Report: Accelerate Performance for Elliptic Curve Scalar Multiplication Based on NAF by Parallel Computing

Student's Name : Mohammad Anagreh
Supervisor: Prof. Eero Vainikko
Distributed System Seminar
University of Tartu
Tartu-Estonia
9-May-2018

***Abstract*—This is the scientific report for the several papers in Elliptic Curve Cryptosystem. The aim of Elliptic Curve Cryptosystem (ECC) is that same security level with RSA but shorter key size. The basic operation in the ECC is scalar multiplication which is an expansive operation. In this report, 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 accelerated the calculation to 50 % compared with the standard method.**

I. Introduction

Elliptic curve cryptosystem (ECC) is a powerful cryptosystem to encrypt data because it has a high-level security but with shorter key size compared with other existing algorithms such as RSA [1]. For example, the keys elliptic curve cryptosystem request 160-bits, compared to 1024-bits of RSA [2]. One of the major application fields of elliptic curve cryptosystem is RFID and smart card. Maybe they are strong this is because of the limitations concerning the key length, shorter key length also makes ECC better to be used on portable devices. Different reasons that enable it to be used widely. Several researchers have been trying to enhance the performance of elliptic curve cryptosystem (ECC), the most important operations in the ECC is the scalar multiplication based on time-consuming. Therefore, many researchers focused to enhance and improve this area.

Elliptic curve cryptosystem *(ECC)*, independently produced by two researchers, Koblitz [3] and Miller [4], The number of adding and doubling operations in elliptic curve are based on length of scalar representation (d), which is an integer number that will be converted to the binary representation $d_i \in \{0,1\}$, $i = (0,1,2,,n-1)$ or to the singed binary representation $d_i \in$ {-1,0,1}, $i = (0,1,2,,n-1)$. Several ECC researchers have been working to accelerate the performance of the EC scalar multiplication by introducing new conversion algorithms to have the hamming weight. Hamming Weight *(HM)* is the number of non-zero bits in scalar representation d. The reducing the bits '1' in the scalar representation will reduce the adding operations in ECC scalar multiplication. Therefore, lower HM is preferred.

Several researchers have proposed new methods to convert the binary representation to the singed binary representation to reduce the hamming weight of scalar d. These methods are Mutual Opposite Form *(MOF)* [5], Joint Sparse Form *(JSF)* [6], Non Adjacent Form *(NAF)* [7] and others [8][9]. As well as, there are several methods proposed to reduce calculation time of the ECC scalar multiplication (Speed-up) by parallel computing [10] [11] [12]. In this paper, accelerate performance of the ECC Scalar multiplication are proposed by parallelizing scalar multiplication algorithm (Add-subtract scalar multiplication algorithm) and transforming algorithm from binary representation to the singed binary representation which is non-adjacent form *(NAF)* algorithm. Scalar multiplication performance can be improved by parallelizing the NAF algorithm.

This paper is organized as follows: Section 1 briefly introduces the ECC and scalar multiplication algorithms. Section 2 briefly reviews the ECC and scalar multiplication algorithms. Section 3 presents the proposed method. Section 4 shows the results and discussion. The last section concludes the proposed method and future works.

II. Elliptic Curve Cryptosystem Overview

Mathematically, in elliptic curve there are two main finite fields which are the binary curves over GF(2m), and the prime curves over FP as following:

A. Binary Field Over $GF(2m)$

Elliptic curve over a finite field $GF(2m)$, consists of $2m$ elements. Both multiplication and addition operations are defined over polynomials of elliptic curve. The proposed method uses a cubic equation as variables such as x and y and the

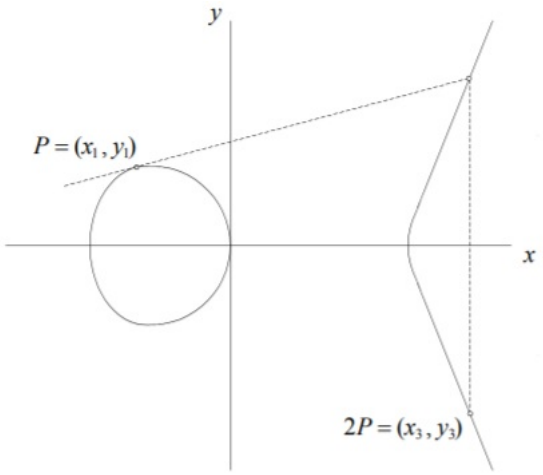

Fig. 1. Doubling Operations

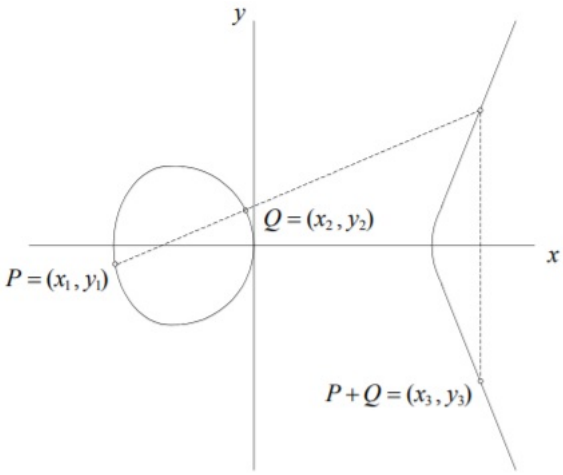

Fig. 2. Adding Operations

coefficients a and b [13]. The form of cubic equation that is used in the ECC is as follows:

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

B. Prime Field Over F_P

In this paper, the focusing is on the prime curves 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 $P = (x_1, x_2)$ and $Q = (x_2, y_2)$ be in the elliptic curve set of F_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

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

If QP, then, R = Q + P, where R = (x_3, y_3) is defined as follows:

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

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

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

III. Preliminaries

A. Non-Adjacent Form (NAF)

Proposed by Reitwiesner [15] in 1960. In Non-Adjacent Form *(NAF)*, we can re-present the binary string by adding the third digit to the bit-string which is '-1'. NAF algorithm is a same value with binary representation but the difference appears in representation itself. The goal of using such algorithms in the ECC is to reduce the number of Hamming Weight which is the number of non-zero bits in the key. Therefore, reducing the hamming weight will reduce the number of adding operations in scalar multiplication. So, finding the scalar multiplication based on the signed binary representation will the reduce the total execution time in elliptic curve. Algorithm 1 is a non-adjacent form algorithm, the input should be integer number and the out put is the signed binary representation that will be used in finding the scalar multiplication of the ECC.

Data: Integer Number C
Result: *(NAF)* Representation
$C = K$, $j = 0$
while $C > 0$ **do**
  if *C is odd* **then**
    $S[j]$ = 2 - (C mod 4)
    C = C - $S[j]$
  end
  else
    $S[j] = 0$
  end
  C = C/2
  j = j + 1
end

Algorithm 1: NAF Representation Algorithm

B. ECC Scalar Multiplication

The scalar multiplication is a main operation in the ECC. Scalar multiplication has two main operations which are

adding a point P to itself in an elliptic curve d times given by $Q = dP$ and doubling operations. An integer number has to be converted to bit-string as a scalar d. The occurrence of bit '1' in scalar d will be computed the adding operation (see Figure 2), which approximately asymptotic to $n/2$, while the number of doubling operations are $n-1$, see Figure 1. In case of signed binary representation, the third digit which is '-1' will be processed the subtracting operations.

Data: Point on EC P, a non-zero n-bit binary string d_n
Result: Q based on $\sum_{i=0}^{n-1} d_i = \{-1, 0, 1\}$
$Q = 0$, $R = P$
for i = $n-1$ *to* 0 **do**
  R = $2R$
  if *(d_i == 1)* **then**
    Q = Q + R
  else if *(d_i == -1)* **then**
    Q = Q - R
end

Algorithm 2: Adding-Subtracting SM Algorithm

Algorithm 2 is An Adding-Subtracting Scalar multiplication algorithm, which is used to compute the elliptic curve scalar multiplication based on bit-string d, regarding where is the bit-string is a binary representation or signed binary representation.

C. Related Work

Many research have been working to enhance the ECC by enhancing the calculation in the scalar multiplication. The improvement of the scalar multiplication can be by improving or proposing some related algorithms in scalar multiplication. The signed binary representation algorithms is an efficient way to reduce the number of non-zero bits in the key. Hamming weight is big player to reduce the number of adding operations in computing scalar multiplication.

In 1951 was proposed a new scalar representation called signed binary representations by Booth, there are many methods to represent 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 [16] which the result it is nearly for NAF, MOF and JSF.

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

Ansari et al [17], proposed a parallel method based on task decomposition, to parallelize ECC scalar multiplication. The binary representation and double-and-add algorithm is used to compute the ECC scalar multiplication.

Anagreh et al [18], proposed a new parallel method to compute scalar multiplication based on the mutual opposite form *(MOF)*. They extract a new algorithm that combined mutual opposite form *(MOF)* and Adding-Subtracting Algorithm. The Method calculates the doubling operation and adding operation at the same time without performing the MOF conversion. The proposed method is performing the comparison operation of the bit-string to decide where the second processor has to add the doubled point in case of non-zero bits. The acceleration is reached to around 90 compared with the standard case.

IV. Proposed Work

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

In this work, we propose a parallel method that calculates the ECC scalar multiplication based on signed binary representation algorithms which is the NAF. Finding the scalar multiplication by applying the add-subtract scalar multiplication algorithm algorithm. The proposed method is combined add-subtract scalar multiplication algorithm with non-adjacent form algorithm. we extracted a new algorithm to perform proposed scheme, see Algorithm 3.

Our parallel method strategy 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 to parts. Part 1 will be processed via Processor-1 which is finding the doubling operations keysize-time, then saving that doubled points in the shared array. We can recognize that there is no relationship between the kind of bits $\{1, 0, -1\}$ and doubling operations. By another way, Processor-1 has to perform doubling operations according to the length of the key $n-1$, regarding the bit is '1', '0' or '-1'. In this case, if the key size is 160-bits, processor-1 has to perform the doubling operations 160 times. As well as, processor-1 has to save the doubled point in the shared array Arr, see Algorithm 3. The size of the shared array Arr is the same size of the key n.

Part 2 of the algorithm, is to find both NAF representation by applying NAF algorithm and perform the adding points according to the number of hamming weight (the number of none-zero bits in the key). Processor-2 has to find the NAF representation before begin performing the adding operations. According to the kind of the bits $\{1,\ 0,\ -1\}$ in NAF representation, processor-2 has to perform the adding or subtracting operations. In case the bit is '1', performing the adding operation and save the result in accumulator Q, while if the bits is '-1' performing the subtracting operations and save the result in the Q as well. Remember, in case the bit is '0', processor-2 has to abort the point. performing adding or subtracting operations by reading the saved points $(2P, 4P, ..., n)$ from the shared memory Arr, see Figure 3. Both processors have ability to reach to the shared array Arr which is located in the shared memory. Processor-1 perform

Data: Integer Number k, Point in EC P
Result: Q based on *(NAF)*
begin
  Processor 1 $\longrightarrow$ Doubling Operations (DBL)
  $R = P$
  for $i = n - 1$ *to* 0 **do**
    $R = 2R$
    Arr_i = R
  end
  Processor 2 $\longrightarrow$ NAF + Adding Operations (ADD)
  $C = K$, $j = 0$
1 **while** $C > 0$ **do**
    if *C is odd* **then**
      $S[j]$ = 2 - (C mod 4)
      C = C - $S[j]$
    end
    else
      $S[j] = 0$
    end
    C = C / 2, j = j + 1
  end
  for y = n-*1 to* 0 **do**
    $T = Arr_y$
    m = $S[y]$
    if m = *1* **then**
      Q = Q + T
    end
    else if m = *-1* **then**
      Q = Q - T
  end
  Return Q
end
Algorithm 3: Parallel Scalar Multiplication based on NAF

the writing in the array Arr, while processor-2 perform the reading in the array. We should manage the reaching to the shared array carefully. Processor-1 should be able to write only, while Processor-2 should be able to read. As well as, Processor-2 should be able to read from the locations which have doubled points already $is\text{-}not\text{-}empty()$. In that case, we should keep Processor-1 starts writing the doubled points in the shared array Arr, while the processor-2 keeps waiting a Little bit, that should be to avoid the bugs or reading some rubbish data from the memory. In our scheme, processor-2 has to perform the NAF conversation before start performing the adding operations, while Processor-1 is now performing the doubling operations. After performing the doubling operations, processor-1 will save the results (doubled points) in the shared array. In this case, doubled points are already saved in the memory and ready to be readable from Processor-1. So, we can say that our method is managed automatically without adding some extra code or protocols to manage reaching to the memory at the same time.

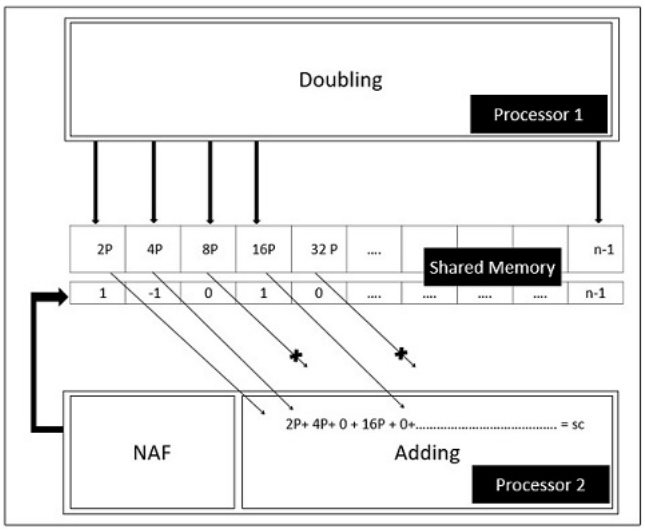

Fig. 3. General Framework of Scheme

V. Results

We can summarize the proposed method is that, extracting a new algorithm that combines two algorithms. Add-Subtract Scalar Multiplication with Non-Adjacent Form. Then performing the parallel computing on the extracted algorithm which is algorithm 3.

In the figure 4, we can recognize clearly, the result we got from the real implementation of our proposed scheme

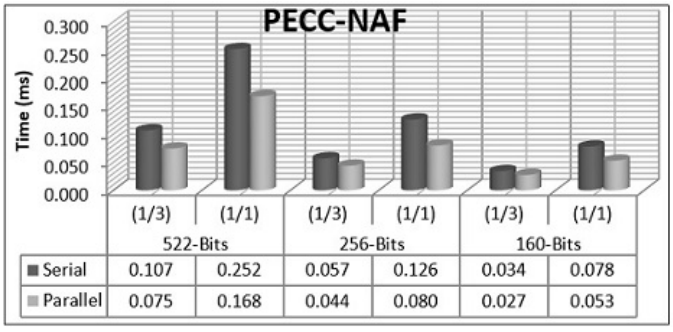

Fig. 4. The Results

In the implementation, we tested three different key size 160-bits, 256-bits and 522-bits. Each size with two cases of occurrence the non-zero bits in the key given by N = Total of non-zero bits / key-size.

The result shows that the best cases when the number of non zero bits is enclosed to the key size. The best result we got in case of key-size is 256-bits and 522-bits in case of $N = 1$

As all most parallel application, its 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 overall the scheme to get the best result as shown in Figure 4. We wrote the both codes sequential and parallel in Visual C++.Net. We use the Open MP Library that is supported in Visual C++.Net package to write the parallel

section in parallel code. Its important to note that we use Intel Dual-Core machine to test the codes using Windows 7. The specification of machine is Intel Dual core with 1 MB L2 cache memory. We performed each key size in both cases of N 10 times and the average execution time is taken as shown in Figure 5.

The aim of our scheme is to reduce the execution time of computing ECC scalar multiplication. Therefore, exploit the variance time between parallel execution and standard case to increase the key size of ECC. The increasing of the key size will be in same execution time compared with sequential time. As example in case of key size is 160-bits and $N = 1/3$, the both serial and parallel time are presented in milliseconds, see Figure 5.

		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%

Fig. 5. The time in milliseconds

In case of key size is 160-bits, the execution time of serial version as shown in Figure 5 is 0.034 ms and parallel time is 0.027 ms the variance among them is 0.007 ms. The execution time for each iteration in serial time is around 0.034/160 = 0.00021. Therefore, we can increase the key size according to the variance among serial and parallel time as follows: 0.007 / 0.00021 = 33.3 times, then we can expand the key size 33 bits higher than 160-bits. We can summarize, that by applying our scheme the execution time of the scheme with key size 190-bits same execution time for serial case with 160-bits. And therefor, the security level of ECC with 190-bits key size is more than security level of ECC with 160-bits key size of the ECC.

VI. Conclusion

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 into two processors. The first processor perform the doubling operations while the second perform the NAF conversion and adding-subtracting operations. We tested our proposed method with different key size. The result shows the best result when number of non-zero bits enclosed to the key size of ECC. The future work is testing almost signed binary representation algorithms such as MOF, JSF complementary method and other methods.

References

[1] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), pp.120-126, (1987).

[2] N. Gura, A. Patel, A. Wander, H. Eberle and S. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In International workshop on cryptographic hardware and embedded systems (pp. 119-132). Springer, Berlin, Heidelberg, (2010).

[3] N. Koblitz, Elliptic curve cryptosystem, Mathematics of Computation 48 203209 (1987).

[4] V. Miller, Use of elliptic curves in cryptography, in: Advances in Cryptology, Proceedings of CRYPTO85, LNCS, vol. 218,pp. 417426, (1986).

[5] K. Okeya, K. Schmidt-Samoa, C. Spahn and T. Takagi. Signed binary representations revisited. In Annual International Cryptology Conference (pp. 123-139). Springer, Berlin, Heidelberg, (2004).

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

[7] A. Booth, A signed binary multiplication technique, Journal of Applied Mathematics 4 (2) 236240, (1951).

[8] K. Pathak, and M. Sanghi. Speeding up computation of scalar multiplication in elliptic curve cryptosystem. International Journal on Computer Science and Engineering, 2(4), pp.1024-1028. (2010).

[9] X. Huang, G. Shah, and D. Sharma. 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 (Vol. 2, pp. 1226-1230). (2010).

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

[11] S. Asif and Y. Kong. Highly parallel modular multiplier for elliptic curve cryptography in residue number system. Circuits, Systems, and Signal Processing, 36(3), pp.1027-1051,(2017).

[12] A. Gutub and S.Arabia. Remodeling of elliptic curve cryptography scalar multiplication architecture using parallel jacobian coordinate system. International Journal of Computer Science and Security (IJCSS), 4(4), p.409, (2010).

[13] R. Nicolas. ICSA Guide to Cryptography, McGraw Hill, New York, USA, (1999).

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

[15] G. Reitwiesner, Binary Arithmetic, Advances in Computers, (1960).

[16] C. Chang, Y. Kuo, C. Lin,Fast algorithms for common multiplicand multiplication and exponentiation by performing Complements, in: Proceedings of the 17th International Conference on Advanced Information Networking and Applications,pp. 807811 (2003).

[17] B. Ansari and H. Wu, 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).

[18] M. Anagreh, A. Samsudin and M. Adib Omar, 2014. Parallel method for computing elliptic curve scalar multiplication based on MOF. Int. Arab J. Inf. Technol., 11(6), pp.521-525, (2014).