Journal of Computer Science

Original Research Paper

Optimizing {0, 1, 3}-NAF Recoding Algorithm Using Block-Method Technique in Elliptic Curve Cryptosystem

Mohsen Bafandehkar, Sharifah Md Yasin and Ramlan Mahmod

Department of Computer Science, University Putra Malaysia, Selangor, Malaysia

Article history
Received: 25-07-2015
Revised: 13-12-2016
Accepted: 14-12-2016

Corresponding Author:
Mohsen Bafandehkar
Department of Computer Science, University Putra Malaysia, Selangor, Malaysia
Email: Bafandehkar@gmail.com

Abstract: The most expensive and time-consuming operation in Elliptic curve cryptosystem is scalar multiplication operation. Optimization of scalar multiplication will substantially enhance the ECC performance. Scalar multiplication can be improved by using an enhanced scalar recoding algorithm that can decrease the number of operations in the scalar representation process. The objective of this research is to introduce an efficient design and implementation of {0,1,3}-NAF scalar recoding algorithm by applying block method technique. The base algorithm has a complex look up table. With block method application on base algorithm, a complex look up table is undesired. Instead a fix look up table is introduced with less computation required for recoding. The Big-*O* notation is used to measure the complexity and (μs) used to evaluate the running time of base and proposed algorithm.

Keywords: Elliptic Curve Cryptosystem, Scalar Multiplication, Elliptic Curve, Scalar Recoding Algorithm, Non-Adjacent Form

Introduction

The efficiency of scalar multiplication operation has direct effects on the performance of ECC (Kodali *et al*., 2013). Scalar multiplication involves with three levels of computations: Scalar arithmetic, point arithmetic and field arithmetic which are demonstrated in Fig. 1.

Level 1: Scalar Arithmetic

This computational level involves with scalar representation and scalar recoding technique. In order to recode the scalar into selected number representation the scalar recoding technique is required. This technique should reduce the hamming weight of scalar *k*.

According to literature, there are different bases to represent scalar *k*. Fundamentally, base 2 is known as the natural representation. Scalar *k* in binary and NAF is represented in this base. Joye and Yen (2002) used different base to represent scalar *k*. Reducing the Hamming weight will enhance the scalar multiplication performance, since less addition and doubling is required (Shah *et al*., 2010). Scalar recoding technique is used to recode a scalar *k* into different representation with less Hamming weight (Yasin *et al*., 2014). The result of this recoding can have the same magnitude to the scalar or lesser.

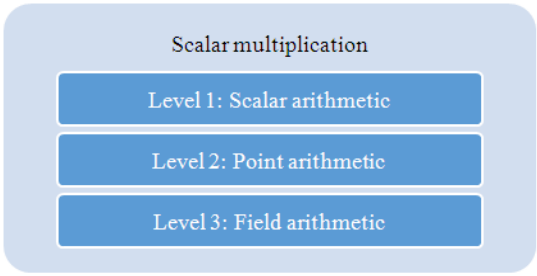


Fig. 1. Level of scalar multiplication computation

Level 2: Point Arithmetic

This level involves with the point operations. The different arithmetic point operations over ECC are point addition, subtraction and doubling. Efficiency of point operations are dependent on the number of field operations involved (Longa and Gebotys, 2010). From elliptic curve, points can be obtained. The Koblitz curve is a special family curve in which point multiplication is considerably faster than generic curve (Sakthivel and Nedunchezhian, 2012).

Level 3: Field Arithmetic

Binary and prime field has different ways and cost of process. Point operations are executed by utilizing

the finite field operation. The impact of efficiency of this level is essential (Hitchcock *et al*., 2003; Morales-Sandoval and Feregrino-Uribe, 2006). Inversion is the most expensive, followed by the multiplication cost and then the squaring cost (Yasin *et al*., 2015).

Among all these operations scalar arithmetic level is the most expensive operation (Aranha *et al*., 2012) and according to literature (Hankerson *et al*., 2003) enhancement in this operation will significantly increase the efficiency of ECC.

Accordingly, improving the first two levels will lead to significant increment in efficiency of scalar multiplication. Scalar recoding algorithm can be improved by employing an enhanced scalar recoding algorithm that can decrease the number of operations and required less running time in the scalar representation process (Bafandehkar *et al*., 2015).

Thus, in this study we focus on optimizing the scalar arithmetic algorithm and this objective c`n be achieved by proposing a new algorithm with less algorithm complexity.

Related Works

{0, 1, 3}-NAF Algorithm

Md Yasin (2011) introduced a scalar representation algorithm to convert binary expansion into {0, 1, 3}-NAF. This recoding algorithm is specifically for binary numbers that have adjacent nonzero digits in its representation. This scalar representation is in base 2 using digit 0, 1 and 3. The special NAF property is adopted in this scalar representation. This recoding method is in left-to-right mode based on the technique proposed by Joye and Yen (2000). A look-up table has been proposed to simplify the recoding technique. The non-adjacency property for each row of the look-up table is proven by using the same technique used by Joye and Yen (2000).

This method is real-time operation. It has homogeneous approach to real-time recoding so the recoded scalar can be used straight away for scalar multiplication algorithm. This is possible because scanning digits of the scalar for recoding and scalar multiplication are done using the same mode, which is from left-to-right. In the literature, this type of recoding promotes better memory usage and mostly preferred for memory constraint devices (Khabbazian *et al*., 2005). Whereas, in heterogeneous approach, the recoded digits are saved before it is used in the scalarmultiplication algorithm. This is because scanning digits of the scalar for recoding and scalar multiplication is initiated from different directions, that is, the recoding mode is right-to-left and the scalar multiplication mode is left-to-right. Generally, this type of recoding needs an additional n-bit RAM for storage, where n is the bit size of the scalar. This is one of the advantage of {0, 1, 3}-NAF algorithm over other recoding algorithms which scan and recode from right to left. Figure 2 shows a flowchart illustrates the steps of {0, 1, 3}-NAF algorithm.

Table 1. Look-up table for {0, 1, 3}-NAF recoding Look-up table for {0, 1, 3}-NAF recoding

	Input						
No.	b_{i+1}	r_{i+1}	r_i	r_{i-1}	b_i	Special case	Output r'_i
1	0	0	0	X	0		0
2	0	0	1	0	0		1
3	0	0	1	1	1	*	0
4	0	0	1	1	1		1
5	0	1	0	X	0	r'_{i+1} = 1 OR 3	0
6	0	1	1	1	1		0
7	1	0	1	0	1		1
8	1	0	1	1	1		0
9	1	1	0	0	0		0
10	1	1	0	1	1	r'_{i+1} = 1 OR 3	0
11	1	1	0	1	1	r'_{i+1} = 0	3
12	1	1	1	0	1	r'_{i+1} = 0	3
13	1	1	1	0	1	r'_{i+1} = 0	0
14	1	1	1	1	1	r'_{i+1} = 0	3
15	1	1	1	1	1	r'_{i+1} = 1 OR 3	0

Note: X = 0 or 1 and $b_i = \left\lfloor \frac{(b_{i+1} + r_i + r_{i-1})}{2} \right\rfloor$ where ⌊⌋ denotes a floor function that will give the largest integer less than or equal to $((bi_{+1}+r_i+r_{i-1})/2)$

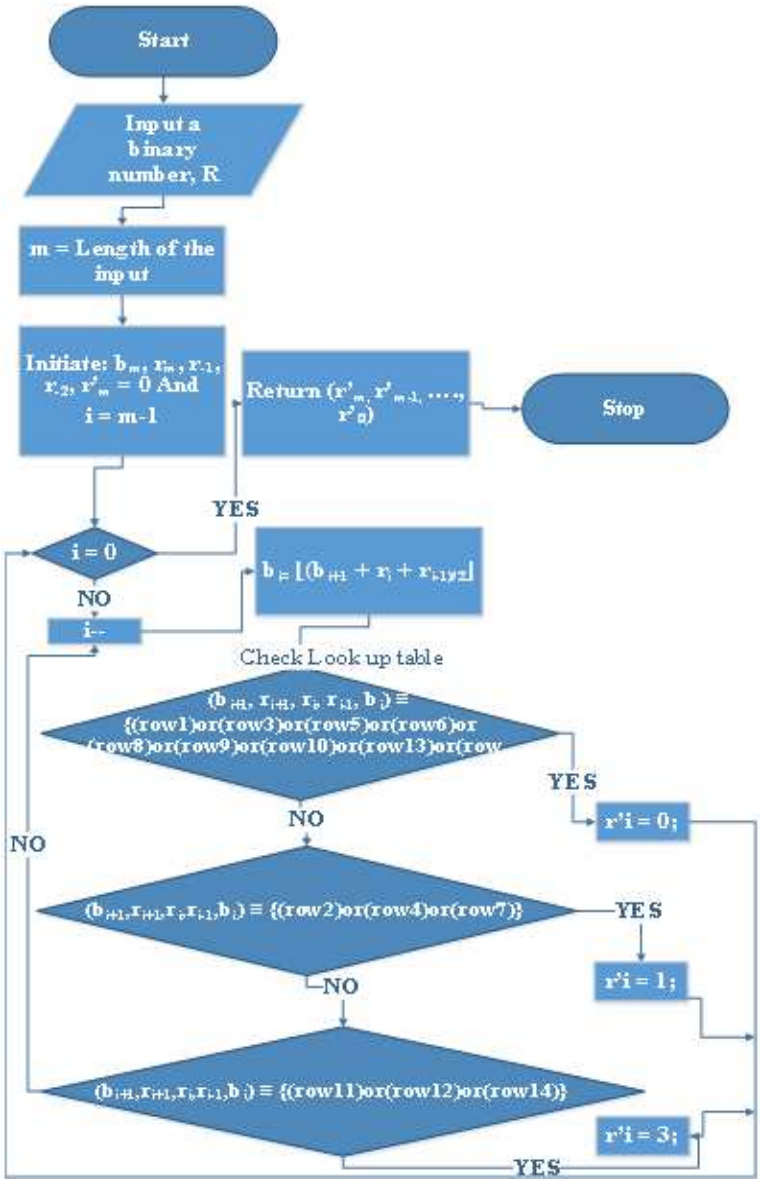


Fig. 2. Flowchart of {0, 1, 3}-NAF algorithm

For example for the given X= $(1000111011010)_2$ to compute the equivalent value in {0, 1, 3}-NAF representation using this algorithm and Table 1. As the result of conversion Y = 1000103003010. If the hamming weight is h and the length of input binary isl. The worst case of a binary expansion is when h = 1 and the average case is when $h = \frac{l}{2}$. At average case, this algorithm has the hamming weight less than $\frac{l}{3}$, whereas the traditional NAF hamming weight is $\frac{l}{3}$ hus, {0, 1, 3}-NAF has better hamming weight that the traditional NAF at average case (Md Yasin, 2011).

Traditional-NAF Blocking Method

Reitwiesner (1960) shows NAF representation with radix $r = 2$, where each digit in the NAF, $a_i \in \{-1,0,1\}$ must satisfy, a_i, $a_{i+1} = 0$, for all $i \geq 0$. The NAF can also be written as $((an_{-1}...a_0)_{NAF}$. The NAF is unique with an average Hamming weight of $\frac{l}{3}$ where l is the bit length of the NAF representation. In the literature, the NAF is commonly used and efficient for elliptic curve scalar multiplication. Traditional NAF is also utilizing left-to-right scanning method which reduce the complexity load of algorithm.

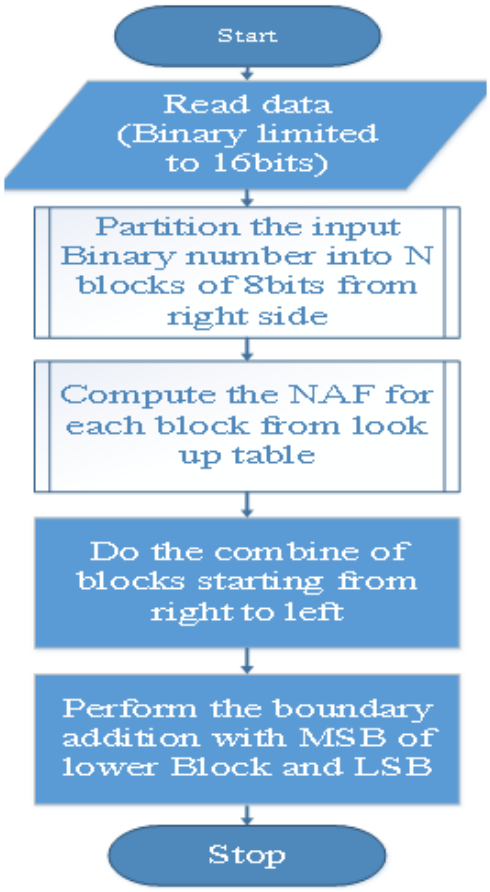


Fig. 3. Flowchart of block method algorithm

In the other hand Pathak and Shanghi (2010) introduced a blocking technique to improve the NAF conversion utilizing a fix look up table. The look up table contains equivalent value for each binary numbers in NAF representation form. Since the given binary number will be partitioned in blocks of 8 bits length so the combination of 2^8 which is equal to 256 numbers in the table is required. Therefore the table will contains the equivalent value for numbers starting from 0 to 255. After partitioning the given binary number and replacing the equivalent value of each block form table, the blocks must be combined together to compute the final result. They conclude utilizing this technique will reduce the number of iteration in traditional NAF. The Fig. 3 shows the flowchart which illustrates the steps taken for block method algorithm.

In this method, in order to compute the equivalent value in NAF for the given X = (1000111011010)2 using Block method, the following steps will be taken:

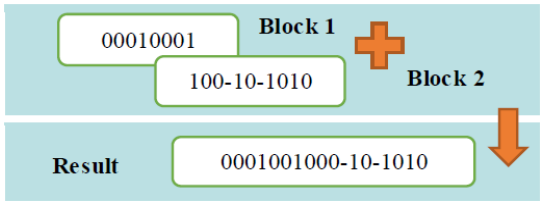
- Partition the input Binary number into N blocks of 8 bits from right side, pad '0' digit to complete the 8 bits block

Block 2	*Block 1*
00010001	11011010

- Represent each binary block's in its equivalent {-1, 0, 1}-NAF representation from Table 2

Block 2	*Block 1*
00010001	100-10-1010

- Combining the blocks the boundary addition with Most Significant Bit (MSB) of lower block and Least Significant Bit (LSB) of upper block must perform to get the final NAF



The Proposed Method

The proposed method begins with creating a look up table contains 256 numbers starting from 0, with equivalent value in {0, 1, 3}-NAF representation. The other properties of this table are similar with the Table 2.

The focus of this research is to improve the performance of {0, 1, 3}-NAF recoding algorithm (Md Yasin, 2011) by applying blocking technique introduced by Pathak and Shanghi (2010). In this method a new look up table which contains equivalent value for each binary numbers in {0, 1, 3}-NAF representation form has been set up.

According to Fig. 4 the given binary number will be partitioned in blocks of 8 bits length. Therefore we need the combination of 2^8 numbers which is equal to 256 numbers in table. The proposed look up table will contains the equivalent value for decimal numbers starting from 0 to 255.

The Algorithm.1 works as follows; Line 1 initiates i and j variables and set them to 0. The variable m is declares length of input. Line 2 is state an iteration to loop through bits in input r. In line 3 another iteration is defined to loop through 8 bits from left to right and line 4 is checking if the index j is bigger or equal to the length of input, (the index has went through all the bits so) it will terminate the iteration process. Line 7 has define a two dimensional container to hold n partition of input in size of 8 bits. In line 9 a key look up has been defined as a list $[r_k]$ to map the equivalent value of the partition from look up table and store it in

r'_k. Line 10 will return the recoded r into {0,1,3}-NAF representation.

Algorithm. 1 Proposed {0, 1, 3}-NAF Block Method
Input: $r = (r_{m-1},\ldots,r_0)_2$
Output: $r' = (r'_n,\ldots,r'_0)_{\{0,1,3\}\text{-}NAF}$
1. $i \leftarrow 0; j \leftarrow 0; m \leftarrow$ *length of input*
2. *while* $j < m$ -1
3. *for i from* 0 *to* 7
4. *if* $j >= m$-1
5. *break*
6. *else*
7. $r_{k,i} \leftarrow r_i$
8. j ++
9. $r'_k \leftarrow$ *list* $[r_k]$
10. *return* $r_{\{0,1,3\}\text{-}NAF}$

To visualize the processes in proposed algorithm, Fig. 5 presents the flowchart of the steps taken for this research.

Table 2. Look up table of block method for traditional NAF Algorithm (Pathak and Shanghi, 2010)

DEC	Binary-8bits	{-1, 0, 1} NAF
0	0	0
1	1	1
2	10	10
3	11	00000010-1
4	100	100
5	101	101
6	110	0000010-10
7	111	00000100-1
.	.	.
.	.	.
.	.	.
250	11111010	10000-1010
251	11111011	100000-10-1
252	11111100	100000-100
253	11111101	100000-101
254	11111110	1000000-10
255	11111111	10000000-1

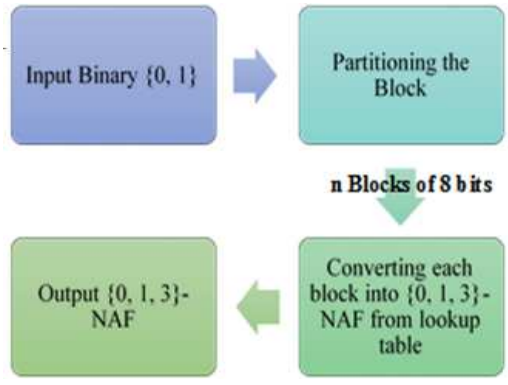


Fig. 4. Proposed {0, 1, 3}-NAF block method algorithm structure

Figure 5 shows a flowchart to convert binary input data in {0, 1, 3}-NAF representation using the proposed algorithm. For example, for the given $X = 4570_{10} = (1000111011010)_2$ using proposed method, the following steps will be taken:

Step 1: Partition the input binary number into N blocks of 8 bits from right side

Block 1	*Block 2*
00010001	11011010

Step 2: Represent each binary block's in its equivalent {0, 1, 3} -NAF using look up Table 3

Block 1	*Block 2*
00010001	03003010

Step 3: Place each converted block consecutively and respectively

Result
0001000103003010

The value of final answer in decimal can be computed as follows:

$$0^{15}0^{14}0^{13}1^{12}0^{11}0^{10}0^{9}1^{8}0^{7}3^{6}0^{5}0^{4}3^{3}0^{2}1^{1}0^{0}$$
$$= 2^{12} + 2^{8} + 3\times 2^{6} + 3\times 2^{3} + 2^{1}$$
$$= 6096 + 256 + 192 + 24 + 2 = 4570$$

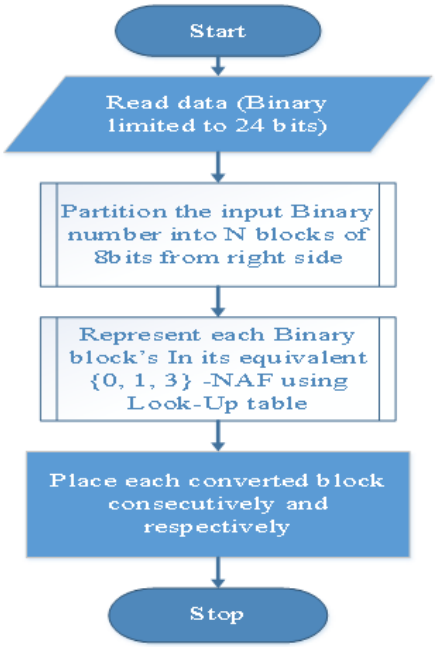


Fig. 5. Flowchart of proposed {0,1,3}-NAF block method algorithm

Table 3. Proposed look up table for {0, 1, 3}-NAF block method algorithm

DEC	Binary-8bits	{0, 1, 3} NAF
0	0	0
1	1	1
2	10	10
3	11	3
4	100	100
5	101	101
6	110	30
7	111	103
.	.	.
.	.	.
.	.	.
250	11111010	10303010
251	11111011	10303003
252	11111100	3030300
253	11111101	3030301
254	11111110	10303030
255	11111111	3030303

Performance Analysis

The Algorithm analysing is defined as predicting the required resources that the algorithm needs to perform computations. Although computer resources are categorized as memory, communication bandwidth, or computer hardware, the crucial concern is the computational time that must be measured. Commonly, identification of the most efficient algorithm for a problem can be done by analysing several candidate algorithms.

An algorithm running time for a certain input is defined as the number of operations or steps to perform a process. It is more appropriate to set the notion of each step so that the analysing method is more machine independent. To execute each line of a pseudo code, a constant amount of time is required. One line might need a different amount of time than another line, but the assumption is that each execution of the *i-th* line takes time c_i, where c_i, is a constant (Cormen *et al*., 2001).

The experimental results were used to evaluate and validate the performance of the base and proposed algorithm. Two metrics are explained in following sections. These metrics are complexity and running time.

Complexity Analysis

A characteristic of an algorithm described as run time performance and memory usage and expressed in Big-*O* notation. The efficiency of algorithm has been measured in terms of asymptotic complexity since 1973 (Gilberg and Forouzan, 2004).

The outcome of complexity analysis commonly is the worst-case complexity of an algorithm. But this does not always give reasonable correspondence with running time. For example, a component of an algorithm may be executed many times, each time with a different cost. Thus, in order to compute the efficiency of algorithm in real environment, the average performance of running time need to be measured (Foster, 1995).

Running Time Analysis

Performance of an algorithm is highly depends on the characteristics of execution environment. Certainly a machine with higher computational power could have better performance. Therefore, the experiment must be carried out on the same environment. As shown in the Fig. 6 benchmarking has been used to compute the running time in microseconds. In this method the start and end point of code snippet must be marked. The difference between this two is the running time of that operation.

Implementation of the Proposed Method

This section discusses the implementation of the proposed algorithm for recoding binary numbers based on {0, 1, 3}-NAF algorithm. There are three stages in proposed algorithm namely:

Stage I: Look up table stage
Stage II: Partitioning
Stage III: Conversion stage

This stages have been successfully implemented in C++ using Visual Studio 2012 (32-bits). The proposed algorithm which is mainly falls in the first and second stages, have been developed and tested.

Stage I: Look up Table Stage

A look up table with the size of 256 integer elements has been generated. These integers are in the range of 0-255. Each integer in this range has a corresponding value to a look up table element in {0,1,3}-NAF representation. As this representation in look up table is in the similar base with {0, 1, 3}-NAF algorithm, the hamming weight is also same with the base algorithm result.

Stage II: Partitioning Stage

A block technique has been designed and developed to divide each input binary to n blocks of 8 bits length. In this method, the partition process will start from the right most side of the input binary. If the most left block is m bits shorter than 8 bits, the left side of this block will be filled by m zeros.

Stage III: Conversion Stage

Every block of binary number has an equivalent value in {0, 1, 3} - NAF. This value has been calculated and it is preserved in the look up table. In this stage each block of binary number will be represented as its {0,1,3}-NAF equivalent and these blocks will be placed consecutively.

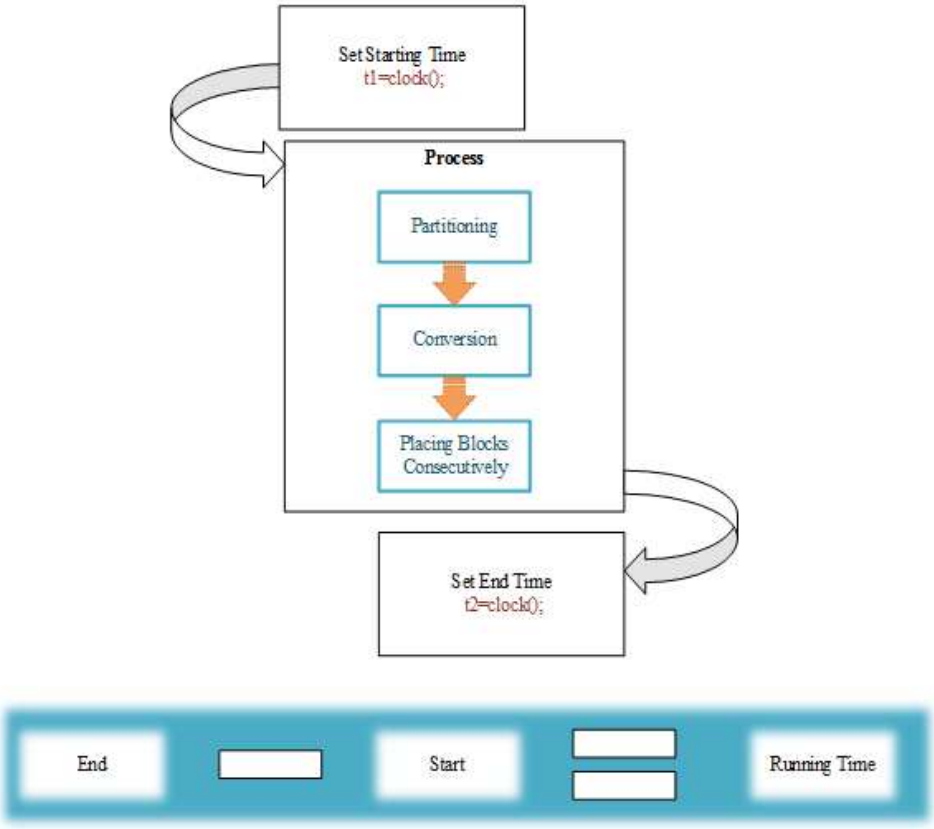


Fig. 6. Illustrates the method for running time computation

System Specification (Test Bed)

A Machine with the below specification has been used to carry out this experiment.

Results

In this section, there are two main analysis, one for the base and another one is for the proposed algorithm. The following analysis have been carried out:

- Algorithm complexity to find the significant efficiency to recognize the optimal algorithm
- Time performance to compare the real time performance of both algorithms in the same environment

As mentioned in (Gilberg and Forouzan, 2004), the most common way to compare the efficiency of two algorithms is to compute the Big *O*.

Moreover according to (Foster, 1995), in order to obtain an execution profile of an implemented algorithm, the behaviour of program can study. This experiment can assist to measure the outsider effects on the performance time such as initialization time, idle time and required time for each phases of computation.

To attain reliable result, the experiment must be performed in the same machine and with similar condition and to increase the validity of results, it must be repeated for several times. To address the above concerns the same machine with the specifications mentioned in Table 4 has been used. The system has been in the same state and the experiment has been repeated several times to show the difference between both algorithms in different numbers of run times.

The details of each analysis are presented in the following sub-sections.

Complexity Analysis

In this analysis, the complexity of base and proposed algorithm has been computed and compared. The complexity algorithm M is a function, $f(n)$ where the running time required for input data of size n. If algorithm contains no loop, f depends on the number of statements. Else f depends on number of elements being process in the loop (Gilberg and Forouzan, 2004). As it is explained in (Gilberg and Forouzan, 2004), based on the complexity of problem, different time is required. Accordingly the complexity of base and proposed algorithm has been computed and presented in Table 5.

Table 4. System specification (Test bed)

CPU	Intel(R) Core i5-3317U (1.7GHz)
RAM	10 GB
GPU	NVIDIA GeForce GT-620M
HDD	Hitachi hts545050A7E380 500gb
OS	Windows 7 Home Premium 64bit

Table 5. Complexity of base and proposed algorithm

Algorithms	Complexity
{0, 1, 3}-NAF	$f(n) = 21n + \frac{n^2+2}{16}$
Proposed algorithm	$g(n) = 4n + \frac{n}{8} + 3$
Reduction ratio	≈ 4.9

Table 6. Big O for base and proposed algorithm

Algorithms	Big *O*
{0, 1, 3}-NAF	$O(f(n)) = n^2$
Proposed algorithm	$O(g(n)) = n$

Table 7. Average performance time for different number of run times

	Average performance time (μs)			
Algorithms	1 time run	5 time run	10 time run	20 time run
{0, 1, 3}-NAF	3135.6	3276	3300.3	3570.7
Proposed algorithm	386.8	429.8	439.5	450.5
Increased performance	2748.8	2846.2	2860.8	3120.2

According to Table 5 the growth rate of the computed mathematical function is proportional to the presented value of the function. Therefore by increasing the input size the number of required operation will increase significantly. Whilst by increasing the size of input, the number of required operations for proposed algorithm has been increased linear. This is due to proposed look up table utilization in which there is no need to check the conditions and has no special cases. Moreover, in order to recode a binary by proposed algorithm the result of each block will consecutively be placed in order and not any operation more than that is required. Lesser number of operation is known as the reason of lesser complexity.

Figure 7 compares the growth rate of operations in base and proposed algorithms. It is clear that the growth rate for base algorithm is exponential.

The Big *O* for {0, 1, 3}-NAF and algorithm has been presented in Table 6.

According to Table 6 and with respect to algorithm efficiency order of magnitude in (Gilberg and Forouzan, 2004), $0(n)<0(n^2)$, the efficiency of proposed algorithm is significant. This complexity optimization in proposed algorithm is expected to cause significant enhancement in running time.

Running Time

In this section, performance time is the run time which is computed using benchmarking method. Read Time Stamp Counter (RDTSC) has been used to obtain the performance time. For applications that require accurate time-stamp counters, this instruction will count the number of processor cycles elapsed. The value returned by RDTSC indicates the number of processor cycles executed and not the number of seconds elapsed. Thus, to get the number of seconds elapsed, the returned value need to be divided by the processor frequency. This process has been applied on the base and proposed algorithm. The result of the performance time computation is analysed in the following sections (Intel Coorporation, 1997).

According to (Senne *et al*., 2000), both algorithms will run 1, 5, 10 and 20 times. This method has advantage to demonstrate the difference between both algorithms in different number of running. The computed average performance time is represented in Table 7 respectively.

Table 7 denotes the details of average performance time for 1, 5, 10 and 20 times run for base and proposed algorithm.

Overall, the performance time has dramatically decreased for proposed algorithm. For 1 time run proposed algorithm shows 87.6% speed up in comparison to base algorithm. According to (Foster, 1995) in order to enhance the accuracy of performance time calculation, the experiment has been repeated several times.

This replication process will actually minimize the effect of background activities in operating system. The performance of proposed algorithm has 86.8% increased for 5 times run against base algorithm. For 10 times run proposed algorithm shows 86.6% speed up in comparison to base algorithm. Whilst for 20 times run proposed algorithm has performed 87.3% times faster than base algorithm.

In average of 5 times run, the performance of base method had risen to just 3276 μs comparing with 1 time run. Although this appears on the graph to be a gentle increase, it is in fact an increase of approximately 4.4%. Average of 10 times run increased by 0.7% compare to 5 times run. However the greatest real interest was in average of 20 times run, where the performance time had increased by approximately 8.1% in comparison to 10 times run. Although performance time in proposed method increased between 1 times run to 20 times run, its increase is steadily.

Figure 8 shows the result for {0, 1, 3}-NAF and proposed algorithm. These results are computed from the average performance time for 1, 5, 10 and 20 run times. Overall, the chart shows that proposed method takes ~10 times less for conversion of same data in comparison with base method.

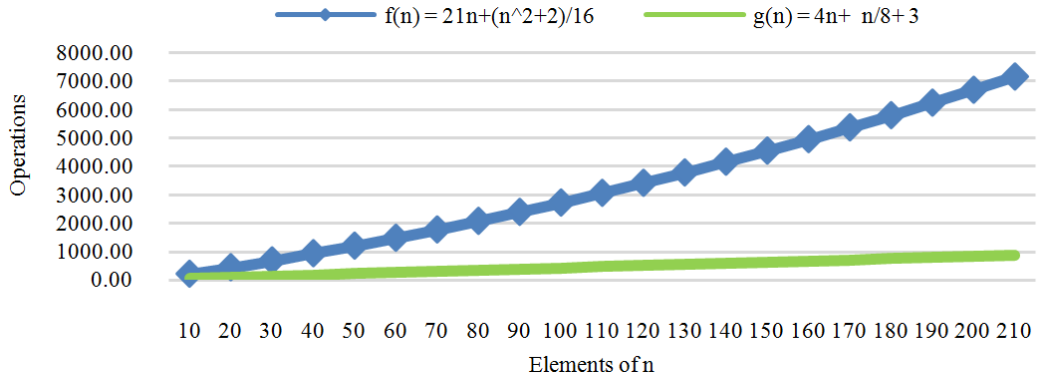


Fig. 7. Growth rate for base and proposed algorithm

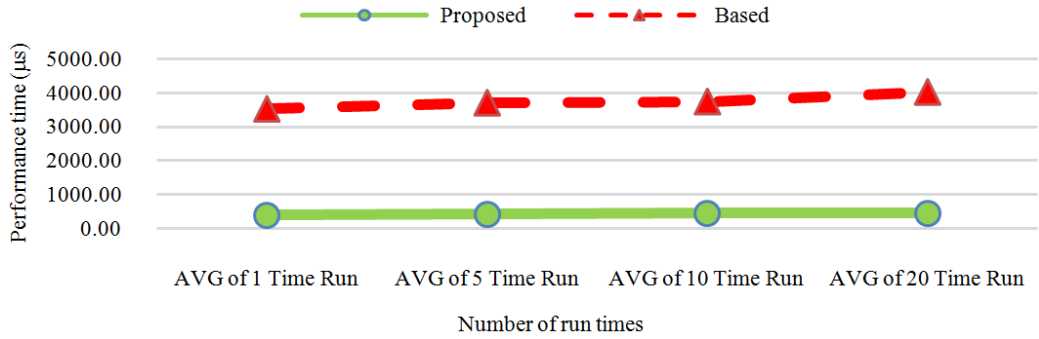


Fig. 8. Average of 1, 5, 10 and 20 times run for {0,1,3}-NAF and proposed algorithm

Table 8. Average performance time for different number of run times per each functions in proposed algorithm

	Average performance time (μs)			
Functions	1 time run	5 time run	10 time run	20 time run
Required time (%) in total	276.5	319.3	320.8	317.6
performance for partitioning	-71.50%	-74.30%	-73%	-70.50%
Required time (%) in total performance	109.3	108.3	115.3	132.5
for look up table	-28.25%	-25.19%	-26.20%	-29.30%

The base algorithm has only a look up table stage, in which all the conversion computations are performed. In contrast with the base algorithm, the proposed algorithm consist of a partitioning function and a fix look up table. Therefore, the total performance time in base algorithm has been elapsed in its look up table. Whilst in proposed algorithm different functions are defined and each requires different time.

Table 8 shows the elapsed time for different functions of the proposed algorithm. This results helps to compute the processing time and the total execution time. As the details indicates the summation of performance time for both functions in every running set of experiment is less than 100% which is the total performance time. The lost time here called elapsed time. This elapsed time is in range of 0.2% to 0.8%. According to (Foster, 1995) this elapsed time has been assumed as initialization of the algorithm or the OS backgrounds activity.

Based on the given information in Table 8, more than 70% of the total performance time in proposed algorithm has been elapsed to perform the partitioning stage of algorithm. From this information it can be concluded that the proposed look up table takes only about $\frac{l}{3}$ of total required performance time in each execution

Conclusion

The importance of algorithmic optimizations for increasing the performance for cryptography acceleration is proven. On the other hand, the importance of utilizing some pre-computation techniques such as blocking is significant.

A performance time problem in {0,1,3}-NAF recoding algorithm has been stated. A new fix look up table has been generated and the blocking method has

been applied. The complexity of base algorithm has been reduced and the performance time has been improved. In order to benchmark a dataset which is composed with all possible hamming weigh with the length of 24 bit binary has been used. As the future research direction and to extend this work, with a minor manipulation on the algorithm, it will take on bigger input size. The effect of resizing the blocks and extending the fixed look up table on performance enhancement can be study.

Funding Information

The authors would like to thank the Department of Computer Science, University Putra Malaysia for providing the facilities and financial support for this publication.

Author's Contributions

Mohsen Bafandehkar: Undertake the required design, implementation and experiments, measure and analyze the obtained results.

Sharifah Md Yasin: Supervise the research and prepare the workflow.

Ramlan Mahmod: Critical evaluation of the work and Technical advice.

Ethics

The authors have confirmed that there is not be any ethical issues involved with this manuscript.

References

Aranha, D.F., A. Faz-Hernández, J. López and F. Rodríguez-Henríquez, 2012. Faster implementation of scalar multiplication on Koblitz curves. Proceedings of the 2nd International Conference on Cryptology and Information Security in Latin America, (ICC' 12), Springer-Verlag Berlin, pp: 177-193. DOI: 10.1007/978-3-642-33481-8_10

Bafandehkar, M., S.M. Yasin and R. Mahmod, 2015. A literature review on scalar recoding algorithms in elliptic curve cryptography. Int. J. Commun. Antenna Propagat., 5: 183-183. DOI: 10.15866/irecap.v5i4.5673

Cormen, T.H., C.E. Leiserson, R.L. Rivest and C. Stein, 2001. Introduction to Algorithms. 2nd Edn., MIT Press, Cambridge, ISBN-10: 0262533057, pp: 1292.

Foster, I., 1995. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. 1st Edn., Addison-Wesley, Reading, Mass, ISBN-10: 0201575949, pp: 381.

Gilberg, R. and B. Forouzan, 2004. Data Structures: A Pseudocode Approach with C. 2nd Edn., Cengage Learning, Boston, ISBN-10: 0534390803, pp: 672.

Hankerson, D., A. Menezes and S. Vanstone, 2003. Guide to Elliptic Curve Cryptography. 1st Edn., Springer, Secaucus, ISBN-10: 038795273X.

Hitchcock, Y., E. Dawson, A. Clark and P. Montague, 2003. Implementing an efficient elliptic curve cryptosystem over GF (p) on a smart card. ANZIAM J., 44: C354-C377.

Intel Coorporation, 1997. Using the RDTSC Instruction for performance monitoring. Techn. Ber., Tech. Rep., Intel Coorporation.

Joye, M. and S.M. Yen, 2000. Optimal left-to-right binary signed-digit recoding. IEEE Trans. Comput., 49: 740-748. DOI: 10.1109/12.863044

Khabbazian, M., T.A. Gulliver and V.K. Bhargava, 2005. A new minimal average weight representation for left-to-right point multiplication methods. IEEE Trans. Comput., 54: 1454-1459. DOI: 10.1109/TC.2005.173

Kodali, R.K., K.H. Patel and N. Sarma, 2013. Implementation of energy efficient scalar point multiplication techniques for ECC. Int. J. Recent Trends Eng. Technol., 1: 14-19.

Longa, P. and H. Gebotys, 2010. Efficient techniques for high-speed elliptic curve cryptography. Proceedings of the 12th International Workshop on Cryptographic Hardware and Embedded Systems, Aug. 17-20, Springer, Santa Barbara, pp: 80-94. DOI: 10.1007/978-3-642-15031-9_6

Md Yasin, S., 2011. New signed-digit {0,1,3}-NAF scalar multiplication algorithm for elliptic curve over binary field. PhD Thesis, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia.

Morales-Sandoval, M. and C. Feregrino-Uribe, 2006. GF(2m) arithmetic modules for elliptic curve cryptography. Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGA's, 2006, Sept. 20-22, IEEE Xplore Press, San Luis Potosi, pp: 1-8. DOI: 10.1109/RECONF.2006.307768

Pathak, H.K. and M. Sanghi, 2010. Speeding up computation of scalar multiplication in elliptic curve cryptosystem. Int. J. Comput. Sci. Eng., 2: 1024-1028.

Reitwiesner, G.W., 1960. Binary arithmetic. Adv. Comput., 1: 231-308.

Sakthivel, A. and R. Nedunchezhian, 2012. Decreasing point multiplication over ECC (Zp) using tree computations. Proceedings of the International Conference on Computing, Communication and Applications, Feb. 22-24, IEEE Xplore Press, Dindigul, pp: 1-5. DOI: 10.1109/ICCCA.2012.6179229

Senne, E.L.F., L.A.N. Lorena, M. Laguna and J.L. Gonzalez-Velarde, 2000. Computing tools for modeling, optimization and simulation: Interfaces in computer science and operations research.

Shah, P.G., X. Huang and D. Sharma, 2010. Algorithm based on one's complement for fast scalar multiplication in ECC for wireless sensor network. Proceedings of the IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, Apr. 20-23, IEEE Xplore Press, Perth, WA., pp: 571-576. DOI: 10.1109/WAINA.2010.48

Yasin, S.M., R. Mahmod and R.N.H. Nor, 2015. Performance analysis of signed-digit {0,1,3}-NAF scalar multiplication algorithm in Lopez-Dahab model.

Yasin, S.M., R.N.H. Nor, J. Din and M. Zaitun, 2014. {0, 1, 3}-NAF representation and algorithms for lightweight elliptic curve cryptosystem in Lopez Dahab model. Int. Rev. Comput. Software, 9: 1541-1547.