An Analysis of Exponentiation Based on Formal Languages

Luke O'Connor

IBM Research Division
Zurich Research Laboratory
Säumerstrasse 4, Rüschlikon
CH-8803, Switzerland
oco@zurich.ibm.com

Abstract. A recoding rule for exponentiation is a method for reducing the cost of the exponentiation a^e by reducing the number of required multiplications. If $w(e)$ is the (hamming) weight of e, and $\bar{e}$ the result of applying the recoding rule A to e, then the purpose is to reduce $w_A(\bar{e})$ as compared to $w(e)$. A well-known example of a recoding rule is to convert a binary exponent into a signed-digit representation in terms of the digits $\{ 1, \bar{1}, 0 \}$ where $\bar{1} = -1$, by recoding runs of 1's. In this paper we show how three recoding rules can be modelled via regular languages to obtain precise information about the resulting weight distributions. In particular we analyse the recoding rules employed by the 2^k-ary, sliding window and optimal signed-digit exponentiation algorithms. We prove that the sliding window method has an expected recoded weight of approximately $n/(k+1)$ for relevant k-bit windows and n-bit exponents, and also that the variance is small. We also prove for the optimal signed digit method that the expected weight is approximately $n/3$ with a variance of $2n/27$. In general the sliding window method provides the best performance, and performs less than 85% of the multiplications required for the other methods for a majority of exponents.

1 Introduction

One of the fundamental operations in cryptography is exponentiation a^e over groups such as $\mathbb{Z}_p^*, \mathbb{Z}_n$, general finite fields, and the group of points on an elliptic curve [21,6,7]. The classical approach to performing this task is the binary method, and the complexity of the exponentiation is usually measured in terms of the number of squarings and multiplications required to determine a^e. Let $e = e_{n-1}e_{n-2}\cdots e_1e_0$ be an n-bit exponent, $e_i \in \{0,1\}$, $0 \leq i < n$, and let $w(e) = \sum_{i=0}^{n-1} e_i$ be the weight of e. A simple analysis of the binary method shows that s squarings and $w(e)-1$ multiplications are required, where s is the index of the most significant bit in e. Many general exponentiation algorithms offer complexity improvements over the binary method include the sliding-window method ([18,4,11] for example), signed-digit representations [20,19,16,13,23], the signed-window method [17], Lempel-Ziv recoding [24], and the string replacement method [8]. The reader is advised to see [18] for a thorough survey.

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 375–388, 1999.

The common approach of these and other methods is to 'collect' exponent bits according to some rule for reducing the weight of e, hence reducing the number of required multiplications. For example, k consecutive bits are collected to form a single digit in the 2^k-ary method [15], and the binary signed-digit method [8] replaces runs of two or more 1's with just two bits, one signed and one unsigned. We will refer to these and other rules for reducing the weight of e as a *recoding rule*. For a given recoding rule A let $\bar{e}_A = \bar{e}_t\bar{e}_{t-1}\dots\bar{e}_1\bar{e}_0$ be the result of applying A to e, and let $w_A(\bar{e}) = \sum_{i=0}^{t}[\bar{e}_i \neq 0]$ denote the recoded weight of $\bar{e}_A$. Once the recoding rule is applied, a variant of the b-ary method (b not necessarily equal to 2^k) can be used to complete the exponentiation, potentially after some precomputation has been done. In practice, the exponent recoding and arithmetic operations of the exponentiation are interleaved (see [18] for examples of specific algorithms).

To analyse the computational saving of recoding e according to rule A, we are required to examine the distribution of $w_A(\bar{e})$, and also the cost of any precomputation implied by A. For the 2^k-ary method, $w_A(\bar{e})$ is approximately binomial with parameters $b(n/k, (2^k-1)/2^k)$, and it is therefore reasonably understood. It is surprising however that in general other recoding methods are discriminated between solely on the basis of $\mathbf{E}[(w_A(\bar{e})]$ and $\max_e w_A(\bar{e})$, the average and worst case weight recodings respectively (see [17,8,23] for such comparisons). We assert that $\mathbf{E}[(w_A(e)]$ and $\max_e w_A(e)$ provide information about the distribution of $w_A(e)$, but without second order statistics, such as the variance, the accuracy and usefulness of this information is uncertain.

In a recoding rule A that produces $\bar{e}_A = \bar{e}_t\bar{e}_{t-1}\dots\bar{e}_1\bar{e}_0$ from e, often the defining properties of the $\bar{e_i}$ are quite simple, such as $\bar{e}_i = 01^k$ (a run of 1's terminated by a 0, $k \geq 2$) used in signed-digit recoding for example. This reflects the requirement that the recoding rule must be efficient, and also that simple recoding rules can be effective in reducing the cost of exponentiation. For many recoding rules of practical interest, the $\bar{e}_i$ can be represented as elements of a specified regular language [10], implying that the recoding can be performed by an appropriate deterministic finite automata (DFA). For example, the recoding rules presented in [19,17] are analysed in terms of their respective recoding DFAs.

The main contribution in this paper is to propose a framework for analysing the weight distribution of recoding rules which can be described by regular languages. For a recoding rule A, the basis of our analysis is to define a bivariate generating function (bgf) $G_A(x,z) = \sum_{n,m\geq 0} a_{n,m}z^m x^n$ such that

$$\Pr(w_A(\bar{e}) = m \mid \#e = n) = a_{m,n}/2^n,$$

where $\#e$ is the bit length of e. Thus $\Omega_n = \{m \mid a_{m,n} \neq 0, 0 \leq m \leq n\}$ and $\Pr(X_n = m) = a_{m,n}/2^n$, will be the probability space describing the distribution of weights for n-bit exponents recoded according to A. For the binary method (BM), the relevant bgf (derived below) is

$$G_{BM}(x,z) = \frac{1}{1-(xz+x)} = \frac{1}{1-x(1+z)} = \sum_{n\geq 0} x^n \sum_{m\geq 0} \binom{n}{m} z^m \qquad (1)$$

which indicates that the weights are distributed binomially, as expected. In general we will derive $G_A(x,z)$ from a A by considering the recoding rules prescribed by A as being performed by a DFA, pass to regular languages, and then enumerate the set of n-bit exponents whose recoded weight is m using standard combinatorial methods (see [22, p.377] or [3, p.342] for example). This analysis technique covers many recoding methods of practical interest, but, for example, does not include the Lempel-Ziv exponentiation method of Yacobi [24], since in this case the $\bar{e_i}$ are produced by the recoding are non-regular (a context-free grammar would be required).

To demonstrate the generality of this approach, we analyse the weight distribution of recoded exponents for the 2^k-ary method (§3), sliding window method (§4) and the optimal signed-digit method (§5). We analyse the 2^k-ary method as it provides an obvious improvement over the binary method, and its analysis is instructive to the bgf approach. The sliding window method was selected since no satisfactory analysis exists (see [11,14] for partial results), and yet it is described as 'the recommended method' for general exponentiation [18, p.617]. We also selected the optimal signed-digit method[8] for analysis since this method and its variants are often suggested for performing elliptic curve scalar multiplication, since group inversion is essentially free [19,16,17,23]. For the 2^k-ary (k, TKM), k-bit sliding window (k, SW), and optimal signed digit (OSD) methods the bgfs for weight are as follows:

$$G_{k,TKM}(x,z) = \frac{z\left(\frac{1-2^k x^k}{1-2x} - \frac{1-x^k}{1-x}\right) + \frac{1-x^k}{1-x}}{1 - x(2^k-1)x^k - x^k}, \tag{2}$$

$$G_{k,SW}(x,z) = \frac{1 - 2x + zx - zx^k 2^{k-1}}{(1 - x - zx^k 2^{k-1})(1-2x)}, \tag{3}$$

$$G_{OSD}(x,z) = \frac{1 - x + xz + -2zx^2 + x^2 z^2}{1 - 2x + x^2 - 2zx^2 + 2zx^3}. \tag{4}$$

Using standard transformations on bgfs we are able to obtain the numerical values of $\mathbf{E}[w_A(\bar{e})]$ and $\mathbf{Var}[w_A(\bar{e})]$ for each bgf from (2) - (4), and thus make comparisons on the number of required multiplications for each method. Since the TKM approximates the binomial distribution $b(n/k, (2^k-1)/2^k)$, the expectation and variance of $w_{k,TKM}(\bar{e})$ can be approximated accurately. Similar computations for the sliding window method are difficult, but we have been able to show by direct calculation that $\mathbf{E}[w_{k,SW}(\bar{e})] \sim n/(k+1) + \frac{k(k-1)}{2(k+1)^2}$ for $n \in \{512, 1024\}$, $k \in \{2, 3, \ldots, 6\}$. We currently have no expression for $\mathbf{Var}[w_{k,SW}(\bar{e})]$ but we note that direct calculations show it to be small (for example less than 7 for 6-bit windows on 1024-exponents), and decreasing with k. The expectation and variance the OSD method can be analysed exactly, mainly because there is no window parameter k to complicate the analysis. We prove that $\mathbf{E}[w_{OSD}(\bar{e})] = \frac{n}{3} + \frac{4}{9} - \frac{4(-1)^n}{9 \cdot 2^n}$, and $\mathbf{Var}[w_{OSD}(\bar{e})] = \frac{2n}{27} + \frac{14}{81} + \frac{2n}{27 \cdot 2^n} + o(1)$.

The paper is organised as follows. In §2 we review some concepts of regular languages, and give the principal enumeration theorems. In §3 we derive the bgfs

for the binary and 2^k-ary method, demonstrating our method of enumeration. In §4 we analyse the sliding window method, and then in §5 we analyse the optimal signed digit method. Conclusions and open problems are presented in the last section.

2 Regular Expressions and Generating Functions

Regular expressions are defined recursively [10] as follows: if R and S are regular expressions then so is $R+S$ (union), RS (concatenation) and R^* (Kleene closure) where $R^* = \sum_{k\geq 0} R^k = \epsilon + R + RR + RRR + \cdots$. Also let r^k denote the concatenation of r with itself k times, and let $r^+ = r^* - \epsilon$. Over a binary alphabet we will call 1^k a k-run, $k \geq 1$, and any word ω that is a k-run will also be simply referred to as a run.

A regular expression R generates words $\omega = w_1 w_2 \cdots w_n$, $w_i \in \Delta$, and ω is said to have length n, written as $\#\omega = n$. The set of all words generated by the regular expression R, denoted by L_R, is called the regular language generated, or given, by R. Let $L_R^n \subseteq L_R$ denote the set of words in L_R of length $n \geq 0$. We will say that the (ordinary) generating function $G_R(x) = \sum_{n\geq 0} a_n x^n$ enumerates L_R by length if $a_n = \#L_R^n$ for all $n \geq 0$. Let $[x^n]$ be the operator that extracts the coefficient of x^n, so that $[x^n]G_R(x) = a_n$. It is clear that the regular expression $R = (1+0)^*$ generates the language L_R which is the set of all binary strings, and since $|L_R^n| = 2^n$, L_R is enumerated by the geometric series $G_R(x) = 1/(1-2x)$. The key property that permits $G_R(x)$ to be derived from R directly is given in the next definition.

Definition 1. A regular expression R is *unambiguous* if there is only one way for R to generate each $\omega \in L_R$. □

For example $(1+0)^*$ is unambiguous, but $(1+0+10)^*$ is ambiguous since the string $\omega = 10$ can be generated by concatenating 1 and 0, or simply selecting 10. Since it is known that any regular language can be generated by an unambiguous regular expression [22, p.378], the following theorem due to Chomsky and Schutzenberger [5] will be our main enumeration tool.

Theorem 2. Let R and S be unambiguous regular expressions, that are enumerated by the gfs $G_R(x)$ and $G_S(x)$. Then if $R+S$, RS and R^* are also unambiguous, $G_R(x) + G_S(x)$ enumerates $R+S$, $G_R(x)G_S(x)$ enumerates RS, and $1/(1-G_R(x))$ enumerates R^*. □

Recall that our goal is to determine the bgf $G_A(x,z) = \sum_{n,m\geq 0} a_{n,m} z^m x^n$ such that $a_{m,n}$ is the number of n-bit exponents recoded to weight m by algorithm A. Fortunately Theorem 2 can also be applied to these bgfs since for the exponent recoding algorithms under consideration there exists a representation of the algorithms in terms of regular expressions for which $w(R+S) = w(R)+w(S)$ and $w(RS) = w(R)w(S)$. We restate this result formally as a corollary to Theorem 2.

Corollary 3. Let R and S be unambiguous regular expressions, that are enumerated by the bgfs $G_R(x,z)$ and $G_S(x,z)$. Then if $R+S$, RS and R^* are also unambiguous, $w(R+S) = w(R) + w(S)$, and $w(RS) = w(R)w(S)$ then $G_R(x,z) + G_S(x,z)$ enumerates $R+S$, $G_R(x,z)G_S(x,z)$ enumerates RS, and $1/(1-G_R(x,z))$ enumerates R^*. □

An advantage of using $G_A(x,z)$ for enumeration is that the expectation and variance of $w_A(\bar{e})$ can be directly determined from manipulating $G_A(x,z)$. Using standard operations on bgfs (see for example [22, p.138]) we have that

$$\mathbf{E}[w_A(\bar{e})] = [x^n]\left(\left.\frac{\partial G_A(x/2,z)}{\partial z}\right|_{z=1}\right), \tag{5}$$

$$\begin{aligned}\mathbf{Var}[w_A(\bar{e})] = [x^n]\left(\left.\frac{\partial^2 G_A(x/2,z)}{\partial^2 z}\right|_{z=1} + \left.\frac{\partial G_A(x/2,z)}{\partial z}\right|_{z=1}\right) \\ - \left([x^n]\left(\left.\frac{\partial G_A(x/2,z)}{\partial z}\right|_{z=1}\right)\right)^2,\end{aligned} \tag{6}$$

where $[x^n]G(x)$ is the coefficient of x^n in $G(x)$. Thus $\mathbf{E}[w_A(\bar{e})]$ and $\mathbf{Var}[w_A(\bar{e})]$ can be extracted by several differentiations of $G_A(x,z)$ with respect to z, and determining the coefficient of x^n after setting $z=1$.

3 The Binary and 2^k-ary Methods

As examples of the techniques presented in the previous section, we now derive $G_{BM}(x,z)$ given in (1) for the binary method, and also $G_{k,TKM}(x,z)$ for the 2^k-ary method given in (2). First observe that the binary method processes the exponent bit-by-bit, so the relevant regular expression is $R = (1+0)^*$, which clearly generates all binary strings unambiguously. Second, marking $(1+0)$ for length and weight gives $zx+x$, and Corollary 3 indicates that R is enumerated by $G_{BM}(x,z) = 1/(1-(zx+x))$, as shown in (1). Though the 2^k-ary method (TKM) is a natural extension of the binary method, the derivation of $G_{k,TKM}(x,z)$ is more complicated than that of $G_{BM}(x,z)$.

Theorem 4. Let $a_{n,m}$ be the number of binary strings of length n for which the TKM-recoding using k-bit windows has weight m, $0 \le m < n$. Then

$$G_{k,TKM}(x,z) = \sum_{n,m\ge 0} a_{n,m} x^n z^m \;=\; \frac{z\left(\frac{1-2^k x^k}{1-2x} - \frac{1-x^k}{1-x}\right) + \frac{1-x^k}{1-x}}{1 - x(2^k-1)x^k - x^k} \tag{7}$$

Proof. Consider the following regular expression

$$R = R_1^* R_2 \;=\; \left((1+0)^k\right)^* \left(\epsilon + \sum_{i=1}^{k-1} 1(1+0)^i\right).$$

n	k	$\mathbf{E}[w_{k,TMK}(\bar{e})]$	$\mathbf{Var}[w_{k,TMK}(\bar{e})]$	0.50	0.60	0.75	0.90	0.95	0.99
512	3	149.5	18.8	7	7	9	14	20	44
512	4	120	7.5	4	5	6	9	13	28
512	5	99.6	3.3	3	3	4	6	9	19
512	6	84.4	1.5	2	2	3	4	6	13
1024	3	298.8	37.5	9	10	13	20	28	62
1024	4	240	15	6	7	8	13	18	39
1024	5	198.6	6.2	4	4	5	8	12	25
1024	6	168.3	2.7	3	3	4	6	8	17

Table 1. The 2^k-ary encoding distributions for 512- and 1024-bit exponents. The columns show the value of $\alpha(w_{k,TMK}(\bar{e}), p)$, $p \in \{0.50, 0.60, 0.75, 0.90, 0.95, 0.99\}$.

R_1^* generates all binary k-bit windows repeatedly, while R_2 generates all binary strings of length less than k. R_1 is marked for length and weight as

$$G_{R_1}(x, z) = z(2^k - 1)x^k + x^k \tag{8}$$

which denotes that all windows have length k, and all windows except one (the all-zero window) cost one multiplication in TKM. The marking for R_2 is as follows

$$G_{R_2}(x, z) = z\left(\frac{1 - 2^k x^k}{1 - 2x} - \frac{1 - x^k}{1 - x}\right) + \frac{1 - x^k}{1 - x}. \tag{9}$$

Note that $(1 - 2^k x^k)/(1 - 2x) - (1 - x^k)(1 - x)$ is the number of binary strings of length less than k that are niether empty or all-zero. These strings each cost a multiply in the TKM. The $(1 - x^k)/(1 - x)$ empty or all-zero strings cost no multiplies. The theorem follows from simplifying $G_{R_1}(x, z)G_{R_2}(x, z)$. □

Using (5) and (6), both $\mathbf{E}[w_{k,TKM}(\bar{e})]$ and $\mathbf{Var}[w_{k,TKM}(\bar{e})]$ can be determined for various values of k and n using a symbolic computation package (we have elected to use Maple [1]). Recall that Chebyshev's inequality bounds the deviation of a random variable X from its mean μ in terms of its variance σ^2: $\Pr(|X - \mu| \geq d) \leq \sigma^2/d^2$. Then define $\alpha(X, p)$ as

$$\alpha(X, p) = \min_d \left[\frac{\sigma^2}{d^2} < (1 - p)\right] \tag{10}$$

which states that d is the smallest for which $\Pr(|X - \mu| < d) > p$ according to bounds derived by Chebyshev's inequality. Table 1 shows the distribution of TKM recoding weights for various value of k for 512- and 1024-bit exponents, and also the deviations $\alpha(w_{k,TKM}(\bar{e}), p)$ for several probabilities p.

4 The Sliding Window Representation

The sliding-window method [4,11] is a variant of the b-ary method [15], and is the 'recommended method' for general exponentiation [18, p.617]. When $b = 2^k$, the 2^k-ary method can be considered as parsing an exponent e into adjacent k-bit windows, where the window covering the least significant bit may be less than k bits. The idea of the sliding-window method is to select the placement of each k-bit window so that its most and least significant bit are equal to one. The advantage of such a partition over the 2^k-ary method is twofold: first the number of windows is expected to be reduced as runs of zeroes may occur between consecutive windows, and secondly, the amount of precomputation is halved as the windows only represent odd powers. We now derive $G_{k,SW}(x, z)$, the bgf for the sliding window encoding of exponents using k-bit windows.

Theorem 5. Let $a_{n,m}$ be the number of binary strings of length n for which the SW-recoding using k-bit windows has weight m, $0 \le m < n$. Then

$$G_{k,SW}(x, z) = \sum_{n,m \ge 0} a_{n,m} x^n z^m \;=\; \frac{1 - 2x + zx - zx^k 2^{k-1}}{(1 - x - zx^k 2^{k-1})(1 - 2x)}. \tag{11}$$

Proof. Consider the following regular expression

$$R = R_1^* R_2 \;=\; \left(0 + 10^{k-1} + \sum_{i=0}^{k-2} 1(0+1)^{k-2-i} 10^i\right)^* \left(\epsilon + \sum_{i=1}^{k-2} 1(1+0)^i\right).$$

R_1^* generates words of length k that start and end with 1, and also the single word 0. Clearly R_1^* then generates all words corresponding to k-bit windows separated by runs of zeroes. R_2 generates either the empty string or a word beginning with 1, of length less than k, which corresponds to the case where the last there are not $k-1$ bits following the most significant bit of the last window.

We now mark R_1 for length and weight: 0 is marked x, 10^{k-1} is marked zx^k meaning it has length k and corresponds to one nonzero digit in the recoding, and $1(0+1)^{k-2-i}10^i$ is similarly marked as $zx^{i+2}(x+x)^{k-2}$. Using the same rules for R_2 we have that

$$G_{R_1}(x, z) = x + zx^k + zx^2 \sum_{i=0}^{k-2} \frac{(x+x)^{k-2}}{(x+x)^i} \;=\; x + zx^k + zx^k 2^{k-2}\left(2 - 2^{2-k}\right),$$

$$G_{R_2}(x, z) = 1 + zx\left(\frac{1 - x^{k-1}2^{k-1}}{1 - 2x}\right).$$

The theorem follows from simplifying $G_{R_1}(x, z)G_{R_2}(x, z)$. □

Using $\alpha(X, p)$ from (10) we can again bound the distribution of weights, which are given in Table 2 for 512-, 768- and 1024-bit exponents. Notice that the expectations are very close to $n/(k+1)$, as previously observed by Hui and Lam

n	k	$n/(k+1)$	$\mathbf{E}[w_{k,SW}(e)]$	$\mathbf{Var}[w_{k,SW}(e)]$	0.50	0.60	0.75	0.90	0.95	0.99
512	4	102.4	102.6	8.3	5	5	6	10	13	29
512	5	85.33	85.6	4.8	4	4	5	7	10	23
512	6	73.14	73.4	3.1	3	3	4	6	8	18
512	7	64	64.3	2.1	3	3	3	5	7	15
1024	4	204.8	205.0	16.5	6	7	9	13	19	41
1024	5	170.67	170.9	9.6	5	5	7	10	14	31
1024	6	146.3	146.6	6.1	4	4	5	8	12	25
1024	7	128	128.3	4.1	3	4	5	7	10	21

Table 2. k,SW encoding distributions for 512- and 1024-bit exponents. The columns show the value of $\alpha(w_{k,SW}(\bar{e}), p)$, $p \in \{0.50, 0.60, 0.75, 0.90, 0.95, 0.99\}$.

[11], and that the variances are quite small. We now consider the case of $k = 5$ explicitly, which is of interest since it is the optimal window size for the 2^k-ary method on 512-bit exponents.

Theorem 6. For a random n-bit exponent and 5-bit windows

$$\mathbf{E}[w_{5,SW}(\bar{e})] \sim \frac{n}{6} + \frac{5}{18}, \quad \mathbf{Var}[w_{5,SW}(\bar{e})] \sim \frac{n}{108} + \frac{35}{324}. \tag{12}$$

Proof. Taking the partial derivative of $G_{k,SW}(x, z)$ with respect to z, setting $k = 5$, and expanding with partial fractions we find that

$$\begin{aligned} \left.\frac{\partial G_{5,SW}(x,z)}{\partial z}\right|_{z=1,k=5} &= \frac{1}{6(1-2x)^2} + \frac{1}{9(1-2x)} + \frac{5+3x-2x^2-8x^3}{8x^4+4x^3+2x^2+x+1} \\ &= \sum_{n\geq 0} \frac{(n+1)(2x)^n}{6} + \sum_{n\geq 0} \frac{(2x)^n}{9} + \sum_{n\geq 0} O(1.77^n) \end{aligned} \tag{13}$$

where 1.77 is the complex root with largest modulus in $x^4 + x^3 + 2x^2 + 4x + 8$, which is the reflected polynomial [9, p.325] of $8x^4 + 4x^3 + 2x^2 + x + 1$. The second derivative with respect to z at $z = 1, k = 5$ has the partial fraction decomposition

$$\begin{aligned} \left.\frac{\partial^2 G_{5,SW}(x,z)}{\partial^2 z}\right|_{z=1,k=5} &= \sum_{n\geq 0} \frac{(n+1)(n+2)(2x)^n}{36} - \sum_{n\geq 0} \frac{4(n+1)(2x)^n}{27} \\ &\quad + n \cdot \sum_{n\geq 0} O(1.77^n). \end{aligned}$$

Thus using (5) and (6), the variance is asymptotic to $n/108 + 35/324$. □

Using similar computations as in Theorem 6 we have verified the following theorem.

Theorem 7. For k in the range $2 \leq k \leq 10$, $\mathbf{E}[w_{k,SW}(\bar{e})] \sim n/(k+1) + \frac{k(k-1)}{2(k+1)^2}$.

We are currently working on extending the proof of the above theorem to all k and n, which involves proving certain terms in the partial fraction expansion of $\mathbf{E}[w_{k,SW}(\bar{e})]$ tend to zero with n. At present we have no expression for $\mathbf{Var}[w_{k,SW}(\bar{e})]$, but note that in general it is small, meaning that the distribution is concentrated around its mean. For example, expanding $G_{5,SW}(x,z)$ directly for 512-bit exponents shows that 99.6% of exponents will be recoded to a weight that lies with ± 6 of $\mathbf{E}[w_{5,SW}(\bar{e})]$. Similarly, 99.998% of 512-bit exponents are recoded to within ± 10 of $\mathbf{E}[w_{5,SW}(\bar{e})]$.

5 Signed-Digit Representations

A signed-digit representation of the number e in base b is of the form $e = \sum_{i=0}^{d} a_i b_i$ where $a_i \in \{0, \pm 1, \pm 2, \ldots, \pm(b-1)\}$, implying that binary numbers are consequently encoded using the digits $\{0, 1, -1 = \bar{1}\}$. In general, the signed-digit representation of a number for a fixed base is not unique, and even the encoding of minimal weight need not be unique. An algorithm for producing minimal a weight signed-digit encoding for a general base b is given by Arno and Wheeler [2].

Working with negative exponents requires group inversions, which can be costly over some groups if the appropriate inverses cannot be precomputed. On the other hand, signed-digit representations are particularly attractive for arithmetic over elliptic curves, since they correspond to addition-subtraction chains, and point addition and subtraction on cryptographic curves have the same cost in terms of group operations [19,16,17,23].

Definition 8. Let $e = \sum_{i=0}^{d} a_i 2_i$, $a_i \in \{0, 1, -1 = \bar{1}\}$ be a minimal weight signed-digit encoding of e. The encoding is called *sparse* if no two consecutive digits a_i, a_{i+1} are both nonzero.

Jedwab and Mitchell [12] prove that sparse encodings are unique and have minimal weight. The algorithm in Figure 1 converts e to a sparse encoding [12] by repeatedly applying the identity $2^{k+1} - 1 = \sum_{i=0}^{k} 2^k$. This guarantees spareness since adjacent bits are encoded as $10 \cdots 0\bar{1}$, and for this reason sparse exponents are also said to be in nonadjacent form [23]. We will refer to exponents recoded according to Figure 1 as *Optimal Signed Digit* encodings, or OSD recodings.

Asymptotic results indicate that the weight of an OSD-encodings approaches $n/3$ for a random n-bit exponent [13,16,2]. It was only recently (1996) that the exact analysis was given by Gollman, Han and Mitchell [8] who proved that the expected weight is $n/3 - 4/9 - \frac{4(-1)^n}{9 \cdot 2^n}$. Previously, Arno and Wheller [2] exhibit a Markov chain P that mimics an OSD-encoding algorithms, whose limiting distribution for the expected number of zeros in the resulting encoding is $\frac{2n}{3}$. We now derive $G_{OSD}(x,z)$ from which we will determine the variance of an OSD-encoding.

$i \leftarrow 0$;
while true
    Find the largest $j > (i+1)$ such that $e' = e_j, e_{j-1}, \ldots, e_i = 01^{j-i}$;
    if there is no such j **then** exit ;
    else replace e' with $10^{j-i-2}\bar{1}$; $i \leftarrow j$;
od

Fig. 1. An algorithm for producing a sparse signed-digit representation of a binary number.

Theorem 9. Let $a_{n,m}$ be the number of binary strings of length n for which the OSD-recoding has hamming weight m, $0 \le m < n$. Then

$$G_{OSD}(x,z) = \sum_{n,m \ge 0} a_{n,m} x^n z^m \;=\; \frac{1 - x + xz + -2zx^2 + x^2z^2}{1 - 2x + x^2 - 2zx^2 + 2zx^3}. \qquad (14)$$

Proof. The proof is based on the following two regular expressions

$$R_1 = 10(10)^*0,$$
$$R_2 = ((10)^+11^+0 + 11^+0)(1^+0)^*0,$$

which describes how bits are propagated between runs of runs separated by at most one 0. Further details are given in the Appendix. □

Using $\alpha(X,p)$ from (10) we can again bound the distribution of weights, which are given in Table 3 for 512-, 768- and 1024-bit exponents.

Theorem 10. For a random n-bit exponent, we have that

$$\mathbf{E}[w_{OSD}(\bar{e})] = \frac{n}{3} + \frac{4}{9} - \frac{4(-1)^n}{9 \cdot 2^n}, \quad \mathbf{Var}[w_{OSD}(\bar{e})] \;=\; \frac{2n}{27} + \frac{14}{81} + \frac{2n}{27 \cdot 2^n} + o(1).$$

Proof. The partial fraction decomposition of the derivative of $G_{OSD}(x,z)$ at $z = 1$ is

$$\left.\frac{\partial G_{k,SW}(x,z)}{\partial z}\right|_{z=1} = \frac{1}{3(1-2x)^2} + \frac{1}{9(1-2x)} + \frac{1}{18(1+x)} - \frac{1}{2(1-x)} \qquad (15)$$

giving that $\mathbf{E}[w_{OSD}(\bar{e})] = n/3 + 4/9 - \frac{4(-1)^n}{9 \cdot 2^n}$. The partial fraction decomposition for the second derivative $G_{OSD}(x,z)$ at $z = 1$ is

$$\left.\frac{\partial^2 G_{k,SW}(x,z)}{\partial^2 z}\right|_{z=1} = \frac{2}{9(1-2x)^3} + \frac{2}{27(1+x)^2} - \frac{8}{27(1-2x)^2} \qquad (16)$$

for which $[x^n]/2^n$ is $(n+1)(n+2)/9 - 8(n+1)/27 - \frac{2(n+1)}{27 \cdot 2^n}$. Then $\mathbf{Var}[w_{OSD}(\bar{e})]$ is determined directly from (6). □

	$\mathbf{E}[w_{OSD}(\bar{e})]$	$\mathbf{Var}[w_{OSD}(\bar{e})]$	0.50	0.60	0.75	0.90	0.95	0.99
512	171.1	38.1	9	10	13	20	28	62
768	256.4	57.1	11	12	16	24	34	76
1024	341.7	76.0	13	14	18	28	39	88

Table 3. OSD-encoding distributions for 512-, 768- and 1024-bit exponents. The columns show the value of $\alpha(w_{OSD}(\bar{e}), p)$, $p \in \{0.50, 0.60, 0.75, 0.90, 0.95, 0.99\}$.

6 Comparisons and Conclusions

In this paper we have analysed three recoding rules for improved exponentiation over the binary method. The analysis is thorough in that for the methods considered it is possible to extract both the expectation and variance of the random variable describing the recoded weight. In several of the cases we have derived closed forms for these statistics with respect to the recoding scheme.

It remains to draw comparisons between the three recoding methods. We will only discriminate on the basis of the number of multiplications required by a method, since the number of squaring required by the 2^k-ary and sliding window methods will be similar, and even taking squarings into account, the OSD method is significantly slower. The 2^k-ary method requires $TMK(k) = 2^k + w_{k,TKM}(\bar{e}) - 4$ multiplications [18], and the k-bit sliding window method requires $SW(k) = 2^{k-1} + w_{k,SW}(\bar{e}) - 2$ multiplications. The OSD method requires at least $w_{OSD}(\bar{e}) - 1$, not counting any precomputation.

For 512-bit exponents, the optimal window size is $k = 5$ for both the TKM and SW methods, yielding an average multiplication cost of 127.8 and 100.3 respectively. Thus on average the optimal sliding window method only performs about 78% of the multiplies that the optimal 2^k-ary method performs. From Tables 1 and 2, since $TMK(k) \leq 127.8 + 19 = 146.8$ over 99% of the time, and $SW(k) \geq 100.3 - 23 = 77.3$ over 99% of the time, the optimal sliding window method will perform over 52% of the multiplications required by the optimal 2^k-ary method for most exponents. For the majority of exponents $SW(k) \leq 100.3 + 4 = 104.3$, while the majority of exponents require $TMK(k) \geq 127.8 - 3 = 124.8$ multiplications, meaning that the optimal sliding widow method requires less than 84% of the multiplications required by the optimal 2^k-ary method for a majority of exponents. Further, from Table 3 we find that over 90% of OSD exponents require at least $170 - 20 = 150$ multiplications, implying that the optimal sliding window only performs less than 70% of the multiplication that OSD requires.

Similarly, the optimal sliding window method is superior to the optimal 2^k-ary method for 1024-bit exponents, as it is to the OSD method. In this case the optimal window size for the sliding window method is $k = 6$, while it is $k = 5$ for the 2^k-ary method. Again from Tables 1 and 2, for the majority of exponents $SW(k) \leq 177.3 + 4 = 181.3$, while the majority of exponents have $TMK(k) \geq 226.6 - 4 = 224.6$, meaning that the optimal sliding widow method

requires approximately 80% of the multiplications required by the optimal 2^k-ary method for a majority of exponents. With high probability the optimal sliding window method performs at least 60% of the multiplications required by the optimal 2^k-ary method, and with almost certainty performs less than 60% of the multiplications required by the OSD method.

Even more accurate statements can be made if the generating functions are expanded, and probabilities computed directly. In the case of 512-bit exponents and $k = 5$ bit windows, both the sliding window and 2^k-ary methods deviate from their expected weights by more than ± 10 with probability less than 10^{-4}. Further the majority of exponents deviate by less than ± 1 from their expected weight.

OSD recoded exponents tend to a weight of approximately $n/3$ on average. This weight cannot be significantly reduced since smaller weight exponents depend on longer runs of 1's occuring in the original exponent, but a run of length k has probability 2^{-k}. One advantage of the OSD coding is that little space is required for precomputation, and if inverses can be computed quickly then the OSD method may be attactive, say for elliptic curve computations on smart cards.

7 Appendix

Proof of Theorem 9.
Let $e = e_0 e_1 \cdots e_{n-2} e_{n-1}$, $e = \sum_{i=0}^{n-1} e_i 2^i$, be an n-bit exponent, written left-to-right as low order to high order bits. OSD-recoding can be interpreted as initially partitioning an exponent e into blocks

$$e = b_1 0^{j_1} b_2 0^{j_2} \cdots 0^{j_{t-2}} b_{t-1} 0^{j_t} b_t, \tag{17}$$

where $j_d \geq 0$, $1 \leq d \leq t$. Each b_i, $1 \leq i < t$, consists of runs separated by a single zero, where the last run ends in two zeros, which for example might be 100 or 1011100. Also b_t is similar except that the last run is followed by either one or no zeroes. Note that since the b_i and b_{i+1} are separated by at least two zeroes then the recoding of b_i and b_{i+1} according to Figure 1 will be independent.

The regular expression $(1^+0)^*0$ generates words containing runs separated by a single zero, where the last run ends in two zeros. The next step is to determine if the trailing pair of zeros after $b_i, 1 \leq i < t$, is encoded as 10 or 00. In the first case we will say that a *carry* has propagated to the second most significant zero, or more simply, that a carry is present b_i. The main observation is that a carry will be present if and only if b_i contains 110. We define the following two regular expressions to detect the presence of a carry:

$$R_1 = 10(10)^*0,$$
$$R_2 = ((10)^+11^+0 + 11^+0)(1^+0)^*0,$$

Here R_1 generates words with no carry, and R_2 generates words with carry (110 is present). Thus $R_3 = (0 + R_1 + R_2)^*$ generates all blocks in (17) except b_t.

Note that the OSD-encoding of each b_i is length preserving and if $\#b_i = k$ the it is enumerated as $z^m x^k$ where $m = \#(\text{runs in } b_i) + [110 \text{ is present in } b_i]$. The gfs for R_1 and R_2 can be derived directly as

$$G_{L_1}(x,z) = \frac{zx^3}{1-zx^2},$$
$$G_{L_2}(x,z) = \frac{zx^3}{1-x}\left(1+\frac{zx^2}{1-zx^2}\right)\frac{1}{1-(zx^2)/(1-x)}\cdot xz.$$

and $G_{L_3}(x,z) = 1/(1-x-G_{L_1}(x,z)-G_{L_2}(x,z))$. It remains to enumerate block b_t which is generated by the regular expression $R_4 = (1^+0)^*1^*$. Expanding R_4 so that it can be marked for length and weight we obtain

$$R_4 = (10)^*(\epsilon + 1 + 11) + 110^+(1^+0)^*1^* + (10)^+11^+0(1^+0)^*1^\cdot$$

R_4 is similar to R_3, and $G_{L_4}(x,z)$ is derived in a manner similar to $G_{L_3}(x,z)$. The theorem follows from simplifying $G_{L_4}(x,z)G_{L_6}(x,z)$.

□

References

1. See the Maple homepage at http://www.maplesoft.com.
2. S. Arno and F. Wheeler. Signed digit representations of minimal hamming weight. *IEEE Transactions on Computers*, 42(8):1007–1010, 1993.
3. E. A. Bender and S. G. Williamson. *Foundations of Applied Combinatorics.* Addison-Wesley Publishing Company, 1991.
4. J. Bos and M. Coster. Addition chain heuristics. *Advances in Cryptology, CRYPTO 89, Lecture Notes in Computer Science, vol. 218, G. Brassard ed., Springer-Verlag*, pages 400–407, 1990.
5. N. Chomsky and P. Schutzenberger. The algebraic theory of context-free languages. In P Braffort and North Holland Hirchberg, D., editors, *Computer programming and formal languages*, pages 118–161, 1963.
6. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):472–492, 1976.
7. T. ElGamal. A public key cryptosystem and signature system based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):473–481, 1985.
8. D. Gollman, Y. Han, and C. Mitchell. Redundant integer representations and fast exponentiation. *Designs, Codes and Cryptography*, 7:135–151, 1996.
9. R. L. Graham, D. E. Knuth, and O. Patshnik. *Concrete Mathematics, A Foundation for Computer Science, First Edition.* Addison Wesley, 1989.
10. J. Hopcroft and J. Ullman. *An Introduction to Automata, Languages and Computation.* Reading, MA: Addison Wesley, 1979.
11. L. Hui and K.-Y. Lam. Fast square-and-multiply exponentiation for RSA. *Electronics Letters*, 30(17):1396–1397, 1994.
12. J. Jedwab and C. Mitchell. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, 25:1171–1172, 1989.

13. C. K. Koç. High-radix and bit encoding techniques for modular exponentiation. *International Journal of Computer Mathematics*, 40:139–156, 1991.
14. C. K. Koç. Analysis of sliding window techniques for exponentiation. *Computers and Mathematics with Applications*, 30(10):17–24, 1995.
15. D. E. Knuth. *The Art of Computer Programming : Volume 2, Seminumerical Algorithms*. Addsion Wesley, 1981.
16. N. Koblitz. CM curves with good cryptographic properties. *Advances in Cryptology, CRYPTO 91, Lecture Notes in Computer Science, vol. 576, J. Feigenbaum ed., Springer-Verlag*, pages 279–287, 1992.
17. K. Koyama and T. Tsuruoka. Speeding up elliptic curve cryptosystems using a signed binary window method. In *Advances in Cryptology, CRYPTO 92, Lecture Notes in Computer Science, vol. 740, E. F. Brickell ed., Springer-Verlag*, pages 345–357, 1992.
18. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.
19. F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoretical Informatics and Applications*, 24(6):531–544, 1990.
20. G. Reitwiesener. Binary arithmetic. In F. L. Alt, editor, *Advances in Computers*, pages 232–308, 1960.
21. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
22. R Sedgewick and P. Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley Publishing Company, 1996.
23. J. A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. *Advances in Cryptology, CRYPTO 97, Lecture Notes in Computer Science, vol. 1294, B. S. Kaliski ed., Springer-Verlag*, pages 357–371, 1997.
24. Y. Yacobi. Exponentiating faster with addition chains. *Advances in Cryptology, EUROCRYPT 90, Lecture Notes in Computer Science, vol. 473, I. B. Damgård ed., Springer-Verlag*, pages 222–229, 1991.