Analyzing the Point Multiplication Operation of Elliptic Curve Cryptosystem over Prime Field for Parallel Processing

Arumugam Sakthivel[1] and Raju Nedunchezhian[2]
[1]Centre for Cryptography and Network Security, Adithya Institute of Technology, India
[2]Department of Information Technology, Sri Ramakrishna Engineering College, India

Abstract: *The Elliptic Curve Cryptosystem shortly called as (ECC) is one of the asymmetric key cryptosystems, which provides a high security for wireless applications compared to other asymmetric key cryptosystem. The implementation of this algorithm over prime field Z_p has a set of point operations, which are point addition, point subtraction, point multiplication, point division, point inversion, and point doubling. In these operations, the time complexity of the point multiplication is higher than any other time complexity of ECC point operations. So, it is necessary to find out an alternative implementation for point multiplication to take minimum amount of clock cycles, to reduce power consumption, and to support the software scheduling for parallel processing on arithmetic operations during execution. Considering this, the proposed implementation is very useful to perform encryption or decryption on texts, and also for analysing the strength of encryption or decryption computation.*

Keywords: *ECC, asymmetric key cryptosystem, time complexity, clock cycles, software scheduling, parallel processing.*

Received January 31, 2012; accepted February 20, 2013; published online April 4, 2013

1. Introduction

Some of the biggest challenges in the field of wireless communication are to provide a high security for various application services, to reduce power consumption of various process and low bandwidth usage for communication media. It is very important to find out suitable asymmetric key cryptosystem to offer the same. In this case the Elliptic Curve Cryptosystem (ECC) is more suitable than any other cryptosystem such as RSA, Elgamal, NIST and Rabinson because this ECC has certain merits such as high security, less key size, low power consumption and low bandwidth requirements to support wireless communications when it compares with other public key cryptosystem [7].

This ECC has two main categories called as ECC over prime field (Z_p) and ECC over binary field (2^p). In these types, ECC over prime field is used for software implementations and ECC over binary field for hardware implementations [8]. This paper suggests a new approach for ECC over prime field implementation and analyses the same. This cryptosystem consists of elliptic curve cryptography and elliptic curve cryptanalysis. The ECC defines the encryption which converts the original text into the secret text and the decryption, the reverse operation of encryption. In addition, the elliptic curve cryptanalysis analyses the strength of ECC based on various constraints and parameters to improve the speed. This ECC is implemented by using a set of point operations such as point addition, point subtraction, point multiplication, point division, point inversion and point doubling [15]. In these operations, the time complexity of point multiplication is higher than any other point operations on elliptic curve. It is necessary to find out optimized implementations for point multiplication. Some of the already available implementations don't support the code scheduling for parallel processing on arithmetic operations [13]. Hence it is necessary to find out a best implementation for point multiplication which improves the performance of the ECC by using parallel computation [6].

This paper is organized as follows to explain the proposed work. Section 2 gives a background of ECC and section 3 analyze the data structure for software implementation of ECC. Subsequently, section 4 describes the mathematical basics of ECC over prime field and section 5 proposes an innovative technique for ECC implementation. Next, the innovative technique is analysed and compared with existing ECC methodology in section 6. Section 7 concludes some of applications for this proposed methodology of ECC. Finally, the appendix becomes to support the proposed methodology.

2. Basis of Elliptic Curve Cryptosystem

A cryptosystem is a security model which consists of cryptography and cryptanalysis. It is classified into two broad categories such as symmetric key and

asymmetric key cryptosystem. The difference between these two cryptosystem is, the merit of symmetric key cryptosystem becomes the demerit of asymmetric key cryptosystem and it is vice-versa. But the software applications point of view, asymmetric key cryptosystem is better than symmetric key cryptosystem for implementing protocols and various web application security models. Some of the examples for asymmetric key cryptosystem are RSA, Elgamal, NIST and ECC. The asymmetric key cryptography is a game to play with encryption and decryption techniques with two keys. The encryption is used to convert the original message called Plain Text (PT) into secret message known as Cipher Text (CT) with help of sender private key and receiver public key and the decryption vice versa with help of sender public key and receiver private key. And the cryptanalysis is used to analyse and test the strength of the plain text, cipher text and algorithm based on various parameters such as execution time, memory space and design issues of algorithm. The ECC has a set of advantages when it is compared to other asymmetric cryptosystem. They are the length of the key size, less memory space to store keys, low power consumption and low bandwidth because of its key size [4].

The cryptography of ECC consists of three parts which are the key generation, encryption and decryption. The key generation part is to generate two keys called private key and public key for encryption and decryption operations. The procedure of key generation is mentioned in following steps:

1. Define E_q (a, b), where a, b and q are elliptic curve parameters, and q is a prime number.
2. Find out G, where G is point on the elliptic curve whose order is large value of integer n.
3. Select sender private key n_A, where $n_A < n$.
4. Calculate sender public key P_A, where $P_A = n_A \times G$.
5. Select receiver private key n_B, where $n_B < n$.
6. Calculate receiver public key P_B, where $P_B = n_B \times G$.
7. Secret key on sender side $K = n_A \times P_B$.
8. Secret key on sender side $K = n_B \times P_A$.

The procedure of encryption is defined by $CT = (kG, PT + kP_B)$ and the procedure of decryption $PT = (PT + kP_B - n_B(kG))$, where k is a random positive integer. In ECC cryptanalysis, the strength of an algorithm is based on the general behaviour of *PT*, *CT*, both, private key and public key. The encryption of this cryptosystem is mathematically and logically more secure when it compares with other cryptosystems. So, it is very difficult to break an encryption. But there is a possibility to attack an encryption by using implementation attacks [1].

3. Data Structure for Software Scheduling

The literature survey shows that there is no optimal scalar multiplication implementation for parallel processing on arithmetic operations. So, this paper suggests the binary tree to compute point doubling based on divide and conquer strategy [14]. A divide and conquer algorithm works iteratively by breaking down multiplication into two or more sub-problems of the same type, until these become simple to compute directly. After dividing, these solutions are combined to find out the required computation. Normally there are four types of dependencies existing in linear scalar multiplication namely data dependencies, name dependencies, control dependencies and loop carried dependencies during execution which affect the speed of computation [5]. The name dependency means two or more points refer the same name and in data dependency, a point is dependent on another point. Besides a control dependency means that a constraint depends on different constraints of point computing and loop-level dependency. It determines whether point accesses in later iterations are dependent on point produced in earlier iterations.

4. Mathematics Back Ground of ECC

An elliptic curve over a finite field is defined by the general Weierstrass equations (IEEE 1363 standard specifications for public key cryptosystem) [9]:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad \text{Where } a_1, a_3, a_2, a_4, a_6 \in E \text{ (real number)} \tag{1}$$

By substituting different values for x and y in equation 1, the ECC points are generated. The point at infinity denoted by 'O' is the additive identity for the abelian group which is discussed in appendix and it is identified from the ECC points [2]. This elliptic curve can be analysed over polynomial basis (10), dual basis (2), triangular basis (6) and redundant basis (3). The finite field discussed in appendix over polynomial basis which known as Galois field is mostly suitable for cryptographic software application [3]. In particular, a prime field (order of a, a is a largest number over Z_a or $GF(a)$) is more suitable for software applications. The general equation 1 simplifies as the following:

$$y^2 = x^3 + ax^2 + b \quad a,b \in Z_r \text{ and } 4a^3 + 27b^2 \neq 0 \tag{2}$$

And it is called non-super singular form of equation [4, 6]. Then points on equation 2 are known as affine and projective co-ordinates [11]. The point addition over $E_r(a, b)$ is the basic operation on Elliptic curve and the rule of point addition is $P, Q \in E_r(a, b)$ where P and Q are points on Elliptic curve straight line and the result of $P + Q$ is also on the same line.

- *Case 1: If* $P = (x_1, y_1)$, $Q = (0,0) \in E$ *then*

$$R = P + Q = P + O = P \quad \text{Where } O \text{ is origin points} \tag{3}$$

- *Case 2: If* $P = (x_1, y_1)$, $Q = (x_1, -y_1) \in E$ *then*

$$R = P + Q = P + (-P) = O \quad (4)$$
Where -P is inverse of P

- *Case 3: If* $P = (x_1, y_1)$, $Q = (x_2, y_2) \in E$ *and* $(P \neq Q)$ *then*

$$R = P + Q = (x_3, y_3) \quad (5)$$
Where $x_3 = \lambda - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, $\lambda = ((y_2 - y_1)/(x_2 - x_1))$

- *Case 4: If* $P = (x_1, y_1)$, $Q = (x_2, y_2) \in E$ *and* $(P = Q)$ *then*

$$R = P + Q = (x_3, y_3) \quad (6)$$
Where $x_3 = \lambda - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, $\lambda = (3x_1^2 + a) / 2y_1))$, *and* $(P = Q)$.

The importance of ECC is to compute the point multiplication $Q = kP$, where k is a scalar value and P is a point on the elliptic curve [10]. The number of points in $E_r(a, b)$ is approximately equal to the number of elements in Z_r, namely p elements and bounded by $(r + 1 - 2\sqrt{r}) \leq N \leq (r + 1 + r\sqrt{p})$, which is known as scalar multiplication [12]. It is denoted in Figure 1 and the equation as follows:

$$kP = P + P + P + P \ldots (k-1)\ P + kP \quad (7)$$
('k' times of point addition).

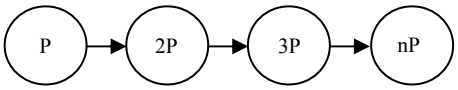

Figure 1. Linear point multiplication of nP, where n is the value of k and p is a point.

This linear scalar multiplication is implemented based on the following algorithm 1:

Algorithm 1: Linear Multiplication Computation
Input: Scalar value k , Point P
Output: kP
1. *Integer I = 1*
2. $Q_0 = (0, 0)$
3. *Repeat*
   a. *If* $(P(x, y) == Q_0(0, 0))$ *then*
      $Q_0(x, y) = P(x, y)$;
   b. *Else if* $(P(x, y) == Q_0(x, -y))$ *then*
      $Q_0(x, y) = P(x, 0)$;
   c. *Else if* $(P(x, y) == Q_0(x, y))$ *then*
      $Q_0(x, y) = P(x, y) + Q_0(x, y)$;
   d. *Else //* $(P(x, y) != Q_0(x, y))$
      $Q_0(x, y) = P(x, y) + Q_0(x, y)$;
   e. *End if*
   f. $I = I + 1$;
4. *Until* $(I \leq k)$;

This point multiplication is most important operation in ECC for encryption and decryption of cryptography and also used in various types of attacks by using cryptanalysis [1]. So, it is necessary to optimize the point multiplication for fast computation.

5. Proposed Methodology

As mentioned in the above reason, the point multiplication is implemented based on the k value from equation 7 in the form of binary trees and skew trees. Each node of these trees holds a point value. Finally, the summing of skew tree value and binary tree value computes a kP value and this computation is diagrammatically represented in Figure 2. When this k value is divided by 2 every time, the quotient and remainder values are obtained. Besides, the binary tree is created by using quotient value and points that are used to compute point doubling operation based on the equation 6. And the same time skew tree is also formed by using reminder value and points that are used to compute point addition operation based on the equation 7. Finally, these two trees are summed by using equations 4, 5, 6 and 7 to compute point multiplication for kP. Algorithms for binary tree computation, skew tree computation, summation of these two trees and formation of Binary tree or Skew tree are explained in algorithms 2, 3, 4, and 5:

Algorithm 2: Point Computation of kP
Input: Scalar value k (assume k = 15), Point P
Output: kP
1. *Point* $P_1 = (0, 0)$;
2. *Point* $P_2 = (0, 0)$;
3. *If k = 1 then*
   $P = P$
4. *Repeat*
   a. $Q \leftarrow k/2$;
   b. *If* $(Q > 0)$ *then*
      P_1 = *call PointDoublingBinary(Point P);*
      $P = P_1$;
   c. *End if*
   d. $R \leftarrow k \bmod 2$;
   e. *If* $(R = 1)$ *then*
      P_2 = *call PointDoublingSkew (Point P);*
      $P = P_2$;
   f. *End if*
   g. $k = k/2$;
5. *Until* $(k > 1)$;
6. *Call Procedure of PointSummazation(Point* P_1, *Point* P_2*)*

Algorithm 3: PointDoublingBinary(Point P)
// Point Doubling Operations of Binary of kP
Input :Point P, Quotient Q.
Output: QP is summation of P
1. *Point Sum (0, 0);*
2. *Sum = QP*
   a. *If (P (x, y) == Sum(0, 0)) then*
      Sum (x, y) = P (x, y);
   b. *Else if (P (x, y) == Sum (x, -y)) then*
      Sum (x, y) = P (x, 0);
   c. *Else if (P (x, y) == Sum (x, y)) then*
      Sum (x, y) = P (x, y) + Sum (x, y);
   d. *Else // (P (x, y)! = Sum (x, y))*
      Sum (x, y) = P (x, y) + Sum (x, y);
   e. *End if*
3. *Return Sum*

Algorithm 4: PointDoublingSkew(Point P)
// Point Doubling Operations of Skew of kP
Input: Point P, Reminder R
Output: RP is summation of P
1. *Point Sum = (0, 0);*
2. *If (P (x, y) == Sum (0, 0)) then*
   Sum (x, y); = P (x, y);

3. *Else if (P (x, y) == Sum (x, -y)) then*
   Sum (x, y) = P (x, 0);
4. *Else if (P (x, y) == Sum (x, y)) then*
   Sum (x, y) = P (x, y) + Sum (x, y);
5. *Else // (P (x, y)! = Sum (x, y))*
   Sum (x, y) = P (x, y) + Sum (x, y);
6. *End if*
7. *Return SUM*

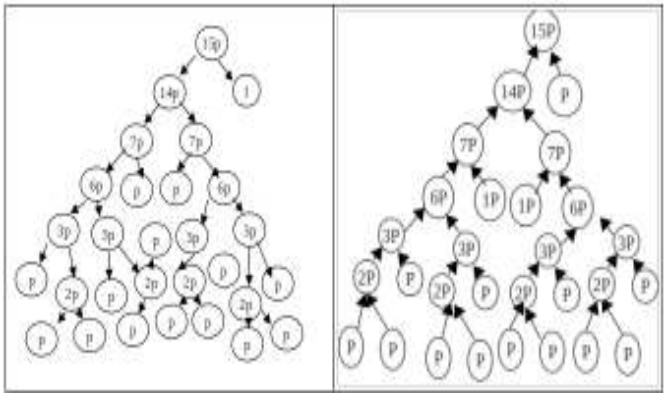

Figure 2. Point multiplication of kP by using binary tree computation, where k is 15 and p is point value.

Algorithm 5: Procedure of PointSummazation
(Point P_1, Point P_2)
//Point Summarization of Binary and Skew Operations of Binary.//
Input: Point P_1,Point P_2.
Output: kP is summation of P_1 and P_2
1. *If (P_1 (x, 0) == P_2 (x, y)) then*
   Sum (x, y); = P_1 (x, y);
2. *Else if (P_1 (x, y) == P_2 (x, 0)) then*
   Sum (x, y); = P2 (x, y);
3. *Else if (P_1 (x, y) == P_2 (x, -y)) then*
   Sum (x, y) = P_1 (x, 0);
4. *Else if (P_1 (x, -y) == P_2 (x, y)) then*
   Sum (x,y)= P_2(x,0);
5. *Else if (P1(x,y)==P_2(x,y)) then*
   Sum (x, y) = P_1 (x, y) + P_2 (x, y);
6. *Else // (P_1 (x, y)! = P_2 (x, y))*
   Sum (x, y) = P_1 (x, y) + P_2 (x, y);
7. *End if*
8. *Return Sum*

The above proposed methodology minimizes the number of loop carried dependency, data dependence and control dependency which helps to improve software scheduling. This software scheduling minimizes the different hazards and stalls which occur at the time of execution for parallel processing.

But the linear scalar multiplication equation 7, *P* is repeated k times alternatively to perform the different case of point multiplication operation. During this processing, the recent iteration is always dependent on early iteration known as loop carried dependency [5]. This type of data dependency creates *N-1* times of hazards and stalls (write after read, read after write, write after read) to affect loop level parallelism [5]. If it is not optimized, it will affect the performance of the computing. But the proposed divide and conquer strategy using trees will reduce number of dependence into log_2N+1 dependent operations [14].

When the encryption or decryption is implemented by using the proposed technique, it reduces execution time of point multiplication. Also, it avoids early and late iterations of point multiplication for encryption or decryption on every occurrence. If the file size is large, the proposed implementations will more suitable to reduce the number of point computations and it executes rapidly which improves the speed of system. And also it is more useful to analyse the PT and CT based on the general characteristic of algorithm.

6. Performance Analysis

The $y^2 = x^3 + ax + b$ equation is selected based on the condition $4a^3 + 27b^2 \neq 0$ to support a set of points for Elliptic Curve Cryptography and $E_p(a, b)$ is defined by using $E_{11111}(1, 1)$. Then the point ($x = 0, y = 1$) is taken from this set to compute point multiplication for both linear scalar multiplication and the proposed method multiplications. In proposed case there are three cases to analyze point doubling by using k value.

To evaluate the performance of the proposed work, the system with Intel (R) Core(TM)2 Quad CPU Q800 at 2.33 GHz speed and 2 GB RAM space configuration is considered under the Ubuntu 10.04 operating system. For experimental to simulate the results, the proposed model is implemented and tested through the following parameter to measure clock cycles for both linear and proposed point multiplication by using structured programming language 'C' as shown in Table 1.

Table 1. Different parameter value needed for simulating point multiplication of kP.

Parameter	Type	Value
E(p)	Input	E(11111)
a	Input	1
b	Input	1
x	Input	0
y	Input	1
P	Input	(x,y)
k	Input	Number of times(N)
k_i	Input	0<N<p
kP	Output	Point multiplication value
kP Execution Time	Output	Number of clock pulses

First case, there is no remainder for k value in all iterations. It means that $k=2^N$ where $N>0$ is called the best case. Because it takes only log_2N times to compute *kP* based on quotient value and no need to compute skew tree computation. But second case, there is a remainder and quotient value for k in all iteration. So, the k value is in the form of 2^N-1 where $N>0$ becomes the worst case. This k value takes only log_2N times to compute *kP* based on quotient value as well as remainder value and the time complexity is defined by $2log_2N$ times. For some iteration there is a

remainder for k value which is not in 2^N or 2^N-*1*. This case is known as average case. The computation time of this case is defined by $log_2N+Prob\{log_2N\}$ times. All three cases, time complexities are redefined by log_2N+1, $2log_2N+1$, and $log_2N+Prob\{log_2N\}+1$. The value 1 is included for finally combining quotient, and remainder values to compute *kP*. The experimental results of linear and proposed point multiplication are as shown in Table 2.

Table 2. Execution time in clock pulses for linear and proposed point multiplication of kP.

Compute kP		Number of Clock Pulses		
i (1 to k)	2^i	Best	Worst	Linear
1	2	0	0	0
2	4	1	1	3
3	8	2	3	4
4	16	3	5	6
5	32	4	6	13
6	64	5	7	24
7	128	6	9	46
8	256	6	10	91
9	512	7	11	191
10	1024	7	11	380
11	2048	8	12	748
12	4096	9	13	1503
13	8192	9	14	3054

Using these samples, the proposed method is analysed in different perspective by using a chart as shown in Figure 3. In this graph, the x-axis denotes k values in the form of 2^i and y axis numbers of clock pulses that are required to compute *kP*. When the *k* value is increased, the computation time is measured with clock pulses for *kP*. It is rapidly increases clock cycles for each value of k in linear scalar multiplication up to 3054. Next, the execution time of tree point multiplication is measured for best and worst cases. In this case, the graph shows that the execution time of the tree point multiplication is better than linear point multiplication. And the graph shows the best case of the point multiplication for the value of *k* within 10 clock cycles. The ratio is identified as $k \approx log_2k+1$ between linear method and best case point processing.

Also, the graph indicates that the worst case point multiplication within 15 clock pulses for the k value. So this comparison shows that the ratio is $k \approx 2log_2k+1$ between linear method and worst case point computation. This proposed technique also helps to minimizes number of data dependencies, control dependences and loop carried dependences which are identified based on different levels of trees. These limited number of dependences are more useful when it is implemented by using parallel processing. This methodology is used to utilize the numbers of hardware parts in minimum level for computing point multiplication, to increase speed of the processing and to increase the life time of hardware units.

The other exponentiation algorithms such as left to right binary methodology and right to left binary methodology are similar to the proposed binary tree methods, but there is no scheduling in those methodologies. These two methodologies can't avoid earlier iterations or late iterations. When the users select a large size file, the proposed methodology is more suitable for existing exponentiation algorithms.

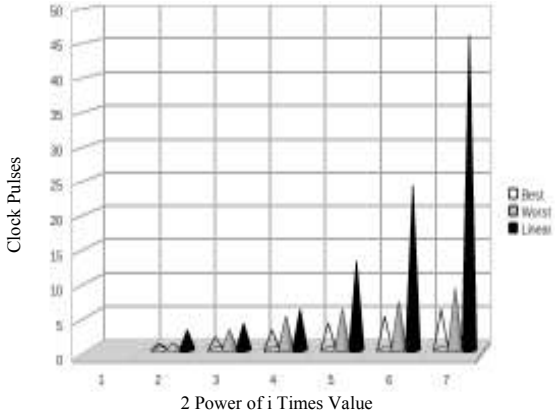

Figure 3. Comparison among linear vs worst case & best case point multiplication of kP.

7. Conclusions

The traditional ECC uses linear scalar point computations for point multiplication. So it needs more clock pulses to compute its processing. But this proposed work of point multiplication using divide and conquer reduces the number of clock pulses and power consumption and it also increases the performance of ECC. This method is also used for scheduling the code which converts dependent operations into independent operations for arithmetic processing on any type of curve on elliptic curve equation. And also it avoids hazards and stalls on data path and control path for parallel processing. This computation is very much useful when it is trying to break security system through cryptanalysis.

Acknowledgements

The authors would like to thank Adithya Institute of Technology and its Research Centre for Information Security and Cryptography to do this work.

References

[1] Catherine H., "Design of Secure Cryptography Against Threat of Power- Attacks in DSP-Embedded Processors," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 1, pp. 92-113, 2004.

[2] Chelton N. and Benaissa M., "Fast Elliptic-Curve Cryptography on FPGA," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 16, no. 2, pp. 198-205, 2008.

[3] Chen J. and Debiao H., "An Efficient Certificateless Designated Verifier Signature Scheme," *the International Arab Journal of*

Information Technology, vol. 10, no. 4, pp. 389-396, 2013.
[4] David J., Welsh M., and Smith D., "Implementing Public Key Infrastructure for Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 4, pp. 1-23, 2008.
[5] Hennessy L. and Pattersom A., *Computer Architecture a Quantitative Approach*, Elsevier, Morgan Kaufmann Publishers, USA, 2007.
[6] Jansirani A., Rajesh R., Balasubramanian R., and Eswaran P., "Hi-Tech Authentication for Palette Images Using Digital Signature and Data Hiding," *the International Arab Journal of Information Technology*, vol. 8, no. 2, pp. 117-123, 2011.
[7] Jarvinen K. and Skytta J., "On Parallel of High Speed Processors for Elliptic Curve Cryptography," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 9, pp. 1162-1175, 2008.
[8] Liu S., King B., and Wang W., "Hardware Organization to Achieve High Speed Elliptic Curve Cryptography for Mobile Devices," *Mobile Networks and Applications*, vol. 12, no. 4, pp. 271-279, 2007.
[9] Longa P. and Miri A., "Fast and Flexible lliptic Curve Cryptography Point Arithmetic over Prime Fields," *IEEE Transactions on Computers*, vol. 57, no. 3, pp. 289-302, 2008.
[10] Pradeep M., "Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems," *IEEE Transactions on Computers*, vol. 55, no. 8, pp. 1000-1010, 2006.
[11] Sakiyama K., Batina L., Preneel B., and Verauwhede I., "Multicore Curve Based Cryptoprocessor with Reconfigurable Modular Arithmetic Logic Units over GF(2^n)," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1269-282, 2007.
[12] Sandro B. and Roberto G., "Effects of Instruction-Set Extension on an Embedded processor: A Case Study on Elliptic-Curve Cryptography over GF(2m)," *IEEE Transaction on Computers*, vol. 57, no. 5, pp. 289-302, 2008.
[13] Stallings W., *Cryptography and Network Security*, Prentice Hall, USA, 2006.
[14] Weiss M., *Data Structures and Algorithm Analysis in C*, Pearson Education, New York, USA, 2003.
[15] Yuan J. and Hung C., "Elixir: High-Throughput Cost-Effective Dual-Field Processors and Design Frame Work for Elliptic Curve Cryptography," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 11, pp. 1567-1580, 2008.

Arumugam Sakthivel is a research scholar at Anna University, Coimbatore, Tamil Nadu, India. He has more than 11 years experience in teaching, and he is currently working in Adithya Institute of Technology, Coimbatore. He has published 4 papers in international journal and 2 papers in international conference. He is also a reviewer of IAJIT and IAENG journals. He edited and authored a chapter in Object Oriented Programming book, which is published in India. His area of interest are mobile computing, soft computing, advanced computer architecture, cryptography, and network security.

Raju Nedunchezhian is currently working as a professor and head of IT dept. He has more than 20 years of experience in research and teaching. He obtained his BE, ME and Ph.D degrees in computer science and engineering. Recently, he has obtained AICTE grant for conducting research in data mining. His research interests include knowledge discovery, soft computing, distributed computing, and information security. He has published 2 books, 45 research journal papers, and 23 conference papers. He is a reviewer for a few international journals/ conferences. He is also the life member of ISTE and ACCS.

Appendix

A. Amdahl's Law

Amdahl's Law states that the performance improvements to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used. It is used to defines Speedup of the machine.

Speedup = (Performance for entire task using the enhancement when possible) / (Performance for entire task without using the enhancement)

(or)

Speedup = (Execution time for entire task without using enhancement) / (Execution time for entire task using the enhancement when possible)

B. The CPU Performance Equation

CPU time = CPU clock cycles for a program × clock cycle time

(or)

CPU time = CPU clock cycles for a program / clock rate

(or)

CPU time = (Instructions / program) ×
(Clock cycles / instructions) ×
(Seconds / clock cycle)

C. Abelian Group, Ring and Finite Field

An abelain group is defined by the following five rules:

1. a Є G and b Є G then a + b Є G
2. a Є G, b Є G and c ЄG then (a + b) + c = a + (b + c) ЄG
3. a Є G and –a Є G then a + (-a) = (-a) + a = 0ЄG
4. a Є G and 0 Є G then a + 0 = 0 + a = a Є G
5. a Є G and b Є G then a + b = b + a Є G

Whare a, b, c, 0, -a, -b, -c are elements of the group. A commutative ring is defined by using the above abelian group rules with the following four rules:

6. a Є G and b Є G then a × b Є G
7. a Є G, b Є G and c Є G then (a × b) × c = a × (b × c) Є G
8. a Є G, b Є G and c Є G then (a + b) × c=(a × c) + (b × c) Є G
9. a Є G, and b Є G then a × b = b × a.

A field is defined by using the above mentioned commutative rings with three rules:

10. a Є G and 1 Є G a × 1 = 1 × a = a Є G
11. a Є G, bЄG and ab = 0 ЄG then either a = 0 or b = 0
12. $a \times a^{-1} = a^{-1} \times a = 1$ where a and a^{-1} Є G

A finite field $GF(p^n)$ is defined with p^n elements.