# FPGA Implementation of an Elliptic Curve Cryptosystem over $GF(3^m)$

İlker Yavuz[1,2], Sıddıka Berna Örs Yalçın[1], and Çetin Kaya Koç[3,4]
[1]Istanbul Technical University
[2]The Scientific & Technological Research Council of Turkey (TUBITAK)
[3] University of California Santa Barbara, [4] City University of Istanbul
E-mails: ilkery@uekae.tubitak.gov.tr, siddika.ors@itu.edu.tr, koc@cs.ucsb.edu

## Abstract

*This paper describes an efficient arithmetic processor for elliptic curve cryptography. The proposed processor consists of special architectural components, the most important of which is a modular multiplication unit implemented using the systolic Montgomery multiplication algorithm. Another novelty of our proposed architecture is that it implements the field $GF(3^m)$, which provides significant performance gains.*

## 1. Introduction

Elliptic curve cryptography (ECC) is a public-key cryptographic technique, proposed by Miller [10] and Koblitz [5] as an alternative to the RSA [14]. The security of ECC is based on the elliptic curve discrete logarithm problem [6]. According to New European Schemes for Signature, Integrity and Encryption (NESSIE) report [13], ECC can provide same security level with RSA using shorter encryption key. Shorter key implies less memory need and lower power consumption. Also, ECC implementations are generally faster than RSA cryptosystem for equal key lengths [13].

In this paper, an ECC defined over $GF(3^m)$ finite field is implemented on a field programmable gate array (FPGA). There are several reasons why we choose $GF(3^m)$. One of these reasons is significant area and performance gain. Also, to the best of our knowledge, there are few hardware implementations of ECC over $GF(3^m)$, while there are many such implementations of ECC over $GF(p)$ and $GF(2^m)$.

In our implementation, we use Montgomery Modular Multiplication (MMM) algorithm [11] for modular multiplication operation which has considerable effect on the ECC performance. Also, other subblocks which form whole ECC are adapted to the selected finite field and implemented. These subblocks are modular addition/subtraction (MAS), modular multiplicative inversion (MMI), elliptic curve point multiplier (EPM) and elliptic curve point doubling/addition (EPDA) blocks. In addition, transformation circuits that convert normal inputs to appropriate forms are implemented for sub-blocks separately. All these sub-blocks are controlled using a state machine.

The remainder of this paper is organized as follows: in Section 2, a summary of previous implementations is given. In Section 3, we give a brief mathematical background for ECC. Section 4 presents hardware implementation of EC processor and implementation results and finally Section 5 concludes the paper.

## 2. Previous Contributions

To the best of our knowledge, there are only a few hardware implementations of elliptic curves defined over $GF(p^m)$. We compare our results only to those given in [1, 12, 4]. There are many software implementations of ECC over $GF(p^m)$, but it is not meaningful to compare our work with them. In 2003, Bertoni report the circuit performance using LSDE multiplier and cubing circuit [1]. Also in 2003, Page and Smart report characteristic 3 arithmetic for use in cryptosystems based on the Tate and Weil Pairing [12]. Page and Smart employ projective coordinate which we have used in this implementations. Another hardware work on characteristic 3 is given in [4] by Kerins et al. They use modified Duursma-Lee algorithm using Tate-Pairing method in 2005.

## 3. Mathematical Background

### 3.1. Montgomery modular multiplication over $GF(p^m)$

For modular multiplication, we choose Montgomery's algorithm [11]. The approach of Montgomery avoids the time consuming trial division that is a common bottleneck

978-0-7695-3474-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ReConFig.2008.66

in naive modular multiplication algorithms. Montgomery's technique is proved to be very efficient in both hardware and software implementations and is the basis of many implementations of the modular multiplication in cryptography. In this paper, we work on elliptic curves defined over $GF(3^m)$ finite field, and thus, we adapt the general form of MMM algorithm to $GF(p^m)$ representation. The Montgomery multiplication is defined as follows;

$$Mont(x, y) = xyR^{-1} \bmod N,$$

where $N = (n_{l-1}n_{l-2}\dots n_1n_0)_b$, $0 < x, y < N$, $R = b^l > N$ with $gcd(N, b) = 1$. In this paper, we describe only the MMM for the field $GF(p^m)$. More details for MMM algorithm can be found in [11, 9, 7].

In $GF(p^m)$, Montgomery modular multiplication of two numbers represented in the polynomial basis is defined as follows:

$$c(x) = a(x)b(x)r^{-1}(x) \bmod n(x),$$

where $r(x) = x^l$. The algorithm is given below.

**Algorithm 1** Montgomery Modular Multiplication Algorithm over $GF(3^m)$

**Require:** $a(x) = a_{l-1}x^{l-1} + a_{l-2}x^{l-2} + \dots + a_1x + a_0$, $b(x) = b_{l-1}x^{l-1} + b_{l-2}x^{l-2} + \dots + b_1x + b_0$, $n(x) = n_lx^l + n_{l-1}x^{l-1} + \dots + a_1x + n_0$, $n_0 = 1$, $-n(x)'n(x) \bmod p(x) = 1$, $r(x) = x^l$ and $gcd(n(x), r(x)) = 1$
**Ensure:** $c(x) = a(x)b(x)x^{-l} \bmod n(x)$
1: $c(x) = 0$
2: **for** $i = 0$ to $l - 1$ **do**
3: $u_i = (c_0 + a_ib_0)n'(x) \bmod 3$
4: $c(x) = c(x) + a_ib(x) + u_in(x)$
5: $c(x) = c(x)/x$
6: **end for**

Montgomery's method for multiplying two numbers defined in $GF(p^m)$ avoids trial division by $n(x)$ which is an expensive operation in hardware.

Algorithm 1 checks the last digit of partial product of $a(x)b(x)$ in every step of the for loop. If the addition of $a_ib_0$ product and the least significant digit of $c(x)$ is 0, it means the partial product is divisible by $x$ and nothing is added to $c(x)$. It is divided by $x$ directly. If the last digit of the partial product is not equal to 0, then a value must be added to the partial product to make it divisible by $x$. At this step, adding a multiple of the reduction polynomial, $n(x)$, does not change the result since the result is given in $\bmod n(x)$. $n'(x)$ is used to determine which multiple of $n(x)$ will be added. For $p = 3$, $n(x)' \bmod 3 = -n(x)^{-1} \bmod 3 = (3 - n_0)^{-1} \bmod 3 = 2^{-1} \bmod 3 = 2$. Division by $x$ is performed by a right shift operation.

### 3.2. Elliptic curve cryptography over $GF(3^m)$

An elliptic curve $E$ which is defined on a field $K$ is expressed with the solutions of the Weierstrass equation and the point at infinity [15]. General form of this equation is given as follows:

$$y^2 + a_1xy + a_5^3y = x^3 + a_2x^2 + a_4x + a_6$$

For characteristic 3, Weierstrass equation can be simplified using the transformations given in [2]. With these transformations, an elliptic curve equation can be expressed in $GF(3^m)$ as follows:

$$\begin{aligned} y^2 &= x^3 + a_2x^2 + a_6,\ j(E) \neq 0 \\ y^2 &= x^3 + a_4x + a_6,\ j(E) = 0 \end{aligned} \quad (1)$$

The solutions of Eq. 1 and the point at infinity form an Abelian group with the addition operation and these points are used in the cryptosystem. Multiplication of a point on the curve with a scalar is the main operation for ECC. This operation can be done using the double-and-add algorithm [10, 5] shown below.

**Algorithm 2** Double-and-Add Algorithm

**Require:** $K = k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + \dots + k_1 + k_0$ with $k_{n-1} = 1$ and $P = (x, y)$
**Ensure:** $Q = KP = (x', y')$
$Q = P$
**for** $i = n - 2$ downto 0 **do**
$Q = 2Q$
**if** $k_i = 1$ **then**
$Q = Q + P$
**end if**
**end for**

Addition and point doubling operations in Algorithm 2 are done in the affine coordinates using formulas given in [16]. These formulas include inversion operations which are very expensive in hardware. We can perform addition and doubling in the projective coordinates with only one inversion operation at the end of the point multiplication. The benefits of using the projective coordinates in ECC are explained in [8].

### 3.3. Polynomial representation

Our algorithm for MMM is defined on the polynomial representation. An element of a of $GF(p^m)$ is represented by a polynomial with m coefficients as follows:

$$a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_1x^1 + a_0,$$

where the coefficients $a_i \in GF(p)$.

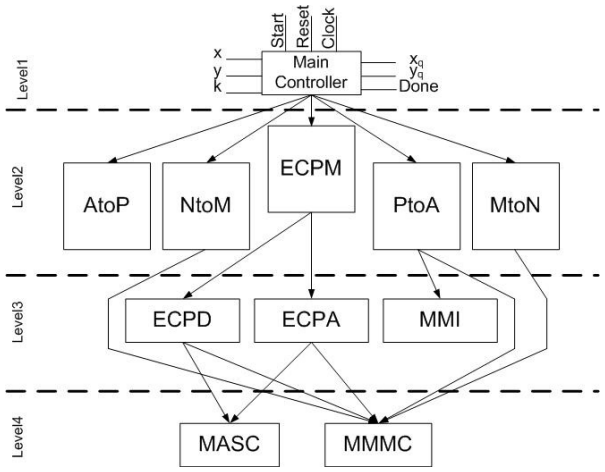


**Figure 1. EC point multiplier circuit block diagram**

## 4. Hardware implementation

Our elliptic curve processor (ECP) can be divided into four hierarchical levels as shown in Fig. 1. The operation blocks at each level from top to bottom are as follows;

1. Main Controller (MC).
2. Affine to projective converter (AtoP), Normal to Montgomery converter (NtoM), EC point multiplier (ECPM), Projective to Affine converter (PtoA), Montgomery to normal converter (MtoN).
3. EC point doubling circuit (ECPD), EC point addition circuit (ECPA), Modular multiplicative inverter (MMI).
4. Modular addition/subtraction circuit (MASC), Montgomery modular multiplication circuit (MMMC).

For simplicity all blocks have their own FSMs and data paths. This allows for independent optimization and testing of the building blocks.

### 4.1. Montgomery modular multiplication circuit design

As seen in Fig. 1, the Montgomery modular multiplication circuit (MMMC) is used by most of the sub-blocks in ECC. Implementation of Algorithm 1 includes a systolic array structure which minimizes the maximum path delay. Also, maximum clock frequency is independent of the input digit size. $c(x) = c(x) + a_i b(x) + u_i n(x)$ value in Algorithm 1 is updated in every iteration and $c(x)$ gives the

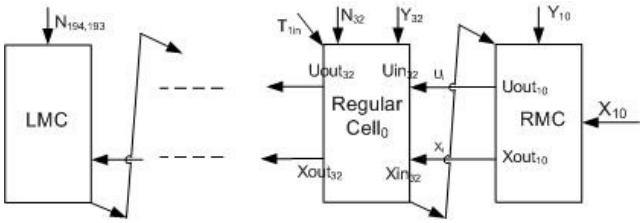


**Figure 2. Systolic array structure**

result after the last iteration. Every digit of $c(x)$ is calculated by a different systolic array cell. Because our finite filed is $GF(3^m)$, every digit of $c(x)$ is expressed by a value in $GF(3)$ field. Total number of cells is 1 more than total digit number.

The systolic array consists of 3 different cell structures. The first one is Right Most Cell (RMC) which calculates $u_i$ in step 3 of Algorithm 1. $a_i$, $b_0$ and $c_0$ are the inputs of this cell. It generates $u_i$ and transfers the input $a_i$ to its output. Second cell structure, Regular Cell (RC), calculates one ternary digit of $c(x)$. For $i^{th}$ iteration of Algorithm 1, $j^{th}$ cell calculates $c_{i,j} = c_{i-1,j+1} + a_i b_j + u_i n_j \bmod 3$. $c_{i-1,j+1}$, $a_i$, $b_j$, $u_i$ and $n_j$ are given as inputs. It calculates $c_{i,j}$ and gives it to previous cell as an input. Also, it transfers $a_i$, $u_i$ inputs to its output. Last cell in the array is the Left Most Cell (LMC). It is a simpler form of the RC. Last digit of $c(x)$ does not take $a_i$ and $b_0$ as inputs so using RC with zero values of $a_i$ and $b_0$ gives us LMC.

In the systolic array structure, ternary digits of the multiplier and ternary digits of the irreducible polynomial $n(x)$ are given parallel to every cell. The least significant digit of the multiplicand is given to RMC in every two clock cycles and then multiplicand is shifted to the right one ternary digit (2 bits).

Total number of clock cycles to obtain the result is $2l + 2m$, where $l$, is the digit size of the inputs and $m$ is the number of cells in the array.

Performance of the MMMC is given for Xilinx Virtex 1000E. The systolic array is 97 ternary digits width. For the total number of iteration steps $l = 97$ and the number of cells $m = 97$, the total number of clock cycles is 388. The total circuit area is 951 slices and the maximum clock frequency is 80,723 MHz. The minimum clock period, $T_p$, is 12,388 nanoseconds. The multiplication time is 4.8 $\mu$sec. The throughput rate is 40.41 Mb/sec.

Because most of the sub-blocks of the EC use the MMMC, the performance of it effects the whole performance of ECC considerably. The MMMC performs multiplication operation with reduction so we do not need to design any separate reduction block. Another advantage of the MMMC is its extendable structure.

### 4.2. Modular addition/subtraction circuit design

Modular addition/subtraction circuit (MASC) adds or subtracts (according to add/subtract switch) the coefficients of the input polynomials in $GF(p)$ domain.

### 4.3. Point addition and point doubling circuits in projective coordinates

The inputs to the given formulas below are points on the curve that are written in the projective coordinates as explained in Section 3.2. The outputs of equations are also in the projective coordinates. Let $P = (x_1, y_1, z_1)$,$Q = (x_2, y_2, z_2)$. The projective coordinate equivalent of the point addition and the doubling equations are as follows [3].
**Point Addition:**$(P + Q = (x_3, y_3, z_3))$
$\lambda_1 = x_1 z_2^2$, $\lambda_2 = x_2 z_1^2$, $\lambda_3 = \lambda_1$ - $\lambda_2$, $\lambda_4 = y_1 z_2^3$
$\lambda_5 = y_2 z_1^3$, $\lambda_6 = \lambda_4$ - $\lambda_5$, $\lambda_7 = \lambda_1 + \lambda_2$
$\lambda_8 = \lambda_4 + \lambda_5$, $z_3 = z_1 z_2 \lambda_3$, $x_3 = \lambda_6^2$ - $\lambda_7$ $\lambda_3^2$
$y_3 = \lambda_8$ $\lambda_3^3$ - $\lambda_6^3$

**Point Doubling:**$(P + P = (x_3, y_3, z_3))$
$\lambda_1 = -z_1^4$, $z_3 = -y_1 z_1$, $\lambda_2 = x_1 y_1^2$, $x_3 = \lambda_1^2 + \lambda_2$
$\lambda_3 = -y_1^4$, $y_3 = \lambda_1(\lambda_2 - x_3) - \lambda_3$

We are using MMMC in these equations for multiplication operations. Also, MASC is used for addition/subtraction operation.

### 4.4. Modular multiplication inverter

The modular multiplicative inversion is computed using Fermat's theorem [6, 9]. For $GF(p^m)$ field Fermat's theorem is given as follows:

$$a^{-1} = a^{p^m - 2} \bmod p(x)$$

where $p(x) = p_m x^m + p_{m-1} x^{m-1} + \ldots + p_1 x^1 + p_0$ is an irreducible polynomial and $p_i \in GF(p)$. To calculate $a^{3^{97}-2} \bmod p(x)$, we converted $3^{97} - 2$ to binary representation and used square-and-multiply algorithm [9]. MMI controls the execution of the square-and-multiply algorithm

### 4.5. Affine to projective coordinates converter

Because the point addition and doubling operations in the affine coordinates include the inversion operation which is very expensive in hardware, we choose to work in the projective coordinates. The benefits of the projective coordinates are explained in [8]. For $GF(p^3)$ finite field, the coordinate transformation between the affine and the projective coordinates is as follows [3],

$$(x, y) = (X/Z^2, Y/Z^3)$$

Using this transformation, affine to projective (AtoP) conversion is as follows;

$$(X, Y, Z) = (xZ^2, yZ^3, Z) \quad (2)$$

In Eq. 2, if we choose $Z = 1$ then it is given as follows;

$$(X, Y, Z) = (x, y, 1)$$

where $(x, y)$ is the point in the affine coordinates and $(X, Y, Z)$ is the same point in the projective coordinates. This operation is just adding 1 as a third coordinate.

### 4.6. Normal to Montgomery representation converter

The multiplication using the MMMC of two polynomials that are in Montgomery representation will produce the Montgomery representation of the product as

$$Mont(a(x)r(x), b(x)r(x)) = a(x)b(x)r(x) \bmod n(x)$$

Also, modular addition of two polynomials that are in the Montgomery representation produce the Montgomery representation of the sum as $a(x)r(x) \bmod n(x) + b(x)r(x) \bmod n(x) = (a(x) + b(x))r(x) \bmod n(x)$. Because of these relations, the Montgomery representation of the coordinates of $P$ point is calculated in the beginning of the point multiplication by NtoM circuit and all the operations during EC point multiplication will be performed in the Montgomery representation. The conversion to the Montgomery representation of any number is computed as follows

$$Mont(a(x), r(x)^2) = a(x)r(x) \bmod n(x)$$

.

### 4.7. Elliptic curve point multiplier

The Elliptic Curve Point Multiplier (EPM) circuit controls the execution of the Algorithm 2. In every iteration of the loop, an ECPD is executed. An ECPA is only performed when the evaluated key bit is 1.

### 4.8. Projective to affine coordinates converter

After completing the EC point multiplication, the result point $Q$ must be converted from the projective coordinates

**Table 1. Area performance of ECC sub-blocks**

| Operation | # of Slices | # of LUTs |
|---|---|---|
| NormaltoMont | 1658 | 1718 |
| ECPM | 11758 | 20482 |
| PtoA | 4115 | 7185 |
| MtoN | 1481 | 1664 |
| ECPD | 4634 | 8693 |
| ECPA | 5100 | 9532 |
| MMI | 1876 | 3179 |
| MAS | 423 | 789 |
| MMMC | 951 | 1719 |

**Table 2. Time performance of ECC sub-blocks**

| Operation | Sub-operations | # of Clock | Time($\mu$sn) |
|---|---|---|---|
| NtoM | 2MMM | 776 | 10.34 |
| ECPM | k.ECPD, (k/2)ECPA | 1134318 | 15124 |
| PtoA | 4MMM+ 1MMI | 91180 | 1215.7 |
| MtoN | 2MMM | 776 | 10.34 |
| ECPD | 8MMM+ 6MAS | 3122 | 41.62 |
| ECPA | 14MMM+ 6MAS | 5450 | 72.66 |
| MMI | (3l/2)MMM | 89628 | 1195 |
| MAS | - | 3 | 0.04 |
| MMMC | - | 388 | 5.17 |
| MC | All | 1225498 | 16339 |

to the affine coordinates. This operation is performed as follows:

$$(x, y) = (X/Z^2, Y/Z^3) \,,$$

where $(x, y)$ is the point in affine coordinates and $(X, Y, Z)$ is the same point in projective coordinates. This conversion needs 1 inversion and 4 multiplications. First, MMI of $Z(x)$, $Z(x)^{-1}$ is calculated. Then $Z(x)^{-2}$ and $Z(x)^{-3}$ are calculated using MMMC twice. Finally, $X(x)Z(x)^{-2}$ and $Y(x)Z(x)^{-3}$ values are calculated using the MMMC. As it is explained in Section 4.6, all MMMC inputs must be converted to the Montgomery representation before calculation.

### 4.9. Montgomery to normal representation converter

Because the coordinates of the product point must be in the normal representation, as a last action, all the results must be converted to the normal representation. Let $a'(x) = a(x)r(x)$ be the Montgomery representation of $a(x)$. Then, we have $Mont(a'(x), 1) = a(x) \bmod n(x)$.

### 4.10. Implementation results

Our ECC processor is implemented on the VirtexE family FPGA. The input parameters are the coordinates of a point on an elliptic curve and the scalar value. Also other control signals such as RESET, START and DONE, are available. Area requirements of all ECC sub-blocks for 97 ternary digits (194 bits) inputs is given in Table 1.

Timing results of ECC sub-blocks for 97 ternary digits inputs are given in Table 2. Because MC is placed at the top of all the other blocks and controls the execution of the point multiplication, the maximum operating frequency of this block is used to calculate the execution time of all the other sub-blocks.

In the literature, there are only a few hardware implementations for elliptic curves defined over $GF(p^m)$. Also, these implementations use different methods and technologies. We cannot compare our design under the same conditions. We have located three studies about elliptic curves over $GF(3^{97})$ [12, 1, 4]. They give multiplication circuit performance instead of the elliptic curve performance. We tried to use the same FPGA technology with the evaluated studies for comparison. We could achieve this comparison for only two of these. The first one is using the Tate-Pairing method and the second work is implemented using the Least Significant Digit-Element (LSDE) multiplication method. The study given in [12] is implemented using the serial bit multiplication method and uses old technology so we cannot use the same technology with this work and thus we only report their performance results. All comparison results are given in Table 3.

As can be seen from Table 3 that our implementation has the smallest area. Also, our study is comparable with [4]. The results given in [1] are optimized for specific parameters therefore it is not meaningful to compare it with our generic structure.

Comparing our $GF(3^m)$ with $GF(2^m)$ is not meaningful because implementation of these fields have different hardware architecture. It is obvious that arithmetic operations in $GF(2^m)$ is simpler because addition and multiplication in binary field are just one operation. In $GF(3^m)$, these operations need slightly larger look-up tables. Therefore, these fields can be compared just in the cryptographic complexity point of view.

Also, we give performance results using latest FPGA technology which is available now. This result is given in the bottom row of Table 3.

## 5. Conclusions

We have described an efficient implementation of a elliptic curve processor over the field $GF(3^m)$. The processor can be programmed to execute a modular multiplication, addition, subtraction, multiplicative inversion, EC

**Table 3. Comparison of final results**

| | Tool/FPGA | Area | Mult.Time |
|---|---|---|---|
| Ref [4] TatePairing | Not Defined Virtex2Pro | 4335 Slice Mult. 2210 Slice Inv. | 0.9$\mu$sn |
| Our Study MMM | Xilinx Virtex2Pro | 1215 Slice | 1.66$\mu$sn@ 223.7MHz |
| Ref [1] LSDE | Synopsis 3.7.1 Xilinx 1000 | 7080 LUT 618 FF | 0.097$\mu$sn@ 94.4MHz |
| Our Study MMM | Xilinx XCV1000 | 1719 LUT 1019 FF | 3.8$\mu$sn@ 101.86MHz |
| Ref [12] SerialBit | Handel-C Xilinx4000XL | Not Defined | 50.68$\mu$sn@ 20MHz |
| Our Study MMM | Xilinx Virtex5 | 1019 LUT 1178 FF | 1.054$\mu$sn@ 368.05MHz |

point addition, point doubling, and point multiplication operations. We use the Montgomery systolic array architecture for modular multiplication. Our architecture uses the Montgomery method for modular multiplication because of its implementation advantages. The systolic array architecture makes the clock frequency independent of the bit length of inputs. Also the circuit is expendable and reusable for higher input values.

## References


[1] G. Bertoni, J. Guajardo, S. S. Kumar, G. Orlando, C. Paar, and T. J. Wollinger, "Efficient $GF(p^m)$ arithmetic architectures for cryptographic applications," in *Topics in Cryptology - CT-RSA, The Cryptographers' Track at the RSA Conference*, Marc Joye, Ed., San Francisco, CA, USA, April 13-17 2003, vol. 2612 of *Lecture Notes in Computer Science*, pp. 158–175, Springer.

[2] A. Enge, *Elliptic Curves and Their Application to Cryptography: An Introduction*, Boston Kluwer Academics, 1999.

[3] K. Harrison, D. Page, and N.P. Smart, "Software implementation of finite fields of characteristic three for use in pairing-based cryptosystems," *LMS Journal of Computation and Mathematics*, vol. 5, pp. 181–193, November 2002.

[4] T. Kerins, W. P. Marnane, E. M. Popovici, and P. S. L. M. Barreto, "Efficient hardware for the tate pairing calculation in characteristic three," in *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Josyula R. Rao and Berk Sunar, Eds., Edinburgh, UK, August 29 - September 1 2005, vol. 3659 of *Lecture Notes in Computer Science*, pp. 412–426, Springer.

[5] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comp.*, vol. 48, pp. 203–209, 1987.

[6] N. Koblitz, *A Course in Number Theory and Cryptography*, vol. 114 of *Graduate Texts in Mathematics*, Springer-Verlag, Berlin, Germany, second edition, 1994.

[7] Ç. K. Koç, *Cryptographic Engineering*, Springer, 2009.

[8] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[9] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[10] V. Miller, "Uses of elliptic curves in cryptography," in *Advances in Cryptology: Proceedings of CRYPTO'85*, H. C. Williams, Ed., Santa Barbara, CA, USA, August 18-22 1985, vol. 218 of *Lecture Notes in Computer Science*, pp. 417–426, Springer-Verlag.

[11] P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. Vol. 44, pp. 519–521, 1985.

[12] D. Page and N. P. Smart, "Hardware implementation of finite field of characteristic three," in *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, Eds., Redwood Shores, CA, USA, August 13-15 2002, vol. 2523 of *Lecture Notes in Computer Science*, pp. 529–539, Springer-Verlag.

[13] B. Preneel et al., "New European Schemes for Signatures, Integrity, and Encryption," European Project IST-1999-12324, April 19, 2004.

[14] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[15] L. C. Washington, *Elliptic Curves: Number Theory and Cryptography*, Chapman & Hall/CRC, Boca Raton, 2003.

[16] I. Yavuz, "FPGA Implementation of an Elliptic Curve Cryptosystem," Master's Thesis, Istanbul Technical University, January 2008.