Smart Card Design Using ECDSA over GF(2^n) with Point Multiplication Scheduling Algorithm

Sakthivel Arumugam

Department of Computer Science and Engineering, Narasu's Sarathy Institute of Technology, Salem, Tamilnadu, India -636305
Author's Mobile: 9790005228, e-mail: asakthivel75@yahoo.com

Abstract

Today, the smart card is used for various applications such as personal attendance, mobile calling, electronic cash transaction, medi-care and financial transaction. But the smart card has a challenge to minimize the design constraints which are the memory, processing circuit, design and an implementation of algorithm. It has an embedded microchip with tiny circuit to handle sensitive data. For this, it needs a good protected security technique against human's unauthorized accesses. Hence this article suggests a technique for the transaction of the safe smart card which is implemented with Elliptic Curve Digital Signature Standard (ECDSA) over binary field with scheduling techniques. Because, ECDSA over binary field is useful for designing the smart card with the less memory space, low bandwidth and limited power constraints on the hardware.

Keywords: *binary field, ECDSA, embedded microchip, scheduling techniques, smart card*

1. Introduction

Now a day, a smart card is playing vital role in the human life and it is used for various applications such as personal attendance, mobile calling, electronic cash transaction, medi-care and financial transaction [13]. It is an artificial card with similar to debit card size and this card has tiny embedded microchip for information processing. The information in microchip is used for the secure transaction to protect from the unauthorized accesses. This sensitive information may be the secret / private key which is embedded in the smart card [1]. These smart cards may be utilized as the memory/microprocessor/Cryptographic Coprocessor/Contactless Smart Cards or USB Tokens.

These smart cards are designed with an asymmetric key cryptosystem to keep the person information in a secure environment. The National Institute of Standards and Technology is suggested the Elliptic Curve (EC) over Finite Field (FF) methodology is more suitable for wireless network based applications and smart card applications. Because, EC needs a short time to generate a key with the limited computing power. Besides, the processing power of EC can be reduced [2]. Because of these reasons, EC over Finite Field based Cryptosystem is suggested for smart cards design. In recent research, Elliptic Curve Based Digital Signature Algorithm (ECDSA) over Finite Field is an important implementations for designing smart cards to process the secure information [14].

The ECDSA based Finite Filed has two different versions: 1. ECDSA over binary filed GF(2^K) and 2: ECDSA over prime field GF(K) [12]. The ECDSA over GF(2^K) is suitable for hardware implementations and the ECDSA over prime field GF(K) for software application. So the ECDSA over GF(2^K) is more convenient for the processing information in smart card applications. But it has some limitations. One of the major limitations is no proper implementation for ECDSA over binary field [3]. Hence, this article suggests a hardware scheduling technique to optimize Elliptic Curve point operation over GF(2^k), which this speed up the computing through the parallel processing.

For this, the article has six section to explain in detail as follows: section II- explains the overview of finite field and ECDSA over binary field, section-III describes the literature survey of different methods, section-IV introduces an innovative implementation for ECDSA over binary field, section-V analyse and compare with existing ECDSA and section-V concludes with future enhancement for various applications which may be implemented with this proposed methodology.

2. Study of ECDSA over Binary Field

ECDSA over GF(2^K) is an effective implementation for smart cards to support speedy operations, because it don't need coprocessor for computing. When it compares with RSA (Rivest, Shamir and Adleman) implementation, it provides a greater security level for a small size of key and also save the processing time [2]. For an example, ECDSA with 163 bit key length and RSA based Digital signature with 1024 bit key provide the same security level. Besides, the effective and efficient way of computing or transmission is to use the minimum length of key bits through the smart card. In this case, it is very difficult to store the 1024 bits of lengthy key inside the smart cart because of the tiny memory, but it is possible to store 163-bit of lengthy key value [6]. Hence, the proposed methodology in this paper is based on ECDSA over GF(2^K) which also needs only less power for computing. And also the proposed methodology fulfills the smart cards requirements in terms such as of memory, processing and cost. The private key of customer is used for signing operation in ECDSA over GF(2^K), which known as cryptographic token and it can be protected inside the smart card. In this section, the basics of ECDSA over GF(2^K) is discussed briefly.

2.1 ECDSA over GF(2^K)

ECDSA over GF(2^K) is the study of Finite Field theory, Elliptic Curve equations and Digital Signature Algorithm [7]. The following subsections explain one by one.

2.1.1 Binary Field GF(2^K) : It is defined with the help of Abelain Group, Commutative Ring and Finite Field through the number theory and remainder theorem modular [11].

Theorem-1: An Abelain Group (G,⊕) has a set of elements (Assume a,b,c,0,-a,-b & -c∈G) which satisfies the following laws through the basic operation ⊕ (binary addition in the form of X-OR): 1.a⊕b∈G, 2.(a⊕b)⊕c=a⊕(b⊕c)∈G , 3.a⊕(-a)=(-a)⊕a=0∈G, 4.a⊕0=0⊕a=a∈G, 5.a⊕b=b⊕a ∈ G.

Theorem-2:A commutative ring (R,⊕,⊗) satisfies an algebraic structure of G through two operations such as addition and multiplication (Left shift with XOR operation) with the following rules: a⊗b∈R, (a⊗b)⊗c=a⊗(b⊗c)∈R, (a⊕b) ⊗c=(a⊗c) ⊕(b⊗c)∈R and a⊗b=b⊗a ∈ R.

Theorem-3: A Field (F,⊕,⊗) is a commutative ring with three more rules:. a⊗1=1⊗a=a, a⊗b=0∈F than either a=0 or b=0 and $a \otimes a^{-1}=a^{-1} \otimes a=1$ where a, $a^{-1} \in F$.
Theorem 4: A Finite Field GF(2^K) has 2^K binary elements over binary addition (⊕) and binary multiplication operations(⊗). Here, the binary elements are represented in the form of reducible or irreducible polynomial equations.

Here the co-efficient value of polynomial denotes the availability of term (1,0) and the exponent value of polynomial indicates the position of the term. Besides, some irreducible polynomials are known as polynomial generator which produces other elements which are also in the set. The next section describes GF (2^K) for Elliptic curve.

2.1.2 Elliptic Curve over GF (2^k): Here, a set of EC points are defined through Elliptic Curve and manipulated with help of polynomial generator. This generator calculates point multiplication with the help repeated additions of polynomial. These Elliptic Curve points are identified fom the Elliptic Curve equations which may be Weierstrass, Pseudo-random or Koblitz curve equation [4]. Here, the non-super singular EC over the binary field GF(2^K) is considered to identify the EC poins as shown in equation 1 [1].

$$y^2 \oplus x \otimes y = x^3 \oplus a \otimes x^2 \oplus b \text{ , } a, b \rightarrow GF(2^K) \quad (1)$$

where a and b are co-efficient values and x & y are variables.

The different EC point values (x,y) are calculated based on polynomial generator with help of exponent and co-efficient values. The negative co-ordinate values should be avoided based on the equation ($4 \otimes a^3 \oplus 27 \otimes b^2 \mod 2^K$) ≠ 0 [2]. The point addition is an atomic operation and it needs less number of clock cycles, Whereas the point multiplication is a complex operation and it needs more number of clock cycles. It is computed based on the number of times of point addition. It is denoted by P=s⊗p, where 's' is a scalar value and 'P' and 'p' are points on the Elliptic Curve shown in Equation 2. It is also known as Linear point multiplication [5].

$$P = p \oplus p \oplus p \ldots \oplus p \text{ (s times of p)} \quad (2)$$

The point multiplication over EC is calculated based on four different cases ways [3]. They are shown in equations (3), (4), (5) and (6).

Case 1: $p=(x_1,y_1), q=(0,0)$ and $p \neq q$ (It is a point identity addition)

$$p \oplus q=(x_1,y_1) \oplus (0,0) = p \quad (3)$$

Case 2: $p=(x_1,y_1)$ and $q=(x_1, x_1 \oplus y_1)$ (It is a point inverse addition)

$$p \oplus q=(x_1,y_1) \oplus (x_1, x_1 \oplus y_1) = O \quad (4)$$

Case 3: $p=(x_1,y_1), q=(x_1,y_1)$ and $p=q$ (It is a point doubling → left shifting)

$$r=p \oplus q=p \oplus p=2 \otimes p \quad (5)$$

where ($x_3=\lambda^2 \oplus \lambda \oplus a, y_3=x_1^2 \oplus (\lambda \oplus 1) \otimes x_3$) and $\lambda=x_1 \oplus y_1/x_1$

Case 4: $p=(x_1,y_1), q=(x_2,y_2)$ and $p \neq q$ (It is a point addition →left shift with x-or)

$$r=p \oplus q=2 \otimes p+q \quad (6)$$

where($x_3=\lambda^2 \oplus \lambda \oplus x_1 \oplus x_2 \oplus a. y_3=\lambda \otimes (x_1 \oplus x_3) \oplus x_3 \oplus y_1$) and $\lambda=(y_2 \oplus y_1)/(x_2 \oplus x_1)$

Normally the point multiplication which a sequence of additions takes more time, because these sequence of point addition is always dependent on previous point addition. So it is difficult for parallel computation. Through hardware scheduling concept, the proposed methodology suggests parallel processing for point multiplication

2.1.2 ECDSA over GF(2^K) : A smart card authenticates the beneficiary of information to its owner and also confirms a communication party known as authorization [14]. During the communication, the information shouldn't be modified and it should be transmitted without any loss through the confidential way. Hence, The implementation of ECDSA over GF (2^K) provides the above securities through three main parts known as digital signing, digital verifying and key generation. The signature can be produced by an individual person who has the private key in the smart cart, but it can be verified by any of beneficiary. The digital signature creates the signature, which is going to checked by the digital verification. Here, the keys are produced for the digital signing and digital verifying through the key generation [6]. The corresponding procedures are as follows:

Common parameters for ECDSA over GF(2^K):

1. Poly1→An irreducible equation where $2^{k-1} < Poly1 < 2^k$
2. Poly2 → An irreducible generator of Poly1
3. Poly3→ $h^{(Poly1-1)\,/\,Poly2}$ mod Poly1, and h is any polynomial with $1 < h < (Poly1-1)$ where $h^{(Poly1-1)\,/\,Poly2}$ mod Poly1 > 1
4. Private_Key → random polynomial equations 0 < Private_Key < Poly2
5. Public_Key → $Poly3^{Private\_Key}$ mod Poly2
6. n = random integer with 0 < n < Poly2

Digital Signature:

1. rem = ($Poly3^n$ mod Poly1) mod Poly2
2. sign = $[n^{-1} \otimes (H(Mess) \oplus (Private\_key \otimes rem))]$ mod q
   where H() is a hash function to produce message digest
3. Signature ← (rem, sign)

Digital Verification:

1. rev_Mess, rev_rem and rev_sign are received versions of Mess, rem and sign
2. T1 = rev_sign^{-1} mod Poly2
3. T2 = [H(rev_Mess') ⊗T1] mod Poly2
4. T3 =[(rev_rem) ⊗T1] mod Poly2
5. ver = [($Poly3^{T2}$ ⊗ $Public\_key^{T3}$) mod Poly1] mod Poly2

TEST: if ver is equal to rem' , it will be accepted

Here, the key generation, digital signature and digital verification have two, three and six point multiplications respectively for digital signature. But the point multiplication is complex and it is not an atomic operation. The following survey section investigates the different implementations of point multiplication for optimizations.

3. SURVEY OF POINT MULTIPLICATION

The survey is conducted over the different implementations for finding the limitations and constraints. In this article, the important implementations of point multiplication such

as the linear scalar point multiplication, binary scalar point multiplication, Montgomery point multiplication and Jacobian point multiplication are investigated based on the number of atomic operation known as point addition in section 3.1. And also the survey in section 3.2 is focusing on the different dependencies among point multiplications, because these dependencies affect the concurrent processing.

3.1 The different implementations of point multiplication:

The scalar point multiplication (Kp) is the result of adding K times of Point to itself. It means that K times shift the values in left direction with optional of x-or operation based on the equations 5 and 6. Normally it will takes more clock cycles. The second limitation is the possibility of parallel computation on point multiplication. It don't support parallel processing because there are four types of dependences existing in this point multiplications such as register value, register name, predictor and loop carried dependencies [12]. So it affects the speed of computation.

The double and add is another method which calculates the point multiplication based on the point value p, scalar value (K) and its binary representation length (n). It depends upon the binary value of K to calculate point multiplication [7]n. Suppose, the bit value is '1', then the equation 6 is used for point multiplication. Otherwise the equation 5 is used for point multiplication. It is implemented in two different ways. One is left to right and another is right to left binary methodology. This point multiplication tremendously reduces the number of point additions in point multiplication. But it also has four types of dependences which are mentioned in the linear point multiplication. Here, there is a possibility for timing attack to predict the point operations which may be equation 5 or equation 6 based on the clock cycles. It is known as side channel attack.

The next point multiplication is montgomery point multiplication which computes the point multiplication using a fixed amount of time for every iteration [8]. So there is no possibility for a side channel attack [10]. It is similar to double-and-add representation. But it repeats the equation 5 and equation 6 each time to calculate the point multiplication. So it's time complexity is not better the previous point multiplication. It is also not supporting parallel processing because of above mentioned dependences.

The jacobsian point multiplication is an important regular binary point multiplication [9]. It requires more field registers for avoiding point inverse operation. Here, the register name and register value dependences are more. So it can not suitable for parallel computation.

3.2 Dynamic code scheduling

The various point multiplications in the literature survey have four different types of dependencies which are register value, name, predictor branch and loop carried dependencies. These dependencies affect the computations during the execution. Suppose two or more points use the same register for computation, then it is known as the register name dependence. Next, the value of a register is dependent on the value of another register, then it is called as register value dependence. The branch dependence predicts the constraints based on the Equations 4, 5, 6 and 7 to find the path for the next computation. The loop carried dependence has a relationship with the previous, current, and next iteration of point computations. These dependencies affect the processing time and introduce the delay time during the computation [4]. These dependencies are identified through the different types of hazards such as read after write (RAW), write after read(WAR) and write after write. These hazards are occurred to the delay time known as

stalls and are controlled by code scheduling. They are two types of code scheduling to avoid hazards and stalls. One is static and another one dynamic code scheduling [12]. Here the dynamic code scheduling is proposed to avoid hardware hazards and stalls during the computation of point multiplication.

The conclusion of survey is given in the *Table 1* which shows that there is no proper implementation for point multiplication to do in concurrently and also it compares the different dependencies, the time complexities, the number of Hazards and stalls and possibility of parallel processing in the different point multiplications. So this article suggests a technique in next section to compute point multiplication through parallel processing.

Table 1: Survey of the point multiplication over GF(2^K)

			Point Multiplication Methodologies			
			Linear	**Double and Add**	**Montgomery**	**Jacobian**
Characteristics	Binary length of s value in s.P denoted by n	Best Case	2n times of Point addition	n times of point doubling	n times of point doubling+ n times of point addition	n times of point doubling without inversion operation
		Worst case		n times of point doubling+ n times of point addition	n times of point doubling+ n times of point addition	n times of point doubling+ n times of point addition (no inversion)
	Number of Dependences	Data	>	<	>	<
		Control	>	<	=	=
		Loop carried	>	<	=	=
		Register	=	=	=	<
	Occurrence of Hazards and Stalls	RAW	YES	YES	YES	YES
		WAR	YES	YES	YES	YES
		WAW	No	No	No	YES
	Parallel Processing		Not Possible		Not Possible	Not Possible

4. Proposed methodology

The proposed methodology is established a relationship between with the double and add point multiplication and Tomasulo methodology for parallel computation. The double and add point multiplication is illustrated in the subseciton 4.1 through the code scheduling to avoid hazards and Tomasulo methodology in the section 4.2 is introduced with code scheduling for parallel processing.

4.1 Double and Add with code scheduling

In this proposed methodology, the scalar value of point multiplication (s) is used to form Huffman tree for dynamic code scheduling. Here the 's' value is shifted by one bit each iteration and remaining bits in register each time. Based on this 's' value, the sum of point doubling to sum1 using the equation 5 or the sum of point addition to sum2 the Equation 6 is computed with help of the following procedures:

Procedure SUM1(point p, bitvalue b)
1. S1←p
2. S1=←S1⊕ p based on Equation 6.
3. return S1

Procedure SUM2(point p, bitvalue b)
1. S2=p
2. S2=S2 ⊕ p based on Equation 7.
3. return S2

Finally, the sum1 and sum2 are summed to give a result of point multiplication. The procedure is diagrammatically shown in *Figure 1* and its procedure is as follows:

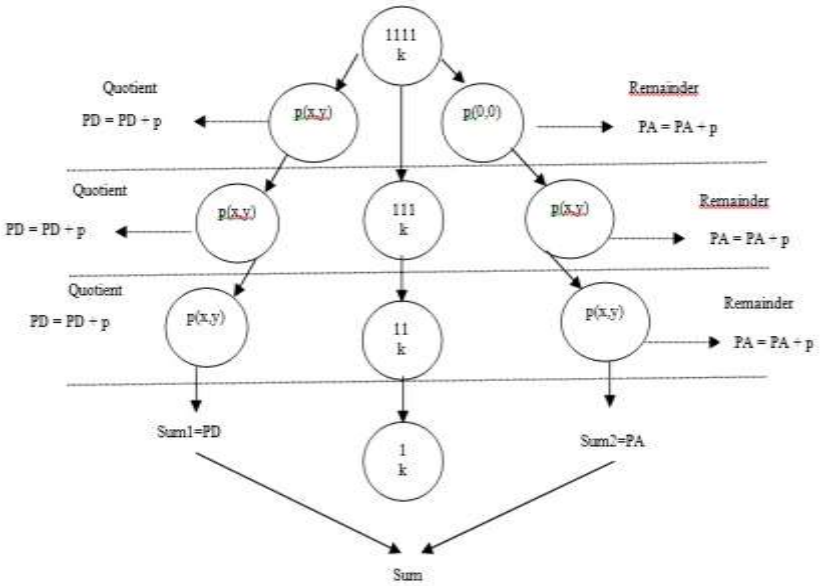

Figure 1. Point multiplication using Huffman Tree

Procedure HuffTreeCompute (point p, scalarvalue s)
point: PA, PD, b: parity bit
1.Is s ←1 then
  1.1 sp ←p
2. PD ← (0,0), PA ← (0,0)
3.Is s > 1 then
  3.1 b ← right shift of s

3.2 is (b = 0) then No operation and exit
3.3 is(b ≠ 0 and s = 0) then
3.3.1 PD ← call SUM1(point p)
3.3.2 p ← PD
3.4 if(b≠1 and s=1) then
3.4.1 PA ← call SUM2(point p)
3.4.2 p ← PA
4. s ←left shift of s
5. goto step 2.
6. sp←PD+PA.
7. return sp

It is no need to refer past point value (p) in the register each time. So the register value dependencies are avoided. And also there is no need to refer the same register name, because the value are stored in two different registers (sum1 and sum2). Besides, the loop carried dependencies are also minimized due to the separate way of handling register dependencies. This type of code scheduling is more useful for parallel processing, because it reduces the different type of hardware hazards and stalls. The next section 4.2 explains the suitability of Tomasulo methodology to avoid hazards for parallel processing .

4.2 Tomasulo Approach for parallel processing

The Tomasulo approach for point multiplication is shown in *Figure 2*. In this approach, an instruction queue issues the different instructions based on the equation 5 and equation 6 for the proposed point multiplication. First, the point is initialized with zero and sends to point addition adder. At the same time, the point p(x,y) sends to point multiplier. Both are computed concurrently and the result is assigned to two different registers for sum1 and sum2. It is repeated until the last bit of scalar value. Finally the sum of sum1 and sum2 is assigned to sum register as point multiplication value (s⊗p). In this approach the point addition and point doubling are executed simultaneously. So the speed of execution is very fast and the corresponding time complexity are analysed in the following section 5.

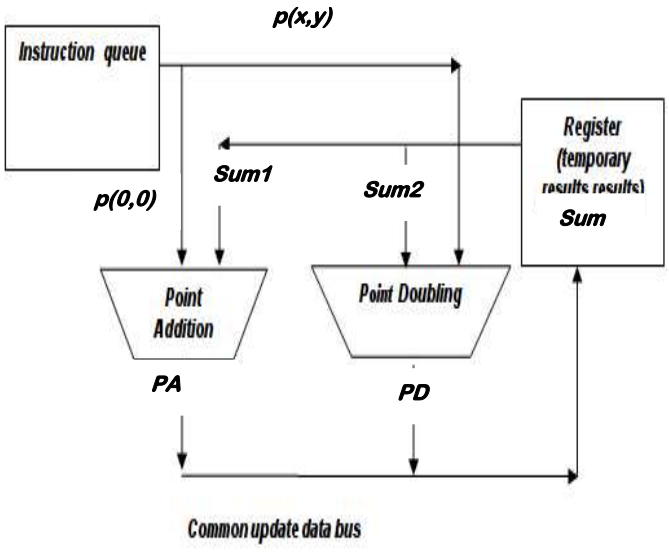

Figure 2. Tomasulo Approach for point multiplication

5. Experimental analysis

For experimental purpose, The following equation is considered for analyzing point operations over Elliptic Curve:($y^2 = x^3 + ax + b$), and the $E_p(a,b)$ is assumed as E_{11111} (1,1) for finding points. The point (x=0,y=1) is taken from this set to test the linear scalar point multiplication as well as the proposed point multiplication. Here, the point multiplication are analyzed in three cases.

Table 2. Shows that the number of clock cycles needed for linear multiplication as well as proposed point multiplication

Compute s.P		Number of Clock pulses		
i (1 to k)	2^i	Best	Worst	Linear
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**

The first is a best case, which has no remainder for the k value in all iterations. It means that the scalar value of 's' is 2^n where N>0. Because it takes only $\log_2 N$ times to compute s.P based on quotient value. The second case is called as worst case, which there is a remainder and quotient values of 's' for all iteration. So the 's' value is in the form of 2^N-1 where N>0. The 's' value takes two times of $\log_2 N$ times to compute s.P based on quotient and remainder values. The corresponding time complexity of this case is defined by $2\log_2 N$ times. Third case is known as average case. In this case, there is a remainder of k value for some iteration and no remainder for some other iteration. Hence, the value of 's' is in the form of 2^N or 2^N-1. The computation time of this case is denoted by $\log_2 N + Prob\{\log_2 N\}$ times.

Finally, All time complexities are redefined as $\log_2 N+1$, $2\log_2 N+1$ and $\log_2 N + Prob\{\log_2 N\}+1$. Here, the value of 1 denotes a final computation of combining the sum1 and sum2 to find s.P. The best case and worst case of proposed methodology are compared with linear scalar point multiplication based on experimental results as shown in the *Table 2*. Besides, this proposed methodology is analyzed in different perspective by using different graphs as shown in *Figures 3, 4, 5 and 6.* The x-axis denotes 's' values in the form of 2^i and y axis denotes the numbers of clock pulses to compute s.P. This proposed strategy will reduce number of dependence into $\log_2 N+1$ for best case and $2\log_2 N+1$ for worst case. So it is useful for parallel computation.

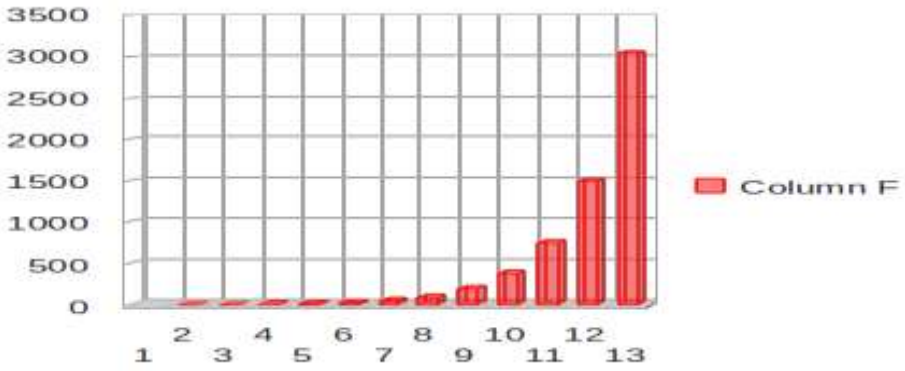

Figure 3. shows that the number of clock pulses needed to compute the linear scalar multiplication.

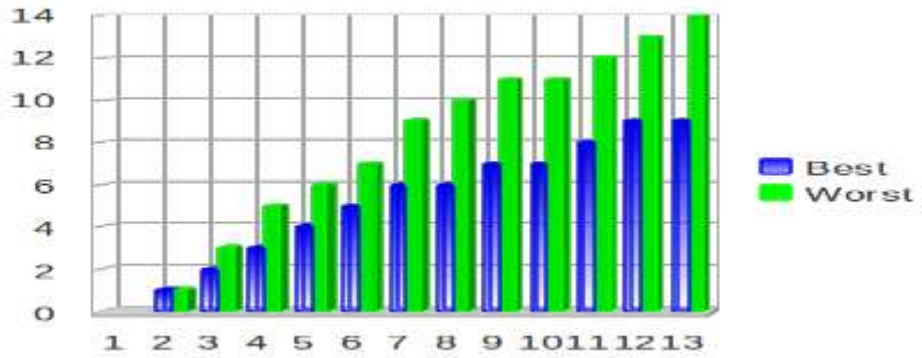

Figure 4. shows that the numbers of clock pulses needed to compute the proposed point multiplication

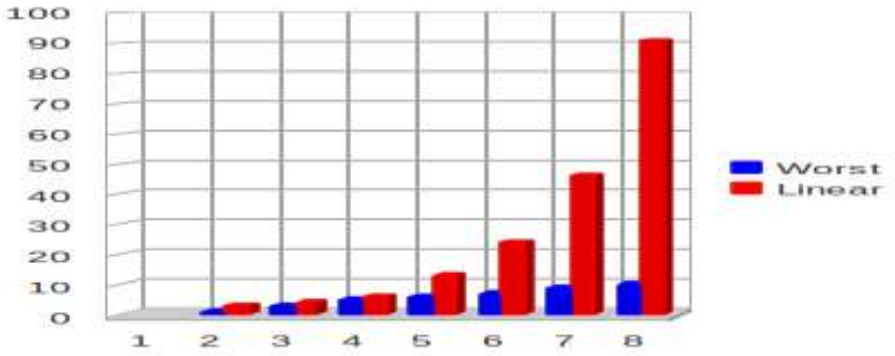

Figure 5. shows that the worst case comparison of the proposed point multiplication with linear point multiplication

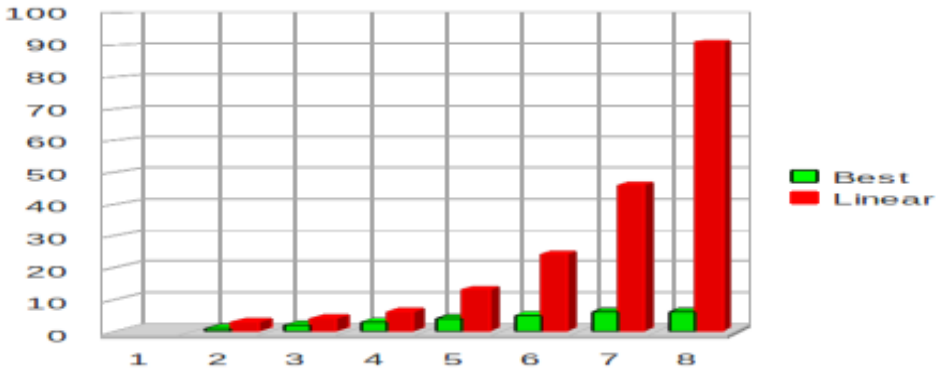

Figure 6. shows that the best case comparison of the proposed point multiplication with linear point multiplication

6. Conclusion with Future Enhancement

The traditional ECDSA over binary field GF(2^n)uses linear scalar point multiplication to perform digital signature signing, verification and key generation. The consequence of using this approach takes more time for execution. But the proposed methodology decreases the number of clock pulses, power consumption and increases the performance of ECDSA. It utilizes hardware units for minimum number of times to find out point multiplication and it increases the life time of hardware units. This is more suitable for smart card application, financial transactions and commercial authentication applications. But the designing circuit of ECDSA is a challenging task. It may be optimized in future with help of nano technology.

References

[1] *N. Panwar, M. S. Rauthan and A. Agarwal, "A comparative analysis and improvement of smart card based authentication scheme",Ninth International Conference on Contemporary Computing (IC3), Noida,india,* ***(2016)****, pp. 1-4.*

[2] *A Sakthivel and R Nedunchezhian, "Analyzing the Point Multiplication Operation of Elliptic Curve Cryptosystem over Prime Field for Parallel Processing", International Arab Jounal of Information Technology. vol. 11, no 4,* ***(2014)****, pp. 3322-328.*

[3] *Arash Reyhani-Masoleh, "Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases", IEEE Transactions on computers. vol. 55,no. 1,* ***(2006)****, pp34-48.*

[4] *Patrick Longa and Ali Miri, "Fast and Flexible Elliptic Curve Cryptography point arithmetic over Prime fields", IEEE Transactions on computers. vol.57,No.3,* ***(2008)****, pp.289-302.*

[5] *Pradeep Kumar Mishra, "Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems", IEEE Transactions on computers. vol 55,No.8,* ***(2006),*** *pp 1000-1010.*

[6] *Gary Locke and Patrick Gallagher, 'Federal Information Processing Standards Publication Digital Signature Standard (FIPS PUB 186-3)', Information Technology Laboratory, National Institute of Standards and Technology,* ***(June, 2009).***

[7] *Sangook Moon, Jaemin Park and Yongsurk Lee, "Fast VLSI Arithmetic Algorithms for High-Security Elliptic Curve Cryptographic Applications", IEEE Transactions on Consumer Electronics. Vol.47,No.3,****(2001),*** *pp. 700-708.*

[8] *E. Savas and C.K. Koc, "The Montgomery Modular Inverse- Revisited", IEEE Transactions on Consumer Electronics. vol. 49, no. 7,****(2000),*** *pp 763-767.*

[9] *Kenny Fong, Darrel Hankerson, Julio Lopez, and Alfred Menezes, "Field Inversion and Point Halving Revisited", IEEE Transactions on Consumer Electronics. vol. 53, no. 8,* ***(2004),*** *pp 1047.*

[10] *Xiaoyu Ruan, and Rajendra S. Katti, "Left-to-Right Optimal Signed-Binary Representation of a Pair of Integers", IEEE Transactions on computers.vol.54, no.2,* ***(2005)****, pp 124.*

[11] *Erkay Savas and Çetin Kaya Ko, "Finite Field Arithmetic for Cryptography", IEEE Circuuts and Systems Magazine. Vol. 10, no. 2,****(2010),*** *pp 40-57.*

[12] *John L.Hennessy & David A. Pattersom, 'Computer Architecture a Quantitative Approach', Elsevier,4th Edition,USA,* ***(2007).***

[13] *J. Kaur, A. Kumar and M. Bansal, "Lightweight cipher algorithms for smart cards security: A survey and open challenges", 4th International Conference on Signal Processing, Computing and Control (ISPCC), Solan, india,* ***(2017)****, pp. 541-546,*

[14] *T. D. Premila Bai, K. M. Raj and S. A. Rabara, "Elliptic Curve Cryptography Based Security Framework for Internet of Things (IoT) Enabled Smart Card", World Congress on Computing and Communication Technologies (WCCCT), Tiruchirappalli,india,* ***(2017)****, pp. 43-46.*

Author Profile

Dr. A. Sakthivel is a professor in the department of CSE, Narasu's Sarathy Institute of Technology, India. He has 20 year's experience in teaching and 10 years in research. He has published 25 Papers in National / International Conferences as well as Journals and reviewed three text books. He got three times best reviewer ward from a reputed SCI journal. He is also reviewer of IEEE Transactions of Cloud Computing. His area of interest is in Cyber Security and Cyber Forensics.