ISSN:2320-0790

An International Journal of Advanced Computer Technology

Comparative Study of Efficient Modular Exponentiation Algorithms

Ibrahim Marouf, Mohammed Mosab Asad, Qasem Abu Al-Haija

King Faisal University, Department of Electrical Engineering, Al-Ahsa 31982, P.O. Box 380

Abstract: This paper presents description on the efficient modular multiplication techniques with numerical examples and flowchart diagrams. This review will help cryptoprocessor designers to in the effective selection of the underlying modular exponentiation unit to improve the hardware cost complexity such as area and speed. We found that conventional techniques of Modular Exponentiation (i.e. L-R binary, R-L binary, Montgomery ladder, and Sliding window) proved its efficiency for many years with average cost complexity of $O(Log(exponent))$ with possibility of saving to $k-1$ multiplications operations for all $k-bits$ of the exponent in the case of sliding window. However, the enhanced modular exponentiation based w-NAF and w-MOF are quite up-to-date and will replace all other algorithms as they have the minimum non-zero representation for the exponent. It was shown that NAF representation minimizes the number of nonzero digits in the binary representation, with an average of one third of nonzero digits while the average non-zero density of $n-bit$ MOF has been reduced to the half for $n \to \infty$.

Keywords: Modular Exponentiation, Left-to-Right, Right-to-Left, Montgomery Ladder, Sliding Window, Non-Adjacent Form, Mutual Opposite Form.

1. INTRODUCTION

The massive contemporary technological revolution in computing and wireless communication offered essential infrastructure to develop different applications and services of today's cloud/fog computing and Internet-Of-Thing (IoT) [1]. Such emerging technologies involve sharing and transferring of confidential and critical information which increased the demand on the security services for individuals and organizations. Humans have distinct perspective regarding security and in many applications security, shouldn't be compromised. In general words, security is a requirement like safety of the system.

Interestingly, cryptographic algorithms were used for many years to provide different security levels for the communication channels by implementing different cryptographic services such as authentication, data integrity, confidentiality. However, cryptographic techniques are significantly relying on the computer arithmetic and number theory algorithms. For instance, the well-known RSA crypto-algorithm [2] involves the use of different operations such as multiplication of two large numbers [3], the greatest common divisor (GCD) [4], the modular inverse and the modular exponentiation as basic operations for the encryption and decryption processes.

Modular exponentiation is an exponentiation performed over a modulus. It is useful in computer science, especially in the field of public-key cryptography as it occupies major degree of the algorithm computations. It plays an essential role in the cost and performance complexity of the implemented cryptosystem (i.e. public key algorithm).

Public key cryptographic algorithms encompass the use of large numbers (even more than 100 digit). Therefore, calculating the exponentiation then perform modular operation is not efficient and will case overflow, keeping in mind that is also need 'e' multiplication operation, where 'e' is the exponent. To illustrate the idea, let's take the following numerical example: lets calculate the: $\mathbf{x}^{2^{1024}}$.

$$\text{Using regular method: } x^{2^{1024}} = \begin{cases} x.x = x^2 \\ x^2.x = x^3 \\ x^3.x = x^4 \\ \quad . \\ \quad . \end{cases}$$

This method has linear complexity, it takes: 2^{1024}-1 Multiplication operation.

$$\text{using square method: } x^{2^{1024}} = \begin{cases} x.x = x^2 \\ x^2.x^2 = x^4 \\ x^4.x^4 = x^{16} \\ \quad . \\ \quad . \end{cases}$$

This method has logarithm complexity, it takes: 1024 multiplication operations only. But 1024 is power of two, how about power 26? This problem can be solved using multiplication. Nevertheless, more sophisticated and advanced methods were proposed to calculate the modular exponentiation to improve the cost of computation as we are going to discuss in the coming sections.

The remaining of this paper is organized as follows: Section 2 discusses the conventional techniques of modular exponentiation numerical examples and flowchart diagrams. Section 3 provides the review of two well-known enhanced and practical modular exponentiation algorithms with numerical examples. Finally, Section 4 concludes the paper.

2. MATHEMATICAL BACKGROUND

Modular arithmetic or Congruences [5] is a very important notions of number theory. It is the central mathematical concept in cryptography. Almost any cipher from the Caesar Cipher to the RSA Cipher use it. The main purpose of modular arithmetic is to return the result into a range called ring (Z_m) {0, 1, …, m-1}

The basic common modular arithmetic operations are: modular addition/subtraction, modular multiplication, and modular in invers. The basic method to retrieve the answer is to find the arithmetic operation and then reduce the answer by specific modulus. However, this is not efficient and take a lot of time and memory. Therefore, many algorithms in the literature were developed to improve the cost of modular operations. For instance, the congruence operation (reduction by modulus) can be defined as follows:

Definition: Let a, b, n be integers with $n \neq 0$;

$$a \equiv b \bmod n \quad \textit{where:}$$

a is congruent to b mod n and can be represented as

$$a = b + kn$$

Where k in an integer.

In modular arithmetic, the answer is closed to a finite group or set Z_n of numbers between 0 and $n-1$ and any number greater than n is congruent to one number only in the finite group. For example, the finite group of $n = 5$ as in Fig.1.

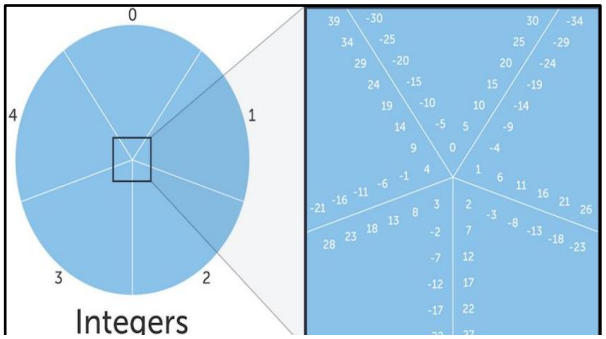

Fig. 1. Finite group for Zn = 5.

Examples: $27 \equiv 2 \bmod 5$, $-12 \equiv 3 \bmod 5$, $39 \equiv 4 \bmod 5$.

Properties: *Let* a, b, c, d, n be integers with $n \neq 0$.

1. $a \equiv 0 \bmod n$ if and only if $n|a$.
2. $a \equiv a \bmod n$
3. $a \equiv b \bmod n$ if and only if $b \equiv a \bmod n$.
4. If $a \equiv b \bmod n$ and $b \equiv c \bmod n$, then $a \equiv c \bmod n$.
5. If $a \equiv$ and $c \equiv d \bmod n$ then, $a + c \equiv b + d, a - c \equiv b - d, ac \equiv bd \bmod n$

Addition, subtraction, and multiplication can be used as usual with congruences. Simply, if the result from any of three operations is larger than $(n-1)$, the result is divided by n and the remainder is the wanted congruent. For instance, the integer modulo 6 has the following addition and multiplication results as shown in table 1.

TABLE 1. Addition and Multiplication modulo 6.

+	0	1	2	3	4	5	×	0	1	2	3	4	5
0	0	1	2	3	4	5		0	0	1	2	3	4	5
1	1	2	3	4	5	0		1	1	2	3	4	5	0
2	2	3	4	5	0	1		2	2	3	4	5	0	1
3	3	4	5	0	1	2		3	3	4	5	0	1	2
4	4	5	0	1	2	3		4	4	5	0	1	2	3
5	5	0	1	2	3	4		5	5	0	1	2	3	4

Division does not always exist in modulo n, the general rule is it can be divided by a when $\gcd(a,n) = 1$.

Definition: Let a, b, c, n with $n \neq 0$ and $\gcd(a,n) = 1$

$$\boldsymbol{If\ ab \equiv ac\ mod\ n,\ then\ b \equiv c\ mod\ n.}$$

In other words, if a and n are co-prime (relatively prime), it is valid to divide both sides of the congruence by a.

Example: *Solve* $3x + 6 \equiv 9\ mod\ 2$.
$3x \equiv 9 - 6\ mod\ 2 \equiv 3\ mod\ n \rightarrow x \equiv 1\ mod\ 2$,
since $gcd(3,1) = 1$, *division by 3 is allowed.*

Example: Solve $\boldsymbol{7x - 1 = -127\ mod\ 5.}$

$\gcd(7,5) = 1 \rightarrow 7x \equiv -127 + 1 \equiv -126 \bmod 5$
$\rightarrow x \equiv -18 \bmod 5$
$x \equiv 2 \bmod 5$, since -18 and 2 are congruent in modulo 5, as shown in Fig.1.

3. CONVENTIONAL TECHNIQUES OF MODULAR EXPONENTIATION

In this section, we discuss the basic techniques for exponentiation: Arbitrary choices of the base and exponent are allowed.

3.1 Left-To-Right Binary Method

This algorithm can also be called square-multiply algorithm [7] as it oscillates between both operation. When numbers are squared, the number of multiplication operations is reduced. Suppose we want to calculate $c \equiv b^e \bmod m$:

- Write 'e' in binary notation.
- Starting from the left (MSB-Most Significant Bit): for each '1' in 'e' square the result then multiply by 'b' and for each '0' square the result only.
- In each step calculate the modular of the result.

Fig.1 summarizes the Left-To-Right Binary algorithm.

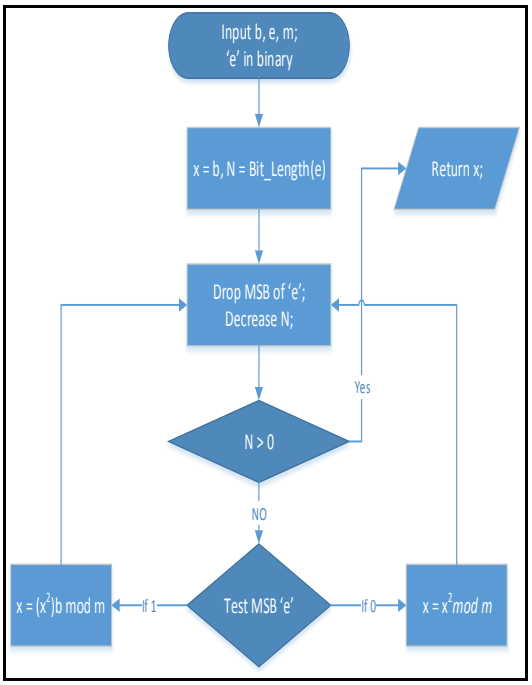

Fig.1. Left-to-right binary algorithm diagram.

Illustration Example: Calculate $4^{13} \bmod 49$.

- *Here we have:* $b = 4$, $e = 13 = 1101_2$ *and* $m = 49$.
- *Now MSB of e is ignored* ➔ *then* $e = 101_2$ *and as LSB of e is one* ➔ $b = (4^2)4\ mod\ 49 = 15$
- *MSB now is zero after dropping one, therefore:*
  ➔ $b = 15^2\ mod 49 = 29$
- *Then dropping another one bit. The last iteration now with '1':* $b = (29^2)4\ mod 49 = 32$
- *Finally, 'e' now is equal to zero, that give the answer* $32 \equiv 4^{13}\ mod\ 49$

3.2 Right-To-Left Binary Algorithm

Unlike left-to-right algorithm [7], this method reads the exponent from the least significant bit (LSB) to most significant bit (MSB). Suppose we want to calculate $c \equiv b^e \bmod m$. We can write the exponent (e) in binary form as:

$$e = \sum_{i=0}^{n-1} a_i 2^i$$

Where n is the bit length of the exponent e. Thus, we can re-write b^e as follows:

$$b^e = b^{(\Sigma_{i=0}^{n-1} a_i 2^i)} = \prod_{i=0}^{n-1} \left(b^{2^i}\right)^{a_i} \quad \ldots\ldots\ldots\ldots (1)$$

Equation (1) is then used as shown in fig.2. The ability to generate digit powers cheaply often enables right-to-left exponentiation to be performed

significantly faster and in less space than left-to-right methods using the same but restricted space [7].

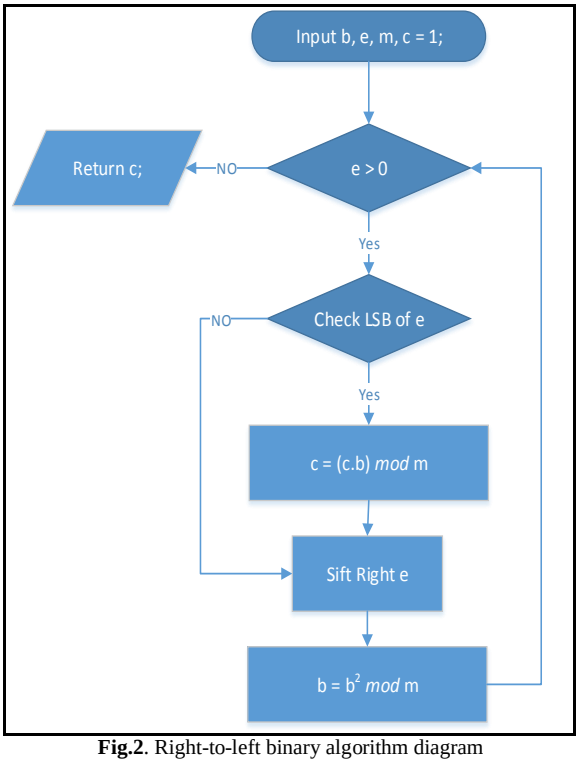

Fig.2. Right-to-left binary algorithm diagram

Illustration Example: Calculate 5^{10} mod 29:

- *Here we have the exponent e = 10= 1010_2.*
- *Starting from LSB which is zero, then, shift the exponent e = 101_2 and b = 5^2 mod 29 = 25.*
- *The second iteration LSB = 1, then c = (1x25) mod 29 = 25, e = 10_2, and b = 25^2 mod 29 = 16.*
- *3^{rd} iteration LSB = 0 ➔ b = 16^2 mod 29 = 24 & e = 1_2.*
- *Finally, e = 1_2 which give: c = (25x24) mod 29 = 20, thus 20 ≡ 5^{10} mod 29*

3.3 Montgomery's Ladder Technique

One aspect of binary exponentiation is side-channel attacks. If the attacker observes the sequence of multiplying and squaring, then he can get the exponent, or in public cryptography language, he gets the secret key. Montgomery ladder exponentiation [6] give higher security against side-channel attackers but it still not protected from timing attacks. Montgomery ladder is a modified version of left-to-right algorithm where it uses the binary representation of the exponent. The main difference from left-to-right algorithm is that Montgomery ladder scans the second bit of the exponent and calculates two possible partial results in one step, which are: raising to the square and raising to the square with multiplying by g. Depending on the value of the next exponent bit, we choose the proper partial result [6]. The complete steps of the algorithm are shown in fig.3.

In this algorithm, there are two multiplications in each loop which accumulate a total number of multiplication operations of $log2(e)$. However, these two multiplications can be performed in parallel to end up with almost equal delay in left-to-right algorithm.

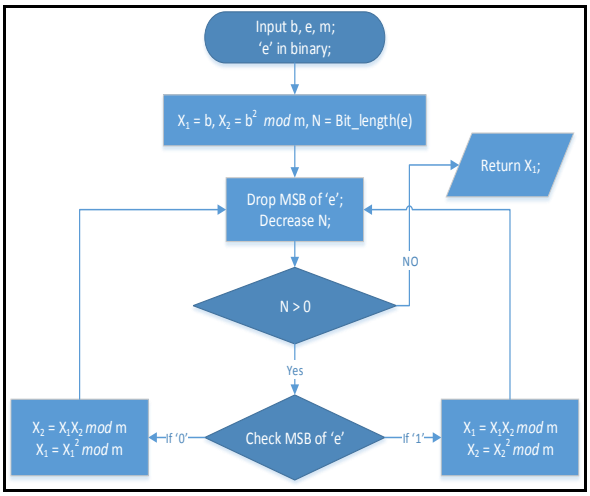

Fig.3. Montgomery's ladder algorithm diagram

Illustration Example: Calculate 4^6 mod 23.

- *Here the exponent e = 110_2 with length N = 3. The pre-calculated numbers: X_1 = 4, X_2 = 16.*
- *Dropping MSB to become: e = 10_2 and N = 2. Thus, X_1= 4x16 mod 23 = 18 and X_2 = 3.*
- *Dropping another bit and decrease N = 1. As a result, we get: X_1 = 18^2 mod 23 = 2. Therefore, 2 ≡ 4^6 mod 23.*

3.4 Sliding-Window Algorithm

Sliding window method [8] is a generalization of binary exponentiation. Instead of following one bit of exponent, we follow a group of bits to calculate the exponent. In the window, we determine the value of a certain partial power as we can use it repeatedly to calculate the correct power. Let assume the window size is k, the precomputation computes all odd powers of b from 1 to 2^{k-1}-1. As 'k' becomes larger, the number of precomputation powers will become large and we may not see any improvements over the binary exponentiation. However, if 'k' chosen wisely, the algorithm will run faster of binary exponentiation. The sliding-window algorithm is summarized in Fig.4.

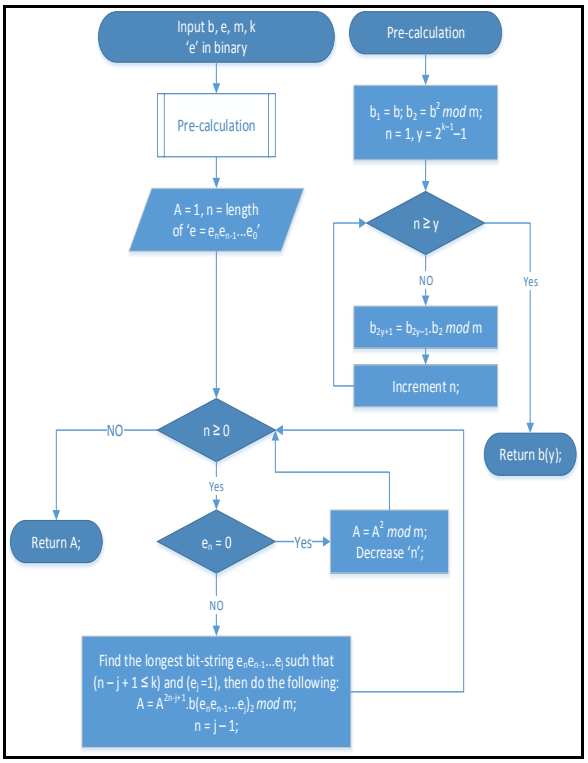

Fig.4. Sliding-window algorithm diagram

Illustration Example: Calculate 16^{2966} mod 63:

1. *The exponent is: 2966 = 101110010110_2, with length n = 11, we choose the windowing length k = 3, A= 1.*
2. *We compute pre-computation powers (all possible odd powers in k-length):*
   - *b_1 = 16, b_2 = 4*
   - *Run a loop from n = 1, to j = 2^{k-1}-1 = 3.*
     - *$b_3 = b_1 . b_2 \bmod 63 = 1$*
     - *$b_5 = b_3 . b_2 \bmod 63 = 4$*
     - *$b_7 = b_5 . b_2 \bmod 63 = 16$*
3. *Run the algorithm from n = 11 to n = 0:*
   - *n = 11 that mean: e_n= 1, j = 9 because its first bit '1' with $n - j + 1 \le k$.*

$$A = 1^{14} b(101) mod\ 63 = b_5 = 4\ ,\&\ n = 9 - 1 = 8$$

   - *n = 8 that mean: e_n= 1, j = 7 because its first bit '1' with $n - j + 1 \le k$.*

$$A = 4^{10} b(11) mod\ 63 = 4\ , and\ n = 7 - 1 = 6$$

   - *n = 6 that mean: e_n= 0,*

$$A = A^2 mod\ 63 = 16\ , and\ n = 6 - 1 = 5$$

   - *n = 5 that mean: e_n= 0,*

$$A = A^2 mod\ 63 = 4\ , and\ n = 5 - 1 = 4$$

   - *n = 4 that mean: e_n= 1, j = 2 because its first bit '1' with $n - j + 1 \le k$.*

$$A = 4^7 b(101) mod\ 63 = 16\ , and\ n = 2 - 1 = 1$$

   - *n = 1 that mean: e_n= 1, j = 0 because its first bit '1' with $n - j + 1 \le k$.*

$$A = 4^3 b(10) mod\ 63 = 4\ , and\ n = 0 - 1 = -1$$

Finally, the answer (as n < 0) is $4 \equiv 16^{2966} \bmod 63$.

3.5 Other Algorithms

The literature is rich with many solutions were proposed to enhance the conventional exponentiation methods. Even though, these solutions have improved the performance and cost complexity of conventional exponentiation methods, the enhanced modular exponentiation based w-NAF and w-MOF [9] are quite up-to-date and will replace all other algorithms as they have the minimum non-zero representation for the exponent (this will be discussed in the next section). However, the other algorithms can be summarized in following categories:

1. Fixed-exponent exponentiation algorithms [10]: The exponent is fixed and arbitrary choices of the base are allowed. RSA encryption and decryption schemes benefit from these algorithms. Examples of such algorithms: the addition chains for additive groups (useful for elliptic curves) and the vector-addition chains
2. Fixed-base exponentiation algorithms [11]: The base is fixed and arbitrary choices of the exponent are allowed. El-Gamal encryption and signatures schemes and Diffie-Hellman key agreement protocols benefit from these algorithms. Examples of such algorithms: Fixed-base windowing method, Fixed-base Euclidean method, Fixed-base comb method.
3. Other algorithms are discussed in [12] such as: Montgomery reduction method, Addition chains method, Signed-digit method, folding-in-half method, SDF (Signed-Digit-Folding) Algorithms, common-multiplicand-multiplicand (CMM), signed-digit recoding and parallel method. SDF-CMM Montgomery binary exponentiation algorithm, on average the total number of single precision multiplications can be reduced by about 61.3% and 74.1% as compared with Chang-Kuo-Lin's CMM modular exponentiation algorithm and Ha-Moon's CMM Montgomery modular exponentiation algorithm, respectively.

4. ENHANCED MODULAR EXPONENTIATION BASED W-NAF AND W-MOF

Exponent recoding Algorithms used to reduce the number of multiplications in the basic repeated square-and-multiply algorithms by replacing the binary representation of the exponent with a representation which has fewer non-zero terms. Since the binary representation is unique, finding a representation with fewer non-zero components necessitates the use of digits besides 0 and 1. Transforming the exponent from one representation to another is called exponent recoding.

The most common method for computing exponentiation of random elements are sliding window schemes, which enhance the efficiency of the binary method at the expense of some precomputation. Signed representations of the exponent are meaningful because they decrease the amount of required precomputation. The asymptotic best signed method is w-NAF (Non-Adjacent Form), where it minimizes the precomputation efforts whilst the non-zero density is nearly optimal. Unfortunately, w-NAF can be computed only from the least significant bit, i.e. right-to-left. However, relating to memory constraint devices, left-to-right recoding schemes are by far more valuable. Therefore, MOF (Mutual Opposite Form), another representation of signed binary strings, which can be computed in any order [13].

4.1 Non-Adjacent Form (NAF)

NAF [14] assures a unique representation of an integer, but its main benefit is that the Hamming weight of the value will be minimal. For regular binary representations of values, half of all bits will be non-zero, on average, but with NAF this drops to only one-third of all digits.

Because every non-zero digit should be adjacent to two 0s, the NAF representation can be implemented such that it only takes a maximum of n + 1 bits for a value that would normally be represented in binary with n bits.

The properties of NAF make it useful in various algorithms, especially in cryptography; e.g., for reducing the number of multiplications needed for performing an exponentiation. In the algorithm, exponentiation by squaring, the number of multiplications depends on the number of non-zero bits. If the exponent here is given in NAF form, a digit value 1 implies a multiplication by the base, and a digit value −1 by its reciprocal.

The generalization of NAF recoding for w > 2 will is called window non-adjacent form (w-NAF) (here, the non-adjacent property states that among any w adjacent bits, at most one is non-zero). According to [14], this strategy is the optimal one for w > 3. w-NAF is computed directly from binary strings using a generalization of NAF recoding. First, we define w-NAF as stated in [15].

Definition (w-NAF): A sequence of signed digits is called w-NAF iff the following three properties hold:

(A) The most significant non-zero bit is positive.
(B) Among any w consecutive digits, at most one is non-zero.
(C) Each non-zero digit is odd and less than 2^{w-1} in absolute value.

Note that 2NAF and NAF are the same. Also, n bit integer produce n+1 NAF representation. Algorithm 1 describes the generation of w-NAF:

ALGORITHM 1: GENERATION OF W-NAF [15]:

Input : width w , an n-bit integer d
Output : w-NAF $(S_n S_{n-1} \dots S_0)$ *of* d
$i \leftarrow 0$
while $d \geq 1$ **do**
  if d *is even* **then**
    $S_i \leftarrow 0$
  else
    $S_i \leftarrow d \bmod^* 2^w ; d \leftarrow d - S_i$
  $d \leftarrow \frac{d}{2}; i \leftarrow i + 1$
return $(S_n S_{n-1} \dots S_0)$

* "mods" means the signed modulo, namely u ≡ a mods b is defined as: $u \equiv a \bmod^s 2^w$

$$\equiv \begin{cases} a \bmod 2^w ; & -(2^{w-1}-1) \leq u < (2^{w-1}-1) \\ a \bmod 2^w - 2^w ; & (2^{w-1}-1) \leq u < -(2^{w-1}-1) \end{cases}$$

$u \equiv a \bmod b$ and $-2^w \leq u < 2^w$ or can be written as: $d \bmod^s 2^w = d \bmod 2^w - 2^w$ if the output is out of range from allowing bits (τ).

The algorithm generates w-NAF from the least significant bit, that is right-to-left generation again. The average density of nonzero bits is asymptotically 1/ (w + 1) for n → ∞, and the digit set equals τ = {±1, ±3, . . ., ± (2^{w-1} − 1)}.

For w= 2 we have 2-NAF or simply NAF which also may be called radix 4 booth recording. The algorithm may be optimized for w= 2 as follows:

ALGORITHM 2: GENERATING NAF:

Input $E = (e_{n-1}e_{n-2}\cdots e_1e_0)_2$
Output $Z = (z_n z_{n-1}\cdots z_1 z_0)_{NAF}$
$i \leftarrow 0$
While $E > 0$ **do**
if E is odd **then**
$z_i \leftarrow 2 - (E \bmod 4)$
$E \leftarrow E - z_i$
else
$z_i \leftarrow 0$
$E \leftarrow E / 2$
$i \leftarrow i + 1$
return Z;

For example: *Recording of* $01011101_{(2)}$ *is* $010\bar{1}00\bar{1}01_{(NAF)}$.

Particularly, NAF representation minimizes the number of nonzero digits in the representation, with an average of one third of nonzero digits [13]. It is straightforward for R-T-L exponentiation to work with signed-digit exponent:

$$e = \sum_{d_i=1} 2^i - \sum_{d_i=\bar{1}} 2^i \Rightarrow x^e = \left(\prod_{d_i=1} x^{(2^i)}\right)\cdot\left(\prod_{d_i=\bar{1}} x^{(2^i)}\right)^{-1}$$

Algorithm 3 shows the R-T-L algorithm with exponent in NAF representation.

ALGORITHM 3: R-T-L EXPONENTIATION USING NAF REPRESENTATION:

Input $x, e = (e_{n-1}e_{n-2}\cdots e_1e_0)_{NAF}$
Output x^e
$S \leftarrow x; R_1 \leftarrow 1; R_{\bar{1}} \leftarrow 1;$
for each *digit* d_i (i *from* 0 *up to* n-1) **do**
if $d_i \neq 0$ **then**
$R_{d_i} \leftarrow R_{d_i} \times S;$
else
$S \leftarrow S^2$
end
return $R_1 \times (R_{\bar{1}})^{-1};$

Illustration Example: Calculate Z = 6^{29} mod 13.

- *We start with S = 6,* $R_1 = R_{\bar{1}} = 1$ *and NAF representation of exponent is:* $100\bar{1}01_{NAF}$.
- *Then, we run the algorithm for i=0 to i= 5,*
- *Finally:* $Z = 5 \times 9^{-1} mod\ 13 = 5 \times 3\ mod\ 13 = 2$

4.2 Mutual Opposite Form (MOF)

New canonical representation for Signed Binary Strings [13]. As to achieve a unique representation, we introduce the following special class of signed binary strings, called the mutual opposite form (MOF).

Definition of (MOF): The n-bit mutual opposite form (MOF) is an n-bit signed binary string that satisfies the following properties:

a) The signs of adjacent non-zero bits (without considering zero bits) are opposite.
b) The most non-zero bit and the least non-zero bit are 1 and -1, respectively, unless all bits are zero.

Some zero bits are inserted between non-zero bits that have a mutual opposite sign. An example of MOF representation is $0100\bar{1}0100\bar{1}00\bar{1}\bar{1}0$. An important observation is that each positive integer can be uniquely represented by MOF. The average non-zero density of n-bit MOF is 1/2 for $n \to \infty$. Algorithm 4 provides an explicit conversion from Binary to MOF.

ALGORITHM 4: CONVERSION FROM BINARY TO MOF

Input: *a non−zero n−bit binary string* $d = d_{n-1}d_{n-2}\ldots d_1d_0$
Output: *MOF* $\mu_n\ldots\mu_1\mu_0$ *of* d
$\mu_n \leftarrow d_{n-1}$
for $i = n - 1$ *down to* 1 **do**
$\mu_i \leftarrow d_{i-1} - d_i$
$\mu_0 \leftarrow -d_0$
return $(\mu_n, \mu_{n-1}, \ldots, \mu_1, \mu_0)$.

Interestingly, the MOF representation of an integer d equals the recoding performed by the classical Booth algorithm for binary multiplication. Algorithm 4 converts a binary string from most significant bit to MOF form, it is also possible to apply this method right-to-left which provides flexibility.

4.3 Right-to-Left: w-NAF using MOF

In order to describe the proposed scheme, we need the conversion table for width w. Starting form LSB must not be zero, then writing all possibilities of allowed numbers and satisfied MOF. For example: In the case of w= 3, we use the following table for the right-to-left sliding window method:

$$Table_{\overline{3SW}}:\ 001 \leftarrow \begin{cases} 001 \\ 01\bar{1} \end{cases} \quad 00\bar{1} \leftarrow \begin{cases} 00\bar{1} \\ 011 \end{cases} \quad 003 \leftarrow \begin{cases} 10\bar{1} \\ 11\bar{1} \end{cases} \quad 00\bar{3} \leftarrow \begin{cases} \bar{1}01 \\ \bar{1}\bar{1}\bar{1} \end{cases}$$

In analogous way, $Table_{wSW}$ is defined for general w. Based on this table, Algorithm 5 provides a simple carry-free w-NAF generation.

ALGORITHM 5: RIGHT-TO-LEFT GENERATION FROM BINARY TO W-NAF

Input : width w , *a non – zero* n *– bit binary string* $d = d_{n-1}d_{n-2} \ldots d_1 d_0$
Output : *wMOF* $v_n..v_1v_0$ *of* d
$d_{n+w-2} \leftarrow 0; d_{n+w-3} \leftarrow 0; ...; d_n \leftarrow 0; d_{-1} \leftarrow 0; i \leftarrow 0$
while $i \leq n$ do
if $d_{i-1} = d_i$ **then**
$v_i \leftarrow 0 ; i = i + 1$
else {MOF window begins with a non-zero digit right-hand
$(v_{i+w-1},..., v_i) \leftarrow \text{Table}_{wSW}\ (d_{i+w-2} - d_{i+w-1} - d_{i+w-1},...,d_{i-1} - d_i)$
$i \leftarrow i + w$
return $(v_n, v_{n-1}, ..., v_1, v_0)$.

Illustration Example: Convert the following sequence to 4-NAF using two methods, d = 110 101 010 111_2 = 3415_{10}

Method I: Generating 4-NAF from binary

i	**12**	**11**	**10**	**9**	**8**	**7**
$\mathbf{S_i}$	1	0	0	0	$\bar{3}$	0
$\mathbf{d_{i+1}}$	0	1	2	4	8	13
i	**6**	**5**	**4**	**3**	**2**	**1**
$\mathbf{S_i}$	0	0	5	0	0	0
$\mathbf{d_{i+1}}$	26	52	104	213	426	852

Method II: Generating 4-NAF from MOF

MOF	1	0	$\bar{1}$	1	$\bar{1}$	1	$\bar{1}$	1	$\bar{1}$	1	0	0	$\bar{1}$
$\mathbf{S_i}$	1	0	0	0	$\bar{3}$	0	0	0	5	0	0	0	7

5. CONCLUSIONS

Modular exponentiation operation is a core operation that significantly affect the performance of public key Cryptoprocessor such as RSA, El-Gamal and Shmit-Samoa. We reported on different efficient modular exponentiation techniques such as Binary L-to-R, Binary R-to-L, Montgomery Ladder, Sliding Window, Non-Adjacent Form (NAF), Mutual Opposite Form (MOF). We found that implementing binary Right-to-Left with w-NAF using MOF will record the highest throughput as it minimizes the number of non-zero digits in the binary representation to the half or less (on average) while recording multiple bits (window size) in each step.

6. REFERENCES

[1]. G. Santucci. The Internet of Things: Between the Revolution of the Internet and the Metamorphosis of Objects. *European Commission Community Research & Development Information Servi*ce, 2016. https://pdfs.semanticscholar.org/adb7/03eb4c53ccba53a8973fbff2f30563363a58.pdf

[2]. Q. Abu Al-Haija, M. Smadi, M. Jaffri and A. Shua'ibi. Efficient FPGA Implementation of RSA Coprocessor Using Scalable Modules. *9th International Conference on Future Networks and Communications (FNC-2014)*. by Elsevier. Ontario. Canada. 17-20, Aug-2014. https://doi.org/10.1016/j.procs.2014.07.092

[3]. Q. Abu Al-Haija, M. Al-Ja'fari and M. Smadi. A comparative study up to 1024-bit Euclid's GCD algorithm FPGA implementation and synthesizing. *2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, Ras Al Khaimah, United Arab Emirates, pp. 1-4. https://DOI.10.1109/ICEDSA.2016.7818535

[4]. M. M. Asad, I. Marouf and Q. Abu Al-Haija. Review of Fast Multiplication Algorithms for Embedded Systems Design. *International Journal of Scientific & Technology Research*, Volume 6, Issue 08, 2017. http://www.ijstr.org/research-paper-publishing.php?month=aug2017

[5]. W. Trappe and L. C. Washington. Introduction to Cryptography with Coding Theory. *By Prentice Hall*, 2002, 1: 1-176. http://calclab.math.tamu.edu/~rahe/2014a_673_700720/Trappe_2006.pdf

[6]. A. Jakubski and R. Perliński. Review of General Exponentiation Algorithms. *Scientific Research of the Institute of Mathematics and Computer Science*, 2(10), 87-98, (2011)

[7]. C. D. Walter. Right-to-Left or Left-to-Right Exponentiation?. *1st International Workshop on Constructive Side-Channel Analysis and Secure Design*, Darmstadt, Germany, February 4-5, (2010).

[8]. N. Nedjah, L. M. Mourelle and R. M. Silva. Efficient Hardware for Modular Exponentiation Using the Sliding-Window Method. *4th International Conference on Information Technology*, 2007. ITNG '07.

[9]. E. Dahmen, K. Okeya, T. Takagi. Efficient Left-to-Right Multi-Exponentiations. *Technical Report TI-2/05*, 2005.

[10]. A. Weimerskirch. Fixed-Exponent Exponentiation. *Encyclopedia of Cryptography and Security*, Springer, pp 485-486, 2011.

[11]. A. Weimerskirch. Fixed-Base Exponentiation. *Encyclopedia of Cryptography and Security*, Springer, pp 482-485, 2011.

[12]. C.L. Wu, D.C. Lou and T.J. Chang. Fast modular multiplication based on complement representation and canonical recoding. *International Journal of Computer Mathematics* 87:13, Pp. 2871-2879, (2010)

[13]. K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi. Signed Binary Representations Revisited. *Annual International Cryptology*

Conference CRYPTO: Advances in Cryptology – CRYPTO 2004 pp 123-139, (2004).

[14]. Blake, I., Seroussi, G., and Smart, N. Elliptic Curves in Cryptography. *Cambridge University Press New York*, NY, USA, 1999, ISBN:0-521-65374-6.

[15]. J.A. Solinas. Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, Springer, Volume 19, Issue 2–3, pp 195–249, (2000).

[16]. J. Jedwab and C.J. Mitchell. Minimum Weight Modified Signed-digit Representations and Fast Exponentiation. *Electronics Letters*, *IET*, 25, 1989.