Journal of Computer Science 6 (11): 1237-1246, 2010
ISSN 1549-3636

A Bit-Serial Multiplier Architecture for Finite Fields Over Galois Fields

[1]Hero Modares, [1]Yasser Salem, [1]Rosli Salleh and [2]Majid Talebi Shahgoli
[1]Department of Computer System and Technology, Faculty of Computer Science, University of Malaya, 50603, Kuala Lumpur, Malaysia
[2]Department of Security, University College of Technology and Innovation, Kuala Lumpur, Malaysia

Abstract: Problem statement: A fundamental building block for digital communication is the Public-key cryptography systems. Public-Key Cryptography (PKC) systems can be used to provide secure communications over insecure channels without exchanging a secret key. Implementing Public-Key cryptography systems is a challenge for most application platforms when several factors have to be considered in selecting the implementation platform. **Approach:** The most popular public-key cryptography systems nowadays are RSA and Elliptic Curve Cryptography (ECC). ECC was considered much more suitable than other public-key algorithms. It used lower power consumption, has higher performance and can be implemented on small areas that can be achieved by using ECC. There is no sub exponential-time algorithm in solving the Elliptic curve discrete logarithm problem. Therefore, it offers smaller key size with equivalent security level compared with the other public key cryptosystems. Finite fields (or Galois fields) is considered as an important mathematical theory. **Results:** Thus, it plays an important role in cryptography. As a result of their carry free arithmetic property, they are suitable to be used in hardware implementation in ECC. In cryptography the most common finite field used is binary field GF (2^m). **Conclusion:** Our design performs all basic binary polynomial operations in Galois Field (GF) using a microcode structure. It uses a bit-serial and pipeline structure for implementing GF operations. Due to its bit-serial architecture, it has a low gate count and a reduced number of I/O pins.

Key words: Public-key cryptography, elliptic curve cryptography, Galois field, scalar multiplication, elliptic curve algorithms

INTRODUCTION

Public-key cryptography and symmetric-key cryptography are two main categories of cryptography. The Well-known public-key cryptography algorithms are RSA (Rivest *et al*., 1978), El-Gamal and Elliptic Curve Cryptography. Presently, there are only three problems of public key cryptosystems that are considered to be both secured and effective. Table 1 shows these mathematical problems and the cryptosystems that rely on such problems. Table 2 shows the complexity of calculative for each of these problems where 'n' is the length of the keys used (Sandoval, 2008; Kumar, 2008).

Providing an equivalent level of security with smaller key size is an advantage of ECC compared to RSA. It is very efficient to implement ECC. ECC obtains lower power consumption and faster computation.

Table 1: Mathematical problem

Mathematical problem	Detail	Cryptosystems
Integer Factorization Problem (IFP)	Given an integer 'n', find its prime factorization	RSA
Discrete Logarithm Problem (DLS)	Given integer 'g' and 'h', find 'x' such that $h = g \times \bmod n$	ELGemal, DSA Diffie-Hellman (DH)
Elliptic Curve Discrete Logarithm Problem (ECDLP)	Given points 'P' and 'Q' on curve, find 'x' such that Q = xP	ECDSA, EC-Diffie-Hellman (DH)

Table 2: Public-key

Public-key system	Best known methods for solving mathematical problem	Runing times
Integer Factorization Problem (IFP)	Number field sieve: $e^{1.923}(\log n)^{1/3} (\log \log n)^{2/3}$	Sub-exponential
Discrete Logarithm Problem (DLS)	Number field sieve: $e^{1.923}(\log n)^{1/3} (\log \log n)^{2/3}$	Sub-exponential
Elliptic Curve Discrete Logarithm Problem (ECDLP)	Pollard-rhlgorithm: $\sqrt{n}$	Fully exponential

Corresponding Author: Hero Modares, Department of Computer System and Technology, Faculty of Computer Science, University of Malaya, 50603, Kuala Lumpur, Malaysia

It also gains small memory and bandwidth because of its key size length (De Dormale *et al*., 2004; Li *et al*., 2008). Such attributes are mainly fascinating in security applications in which calculative power and integrated circuit space are limited.

A modular arithmetic performs a main role in public key cryptographic systems (De Dormale *et al*., 2004). Some of these PKC are the Diffie-Hellman keys exchange algorithm (Kaabneh and Al-Bdour, 2005), the decipherment operation in the RSA algorithm (Quisquater and Couvreur, 1982), the US Government Digital Signature Standard (Kammer and Daley, 2000) and also elliptic curve cryptography (Koblitz, 1987).

Arithmetic in elliptic curves requires a number of modules to calculate ECC operations (modular multiplication, modular division and modular addition/subtraction operations) (Al-Somani *et al*., 2006). The division modular is one of the most critical operations, which is expensive and computationally extensive. Many implementations are completed using projective coordinates in order to represent the points on the curve by reducing inversion/division to one. However, a final division is still needed to convert the projective coordinates into affine coordinates. In some other cases, modular division can be replaced by modular inversion followed by modular multiplication.

In this field, modular multiplication gets much attention and numerous algorithms have been published. The modular inversion can be performed using Fermat little theorem or the well-known extended Euclidian algorithm and Montgomery inverse as well (Kaliski, 1995; Savas and Koc, 2000).

In this research, the Double-and-Add alternative is used in our system as it is mainly necessary for our algorithms. Galois Field is a finite field that consists of a finite number of elements. It contains three operations which are Addition, Multiplication and division modular's. Galois Field Modular division is replaced by modular inverse followed by modular multiplication. Montgomery modular inversion method is chosen as an inversion algorithm and Right-to-left shift method as a multiplication algorithm.

Most of the hardware implementations of ECC are based on bit-parallel but in this study bit-serial architecture is used. Bit-serial operators are noticeably smaller than those operators in bit-parallel. They do not depend on word width. A multiplier is said to be bit-serial if it produces only one bit of the product at each clock cycle. Moreover, bit-serial architectures only demand an equally small amount of input and output pins. An implemented multiplication in every bit-serial type has to be directly fitted to the data-width. As a result, the area complexity is reduced to O(n) and parallel multiplier $O(n^2)$.

Bit-serial design makes it compulsory to operate with particular registers. These registers are able to store one bit for the period of a clock cycle. After this cycle the information is passed on to the output and the next information can enter the register. These registers offer the core functionality of shifting numbers.

Mathematical of elliptic curve cryptography: There are many ways to calculate the points over the prime field elliptic curve. A direct method is by applying the next equation:

$$y^2 = x^3+ax+b \text{ where } 4a^3+27b^2 \neq 0$$

Different elliptic curve is produced by changing the values of 'a' and 'b'. In elliptic curve cryptography, calculating the public-key can be done by multiplying the private key with the generator point 'G' in the curve. The generator point 'G' is the point on the curve. The private key is the random number in the interval [1, n-1],'n' is the curve's order (Anoop, 2007).

The strength of ECC security comes from the difficulty of Elliptic Curve Discrete Logarithm Problem. If 'P' and 'Q' are points on the curve, then kP = Q where 'k' is a scalar. Thus, point multiplication is the basic operation in ECC. For example, the multiplication of a scalar 'k' with any point 'P' on the curve in order to obtain another point 'Q' on the curve.

Point multiplication: Scalar point multiplication is a block of all elliptic curve cryptosystems. It is an operation of the form k.P. 'P' is a point on the elliptic curve and 'k' is a positive integer. Computing k.P means adding the point 'P' exactly d-1 times to itself, which results in another point 'Q' on the elliptic curve. Point multiplication uses two basic elliptic curve operations:

- Point addition (add two point to find another point)
- Point doubling (adding point p to itself to find another point)

For example to calculate kP = Q if 'K' is 23 then kP = 23P = 2(2(2(2P) + P) + P) + P so to get the result point addition and point doubling is used repeatedly (Anoop, 2007).

Point addition: Let 'J' and 'K' be two points on the elliptic curve. To achieve another point like 'L' point addition, it is shown in the Fig. 1.

According to the Fig. 1a 'K' and 'J' are two points on the EC and K≠-J. The line through the points 'J' and 'K' intersect the elliptic curve at one more point -L.

The result of adding point 'J' to 'K' is point 'L', which is -L reflection with respect to x-axis (Anoop, 2007).

If K = -J then there is a line through the points. 'J' and 'K' intersect the elliptic curve at a point at infinity '0' because J + (-J) = 0 as shown in Fig. 1b.

To analyze point addition, let assume J = (x_j,y_j), K = (x_k,y_k), L = J+K where L = (X_l,Y_l) and s is the incline of the line through 'J' and 'K' then:

$S = (y_j-y_k)/(x_j-x_k)$
$X_l = s^2-x_j-x_k$
$Y_l = -y_j+s(x_j-x_l)$

So if K= -J then J+K = O, where 'O' is the point at infinity. And if K = J then J+K = 2J and needs to use point doubling equation.

Point doubling: Let 'J' be a point on the elliptic curve. Then addition of the point 'J' to itself return another point like 'L' on the same curve as shown in the Fig. 2.

According to the Fig. 2a, 'J' is a point on the EC and to get 'L' which is equal to 2J, the tangent line at 'J' will intersect the EC at exactly point -L only if the value of 'y' axis of the point 'J' not equal to zero. However, the result of doubling is the point 'L' the reflection of the point -L with respect to x-axis (Anoop, 2007).

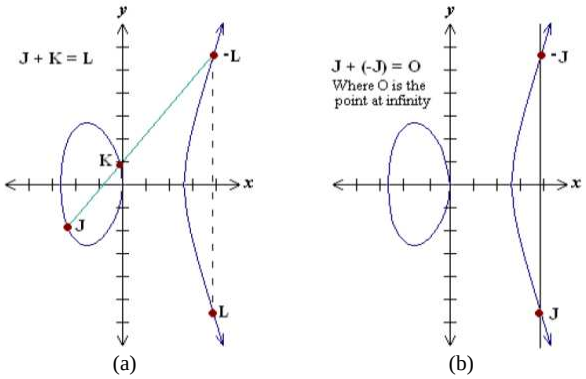

Fig. 1: Point addition

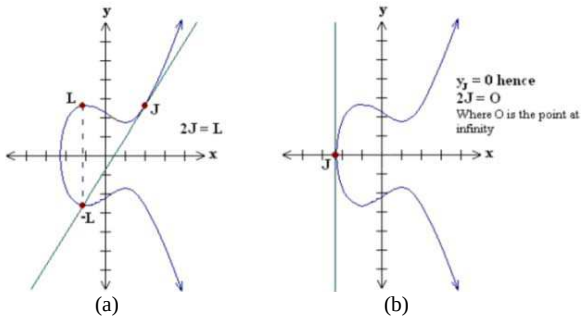

Fig. 2: Point doubling

As shown in the Fig. 2b, if 'y' coordinates of the point 'J' is zero. So the tangent at this point intersects at a point at infinity 'O' that means when $y_j = 0$, 2 J = O.

To analyze point addition let assume J= (x_j,y_j) where $y_j \neq 0$, L = 2J where L = (x_l,y_l) and 'S' is the tangent at point 'J' then:

$S = (3x_j^2+a)/(2y_j)$ ('a' is one of the parameters which is chosen by EC)
$X_l = s^2-2x_j$
$Y_l = -y_j+s(x_j-x_l)$

and 'O' is the point at infinity if $y_j = 0$ then 2J = O.

Elliptic curve domain parameters:

Domain parameters for EC over field Fp: Elliptic curve over Fp has list of domain parameters which includes 'p', 'a',' b',' G', 'n' and 'h' parameters:

'a' and 'b': Define the curve $y^2 \bmod p = x^3 + ax + b \bmod p$
'p': Prime number defined for finite field Fp
'G': Generator point (XG,YG) on the EC that selected for cryptography operations
'n': The Elliptic curve order
'h': If #E(Fp) is the number of points on an elliptic curve then 'h' is cofactor where h=#E(Fp)/n

Domain parameters for EC binary fields: Elliptic curve over F_2m has a list of domain parameters which includes 'm', f(x), 'a', 'b', 'G', 'n' and 'h' parameters:

'm': An integer to finite field F_2m
F(x): The irreducible polynomial of degree m that it used for elliptic curve operations
'a' and 'b': Define the curves $y^2 + xy = x^3 + ax^2 + b$
'G': The generator point (x_G, y_G) on the EC that selected for cryptography operations
'n': The ore elliptic curve
'h': if $\#E(F_2^m)$ is the number of points on an elliptic curve then 'h' is cofactor where $h=\#E(F_2^m)/n$

Field arithmetic: Modular arithmetic and polynomial arithmetic are two different types applied in ECC operations depending on the chosen field. In Modular arithmetic, over a number 'p' arithmetic covers the number in the interval [0 and p-1]. Modular Arithmetic contains Addition, Subtraction, Multiplication, Division, Multiplication Inverse and Finding x mod y operations. Polynomial Arithmetic contains Addition, Subtraction, Multiplication, Division, Multiplicative

Inverse and Irreducible Polynomial (Anoop, 2007). This research relies on polynomial arithmetic. Therefore, this part gives an overview of polynomial arithmetic.

EC over field F_2m includes arithmetic of integer with length m bits. The binary string can be declared as polynomial:

Binary string: $(a_{m-1} \ldots a_1 a_0)$
Polynomial: $a_{m-1} x^{m-1} + a_{m-2} x^{m-2} + \ldots + a_2 x^2 + a_1 x + a_0$

where $a_i = 0$.

For example: $x^3 + x^2 + 1$ is polynomial for a four bit number 11012.

Addition: If $A = x^3+x^2 + 1$ and $B = x^2 + x$ are two polynomial then A+B called polynomial addition that returns $x^3 + 2x^2 + x + 1$ after taking mod 2 over coefficients $A + B = x^3 + x + 1$ (Ali, 2009).

On binary representation, polynomial addition can be achieved by simple XOR of two numbers. For example, over $GF(2^4)$ there are 16 elements where $f(x)=x^4 +x+1$ as follow:

$g^0 = (0001)$	$g^1 = (0010)$	$g^2 = (0100)$	$g^3 = (1000)$
$g^4 = (0011)$	$g^5 = (0110)$	$g^6 = (1100)$	$g^7 = (1011)$
$g^8 = (0101)$	$g^9 = (1010)$	$g^{10} = (0111)$	$g^{12} = (1110)$
$g^{12} = (1111)$	$g^{13} = (1101)$	$g^{14} = (1001)$	$g^{15} = (0001)$

$$\text{So if :} \left| \begin{matrix} A = 1101_2 \\ B = 0110_2 \end{matrix} \right\} A + B = A \text{XOR} B \rightarrow A + B = 1011_2$$

Subtraction: If $A = x^3+x^2+1$ and $B = x^2+x$ are two polynomials, then A-B is called polynomial subtraction that returns x^3-x+1 after taking mod 2 over coefficients $A-B = x^3+x+1$.

On binary representation, polynomial addition can be achieved by a simple XOR of two numbers same as Addition operation in f_{2M}:

$$\left. \begin{matrix} A = 1101_2 \\ B = 0110_2 \end{matrix} \right\} A - B = A \text{ XOR } B \rightarrow A + B = 1011_2$$

Multiplication: If $A = x^3+x^2+1$ and $B = x^2+x$ are two polynomials then A-B is called polynomial multiplication that returns $x^5+x^3+x^2+x$, m = 4. The result should be reduced to a degree less than 4 by irreducible polynomial x^4+x+1:

$x^5+x^3+x^2+x \pmod{f(x)} = (x^4+x+1)x+x^5+x^3+x^2+x = 2x^5+x^3+2x^2+2x = x3$(after reducing the coefficient on mod 2)

$A = 1101_2$
$B = 0110_2$
$A*B = 1000_2$

Division: $a * b^{-1} \pmod{f(x)}$ has the same result of a/b (mod f(x)).So in order to find a/b(mod f(x)), a * b-1 (mod f(x)) can be used. Instead of it, b-1 is the multiplicative inverse of 'b':

0(0000)	1(0001)	x(0010)
X+1(0011)	x^2(0100)	x^2+1(0101)
X^2+x(0110)	x^2+x+1(0111)	x^2(0100)
X^2+1(1001)	x^2+x(1010)	x^2+x+1(1011)
X^2+x^2(1100)	x^3+x^2+1(1101)	x^3+x^2+x(1110)
X^3+x^2+x+1(1111)		

Multiplicative inversion: There are 10 powers for g where the element g = (0010) is a generator.

The multiplicative identity $g^0 = (0001)$ and the multiplicative inverse for $g^9 = (1010)$ is:

$$g^9 = (1010) \text{ is } g^{-9} \bmod 15 = g^6 \bmod 15$$

To assure that multiplicative inverse of g^9 their multiplicative result should be one:

Proofing that $g^6 \times g^9 = 1 \bmod f(x)$
$g^6.g^9 = (1010).(1100)$
$(x^3+x).(x^3+x^2) \bmod f(x)$
$(x^6+x^5+x^4+x^3) \bmod f(x) = 1$(which is the multiplicative identity)
This means $g^{-9}=g^6 \bmod f(x)$

MATERIALS AND METHODS

Elliptic curve algorithms: According to the hierarchy of Elliptic curve, three operations are needed by the ECC operations which are (addition, multiplication and inversion). Addition operation in binary field is an XOR operation. This part describes the basic arithmetic modular multiplication and inversion which are used in elliptic curve over Galois Fields.

Right-to-left shift-and-add field multiplication in F_2^m: The shift-and-add for field multiplication is based on the:

$X(z).y(z) = x_{m-1} z^{m-1} y(z)+\ldots x_2 z^2 y(z)+x_1 z y(z)+x_0 y(z)$

Repetition 'i' in the algorithm 1 compute $z^i y(x)$ mod f(z) and if x_i=1 the result will be add accumulator c'. if $y(z)= y_{m-1} z^{m-1}+\ldots y_2 z^2+y^1 z+y_0$ then:

$y(z).z = y_{m-1} z^{m-1}+y_{m-2} z^{m-1}+\ldots y_2 z^3+y_1 z^2+y_0 z$
$y(z).z = y_{m-1} r(z)+(y_{-2} z^{m-1}+\ldots y_2 z^3+y_1 z^2+y_0 z)$ (mod f(z))

So y(z).z mod f(z) can be calculated by a left-shift of the vector representation of y(z), followed by addition of r(z) to y(z) if the high order bit y_{m-1} is 1.

Algorithm 1: Right-to-left shift-and-add field multiplication in F_2m:

Input: Binary polynomials x(z) and y(z) of degree at most m-1
Output: c(z)=x(z).y(z) mod f(z)
1. If x_0 = 1 then ← y else c←0
2. For i from 1 to m-1 do
   2.1y←y.z mod f(z)
   2.2 if a_i = 1 then c←c+y
3. Return (c)

x' vector shift in hardware can be performed in one clock cycle, by making Right-to-left shift-and-add field multiplication algorithm that is suitable for the hardware.

Montgomery modular inverse algorithm: Kaliski (1995) introduced the Montgomery modular inverse. It definite as the Montgomery demonstration of the modular inverse, A^{-1} (mod P) in which 'm' is the bit-length of 'P'. Montgomery Modular Inverse Algorithm is based on the extended binary GCD algorithm. This algorithm contains two phases as shown in algorithm 4.8. The result of the second phase can be obtained either by iterative half modulo 'P' or multiplication modulo 'P'. At the end of the loop, the values of g_1 = 1and g_2 = 0 allow to check $G = -A^{-1} 2^i$(mod P) which then bring the result back in the range [1, P-1]. The following algorithm rewrites the Kaliski Montgomery inverse algorithm with combination of the two phases in one algorithm.

Algorithm 2: Montgomery modular inverse algorithm:

Input: A and P, where a $\in$GF (2^m) and P is the modulus P.
Output: G, where $G \equiv A^{-1}$ (mod P)
1. Set U = P, V = A, G = 0 and K =1
2. Set i = 0, where I is an integer with m≤i<2m
3. While V >0 do
   3.1: If U is even, then set $U = \frac{U}{2}, K = 2K.$
   3.2: Else if V is even, then set $V = \frac{V}{2}, G = 2G.$
   3.3: Else if U<V, then set $U = \frac{U-V}{2}, G = G+K, K = 2K.$
   3.4: Else if V≥U, then set $V = \frac{U-V}{2}, K = G+K, G = 2G.$
4. For j from 1 to I do
   4.1: If G is even, then set $G = \frac{G}{2}.$
   4.2: Else set $G = \frac{G+P}{2}.$
5. If G≥P, then set G = 2P-G, else set G = P-G.
6. Output G.

Modular multiplication: Finite field multiplier over F_2 m always plays a major role in determining the performance of hardware accelerators of cryptography applications. It is necessary to design the multipliers with high efficiency. Bit-parallel and Bit-serial are two options to design a modular. Bit-parallel multiplier can get high operation speed by completing one multiplication in one clock cycle. On the other hand, it has maximum circuit complexity in which a large operand size makes it unsuitable. Bit-serial operators are noticeably smaller than those operators in bit-parallel. They are independent of word width.

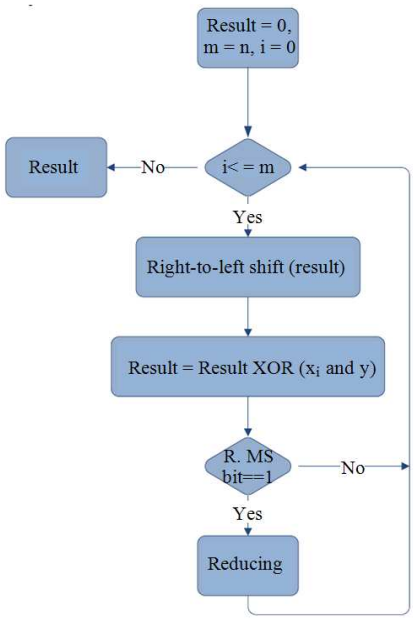

Fig. 3: Flowchart for right-to-left algorithm

A multiplier is said to be bit-serial if it produces only one bit of the product at each clock cycle. Demanding only on an equal small amount of input and output pins is an advantage of bit-serial. An implemented multiplication in every bit-serial type has to be directly fitted to the data-width. As a result, the area of complexity is reduced to O(n) than in parallel multiplier $O(n^2)$. Based on the bit-serial advantages, bit-serial is chosen for our design. This research is continued with the efficient designs for polynomial multiplier operation. Right-to-left algorithm is introduces based on the bit-serial architecture (Fig. 3).

RESULTS AND DISCUSSION

Bit-serial multiplier structure: The following flowchart shows the bit-serial multiplier steps. When the multiplier bits shifted, the result is stored in 'R'. When R(n) is a 1, it indicates that the recent partial result overflows the n-bit register. It also reduces one copy of the irreducible polynomial. The reduction is XOR operation. It completes the overall "modulo an irreducible polynomial" correction operation (Modares, 2009).

Figure 4 is depicts a multiplier 'X' and a multiplicand 'Y' when $X,Y \in F_2^m$. It processes the bits of 'x' from left to right. The multiplier is called a Most Significant Bit (MSB) multiplier. The MSB multiplier can present a multiplication in F_2m in 'm' clock cycles.

Table 3 via $f4(x) = x^4 + x +$ for two different 'x' and 'y' as input summarizes the behavior where $\wedge$ symbolize "AND" gate and $\oplus$ symbolized an XOR gate.

It is also for 79 bits, our bit- serial multiplier needs 79 ANDs, 79 XORs and less than 400 FFs (Flip-Flops). A 79 bits multiplication is computed within 79 clocks, which is not including data input and output. In this implementation, control and memory access overheads go to a total time of execution less than 280 clocks. It starts from the processor by sending memory addresses of 'X', 'Y' and 'R' to the last result which is stored in 'R'. The multiplier is also used for squaring by loading X = Y.

The first partial result is 1101, which shift left (zero fill), then is XORed with 1101. It returns the value 11010, which goes over the 4-bit register limit. It and needs reduction using XOR with the "irreducible polynomial" which is appeared by 10011 in binary for $f4(x) = x^4 + x + 1$. So, in this stage the most significant bit is zeroed and 1001 is as an adjusted result. Yet, it needs to be shifted to left and it returns 10010 as a second line result which XORed to $0 \wedge (1101)$ in third line. As shown in Table 3, MS Bit is one. The Final Result is again 5 bits again. The partial result makes the same situation. In order to get the bit alignment, another reduction (XOR with 10011) is needed. The result of shifting in the third step is 0010. It is used in the last step. As shown in the last step, there is no overflow and the exact 4-bit is a final result.

The following block diagram shows a basic multiplier structure which gets the multiplier data serially. 'X' participates as multiplier and resides in n-bit shift register 'Y' participates as multiplicand in n-bit register. 'R' is used to show the result and 'P' is put as an irreducible polynomial which is used when any overflow happens. External data input connections and Clocks are not shown in Fig. 5 noticed that the polynomial does not need to represent the leftmost bit of the polynomial.

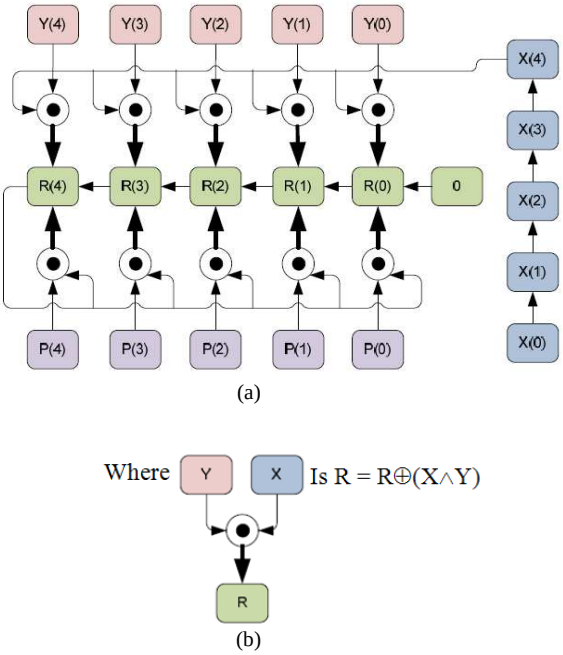

Fig. 4: Most Significant Bit first (MSB) multiplier for F_2m

Table 3: XOR (⊕) and (^)

Result XOR (x_i AND y)	Initial	Result	Check MS bit	Reduction	Shift result
Result XOR (x_1 AND y)	0000⊕1∧(1101)	1101	0	-	11010
Result XOR (x_2 AND y)	11010⊕0∧(1101)	11010	1	11010⊕10011=1001	10010
Result XOR (x_3 AND y)	10010⊕0∧(1101)	10010	1	10010⊕10011=0001	0010
Result XOR (x_4 AND y)	0010⊕1∧(1101)	1111	0	-	1111

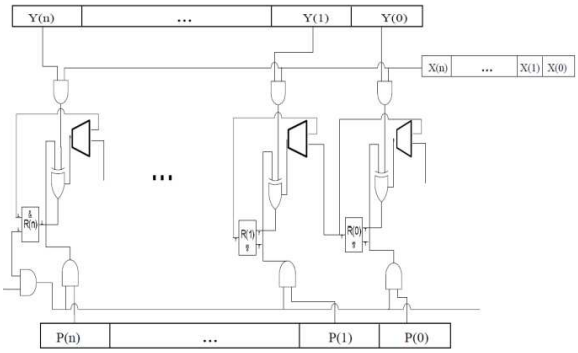

Fig. 5: The n-bit multiplier

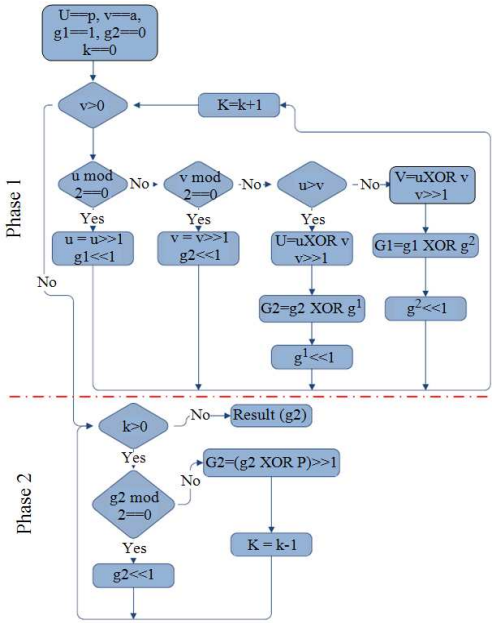

Fig. 6: Flowchart for Montgomery inversion algorithm, contain phase 1 and 2

Modular inversion: Inversion is the most difficult finite field operation to be implemented in hardware. The division in GF (2^m) x/y is implemented as two sequential operations, which are the inversion y^{-1} and then the multiplication xy^{-1}. Based on the Algorithm 2 Fig. 6 summarizes the Montgomery inversion algorithm that has been chosen to compute inversion in polynomial representations.

Based on this structure, the diagram will never reach n+1 bit length. It deals with the correction subtraction. By observing the 'n' bit result with the leftmost bit equals to 1, the multiplication mode is set. It can be done by reducing the current result registered by the irreducible polynomials.

Bit-serial inversion structure: We already discussed about the original Montgomery modular inverse. For the given advantages, the bit-parallel design is modified into a bit-serial structure. The modified structure is presented which is based on bit-serial architecture for the most computationally inversion algorithm over Galois field.

The previous block diagram shows the inversion architecture which loads the inputs serially. All the next steps are following algorithm 2 including the variables (U, V, K and G) to get the inversion result. If 'U' is even, then the new value of 'U' and 'K' is cleared. In order to use bit-serial, it depends on the Less Significant Bit (LSB). Then 'U' will be known either even or not. If LSB is '0', it means 'U' is even, then, it needs one shift to the right to find U = U/2 and one shift to left 'K' to find K = 2*K. There is the same row for find 'V' and 'G'.

In the architecture block diagram, there are a Multiplexer and a De-multiplexer which are controlled by using selector. They are used to pass the correct result to the output. In the De-multiplexer, if 'C0' and 'C3' are equals to '0', the output is placed in 'U'. Otherwise the value is placed in 'V'.

If (U<V) then (U+V)/2 is the exact result for 'V'. Otherwise it appears as a result for 'U'. The reminding steps are excluded from this dialog in order to not complicate the diagram.

Caused by its architecture which is based on bit-serial, it has low power consumption, regular structure, low cost in term of area occupied and a reduced number of pins. Thus, it is appropriate for embedded applications.

There are two phases depicted in Fig. 7. Both phases require shift left and shift right registers. When shifting to right, the number needs to be divided by two. When shifting to the left, the number needs to be multiplied by two.

In this research, this aim is achieved by using shift-left and shift-right which work in serial-in parallel-out. The two following algorithms are used.

Algorithm 3-shift_Right:

Input: C, SI [0, m].
Output: PO [0, m].
Variable tmp[m:0], PO [m:0].
1: If (posedge C)
Begin
2: Tmp = {1'b0,SI[m:1]}
End
3: PO = tmp;
End

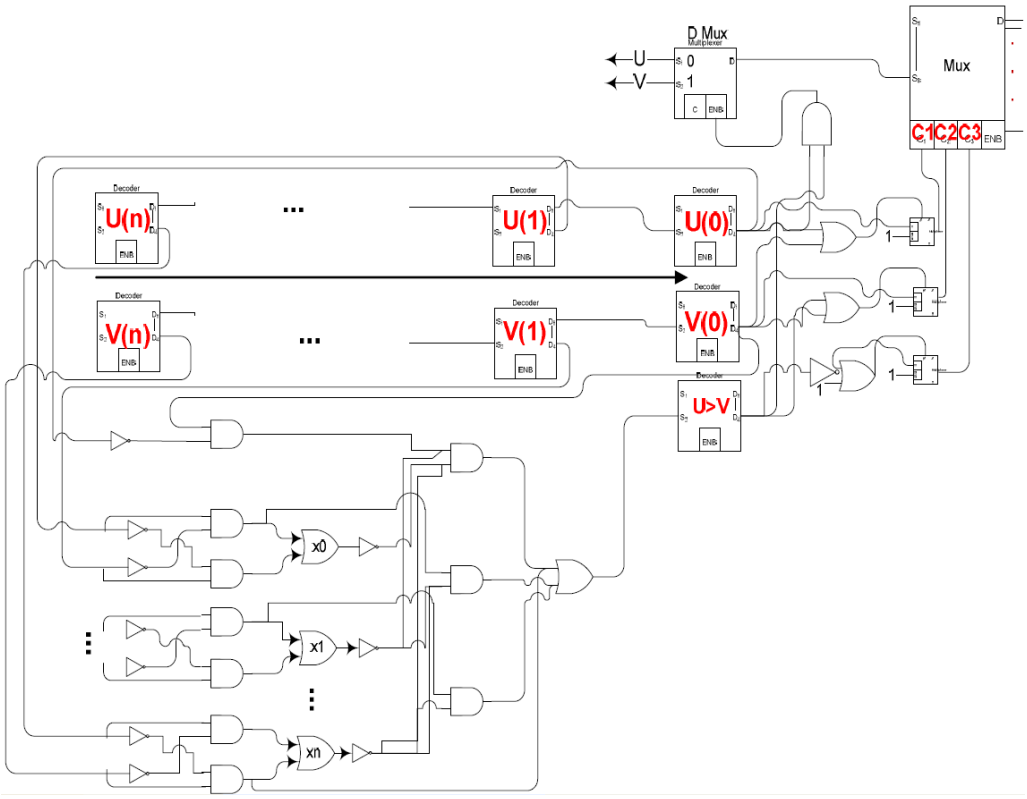

Fig. 7: Bit-serial inversion block diagram

Algorithm 4-Shift_Left

Input: C, SI [0, m].
Output: PO [0, m].
Variable tmp[m:0], PO [m:0].
1: If (posedge C)
Begin
2: Tmp = {SI[m:1,1'b0]}
End
3: PO = tmp;
End

Scalar multiplication: The most important arithmetic on elliptic curve applications is the scalar multiplication that computes 'dP'. 'd' is an arbitrary integer and 'P' is a point on elliptic curve. The scalar multiplication 'dP' can simply be clear by adding the (d-1) copies of 'P' to itself.

If the bit check starts from left to right, it is called Most Significant Bit (MSB) or double-and-add. However, if it starts from right to left, it uses Less Significant Bit (LSB) to obtain scalar multiplication.

MSB (double-and-add): To calculate MSB, two parameters are required (d,P) to get the resulted point Q. The next steps are used to calculate MSB.

Require d= (di-1, di-2… d0)2, di=1 //'d' is Input
Compute Q=dP
Q=P
For j=i-2 to 0
Q=2Q // it start with Doubling
If dj=1 then
Q=Q+P // Addition
End if
End for
Return Q

Scalar multiplication: LSB first: As in MSB, LSB needs the same input arguments (d,P) to calculate the point Q. The next steps are used to calculate LSB.

Require d= (di-1, di-2… d0)2, di=1//'d' is Input
Compute Q=dP
Q=0, R=P
For j=0 to i-1
If ki=1 then
Q=Q+R
End if
R=2R
End for
Return Q

Table 4: Scalar multiplication

Addition (where $x_1<>x_2$) $(x_3,y_3) = (x_1,y_1)+(x_2,y_2)$	Doubling (where $x_1<>0$) $(x_3, y_3) = (x_1,y_1)+(x_1,y_1)$
GF(p)	
$S = (y_2-y_1)/(x_2-x_1)$	$S = (3(x_1)2+a)/(2y_1)$
$X_3 = S_2-x_1-x_2$	$X_3 = S-2x_1$
$Y_3 = S(x_1-x_3)-y_1$	$Y_3 = S(x_1-x_3)-y_1$
GF(2^m)	
$S = (y_2+y_1)/(x_2+x_1)$	$S = x_1+(y_1)/(x_1)$
$X_3 = S_2+S+x_1+x_2+a$	$X_3 = S_2+S+a$
$Y_3 = S(x_1+x_3)+y_1+x_3$	$Y_3 = (x_1)2+(S+_1)x_3$

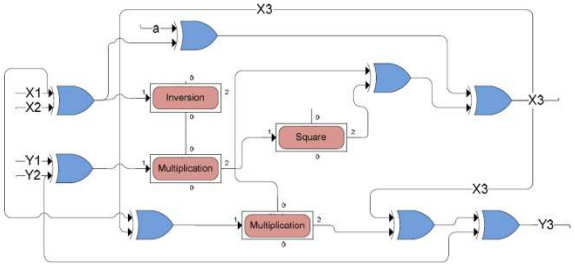

Fig. 8: Point addition in GF(2^m)

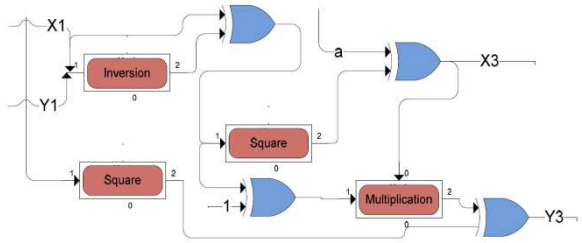

Fig. 9: Point doubling in GF(2^m)

There are many advanced researches on modular arithmetic operations over finite fields. For example, Right-to-left shift field multiplication in F_{2^m} and Montgomery inversion method. These algorithms are used in our bit-serial hardware architecture to calculate scalar multiplication.

Double-and-add algorithm is chosen as it is dictated in ANSI (Greenlee, 1999) for scalar multiplication (Vijayalakshmi and Palanivelu, 2007). The double-and-add algorithm is a fundamental technique in calculating scalar multiplication. It performs by repeating point addition and point doubling operations which is discussed earlier. In this explanation, all equations for point addition and point doubling in GF(p) and GF(2^m) are summarized.

The numbers of ones in the binary representation of 'k' are expected to be m/2. 'm' represents the length of the integer number 'k'. The number of ones in 'k' shows how many times the point addition will be performed.

The number of times for point doubling operation performed is approximately equal to 'm'. Therefore, double-and-add algorithm averagely takes 'm' times point doubling. m/2 times point is added addition to perform m-bit elliptic curve scalar multiplication at one time.

Data flow for ECC point doubling and point addition is based on the Table 4 in GF(2^m)which is presented in Fig. 8 and 9.

A Montgomery bit-serial modular inversion algorithm and Right-to-left shift multiplication bit-serial are developed in this research to reduce the number of Input/output pins for scalar multiplication on elliptic curves by using double-and-add algorithm.

Therefore, in this research the scalar multiplication on elliptic curves in GF(2^m) can be used to deal with various binary polynomials in GF(2^m). The arithmetic which are introduced in GF(2^m) fields are suitable to be implemented in hardware, since they are binary arithmetic.

CONCLUSION

Nowadays, RSA generally is used as public key cryptosystem in most applications that use PKC. However, recently ECC has a trend which makes it become the convenient cryptography system. ECC is also becomes substitute for RSA in efficacious applications caused by its efficiency in software as well as in hardware realizations. ECC provides a better security with shorter bit sizes than in RSA. Shorter key length saves bandwidth, power and it enhances the performance. In contrast with the experts because it can be used to build a number of cryptographic schemes that cannot be constructed in any other way. The research starts with survey of cryptography, Elliptic Curve arithmetic and Elliptic Curve operations hierarchy algorithms. Our approach is begun with competent design for finite field arithmetic, mostly focusing on inversion and multipliers. The design of efficient arithmetic algorithm in bit-serial structure for Right-to-left shift multiplication and Montgomery inversion is shown. Montgomery inversion plays a consequential task in elliptic curve scalar multiplication. A bit-serial approach minimizes the number of Input/outputs which has a direct effect on power consumption. Three macrocells per bit are exploited for the multiplicand, multiplier and the product. Eventually, Area saving can be achieved because it does not need to store reduction polynomial in a register. The result of proposed bit-serial architecture for the multiplication and inversion on finite field arithmetic appears to be an important consumption of area in comparison with others.

REFERENCES

Al-Somani, T.F., M.K. Ibrahim and A. Gutub, 2006. Highly efficient elliptic curve crypto-processor with parallel GF(2^m) field multipliers. J. Comput. Sci., 2: 395-400. http://www.scipub.org/fulltext/jcs/jcs25395-400.pdf

Ali, Y.S.M., 2009. Implementation of elliptic curve cryptography using biometric features to enhance security services. M.Sc. Thesis, University of Malaya. http://dspace.fsktm.um.edu.my/handle/1812/911

Anoop, M.S., 2007. Elliptic curve cryptography-an implementation guide. TATAELXSI. http://www.tataelxsi.com/whitepapers/ECC_Tut_v1_0.pdf?pdf_id=public_key_TEL.pdf

De Dormale, G.M., P. Bulens and J.J. Quisquater, 2004. An improved Montgomery modular inversion targeted for efficient implementation on FPGA. Proceeding of the IEEE International Conference on Field-Programmable Technology, Dec. 6-8, IEEE Xplore Press, USA., pp: 441-444. DOI: 10.1109/FPT.2004.1393320

Greenlee, M.B., 1999. ANSI X9.62:2005 public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standards Institute. http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.62%3A2005

Kaabneh, K. and H. Al-Bdour, 2005. Key exchange protocol in elliptic curve cryptography with no public point. Am. J. Applied Sci., 2: 1232-1235. http://www.scipub.org/fulltext/ajas/ajas281232-1235.pdf

Kaliski, Jr., J.B., 1995. The Montgomery inverse and its applications. IEEE Trans. Comput., 44: 1064-1065. DOI: 10.1109/12.403725

Kammer, R.G. and W.M. Daley, 2000. Digital Signature Standard (DSS). National Institute for Standards and Technology. http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf

Koblitz, N., 1987. Elliptic curve cryptosystems. Math. Comput., 48: 203-209. http://www.jstor.org/pss/2007884

Kumar, S.S., 2008. Elliptic Curve Cryptography for Constrained Devices: Algorithms, Architectures, and Practical Implementations. 1st Edn., VDM Verlag, New York, ISBN: 3639068599, pp: 160.

Li, H., J. Huang, P. Sweany and D. Huang, 2008. FPGA implementations of elliptic curve cryptography and Tate pairing over binary field. EUROMICRO, 54: 1077-1088. DOI: 10.1016/j.sysarc.2008.04.012

Modares, H., 2009. A scalar multiplication in elliptic curve cryptography with binary polynomial operations in Galois Field. M.Sc. Thesis, University of Malaya. http://dspace.fsktm.um.edu.my/bitstream/1812/913/1/Hero_Modares_thesis_11_11_09.pdf

Quisquater, J.J. and C. Couvreur, 1982. Fast decipherment algorithm for RSA public-key cryptosystem. Elect. Lett., 18: 905-907. DOI: 10.1049/el:19820617

Rivest, R.L., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21: 120-126**.** DOI: 10.1145/359340.359342

Sandoval, M.M., 2008. A reconfigurable and interoperable hardware architecture for elliptic curve cryptography. Ph.D. Thesis, National Institute for Astrophysics, Optics and Electronics. http://ccc.inaoep.mx/~mmorales/documents/ThesisPhD.pdf

Savas, E. and C. Koc, 2000. The Montgomery modular inverse-revisited. IEEE Trans. Comput., 49: 763-766. DOI: 10.1109/12.863048

Vijayalakshmi, V. and T.G. Palanivelu, 2007. Secure antnet routing algorithm for scalable adhoc networks using elliptic curve cryptography. J. Comput. Sci., 3: 939-943. http://www.scipub.org/fulltext/jcs/jcs312939-943.pdf