International Journal of Computer Theory and Engineering, Vol. 1, No. 4, October2009
1793-8201

# A Study of Suitability and Effectiveness of Various Implementation Options Of Finite Field Arithmetic on Elliptic Curve Crypto System

Bimal Kumar Meher

***Abstract*—Finite field or Galois field plays an important role in efficient architecture design and implementation of Elliptic curve cryptosystem. A lot of research work is going on in this area since it is suitable for cryptography as well as error correcting codes useful for digital communication, compact disks etc. In this paper we discuss the basic concepts of finite field and its application to elliptic curve cryptography (ECC). A detailed study and analysis of various implementation options available in finite field has been explored and highlighted for effective system design. In section IX we discuss a few efficient hardware design approaches adopted by many researchers useful for ECC.**

***Index Terms*—Finite field, elliptic curve, multiplier, architecture.**

## I. Introduction

Elliptic curve cryptography (ECC) was invented in the mid eighties independently by Victor Miller & Neal Koblitz. Since its invention, it has proved itself a strong alternative to the public key cryptographic systems like RSA and Deffie-Hellman (DH). It offers equivalent security with smaller key sizes resulting in faster computations, lower power consumption, as well as memory and bandwidth savings. This is because of its flexibility in the underlying mathematical concepts, efficient algorithm development & subsequent efficient implementation in both hardware & software. These advantages are especially important in applications on constrained devices such as smart cards, mobile phones, PDAs.

The efficiency of every public key cryptosystem depends on a hard mathematical problem, which is computationally intractable. For instance, RSA and Diffie-Hellman (DH) rely on the hardness of integer factorization and the discrete logarithm problem respectively. Unlike these cryptosystems, which operate over integer fields, the Elliptic Curve Cryptosystems (ECC) operates over a group of points on an elliptic curve over finite field.

The basic mathematical operation in RSA and Diffie-Hellman is modular integer exponentiation. But, elliptic curve arithmetic relies on a operation called scalar point multiplication, which computes $Q = kP$, where $P$ is a point on a selected elliptic curve & $k$ is a sufficiently large integer which is private in nature. This multiplication gives rise to a new point called $Q$ on the same elliptic curve. Scalar multiplication is performed through a combination of point-additions (which add two distinct points together) and point-doublings (which add two copies of a point together). For example, $11P$ can be expressed as $11P = (2.((2.(2.P)) + P)) + P$.

The security of ECC depends on efficient computation of $k$ for given $P$ & $Q$. Although computing $Q$ is easier, but computing $k$ is too hard. This is, in fact, called Elliptic Curve Discrete Logarithm Problem (ECDLP).

Although applying a brute-force approach to compute all multiples of $P$ until $Q$ is found, looks sound, but $k$ is so large in a real cryptographic application that it would be infeasible to determine $k$ in this way. Mathematicians have given their best effort to attack such problem since many years. But, the best known algorithm to attack ECDLP so far takes exponential time [1], where as sub-exponential time algorithm do exist, to solve the hard problems of RSA [2] & DH [3].

The remaining part of this paper is organized as follows. Section 2 discusses the issues related to finite field & basis representation. A brief idea about finite field arithmetic operations is presented in section 3. Mathematical concepts of Elliptic curves over finite fields and representation of points in different coordinate system is presented in section 4. Performance and complexity metrics is explained in section 5. A note on how to choose architecture is given in section 6. Finally section 7 gives the conclusion.

## II. Finite Field

A finite field is an algebraic system consisting of a finite set F together with two binary operations + & . , defined on F, satisfying the following axioms:

- F is an abelian group with respect to "+";
- F\{0} is an abelian group with respect to ".";
- Distributive: for all $x,y$ and $z$ in F we have:
  - $x.(y+z)=(x.y)+(x.z)$
  - $(x+y).z=(x.z)+(y.z)$.

The order of a finite field is the number of elements in the field. [4] Suggests that there exists a finite field of order $q$ if and only if $q$ is a prime power. In addition if $q$ is a prime power, then there is essentially only one finite field of order $q$; denoted by F$q$ or GF($q$). There are many ways of representing the elements of F$q$ and some of them lead to more efficient implementations of the field arithmetic in hardware or in software.

If $q = p$, where $p$ is a prime, then we can have different finite fields as follows:

GF($p$) called as general prime field.

Manuscript received May 02, 2009. Bimal Kumar Meher is with Information Technology Department of Silicon Institute of Technology, Bhubaneswar, India.(phone+919937156042)

GF( $2^n - c$ ) called pseudo mersenne prime field.

GF( $2^n - 2^s - \ldots - 1$ ) called generalized mersenne prime field.

If $q = p^m$ , where *p* is a prime and *m* is positive integer, then *p* is called characteristic (char.) of Fq and *m* is called the extension degree of Fq. So F*q* is named as extension field. We can have different extension fields as follows:

GF( $2^m$ ), called binary field, when char.=2.

GF( $(2^n)^m$ ), called composite field when char.=2.

GF( $(2^n - c)^m$ ), called Optimal extension field when char.>2.

## III. Finite Field $F_p$ or GF(P)

Let p be a prime number. The finite field Fp, called a prime field, consists of set of integers {0,1,2,3,…, p-1} with the following operations:

Addition: *If* a, b ∈ *F*p, *then* a+b=r, *where* r *is the* reminder of the division of a+b by p and $0 \le r \le p-1$. This operation is called addition modulo p.

Multiplication: If a, b ∈ Fp, then a.b=s, where s is the reminder of the division of a.b by p and $0 \le s \le p-1$. This operation is called multiplication modulo p.

It is to be noted that one of the common aspect in both the arithmetic operations is the reduction modulo p., which is accelerated by the NIST-recommended primes such as $p192 = 2^{192} - 2^{64} - 1$ (this is an example of generalized mersenne prime [5] defined above). This is because these primes can be written as the sum or difference of a small number of powers of 2.Again the powers appearing in these expressions are all multiples of 32. So the reduction algorithms are especially fast on machines with word size 32. For a detailed list of such primes and corresponding algorithms, the reader is referred to [4].

## IV. Finite Field $F_{2^m}$

The finite field $F_{2^m}$, is called a binary finite field. The char.=2 is of special importance because of its easiness in hardware implementation It has a set of m elements $\{a_0, a_1, a_2, \ldots a_{m-1}\}$ in $F_{2^m}$ such that each $a \in F_{2^m}$ can be written uniquely in the form $a = \sum_{i=0}^{m-1} a_i a_i$, *where* $a_i \in \{0,1\}$. So, $a$ can be represented as a binary vector $\{a_0, a_1, a_2 \ldots, a_{m-1}\}$. The set $\{a_0, a_1, a_2, \ldots a_{m-1}\}$ is called a basis of $F_{2^m}$ *over* $F_2$. There are different bases proposed in literatures like Polynomial basis, Normal basis & Dual basis.

## V. Basis Representation in $F_{2^m}$ or $GF(2^m)$

There are three important basis representations in $F_{2^m}$: polynomial (standard, canonical), normal and dual basis. Different basis representation greatly affects the underlying arithmetic operations in the finite field and hardware implementations.

### *A. Polynomial Basis*

A polynomial basis over GF( $2^m$ ) is represented as $\{1, a, a^2, a^3, \ldots, a^{m-1}\}, a \in GF(2^m)$. Let f(x)= $x^m + r(x)$ be an irreducible binary polynomial of degree m. The elements of $F_{2^m}$ are the binary polynomials of degree at most m-1 with addition and multiplication performed modulo f(x). A field element is given as $a(x) = a_{m-1}x^{m-1} + \ldots + a_2 x^2 + a_1 x + a_0$, where $a_i \in \{0,1\}$

### *B. Dual Basis*

Unfortunately the conventional representation of finite field elements in a basis consisting of powers of $a$ is not well suited to hardware implementation of shift registers. [6] Presents a new basis representation by Berlekamp. The dual basis $\{b_0, b_1, b_2, \ldots, b_{n-1}\}$ of $\{1, a, a^2, a^3, \ldots, a^{m-1}\}$, is defined as $tr(a^i b_j) = d_{ij}$ . The trace over GF(2) of an element $x \in GF(2^m)$ is given by: $tr(x) = \sum_{i=0}^{n-1} x^{2^i}$

A typical problem with the dual basis is that, a multiplier designed using dual basis use polynomial basis and dual basis representations; therefore, we also have to address the problem of basis conversion. This problem certainly does not arise when $a$ is a self-dual basis. A basis is called self-dual if it is equal to its dual basis. Self-dual bases of $GF(2^m)$ exist only for few values of m.

### *C. Normal Basis*

A normal basis over GF( $2^m$ ) is represented as $\{b, b^2, b^4, b^8, \ldots, b^{2^{m-1}}\}$ ,where $b \in$ GF( $2^m$ ). So, a field element in such basis is given as $a(x) = a_0 b + a_1 b^2 + a_2 b^4 + \ldots + a_{m-1} b^{2^{m-1}}$ , where $a_i \in \{0,1\}$ . The concept of optimal normal basis(ONB) was

introduced in [11] to reduce the complexity of hardware architecture. Again it can be of two types i.e. Type I & Type II. The basic difference between these two variants is that, Type I has less irreducible polynomials then Type II.

## VI. FINITE FIELD OPERATIONS

The generalized operations on the finite field are addition/subtraction, multiplication, squaring, inversion, exponentiation, reduction and division. However, all the operations are not individually realized in hardware or software. Because some primitive operations like addition can do subtraction under modulo 2 operation, multiplication can be helpful for realizing inversion, division can be implemented by multiplication and inversion and so on. Again these operations are basis dependent except addition/subtraction.

Multiplication under polynomial basis is performed by multiplying polynomial *a(x)* and *b(x)* and taking the modulo with respect to a reduction polynomial *f(x).* The following procedure is commonly used to choose a reduction polynomial: if an irreducible trinomial $x^m + x^k + 1$ exists over $F_2$ ,then the reduction polynomial *f(x)* is chosen to be the irreducible trinomial with the lowest-degree middle term $x^k$. If no irreducible trinomial exists, then select instead a pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, such that $k_1$ has the minimal value; the value of $k_2$ is minimal for the given $k_1$; and $k_3$ is minimal for the given $k_1$ and $k_2$.

Squaring an element in normal basis is a simple cyclic shift of the vector representation of the polynomial *a(x)*
i.e. a linear operation in $F_{2^m}$ as follows:

$$a(x)^2 = \left(\sum_{i=0}^{m-1} a_i b^{2^i}\right)^2 = \sum_{i=0}^{m-1} a_i b^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \bmod m b^{2^i} =$$

$(a_{m-1}a_0a_1\ldots a_{m-2})$

Extended Euclidean algorithm and the other one by employing Fermat's little theorem can achieve inversion in the finite fields. A method for efficiently implementing division was proposed by Itoh and Tsuji [14].

### A. Finite Field Multipliers

Since the central operation of ECC is the elliptic scalar multiplication (described in section 1), this section gives some more insight into finite field multipliers.
Finite field multipliers can be grouped into two major categories, namely serial and parallel. Serial multipliers have less complex structure than parallel but produce only a few result bits per clock cycle [9]. These are more attractive under hard area constraints, since they require fewer gates than parallel ones. On the other hand, parallel ones can perform the total multiplication in one clock cycle at the cost of large chip area [10]. Efficient bit-parallel multipliers for both polynomial and normal basis representation have been proposed [15, 16, 17], including the Mastrovito multiplier [18]. In a recent paper [19], Huapeng Wu has proposed low complexity bit parallel multipliers for three classes of finite fields.

Polynomial multiplication can be efficiently implemented using well-known techniques such as the Ofman Karatsuba method [16]. Field multiplication, i.e. polynomial multiplication combined with reduction, can be implemented using techniques such as the least significant digit (LSD) first or most significant digit (MSD) first multiplication method [8]. Division *a(x)/b(x); a(x), b(x)* $\in$ $F_{2^m}$ is defined as a multiplication of the dividend *a(x)* with the multiplicative inverse of the divisor *b(x).* This also necessitates efficient exponentiation division circuit.

## VII. ELLIPTIC CURVES OVER FINITE FIELDS

There are several ways of defining equations for elliptic curves, which depend on whether the field is a prime field $F_p$ or binary field $F_{2^m}$.

### B. Elliptic curves over GF(p)

**LET *P*>3 BE AN ODD PRIME AND LET** $a, b \in F_p$ **SATISFY** $4a^3 + 27b^2 \neq 0$ **MOD *P*. THEN AN ELLIPTIC CURVE** $E(F_p)$ **OVER** $F_p$ **DEFINED BY PARAMETERS** $a, b \in F_p$ **CONSISTS OF THE SET OF SOLUTIONS OR POINTS** $P = (x, y)$ **FOR** $x, y \in F_p$ **TO THE EQUATION** $y^2 = x^3 + ax + b$ **TOGETHER WITH A SPECIAL POINT** O **CALLED THE *POINT AT INFINITY*.**

The operation in $E(F_p)$ is specified as follows:

**1.** $P + \mathrm{O} = \mathrm{O} + P$ for all $P \in E(F_p)$.

**2.** *If* $P = (x,y) \in E(F_p)$, *then* $(x,y) + (x,-y) = O$. The point $(x,-y) \in E(F_p)$ is denoted as $-P$, and is called the negative of $P$.

**3.** Let $P = (x_1, y_1) \in E(F_p)$ and $Q = (x_2, y_2)$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where $x_3 = l^2 - x_1 - x_2$, $y_3 = l(x_1 - x_3) - y_1$, and $l = (y_2 - y_1)/(x_2 - x_1)$.

**4.**Let $P = (x_1, y_1) \in E(F_p)$. Then $P + P = 2P = (x_3, y_3)$ where $x_3 = l^2 - 2x_1$, $y_3 = l(x_1 - x_3) - y_1$ and $l = (3x_1^2 + a)/2y_1$. This operation is called doubling of a point.

Note that addition of two different elliptic curve points in $E(F_p)$ requires the following arithmetic operations in $F_p$: one inversion, two multiplications, one squaring and six additions. Similarly, doubling an elliptic curve point requires one inversion, two multiplications, two squarings and 8 additions.

Since inversion in $F_p$ is an expensive operation, an alternative method to compute the sum of two elliptic points is to use projective coordinates. In such case, the inversion operation is performed by more multiplications and other less expensive finite field operations.

### C. *Elliptic curves over* $F_{2m}$

A non-supersingular elliptic curve $E(F_{2m})$ over $F_{2m}$ defined by the parameters $a, b \in F_{2m}, b \neq 0$, consists of the set of solutions or points $P = (x, y)$ for $x, y \in F_{2m}$ to the equation: $y^2 + xy = x^3 + ax + b$ together with a special point O called the *point at infinity.*

The operation in $E(F_{2m})$ is specified as follows:

**1.** $P \in E(F_{2^m})$ for all $P \in E(F_{2^m})$.

**2.** $P = (x, y) \in E(F_{2^m})$, then $(x, y) + (x, -y) = O$. The point $(x, -y) \in E(F_{2^m})$ is denoted *–P and is called the negative of P.*

**3.** Let $P = (x_1, y_1) \in E(F_{2^m})$ and $Q = (x_2, y_2) \in E(F_{2^m})$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where

$x_3 = l^2 + l + x_1 + x_2 + a$, $y_3 = l(x_1 + x_3) + x_3 + y_1$

and $l = (y_2 + y_1)/(x_2 + x_1)$.

**4.** Let $P = (x_1, y_1) \in E(F_{2^m})$. Then $P + P = 2P = (x_3, y_3)$ where $x_3 = l^2 + l + a$, $y_3 = l(x_1 + x_3) + x_3 + y_1$ and

$l = x_1 + (x_1 / y_1)$

Note that the addition of two elliptic curve points in $E(F_{2m})$ requires one inversion, two multiplications and eight additions in $F_{2m}$. Doubling a point requires one inversion, two multiplications, one squaring and six additions. Since inversion is an expensive operation comparison to multiplication, hence projective coordinates can be useful, where inversion is expressed as multiplication and other field operations. A hardware and a software implementation of $F_{2m}$ can be found in [12] and [13] respectively, where an inversion costs about 24 and 10 multiplications respectively.

## VIII. Performance and Complexity Metrics

*Several performance and complexity metrics are used to compare and evaluate finite field multipliers. Following is a short description of those metrics.*

**Gate count** This is the main complexity metric which is usually considered as the numbers of 2-input AND and XOR gates, flip-flops and switches or 2-to-1 multiplexers. It is sometimes tied to the silicon area used for implementation using the area and count of an equivalent 2 input NAND gate to represent the hardware complexity.

**Latency** The delay between first input and first output of the multiplier expressed in clock cycles is defined as latency. This measure is of special importance in systolic and semi-systolic design as the output makes a delay of multiple clock cycles after the arrival of the input.

**Regularity and Modularity** These two metrics are interrelated. Design regularity helps in extending a system to perform more operations easily and modularity helps in visualizing the entire system as a combination of independent modules. Polynomial basis multipliers are most preferred in terms of regularity while normal basis are less preferred.

## IX. How to Choose an Architecture

The choice of $GF(2^m)$ multiplier architecture depends heavily on the basis representation as well as hardware complexity and the critical path delay of the architecture. Polynomial basis representation has the advantage over the other bases as it can be performed using ordinary polynomial arithmetic. It is also easier to extend to high order finite fields than the dual or normal basis [8]. In the normal basis where squaring is only a shift left operation is crucial to inversion operation. For example, Massey-Omura multiplier [7] is very effective in performing squaring, exponentiation, and inversion operations. The dual basis yields the simplest architecture. The dual basis multiplier [6] for example, needs the least number of gates, which leads to the smallest area required for VLSI implementation.

*Selecting serial or parallel architecture depends on the availability of operands during computation.*

Coming to systolic and non-systolic, systolic architecture allows for pipelining while non-systolic are more hardware efficient.

Taking all those factors into account although absolute hardware architecture is difficult to realize but it helps in focusing the target. For example if the target is for low hardware complexity then non-systolic design is a better choice. Although, semi-systolic design gives still less hardware complexity than fully systolic but the common control signal in semi-systolic make it difficult to expand the multiplier to higher fields. In a recent paper Meher [20] has proposed a bit-level-pipelined systolic design, which takes nearly half of the time complexity of the corresponding existing design that of Wang and Lin [21], Lee [22], Lee et al. [23] by appropriate cut-set retiming and logic optimization in the processing elements (PEs).

Using composite fields to construct the multiplier architecture is also helpful in lowering the area complexity and increases the modularity of the architecture. Because, multiplication over composite field $GF((2^n)^m)$ is performed

using arithmetic modules of *GF*( $2^n$ ). Multiplier architectures for two different classes of composite fields GF( $(2^n)^2$ ) and GF( $(2^n)^4$ ) and their complexity analysis can be found in [24].

## X. Conclusion

This study of different implementation options in finite field multipliers helps in identifying the suitable target easily and effectively. Although there are many papers on arithmetic operations in finite field, but more and more optimal hardware architecture is what we are looking for. Because of the miniaturization of electronic devices, power efficiency and cost-effectiveness without compromising the security will be the real challenge of future ECC based system design.

## References

[1] R. Balasubramanian and N. Koblitz, "The improbability that an elliptic curve has a sub-exponential discrete log problem under the Menezes-Okamoto-Vanstone algorithm", Journal of Cryptology,11,pp.141-145, (1998)
[2] P. Fahn and M.J.B. Robshaw. Results from the RSA Factoring Challenge. Technical Report TR-501, version 1.3, RSA Laboratories, January 1995.
[3] W. Diffie and M.E. Hellman. "New directions in cryptography", IEEE Transactions on Information Theory, IT-22: 644-654, 1976.
[4] D. Hankerson, A. Menezes and S. Vanstone. Guide to Elliptic Curve Cryptography,springer.
[5] J. Solinas, "Generalized Mersenne numbers", Technical report CORR 99-06, Department of Combinatorics & Optimization, University of Waterloo,1999. Available at http://www.cacr.math.uwaterloo.ca/
[6] E. R. Berlekamp, "Bit-serial Reed-Solomon encoders," IEEE Transactions on Information Theory, vol. 28, no. 6, pp. 869–874, 1982.
[7] J. L. Massey and J. K. Omura, "Computational method and apparatus for finite field arithmetic," U.S. Patent application, 1981
[8] L. Song and K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," IEEE Journal of VLSI Signal Processing, vol. 19,no. 2, pp. 149–166, 1998.
[9] G. Orlando and C. Paar, "A Super-Serial Galois Fields Multipliers for FPGA and its Application to public key Algorithms," in FCCM'99, April 1999.
[10] Christof Paar, "A new Architecture for a Parallel finite field Multiplier with low complexity on Composite fields," IEEE Transactions on Computers, vol. 45, no. 7, pp.856-861, July 1996.
[11] R.C. Mullin, I. M. Onyszhchuk, S. A. Vanstone, R. M. Wilson, "Optimal Normal Bases in GF(pn) ," Discrete Applied Mathematics, Pages 149-161, 1988/89
[12] G. B. Agnew, R. C. Mullin and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over F2155 ," IEEE journal on selected areas in communications, Vol. 11, No. 5, pp. 804-813, 1993.
[13] D. Hankerson, J. Lopez and A. Menezes, "Software implementation of elliptic curve cryptography over fields of characteristic two", draft, 2000.
[14] T. Itoh and S. Tsuji, " A fast algorithm for computing multiplicative inverses in GF(2m) using normal bases." Information and Computation, 78:171-177,1998.
[15] M. A. Hasan, M. Z. Wang, and V. K. Bhargava. "A modified Massey-Omura parallel multiplier for a class of finite fields", IEEE Transactions on Computers, 42(10):1278–1280, November 1993.
[16] C. Paar. Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields. PhD thesis, University at GH Essen, VDI Verlag, 1994.
[17] H. Wu. Low complexity bit-parallel finite field arithmetic using polynomial basis. In C .K. Koc and C. Paar, editors, Cryptographic Hardware and Embedded Systems, LNCS No. 1717, pages 280–291, 1999.
[18] B. Sunar and C. K. Koc. "Mastrovito multiplier for all trinomials. IEEE Transactions on Computers", 48(5):522–527, May 1999.
[19] H. Wu. "Bit parallel polynomial basis multiplier for new classes of finite fields", IEEE Transaction on Computers, 57(8):1023-1031, August 2008.
[20] P. K. Meher, "Systolic and Super-Systolic Multipliers for Finite Field GF(2m) based on irreducible trinomials", IEEE Transactions on Circuits and Systems-I, 55(4):1031-1040, May 2008.
[21] C. L. Wang and J. L. Lin, "Systolic array implementation of multipliers for finite fields GF(2m)",IEEE Transactions on Circuits and Systems,38(7):796-800, July 1991.
[22] [C. Y. Lee, "Low complexity bit-parallel systolic multiplier over GF(2m) using irreducible trinomials, "IEE Proc. Comput. Digit. Tech.,150(1):39-42,Jan. 2003.
[23] C. Y. Lee, J. S. Horng, I..C. Jou and E. H. Lu, " Low complexity bit parallel systolic Montgomery multipliers for special class of GF(2m)", IEEE Transactions on Computers, 54(9):1061-1070, September 2005.
[24] C. Paar, "Efficient VLSI Architectures for bit-parallel computation in Galois Fields", PhD Thesis, Universitat GH Essen, VDI Verlag, 1994.

**Bimal Kumar Meher** is a Senior Lecturer in the Department of Information Technology, Silicon Institute of Technology Bhubaneswar, Orissa, India. He has done his M.Sc. in Electronics from Berhampur University and M.Tech. in Computer Science from Utkal University. Presently he is perusing his Ph.D from Utkal University. His research area is developing algorithm and architecture for recourse-constraint hardware and elliptic curve cryptography.