Parallel Method for Computing Elliptic Curve Scalar Multiplication Based on MOF

Mohammad Anagreh, Azman Samsudin and Mohd Adib Omar
School of Computer Sciences, Universiti Sains Malaysia, Malaysia

Abstract: *This paper focuses on optimizing the Elliptic Curve Cryptography (ECC) scalar multiplication by optimizing one of the ECC calculations, which is based on the Mutual Opposite Form (MOF) algorithm. A new algorithm is introduced that combines the add-subtract scalar multiplication algorithm with the MOF representation to speed-up the ECC scalar multiplication. The implementation of the algorithm produces an efficient parallel method that improves the computation time. The proposed method is an efficient ECC scalar multiplication that achieves a 90% speed-up compared to the existing methods.*

Keywords: *ECC, MOF, circular buffer, parallel computing.*

Received March 6, 2012; accepted April 27, 2013; published online March 13, 2014

1. Introduction

Known as one of the oldest sciences, cryptography is the science of using mathematical rules to encrypt and decrypt information. Many inventive cryptography algorithms have been proposed and implemented over the years. Among them are the asymmetric cipher algorithms that are based on number theory and elliptic curve.

Two researchers, Koblitz and Miller [8, 9] independently introduce Elliptic Curve Cryptography (ECC). ECC is one of the public key cryptography methods, which depends on the Elliptic Curve Discrete Logarithm Problem (ECDLP) of the elliptic curve. ECC is commonly used in secure applications to achieve the highest security level with shorter key size. In fact, the ECC with 160 bit key length is as secure as 1024 bit key length for RSA [14]. Shorter key size and yet strong is desired for portable devices such as smart card, footprint hardware, RFID, and others.

Many researchers have devoted their efforts in ECC [1, 2, 3], although the scalar multiplication in ECC is an expensive operation. Scalar multiplication (dP) is known as an expansive operation in ECC, where d is an integer converted to the binary representation, while p is a point on the elliptic curve. Scalar multiplication includes two main operations: Adding and doubling points on the elliptic curve as shown in Figures 1 and 2.

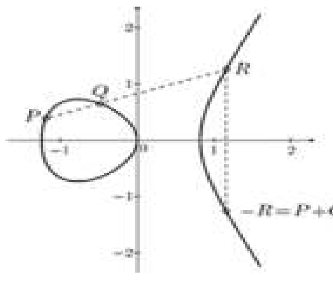

Figure 1. Adding operation on elliptic curve.

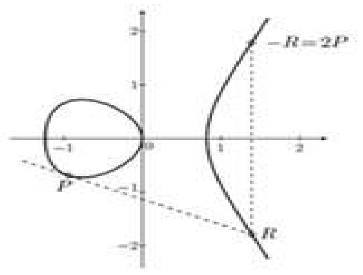

Figure 2. Doubling operation on elliptic curve.

The numbers of these operations are based on the length of scalar d, as well as the Hamming weight of d [13]. The doubling operation is based on the number of digits in scalar d. On the other hand, adding operation is based only on the number of non-zero bits of scalar d. Our work focuses on the optimization of the ECC scalar multiplication for case of the standard curve over the prime field F_P. Several researchers have been working on enhancing the performance of the scalar multiplication by using signed binary representation [4, 6, 9, 11, 12]. One of the most common methods to compute the exponentiation of random elements in an abelian group is by applying the Mutual Opposite Form (MOF) representation and sliding window schemes for the w on MOF (wMOF) [11]. There have been several methods proposed to speed-up the scalar multiplication through parallelization [2, 3, 5, 7]. For parallelization, two common methods are used: Data decomposition and task decomposition. In this paper, scalar multiplication performance is improved by parallelizing the related ECC algorithm. Parallelizing the conversion algorithm (from binary to MOF representation) and scalar multiplication algorithm has been identified in as one of the ways to improve the overall ECC performance. This paper is organized as follows: Section 1 briefly introduces ECC and scalar multiplication algorithms. Section 2 briefly reviews ECC and scalar multiplication algorithms. Section 3 reviews the related work. Section 4 describes the proposed method. Section 5 presents result and discussion. The last section concludes the proposed method.

2. Elliptic Curve Cryptosystem Overview

Two main finite fields are used in ECC: Binary curves over *GF* (2^m), and prime curves over F_P [9]. In our work, the focus is on the prime curves over F_P.

2.1. Binary Filed Over GF (2^m)

Elliptic curve over a finite field *GF* (2^m), consists of 2^m elements. Both multiplication and addition operations are defined over polynomials of elliptic curve [9]. The proposed method uses a cubic equation as variables such as *x* and *y* and coefficients *a* and *b* [9, 10]. The form of cubic equation that is used for ECC is as follows:

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

2.2. Prime Filed Over F_P

The prime curves over F_P make use of the cubic equation as identified in Equation 1 with Cartesian coordinate variables (*x, y*) and coefficients (*a, b*) as elements of F_P. All the values are integers and calculated by performing modulo *p* [14]. The cubic equation with coefficient (*a, b*) and variables (*x, y*) for the elliptic curves over F_P are as follows:

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

Now, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be in the elliptic curve set of $E_P(a, b)$. In addition, let *O* be the infinity point. The rules for adding operation in elliptic curve is as follows:

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

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

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

- If $Q \neq P$, then $R = Q + P$, where $R = (x_3, y_3)$ is defined as follows:

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

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

$$\lambda = \left\{ \begin{array}{l} \left(\dfrac{y2 - y1}{x2 - x1}\right) mod \; p, \quad if \;\; p \neq Q \\ \left(\dfrac{3x^2 + a}{2y1}\right) mod \; p, \quad if \;\; p \neq Q \end{array} \right\} \tag{7}$$

3. Related Work

3.1. Mutual Opposite Form Representation

The MOF was proposed by Okeya *et al.* [11] to convert the binary representation to signed binary representation, where it can be computed in any order (left-to-right and right-to-left). MOF obtained the first left-to-right signed exponent recoding scheme by applying a sliding window conversion on MOF.

Each integer has a unique scalar in the MOF representation, which is at most one bit longer than its binary representation. The MOF representation of the binary string *d* is obtained by copying the binary string, shifting *d* one bit to the right, followed by bitwise subtraction.

Many algorithms have been designed to convert binary to signed binary representation in order to reduce the Hamming weight, in which MOF representation is one of them. MOF representation satisfies the following properties:

1. MOF can implement sliding window method with width *w*, but some other existing methods [4, 6, 13] cannot be implemented with the sliding window scheme.
2. Less storage is required compared to other existing algorithms [11].
3. MOF method can be computed in both directions: left-to-right and right-to-left.
4. MOF eases the conversion from binary to signed binary representation.

3.1.1. Right-to-Left MOF

Algorithm 1 distinguishes the Right-to-Left MOF representation from the binary representation. The MOF representation has one bit longer than its binary counterpart.

Algorithm 1: Right-to-Left Binary to MOF

```
Input: A non-zero n-bit binary string $d = d_{n-1}, ..., d_1, d_0$.
Output: MOF $md_n, ..., md_1, d_0$ of d.
1. $md_0 \leftarrow -d_0$
2. For i = 1 to n-1 do
3.     $md_i \leftarrow d_{i-1} - d_i$
4. End for
5. $md_n \leftarrow d_{n-1}$
6. Return ($md_n, ..., md_1, d_0$)
```

3.1.2. Left-to-Right MOF

Algorithm 2 shows the Left-to-Right MOF representation converted from the binary representation. Similar to the representative found in Algorithm 1, the MOF representation in Algorithm 2 has one bit longer than the binary counterpart.

Algorithm 2: Left-to Right Binary to MOF

```
Input: A non-zero n-bit binary string $d = d_{n-1}, ..., d_1, d_0$.
Output: MOF $md_n, ..., md_1, d_0$ of d.
1. $md_n \leftarrow -d_{n-1}$
2. For i = n-1 to 1 do
3.     $md_i \leftarrow d_{i-1} - d_i$
4. End for
5. $md_0 \leftarrow -d_0$
6. Return ($md_n, ..., md_1, d_0$)
```

3.2. Addition-Subtraction Scalar Multiplication (ASSM)

The scalar multiplication is an operation of adding a point *P* to itself *d* times and denoted as *Q=dP*. Algorithm 3 is used to compute the scalar multiplication based on the binary representation. The occurrence of bit `1' in the scalar *d* will be processed as the addition operation, which approximately asymptotic to $\frac{n}{2}$, while the number of doubling operations in the algorithm is *n-1*, where *n* is the length of *d* in bits.

Algorithm 3: Binary Scalar Multiplication algorithm

```
Input: A non-zero n-bit binary string $d = \sum_{i=0}^{n-1} 2d^i \in \{0,1\}$
Output: MOF $Q = dP$
1. $Q = 0, R = P$
2.  For $i = n\text{-}1$ to 1 do
3.        $R=2R$
4.      If ($d_i$==1) then $Q = Q+R$
5.  End for
6. Return Q based on MOF
```

Given that the MOF representation is one of the signed binary representations, we can compute the scalar multiplication by applying Algorithm 4.

Algorithm 4: The Add-Subtract Scalar Multiplication algorithm

```
Input: A nonzero n-bit binary string $d = \sum_{i=0}^{n-1} 2d^i \in \{-1,0,1\}$ .
Output: MOF $Q = dP$.
1.  $Q = 0, R = P$
2.  For $i = n\text{-}1$ to 1 do
3.      $R = 2R$
4.      If ($d_i$==1 ) then $Q = Q+R$
5.      If ($d_i$==−1) then $Q = Q−R$
6.  End for
7.  Return Q based on MOF
```

4. Proposed Method

In our work, we propose a combination ASSM-MOF of the MOF representation algorithm with the add- subtract scalar multiplication algorithm to constitute one algorithm for calculating the scalar multiplication based on the MOF representation without any conversion from binary to signed binary representation.

4.1. Combination Mutual Opposite Form and Scalar Multiplication

First, we propose a new serial multiplication algorithm, ASSM-MOF, to calculate scalar multiplication based on the MOF representation. The algorithm combines the ASSM algorithm with the MOF algorithm. Algorithms 2 and 4 show two iteration loops: Each loop for each algorithm. When the integer *d* is generated, it should be converted to the MOF representation by applying the MOF algorithm. Then, we use the scalar multiplication algorithm to compute *dP*, where *d* is a MOF representation and *P* is a point on the elliptic curve. We calculate the scalar multiplication directly prior to comparing the scalar *2d* and *d*. Two possibilities for the binary scalar, that is either bit `0' or bit `1'. If the scalar $2d_i$ is `0' and $d_i$ is `1', we calculate both doubling (ECCDBL) and subtracting (ECCSUB), because $2d_i\theta d_i$ is already `1', where `θ' is a bitwise subtraction. If $2d_i$ is `1' and $d_i$ is `1', we calculate doubling (ECCDBL) and adding (ECCADD), because $2d_i\theta d_i$ is already `1'. Other cases, we calculate only the doubling (ECCDBL) for all *n*-bits, because $2d_i\theta d_i$ is already `0'. Algorithm 5 shows the proposed combination algorithm (ASSM-MOF) of the MOF algorithm and scalar multiplication algorithm.

Algorithm 5: Combination algorithm (ASSM-MOF)

```
Input: A non-zero n-bit binary string $d = \sum_{i=0}^{n-1} 2d^i \in \{0,1\}$
Output: MOF $Q = dP$ based on MOF representation.
1. $Q = 0, R = P$
2.  For $i = n\text{-}1$ to 1 do
3.        If ($2d_i$ == 1 & $d_i$ == 0) then $Q = Q+R$
4.        If ($2d_i$ == 0 & $d_i$ = = 1 )then $Q = Q−R$
5.        $R = 2R$
6.  End for
7. Return Q based on MOF
```

4.2. Parallel Scalar Multiplication Based on MOF

Second, parallelize the proposed method by using two processors. The proposed method for calculating the *dP* based on the MOF uses two processors, one for the execution of doubling operation, ECCDBL-Processor, and the other for adding operation, ECCADD-Processor. The adding operation uses a point from doubling operations. The algorithm works based on the task decomposition principal which means two processors work together to calculate *dP*. It is important to note that there is dependency in both main basic operations (doubling and the other side adding and subtracting) which means the execution of the adding operation is based on data that comes from the doubling operation. However, we need to achieve data independent requirement before decomposing the task into two independent sub-tasks. To achieve this, a circular buffer is used to transmit data between processors. The proposed algorithm is divided into two sections. The two main sections should be executed at the same time. The doubling operation is executed in the ECCDBL-Processor while adding and subtracting operations are executed in the ECCADD-Processor. It is important to note that each processor has a single thread. The circular buffer is the communicating channel between the two processors used to transmit the doubling point to the other processor as shown in Figure 3.

We also have to make sure the MOF scalar is not zero before calling the ECCADD-Processor. Notably, we make sure the maximum size of circular buffer is enough to accommodate doubling points before the computations begin. The ECCDBL-Processor reads the given point P and then begins performing the doubling operations. The ECCDBL-Processor fills up the buffer with d_iP when the d_i is a non-zero bit. The ECCADD-Processor reads the value d_iP and calculates the addition or subtraction. According to this method, both processors are working simultaneously at any given time. Another thing to note is that, when the ECCADD-Processor begins consuming the point from the buffer, we should check whether the buffer is empty or not. It should be noted as well, that the index of the push operation to the circular buffer should be more than the index of the pop operation to avoid data starvation problems. The parallel code used section directives to write the parts for each processor. The parallel code consists of two main parts (ECCDBL-Processor and ECCADD-Processor) as described below:

```
# pragma omp sections
{
    # pragma omp section //For ECCDBL-Processor
  {
        Calculate the doubling operations.
        Fill circular buffer when the MOF is non-zero bits.
  }
    # pragma omp section //For ECCADD-Processor
  {
        Check the circular buffer.
        If the buffer is not empty, read the value.
        To check the MOF bit representation:
          - If the MOF bit is '1', calculate the adding
            operations.
          - If the MOF bit is '‾1', calculate the subtracting
            operations.
  }
}
```

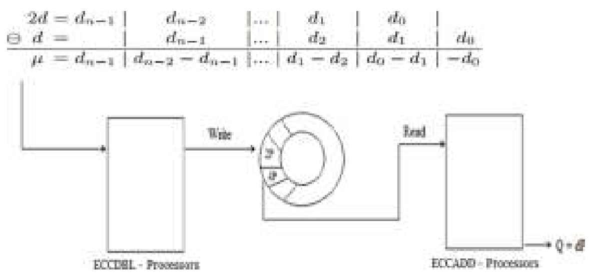
$2d = d_{n-1}$ | d_{n-2} | … | d_1 | d_0 |
$\ominus\ d =$ | d_{n-1} | … | d_2 | d_1 | d_0
$\mu = d_{n-1}$ | $d_{n-2} - d_{n-1}$ | … | $d_1 - d_2$ | $d_0 - d_1$ | $-d_0$
Write
Read
$2P$
P
$Q = dP$
ECCDBL - Processors
ECCADD - Processors

Figure 3. Parallel ASSM-MOF using circular buffer.

5. Result and Discussion

The performance of the propose algorithm (ASSM-MOF) is measured according to the data size as well as the occurrence of the non-zero bits in the MOF representation of the scalar d. The performance depends on the ratio of *N=Num of ECCADD/Num of ECCDBL*.

We assume that every bit of the MOF representation is non-zero bit; thus, the number of adding and doubling operations are the same. It means the performance of the proposed algorithm is good because the two processors are busy. Let further assume that the most bits of the MOF representation are zeros. Under this assumption, the execution time of the proposed method is similar to the standard method. This is because the idle time of the ECCADD-Processor is very big. However, it is important to note that the Hamming weight of the MOF representation is hardly to be zero.

For ASSM-MOF algorithm, it is important to produce the optimal serial code so that the best serial performance for overall ECC application is achieved. We use Visual C++ .Net in the implementation to write the serial code of the combination algorithm (ASSM-MOF). We use the Open MP Library that is supported in Visual C++. Package to parallelize serial code. It is also important to note that we use the Visual Studio.Net 2008 environment to facilitate the dual-processor implementation using Microsoft Windows 7 Professional Edition. All codes are tested on the same machine.

We use the Intel Pentium machine to implement the algorithm. The specification of the Intel Pentium is: Dual-Core T4500 Processor (2.30 GHz, 1 MB L2 cache). The total operations are performed 10 times and the average execution time is taken as a result for both sequential and parallel codes. The execution times for both codes are represented in millisecond in Table 1.

Table 1. Result for the proposed parallel method against the standard method.

Key Size (bits)	N	Serial (ms)	Parallel (ms)	Speed-up	Efficiency
160	1	0.0240	0.0126	1.90	95%
	1/2	0.0135	0.0082	1.64	82%
192	1	0.0259	0.0142	1.83	91%
	1/2	0.0185	0.0115	1.61	80%
224	1	0.0311	0.0168	1.85	92%
	1/2	0.0223	0.0138	1.61	80%
256	1	0.0421	0.0220	1.91	95%
	1/2	0.0244	0.0149	1.64	82%
384	1	0.0880	0.0470	1.87	94%
	1/2	0.0613	0.0434	1.41	71%
512	1	0.2256	0.1269	1.78	89%
	1/2	0.1686	0.1155	1.46	73%

The result shows the difference between serial and parallel implementation of the algorithm for various data size: 160 bit, 192 bit, 224 bit, 256 bit, 384 bit and 512 bit. The execution time of the proposed method is better than existing method, which reached 90% speed-up for 160 bit data size. The speed-up and efficiency also shows that the performance for various data sizes are different according to the idle time when the number of adding operation less than number of doubling operation (when $N = \frac{1}{2}$).

The domain of the speed-up is between 1.41 to 1.9 times. According to the result in Table 1 we can note that the best case is when the key size is short (160 bit) and the worst case when the key size is long (512 bit). This is the benefit of our method because the best case of the result fit with nature of ECC, which has a shorter key size compared with other algorithms such as RSA.

6. Conclusions

In this work, an efficient parallel method has been designed and implemented by parallelizing the combination of the add-subtract ECC scalar multiplication and the MOF algorithm. The result shows that the parallelized combination algorithm is an efficient method where the execution time has been reduced. All efforts in this work lie mainly on optimizing the ECC scalar multiplication based on the MOF algorithm on standard curves over prime filed. The future works could then focus on parallelizing the sliding window scheme for general width wMOF. This can be applied to parallelize the signed binary representation methods or any related algorithm with the ECC scalar multiplication.

References

[1] Al-Daoud E., "An Improved Implementation of Elliptic Curve Digital Signature by using Sparse Elements," *the International Arab Journal of Information Technology*, vol. 1, no. 2, pp. 203-208, 2004.

[2] Al-Somani T. and Ibrahim M., "Generic-Point Parallel Scalar Multiplication without Precomputation," *IEICE Electronics Express*, vol. 6, no. 24, pp. 1732-1736, 2009.

[3] Ansari B. and Wu H., "Parallel Scalar Multiplication for Elliptic Curve Cryptosystems," *in Proceedings of International Conference on Communications, Circuits and Systems*, Ontario, Canada, vol. 1, pp. 71-73, 2005.

[4] Balasubramaniam P. and Karthikeyan E., "Fast Simultaneous Scalar Multiplication," *Applied Mathematics and Computation*, vol. 192, no. 2, pp. 399-404, 2007.

[5] Gutub A., "Remodeling of Elliptic Curve Cryptography Scalar Multiplication Architecture Using Parallel Jacobian Coordinate System," *International Journal of Computer Science and Security*, vol. 4, no. 4, pp. 409-425, 2010.

[6] Huang X., Shah P., and Sharma D., "Minimizing Hamming Weight Based on l's Complement of Binary Numbers Over GF(2^m)," *in Proceedings of the 12th International Conference on Advanced Communication Technology*, Piscataway, USA, pp. 1226-1230, 2010.

[7] Izu T. and Takagi T., "A Fast Parallel Elliptic Curve Multiplication Resistant Against Side Channel Attacks," *in Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, Paris, France, vol. 2274, pp. 280-296, 2002.

[8] Koblitz N., "Elliptic Curve Cryptosystem," *Mathematics of Computation*, vol. 48, no. 173, pp. 203-209, 1987.

[9] Miller V., "Use of Elliptic Curves in Cryptography Advances," *in Proceedings of Cryptology*, *Lecture Notes in Computer Science*, Berlin, Germany, vol. 218, pp. 417-426, 1986.

[10] Nicolas R., *ICSA Guide to Cryptography*, McGraw Hill, New York, USA, 1999.

[11] Okeya K., Samoa K., Spahn C., and Takagi T, "Signed Binary Representations Revisited," *in Proceeding of Annual International Cryptology Conference Advances in Cryptology Crypto*, Santa Barbara, USA, pp. 123-139, 2004.

[12] Pathak H. and Sanghi M., "Speeding-up Computation of Scalar Multiplication in Elliptic Curve Cryptosystem," *International Journal on Computer Science and Engineering*, vol. 2, no. 4, pp. 1024-1028, 2010.

[13] Purohi G. and Rawat A., "Efficient Implementation of Arithmetic Operations in ECC Over Binary Field," *International Journal of Computer Applications*, vol. 6, no. 2, pp. 5-9, 2010.

[14] Stalling M., *Cryptography and Network Security*, Prentice Hill, USA, 2000.

Mohammad Anagreh received his BSc degree in computer science from Irbid National University, Jordan. In 2012, he received his MSc in computer sciences from Universiti Sains Malysia, Malaysia. His current research interests are parallel computing and information security.

Azman Samsudin is an associate professor at the School of Computer Sciences, Universiti Sains Malaysia. He earned his BSc in computer science from University of Rochester, USA, in 1989. Later, he received his MSc in computer science in 1993 and his PhD in computer science in 1998, both from University of Denver, USA. His research interests include cryptography, switching networks and distributed computing.

Mohd Adib Omar completed his BSc artificial intelligence and MSc computer networks in computer science from American University, Washington DC, USA in 1996 and 1997 respectively. He received his PhD in Collaborative Computing from University Sains Malaysia, in 2009. Currently, he is a senior lecturer at School of Computer Sciences, USM. His research interests include wireless networks, collaborative and service computing, distributed and parallel computing, and information security.