AN IMPLEMENTATION OF ELLIPTIC CURVE CRYPTOGRAPHY SCALAR MULTIPLICATION BLOCK IN $GF(2^m)$ WITH VHDL

Serkan Acar
serkan@ehb.itu.edu.tr

Ece Olcay Güneş
ece.gunes@itu.edu.tr

***Abstract*—In this study, the finite field $GF(2^m)$ used in elliptic curve cryptography and their properties, elliptic curves for this usage, safety of elliptic curve cryptography are examined. Also, scalar multiplication block which is the most important and the most time consuming block used in elliptic curve cryptography protocols is designed and this block is described with VHDL. The block designed is programmed with FPGA and a simple El Gamal elliptic curve protocol is implemented with constituting computer interface.**

I. INTRODUCTION

Elliptic curve cryptography (ECC) is developed alternatively RSA and DSA public key cryptosystems by Victor Miller and Neal Koblitz in 1985. The security of elliptic curve cryptography depends on discrete logarithm problem defined on an appropriate elliptic curve. The elliptic curve discrete logarithm problem can still not be solved in sub-exponential time. And so, this means elliptic curve cryptography provides equal security as RSA and DSA cryptography with smaller parameters. Working with smaller parameters is ideal for mobile phones or smart cards which requires high processing speed, power dissipation and smaller memory storage. Furthermore, there are many options for implementing elliptic curve cryptography, because implementation of elliptic curve cryptography is dependent on underlying finite field, algorithms for finite field arithmetic, elliptic curve and algorithms for elliptic curve arithmetic chosen.[1]

II. FINITE FIELDS

<G,*> algebraic structure, where G is a set and * is a binary operation, forms a *group* and satisfies the following axioms.

- Closure: For $\forall x, y \in G$, so $x * y \in G$
- Associativity: For $\forall x, y, z \in G$, so $(x * y) * z = x * (y * z) \in G$
- Identity: For $\forall x \in G$, so $e \in G$ is the identity element provides $x * e = e * x = x$
- Inverse: For $\forall x \in G$, so $y \in G$ is the inverse of x provides $x * y = y * x = e$

If the * operation is abelian then the group is called an *abelian group*.

Let (+) and (×) are two binary operations defined over the set F. If the following axioms are satisfied then F is called a *finite field.*

- F forms an abelian group with (+).
- $F \setminus \{0\}$, forms an abelian group with (×).
- Distributive: For $\forall x, y, z \in F$, so
  $x \times (y + z) = (x \times y) + (x \times z)$
  $(x + y) \times z = (x \times z) + (y \times z)$

The *order* of the finite field is the number of elements in the field.

$GF(2^m)$ finite field is the binary finite field where can be observed m-dimension vector space over $GF(2)$. So, an element a of the finite field is represented as m-bit binary vector, $(a_{m-1}a_{m-2}. . . . a_1a_0)$ where $a_i \in \{0, 1\}$. It is commonly used bases in $GF(2^m)$ are *polynomial* and *normal basis*.

In polynomial basis representation, each finite field element corresponds polynomials whose degree is less than m with binary coefficients.

$$a = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + . . . + a_1x + a_0, \quad a_i \in \{0, 1\}$$

$a \in GF(2^m)$ is always denoted as $(a_{m-1}a_{m-2}. . . . a_1a_0)$.

Let $f(x) = x^m + \sum_{i=0}^{m-1} f_i.x^i$, $f_i \in \{0, 1\}$ and $i = 0, 1, 2, . . . , m - 1$, is defined in $GF(2)$ with degree m. If $f(x)$ can not written as two other polynomials product in the same field, then it is *irreducible* and called *reduction polynomial*.

Let $a = (a_{m-1}a_{m-2}. . . . a_1a_0)$, $b = (b_{m-1}b_{m-2}. . . . b_1b_0) \in GF(2^m)$ and $f(x)$ be reduction polynomial, then the field operations are:

- Addition: $a + b = c = (c_{m-1}c_{m-2}. . . . c_1c_0)$, $c_i = a_i + b_i \pmod 2$. That is, addition in $GF(2^m)$ using polynomial basis is bitwise EXOR.
- Multiplication: $a \times b = c = (c_{m-1}c_{m-2}. . . . c_1c_0)$, $c(x) = \sum_{i=0}^{m-1} c_i.x^i$, is the remainder of product $(\sum_{i=0}^{m-1} a_i.x^i)(\sum_{i=0}^{m-1} b_i.x^i)$ divided by $f(x)$.
- Inverse: Inverse of an element can be computed using Extented Euclid Algorithm.

The normal basis defined on $GF(2^m)$ is $\{\beta, \beta^2, \beta^4, \ldots, \beta^{2^{m-1}}\}$ where $\beta \in GF(2^m)$. $\forall\, a \in GF(2^m)$ is represented as,

$$a = \sum_{i=0}^{m-1} a_i.\beta^{2^i},\ a_i \in \{0, 1\}$$

$a \in GF(2^m)$ is always denoted as $(a_{m-1}a_{m-2}. \ldots a_1a_0)$. Squaring an element of $GF(2^m)$ with normal basis representation is only a rotate operation of binary vector. So, it can implement in hardware easily. Although squaring is very simple, multiplication is a bit hard in normal basis representation. But, for a group of normal bases, *Gaussian Normal Bases (GNB)*, multiplication can be realised efficiently. *Type T* of a GNB is a positive number that is the complexity indicator of multiplication. The value of T increases, then the complexity of multiplication increases. Type I and type II GNBs are called *optimal normal bases*.

Let $a = (a_{m-1}a_{m-2}. \ldots a_1a_0)$, $b = (b_{m-1}b_{m-2}. \ldots b_1b_0) \in GF(2^m)$ then the field operations in a type *T* GNB are:

- Addition: $a + b = c = (c_{m-1}c_{m-2}. \ldots c_1c_0)$, $c_i = a_i + b_i$ (mod 2). That is, addition in $GF(2^m)$ using normal basis is bitwise EXOR.

- Square: $a^2 = (\sum_{i=0}^{m-1} a_i \beta^{2^i})^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1 \bmod m} \beta^{2^i} = (a_{m-2}a_{m-3}. \ldots a_1a_0a_{m-1})$

  Squaring a finite field element corresponds a cyclic shift in vector representation.

- Multiplication: Let $p = Tm + 1$ and $u \in GF(p)$ and the order of u mod p is T. $F(1),F(2),. \ldots .$, $F(p - 1)$ sequence is defined as follows:[2]

  $$F(2^i u^j \bmod p) = i,\ 0 \le i \le m - 1,\ 0 \le j \le T\text{-}1$$

  For $\forall\, l,\ 0 \le l \le m - 1$, A_l and B_l is defined as follows:

  $$A_l = \sum_{k=1}^{p-2} a_{F(k+1)+l} b_{F(p-k)+l}$$

  and

  $$B_l = \sum_{k=1}^{m/2} (a_{k+l-1} b_{m/2+k+l-1} + a_{m/2+k+l-1} b_{k+l-1}) + A_l$$

$$a \times b = c = (c_{m-1}c_{m-2}. \ldots c_1c_0),\ \forall\, l, 0 \le l \le m - 1,$$

All indices are reduced mod *m*.

$$c_l = \begin{cases} A_l, & \text{if } T \text{ is even} \\ B_l, & \text{if } T \text{ is odd} \end{cases}$$

- Inverse: For any $a \in GF(2^m)$ satisfies the equation $a^{2^m} = a$. Then, $a^{-1} = a^{2^m - 2}$ and $2^m - 2 = 2 + 2^2 + 2^3 + \ldots + 2^{m-1}$ so, a^{-1} is computed as follows:[3]

$$a^{-1} = (a^2).(a^{2^2}).(a^{2^3}).\ldots.(a^{2^{m-1}})$$

It can be computed as the sequence of operations rotation and multiplication.

III. FINITE FIELD ELLIPTIC CURVES

An elliptic curve equation is in general given as follows;

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

In figure 1 an elliptic curve in the field of real numbers is given. With the set of points (x, y) in the curve and an operation defined over this set a group can be constructed. So, this means it can be used for cryptographic applications.

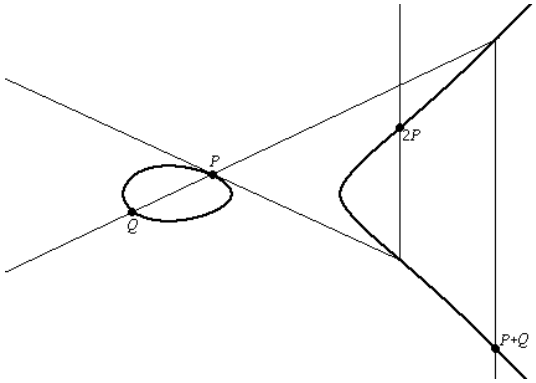

Figure 1: An Elliptic Curve Defined In Real Numbers

The operation defined on the curve points is addition. It is defined as follows:

Let P and Q are two points on the curve. The line is drawn through these two points. This line always intersects the curve at a third point [4], and then a vertical line through this point is drawn. The intersection point is the point $P + Q$. If the points are same, $P = Q$, then the line through this point is tangent of the point and known that it intersects the curve at this point twice. It is assumed the identity point of the group is the *point at infinity*, *O*, through y-axis. Elliptic curve is denoted by $E(GF(2^m))$.

It is hard to apply group operation graphically. But, using the coordinates of P and Q points, $P=(x_1, y_1) \in E(GF(2^m))$ and $Q = (x_2,\ y_2) \in E(GF(2^m))$, $P + Q = (x_3, y_3)$ can obtain by *Weierstrass equations*.[8]

The elliptic curve equation in $GF(2^m)$ is:

$$y^2 + xy = x^3 + ax^2 + b$$

where $a, b \in GF(2^m)$, and $b \neq 0$.

If $P \neq \pm Q$;

$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$; $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$;

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1}$$

If $P = Q$, $x_1 = x_2$ and $y_1 = y_2$, $P + P = 2P$, is called *point doubling*;

$$x_3 = \lambda^2 + \lambda + a;\ y_3 = \lambda(x_1 + x_3) + x_3 + y_1;\ \lambda = x_1 + \frac{y_1}{x_1}$$

If $P = -Q$, $x_1 = x_2$ and $y_1 = x_2 + y_2$, $P - P = O$, Q is the *negative* of P.

Addition of two distinct points in $GF(2^m)$ requires one inverse, two multiplications, one square and eight additions and similarly point doubling requires one inverse, two multiplications, one square and six additions. Inversing a field element is more expensive than the other operations and considering the number of point addition or point doubling in a cryptographic application it is more efficient to use *projective coordinates* where the inverse operation is eliminated. Only one inversion is needed for returning affine coordinates.

$\forall P$ = (X, Y, Z) point in projective coordinates corresponds $x = X / Z$, $y = Y / Z$ in affine coordinates. So, projective plane is the union of (x, y) points in affine plane, which can be represented as $(x, y, 1)$ in projective coordinates and the points where $Z = 0$.[5] The elliptic curve equation in $GF(2^m)$ can be modified as follows after $x = X / Z$, $y = Y / Z$ conversion.

$$y^2z + xyz = x^3 + ax^2z + bz^3$$

The O point is (0,1,0) in projective coordinates. The Weierstrass equations can be reformulated for $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$ normalising $(x_1 / z_1, y_1 / z_1, 1)$ and $(x_2 / z_2, y_2 / z_2, 1)$ respectively.

Let $P = (x_1, y_1, z_1)$, $Q = (x_2, y_2, z_2)$, $P + Q = (x_3, y_3, z_3)$, P and $Q \neq O$ and $P \neq -Q$
If $P \neq -Q$;
$x_3 = AD$; $y_3 = CD + A^2(Bx_2 + Ay_2)$; $z_3 = A^3z_1z_2$
$A = x_2z_1 + x_1z_2$; $B = y_2z_1 + y_1z_2$; $C = A + B$;
$D = A^2(A + az_1z_2) + z_1z_2BC$
If $P = Q$;
$x_3 = AB$; $y_3 = x_1^4 A + B(x_1^2 + y_1z_1 + A)$; $z_3 = A^3$
$A = x_1z_1$ ve $B = b z_1^4 + x_1^4$

Addition of two distinct points in $GF(2^m)$ using projective coordinates requires thirteen multiplications[1], one square and seven additions and similarly point doubling requires seven multiplications, four squares and four additions. Inversion is eliminated, only one inversion is needed for returning affine coordinates when kP product is computed. Besides standard projective coordinates, there are other variants of conversions used.

Let k be an integer and P is an elliptic curve point. *Elliptic scalar kP product* is k times addition of point P. This is the most important block used in elliptic curve cryptography. The *order* of point P is the smallest positive integer r satisfied $rP = O$.

P is the curve point and $1 \leq k \leq \text{order}(P)$ then the elliptic curve scalar multiplication computations are formed as $Q = kP = \underbrace{P + P + \ldots + P}_{k \text{ times}}$

Binary Method: It is the oldest and the simplest method based the binary representation of k. If $k = \sum_{j=0}^{l-1} k_j 2^j$, $k_j \in \{0, 1\}$, then kP can be computed as follows:

$$kP = \sum_{j=0}^{l-1} k_j 2^j P = 2(\ldots 2(2k_{l-1}P + k_{l-2}P) + \ldots) + k_0P$$

This method requires l times point doubling and w_k - 1 times addition. w_k is the number of non-zero bits in binary representation of k, called *weight* of k.

This method can be optimised as follows:

Each integer k can be represented as $k = \sum_{j=0}^{l-1} k_j 2^j$, $k_j \in \{-1, 0, 1\}$. In this unique representation there is no consecutive non-zero digits and called *non-adjacent form, NAF*.[1]

An l-bit number has weight approximately $l/3$ in NAF representation. Because of the negative of point $P = (x, y)$ is $-P = (x, -y)$ or $-P = (x, x + y)$, point addition and point subtraction costs same.

So, kP product can be computed as binary method using the NAF representation of k instead of performing addition or subtraction according to the sign for non-zero digits.

Considering the memory storage window methods can be used for accelerating scalar multiplication where the process for a block of digits of k is performed at each step.

IV. SECURITY of ECC

The operation defined on elliptic curves is addition of curve points. *Discrete Logarithm Problem (DLP)* is the problem to find logarithms of numbers defined on a G group. Because the points of elliptic curves and the operation addition forms an abelian group, elliptic curves can be used in cryptography. Similarly as DLP, *Elliptic*

[1] z_1 or z_2 is assumed one.

Curve Discrete Logarithm Problem (ECDLP) is the problem for known P and Q points to find k, where $kP = Q$. ECDLP is more difficult to apply than DLP. Because, strongest algorithms that solved DLP can not be adapted to solve ECDLP. So, smaller parameters with ECC provide equal security as RSA cryptosystems. Firstly, the hardness of ECDLP depends on the *base point, P*. The order of the base point P should be high as possible. This allows more points on the curve useable. Also, because Pohlig and Hellman reduced ECDLP to the prime factors of the order of base point, it should be prime. Secondly, the number of points on the curve effects the security. It should be divisible by a high prime to strength security. The problem is to compute the number of points on the elliptic curve. At the first time, it can not compute strictly, mostly it was computed as boundaries of intermediate. In 1985 this was computed using Schoof's Algorithm. This polynomial time algorithm is a deterministic algorithm to compute number of curve points. While it is the best algorithm known, it is so slow when the order of group becomes higher. Thirdly, in 1991 Menezes, Okamoto and Vanstone reduced ECDLP to DLP for a group of elliptic curves, *supersingular elliptic curves.* So, the curve chosen effects the security. While the curves were chosen randomly during some attacks developed, it is began to construct the elliptic curves of desired specifications and strong to the known attacks. Ideally the curve should be constructed after designating the properties of curve, but the best strategy is using Koblitz's random choose method while the desired conditions are satisfied.[1,5]
NIST (National Institute of Standards and Technology) recommended elliptic curves are resistant all known attacks and are used in cryptosystems recently.[1]

V. A HARDWARE IMPLEMENTATION OF SCALAR MULTIPLICATION BLOCK AND APPLICATION

While an elliptic curve application is implementing, some choices should be chosen due to the requirements of the system. Some of these are security of the system, finite field arithmetic methods, elliptic scalar multiplication methods, application platform, processor limits and communication limits.

Firstly, the finite field should be chosen. $GF(q)$ prime finite field is appropriate for software implementations. Then the finite field was chosen as binary finite field, $GF(2^m)$. Secondly, because the advantages for hardware implementations, the normal basis representation was used for finite field elements. The *extension degree* of the field, m was chosen 163. Because it is long enough to provide sufficient level of security and it is short to implement easily. Projective coordinates were used when realising elliptic curve arithmetic. So, the inversion was eliminated at each step of scalar multiplication. Inversion was performed only once to convert projective coordinates to affine coordinates. The NIST recommended elliptic curve and base point for GF(2^{163}) were used.

The elliptic curve scalar multiplication block was designed and described in VHDL. The diagram of the block is given in the following.

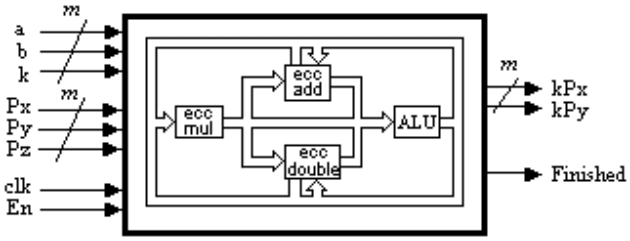

Figure 2: The Diagram of Scalar Multiplication Block

Firstly, an ALU (Arithmetic Logic Unit) was designed for implementing finite field arithmetic when normal basis representation is used. ecc_add block gets the elliptic curve points as input and gives the addition of two points as output. Similarly, ecc_double block was used to realise point doubling. ecc_mul block was the control block for scalar multiplication. It controls whether point addition or point doubling is needed due to the bits of k. The block converts affine coordinates in its input to projective coordinates[2].

The most important block used in elliptic curve cryptography is the scalar multiplication block. After designing this block, simple elliptic curve cryptography protocols may implement in an integrated circuit. At this time, message representation is important in the application. There are two methods for message representation, *imbedding* and *masking.* Imbedding is converting the message into curve points. So, there should be fast algorithms to convert message to point and point to message. The message space is restricted to the number of curve points. Masking is the method to represent message as finite field elements. Masking does not need to convert message, therefore it is applicable directly. Altogether, the securities of these two methods depend on ECDLP, their securities may be assumed equal. A simple Elliptic Curve El Gamal Protocol needs the steps below. Because of its easiness masking was used when implementing this protocol.

Let the receiver B parameters are (q, FR, a, b, P, n, h) where q represents the finite field, FR represents the method used for field element representation, a and b are elliptic curve equation coefficients, P is the base point and n is the order of base point. h is the number where n times h results the number of points in the elliptic curve. Q_B is the public key and d_B is the secret key of B. The sender A has authentic copies of the receiver's parameters and the

[2] Pz input is (111....111) as one when using normal basis representation.

public key Q_B. The message m is represented as a pair of field elements, $(M_1, M_2) \in GF(2^m)$.

The sender A does the following to encrypt the message m.[11]

1. Random integer k is selected in the interval $[1,n - 1]$.
2. $kQ_B = (\overline{x}, \overline{y})$ is computed, $\overline{x} \neq 0$ and $\overline{y} \neq 0$ is checked.
3. $kP = (x, y)$ is computed.
4. $M_1 \overline{x}$ and $M_2 \overline{y}$ is computed.
5. $kP = (x, y)$, $M_1 \overline{x}$ and $M_2 \overline{y}$ is sent to the receiver B.

The receiver B does the following to decrypt the cipher $(kP,(M_1 \overline{x},M_2 \overline{y}))$.

6. $d_B(kP) = kQ_B = (\overline{x}, \overline{y})$ is computed.
7. $M_1 = M_1 \overline{x} (\overline{x})^{-1}$ and $M_2 = M_2 \overline{y} \ (\overline{y})^{-1}$ is computed.

In this protocol, if M_1 is known then M_2 or if M_2 is known then M_1 can find easily. Only $(kP, M_1 \overline{x})$ may send to prevent this.

Also, four field elements are sent for two field elements. This is called *message expansion*. The message expansion factor is 2 in this protocol. This factor can be reduced to 3/2 by sending only the coordinate x and 1 bit for the y coordinate instead of $kP = (x, y)$. Receiver can recover y by using the elliptic curve equation.

Because of the higher number of inputs and outputs, communication with computer is performed by RS232 protocol. Because the elliptic curve parameters a and b are constant in the application and considering only the scalar key and the elliptic curve point are altered, the k input and the (x, y) coordinate are loaded at each elliptic scalar multiplication step over the serial port.

VI. CONCLUSION

Elliptic curve cryptography provides equal security level with smaller parameters to the other known public key crytosystems. So, when higher process speeds, power dissipation and lower storage are needed, elliptic curve cryptography may be chosen. Also, its security could not be proven exactly like other public key cryptosystems and the level is designated as attacks not discovered. Because these attacks appear a bit time later, it is important to develop another alternative.[4]

In this study, the finite field $GF(2^m)$ and most popular bases used in this field are examined. Elliptic curves for crtyptographic usage of this field and security issues of elliptic curve cryptography are given. Also, scalar multiplication block which is the most important and the most time consuming block used in elliptic curve protocols is designed and this block is described with VHDL[3]. The block designed is programmed with FPGA and a simple El Gamal elliptic curve protocol application is implemented.

REFERENCES

[1] Lopez, J. and Dahab, R., 2000. An Overview of Elliptic Curve Cryptography, *Institute of Computing,* State University of Campinas, Brazil, May 22

[2] P1363, 2000. Standard Specifications for Public Key Cryptography, *IEEE*, New York

[3] Wang, C.C., Truong, T.K., Shao, H.M., Deutsch, L.J., Omura, J.K. and Reed, I.S., 1985. VLSI Architectures for Computing Multiplications and Inverses in GF(2^m), *IEEE Transactions on Computers*, 34, 709-717

[4] De Win, Erik and Preneel, B., 1997. Elliptic Curve Public Key Cryptosystems – an introduction, *State of the Art in Applied Cryptography*, 131-141

[5] Mugino, S., 1997. Elliptic Curve Cryptosystems, *Msc. Thesis,* McGill University, Montreal

[6] Jurisic, A. and Menezes, A.J., 1997. Elliptic Curves and Cryptography, *Dr. Dobb's Journal*, 23-36

[7] Araki, K., Satoh, T. and Miura, S., 1998. Overview of Elliptic Curve Cryptography, *Proc. of Public Key Cryptography*, 19-48

[8] Agnew, G. B., Mullin, R.C. and Vanstone, S.A., 1993. An Implementation of Elliptic Curve Cryptosystems Over $F_{2^{155}}$, *IEEE Journal on Selected Areas in Communication*, 11, 804-813

[9] Lopez, J. and Dahab, R., 1998. Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n), *Selected Areas in Cryptography,* 201-212

[10] Hankerson, D., Hernandez, J.L. and Menezes, A., 2000. Software Implementation of Elliptic Curve Cryptography Over Binary Fields, *Cryptographic Hardware and Embedded Systems*, 1-24

[11] Menezes, A., 1993. Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, United States of America

[3] The VHDL codes can be requested from serkan@ehb.itu.edu.tr