Accelerate Performance for Elliptic Curve Scalar Multiplication based on NAF by Parallel Computing

Mohammad Anagreh[1,2], Eero Vainikko[1] and Peeter Laud[2]
[1] *Institute of Computer Science, University of Tartu, J. Liivi 2, Tartu, Estonia*
[2] *Cybernetica, Mäealuse 2/1, Tallinn, Estonia*

Keywords: ECC, NAF, Hamming Weight, Parallel Computing, Signed Binary Representation.

Abstract: The aim of Elliptic Curve Cryptosystems (ECC) is to achieve the same security level as RSA but with shorter key size. The basic operation in the ECC is scalar multiplication which is an expensive operation. In this paper, we focus on optimizing ECC scalar multiplication based on Non-Adjacent Form (NAF). A new algorithm is introduced that combines an Add-Subtract Scalar Multiplication Algorithm with NAF representation to accelerate the performance of the ECC calculation. Parallelizing the new algorithm shows an efficient method to calculate ECC. The proposed method has sped up the calculation up to 60% compared with the standard method.

1 INTRODUCTION

Elliptic curve cryptosystems (ECC) are powerful cryptosystem to encrypt data because they have a high-level of security but shorter key sizes compared with other existing algorithms such as RSA (Rivest et al., 1978). For example, the elliptic curve cryptosystems with 160-bit keys are considered to have comparable security to 1024-bit RSA (Gura et al., 2004). Several application fields of elliptic curve cryptosystems such as RFIDs, smart mobile, and smart cards, as the shorter key length also makes ECC better to be used on portable devices. There are different reasons that enable them to be used widely.

The most important operations in the ECC are the time-consuming scalar multiplications. Therefore, many researchers have focused to enhance and improve this area.

Elliptic curve cryptosystems (ECC) were independently proposed by two researchers, Koblitz (Koblitz, 1987) and Miller (Miller, 1986). The security of ECC is based on the Elliptic curve Discreet logarithm Problem (ECDLP), two given points in the elliptic curve P and Q, the equation is $Q = d.P$, where d is an integer converted to the binary or signed binary, while a given P is a point in elliptic curve.

The number of adding and doubling operations on an elliptic curve are based on length n of the scalar d. It is an integer which, for the purposes of the computation, is represented in binary — $d = \sum_{i=0}^{n-1} 2^i d_i$, $d_i \in \{0,1\}$ —, or in signed binary — $d = \sum_{i=0}^{n-1} 2^i d_i$, $d_i \in \{\bar{1},0,1\}$, where $\bar{1}$ is a '-1'. Several ECC researchers have been working to accelerate the performance of the ECC scalar multiplication by introducing new conversion algorithms to give low-Hamming-weight representations of d. Hamming Weight (HW) is the number of non-zero bits in the scalar representation d. Reducing the number of the bits '1' (or '$\bar{1}$') in the scalar representation will reduce the number of addition operations in the ECC scalar multiplication. Therefore, lower HW is preferred.

Several researchers have proposed new methods to convert the binary representation to the signed binary representation in order to reduce the Hamming Weight of the scalar d. These methods are Mutual Opposite Form (MOF) (Okeya et al., 2004), Joint Sparse Form (JSF) (Solinas, 2001), Non-Adjacent Form (NAF) (Booth, 1951). As well as, Complementary Recoding Technique (CRT) is proposed (Balasubramaniam and Kathikeyan, 2007) which enhanced to be Direct Recoding method (DRM) (Pathak and Sanghi, 2010) and other methods (Huang et al., 2010). Also, there are several methods proposed to reduce calculation time of the ECC scalar multiplication by parallel computing (Azarderakhsh and Reyhani-Masoleh, 2015) (Asif and Kong, 2017) (Gutub and Arabia, 2010).

In this paper, acceleration of the performance of the ECC scalar multiplication is proposed by parallelizing scalar multiplication algorithm (Add-subtract

Anagreh, M., Vainikko, E. and Laud, P.
Accelerate Performance for Elliptic Curve Scalar Multiplication based on NAF by Parallel Computing.
DOI: 10.5220/0007312702380245
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 238-245
ISBN: 978-989-758-359-9

scalar multiplication algorithm) and transforming the scalar from binary representation to the signed binary representation using the non-adjacent form (NAF) algorithm. Scalar multiplication performance can be improved by parallelizing the NAF algorithm.

This paper is organized as follows: Section 2 briefly introduces the ECC and scalar multiplication algorithms. Section 3 gives the preliminaries. Section 4 is the related work. Section 5 presents our approach. Section 6 is the results and discussion. The last section concludes the proposed method and discusses future works.

2 ELLIPTIC CURVE CRYPTOSYSTEM OVERVIEW

For cryptographic applications, elliptic curves are considered over two main kinds of finite fields: the binary fields F_{2^m} and the prime fields F_p, as described below.

2.1 Binary Fields F_{2^m}

The finite field F_{2^m} consists of 2^m elements, which are at most $(m-1)$-th degree polynomials over $Z_2 = \{0,1\}$, taken *modulo* an irreducible m-th degree polynomial. An elliptic curve over this field is determined by two elements $a,b \in F_{2^m}$ (Nichols, 1998). It consists of all pairs (x,y) of elements of F_{2^m} satisfying the following cubic equation:

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

Additionally, there is one more element called the *point at infinity*.

2.2 Prime Fields F_p

In this paper, we focus on the prime curves over F_p. The prime curves over F_p make use of the cubic equation as identified in Equation (2) with Cartesian coordinate variables (x,y) and coefficients (a,b) as elements of F_p. All the values can be considered integers that are computed *modulo* the prime number p (Stallings, 2005). The cubic equation with coefficients (a,b) and variables (x,y) for the elliptic curves over F_p is the following:

$$y^2 \mod = (x^3 + ax + b) \mod p \tag{2}$$

let the point $P = (x_1,x_2)$ and point $Q = (x_2,y_2)$ be in the elliptic curve over F_p, defined by the coefficients (a,b). In addition, let O be the point at infinity. The rules for addition operation in the EC is as follows:

$$P + O = P \tag{3}$$

Given point P and point Q, if $x_1 = x_2$ and $y_2 = -y_1$ then

$$P + Q = 0 \tag{4}$$

In general, $R = Q + P$, where the result $R = (x_3,y_3)$ is defined as follows:

$$x_2 = \lambda^2 - x_1 - x_2 \mod p \tag{5}$$

$$y_3 = \lambda(x_1 - x_3) - y_2 \mod p \tag{6}$$

$$\lambda = \begin{cases} \left(\frac{y_2-y_1}{x_2-x_1}\right) \mod p, & \text{if } P \neq Q \\ \left(\frac{3x_1^2+a}{2y_1}\right) \mod p, & \text{if } P = Q \end{cases} \tag{7}$$

3 PRELIMINARIES

3.1 Non-Adjacent Form (NAF)

This representation of integers was proposed by Reitwiesner (Reitwiesner, 1960). In Non-Adjacent Form (NAF), a third digit with the value -1 (subsequently denoted $\bar{1}$) is added to the representation of integers, beside the binary digits 0 and 1. The NAF representation corresponds to the same value as the binary representation but the difference appears in the representation itself. The goal of using such algorithms in the ECC is to reduce the Hamming Weight which is the number of non-zero bits in the key. Therefore, reducing the Hamming Weight will reduce the number of addition operations in the calculation of the scalar multiplication. So, calculating the scalar multiplication based on the signed binary representation will the reduce the total execution time of an ECC operation.

Example : d = 958, converting d to the binary representation is $(1110111110)_2$. The converting binary representation to the NAF is $(1000\bar{1}0000\bar{1}0)$. To prove the solution, let us convert the NAF to the decimal, d = 1024-64-2 = 958.
The hamming weight of 958 is reduced from 8 to 3. As known, one addition operation requires 2 squaring, 2 multiplications and 1 inversion. Using the NAF representations will save 10 squarings, 10 multiplications and 5 inversions.

```
Data: $n$-bit integer $K$
Result: $NAF$ Representation
        $(d_0, d_1, \ldots, d_{n-1})$
begin
    $C \leftarrow K$, $j \leftarrow 0$
1   while $C > 0$ do
        if $C$ is odd then
            $d_j \leftarrow 2$ - ($C$ mod 4)
            $C \leftarrow C$ - $d_j$
        else
            $d_j \leftarrow 0$
        end
        $C \leftarrow C/2$
        $j \leftarrow j + 1$
    end
    return $NAF$
end
```

Algorithm 1: NAF Representation Algorithm.

Algorithm 1 is the NAF Representation Algorithm. The input must be an integer and the output is the signed binary representation $(d_0, \ldots, d_{n-1})$, where $d_i \in \{\bar{1}, 0, 1\}$. This representation will be used in finding the scalar multiplication of the ECC.

3.2 ECC Scalar Multiplication

The scalar multiplication is the main operation in the ECC. Scalar multiplication is built up from two main operations — addition of points, and the doubling of a point. The scalar d — an integer — has to be converted to a bit-string. The occurrence of bit '1' in the representation corresponds to the operation of adding two points. There are approximately $n/2$ such additions in a scalar multiplication. On the other hand, the number of doubling operations is $n-1$. In case of signed binary representation, the third digit which is $\bar{1}$ will be processed by the subtracting operations.

Algorithm 2 is an Adding-Subtracting Scalar multiplication algorithm, which is used to compute the elliptic curve scalar multiplication based on a scalar $d = \sum_{i=0}^{n-1} 2^i d_i$, represented either in binary — $d_i \in \{0,1\}$ — or in signed binary — $d_i \in \{\bar{1}, 0, 1\}$

4 RELATED WORK

The improvement of the scalar multiplication can be achieved by improving or proposing some related algorithms in the scalar multiplication. Applying the signed binary representation algorithms to find the scalar multiplication is an efficient way to reduce the number of non-zero bits in the key. Hamming Weight is a big player to reduce the number of addition operations in computing the scalar multiplication.

```
Data: Point on EC $P$, a non-zero string
      $(d_0, \ldots, d_{n-1})$ representing $d$
Result: $Q = dP$
begin
    $Q \leftarrow 0$, $R \leftarrow P$
1   for $i$ = 0 to n-1 do
        if ($d_i$ = 1) then
            $Q \leftarrow Q + R$
        else if ($d_i$ = $\bar{1}$) then
            $Q \leftarrow Q$ - $R$
        end
        $R \leftarrow 2R$
    end
    return $Q$
end
```

Algorithm 2: Adding-Subtracting Scalar Multiplication Algorithm.

There are many methods to represent integers in signed binary such as NAF, JSF and MOF, Also in 2003 a new method to compute general multiplication was proposed by Change et al (Chang et al., 2003) which is the result of using the NAF, MOF and JSF.

Different researchers proposed methods to calculate the scalar multiplication in parallel computing using the binary or signed binary representation.

Ansari et al (Ansari and Wu, 2005), proposed a parallel method based on task decomposition, to parallelize the ECC scalar multiplication. The binary representation and double-and-add algorithm are used to compute the ECC scalar multiplication.

In different fields of the applications such as in a cloud computing need high-speed applications, Chung et al (Chung et al., 2012) proposed a new elliptic curve cryptography (ECC) processor architecture. The proposed processor requires much fewer execution cycles than that of previous methods which includes a 3 pipelined-stage full-word Montgomery multiplier. As well as, the time-cost precomputation steps of Montgomery modular multiplication are achieved by the processor to reach real-time requirement. To improve hardware performance, parallelization techniques and hardware sharing are used. The results show that by using the proposed elliptic curve cryptosystem processor in comparison with relative works is 25% faster.

Pabbuleti et al (Pabbulti et al., 2013), proposed vector processing techniques to accelerate modular multiplications in prime fields. They implemented their proposed work on two different embedded computing platforms which are Intel Atom N2800 and Qualcomm Snapdragon APQ8060. As well as, they used a NIST- standard Prime filed curve. The result shows the proposed technique is faster than the OpenSSL versions of the same ECC operations two times.

Anagreh et al (Anagreh et al., 2014), proposed a parallel method to compute scalar multiplication based on the mutual opposite form (MOF). They extracted a new algorithm that combined Adding-Subtracting Algorithm and mutual opposite form (MOF). The Method calculates the doubling operation and addition operation at the same time without performing the MOF conversion. The proposed method is performing the comparison operation of the bit-string to decide that the second processor has to perform the addition operation in case of non-zero bits. Robert (Robert, 2014) proposed different software implementations for finding the elliptic curve scalar multiplication. He tested the implementations within two threads, and four threads algorithms on various elliptic curves over the prime F_p and the binary fields F_{2^m}. The scalar multiplication is performed by using the *Double-and-add* and *Halve-and-add* algorithms.

This work talks about putting doubling operations into one thread (producer) while additions and subtractions into another thread (consumer). His method was to avoid using the mutex synchronization as much as possible by using one single mutex at the beginning of the computation. The goal of using the mutex is to keep the thread which is responsible for performing the addition or subtraction in an inactive state. At the same time, the first batch of doubling operations is computed by the producer thread.

The method shows some violation of the read-after-write dependency. The memory violation may happen because the size of the first batch of points which is before releasing the mutex was too small. As well as, in the case of the long sequence of zeros in the binary or NAF scalar representation. The results have shown that this error rate is limited to less than 1%, but is not acceptable because that causes the corruption of the calculation.

The variable which is stored in the global memory as the loop counter is used to eliminate this problem, this allows the checking if the addition operation uses a point which has already been produced to the shared memory from the producer thread. The test which is used on the addition-thread is an extra operation that will reduce the parallel time. Computing the scalar multiplication is based on the NAF representation which is already given. The conversion method is not a part of the proposed work. The software implementations achieved an enhancement 15% in the comparison with their sequential case.

New parallel approaches are proposed by Negre et al (Negre and Robert, 2015), they introduced parallel approaches which split the scalar multiplication based on NAF into two parts for the prime field F_p, and three parts for the binary field F_{2^m}. In this approaches, both operations (doubling and addition) or subtraction will be performed in each thread, the splitting is for finding scalar multiplication itself. In the case of the prime field, they divided the scalar multiplication into two parts, the first part $Q_1 = k_1P$ will be performed in a thread and the second part $Q_2 = 2^s k_2 P$ will be performed in another thread. Then at the end, the two points Q_1 and Q_2 are added to get the scalar multiplication $Q = Q_1 + Q_2$. Computing the scalar multiplication by parallel calculation is based on the window size of Non-Adjacent Form (w-NAF). The result shows, that the proposed approaches achieved an improvement by at least 10% the computation time of the scalar multiplication.

Liu et al (Liu et al., 2016), proposed ECC software implementation based on the NIST curve for wireless sensor nodes and similar devices equipped with an 8-bit AVR processor. The paper includes a thorough evaluation of different protocols, tools, and platforms. They demonstrated that the performance of their elliptic curve cryptosystem implementation through three widely-used protocols, ECDH, ECDSA, and ECMQV. In comparison to the state-of-the-art, the proposed ECC implementation which uses the ECDH and ECDSA achieve an enhancement by factors of 1.35 and 2.33, respectively.

An optimal representation for right-to-left parallel elliptic curve scalar multiplication is introduced by Phalakarn et al (Phalakarn et al., 2018). They devised a mathematical model that will reduce the calculation time. As well as, proposed algorithms that will generate the representations which will reduce the execution time of the model. The optimal representation which is generated for multi-scalar point multiplication is under a condition. That a three processor will perform the calculations, two for doubling P, Q and the third processor for addition operation using two binary representation n, m. They improved the parallel double-and-add model by defining the time which used in the model when performing the calculation. Anyway, the issue of the communication time between the processors in the model is still opened and is not considered yet. That might affect the proposed optimal representation especially in the case of using three processors or more.

5 PROPOSED WORK

The minimum key size of the ECC is 160 bits, this size can ensure same security level as 1024-bits key size of the RSA. Therefore, reducing the execution

```
   Data: Integer $k$, Point in EC $P$
   Result: $Q = kP$, based on (NAF)
   Processor 1 performs Doubling Operations
     (DBL):
   begin
       $R \leftarrow P$
1      for $i$ = 0 to n-1 do
           $R \leftarrow 2R$
           $A_i \leftarrow R$
       end
   end
   Processor 2 performs NAF conversion,
     followed by Addition Operations (ADD):
   begin
       $C \leftarrow k$, $j \leftarrow 0$
2      while $C > 0$ do
           if $C$ is odd then
               $d_j \leftarrow 2$ - ($C$ mod 4)
               $C \leftarrow C - d_j$
           else
               $d_j \leftarrow 0$
           end
           $C \leftarrow C / 2$, $j \leftarrow j + 1$
       end
3      for $y$ = 0 to n-1 do
           $T \leftarrow A_y$
           $m \leftarrow d_y$
           if $m$ = 1 then
               $Q \leftarrow Q + T$
           else if $m$ = -1 then
               $Q \leftarrow Q - T$
           end
       end
       return $Q$
   end
```

Algorithm 3: Parallel Scalar Multiplication based on NAF (PECC-NAF) Algorithm.

time of scalar multiplication by applying some an efficient method is desired. In that case, we can increase the size of the key that will increase the security level of the ECC with the same execution time. In this work, we propose a parallel method that calculates the ECC scalar multiplication based on signed binary representation algorithm which is the NAF. We calculate the scalar multiplication by applying the add-subtract scalar multiplication algorithm. The proposed method combines add-subtract scalar multiplication algorithm with the non-adjacent form algorithm. We extracted a new algorithm (PECC-NAF) to perform the proposed scheme, see Algorithm 3.

Our parallel method strategy in this work is to parallelize the algorithm 3. Task decomposition strategy is used to perform the parallel calculation by splitting the adding-subtracting scalar multiplication algorithm into two parts.

Part 1 will be processed by Processor-1 which computes the $n-1$ doubling operations and saves that doubled points in the shared array A. We can recognize that there is no dependency between the kind of bits $\{\bar{1}, 0, 1\}$ and doubling operations. Processor-1 has to perform the doubling operations based on the length of the key $n-1$, regardless of whether the bit is '1', '0' or '$\bar{1}$'. In this case, if the key size is 160-bits, Processor-1 has to perform the doubling operations 159 times. Also, Processor-1 has to save the doubled point in the shared array A, see Algorithm 3. The size of the array A is the same size of n.

Part 2 of the algorithm, is to find the NAF representation by applying NAF algorithm. Performing the addition operations in the part 3 of the algorithm (both part 2 and part 3 will be performed in the processor-2). The total number of addition operations is equal to the Hamming weight (the number of non-zero bits) of the key. Processor-2 has to find the NAF representation before performing the addition operations. According to the kind of the bits (1 or $\bar{1}$) in the NAF representation, Processor-2 has to perform the addition or subtraction operations.

In case the bit is '1', Processor-2 perform the addition operation and saves the result in accumulator Q, while if the bit is '$\bar{1}$', performs the subtracting operations and saves the result in the Q. If the bit is '0', Processor-2 does not have to perform any elliptic curve operations.

Processor-2 performs addition or subtraction operations by reading the saved points $(2P, 4P, \ldots, 2^{n-1}P)$ in the shared array A, see Figure 1. Both processors

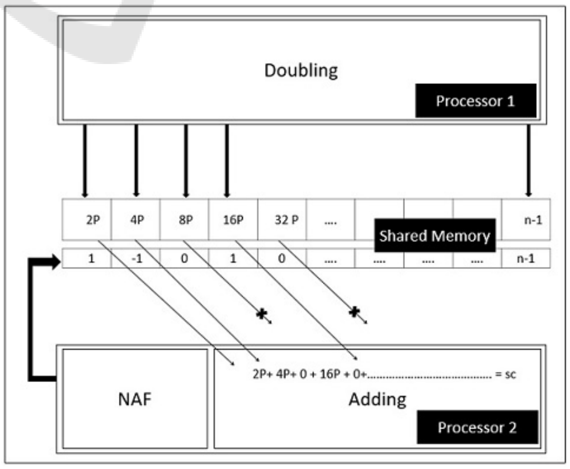


Figure 1: General Framework of Scheme.

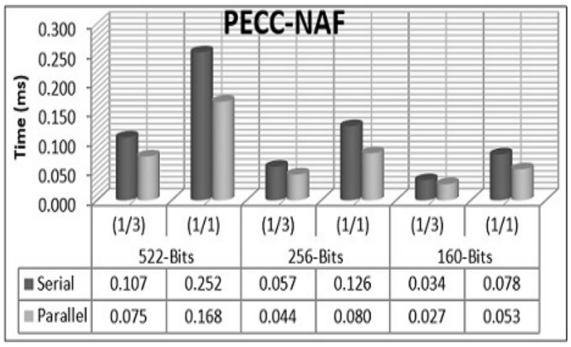
have the ability to access the shared array A which is located in the shared memory. Processor-1 perform the writing in the array A, while Processor-2 perform the reading in the array. We should carefully manage the reaching to the shared array. Processor-1 should be able to write only, while Processor-2 should only be able to read. Also, Processor-2 should read only from the locations which already contain doubled points. We achieve this by making Processor-1 write the doubled points in the shared array A,

Processor-2 computes the NAF representation before performing the addition operations, in order to avoid the bugs or reading some rubbish data from the memory, while processor-1 performs the doubling operation and save the result in the array A. In our scheme, Processor-2 perform the NAF conversion before starting to perform the addition operations. After performing the doubling operations, Processor-1 will save the results (doubled points) in the shared array A. In this case, doubled points are already saved in the memory and ready to be read by Processor-2. So, we can say that our method is managed automatically without adding some extra code or protocols to manage the synchronization of the memory accesses. During the execution time of finding the NAF representation by Processor-2, Processor-1 performs the doubling operations and save the points in the shared memory. So, some doubled points stored already in the array A by Processor-1 to be readable by Processor-2 once finish the converting to the NAF. Processor-1 will write the points in location d_i, while Processor-2 will read the points from location d_j, where $d_i > d_j$.

6 RESULTS AND DISCUSSION

We can summarize that the proposed method is extracting a new algorithm that combines two algorithms: Add-Subtract Scalar Multiplication, and Non-Adjacent Form. It performs the parallel computing on the extracted algorithm (PECC-NAF), given in Algorithm 3. In Figure 2, we can recognize clearly the performance result we get from the real implementation of our proposed scheme. In the implementation, we tested three different key sizes: 160-bits, 256-bits, and 522-bits. Each key size is represented with two fractions of occurrence of the non-zero bits in the key given by N = Total of non-zero bits / key-size.

The result shows that the best case is when the number of non-zero bits is close to the key size. The the best difference time (between parallel and serial versions) we got in case of key-size is 256-bits and 522-bits in case of $N = 1$. Its important to note, we tested our implementation when $N = 1$ to get the variance time between the parallel version and standard version to test the parallel method, not to get the best



	(1/3)	(1/1)	(1/3)	(1/1)	(1/3)	(1/1)
	522-Bits		256-Bits		160-Bits	
■ Serial	0.107	0.252	0.057	0.126	0.034	0.078
■ Parallel	0.075	0.168	0.044	0.080	0.027	0.053

Figure 2: The Performance Results.

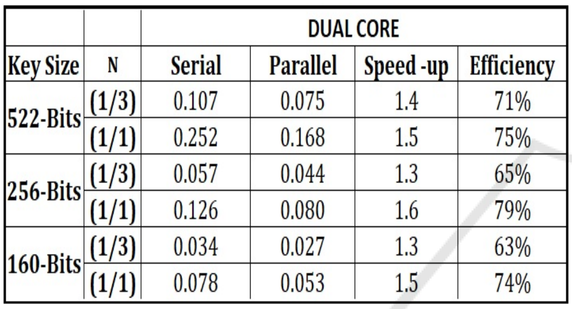
result. In the Figure 3, we can see that the execution time is high cost in the case $N = 1$. In the real-time, the number of hamming weight in the NAF representation is less than in the binary representation, it's rarely to be $N = 1$.

As with almost all parallel applications, it's important to produce the best sequential code before starting to parallelize the code. Task decomposition is used to divide the work into two Processors to perform the overall scheme to get the best result as shown in Figure 2. We wrote both the sequential and the parallel code in Visual C++.Net. We use the Open MP Library that is supported in the Visual C++.Net package in order to write the parallel section in the parallel version. It's important to note that we use Intel Dual-Core machine to test both versions using Windows 7. The specification of the machine is Intel Dual core with 1 MB L2 cache memory. We performed each key size in both N cases 10 times and the average execution time is taken as shown in Figure 3.

The execution of the code for both serial and parallel versions show that the result of the scalar multiplication is the same in both cases. The scalar multiplication in the parallel version (proposed method) is the same as the scalar multiplication in the serial version (standard method), meaning that Processor-2 reads the doubling point from the location in the array correctly to perform the addition operation. In such cases, we have to organize the reading and writing operations correctly to avoid reading rubbish data from the memory.

In our proposed method as we can see in Algorithm 3, Processor-1 has to write the doubled point R_i in the shared array A each round n_i in the whole rounds of the key size n. The writing is always performed in a different location of the array A respectively. Each point has a unique location that will be readable from Processor-2. There is no possibility to write two points at the same location because the doubling operation is done serially. Processor-2 perform the NAF conversion before performing the addition or subtraction operations. while at the same time Processor-1 performs the doubling operations and writes the doubled points in the array. Processor-2 reads the doubled points from the shared array A and performing either addition operation in case the d_i is 1 or performing the subtraction operation in case the d_i is $\bar{1}$. In case the d_i is 0, Processor-2 ignores performing the addition or subtraction in that location, see in Figure 1.

We tested our implementation several times, we did not record any occurrence of error. The main point in our implementation is that Processor-1 writes the doubled point in location d_i while Processor-2 reads the doubled point from location d_j, while $d_i > d_j$. As well as, Processor-2 starts reading the doubled points after finding the NAF conversion.



		DUAL CORE			
Key Size	N	Serial	Parallel	Speed -up	Efficiency
522-Bits	(1/3)	0.107	0.075	1.4	71%
	(1/1)	0.252	0.168	1.5	75%
256-Bits	(1/3)	0.057	0.044	1.3	65%
	(1/1)	0.126	0.080	1.6	79%
160-Bits	(1/3)	0.034	0.027	1.3	63%
	(1/1)	0.078	0.053	1.5	74%

Figure 3: The time in milliseconds.

The aim of our scheme is to reduce the execution time of computing the ECC scalar multiplication. Therefore, we can exploit the time variance between the parallel execution and the standard case to increase the key size of the ECC. The execution time of the expanded key size will be the same as the execution time of the sequential version.

For example, consider the key size of 160-bits and $N = 1/3$. Both the serial and parallel time (in milliseconds) can be looked up from Figure 3. Assuming, the key size is 160-bits, the execution time of the serial version as shown in Figure 3 is 0.034 ms and parallel time is 0.027 ms. The difference between them is 0.007 ms. The execution time for each iteration in parallel time is around 0.027 ms/160 = 0.00016 ms. Therefore, we can increase the key size according to the difference among serial and parallel time as follows: 0.007 / 0.00016 = 43.3 iterations, thus we can expand the key size to 43 bits higher than 160-bits. We can summarize, that by applying our scheme the execution time of the scheme with a key size of 203-bits gives us the same execution time as the serial version with 160-bits. And therefore, the security level of the ECC with 203-bits key size is more than the security level of the ECC with 160-bits key size.

In the best cases of our method as 256-bits while $N = 1$, the speed-up is around 1.6 times. Let us test another best case of our method which is 160-bits while $N = 1$, the speed-up is around 1.5x. The time difference between serial time and parallel time is 0.025ms. The execution time for one iteration in the case of parallel version is 0.053 ms/160 = 0.00033 ms. Therefore, we can increase the key size by 0.025/0.00033 = 75.7 bits, then we can expand the size of the ECC key 75-bits more and of course, that will increase the security level of the scheme. As we mentioned above, the security level of the ECC with 160-bits key size is the same security level of the RSA with 1024-bits key size. In case the RSA has a key size 2048-bits, it's considered as the same security level of the ECC which has 224-bits key size. Therefore, our method will decrease the execution time in the best cases around 50% to 60%. So, we can increase the security level of the standard scheme by applying our proposed method. Increasing the security level by expanding the key size 160 + 75 = 235 bits which have the same execution time of the standard method.

We can summarize, the execution time of finding the ECC scalar multiplication with a key size 235- bits in the parallel version is the same execution time of finding the ECC scalar multiplication with key size 160-bits in the serial version. Therefore, the security level of the proposed method is reached to the 235-bits which is more than the security level of 2048-bit key size of the RSA (235-bits > 224-bits), and that is the significance of the proposed method.

7 CONCLUSIONS

In this work, we extracted a new algorithm that combines the adding-subtracting scalar multiplication algorithm with non-adjacent form (NAF). We parallelized the proposed algorithm using two processors. The first processor performs the doubling operations while the second processor performs the NAF conversion and adding or/and subtracting operations. We tested our proposed method with different key sizes. In Processor-2, finding the NAF conversion organized to be before performing the addition and/or subtraction to avoid any error occurrence. Processor-1 supports the doubled point to the Processor-2. The difference in hamming weight can affect the execution time in both parallel and serial versions.

The future work is testing almost signed binary representation algorithms such as MOF, JSF complementary method, and other signed binary representation methods. Testing our proposed method is by using two processors and passing the data among two processors using a shared memory. We assume that using three processors will reduce the execution time more, especially in performing the addition operations.

REFERENCES

Anagreh, M., Samsudin, A., and Omar, M. (2014). Parallel method for computing elliptic curve scalar multiplication based on mof. *Int. Arab J. Inf. Technol*, 11(6):521–525.

Ansari, B. and Wu, H. (2005). Parallel scalar multiplication for elliptic curve cryptosystems. In *Proceedings of International Conference on Communications, Circuits and Systems, vol. 1*, pages 71–73.

Asif, A. and Kong, Y. (2017). Highly parallel modular multiplier for elliptic curve cryptography in residue number system. *Circuits, Systems, and Signal Processing*, 36(3):1027–1051.

Azarderakhsh, R. and Reyhani-Masoleh, A. (2015). Parallel and high-speed computations of elliptic curve cryptography using hybrid-double multipliers. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1668–1677.

Balasubramaniam, P. and Kathikeyan, E. (2007). Ellptic curve scalar multiplication algorithm using complementary recoding. *Applied mathematics and computation*, 190(1):51–56.

Booth, A. (1951). A signed binary multiplication technique. *Journal of Applied Mathematics*, 4:236–240.

Chang, C., Kuo, Y., and Lin, C. (2003). Fast algorithms for common-multiplicand multiplication and exponentiation by performing complements. In *Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on*, pages 807–811. IEEE.

Chung, S., Lee, J., chang, C., and Lee, C. (2012). A high-performance elliptic curve cryptographic processor over gf(p) with spa resistance. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 1456–1459.

Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S. (2004). Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Proceedings of the International workshop on cryptographic hardware and embedded systems*, pages 119–132. Springer.

Gutub, A. and Arabia, S. (2010). Remodeling of elliptic curve cryptography scalar multiplication architecture using parallel jacobian coordinate system. *International Journal of Computer Science and Security (IJCSS)*, 4(4):409–ff.

Huang, X., Shah, G., and Sharma, D. (2010). Minimizing hamming weight based on 1's complement of binary numbers over gf (2 m). In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, pages 1226–1230.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209.

Liu, Z., Seo, H., Großschädl, J., and Kim, H. (2016). Efficient implementation of nist-compliant elliptic curve cryptography for 8-bit avr-based sensor nodes. *IEEE Transactions on Information Forensics and Security*, 11(7):1385–1397.

Miller, V. (1986). Use of elliptic curves in cryptography. In *Advances in Cryptology, Proceedings of CRYPTO85 (LNCS 218)*, pages 417–426. Springer.

Negre, C. and Robert, M. (2015). Parallel approaches for efficient scalar multiplication over elliptic curve. In *SECRYPT: International Conference on Security and Cryptography*, pages 202–209.

Nichols, R. K. (1998). *Biometric Encryption*, chapter 22. McGraw-Hill.

Okeya, K., Schmidt-Samoa, K., Spahn, C., and Takagi, T. (2004). Signed binary representations revisited. In *Annual International Cryptology Conference, CRYPTO 2004*, pages 123–139. Springer.

Pabbulti, K., Mane, H., Desai, A., Albert, C., and Schaumont, P. (2013). Simd acceleration of modular arithmetic on contemporary embedded platforms. In *High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE.

Pathak, K. and Sanghi, A. (2010). Speeding up computation of scalar multiplication in elliptic curve cryptosystem. *International Journal on Computer Science and Engineering*, 2(4):236–240.

Phalakarn, K., Phalakarn, K., and Suppaktpaisarn, V. (2018). Optimal representation for right-to-left parallel scalar and multi-scalar point multiplication. *International Journal of Networking and Computing*, 8(2):166–185.

Reitwiesner, G. (1960). Binary arithmetic. *Advances in Computers*, 1:231–308.

Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Robert, M. (2014). Parallelized software implementation of elliptic curve scalar multiplication. In *International Conference on Information Security and Cryptology*, pages 445–462. Springer.

Solinas, J. (2001). Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, Center for Applied Cryptographic Research, University of Waterloo.

Stallings, W. (2005). *Cryptography and Network Security Principles and Practices*. Prentice Hall, 4th edition.