Complexity Analysis of a Fast Modular Multiexponentiation Algorithm

Haimin Jin[1,2], Duncan S. Wong[2⋆], Yinlong Xu[1]

[1] Department of Computer Science
University of Science and Technology of China
China
jhm1213@mail.ustc.edu.cn, ylxu@ustc.edu.cn
[2] Department of Computer Science
City University of Hong Kong
Hong Kong, China
duncan@cityu.edu.hk

Abstract. Recently, a fast modular multiexponentiation algorithm for computing $A^X B^Y \pmod N$ was proposed [15]. The authors claimed that on average their algorithm only requires to perform $1.306k$ modular multiplications (MMs), where k is the bit length of the exponents. This claimed performance is significantly better than all other comparable algorithms, where the best known result by other algorithms achieves $1.503k$ MMs only. In this paper, we give a formal complexity analysis and show the claimed performance is not true. The actual computational complexity of the algorithm should be $1.556k$. This means that the best known modular multiexponentiation algorithm based on canonical-sighed-digit technique is still not able to overcome the $1.5k$ barrier.

Keywords: modular multi-exponentiation, modular arithmetic, canonic signed-digit representation, Hamming weight, Markov chain.

1 Introduction

Modular multi-exponentiation is an arithmetic operation that on input integers $(x_1, \cdots, x_l)$, $(e_1, \cdots, e_l)$ and n, computes $\prod_{i=1}^{l} x_i^{e_i} \pmod n$ for $l > 1$. This operation has been used in many number-theoretic cryptosystems [4,12,2,14] and the efficient implementation of this operation is very important to the performance of those cryptosystems as multi-exponentiation is one of the most expensive operations for them.

Fast algorithms for performing modular multiexponentiation are especially important when $l = 2$, that is, given integers A, B, X, Y and N, compute $C = A^X B^Y \pmod N$. Digital Signature Algorithm (DSA) [7], ElGamal digital signature scheme [4], Schnorr's signature scheme [12], Camenisch-Shoup public key encryption scheme [2], Waters' identity-base encryption [14] and many other cryptographic algorithms

⋆ The work was supported by a grant from CityU (Project No. 7002001).

require to perform modular multiexponentiations for $l = 2$. However, modular multi-exponentiation is generally a very time-consuming arithmetic operation. To compute $C = A^X B^Y \pmod N$, the traditional method is to solve the modular exponentiations of $A^X \pmod N$ and $B^Y \pmod N$ individually first and then multiply them together (followed by a modular reduction). This method is not optimal, even if one uses an optimal algorithm for the computation of $A^X \pmod N$ and $B^Y \pmod N$. Any improvement on the computational complexity of this operation will immediately result in improving the performance of many cryptographic algorithms and protocols.

There have been many algorithms proposed [4,11,3,13] for performing fast modular multiexponentiation. Among them, Canonic Signed-Digit (CSD) representation technique has been commonly used. In [3], Dimitrov, Jullien and Miller proposed two algorithms, which we call them as DJM-I and DJM-II. The expected number of Modular Multiplications (MMs) of these algorithms are $1.75k$ and $1.556k$, respectively, where k is the larger bit length of the two exponents X and Y. Recently, Wu, Lou, Lai and Chang [15] proposed an improved algorithm (which is termed **WLLC** in our paper) of DJM-II by introducing a complement representation method into the original DJM-II algorithm. They also claimed that the expected number of MMs can be significantly reduced to $1.306k$. Table 1 shows the performance of various modular multiexponentiation algorithms.

Multi-Exponentiation Algorithms	**No. of Modular Multiplications (MMs)**
Traditional computation	$2.000k$
ElGamal's method [4]	$1.750k$
Yen-Laih-Lenstra [11]	$1.625k$
DJM-I [3]	$1.750k$
DJM-II [3]	$1.556k$
DJM-II with heuristic further reduction [3]	$1.534k$
Solinas algorithm [13]	$1.503k$
WLLC [15]	$1.306k$ (to be corrected)

Table 1. Complexity comparison among various modular multi-exponentiation algorithms when $l = 2$. k is the larger bit length of the two exponents.

Our Results. We show that the expected number of MMs that the WLLC algorithm [15] can achieve is actually $1.556k$ rather than $1.306k$. This result is shown by providing a formal complexity analysis supported with experimental results. This also indicates that the complement representation method introduced in WLLC does not improve the efficiency of DJM-II, while it was originally thought to be in [15]. As of independent interest, we attain a useful theorem which says that the CSD (Canonic Signed-Digit) binary representation of any k-bit integers with Hamming

weight at most $\frac{k}{2}$ has the expected fraction of bits that are zero is about $\frac{2}{3}$ which is interestingly the same if we consider all k-bit integers.

Paper Organization. In the next section, we define several useful mathematical terms. This is followed by the review of WLLC and an informal comparison with DJM-II in Sec. 3. In Sec. 4, the formal analysis of the computational complexity of WLLC is given. It also determines the expected fraction of zero digits of k-bit integers with Hamming weight $\leq k/2$. We conclude the paper in Sec. 5.

2 Mathematical Preliminaries

2.1 Canonic Signed-Digit Binary Representation (CSDBR)

Let B be a k-bit integer. A signed-digit binary representation of B is in the form:

$$B = \sum_{i=0}^{k-1} d_i 2^i \qquad d_i \in \{0, 1, -1\}. \tag{1}$$

A *Canonic Signed-Digit Binary Representation* (CSDBR) is a unique signed-digit representation which has no consecutive nonzero bits. It is commonly used to increase the efficiency of computer arithmetic. There are proofs [10,16,1,8] showing that the expected fraction of bits of an integer in CSDBR that are zeros is about $\frac{2}{3}$.

The Hamming weight of an integer B, denoted $\mathrm{Ham}(B)$, is defined to be the number of nonzero bits in B. It has been well-studied with efficient algorithms available for finding the CSDBR, which gives the minimal Hamming weight among all the binary representations. From the fact above, on average, the Hamming weight of a k-bit integer in CSDBR is about $\frac{k}{3}$.

2.2 A Signed-Digit Complex Arithmetic

Gaussian integers are complex numbers of the form $a + bi$, where a and b are integers. In 1989, Pekmestzi [9] introduced the following "binary-like" representation of Gaussian integers:

$$z = \sum_{j} d_j 2^j \qquad d_j \in \{0, 1, i, 1 + i\}. \tag{2}$$

This can be considered as a binary representation with complex digits. The digit encoding is shown in Table 2.

As an example of using this representation, let $z = 2845 + 4584i$, then we have $2845 = (0101100011101)_2$, $4584 = (1000111101000)_2$ and $z = (i, 1, 0, 1, 1 + i, i, i, i, 1, 1 + i, 1, 0, 1)$. By employing the encoding method of Table 2, we obtain the following "binary-like" representation of z:

$$z = (\underline{01}10\underline{00}10\underline{11}0\underline{10}\underline{10}11\underline{01}11\underline{00}\underline{10}).$$

Complex Digit	Binary Code
0	00
1	10
i	01
$i+1$	11

Table 2. Digit Encoding of Complex Digits

3 Review of WLLC [15]

As mentioned in the Introduction (Sec. 1), Wu, Lou, Lai and Chang [15] have recently proposed an improved algorithm of DJM-II. In this paper, we call their algorithm as WLLC. When comparing DJM-II and WLLC, the WLLC has a conditional complement operation added with the purpose of further reducing the Hamming weight of the exponents, so to reduce the total number of modular multiplications (MMs). They claim that by adding this conditional complement operation, the Hamming weight of the exponents (which are represented as a "binary-like" Gaussian integer) can further be reduced and result in faster algorithm for computing $A^X B^Y \pmod N$. For exponents X and Y such that the length of $\max(X,Y)$ is k bits long, the average number of MMs required in DJM-II is $1.556k$ while it is claimed in [15] that WLLC only requires $1.306k$ MMs on average.

Let $(X)_{SD}$ be the CSDBR of X and $\overline{X}$ the complement of X. Fig. 1 shows the pseudo-code of the WLLC algorithm. Below is an example (cited from [15]) for illustrating how this algorithm works.

Example 1. Compute $C = A^{248}B^{31} \pmod N$, where $X = 248 = (11111000)_2$, $Y = 31 = (00011111)_2$ and $k = \lceil \log_2(\max(X,Y)) \rceil = 8$. Since $\mathrm{Ham}(X) > \frac{k}{2}$ and $\mathrm{Ham}(Y) > \frac{k}{2}$, we set $X = -(\overline{X})_{SD} = -(00001001\bar{1}) = (0000\bar{1}001)$ and $Y = -(\overline{Y})_{SD} = -(100\bar{1}00000) = (\bar{1}00100000)$, where $\bar{1}$ denotes –1. Therefore, the result is:

$$\begin{aligned}
C =& A^{248}B^{31} \pmod N \\
=& A^{11111000}B^{00011111} \pmod N \\
=& A^{100000000-(0000100\bar{1})-1}B^{100000000-(100\bar{1}00000)-1} \pmod N \\
=& A^{100000000+(0000\bar{1}001)-1}B^{100000000+(\bar{1}00100000)-1} \pmod N \\
=& (A^{2^8}B^{2^8} \bmod N) \times (A^{0000\bar{1}001}B^{\bar{1}00100000} \bmod N) \times (A^{-1}B^{-1} \bmod N) \pmod N
\end{aligned} \tag{3}$$

The Hamming weight of the original $X + Yi = (1,1,1,1+i,1+i,i,i,i)$ is 8. After the transformation described in Fig. 1, the Gaussian integer $z = (-(\overline{X})_{SD}) + (-(\overline{Y})_{SD})i = (-i,0,0,i,0,-1,0,0,1)$ which contains only 4 non-zero digits. Therefore, the total number of MMs (Modular Multiplications) taken (after excluding the precomputation, i.e. Step 1 in Fig. 1) for getting the result C is 14. This includes

Input: A, B, X, Y, N
Output: $C = A^X B^Y \pmod N$
Step 1: Pre-compute
$k_x = \lceil \log_2(X) \rceil$; $k_y = \lceil \log_2(Y) \rceil$; $k = \max(k_x, k_y)$;
$a_1 = AB \pmod N$; $a_2 = A^{-1} \pmod N$; $a_3 = B^{-1} \pmod N$;
$a_4 = A^{-1}B \pmod N$; $a_5 = AB^{-1} \pmod N$; $a_6 = A^{-1}B^{-1} \pmod N$;
$a_7 = A^{2^k} \pmod N$; $a_8 = B^{2^k} \pmod N$; $a_9 = A^{2^k} B^{2^k} \pmod N$.
Step 2: **If** $\mathrm{Ham}(X) > \frac{k}{2}$ **then** set $X = -(\overline{X})_{SD}$; $HW_x = 1$ **else** set $X = X_{SD}$; $HW_x = 0$.
If $\mathrm{Ham}(Y) > \frac{k}{2}$ **then** set $Y = -(\overline{Y})_{SD}$; $HW_y = 1$ **else** set $Y = Y_{SD}$; $HW_y = 0$.
Let $X = \sum_{j=0}^{k_x} x_j 2^j$ and $Y = \sum_{j=0}^{k_y} y_j 2^j$ for $x_j, y_j \in \{0, 1, -1\}$.
Step 3: Set Gaussian integer $z = X + Yi$, that is, $z = \sum_{j=0}^{k} z_j 2^j$, where $z_j = x_j + y_j i$.
Step 4: Set $C = 1$.
For $j = k$ down to 0 **do**
$C = C \times C \pmod N$
switch(z_j)
case 1: $C = C \times A \pmod N$
case i: $C = C \times B \pmod N$
case $1 + i$: $C = C \times a_1 \pmod N$
case -1: $C = C \times a_2 \pmod N$
case $-i$: $C = C \times a_3 \pmod N$
case $-1 + i$: $C = C \times a_4 \pmod N$
case $1 - i$: $C = C \times a_5 \pmod N$
case $-1 - i$: $C = C \times a_6 \pmod N$
Step 5: **If** ($HW_x = 1$ and $HW_y = 0$) **then** set $C = a_7 \times C \times a_2 \pmod N$
If ($HW_x = 0$ and $HW_y = 1$) **then** set $C = a_8 \times C \times a_3 \pmod N$
If ($HW_x = 1$ and $HW_y = 1$) **then** set $C = a_9 \times C \times a_6 \pmod N$
Step 6: Output C

Fig. 1. The WLLC Algorithm for Computing $A^X B^Y \pmod N$

12 MMs[1] for computing $A^{0000\bar{1}001} B^{\bar{1}00100000} \pmod N$ and two more for computing the final product, that is, $a_9 \times (A^{0000\bar{1}001} B^{\bar{1}00100000} \bmod N) \times a_6 \pmod N$.

3.1 A Comparison With DJM-II

We now compare the complexity of WLLC with that of the original DJM-II [3] which is reviewed in Fig. 4, Appendix A. As we can see, for $X = 248 = (11111000)_2$ and $Y = 31 = (00011111)_2$, their CSDBRs are $(X)_{SD} = (10000\bar{1}000)$ and $(Y)_{SD} = (0010000\bar{1})$, respectively. Hence the Gaussian integer $z = (1, 0, 0, i, 0, -1, 0, 0, -i)$ and $\mathrm{Ham}(z) = 4$. The reduction of the Hamming weight in DJM-II is also from 8 to 4. As a result, by using the original DJM-II, the number of MMs taken for Example 1 above is 12.

[1] We ignore the MM for computing $C = C \times C \pmod N$ when $j = k$ since the value of C is always equal to one at that moment. As a result, there are 8 MMs corresponding to $C = C \times C \pmod N$ for $j = k-1, \cdots, 0$ (where $k = 8$) and 4 MMs corresponding to the 4 non-zero digits in z.

Example 2. We now take a look at another example from the WLLC paper [15], which computes $C = A^{248}B^{15} \pmod N$. Similarly, $X = 248 = (11111000)_2$ and $Y = 15 = (00001111)_2$. Since $\mathrm{Ham}(X) > \frac{k}{2}$ but $\mathrm{Ham}(Y) \leq \frac{k}{2}$, we compute $-(\overline{X})_{SD} = (0000\bar{1}001)$ and $Y_{SD} = (00010001)$, and set the Gaussian integer $z = (0, 0, 0, i, -1, 0, 0, 1 - i)$. Hence $\mathrm{Ham}(z) = 3$ which yields the total number of MMs taken for getting the result of C to be 13 (i.e. 11 MMs for computing $A^{0000\bar{1}001}B^{00010001} \pmod N$ and two more for computing the final product, that is, $a_7 \times (A^{0000\bar{1}001}B^{00010001} \bmod N) \times a_2 \pmod N$.

If DJM-II is used, it is easy to verify that the Hamming weight of the Gaussian integer z is 4 and therefore, the total number of MMs required for computing C is 12.

Remark 1. It is interesting to note that in both of the examples above, DJM-II performs better than WLLC. The conditional complement operation introduced in WLLC does not seem to further reduce the Hamming weight of the Gaussian integer z in any significant way, if there is any, and sometimes, it may be offset by the two additional MMs for computing the final product.

In [15], the authors also give the computational complexity analysis for the WLLC algorithm, according to which, they claim that the average number of MMs is $1.306k$, which is much more efficient than all the known algorithms, as shown in Table 1. In the next section, we show that the actual computational complexity should be $1.556k$ which is comparable to that of the original DJM-II algorithm.

4 Complexity Analysis

According to Fig. 1, we can see that on average, the number of MMs taken by WLLC is

$$N_{WLLC} \approx k + (1 - f_{k/2}^2)k = (2 - f_{k/2}^2)k \tag{4}$$

when k is large, where $f_{k/2}$ denotes the average fraction of zero digits in each of X and Y of the Gaussian integer z in Step 3 (Fig. 1). In Eq. (4) above, the first term k corresponds to the total number of $C = C \times C \pmod N$ evaluations in Step 4 (Fig. 1)[2]. The second term $(1 - f_{k/2}^2)k$ corresponds to the number of non-zero digits in the Gaussian integer z which incurs the MM operations in the switch-case statement of Step 4 (Fig. 1). For large k, the two additional MM operations for obtaining the final product in Step 5 (Fig. 1) can be ignored. N_{WLLC} is used to measure the computational complexity of WLLC.

For finding the value of N_{WLLC}, we need to determine the value of $f_{k/2}$. Note that although it has been shown [10,16,1,8] that the average fraction of zero digits

[2] As before, we ignore the evaluation when $j = k$ since the value of C is always equal to one at that moment.

of the CSDBR of k-bit integers is $\frac{2}{3}$, this does not imply that the average fraction of zero digits of the CSDBR of the *subset* of k-bit integers under the condition that their Hamming weight is at most $\frac{k}{2}$ is also $\frac{2}{3}$.

The subsections below are organized as follows. We first review a CSDBR encoding algorithm in Sec. 4.1. In Sec. 4.2, we formalize the CSDBR encoding algorithm as a state machine under the Markov chain model. This will facilitate our evaluation of $f_{k/2}$. Then, we determine the expected Hamming weight of the CSDBR of k-bit integers with Hamming weight at most $k/2$ and this is the value of $f_{k/2}$. In Sec. 4.3, we compute the corrected computational complexity for WLLC, that is, the value of N_{WLLC}.

4.1 A CSDBR Encoding Algorithm

In Fig. 2, we describe a CSDBR encoding algorithm which is derived from [1, Eq. (3)]. Let $b = (b_k b_{k-1} \dots b_2 b_1)$ be a k-bit integer (where $b_i \in \{0, 1\}$, for $1 \le i \le k$.

Input: $b = (\dots 0 b_k \dots b_i b_{i-1} \dots b_1)$
Output: $b = (\dots b_i b_{i-1} \dots b_1)$ — CSDBR of b
$\epsilon_0 = \cdots = \epsilon_{k+1} := 0$
For $t = 1$ to $k+1$ **do**
    If $b_t + \epsilon_{t-1} = 0$ **then** set $b_t = 0$; $\epsilon_t = 0$
    else if $b_t + \epsilon_{t-1} = 2$ **then** set $b_t = 0$; $\epsilon_t = 1$
    else if $b_t + \epsilon_{t-1} = 1$ and $b_{t+1} = 0$ **then** set $b_t = 1$; $\epsilon_t = 0$
    else if $b_t + \epsilon_{t-1} = 1$ and $b_{t+1} = 1$ **then** set $b_t = -1$; $\epsilon_t = 1$
return b

Fig. 2. Arno-Wheeler CSDBR Encoding Algorithm

Let $\hat{b}_t$ be the generated CSDBR (i.e. the output bit of the algorithm) of b_t. We can see that the algorithm in Fig. 2 has four states as shown in Table 3.

State	Condition	$\hat{b}_t$	ϵ_t
1	$b_t + \epsilon_{t-1} = 0$	0	0
2	$b_t + \epsilon_{t-1} = 2$	0	1
3	$b_t + \epsilon_{t-1} = 1$; $b_{t+1} = 0$	1	0
4	$b_t + \epsilon_{t-1} = 1$; $b_{t+1} = 1$	-1	1

Table 3. State Definitions and Transition Output

4.2 CSDBR Hamming Weight for k-bit Integers with Hamming Weight at Most $k/2$

The Arno-Wheeler CSDBR encoding algorithm reviewed in Table 3 has a set of four states $S = \{s_1, s_2, s_3, s_4\}$. The process of the Arno-Wheeler CSDBR encoding algorithm starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state s_i, then it moves to state s_j at the next step with a probability denoted by p_{ij}, for some $1 \leq i, j \leq 4$, and this probability does not depend upon which states the process was in before the current state. Therefore, the state transition satisfies the definition of a Markov chain. The probability p_{ij} is called transition probability. The process can remain in the state it is in, and this occurs with probability p_{ii}. A transition matrix can be composed accordingly. According to the state transition rules (see Table 3), we have Markov Chain Transition matrix as follows:

$$S = \begin{bmatrix} 1-a & 0 & a(1-a) & a^2 \\ 0 & a & (1-a)^2 & (1-a)a \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5}$$

where a is the probability that any bit equals 1. The stationary distribution is:

$$\begin{aligned} w &= (w_1, w_2, w_3, w_4) \\ &= (\frac{(1-a)^2}{a^2-a+1}, \frac{a^2}{a^2-a+1}, \frac{a(1-a)^2}{a^2-a+1}, \frac{(1-a)a^2}{a^2-a+1}) \end{aligned} \tag{6}$$

The correctness of this distribution can easily be verified from the fact that $wS = w$ for all ergodic Markov chains ([6, Sec. 11.3]) and $w_1 + w_2 + w_3 + w_4 = 1$.

Let $N_k(j)$ be the frequency of state j (for $j = 1, 2, 3, 4$ in Table 3) that appears in the CSDBR transition process. According to the **Law of Large Numbers for Markov Chains** ([6, Sec. 11.3]), $N_k(j) \approx w_j * k$ when $k \to \infty$.

Since when $j = 1$ or 2, the output digit in CSDBR is zero, the fraction Z_k of zero digits in CSDBR is:

$$\begin{aligned} Z_k &= \frac{N_k(1) + N_k(2)}{k} = w_1 + w_2 \\ &= \frac{(1-a)^2}{a^2-a+1} + \frac{a^2}{a^2-a+1} \\ &= 2 - \frac{1}{a^2-a+1}. \end{aligned} \tag{7}$$

Remark 2. From this equation, we can see that the value of Z_k is minimum (i.e. $Z_k = 2/3$) when $a = 1/2$.

Lemma 1. *Let $E(a)$ be the expected probability that a non-zero digit appears in a uniformly distributed k-bit integer with Hamming weight $\leq k/2$. $E(a) \approx \frac{1}{2}$ when $k \to \infty$.*

Proof. Let T be the number of k-bit integers with Hamming weight $\leq \frac{k}{2}$. Let i be the number of non-zero digits. C_k^i is the number of k-bit integers that have the Hamming weight equal to i.

1. If k is odd, $T = C_k^0 + C_k^1 + \cdots + C_k^{\frac{k-1}{2}}$. Since $C_k^0 + C_k^1 + \cdots + C_k^{\frac{k-1}{2}} + C_k^{\frac{k+1}{2}} + \cdots + C_k^{k-1} + C_k^k = 2^k$, we have
$$T = 2^{k-1} \tag{8}$$
2. If k is even, $T = C_k^0 + C_k^1 + \cdots + C_k^{\frac{k}{2}}$. Since $C_k^0 + C_k^1 + \cdots + C_k^{\frac{k}{2}-1} + C_k^{\frac{k}{2}} + C_k^{\frac{k}{2}+1} + \cdots + C_k^{k-1} + C_k^k$, we have
$$T = \frac{2^k + C_k^{\frac{k}{2}}}{2} \tag{9}$$

Note that C_k^i/T denotes the fraction of the k-bit integers with i non-zero digits among all the k-bit integers with Hamming weight $\leq \frac{k}{2}$. Given i, for each of those C_k^i integers with Hamming weight i, a non-zero digit appears with the probability $\frac{i}{k}$. Hence the expected fraction of non-zero digits of any uniformly distributed k-bit integers with Hamming weight $\leq k/2$ can be computed as follows:

$$E(a) = \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \frac{C_k^i}{T} \cdot \frac{i}{k} \tag{10}$$

According to [5, Eq. (5.18) on page 166], we know that

$$\sum_{i \leq m} C_k^i \cdot (\frac{k}{2} - i) = \frac{m+1}{2} \cdot C_k^{m+1} \tag{11}$$

By applying this to Eq. (10), we have

$$E(a) = \frac{1}{T \cdot k} [\sum_{i \leq \lfloor \frac{k}{2} \rfloor} C_k^i \cdot \frac{k}{2} - \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} \cdot C_k^{\lfloor \frac{k}{2} \rfloor + 1}] \tag{12}$$

1. If k is odd, $T = 2^{k-1}$, we have
$$\begin{aligned} E(a) &= \frac{1}{2^{k-1} \cdot k} \cdot \frac{k}{2} \cdot 2^{k-1} - \frac{\lfloor \frac{k}{2} \rfloor + 1}{2^k \cdot k} \cdot C_k^{\lfloor \frac{k}{2} \rfloor + 1} \\ &\approx \frac{1}{2} \ (k \to \infty) \end{aligned} \tag{13}$$

2. If k is even, $T = (2^k + C_k^{\frac{k}{2}})/2$, we have

$$\begin{aligned} E(a) &= \frac{1}{\frac{2^k + C_k^{k/2}}{2} \cdot k} \cdot \frac{k}{2} \cdot 2^{k-1} - \frac{\frac{k}{2}+1}{(2^k + C_k^{k/2}) \cdot k} \cdot C_k^{\frac{k}{2}+1} \\ &\approx \frac{1}{2} \ (k \to \infty) \end{aligned} \tag{14}$$

For both of the approximations above, we use the fact that

$$\lim_{k\to\infty} \frac{C_k^{\lfloor k/2 \rfloor}}{2^{k-1}} = 0. \tag{15}$$

□

We now have the following theorem.

Theorem 1. *For a uniformly distributed space constituted by all k-bit integers with Hamming Weight at most $k/2$, the expected fraction of zero digits in the CSDBR of integers in the space is $\frac{2}{3}$.*

Proof. According to Lemma 1, we have $E(a) \approx \frac{1}{2}$ $(k \to \infty)$. Then from Eq. 7, we have that $Z_k = \frac{2}{3}$, when k is large. □

Experimental Results. We have done an experiment to determine the average fraction of zero bits in the CSDBR of k-bit integers with Hamming weight at most $k/2$. The experimental result is shown in Table 4 and illustrated in Fig. 3.

4.3 Computational Complexity of WLLC

Theorem 2. *The computational complexity of WLLC for computing $C = A^X B^Y \pmod N$ is about $1.556k$ when k is large, where $k = \max(\lceil \log_2(X) \rceil, \lceil \log_2(Y) \rceil)$.*

Proof. According to Theorem 1, the expected fraction of zero digits in the CSDBR of k-bit integers with Hamming weight $\leq k/2$ is $\frac{2}{3}$. This is the value of $f_{k/2}$, which is the expected fraction of zero digits in each of X and Y of the Gaussian integer z in Step 3 (Fig. 1). Now from Eq. (4), we can compute the computational complexity of WLLC, that is, the value of $N_{WLLC} \approx (2 - (\frac{2}{3})^2)k = 1.556k$ when k is large. □

5 Conclusion

We provided a formal complexity analysis and found that for the CSDBR of k-bit integers with Hamming weight $\leq \frac{k}{2}$, the expected fraction of zero digits is about $\frac{2}{3}$ which is the same as the case when we consider the CSDBR of *all* k-bit integers.

Length k	Fraction	Length k	Fraction
4	0.645	5	0.692
6	0.644	7	0.673
8	0.646	9	0.666
10	0.648	11	0.663
12	0.650	13	0.661
14	0.652	15	0.661
16	0.653	17	0.660
18	0.654	19	0.660
20	0.65543	21	0.66044
22	0.65624	23	0.66057
24	0.65694	25	0.66073
26	0.65754	27	0.66090
28	0.65808	29	0.66107
30	0.65855		

Table 4. Expected Fraction of Non-zero Digits in CSDBR of k-bit integers with Hamming weight $\leq k/2$

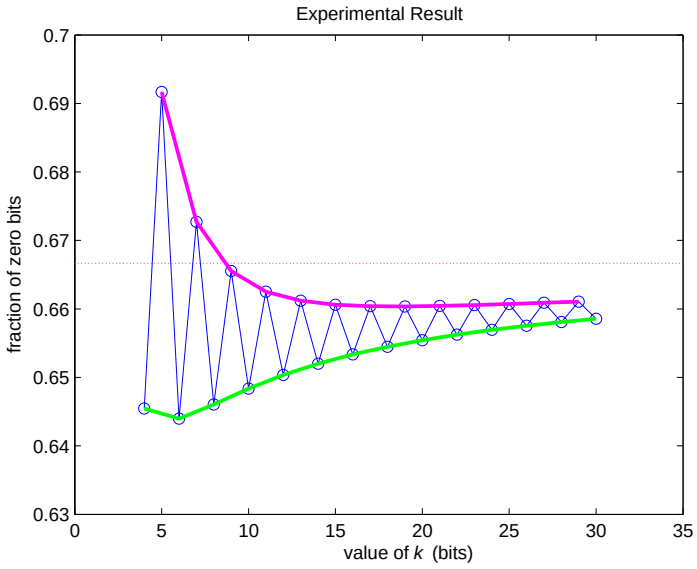

Fig. 3. Expected Fraction of Non-zero Digits in CSDBR of k-bit integers with Hamming weight $\leq k/2$ — A Graphical Illustration

Based on this fact, we showed that the computational complexity of the WLLC algorithm is $1.556k$ rather than $1.306k$ (in terms of the number of MMs taken) as it is originally claimed in [15]. This means that the best modular multi-exponentiation algorithm based on canonical-sighed-digit technique is still not able to overcome the $1.5k$ barrier.

References

1. S. Arno and F. S. Wheeler. Signed digit representations of minimal Hamming weight. *IEEE Transactions on Computers*, 42(8):1007–1010, 1993. (Cited on pages 3, 6, and 7.)
2. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology - CRYPTO 2003*, volume 2729/2003, pages 126–144. Springer Berlin, 2003. Lecture Notes in Computer Science. (Cited on page 1.)
3. V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Complexity and fast algorithms for multiexpontations. *IEEE Transactions on Computers*, 49:141–147, 2000. (Cited on pages 2, 5, 12, and 13.)
4. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985. (Cited on pages 1 and 2.)
5. R. L. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley Publishing Company, 1989. (Cited on page 9.)
6. C. M. Grinstead and J. L. Snell. *Introducation to Probability*. American Mathematical Society, 1997. (Cited on page 8.)
7. ITL. Digital signature standard (DSS). Technical Report FIPS 186, National Institute of Standards and Technology, 1991. (Cited on page 1.)
8. K. Koyama and Y. Tsuruoka. A signed binary window method for fast computing over elliptic curves. *IEICE Trans. Fundamentals*, E76-A:55–62, 1993. (Cited on pages 3 and 6.)
9. K. Z. Pekmestzi. Complex number multipliers. In *Computers and Digital Techniques, IEE Proceedings*, volume 136, pages 70–75, 1989. (Cited on page 3.)
10. G. W. Reitweisner. Binary arithmetics. *Advances in Computers*, 1:231–308, 1960. (Cited on pages 3 and 6.)
11. S.-M.Yen, C.-S. Laih, and A. Lenstra. Multi-exponentiation. In *Computers and Digital Techniques, IEE Proceedings*, volume 141, pages 325–326, 1994. (Cited on page 2.)
12. C. P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - EUROCRYPT '89*, volume 434, pages 688–689. Springer Berlin, 1990. Lecture Notes in Computer Science. (Cited on page 1.)
13. J. A. Solinas. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, University of Waterloo, 1998. (Cited on page 2.)
14. B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494/2005, pages 114–127. Springer Berlin, 2005. Lecture Notes in Computer Science. (Cited on page 1.)
15. C.-L. Wu, D.-C. Lou, J.-C. Lai, and T.-J. Chang. Fast modular multi-exponentiation using modified complex arithmetic. *Applied Mathematics and Computation*, 186:1065–1074, 2007. (Cited on pages 1, 2, 4, 6, and 11.)
16. C. N. Zhang. An improved binary algorithm for RSA. *Computers and Math. with Applications*, 25:15–24, 1993. (Cited on pages 3 and 6.)

A Review of DJM-II

Fig. 4 shows the DJM-II algorithm [3]. In the algorithm, we recall that $(X)_{SD}$ represents the CSDBR of X.

Input: A, B, X, Y, N
Output: $C = A^X B^Y \pmod N$
Step 1: Pre-compute
$k_x = \lceil \log_2(X) \rceil$; $k_y = \lceil \log_2(Y) \rceil$; $k = \max(k_x, k_y)$;
$a_1 = AB \pmod N$; $a_2 = A^{-1} \pmod N$; $a_3 = B^{-1} \pmod N$;
$a_4 = A^{-1}B \pmod N$; $a_5 = AB^{-1} \pmod N$; $a_6 = A^{-1}B^{-1} \pmod N$
Step 2: set $X = (X)_{SD}$ and $Y = (Y)_{SD}$
Let $X = \sum_{j=0}^{k_x} x_j 2^j$ and $Y = \sum_{j=0}^{k_y} y_j 2^j$ for $x_j, y_j \in \{0, 1, -1\}$.
Step 3: Set Gaussian integer $z = X + Yi$, that is, $z = \sum_{j=0}^{k} z_j 2^j$, where $z_j = x_j + y_j i$.
Step 4: Set $C = 1$.
For $j = k$ down to 0 **do**
$C = C \times C \pmod N$
switch(z_j)
case 1: $C = C \times A \pmod N$
case i: $C = C \times B \pmod N$
case $1 + i$: $C = C \times a_1 \pmod N$
case -1: $C = C \times a_2 \pmod N$
case $-i$: $C = C \times a_3 \pmod N$
case $-1 + i$: $C = C \times a_4 \pmod N$
case $1 - i$: $C = C \times a_5 \pmod N$
case $-1 - i$: $C = C \times a_6 \pmod N$
Step 5: Output C

Fig. 4. DJM-II [3]