Speeding Up the Computation of Elliptic Curve Scalar Multiplication based on CRT and DRM

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, Parallel Computing, CRT, DRM.

Abstract: In this paper, we study the parallel implementations of elliptic curve scalar multiplication over prime fields using signed binary representations. Our implementation speeds up the calculation of scalar multiplication in comparison with the standard case. We introduce parallel algorithms for computing elliptic curve scalar multiplication based on representing the scalar by the Complementary Recoding Technique (CRT) and the Direct Recording Method (DRM). Both implementations of the proposed algorithms show speed-ups reaching up to 60% in comparison with execution time for sequential cases of the algorithms. We find that ECC-DRM is faster than ECC-CRT in both parallel and sequential counterparts.

1 INTRODUCTION

Elliptic curve cryptosystems (ECC) were independently proposed by Koblitz (Koblitz, 1987) and Miller (Miller, 1986). They are widely used in many cryptographic primitives and protocols such as asymmetric encryption, digital signature and key exchange. One of the most important advantages of ECC is its suitability for using it in case of limited memory resources, such as portable devices, because it has a shorter key size. ECC shows a high-level of security with shorter key sizes in comparison with other existing algorithms like RSA (Rivest et al., 1978). The minimum key size of the ECC is 160-bits having the same security level as a standard key size of RSA of 1024-bits (Gura et al., 2004). Computing the scalar multiplication is an expensive operation in the elliptic curve cryptosystem. Elliptic curve scalar multiplication is the operation of successively adding an EC point along an elliptic curve to itself d times repeatedly: $Q = dP$, where $P = (x, y)$ is a given point on the elliptic curve. The multiplication algorithms typically consider the binary representation of d. Therefore, many researchers have focused to enhance the calculation of scalar multiplication by proposing new related algorithms such as signed binary representation, as well as by enhancing the calculation method itself such as using a parallel calculation. The Hamming Weight (HW) of a (signed) binary representation of d is the number of non-zero bits in it. The number of adding and doubling operations on an elliptic curve scalar multiplication is based on the length n of the binary representation of d.

Reducing the number of non-zero bits in the scalar representation d will reduce the number of adding operations in the ECC scalar multiplication. Therefore, lower HW is preferred to be used in the ECC scalar multiplication. Several researchers have proposed new methods to convert the binary representation to some signed binary representation in order to reduce the Hamming Weight of the representation of d. These representations are Mutual Opposite Form (MOF) (Okeya et al., 2004), Joint Sparse Form (JSF) (Solinas, 2001), Non-Adjacent Form (NAF) (Booth, 1951). In this paper, we consider Complementary Recoding Technique (CRT) (Balasubramaniam and Kathikeyan, 2007), which enhanced by Direct Recoding method (DRM) (HK and Sanghi, 2010) and other methods (Huang et al., 2010). On the other hand, there are several methods proposed to accelerate the calculation of the ECC scalar multiplication by parallel computing (Azarderakhsh and Reyhani-Masoleh, 2015) (Asif and Kong, 2017) (Gutub, 2010).

In this paper, we propose algorithms to accelerate the performance of computing elliptic curve scalar multiplication by parallelizing the scalar multiplication algorithm. The proposed algorithms are based on combining the Add-subtract scalar multiplication algorithm and transforming the scalar d from the binary representation to the signed binary representa-

Anagreh, M., Vainikko, E. and Laud, P.
Speeding Up the Computation of Elliptic Curve Scalar Multiplication based on CRT and DRM.
DOI: 10.5220/0009129501760184
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 176-184
ISBN: 978-989-758-399-5; ISSN: 2184-4356

tion. One of our algorithms makes use of the Complementary Recoding Technique (CRT), while the other one is based on the Direct Recoding method (DRM). For both representations, we consider different ways of scheduling the computation on two processors. Our implementation of the two algorithms shows that the proposed methods are faster than the sequential calculation of the ECC scalar multiplication.

This paper is organized as follows: Section 2 briefly presents the preliminaries. Section 3 shows some related work while section 4 is the proposed work and the algorithms. Section 5 shows the results and presents the experiments. The last section concludes the proposed method and discusses future work.

2 PRELIMINARIES

2.1 Elliptic Curves over Prime Fields F_p

In this paper, we focus on the curves over prime fields F_p. These curves are defined through 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. The cubic equation with coefficients (a,b) and variables (x,y) for the elliptic curves over F_p is the following:

$$y^2 = (x^3 + ax + b) \ mod \ p \tag{1}$$

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:

$$P + O = P \tag{2}$$

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

$$P + Q = 0 \tag{3}$$

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

$$x_2 = \lambda^2 - x_1 - x_2 \ mod \ p \tag{4}$$

$$y_3 = \lambda(x_1 - x_3) - y_2 \ mod \ p \tag{5}$$

$$\lambda = \begin{cases} \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \ mod p, & \text{if } P \neq Q \\ \left(\frac{3x_1^2 + a}{2y_1}\right) \ mod p, & \text{if } P = Q \end{cases} \tag{6}$$

In summary, for any two points P, Q on a given elliptic curve, there are two main operations. The operation $R = P + Q$ when $P \neq Q$ is called point addition and $R = 2P$ is called point doubling. Addition operation has 5 sub-operations: 2 squaring, 2 multiplications and 1 inversion. Consequently, for non-negative integer number d, it is possible to define the scalar point multiplication $Q = dP$ on the elliptic curve through the application of doubling and adding operations, illustrated in Figure 1.

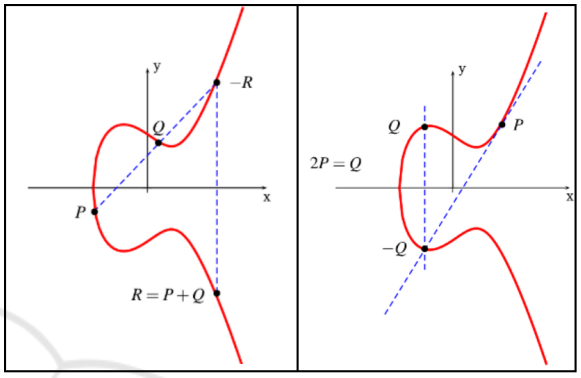

Figure 1: Adding and doubling points on EC.

2.2 Signed Binary Presentation

A *signed binary representation* of d is a vector $(d_0, \ldots, d_{n-1})$, where $\sum_{i=0}^{n-1} 2^i d_i = d$ and each d_i is an element of $\{-1, 0, 1\}$. Aiming to reduce the Hamming weight of the representation, a number of different signed binary representations have been proposed, including MOF (Okeya et al., 2004), NAF (Booth, 1951), CRT (Balasubramaniam and Kathikeyan, 2007), DRM (HK and Sanghi, 2010) and others.

In the following, we denote $\overline{1} = -1$, $\overline{0} = 0$, and $\overline{-1} = 1$.

2.2.1 Complementary Recoding Technique (CRT)

CRT is one of the techniques to convert a number to a canonical signed binary representation that reduces the Hamming weight (Balasubramaniam and Kathikeyan, 2007). If d denotes an n-bit integer, as well as its (usual) binary representation, then its CRT representation is $\sum_{i=0}^{n-1} 2^i d_i = (100...0)_{(n+1) \text{ bits}} - \hat{d} - 1$, where $\hat{d} = 2^n - 1 - d$ denotes the binary complement of d. This conversion is very simple, efficient and low time complexity in comparison with other methods (HK and Sanghi, 2010).

Example 1: Let d = 7327, its binary representation is $(1110010011111)_2$. Converting the binary repre-

sentation to signed binary representation by applying CRT is $d = \sum_{i=0}^{n-1} 2^i d_i$
$= (100...0)_{(n+1)bits} - \hat{d} - 1$ = $(10000000000000)_2$ - $(0001101100000)_2$ - 1 = $(1000\overline{1}10\overline{1}10000\overline{1})_2$. Indeed, converting the signed binary representation to decimal, we get $(1000\overline{1}10\overline{1}10000\overline{1})_2 = 8192 - 512 - 256 - 64 - 32 - 1 = 7327 = d$.

The Hamming weight for the binary representation of 7327 is 9, while the Hamming weight for signed binary representation using CRT is 6. Smaller hamming weight will save the number of operations of calculating the EC scalar multiplication.

2.2.2 Direct Recoding Method (DRM)

DRM is another converting method to signed binary representation (HK and Sanghi, 2010). This method is based on the CRT but with time complexity less than CRT because it uses only the single operation of bitwise subtraction with $0 - 1 = \overline{1}$. Also, DRM generally results in smaller Hamming weight of the result than CRT (HK and Sanghi, 2010).

The procedure to convert d to the signed binary representation using DRM is the following. Let p be the integer satisfying $2^p \geq d > 2^{p-1}$. then $d = (2^p)_2 - (2^p - k)_2$, where the subtraction of the bit 1 from the bit 0 results in $\overline{1}$.

Example 2: Let d = 248. The binary representation of d is $(11111000)_2$. Converting the binary representation to the signed binary representation by applying DRM as follows:
$2^8 = (100000000)_2$ and $(2^8 - 248) = (1000)_2$. Then $d = (100000000)_2 - (1000)_2 = (10000\overline{1}000)_2$.
Indeed, let us convert the signed binary representation $(10000\overline{1}000)$ we got by applying the DRM to decimal, $d = 256 - 8 = 248$. The hamming weight for the binary representation of 248 is 5, while the hamming weight for signed binary representation using DRM is 2. So, the conversion will bring savings during the calculation of the EC scalar multiplication. Note that the signed binary representation of 248 using CRT is $(10000\overline{1}\overline{1}\overline{1}\overline{1})_2$, which has the Hamming weight 5.

2.3 ECC Scalar Multiplication

The scalar multiplication is one of the main operations in the ECC. Scalar multiplication is built up from two main operations — the addition of points, and the doubling of a point. The scalar d is an integer that has to be represented in (signed) binary. The occurrence of a 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 the case of signed binary representation, the third digit which is $\overline{1}$ will be processed by the subtracting operations. Algorithm 1 is an Adding-Subtracting Scalar Multiplication Algorithm, which is used to compute the elliptic curve scalar multiplication based for 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 ($d_i \in \{\overline{1},0,1\}$).

Algorithm 1: Adding-Subtracting Scalar Multiplication.

Data: Point on EC P, a string of signed bits $(d_0, \ldots, d_{n-1})$
Result: $Q = dP$
begin
  $Q \leftarrow 0, R \leftarrow P$
  for $i = 0$ **to** $n - 1$ **do**
    if $(d_i = 1)$ **then**
      $Q \leftarrow Q + R$
    else if $(d_i = \overline{1})$ **then**
      $Q \leftarrow Q$ - R
    end
    $R \leftarrow 2R$
  end
  return Q
end

The example below shows how to find the ECC scalar multiplication for a small scalar d.

Example 3: *Finding the ECC scalar multiplication for d = 115 .*
First, convert the integer d to the binary, so $d = (1001101)_2$
Then, find the ECC scalar multiplication based on scalar d from right to left as illustrated in Figure 2.

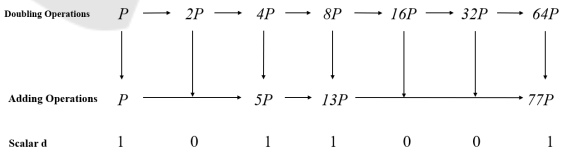

Figure 2: Finding ECC Scalar Multiplication.

3 RELATED WORK

Many researchers have been working to enhance the ECC by enhancing the calculation in the scalar multiplication. The improvement of the scalar multiplication can be achieved by improving or proposing some related algorithms in 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 adding operations in computing the scalar multiplication.

In 1951 Booth proposed a new scalar representation called signed binary representation. 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 Chang et al. (Chang et al., 2003) which is the result of using NAF, MOF, and JSF. Different researchers proposed methods to calculate the scalar multiplication in parallel computing using the binary or signed binary representation.

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 Scalar Multiplication Algorithm and Mutual Opposite Form (MOF). They used two processors to perform the parallel calculation, the Method calculates the doubling operation in a processor and adding operation in another processor at the same time. The proposed method computes the scalar multiplication without performing the MOF conversion. The proposed method is performing the comparison operation of the given bit-string d to decide where the second processor has to add or subtract the doubled point in case of non-zero bits $\{\bar{1}, 1\}$. The proposed method achieves the speed-up 90% faster than the sequential version of the ECC scalar multiplication with MOF.

Negre et al. (Negre and Robert, 2015) proposed a new parallel approach for finding the scalar multiplication. They split the scalar multiplication based on NAF into two parts for the prime field F_p and three parts for the binary field F_{2m}. In their method, both operations doubling and (addition or subtraction) will be performed in a separate thread. In the case of prime fields, the operations of scalar multiplication are split into two sections, based on representing d as $d = k_1 + 2^s k_2$. The first section $Q_1 = k_1 P$ will be performed in the first thread. The second part $Q_2 = 2^s k_2 P$ will be performed in the second thread. Finding the scalar multiplication in their proposed job given by $Q = Q_1 + Q_2$, the two points Q_1 and Q_2 are added to get the scalar multiplication Q. The proposed method achieved an improvement by at least 10% the computation time of the scalar multiplication.

Software implementation proposed by Robert (Robert, 2014) for finding ECC scalar multiplication. In their proposed method, they used two threads to perform the parallel calculation. As well as, for various elliptic curves over the prime F_p used four threads. Two algorithms are used in their job Double-and-add and Half-and-add algorithms. In this work, putting the doubling operations into one thread (producer) while additions and subtractions operation into another thread (consumer). One single mutex at the beginning of the computation is used to avoid using the mutex synchronization as much as possible. The goal of using the mutex is to keep the consumer in inactive state at the beginning of the processing while the producer processes the doubling operation. The method shows some violation of read-after-write dependency. The memory violation might happen because of 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 show that there is an error rate that is limited to less than 1% but is not acceptable. To eliminate this problem, a variable in a global memory as a loop counter is used. An extra operation is added to the scheme that will cause the reduction of the execution time in the parallel version. The NAF conversion is not a part of the parallel section. The result shows that the enhancement reached to 15% in comparison with the sequential version.

Phalakarn et al. (Phalakarn et al., 2018) proposed a new representation for right-to-left parallel elliptic curve scalar multiplication. The mathematical model reduced the calculation time for finding ECC scalar multiplication. Authors proposed algorithms that will generate the representations which will reduce the execution time of the scheme. Three processors are used to perform the whole calculation in the scheme. Two processors are for performing the doubling P and Q. The third processor is for performing the addition operation using two binary representations m and n. The issue of the communication between the processors in the model is still opened and may it cause an increasing time complexity because it is an extra operation.

Anagreh et al. (Anagreh et al., 2019) introduced an algorithm to find the ECC scalar multiplication based on NAF representation. They used two processors to perform the whole calculation in Parallel computing. The first processor performs the doubling operations while the second processor performs the NAF conversion and (addition or subtraction) operations at the same time. Shared memory is used to transmit the doubled points from the first processor to the second processor. They performed the NAF conversion by the second Processor before starting to calculate the addition or subtraction operations. This method eliminates the use of mutexes, as the consumption of doubled points by the second processor will not overtake their production by the first processor. The result shows an enhancement is 60% faster than the standard version of the ECC calculation based on NAF.

4 PARALLEL ALGORITHM

Reducing the execution time of the scalar multiplication by applying some efficient method is desired.

In this work, we propose and compare two parallel algorithms to calculate the scalar multiplication based on signed binary representations. We extract both algorithms by combining the Add-Subtract Scalar Multiplication Algorithm and Converting Methods for finding the signed binary representation. The converting methods from binary representation to signed binary representation are CRT and DRM respectively. The first algorithm based on circular buffers and the second is based on the delayed consumption of doubled points. The first algorithm optimizes the interprocessor communication costs, while the second algorithm optimizes the synchronization costs.

4.1 Algorithm based on Circular Buffers

In our first parallel algorithm, we use a circular buffer to transmit the processed data among the two processors in the scheme. The circular buffer is considered a shared memory. The processors can access the shared memory at any time to perform both operation read and write. Processor-1 can write the doubled point P and the scalar d_i in a specific location in the circular buffer. Processor-2 can read the doubled point P and the scalar d_i from the circular buffer to perform the addition or subtraction operations. Circular buffer has two pointers front and rear to organize the reading and the writing operations. In each iteration in the scheme, writing should be in a location pointed by a front pointer $Push_{front}$. The reading in the circular buffer should be in a location pointed by a rear pointer $Pull_{rear}$, where $front > rear$ for all writing and reading operations in the scheme. Such reading and writing operation is the most important issue to avoid any corruption in the calculation. As well as, we use two attributes for performing the reading and writing operations which are *is-full()* and *is-not-empty()*. The main goal of using the attributes is to check the situation of the circular buffer before performing the reading or the writing operations. In case the circular buffer is full, then keep cycling without performing any operation until there is an empty location in the circular buffer, then Processor-1 write the point and scalar in the empty location in the circular buffer. The second attribute will be used by Processor-2 before performing the addition or subtraction operations. The number of writing operations in the scheme that will be performed in the Processor-1 is based on the number of the bits n in the scalar d. Moreover, the number of reading operations that will be performed by the Processor-2 is based on the number of non-zero bits $\{\overline{1},1\}$ in the scalar d.

Task decomposition strategy is applied in our parallel implementation of the scalar multiplication Algorithm 2. We use two Processors to perform the multiplication. Processor-1 is responsible for performing three subtasks, see Processor-1 section in Algorithm 2.

Algorithm 2: Parallel Scalar Multiplication based on circular buffers and signed binary representations.

Data: Integer d, Point in EC P
Result: $Q = dP$, based on a signed binary representation

```
begin
    Processor 1 signed binary conversion, Doubling Operations
    begin
        R ← P
        REP = Convert_to_signed_binary(d)
        for i = 0 to n − 1 do
            repeat
            until ¬ buffer_is_full()
            if REP_i ≠ 0 then
                Push(R, REP_i)
            end
            R ← 2R
        end
        Push(0, 0)
    end
    Processor 2 Addition Operations
    begin
        Q ← 0
        repeat
            if buffer_is_not_empty() then
            Pull(R, d_i)
            if d_i = 1 then
                Q ← Q + R
            else
                Q ← Q - R
            end
        until d_i = 0
        return Q
    end
end
```

The first task, is the conversion of the scalar d to a signed binary representation with digits $\{\overline{1},0,1\}$, using one of the conversion algorithms discussed in Sec. 2.2. In our experiments, we have considered the CRT and DRM representations. **The second task** is calculating the doubling operations in the elliptic

curve based on the number of bits n in the scalar d, where the point in elliptic curve $P = (x,y)$ is given. Performing the doubling operation by calling the function n times, where n is the number of bits in the signed binary representation. Regardless, is it a $\overline{1}$, 0 or 1. **The last task** performed by Processor-1 is writing the doubled point R and the digit REP_i in an empty location in the circular buffer. As we explained above, the circular buffer is shared memory and both Processors can access the shared data for performing reading or writing operations. To indicate that no more points will be pushed into the buffer, Processor-1 will finish by pushing the pair $(0,0)$.

Processor-2 is responsible for performing three sub-task as well, see Processor-2 section in Algorithm 2. **The first task** is reading the doubled point R and the digit d_i from the circular buffer. Note, that each doubled point has a specific digit d_i, that will be stored together in the circular buffer to keep the sequence of the doubling operations $P, 2P, 4P, 8P, ..., 2^nP$. **The second task** is performing the addition or subtraction operations based on the non-zero bits of the scalar d. If the bit d_i is 1, Processor-2 has to perform the addition operation. If the bit d_i in the scalar d is $\overline{1}$, Processor-2 has to perform the subtraction operation which is **the third task** Processor-2 has to perform. Calculating the addition operation or/and subtracting operation will be saved in the accumulator Q which is the final result of finding EC scalar multiplication. The circular buffer is used to organize transmitting the data between two processors in the whole scheme. The data which has to transmit from Processor-1 to Processor-2 is located in the shared memory. Processor-1 writes in the circular buffer while Processor-2 reads the stored data from the circular buffer. Every time Processor-1 is going to write in the circular buffer, Processor-1 has to check that circular is not full and there is an empty location to the doubled point R and the scalar d_i.

In case the circular buffer is full, Processor-1 has to keep looping until there is an available location in the circular buffer. Processor-2 has to check every time that there is new data stored in the circular buffer by Processor-1. Then, Read the data and performing the addition or subtraction operation based on the scalar d.

4.2 Algorithm based on Delayed Consumption

Compared to Alg. 2, the proposed Algorithm 3 moves the task of doing the signed binary conversion of d from Processor-1 to Processor-2. Hence Processor-1 only computes the point doublings. These are stored in the array $R = (R_0, \ldots, R_{n-1})$, which has to be kept in the shared memory. All the points of R will be doubled regardless of where is the d_i is zeros or ones.

Processor-2 reads the elements of R and either adds or subtracts them from the accumulated value Q, according to the signed bit representation of the scalar d. Processor-2 will perform the signed binary conversion first while the Processor-1 performs the doubling operations and save the R_i in circular Buffer. Once Processor-2 finishes performing the conversion, Processor-2 will start reading R_i to perform the addition and subtraction operations.

Algorithm 3: Parallel Scalar Multiplication based on the delayed consumption of doubled points.

Data: Integer d, Point in EC P
Result: $Q = dP$, based on a signed binary representation

```
begin
    Processor 1 Doubling Operations
    begin
        R_0 ← P
        for i = 1 to n − 1 do
            R_i ← 2R_{i−1}
        end
    end
    Processor 2 signed binary conversion, Addition Operations
    begin
        Q ← 0
        (d_0, ..., d_{n−1}) = Convert_to_signed_binary(d)
        for i = 0 to n − 1 do
            if d_i = 1 then
                Q ← Q + R_i
            else
                Q ← Q - R_i
            end
        end
        return Q
    end
end
```

Again, in our experiments, we have considered both the CRT and DRM conversion methods in order to compute a signed binary representation of d.

5 EXPERIMENTAL EVALUATION

5.1 Algorithm based on Circular Buffers

We can summarize that the proposed method is extracting a new algorithm that combines two algorithms: Add-Subtract Scalar Multiplication, and a method to give a signed binary representation of the scalar. It performs the parallel computing on the extracted algorithm, given in Algorithm 2. We realized the algorithm with either the CRT or the DRM method in two versions of the code, Parallel and Sequential. The evaluation of the algorithm is based on the parallel and sequential versions for both the CRT and the DRM method.

As with almost all parallel applications, it is important to produce the best sequential code before starting to parallelize the code. Task decomposition strategy is used to divide the work into two Processors to perform the overall scheme to get the best result. Both sequential and parallel codes are written 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. As well as, we used a ttmath library under C++ to define a big integer number (bigger than or equal 1024-bits). It is important to note that we use an Intel Core i5 7th-Gen machine to test both versions (Parallel and Sequential) using Windows 10. We performed each key size 10 times and the average execution time is taken for all key sizes as shown in Figures 3 and 4.

In the implementation, we tested six different key sizes for both algorithms in both cases parallel and sequential: 160-bits, 192-bits, 224-bits, 256-bits,384-bits, and 521-bits. We generated a big integer number randomly for all key sizes we use in the implementation. Each number used in both parallel and sequential versions to determine the number of addition and subtraction operations.

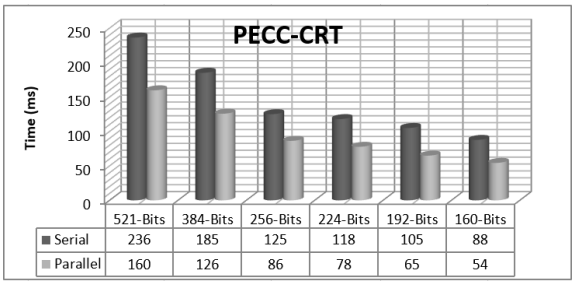

Figure 3: Execution time for Algorithm 2 using CRT.

The execution times for serial and parallel versions are taken as shown in the figures for the different key sizes of the ECC. In the case of the CRT encoding method, the differences between serial time and parallel time are a big difference in the case of key size 521-bits, 192-bits and 160-bits as shown in figure 3. The speed-up reaches 60% in comparison with the serial version of the same key size.

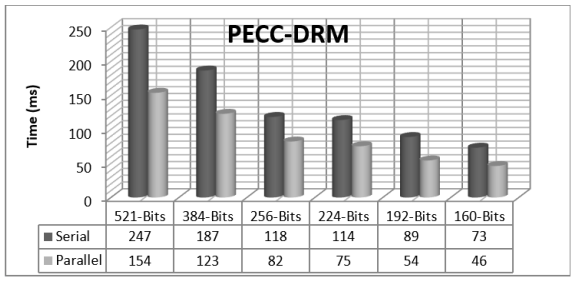

Figure 4: Execution time for Algorithm 2 using DRM.

In the case of the DRM encoding method, the difference between serial time and the execution time in the parallel version is significant in 192-bits and 160-bits key size. The speedup is 60% in comparison with the execution time of the serial version of the same key size.

The testing is according to a random key generated to perform the scalar multiplication. The same key is used to perform the calculation of the scalar multiplication in both version parallel and serial for each key size.

Key Size	Serial	Parallel	Speed -up	Efficiency
521-Bits	236	160	1.5	74%
384-Bits	185	126	1.5	74%
256-Bits	125	86	1.4	72%
224-Bits	118	78	1.5	76%
192-Bits	105	65	1.6	81%
160-Bits	88	54	1.6	81%

Figure 5: Speed up and Efficiency for CRT.

The number of non-zero bits in the key effect in the calculation of the ECC scalar multiplication. The occurrence of the bit 1 or/and $\bar{1}$ means performing the adding or/and subtraction operations by the Processor-2. The average number of the non-zero bits in the key is around 50% or less because of using the signed binary representation.

Key Size	Serial	Parallel	Speed -up	Efficiency
521-Bits	247	154	1.6	80%
384-Bits	187	123	1.5	76%
256-Bits	118	82	1.4	72%
224-Bits	114	75	1.5	76%
192-Bits	89	54	1.6	82%
160-Bits	73	46	1.6	79%

Figure 6: Speed up and Efficiency for DRM.

The execution time of one adding operation (or subtraction) is around two times and half of execution time of the doubling operations. Adding operation is much costly in comparison with doubling operation. Therefore, the occurrence of non-zero bits in the key even its around 50% doesn't mean that the Processor-1 will process the operation more than Processor-2. In this case, it is important to note, that one adding operation has a 5 sub-operations which are 2 squaring, 2 multiplications and 1 inversion, that make an adding operation is an expansive operation in comparison with doubling operation. Therefore, performing the whole calculation of the scalar multiplication by this method ensures some kind of balancing. We can see the efficiency of the whole calculation of the different key size is around 70% to 80%, see both figures 5 and 6.

5.2 Algorithm based on Delayed Consumption

In Alg. 2, Processor-1 has to find a signed binary representation of d and perform the doubling operations. Then, for non-zero bits in the signed binary representation, save the doubled points in the circular buffer. The doubled points that have been saved in the circular buffer will be readable by the Processor-2 to perform addition (or subtraction) operations. In Alg. 3, the finding of a signed binary representation is done by Processor-2. In this method, Processor-1 has to perform the doubling operations and save all doubled points in shared memory, no matter whether the corresponding bit in the signed binary representation is zero or non-zero. The number of writing operations in the shared memory is the same as the length of the scalar d. Processor-2 has to read all saved points from the shared memory. It also has to decide whether the doubled point should be added or subtracted, based on the CRT or DRM representation. Therefore, in case the bit is 1, it performs the addition operation, in case the bit is -1, it performs the subtraction operation, while in case the bit is 0, it drops the point and keeps reading. Figure 7, shows the benchmarking result of Alg. 3 for both implementation of the CRT and DRM. In general, the results show that the second algorithm is less efficient than the first, especially when using a small key size.

DRM is a low cost operation in comparison with CRT and another conversion method. In DRM, the time complexity of the conversion is less than the time complexity of conversion by applying the CRT. As well as, the number of non-zero bits in the signed binary converted by DRM is less than the signed binary converted by CRT and other standard methods. As

	CRT			DRM		
Key Size	**Serial**	**Parallel**	**Speed -up**	**Serial**	**Parallel**	**Speed -up**
521-Bits	509	312	1.6	502	317	1.6
384-Bits	422	282	1.5	422	272	1.6
256-Bits	257	203	1.3	240	194	1.2
224-Bits	240	202	1.2	239	171	1.4
192-Bits	198	164	1.2	192	150	1.3
160-Bits	142	124	1.2	114	89	1.3

Figure 7: Execution time for the second method.

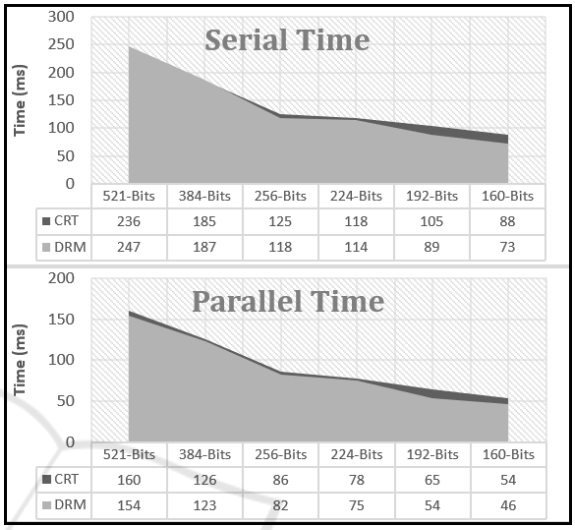

Figure 8: Execution time for CRT and DRM.

mentioned above in example 2. The hamming weight of DRM representation is 2, which is less than the hamming weight of CRT representation. Less hamming weight will save the calculation time of finding ECC scalar multiplication in comparison of using another representation. In figure 5, we can recognize the difference in the execution time of both DRM and CRT for both serial and Parallel version. The calculation of the ECC scalar multiplication using DRM Representation is faster than using CRT representation. Overall Key sizes and in both serial and parallel versions, finding scalar multiplication based on DRM representation is faster than finding the scalar multiplication based on CRT.

6 CONCLUSION

In this work, we proposed two algorithms to calculate the ECC scalar multiplication based on CRT and DRM representation. The first algorithm based on CRT representation and the second algorithm based on DRM representation. We proposed a parallel algorithm to perform both calculations of the two proposed algorithms separately using two processors. The results show speed-up reach to 60% in compari-

son with a serial version for both algorithms. As well as, we introduced the difference in execution time for both DRM and CRT. Future work includes using three threads to perform the calculation in case the number of non-zero bits in the key is more than usual, which will make the calculation of adding point more costly – the third thread will help to reduce the execution time in this case.

REFERENCES

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

Anagreh, M., Vainikko, E., and Laud, P. (2019). Accelerate performance for elliptic curve scalar multiplication based on naf by parallel computing. In *ICISSP 2019 - 5th International Conference on Information System Security and Privacy*. SITEPRESS.

Asif, S. and Kong, Y. (2017). Highly parallel modular multiplier for elliptic curve cryptography in residue number system. *In Circuits, Systems, and Signal Processing*, 26(6).

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

Balasubramaniam, P. and Kathikeyan, E. (2007). Elliptic curve scalar multiplication algorithm using complementary recoding. *In Applied mathematics and computation*, 1(190).

Booth, A. (1951). A signed binary multiplication technique. *In Journal of Applied Mathematics*, 4.

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

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

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

HK, P. and Sanghi, M. (2010). Speeding up computation of scalar multiplication in elliptic curve cryptosystem. *In International Journal on Computer Science and Engineering*, 4(2).

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

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

Miller, V. (1986). Use of elliptic curves in cryptography. In *In Conference on the theory and application of cryptographic techniques*, number 108 in LNCS, pages 417–426, Berlin, Heidelberg. Springer.

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

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

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

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

Robert, J.-M. (2014). Parallelized software implementation of elliptic curve scalar multiplication. In *In International Conference on Information Security and Cryptology*, pages 445–262. Springer.

Solinas, J. (2001). Low-weight binary representations for pairs of integers. In technical report corr 2001-41, Center for Applied Cryptographic Research, University of Waterloo, Canada.