rangle^3)_2$ is $(0\ 1\ 0\ 1\ 0\ 1)_2$. From (19) we can deduce that for n even, $k_{max_1,n}$ is of the form $(\langle 0\ 1\rangle^{\frac{n}{2}})_2$ and hence from (18) for n odd, $k_{max_1,n}$ is of the form $(\langle 0\ 1\rangle^{\frac{n-1}{2}}1)_2$.

Thus, after specifying the binary structure of the integer with maximum number of BSD representations, we will use it as the worst-case input to our left-to-right generation algorithm in Section 4.2. We will hence derive an expression for the number of BSD representations of that integer.

3 Algorithm to Compute the Number of BSD Representations for an Integer

In this section, we will present an algorithm that computes $\lambda(k, n)$ for any integer $k \in [0, 2^n - 1]$. This algorithm is based on the lemmas presented in Section 2.1.

Algorithm 1. Number of BSD representations of an integer k in n sbits

```
INPUT: $k \in [0, 2^n - 1]$, $n$
OUTPUT: $C = \lambda(k, n)$
external $\lambda 2(k_e, k_o, w_e, w_o, n)$ /*computed by Algorithm 2 that follows*/

  1. if $(k = 0)$ then
         $C \leftarrow 1$

  2. else if $(k = 1)$ then
         $C \leftarrow n$

  3. else if $(k \geq 2^{n-1})$ then
         $C \leftarrow \lambda(k - 2^{n-1}, n - 1)$

  4. else if ($k$ is even) then
         $C \leftarrow \lambda(\frac{k}{2}, n - 1)$

  5. else

      5.1 if $(k \equiv 1 \pmod 4)$ then
              $C \leftarrow \lambda 2(\frac{k-1}{2}, \frac{k+1}{2}, 1, 1, n - 1)$

      5.2 else
              $C \leftarrow \lambda 2(\frac{k+1}{2}, \frac{k-1}{2}, 1, 1, n - 1)$

  6. return $C$
```

Algorithm 1 uses Lemma 1(i) and 1(ii) to return the value of $\lambda(k, n)$ directly if the value of k is either 0 or 1. Otherwise, it uses Lemmas 2 and 3 to trim k recursively from any leading 1's or trailing 0's since they don't add to the number of BSD representations of k as mentioned in the proofs of these lemmas. Then, this algorithm calls Algorithm 2 to find the actual number of BSD representations of k which is then an odd integer in the range $[0, 2^{n'-1} - 1]$, for some $n' \leq n$. Hence, Lemma 4 is applicable to this k.

Algorithm 2. Auxiliary algorithm used by Algorithm 1 to compute $\lambda(k, n)$

```
INPUT: $k_e$, $k_o$, $w_e$, $w_o$, $n$
OUTPUT: $c = \lambda 2(k_e, k_o, w_e, w_o, n)$

  1. if $(k_o = 1$ AND $k_e = 2)$ then
         $c \leftarrow n * w_o + (n - 1) * w_e$

  2. else

      2.1 if $(k_e \equiv 0 \pmod 4)$ then

          2.1.1 if $(k_o \equiv 1 \pmod 4)$ then
                    $c \leftarrow \lambda 2(\frac{k_e}{2}, \frac{k_e}{2} + 1, w_o + w_e, w_o, n - 1)$
```

```
        2.1.2 else
                $c \leftarrow \lambda 2(\frac{k_e}{2}, \frac{k_e}{2} - 1, w_o + w_e, w_o, n - 1)$
    2.2 else
        2.2.1 if ($k_o \equiv 1 \pmod 4$) then
                $c \leftarrow \lambda 2(\frac{k_e}{2} - 1, \frac{k_e}{2}, w_o, w_o + w_e, n - 1)$
        2.2.2 else
                $c \leftarrow \lambda 2(\frac{k_e}{2} + 1, \frac{k_e}{2}, w_o, w_o + w_e, n - 1)$
3. return $c$
```

Using Lemma 4, $\lambda(k, n)$ for k odd constitutes of two other evaluations of the same function λ; one is for an even integer, k_e which is the closest even integer to $k/2$, and the other is for the preceding or the following odd integer, k_o. If we start using Lemmas 3 and 4 recursively to evaluate λ for k_e and k_o respectively, at each iteration there will be always two terms for the λ function multiplied by a certain weight each, w_e and w_o.

In general, at the i^{th} iteration

$$\lambda(k, n) = w_{e,n-i}\ \lambda(k_{e,n-i}, n - i) + w_{o,n-i}\ \lambda(k_{o,n-i}, n - i).$$

At the beginning, $w_{e,n} = w_{o,n} = 1$. From Lemma 3 we have

$$\lambda(k_{e,n-i}, n - i) = \lambda\left(\frac{k_{e,n-i}}{2}, n - i - 1\right).$$

From Lemma 4 we have

$$\lambda(k_{o,n-i}, n - i) = \lambda\left(\frac{k_{o,n-i} - 1}{2}, n - i - 1\right) + \lambda\left(\frac{k_{o,n-i} + 1}{2}, n - i - 1\right).$$

There are two possible cases for k_e in each iteration and two corresponding subcases for k_o.

Case 1: $k_{e,n-i} \equiv 0 \pmod 4 \quad \Rightarrow \quad \frac{k_{e,n-i}}{2}$ is even

Case 1.1: $k_{o,n-i} \equiv 1 \pmod 4$, *i.e.*, $k_{o,n-i} = k_{e,n-i} + 1 \Rightarrow \frac{k_{o,n-i}-1}{2}$ is even

Hence,

$k_{e,n-i-1} = \frac{k_{o,n-i}-1}{2} = \frac{k_{e,n-i}}{2}$

$k_{o,n-i-1} = \frac{k_{o,n-i}+1}{2} = \frac{k_{e,n-i}}{2} + 1$

Case 1.2: $k_{o,n-i} \equiv 3 \pmod 4$, $i.e.$, $k_{o,n-i} = k_{e,n-i} - 1 \Rightarrow \frac{k_{o,n-i}-1}{2}$ is odd

Hence,

$$k_{e,n-i-1} = \tfrac{k_{o,n-i}+1}{2} = \tfrac{k_{e,n-i}}{2}$$

$$k_{o,n-i-1} = \tfrac{k_{o,n-i}-1}{2} = \tfrac{k_{e,n-i}}{2} - 1$$

In case 1, the weights are updated as follows

$$w_{e,n-i-1} = w_{o,n-i} + w_{e,n-i}$$

$$w_{o,n-i-1} = w_{o,n-i}$$

Case 2: $k_{e,n-i} \equiv 2 \pmod 4 \quad \Rightarrow \quad \frac{k_{e,n-i}}{2}$ is odd

Case 2.1: $k_{o,n-i} \equiv 1 \pmod 4$, $i.e.$, $k_{o,n-i} = k_{e,n-i} - 1$

Hence,

$$k_{e,n-i-1} = \tfrac{k_{o,n-i}-1}{2} = \tfrac{k_{e,n-i}}{2} - 1$$

$$k_{o,n-i-1} = \tfrac{k_{o,n-i}+1}{2} = \tfrac{k_{e,n-i}}{2}$$

Case 2.2: $k_{o,n-i} \equiv 3 \pmod 4$, $i.e.$, $k_{o,n-i} = k_{e,n-i} + 1$

Hence,

$$k_{e,n-i-1} = \tfrac{k_{o,n-i}+1}{2} = \tfrac{k_{e,n-i}}{2} + 1$$

$$k_{o,n-i-1} = \tfrac{k_{o,n-i}-1}{2} = \tfrac{k_{e,n-i}}{2}$$

In case 2, the weights are updated as follows

$$w_{e,n-i-1} = w_{o,n-i}$$

$$w_{o,n-i-1} = w_{o,n-i} + w_{e,n-i}$$

Following are some examples that illustrate the iterations of Algorithm 2 for some chosen values of k. In these examples we have used the notation (k) that means $\lambda(k,n)$. We have also used a tree-like representation where, for a parent node k, the number of BSD representations is equal to the sum of the number of BSD representations of its child nodes. The weight update procedure is depicted by the number of links between the parent and child nodes. The examples are chosen such that k_e and k_o that are passed as arguments to Algorithm 2, *e.g.*, 12 and 13 in Figure 1 (a), correspond to the four different cases discussed.

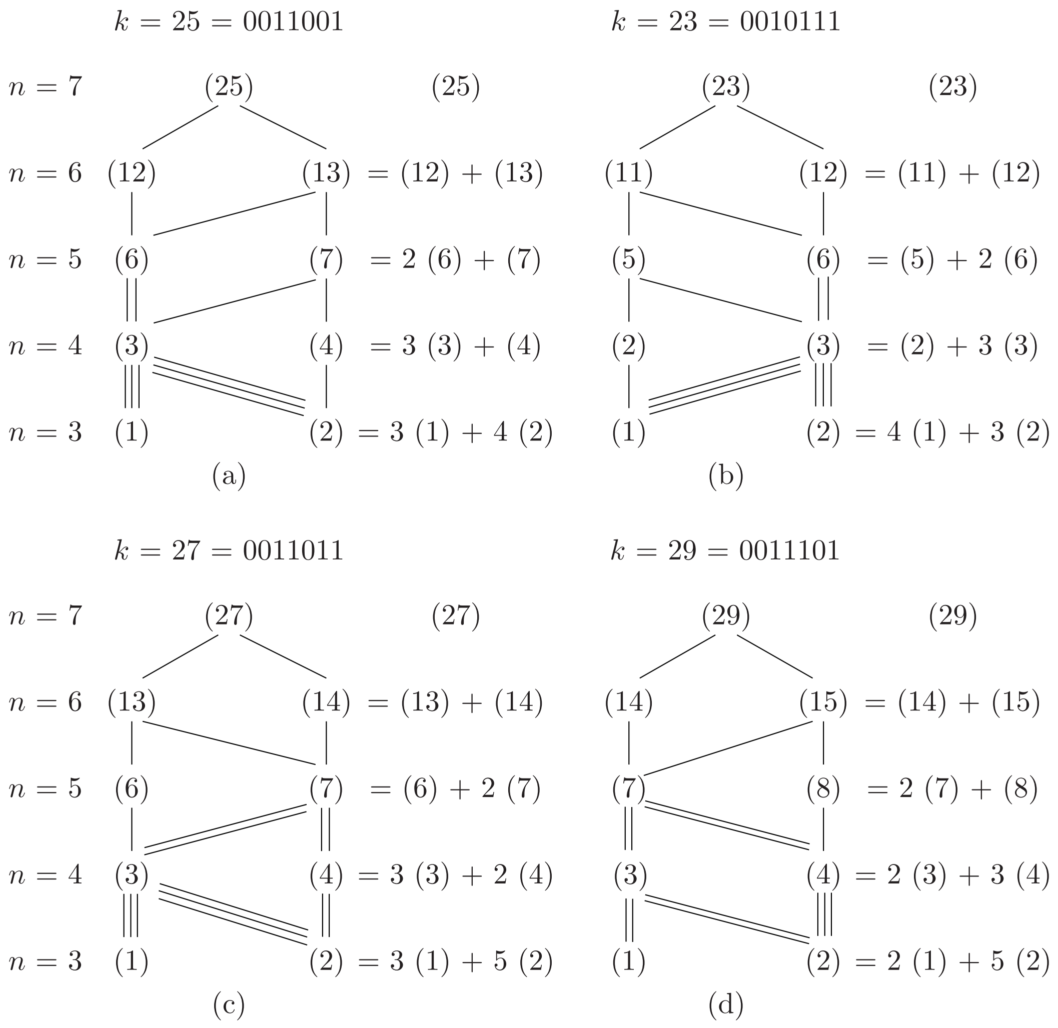

Figure 1: (a) $\lambda(25, 7)$, corresponds to case 1.1. (b) $\lambda(23, 7)$, corresponds to case 1.2. (c) $\lambda(27, 7)$, corresponds to case 2.1. (d) $\lambda(29, 7)$, corresponds to case 2.2.

For the time and space complexity of Algorithm 1—including its usage of Algorithm 2—it is clear that it runs in $O(n)$ time and occupies $O(n)$ bits in memory. This time complexity results from the fact that both algorithms deal with the integer k one bit at a time. As for the space complexity, the new values generated for k and n by Algorithm 1 and for k_e, k_o, w_e, w_o

and n by Algorithm 2 can replace the old values in memory. That is, though the algorithms are illustrated in a recursive form, they can be transformed into an iterative form.

4 Left-to-Right BSD Randomization and Generation Algorithms

The algorithms that generate a random BSD representation of an integer [2, 5, 9] scan its bits from the least significant to the most significant bit, *i.e.*, from right to left. This is because they are based on the NAF-generating algorithms [8, 10, 12] that scan the bits of the integer in the same direction.

In this section, we present an algorithm that generates a random BSD representation of k while scanning it from left to right. Then we modify it in order to generate *all* of the possible BSD representations of k. The modified algorithm helps us demonstrate the exponential growth with n of the number of BSD representations of $k_{max_1,n}$, as well as the effect of prepending 0s to any integer k.

4.1 Left-to-Right Randomization Algorithm

We start with an example. Let $k = 10001001 = 2^7 + 2^4 + 2^0$. That is k is the result of adding the following three integers:

$$\begin{aligned} k &= 10000000 \\ &+ 00001000 \\ &+ 00000001 \end{aligned}$$

We will consider the different BSD representations for each integer separately and then add those different representations together to get those of k.

The first integer has no other representation according to Lemma 1(iii). From the same lemma, the second integer has the following additional representations $0001\bar{1}000$, $001\bar{1}\bar{1}000$, $01\bar{1}\bar{1}\bar{1}000$ and $1\bar{1}\bar{1}\bar{1}\bar{1}000$. If we add this last representation to the first integer, then k would need more than 8 sbits in this example to represent it, so we will not take it into consideration. Finally the third integer has this set of additional representations $0000001\bar{1}$, $000001\overline{11}$, $00001\overline{111}\ldots$, $01\overline{111111}$. We will notice that we only need to take into account the first three representations. This is because all the representations starting from the third one when added to the first two integers will yield a 0 in sbit position 3, $\bar{1}$s in the lower positions and a sbit pattern in the upper positions that has been accounted for in the different representations of the second integer.

Also, we can consider that this representation of k

$$\begin{array}{rcl} k & = & 10000000 \\ & + & 001\overline{11}000 \\ & + & 00001\overline{111} \\ \hline & = & 101\bar{1}0\overline{111} \end{array}$$

could be obtained from this one

$$\begin{array}{rcl} k & = & 10000000 \\ & + & 001\overline{11}000 \\ & + & 000001\overline{11} \\ \hline & = & 101\overline{11}1\overline{11} \end{array}$$

after changing the $\bar{1}1$ at sbit positions 3 and 2 to $0\bar{1}$.

Thus, the underlying idea of the algorithm is that the binary representation of k is subdivided into groups of bits—of different lengths—such that each group is formed by a number of consecutive 0s ending with a single 1. For each of these groups a random BSD representation is chosen as in the proof of Lemma 1(ii). Whenever the choice yields a $\bar{1}$ at the end of a group—which happens when any representation for that group other than the binary one is chosen—and a 1 at the beginning of the next group—which happens when the representation chosen for the next group is the one that has no 0s—another random decision is taken so as whether to leave those two sbits (*i.e.*, $\bar{1}1$) as they are or to change them to $0\bar{1}$.

The choice of a random representation for a certain group is done by counting the number of 0s in it, say z, and choosing a random integer $t \in [0, z]$ which will be the number of 0s to be written to the output. If t is equal to z, the last sbit to be written to the output for this group is 1, and it is actually written before considering the next group. Otherwise, t 0s, a 1 and only $z - t - 1$ $\overline{1}$s are written to the output, that is the last $\overline{1}$ is not written, but saved as the value of a variable labeled as *last*. We then do the same for the next group. If for the next group $t = 0$, we take a random decision whether to write $\overline{1}1$ to the output or $0\overline{1}$ at the boundary of the two groups. This leads to the following algorithm. Note that a 0 is prepended to k so that the BSD representation k' generated is of length $n + 1$ sbits.

Algorithm 3. Left-to-Right Randomization of an integer's BSD representation

```
INPUT: $k = (k_{n-1} \ldots k_0)_2$
OUTPUT: $k' = (k'_n \ldots k'_0)_{\mathrm{BSD}}$, a random BSD representation of $k$

1. Set $k_n \leftarrow 0$; $i \leftarrow n + 1$; $last \leftarrow 1$

2. for $j$ from $n$ down to 0 do
     if $(k_j = 1)$ then

     2.1 $t \leftarrow_R [0, i - j - 1]$
     2.2 if $(t = 0$ AND $last = \overline{1})$ then
         2.2.1 $c \leftarrow_R \{0, 1\}$
         2.2.2 if $(c = 0)$ then
                   $k'_i \leftarrow \overline{1}$
                   $i \leftarrow i - 1$; $k'_i \leftarrow 1$
         2.2.3 else
                   $k'_i \leftarrow 0$
     2.3 else
         2.3.1 if $(last = \overline{1})$ then
                   $k'_i \leftarrow \overline{1}$
         2.3.2 while $(t > 0)$ do
                   $i \leftarrow i - 1$; $k'_i \leftarrow 0$; $t \leftarrow t - 1$
         2.3.3 $i \leftarrow i - 1$; $k'_i \leftarrow 1$
     2.4 if $(i = j)$ then
             $last \leftarrow 1$
     2.5 else
         2.5.1 while $(i > j + 1)$ do
                   $i \leftarrow i - 1$; $k'_i \leftarrow \overline{1}$
         2.5.2 $i \leftarrow i - 1$; $last \leftarrow \overline{1}$
```

```
3. if ($last = \overline{1}$) then
       $k'_i \leftarrow \overline{1}$
4. while ($i > 0$) do
       $i \leftarrow i - 1$; $k'_i \leftarrow 0$
```

We note the following about the algorithm:

- The algorithm runs in $O(n)$ time.

- The sbits of k' are written to the output one at a time in their correct order from left to right. This means that there is no need to store k' in an application where the sbits are processed from left to right as they are generated. For example, it is advantageous to perform the elliptic curve scalar multiplication from left to right, especially when a mixed (projective and affine) coordinate system is used for saving computational cost in scalar multiplication [1, 7]. If the randomization of the BSD representation of the key is needed during the scalar multiplication, then Algorithm 3 can be readily interleaved with point doubling and point addition operations. Thus, it is more beneficial than Ha-Moon's algorithm [5], where the generated representation is first stored in order to use it in the scalar multiplication. Note that a BSD representation would probably need twice the storage required for the binary representation, if each sbit is internally represented by two bits.

- The minimum and/or the maximum number of 0s allowed in each group of the resulting BSD representation can be set by changing the range from which t is randomly chosen in step 2.1, *e.g.*, the minimum value can be some fraction of $z = i - j - 1$. This is interesting if it is desired to keep the Hamming weight of the representation low. Moreover, in step 2(.2).1, a bias could be given to choosing 1 more than 0 in order to make it more likely to choose $0\overline{1}$ at the group boundaries than $\overline{1}1$.

- When there is a group with a long run of 0s in k, considering the possible random representation for that group, we can see that the most significant bit of that group will be 0 with high probability among the resulting representations. This is also the case for a long run of 1s. Though each of those 1s forms a group, the randomization at the boundary of the group ensures that the most significant bit of that run is also more likely to be 0. That is $01111 = 1\bar{1}111 = 10\bar{1}11 = 100\bar{1}1 = 1000\bar{1}$. This agrees with the observation of Fouque *et al.* [4] on their right-to-left attack that, after a long run of 0s or 1s, the probability of the sbit being 0 becomes close to 1.

- The number of representations of a group is the length of that group. Assume that an n-bit integer k has all groups of the same length $l \geq 2$ (since for $l = 1$, k can be considered as having one group as in the previous note). If we don't consider the randomization at the group boundaries, then the lower bound on the number of representations of k, $\lambda(k, n)$, is $(l)^{\frac{n}{l}}$. If we consider the boundary randomization as one more representation of the group—except for the last group, then the upper bound on $\lambda(k, n)$ is $(l+1)^{\frac{n}{l}-1} l \approx (l + 1)^{\frac{n}{l}}$. The actual number is closer to the upper boundary than the lower one. The upper boundary strictly decreases with l. We conclude that an integer with more groups has a larger number of BSD representations than another integer of the same length with fewer groups. As mentioned in the previous note, a run of 1s can be considered as one group. Thus, the former conclusion is equivalent to saying that an integer with a better random distribution of its bits has more representations than another integer with longer runs of 0s or 1s. In Section 4.2, we will derive an expression for the number of BSD representations when $l = 2$, which will be of the order of the upper boundary.

4.2 Left-to-Right Generation Algorithm

The algorithm presented here is a modified version of Algorithm 3 that recursively and exhaustively substitutes every group of 0s ending with a 1 with one of its possible forms. It also takes into consideration the alternative substitutions at the group boundary when the representation of a group ends with a $\bar{1}$ and that of the next group starts with 1. This algorithm can be used as a straight-forward check that Algorithm 3 is capable of generating any possible—$(n+1)$ sbits long—BSD representation of an integer k in the range $[1, 2^n - 1]$, *i.e.*, there is no BSD representation of k that cannot be generated by that algorithm. It was tested on small values of n.

Algorithm 4. Left-to-Right Generation of all BSD representations of k

```
INPUT: $k = (k_{n-1} \ \ldots \ k_0)_2$
OUTPUT: all possible strings $k' = (k'_n \ \ldots \ k'_0)_{\mathrm{BSD}}$

  1. Subdivide $k$ from left to right into groups of consecutive 0s each ending with a single 1. Store
     the length of each group in a look-up table $G$. Let $g$ be the index of the table.

  2. Set $g \leftarrow 0$;  $i \leftarrow n+1$;
     $last \leftarrow 1$;  $j \leftarrow i - G[g]$;  $k' \leftarrow \langle\rangle$

  3. for $t = 0$ to $i - j - 1$ do
         ChooseForm$(k, g, t, i, j, last, k')$
```

Algorithm 5. ChooseForm$(k, g, t, i, j, last, k')$, a recursive procedure employed by Algorithm 4

```
INPUT: $k$, $g$, $t$, $i$, $j$, $last$, $k'$
OUTPUT: returns $k'$, a string of sbits, as a possible BSD representation of $k$.

  1. if $(t > 0)$ OR $(last = 1)$ then      //this step is equivalent to step 2.3 in Algorithm 3

       1.1 if $(last = \bar{1})$ then
               $k' \leftarrow k'|\bar{1}$                 //concatenate $k'$ with $\bar{1}$

       1.2 while $(t > 0)$ do
               $i \leftarrow i - 1$;  $k' \leftarrow k'|0$;  $t \leftarrow t - 1$

       1.3 $i \leftarrow i - 1$;  $k' \leftarrow k'|1$

  2. if $(i = j)$ then
         $last \leftarrow 1$

  3. else

       3.1 while $(i > j + 1)$ do
               $i \leftarrow i - 1$;  $k \leftarrow k'|\bar{1}$

       3.2 $i \leftarrow i - 1$;  $last \leftarrow \bar{1}$

  4. if $(j = 0)$ then

       4.1 if $(last = \bar{1})$ then
               $k' \leftarrow k'|\bar{1}$

       4.2 return $k'$

  5. $g \leftarrow g + 1$;  $j \leftarrow j - G[g]$

  6. if $(j = 0)$ AND ($k$ is even) then

       6.1 if $(i > 0)$ then
```

```
            6.1.1 if ($last = \overline{1}$) then
                      $k' \leftarrow k'|\overline{1}$
            6.1.2 while ($i > 0$) do
                      $i \leftarrow i - 1;\ k' \leftarrow k'|0$
        6.2 return $k'$

   7. $t = 0$

   8. if ($last = \overline{1}$) then
          ChooseForm($k, g, t, i-1, j, last, k'|\overline{1}1$)
          ChooseForm($k, g, t, i, j, last, k'|0$)

   9. else
          ChooseForm($k, g, t, i, j, last, k'$)

  10. for $t = 1$ to $i - j - 1$ do
          ChooseForm($k, g, t, i, j, last, k'$)
```

To better explain how this algorithm works we present in Figure 2 the tree explored by the algorithm for $k = 21$ and $n = 5$. This tree is explored by the algorithm in a *depth-first* fashion. That is, the recursive function *ChooseForm* is first called from Algorithm 4 at node a in the figure. Then this function calls itself at node b and then at node c where it returns with the first BSD representation for $k = 21$ which is $1\overline{1}1\overline{1}1\overline{1}$. With the flow control at node b the function calls itself at node d where the second BSD representation is generated and so forth.

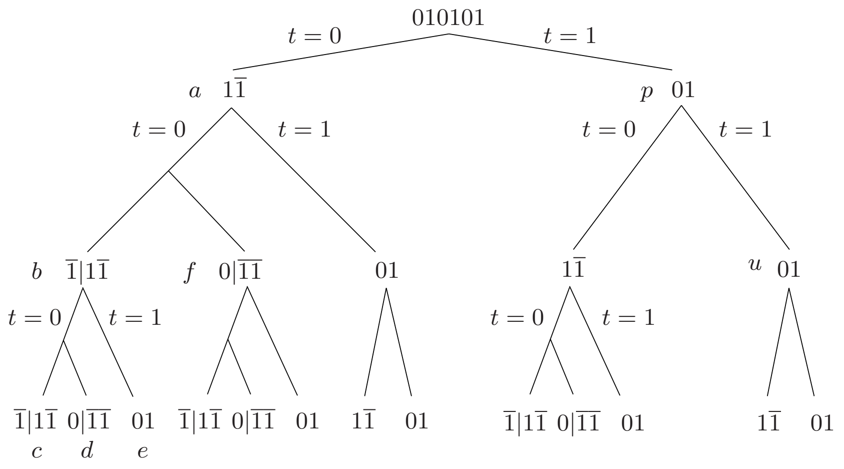
010101
$t = 0$
$t = 1$
a $1\overline{1}$
p 01
$t = 0$
$t = 1$
$t = 0$
$t = 1$
b $\overline{1}|1\overline{1}$
f $0|\overline{1}\overline{1}$
01
$1\overline{1}$
u 01
$t = 0$
$t = 1$
$t = 0$
$t = 1$
$\overline{1}|1\overline{1}$ $0|\overline{1}\overline{1}$ 01 $\overline{1}|1\overline{1}$ $0|\overline{1}\overline{1}$ 01 $1\overline{1}$ 01 $\overline{1}|1\overline{1}$ $0|\overline{1}\overline{1}$ 01 $1\overline{1}$ 01
c d e

Figure 2: The tree explored by Algoritm 4 for $k = 21$ and $n = 5$.

The algorithm works as follows. The integer k is first subdivided into groups of bits where each group consists of consecutive 0s and a single 1 at the end as for Algorithm 3. Starting from the leftmost group, a particular BSD representation for that group is chosen starting with the one that contains no 0s (*i.e.*, $t = 0$, where t is the number of 0s to be written to the output for that group as before). The representation is formed inside the function *ChooseForm* which takes t as one of its arguments. In turn, this function goes through the possible values of t for the following group and, for each one, calls itself to form the corresponding BSD representation of that group. When t is equal to 0, the two possible alternatives at the group boundary are considered as was explained in Section 4.1. For example, in Figure 2, the last sbit in the group at node a may remain $\bar{1}$ or change to 0 depending on the random decision taken at the group boundary when $t = 0$ for the next group. This is why this last sbit is written at nodes b and f before the symbol '|' which designates the boundary between the groups.

The *worst-case* complexity analysis of Algorithm 4 is presented in the following. The worst case occurs for the integer with the maximum number of BSD representations in the range $[1, 2^n - 1]$. There are actually two integers with this property for any given n, which we referred to as $k_{max_1,n}$ and $k_{max_2,n}$ in Section 2.4. We mentioned that, for n even, $k_{max_1,n}$ is of the form $(\langle 0\ 1\rangle^{\frac{n}{2}})_2$. For example, $k_{max_1,6} = 21 = (010101)_2$ (see Figure 3). We also mentioned that, for any n, $k_{max_2,n} = k_{max_1,n+1}$. For example, $k_{max_1,5} = 11 = (01011)_2$ and $k_{max_2,5} = 21 = (10101)_2$ (see Figure 3). Therefore, our analysis is conducted on those integers k of the binary form $(\langle 0\ 1\rangle^{\frac{n}{2}})_2$ for n even and $(1\langle 0\ 1\rangle^{\frac{n-1}{2}})_2$ for n odd. In the following discussion, we will drop the subscript n from $k_{max_1,n}$ and $k_{max_2,n}$ for simplicity, since it will be obvious from the context. For n even, we have the following.

$n = 2$: $k_{max_1} = k_{max_2} = 1 = (01)_2$,

$\delta(1, 2) = \lambda(1, 3) = 3.$

The tree explored for this integer is the same as the one having as root the node b in Figure 2. The difference is that in this case t can take the values 0, 1 and 2 with only one representation for $t = 0$.

$n = 4$: $k_{max_1} = 5 = (0101)_2$,

$\delta(5, 4) = \lambda(5, 5) = 8 = 3 \cdot 3 - 1$.

The tree explored for this integer is the same as the one having as root a in Figure 2.

$n = 6$: $k_{max_1} = 21 = (010101)_2$,

$\delta(21, 6) = \lambda(21, 7) = 21 = 3 \cdot 3 \cdot 3 - (3.1 + 3)$.

The tree explored for this integer is illustrated in Figure 3.

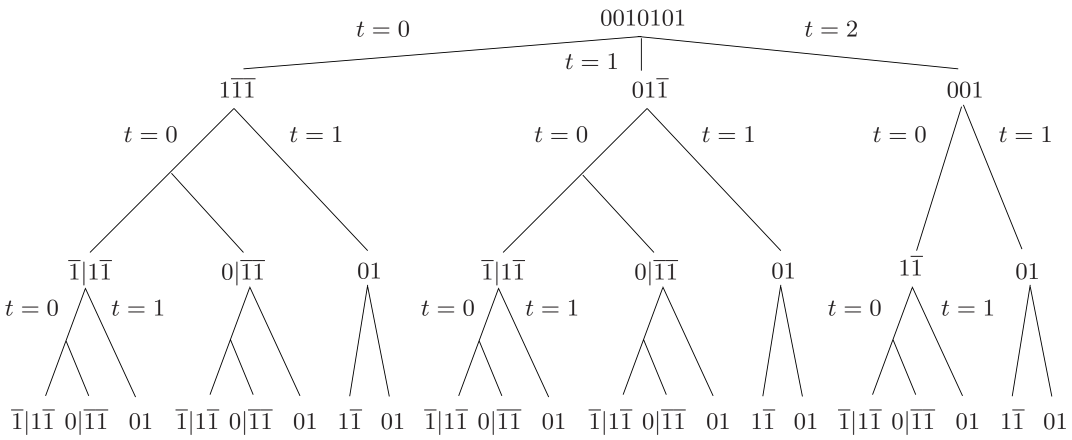

Figure 3: The tree explored by Algoritm 4 for $k = 21$ and $n = 6$.

Let $m = \frac{n}{2}$. By induction we can deduce the following

$$\lambda(k_{max_1}, n+1) = 3^m - (m-1)3^{m-2} + \left(\sum_{i_1=1}^{m-3} i_1\right) 3^{m-4} - \left(\sum_{i_1=1}^{m-5}\sum_{i_2=1}^{i_1} i_2\right) 3^{m-6} + \left(\sum_{i_1=1}^{m-7}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2} i_3\right) 3^{m-8} - \cdots . \tag{24}$$

For n odd, we have the following.

$n = 1$: $k_{max_1} = k_{max_2} = 1 = (1)_2$,

$\delta(1, 1) = \lambda(1, 2) = 2$.

The tree explored for this integer is simply the same as the one having as root the node u in Figure 2.

$n = 3$: $k_{max_2} = 5 = (101)_2$,

$\delta(5, 3) = \lambda(5, 4) = 5 = 2 \cdot 3 - 1$.

The tree explored for this integer is the same as the one having as root the node p in Figure 2.

$n = 5$: $k_{max_2} = 21 = (10101)_2$,

$\delta(21, 5) = \lambda(21, 6) = 13 = 2 \cdot 3 \cdot 3 - (3 + 2 \cdot 1)$.

The tree for this integer is the one illustrated in Figure 2.

Let $m = \frac{n-1}{2}$. By induction we can deduce the following

$$\begin{aligned}\lambda(k_{max_2}, n+1) = 2 \cdot 3^m &- \left[3^{m-1} + 2(m-1)3^{m-2}\right] \\ &+ \left[(m-2)3^{m-3} + 2\left(\sum_{i_1=1}^{m-3} i_1\right)3^{m-4}\right] \\ &- \left[\left(\sum_{i_2=1}^{m-4} i_2\right)3^{m-5} + 2\left(\sum_{i_1=1}^{m-5}\sum_{i_2=1}^{i_1} i_2\right)3^{m-6}\right] \\ &+ \left[\left(\sum_{i_2=1}^{m-6}\sum_{i_3=1}^{i_2} i_3\right)3^{m-7} + 2\left(\sum_{i_1=1}^{m-7}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2} i_3\right)3^{m-8}\right] \\ &- \cdots\end{aligned} \tag{25}$$

From this discussion, we state the following theorem.

Theorem 1 *For any n, the number of BSD representations generated by Algorithm 4 is, in the worst case, $\mathcal{O}(3^{\lfloor \frac{n}{2} \rfloor})$.*

4.3 Effect of Prepending 0s to k on the Number of its BSD Representations

In this section, we show how the number of BSD representations of k increases if we lengthen its binary representation by adding 0s at the most significant end.

If we compare Figure 2 with Figure 3, we see that for the same integer $k = 21$, increasing n from 5 to 6 had the effect of increasing the number of branches emerging from the root by one. The added branch has the same tree structure as the leftmost branch a in Figure 2. This is because the number of BSD representations of the first group—recall how the integer is subdivided into groups—has increased by one. Since all representations of a group, except for the original binary representation, end with a $\bar{1}$, the added representation would generate two alternatives when, for the next group, $t = 0$. If we increase n to 7, another subtree like the one having as root a will be added to the tree. The same subtree is repeated with every 0 prepended to the binary representation of k. It is easy to verify that this is true for any integer k.

As was mentioned before, the subtree having as root the node a is the tree explored for $k = 5$ and $n = 4$. In general, the subtree that is repeated is the one formed for the integer with the binary representation having the same groups as k except for the leftmost group, *i.e.*, with the most significant 1 removed. This integer can be expressed as $k - 2^{\lfloor \log_2 k \rfloor}$ for any k that is not a power of 2. The representation of this integer should have only 2 prepended 0 in order to have 3 branches at the root node, this means that the length of the representation should be $\lfloor \log_2 k \rfloor + 1$. That is

$$\begin{aligned} \Delta(k) &= \lambda(k, n+1) - \lambda(k, n), \\ &= \lambda(k, n+i+1) - \lambda(k, n+i) \quad \text{for any } i \geq 0, \\ &= \lambda(k - 2^{\lfloor \log_2 k \rfloor}, \lfloor \log_2 k \rfloor + 1). \end{aligned} \tag{26}$$

where $\Delta(k)$ is the number of leaves in the repeated subtree. Based on (26), we have the following theorem.

Theorem 2 *If Δn is the number of 0s prepended to the binary representation of an integer k, then the number of its BSD representations increases by $\Delta n \cdot \Delta(k)$.*

5 Experimental Results

In this section, we present experimental results related to the speed and usage of Algorithm 3. We consider an application where we need to choose long integers, *e.g.*, $n = 160$, having a large number of BSD representations, *e.g.*, more than 2^{40}.

First, we have shown in Section 3 that Algorithm 1 runs in $\mathcal{O}(n)$. For instance, for $n = 160$, the integer that has the maximum number of BSD representations is $k_{max_1,160} = (\langle 0\ 1\rangle^{80})_2$ and the number of these representations is

$$\delta(k_{max_1,160}, 160) = (9E449CF5F9D5F28B6248B9097ED8)_{hex}$$

. This result was computed in approximately $0.38\mu s$ on a 1.5 GHz Pentium M processor (Centrino technology), using the BN (big numbers)library[1]. This is an indication of how fast this algorithm can be executed.

In Theorem 1, we have proven that $\delta(k_{max_1,n}, n)$ is $\mathcal{O}(3^{\lfloor \frac{n}{2} \rfloor})$. As was mentioned in the example above, for $n = 160$, this number is a 111-bit integer. Using this information, one can proceed as follows in order to choose an integer with a large number of BSD representations:

Step 1: Specify the minimum number of BSD representations that an integer should have, that is a selection threshold T.

Step 2: Choose at random an integer k of length n.

Step 3: Use that integer as an input to Algorithm 1 to compute the number of its BSD representations, $\delta(k, n)$.

Step 4: If $\delta(k, n) \geq T$, then accept the integer k. Otherwise, reject it and go to Step 2.

From *the theory of runs* [6, Sec. 2.7], if k is an n-bit integer with $\frac{n}{2}$ 0s and $\frac{n}{2}$ 1s, then the most probable number of runs in k is between $\frac{n}{2}$ and $\frac{n}{2} + 3$, that is the probability that k would have several long runs of 0s and 1s—and hence fewer runs—is small. Moreover, from the same theory, we can calculate the probability of having h runs for such an integer for any h. For example, for $n = 160$, an integer with 80 0s and 80 1s has the following probabilities

[1]Provided by Eric Young as part of his implementation of SSL, known as *openssl*. Available from `http://www.openssl.org/source/openssl-0.9.7d.tar.gz`

of having h runs.

$$Pr(h = 20) = 9 \times 10^{-25},$$

$$Pr(h = 40) = 1.7 \times 10^{-11}.$$

Based on this theory and the last note in Section 4.1, it is expected that the percentage of integers that would be rejected by the above selection procedure is negligible for large n. We can illustrate this fact experimentally as follows.

Let $r = \log(\delta(k, n))/\log(\delta(k_{max_1,n}, n))$. For $n = 160$, let the selection threshold be $T = 2^{40}$, then we would like to know the percentage of integers in the range $[0, 2^n - 1]$ having $r < 40/111 = 0.36036$. For large n, it is computationally infeasible to calculate this percentage in a deterministic way. However, for small n ($n < 32$), our experiments show that this percentage is negligible and is strictly decreasing as n increases. For example, this percentage is 0.00132%, 0.00093% and 0.00065% for $n = 29, 30$ and 31, respectively. If we set $T = 2^{80}$, then the percentage of rejected integers, *i.e.*, those having $r < 80/111 = 0.72072$ is 10.09%, 9.62% and 9.17% for $n = 29, 30$ and 31, respectively.

6 Conclusion

In this paper, we have presented some interesting issues related to the number of binary-signed digit (BSD) representations of an integer $k \in [0, 2^n - 1]$, such as the average number of representations among integers of the same length and the bit patterns of $k_{max_1,n}$ and $k_{max_2,n}$, *i.e.*, the integers of length n bits that have the maximum number of BSD representations.

We have presented an algorithm that calculates in $\mathcal{O}(n)$ the exact number of BSD representations of k that are of length n sbits, and have illustrated the algorithm's efficiency for $n = 160$, which is of interest for elliptic curve cryptographic applications. We have also presented an algorithm that generates in $\mathcal{O}(n)$ a random BSD representation of k by scanning its bits starting from the most significant end, and outputs the sbits in their correct order one at a time. In addition, we have presented an algorithm that can generate all BSD representations of an integer, which has helped us prove that the number of representations of $k_{max_1,n}$ is $\mathcal{O}(3^{\lfloor \frac{n}{2} \rfloor})$. We have also proven that prepending 0s to the binary representation of an integer results in only a linear increase in the number of its BSD representations.

We have also presented some experimental results that show that the percentage of integers of a certain length having a relatively small number of representations is negligible.

Acknowledgement

The authors thank the reviewers for their useful comments to improve the paper. The authors also thank S. Brown and M. S. Brown for brining Ref. [6] to their attention. A preliminary version of this work was presented at SAC 2003 [3]. This work has been supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) Post Graduate Scholarship - Doctorate, as well as by OCIPEP (Office of Critical Infrastructure Protection and Emergency Preparedness) and RIM (Research in Motion) through an NSERC strategic project.

References

[1] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *LNCS*, pages 51–65. Springer-Verlag, 1998. 22

[2] N. Ebeid and A. Hasan. Analysis of DPA countermeasures based on randomizing the binary algorithm. CACR Technical Reports CORR 2003-14, University of Waterloo, 2003. 2, 9, 19

[3] N. Ebeid and M. A. Hasan. On randomizing private keys to counteract DPA attacks. In *Selected Areas in Cryptography – SAC '03*, volume 3006 of *LNCS*, pages 58–72. Springer-Verlag, 2003. 32

[4] P.-A. Fouque, F. Muller, G. Poupard, and F. Valette. Defeating countermeasures based

on randomized bsd representations. In *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 of *LNCS*, pages 312–327. Springer-Verlag, 2004. 2, 23

[5] J. Ha and S. Moon. Randomized signed-scalar multiplication of ECC to resist power attacks. In *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 of *LNCS*, pages 551–563. Springer-Verlag, 2002. 2, 9, 19, 22

[6] J. G. Kalbfleisch. *Probability and Statistical Inference. Volume 1: Probability.* Springer-Verlag, 1985. 30, 32

[7] J. López and R. Dahab. Improved algorithms for elliptic curve arithmetic in GF(2^n) without precomputation. In *Selected Areas in Cryptography – SAC '98*, volume 1556 of *LNCS*, pages 201–212. Springer-Verlag, 1999. 22

[8] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Informatique théorique et Applications/Theoretical Informatics and Applications*, 24(6):531–544, 1990. 2, 19

[9] E. Oswald and M. Aigner. Randomized addition-subtraction chains as a countermeasure against power attacks. In *Cryptographic Hardware and Embedded Systems – CHES '01*, volume 2162 of *LNCS*, pages 39–50. Springer-Verlag, 2001. 2, 19

[10] G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960. 2, 3, 9, 19

[11] S. G. Sim, D. J. Park, and P. J. Lee. New power analysis on the Ha-Moon algorithm and the MIST algorithm. In *Information and Communications Security – ICICS '04*, volume 3269 of *LNCS*, pages 291–304. Springer-Verlag, 2004. 2

[12] J. A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19:195–249, 2000. 2, 9, 19