Efficient Algorithms for Computing Modular Exponentiation

D. J. Guan
Department of Computer Science
National Sun Yat-Sen University
Kaohsiung, Taiwan 80424
guan@cse.nsysu.edu.tw

November 25, 2007

Abstract

Computing $a^x \bmod n$ and $a^x b^y \bmod n$ for very large x, y, and n are fundamental to the efficiency of almost all public key cryptosystems and digital signature schemes. Efficient algorithms for computing $a^x \bmod n$ and $a^x b^y \bmod n$, are presented in this talk. All algorithms are simple and only a small number of additional spaces are required to store the precomputed values. These algorithms are suitable for devices with low computational power and limited storage, such as smart cards.

Algorithm for Computing $a^x \bmod n$

Input: a, x, and n
Output: $c = a^x \bmod n$

---

let $x = (x_{m-1}, x_{m-2}, \ldots, x_0)_2$;
$c = 1$;
for $(i = m-1;\ i \geq 0;\ i = i-1)$ {
  $c = c^2 \bmod n$;
  if $(x_i \neq 0)$ $c = (c \times a) \bmod n$;
}
print c;

An Example: $c = a^{149} \bmod n$

$x = 149$	1	0	0	1	0	1	0	1
$\times$	a	-	-	a	-	a	-	a
c	a	a^2	a^4	a^9	a^{18}	a^{37}	a^{74}	a^{149}

1. squaring ($c^2 \bmod n$):
   $m = \lfloor \log x \rfloor + 1$, the size of x.

2. multiplication ($c \times a \bmod n$):
   $h(x)$, number of 1's in the binary code of x.

Algorithm for Computing $a^x b^y \bmod n$

Input: a, x, b, y, and n
Output: $c = a^x b^y \bmod n$

---

let $x = (x_{m-1}, x_{m-2}, \ldots, x_0)_2$;
let $y = (y_{m-1}, y_{m-2}, \ldots, y_0)_2$;
compute and store the value of $a \times b \bmod n$;
$c = 1$;
for $(i = m - 1;\ i \geq 0;\ i = i - 1)$ $\{$
  $c = c^2 \bmod n$;
  if $(x_i \neq 0$ or $y_i \neq 0)$ $c = (c \times a^{x_i} b^{y_i}) \bmod n$;
$\}$
print c;

An Example: $c = a^{149} b^{170} \bmod n$

$x = 149$	1	0	0	1	0	1	0	1
$y = 170$	1	0	1	0	1	0	1	0
$\times$	ab	-	b	a	b	a	b	a
c	ab	a^2b^2	a^4b^5	a^9b^{10}	$a^{18}b^{21}$	$a^{37}b^{42}$	$a^{74}b^{85}$	$a^{149}b^{170}$

1. squaring:
   $m = \lfloor \log x \rfloor + 1$
2. multiplication:
   $h(x, y)$, the joint Hamming weight of x and y.

Reduce Computational Cost

Assume that the best known algorithm has been used in computing the group multiplications.

The algorithm needs

1. $m = \lfloor \log x \rfloor + 1$ squarings
2. $h(x)$ or $h(x, y)$ multiplications.

The goal is to reduce the number of multiplications.

Methods to Reduce $h(x)$ and $h(x, y)$

1. a^x mod n
   Try to use different codes with smaller $h(x)$.
   For example, use *binary-signed-digit* code, instead of *binary* code.

2. $a^x b^y$ mod n
   Try to *align more non-zero digits* by choosing different binary-signed-digit codes.
   For example, (1) DJM method, and (2) Solinas' method.

Signed-digit Code for the Exponents

Binary code for an integer is unique.

Encode the exponents with $\{-1, 0, 1\}$, instead of $\{0, 1\}$, can reduce the number of non-zero bits.

For example, $15 = (01111)_2 = (1000\bar{1})_{\bar{2}}$.

Computing a^x mod n

In binary code, the *average Hamming weight* of x, $\bar{h}(x) = m/2$, where $m = \lfloor \log x \rfloor + 1$ is the number of bits in the binary code of x.

The signed-digit code of an integer is not unique.

There are algorithms for computing *optimal signed-digit code*, a code with minimum weight, of x.

It can be shown that the average Hamming weight for optimal sign-digit code of x is $\bar{h}(x) = m/3$.

However, inverses are required if the exponents are coded by $\{-1, 0, 1\}$.

Therefore, it is feasible only when the inverses have been precomputed, or when computing inverses are almost free, e. g., the additive group generated by an elliptic curve.

Computing $a^x b^y$ mod n

Assume that both x and y are coded in optimal signed-digit code.

The probability of $x_i \neq 0$ or $y_i \neq 0$ is $(1-(2/3)^2)m = (5/9)m \approx 0.556m$.

In 2000, Dimitrov, Jullien, and Miller proposed rules to re-code the exponents x and y into x' and y'.

x	010	010	$0\bar{1}0$	$0\bar{1}0$	$10\bar{1}$	$10\bar{1}$	$\bar{1}01$	$\bar{1}01$
y	$10\bar{1}$	$\bar{1}01$	$10\bar{1}$	$\bar{1}01$	010	$0\bar{1}0$	010	$0\bar{1}0$
x'	010	010	$0\bar{1}0$	$0\bar{1}0$	011	011	$0\bar{1}\bar{1}$	$0\bar{1}\bar{1}$
y'	011	$0\bar{1}\bar{1}$	011	$0\bar{1}\bar{1}$	010	$0\bar{1}0$	010	$0\bar{1}0$

They showed that the average joint Hamming weight after re-coding is $\bar{h}(x', y') \approx 0.534m$.

Codes with Minimum Joint Hamming Weight

In 2001, Solinas defined *joint sparse* form for x and y.

1. In any 3 consecutive columns of the bits of x and y,
   $\begin{matrix} x_i & x_{i-1} & x_{i-2} \\ y_i & y_{i-1} & y_{i-2} \end{matrix}$, at least one of them is $\begin{matrix} 0 \\ 0 \end{matrix}$.
2. Adjacent bits do not have opposite signs:
   $x_i x_{i-1} \neq -1, \quad y_i y_{i-1} \neq -1.$
3. If $x_i x_{i-1} \neq 0$, then $y_i \neq 0$ and $y_{i-1} = 0$.
   If $y_i y_{i-1} \neq 0$, then $x_i \neq 0$ and $x_{i-1} = 0$.

The Optimality of Join Space From

Solinas presented algorithms for converting pair of integers into joint sparse form.

They show that the average joint Hamming weight of two integers in joint sparse form $\bar{h}(x', y') \to 0.5m$ as $m \to \infty$.

They also show that this is *optimal*, no other codes can do better.

New Strategy for Computing $a^x \bmod n$ and $a^x b^y \bmod n$

Try to match nearby non-zero bits and do the multiplication together.

1. delay method

2. pairing method

3. block method

Delay Method

$x = 149$	1	0	0	1	0	1	0	1
	|			/		/		/
$y = 170$	1	0	1	0	1	0	1	0
$\times$	ab	-	-	ab^2	-	ab^2	-	ab^2
c	ab	a^2b^2	a^4b^4	a^9b^{10}	$a^{18}b^{20}$	$a^{37}b^{42}$	$a^{74}b^{84}$	$a^{149}b^{170}$

When a non-zero bit of y is matched to a less significant non-zero bit of x the multiplication is delayed.

In addition to ab, we need to pre-compute and store the value of ab^2.

In the above example, only 4 multiplications, instead of 7.

Finite State Machine Representation

1. S_0: normal state, multiply by ab.
2. S_1: shift right by 1, multiply by ab^2.

1. $x_i \sim y_j$: $x_i = 0$ and $y_j = 0$ or $x_i \neq 0$ and $y_j \neq 0$.
2. $x_i \not\sim y_j$: $x_i = 0$ and $y_j \neq 0$ or $x_i \neq 0$ and $y_j = 0$.

The transition function δ is

	$x_i \not\sim y_i$, $x_i \sim y_{i+1}$	other cases
S_0	S_1	S_0
S_1	S_1	S_0

Performance of the Delay Multiplication

Signed-Digit	$0.556m$	$0.444m$
DJM code	$0.534m$	$0.453m$
Joint Sparse Form	$0.500m$	$0.469m$

1. The new method reduces the computational cost for all codes.

2. It performs even better than the joint space form.

3. Applying to *signed-digit* code produces the best results.

4. Our algorithm needs to pre-compute and store ab^2 mod n.

Extend to Both x and y

1. S_0: normal state, multiply by ab.
2. S_1: shift right by 1, multiply by ab^2.
3. S_{-1}: shift left by 1, multiply by a^2b.

code	no shift	shift left or right	shift left and right
sparse signed-digit	$0.556m$	$0.444m$	$0.407m$
DJM code	$0.534m$	$0.453m$	$0.414m$
Joint Sparse Form	$0.500m$	$0.469m$	$0.438m$

1. Applying to *signed-digit* code produces the best results.
2. The algorithm needs to pre-compute and store $ab^2 \bmod n$ and a^2b.

Apply to Binary Code

In integer multiplication modulo n, $\mathbf{Z}_n^*$, the computational complexity of computing an inverse is the same as computing exponentiation.

If the inverse is not "easy" to obtain, binary code is the only practical code.

	shift 0	shift 1	shift 2	$\cdots$	shift d
$\bar{h}(x)$	$0.750m$	$0.667m$	$0.636m$	$\cdots$	$\left(\dfrac{3^d - 2^{d-1}}{5 \cdot 3^{d-1} - 2^d}\right) m$
space	1	2	3	$\cdots$	$d + 1$

The *space* is the additional spaces for the precomputed values.

$d = 1$: ab, ab^2

$d = 2$: ab, ab^2, ab^4

Paring Method

Another method to reduce the computational cost is to pair nearby non-zero bits, and do the multiplication for these bits together.

$x = 161$	1	0	1	0	0	0	0	1
$y = 138$	1	0	0	0	1	0	1	0
$\times$	ab	-	-	-	a^4b	-	-	ab^2
c	ab	a^2b^2	a^4b^4	a^8b^8	$a^{20}b^{17}$	$a^{40}b^{34}$	$a^{80}b^{68}$	$a^{161}b^{138}$

Paring Method for $a^x b^y \bmod n$

Exponents x and y are in optimal signed-digit code.

Number of bits: 1024; Sample size: 1000

d: to pair bits which are at most d bits apart.

	$d=0$	$d=1$	$d=2$	$d=3$	$d=4$
$\bar{h}(x,y)$	$0.556m$	$0.408m$	$0.378m$	$0.365m$	$0.358m$
space	2	6	10	14	18

$d=0$: ab, ab^{-1} (All the inverses are not stored).

$d=1$: ab, ab^2, ab^{-1}, ab^{-2}, a^2b, a^2b^{-1},

Additional spaces: $4(2d+1)/2 = 4d+2$

Paring Method for $a^x \bmod n$

Exponent x is in optimal signed-digit code.

Number of bits: 1024; Sample size: 1000

d: to pair bits which are at most d bits apart.

	$d=0$	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$
$\bar{h}(x)$	$0.333m$	$0.333m$	$0.222m$	$0.191m$	$0.178m$	$0.172m$
space	0	1	2	3	4	5

$d=1$: (There are no adjacent non-zero bits in optimal sign-digit code.)

$d=2$: a^3, a^5

$d=3$: a^3, a^5, a^7

Paring Method for $a^x b^y \bmod n$

Exponents x and y are in binary code.

Number of bits: 1024;   Sample size: 1000

d: to pair bits which are at most d bits apart.

	$d = 0$	$d = 1$	$d = 2$	$d = 3$	$d = 4$
$\bar{h}(x, y)$	$0.750m$	$0.625m$	$0.583m$	$0.562m$	$0.550m$
space	1	5	7	9	11

$d = 1$: ab, ab^2, a^2b

$d = 2$: ab, ab^2, ab^4, a^2b, a^4b

additional space: $2d + 1$

Block Method

Let $d > 0$. $x_i, x_{i-1}, \ldots, x_{i-d}$ is a block if both x_i and x_{i-d} are non-zero, but all other bits in the block, $x_{i-1} \cdots x_{i-d+1}$, are zero.

Similarly, $\begin{matrix} x_i, x_{i-1}, \ldots, x_{i-d} \\ y_i, y_{i-1}, \ldots, y_{i-d} \end{matrix}$ is a block if

1. x_i and y_i are not all zero.
2. x_{i-d} and y_{i-d} are not all zero.
3. all other bits in the block are zero.

There will be only *one* multiplication in each block.

Block Method for computing $a^x \bmod n$

Input: a, x, and n
Output: $c = a^x \bmod n$

```
let x = (x_{m-1}, x_{m-2}, ..., x_0)_2;
c = 1;
i = m - 1;
while (i >= 0) {
    c = c^2 mod n;
    if ((x_i != 0) and (x_{i-1} != 0)) {
        c = c^2 × a^3 mod n;
        i = i - 1;
    } else if (x_i != 0) {
        c = (c × a) mod n;
    }
    i = i - 1;
}
print c;
```

Block Method for $a^x \bmod n$, $d = 1$

$m = 1$	$m = 2$	$m = 3$	$m = 4$	
0 (0)	00 (0)	000 (0)	0000 (0)	1000 (1)
1 (1)	01 (1)	001 (1)	0001 (1)	1001 (2)
	10 (1)	010 (1)	0010 (1)	1010 (2)
	11 (1)	011 (1)	0011 (1)	1011 (2)
		100 (1)	0100 (1)	1100 (1)
		101 (2)	0101 (2)	1101 (2)
		110 (1)	0110 (1)	1110 (2)
		111 (2)	0111 (2)	1111 (2)
$h_1(1) = 1/2$	$h_1(2) = 3/4$	$h_1(3) = 9/8$	$h_1(4) = 23/16$	

Theorem 1 *The average number of multiplications for computing* a^x mod n *by using block method with* $d = 1$ *is*

$$h_1(m) = \frac{1}{3}m + \frac{1}{9} - \frac{1}{9}\left(-\frac{1}{2}\right)^m .$$

$0(0+1)^{m-1}$	$1/2$	$f_1(m-1)$
$10(0+1)^{m-2}$	$1/4$	$f_1(m-2)+1$
$11(0+1)^{m-2}$	$1/4$	$f_1(m-2)+1$

$$f_1(m) = \frac{1}{2}f_1(m-1) + \frac{1}{2}f_1(m-2) + \frac{1}{2}, \quad m > 0$$

Solve the Recurrence Equation by Generating Function, $d = 1$

Define $f_1(m) = 0$ for $m \le 0$.

Let $G(z) = \sum_m f_1(m) z^m$.

Rewrite the equation for $m \ge 0$.

$$f_1(m) = \frac{1}{2} f_1(m-1) + \frac{1}{2} f_1(m-2) + \frac{1}{2} - \frac{1}{2}[m=0]$$

$$[m=0] = \begin{cases} 1 & \text{if } m = 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$2f_1(m) = f_1(m-1) + f_1(m-2) + 1 - [m=0]$$

$$2\sum_m f_1(m) z^m = \sum_m f_1(m-1) z^m + \sum_m f_1(m-2) z^m + \sum_m z^m - \sum_m [m=0] z^m$$

$$2G(z) = zG(z) + z^2 G(z) + \frac{1}{1-z} - 1$$

$$G(z) = \frac{z}{(1-z)(2-z-z^2)} = \frac{(1/3)}{(1-z)^2} + \frac{-(2/9)}{(1-z)} + \frac{-(2/9)}{(2+z)}$$

$$f_1(m) = \frac{m+1}{3} - \frac{2}{9} - \frac{1}{9}\left(-\frac{1}{2}\right)^m = \frac{1}{3}m + \frac{1}{9} - \frac{1}{9}\left(-\frac{1}{2}\right)^m.$$

Block Method for $a^x \bmod n$, $d = 2$

$m = 1$	$m = 2$	$m = 3$	$m = 4$	
0 (0)	00 (0)	000 (0)	0000 (0)	1000 (1)
1 (1)	01 (1)	001 (1)	0001 (1)	1001 (2)
	10 (1)	010 (1)	0010 (1)	1010 (1)
	11 (1)	011 (1)	0011 (1)	1011 (2)
		100 (1)	0100 (1)	1100 (1)
		101 (1)	0101 (1)	1101 (2)
		110 (1)	0110 (1)	1110 (2)
		111 (2)	0111 (2)	1111 (2)
$h_2(1) = 1/2$	$h_2(2) = 3/4$	$h_2(3) = 1$	$h_2(4) = 21/16$	

Block Method for $a^x \bmod n$, $d = 2$

Theorem 2 *The average number of multiplications for computing* $a^x \bmod n$ *by using block method with* $d = 2$ *is*

$$h_2(m) = \frac{2}{7}m + \frac{20}{49} + o(1).$$

$0(0+1)^{m-1}$	$1/2$	$f_2(m-1)$
$11(0+1)^{m-2}$	$1/4$	$f_2(m-2)+1$
$101(0+1)^{m-3}$	$1/8$	$f_2(m-3)+1$
$100(0+1)^{m-3}$	$1/8$	$f_2(m-3)+1$

$$f_2(m) = \frac{1}{2}f_2(m-1) + \frac{1}{4}f_2(m-2) + \frac{1}{4}f_2(m-3) + \frac{1}{2}, \quad m > 0$$

Solve the Recurrence Equation by Generating Function, $d = 2$

Define $f_2(m) = 0$ for $m \leq 0$.

Let $G(z) = \sum_{m} f_2(m) z^m$.

Rewrite the equation for $m \geq 0$.

$$f_2(m) = \frac{1}{2} f_2(m-1) + \frac{1}{4} f_2(m-2) + \frac{1}{4} f_2(m-3) + \frac{1}{2} - \frac{1}{2}[m = 0]$$

$$4f_2(m) = 2f_2(m-1) + f_2(m-2) + f_2(m-3) + 2 - 2[m=0]$$

$$4\sum_m f_2(m)z^m = 2\sum_m f_2(m-1)z^m + \sum_m f_2(m-2)z^m + \sum_m f_2(m-3) + 2\sum_m z^m - 2\sum_m [m=0]z^m$$

$$4G(z) = 2zG(z) + z^2G(z) + z^3G(z) + \frac{2}{1-z} - 2$$

$$G(z) = \frac{2z}{(1-z)(4-2z-z^2-z^3)} = \frac{2/7}{(1-z)^2} + \frac{6/49}{(1-z)} + \frac{-(1/49)(32+6z)}{(4+2z+z^2)}$$

$$f_2(m) = \frac{2(m+1)}{7} + \frac{6}{49} + o(1) = \frac{2}{7}m + \frac{20}{49} + o(1).$$

Block Method for $a^x \bmod n$, $d = 3$

$m=1$	$m=2$	$m=3$	$m=4$	
0 (0)	00 (0)	000 (0)	0000 (0)	1000 (1)
1 (1)	01 (1)	001 (1)	0001 (1)	1001 (1)
	10 (1)	010 (1)	0010 (1)	1010 (1)
	11 (1)	011 (1)	0011 (1)	1011 (2)
		100 (1)	0100 (1)	1100 (1)
		101 (1)	0101 (1)	1101 (2)
		110 (1)	0110 (1)	1110 (2)
		111 (2)	0111 (2)	1111 (2)
$h_3(1) = 1/2$	$h_3(2) = 3/4$	$h_3(3) = 1$	$h_3(4) = 5/4$	

Block Method for a^x mod n, $d = 3$

Theorem 3 *The average number of multiplications for computing* a^x mod n *by using block method with* $d = 3$ *is*

$$h_3(m) = \frac{4}{15}m + \frac{44}{225} + o(1).$$

$0(0+1)^{m-1}$	$1/2$	$f_3(m-1)$
$11(0+1)^{m-2}$	$1/4$	$f_3(m-2)+1$
$101(0+1)^{m-3}$	$1/8$	$f_3(m-3)+1$
$1001(0+1)^{m-4}$	$1/16$	$f_3(m-4)+1$
$1000(0+1)^{m-4}$	$1/16$	$f_3(m-4)+1$

$$f_3(m) = \frac{1}{2}f_3(m-1)+\frac{1}{4}f_3(m-2)+\frac{1}{8}f_3(m-3)+\frac{1}{8}f_3(m-4)+\frac{1}{2}, \quad m > 0$$

Solve the Recurrence Equation by Generating Function, $d = 3$

Define $f_3(m) = 0$ for $m \leq 0$.

Let $G(z) = \sum_m f_3(m) z^m$.

Rewrite the equation for $m \geq 0$.

$$f_3(m) = \frac{1}{2} f_3(m-1) + \frac{1}{4} f_3(m-2) + \frac{1}{8} f_3(m-3) + \frac{1}{8} f_3(m-4) + \frac{1}{2} - \frac{1}{2}[m = 0]$$

$$8f_3(m) = 4f_3(m-1) + 2f_3(m-2) + f_3(m-3) + f_3(m-4) + 4 - 4[m=0]$$

$$8\sum_m f_3(m)z^m \;=\; 4\sum_m f_3(m-1)z^m + 2\sum_m f_3(m-2)z^m + \sum_m f_3(m-3) + \sum_m f_3(m-4)z^m + 4\sum_m z^m - 4\sum_m [m=0]z^m$$

$$8G(z) = 4zG(z) + 2z^2G(z) + z^3G(z) + z^4G(z) + \frac{4}{1-z} - 4$$

$$G(z) = \frac{4z}{(1-z)(8-4z-2z^2-z^3-z^4)} = \frac{4/15}{(1-z)^2} + \frac{-16/225}{(1-z)} + \frac{-(1/9)}{(2+z)} + \frac{-(1/25)(z-14)}{(4+z^2)}$$

$$f_3(m) = \frac{4(m+1)}{15} - \frac{16}{225} + o(1) = \frac{4}{15}m + \frac{44}{225} + o(1).$$

Block Method for $a^x \bmod n$, $d > 0$

$0(0+1)^{m-1}$	$1/2$	$f_d(m-1)$
$11(0+1)^{m-2}$	$1/2^2$	$f_d(m-2)+1$
$101(0+1)^{m-3}$	$1/2^3$	$f_d(m-3)+1$
$\vdots$	$\vdots$	$\vdots$
$10^{d-1}1(0+1)^{m-d}$	$1/2^d$	$f_d(m-d-1)+1$
$10^{d-1}0(0+1)^{m-d}$	$1/2^d$	$f_d(m-d-1)+1$

$$f_d(m) = \sum_{i=1}^{d} \frac{1}{2^i} f_d(m-i) + \frac{1}{2^d} f_d(m-d-1) + \frac{1}{2}, \quad m > 0$$

Solve the Recurrence Equation by Generating Function, $d > 0$

Define $f_d(m) = 0$ for $m \leq 0$.

Let $G(z) = \sum_m f_d(m) z^m$.

Rewrite the equation for $m \geq 0$.

$$f_d(m) = \sum_{i=1}^{d} \frac{1}{2^i} f_d(m-i) + \frac{1}{2^d} f_d(m-d-1) + \frac{1}{2} - \frac{1}{2}[m=0]$$

$$2^d f_d(m) = \sum_{i=1}^{d} 2^{d-i} f_d(m-i) + f_d(m-d-1) + 2^{d-1} - 2^{d-1}[m=0]$$

$$\sum_m 2^d f_d(m) z^m = \sum_m \sum_{i=1}^{d} 2^{d-i} f_d(m-i) z^m + \sum_m f_d(m-d-1) z^m + \sum_m 2^{d-1} z^m - \sum_m 2^{d-1} [m=0] z^m$$

$$2^d G(z) = \sum_{i=1}^{d} 2^{d-i} z^i G(z) + z^{d+1} G(z) + 2^{d-1} \left(\frac{z}{1-z} \right)$$

$$G(z) = \frac{2^{d-1} z}{(1-z)\left(2^d - \left(\sum_{i=1}^{d} 2^{d-i} z^i\right) - z^{d+1}\right)}$$

$$G(z) = \frac{z(1-(z/2))}{2(1-z)^2 \left(1-(z/2)^{d+1}\right)}$$

$$f_d(m) = \left(\frac{2^{d-1}}{2^{d+1}-1} \right) m + o(1).$$

Block Method for $a^x b^y \bmod n$**,** $d = 1$

Theorem 4 *The average number of multiplications for computing* $a^x b^y$ mod n *by using block method with* $d = 1$ *is*

$$g_1(m) = \frac{3}{7}m + \frac{9}{49} + \frac{9}{49}\left(-\frac{3}{4}\right)^m.$$

$0(0+1)^{m-1}$ $0(0+1)^{m-1}$	$1/4$	$g_1(m-1)$
$a_1a_2(0+1)^{m-1}$ $b_1b_2(0+1)^{m-1}$	$3/4$	$g_1(m-2)+1$

$$g_1(m) = \frac{1}{4}g_1(m-1) + \frac{3}{4}g_1(m-2) + \frac{3}{4}, \quad m > 0$$

Possible values of $\begin{matrix} a_1 a_2 \\ b_1 b_2 \end{matrix}$ are

		10 00	11 00
		10 01	11 01
00 10	01 10	10 10	11 10
00 11	01 11	10 11	11 11

$$\frac{12}{16} = \frac{3}{4}$$

Use generating function to solve the recurrence equation

$$g_1(m) = \frac{1}{4}g_1(m-1) + \frac{3}{4}g_1(m-2) + \frac{3}{4} - \frac{3}{4}[m=0].$$

$$G(z) = \frac{3z}{(1-z)(4-z-3z^2)} = \frac{3}{7(1-z)^2} - \frac{12}{49(1-z)} - \frac{36}{49(4+3z)}.$$

Therefore, $g_1(m) = \frac{3}{7}(m+1) - \frac{12}{49} - \frac{9}{49}\left(-\frac{3}{4}\right)^m = \frac{3}{7}m + \frac{9}{49} - \frac{9}{49}\left(-\frac{3}{4}\right)^m.$

Block Method for $a^x b^y \bmod n$, $d = 2$

Theorem 5 *The average number of multiplications for computing* $a^x b^y$ mod n *by using block method with* $d = 2$ *is*

$$g_2(m) = \frac{12}{31}m + \frac{216}{961} + o(1).$$

$0(0+1)^{m-1}$ $0(0+1)^{m-1}$	1/4	$g_2(m-1)$
$a_1 a_2 (0+1)^{m-1}$ $b_1 b_2 (0+1)^{m-1}$	9/16	$g_2(m-2)+1$
$c_1 0 c_2 (0+1)^{m-1}$ $d_1 0 d_2 (0+1)^{m-1}$	3/16	$g_2(m-3)+1$

$$g_2(m) = \frac{1}{4}g_2(m-1) + \frac{9}{16}g_2(m-2) + \frac{3}{16}g_2(m-3) + \frac{3}{4}.$$

Possible values of $\begin{matrix} a_1a_2 \\ b_1b_2 \end{matrix}$ are

			11 00
		10 01	11 01
	01 10		11 10
00 11	01 11	10 11	11 11

$$\frac{9}{16}$$

Possible values of $\begin{matrix} c_1 0 c_2 \\ d_1 0 d_2 \end{matrix}$ are

		100 000	101 000
		100 001	101 001
000 100	001 100	100 100	101 100
000 101	001 101	100 101	101 101

$$\frac{12}{64} = \frac{3}{16}$$

Use generating function to solve the recurrence equation

$$g_2(m) = \frac{1}{4}g_2(m-1) + \frac{9}{16}g_2(m-2) + \frac{3}{16}g_2(m-3) + \frac{3}{4} - \frac{3}{4}[m=0].$$

$$G(z) = \frac{12z}{(1-z)(16-4z-9z^2-3z^3)}$$
$$= \frac{12}{31(1-z)^2} - \frac{156}{961(1-z)} - \frac{3456+468z}{961(16+12z+3z^2)}.$$

Therefore, $g_2(m) = \frac{12}{31}(m+1) - \frac{156}{961} + o(1) = \frac{12}{31}m + \frac{216}{961} + o(1).$

Block Method for $a^x b^y \bmod n$, $d = 3$

Theorem 6 *The average number of multiplications for computing* $a^x b^y$ mod n *by using block method with* $d = 3$ *is*

$$g_3(m) = \frac{48}{127}m + \frac{3888}{16129} + o(1).$$

$0(0+1)^{m-1}$ $0(0+1)^{m-1}$	1/4	$g_3(m-1)$
$a_1a_2(0+1)^{m-1}$ $b_1b_2(0+1)^{m-1}$	9/16	$g_3(m-2)+1$
$c_10c_2(0+1)^{m-1}$ $d_10d_2(0+1)^{m-1}$	9/64	$g_3(m-3)+1$
$e_100e_2(0+1)^{m-1}$ $f_100f_2(0+1)^{m-1}$	3/64	$g_3(m-3)+1$

$$g_3(m) = \frac{1}{4}g_3(m-1) + \frac{9}{16}g_3(m-2) + \frac{9}{64}g_3(m-3) + \frac{3}{64}g_3(m-4) + \frac{3}{4}.$$

Possible values of $\begin{matrix} a_1 a_2 \\ b_1 b_2 \end{matrix}$ are

			11 00
		10 01	11 01
	01 10		11 10
00 11	01 11	10 11	11 11

$$\frac{9}{16}$$

Possible values of $\begin{array}{c} c_1 0 c_2 \\ d_1 0 d_2 \end{array}$ are

			101 000
		100 001	101 001
	001 100		101 100
000 101	001 101	100 101	101 101

$$\frac{9}{64}$$

Possible values of $\begin{matrix} e_1 00 e_2 \\ f_1 00 f_2 \end{matrix}$ are

		1000 0000	1001 0000
		1000 0001	1001 0001
0000 1000	0001 1000	1000 1000	1001 1000
0000 1001	0001 1001	1000 1001	1001 1001

$$\frac{12}{256} = \frac{3}{64}$$

Use generating function to solve the recurrence equation

$g_3(m) = \frac{1}{4}g_3(m-1) + \frac{9}{16}g_3(m-2) + \frac{9}{64}g_3(m-3) + \frac{3}{64}g_3(m-4) + \frac{3}{4} - \frac{3}{4}[m = 0]$.

$$G(z) = \frac{48z}{(1-z)(64-16z-36z^2-9z^3-3z^4)}$$

$$= \frac{48}{127(1-z)^2} - \frac{2208}{16129(1-z)} - \frac{248832+51408z+6624z^2}{16129(64+48z+12z^2+3z^3)}.$$

Therefore, $g_3(m) = \frac{48}{127}(m+1) - \frac{2208}{16129} + o(1) = \frac{48}{127}m + \frac{3888}{16129} + o(1)$.

Block Method for $a^x b^y \bmod n$, $d > 0$

$0(0+1)^{m-1}$ $0(0+1)^{m-1}$	$1/4$	$g_3(m-1)$
$a_1 a_2 (0+1)^{m-1}$ $b_1 b_2 (0+1)^{m-1}$	$9/4^2$	$g_3(m-2)+1$
$c_1 0 c_2 (0+1)^{m-1}$ $d_1 0 d_2 (0+1)^{m-1}$	$9/4^3$	$g_3(m-3)+1$
$e_1 00 e_2 (0+1)^{m-1}$ $f_1 00 f_2 (0+1)^{m-1}$	$9/4^4$	$g_3(m-3)+1$
$\vdots$	$\vdots$	$\vdots$
$u_1 0^{d-1} u_2 (0+1)^{m-1}$ $v_1 0^{d-1} v_2 (0+1)^{m-1}$	$3/4^d$	$g_3(m-d-1)+1$

$$g_d(m) = \frac{1}{4} g_d(m-1) + \sum_{i=2}^{d} \frac{9}{4^i} g_d(m-i) + \frac{3}{4^d} g_d(m-d-1) + \frac{3}{4}, \quad m > 0$$

Solve the Recurrence Equation by Generating Function, $d > 0$

Define $g_d(m) = 0$ for $m \le 0$.

Let $G(z) = \sum_m g_d(m) z^m$.

Rewrite the equation for $m \ge 0$.

$$g_d(m) = \frac{1}{4} g_d(m-1) + \sum_{i=2}^{d} \frac{9}{4^i} g_d(m-i) + \frac{3}{4^d} g_d(m-d-1) + \frac{3}{4} - \frac{3}{4}[m=0]$$

$$4^d g_d(m) = 4^{d-1} g_d(m-1) + \sum_{i=2}^{d} 4^{d-i} g_d(m-i) + 3 g_d(m-d-1) + 3 \cdot 4^{d-1} - 3 \cdot 4^{d-1}[m=0]$$

$$\sum_m 4^d g_d(m) z^m = \sum_m 4^{d-1} g_d(m-1) z^m + \sum_m \sum_{i=2}^{d} 9 \cdot 4^{d-i} g_d(m-i) z^m + \sum_m 3 g_d(m-d$$

$$\sum_m 3 \cdot 4^{d-1} z^m - \sum_m 3 \cdot 4^{d-1} [m=0] z^m$$

$$4^d G(z) = 4^{d-1} G(z) + \left(\sum_{i=2}^{d} 9 \cdot 4^{d-i} z^i G(z) \right) + 3 z^{d+1} G(z) + 3 \cdot 4^{d-1} \left(\frac{z}{1-z} \right)$$

$$G(z) = \frac{3 \cdot 4^{d-1} z}{(1-z) \left(4^d - 4^{d-1} z - \left(\sum_{i=2}^{d} 9 \cdot 4^{d-i} z^i \right) - 3 z^{d+1} \right)}$$

$$G(z) = \frac{3z(1-(z/4))}{4(1-z)^2 \left(1 - 2(z/4) - 3(z/4)^{d+1} \right)}$$

$$g_d(m) = \frac{3 \cdot 4^{d-1}}{2 \cdot 4^d - 1} m + o(1).$$

Conclusions

Exponentiation need not be computed from left to right, or from right to left, bit-by-bit.

Proper grouping of the nearby non-zero bits can improve the efficiency of computing $a^x b^y \bmod n$ and $a^x \bmod n$.

The sliding windows method need exponential number of pre-computations, while our methods need only linear number of pre-computations.

References

1. S. Arno and F. S. Wheeler. Signed digit representations of minimal hamming weight. *IEEE Transactions on Computers*, 42(8):1007–1010, 1993.

2. H. Cohen. Analysis of the sliding window powering algorithm. *Journal of Cryptology*, 18(1):63–76, 2005.

3. V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Complexity and fast algorithms for multi-exponentiation. *IEEE Transactions on Computers*, 49(2):141–147, 2000.

4. D. M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.

5. Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, second edition, 1994.

6. J. Jedwab and C. J. Mitchell. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, 25(17):1171–1172, 1989.

7. M. Joye and S. M. Yen. Optimal left-to-right binary signed-digit recoding. *IEEE Transactions on Computers*, 49(7):740–748, 2000.

8. X. Ruan and R. S. Katti. Left-to-right optimal signed-binary representation of a pair of integers. *IEEE Transactions on Computers*, 54(2):124–131, 2005.

9. J. A. Solinas. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, University of Waterloo, 2001. http://www.cacr.math.uwaterloo.ca.

10. W. C. Yang, **D. J. Guan**, and C. S. Lai. Fast multi-computation with asynchronous strategy. *IEEE Transactions on Computers*, 56(2):234–242, February 2007.

11. W. C. Yang, D. J. Guan, and C. S. Laih. Fast multi-computations with integer similarity strategy. In *International Workshop on Practice and Theory in Public key Cryptography-PKC 2005, LNCS 3386*, pages 139–153. Springer-Verlag, 2005.

12. S. M. Yen, C. S. Laih, and A. K. Lenstra. Multi-exponentiation. *IEE Proceedings, Computers and Digital Techniques*, 141(6):325–326, 1994.