Design of elliptic curve cryptoprocessors over GF(2^{163}) using the Gaussian normal basis

Diseño de criptoprocesadores de curva elíptica sobre GF(2^{163}) usando bases normales Gaussianas

P.C. Realpe-Muñoz[1], V. Trujillo-Olaya[2] and J. Velasco-Medina[3]

ABSTRACT

This paper presents an efficient hardware implementation of cryptoprocessors that perform the scalar multiplication kP over a finite field GF(2^{163}) using two digit-level multipliers. The finite field arithmetic operations were implemented using the Gaussian normal basis (GNB) representation, and the scalar multiplication kP was implemented using the Lopez-Dahab algorithm, the 2-non-adjacent form (2-NAF) halve-and-add algorithm and the w–τNAF method for Koblitz curves. The processors were designed using a VHDL description, synthesized on the Stratix-IV FPGA using Quartus II 12.0 and verified using SignalTAP II and Matlab. The simulation results show that the cryptoprocessors provide a very good performance when performing the scalar multiplication kP. In this case, the computation times of the multiplication kP using the Lopez-Dahab algorithm, 2-NAF halve-and-add algorithm and 16–τNAF method for Koblitz curves were 13.37 µs, 16.90 µs and 5.05 µs, respectively.

Keywords: elliptic curve cryptography, Gaussian normal basis, digit-level multiplier, scalar multiplication.

RESUMEN

En este trabajo se presenta la implementación eficiente en hardware de criptoprocesadores que permiten llevar a cabo la multiplicación escalar kP sobre el campo finito GF(2^{163}) usando dos multiplicadores a nivel de digito. Las operaciones aritméticas de campo finito fueron implementadas usando la representación de bases normales Gaussianas (GNB), y la multiplicación escalar kP fue implementada usando el algoritmo de López-Dahab, el algoritmo de bisección de punto 2–NAF y el método w–τNAF para curvas de Koblitz. Los criptoprocesadores fueron diseñados usando descripción VHDL, sintetizados en el FPGA Stratix-IV usando Quartus II 12.0 y verificados usando SignalTAP II y Matlab. Los resultados de simulación muestran que los criptoprocesadores presentan un muy buen desempeño para llevar a cabo la multiplicación escalar kP. En este caso, los tiempos de computo de la multiplicación kP usando Lopez-Dahab, bisección de punto 2–NAF y 16–τNAF para curvas de Koblitz fueron 13.37 µs, 16.90 µs and 5.05 µs, respectivamente.

Palabras clave: criptografía de curva elíptica, bases normales Gaussianas, multiplicador a nivel de digito, multiplicación escalar.

Received: October 29th 2013
Accepted: February 25th 2014

Introduction

The use of computer networks and the steady increase in the number of users of these systems have driven the need to improve security for the storage and transmission of information. There are many applications that must ensure the privacy, integrity or authentication of the information stored or transmitted. The security of the applications has been resolved by using different cryptographic algorithms, which are used in private- or public-key cryptosystems.

The security of public-key cryptosystems is based on mathematical problems that are computationally difficult to resolve, i.e., problems for which there are no known algorithms to resolve them in a practical time. Because of the high volume of information processed, electronic systems are required to perform the encryption and decryption processes in the shortest time possible without compromising the security. In this regard, hardware implementations of cryptographic algorithms have advantages, such as high speed, high security levels and low cost.

One of the most important cryptosystems is the elliptic curve cryptosystem (ECC), proposed independently by Koblitz (Kobliz, 1987) and Miller (Miller, 1986). There have been several investigations of the theory and practice of this cryptosystem. The results of the investigations demonstrated the ability of these systems to encrypt information and concluded that this cryptosystem offers

[1] Paulo Cesar Realpe Muñoz. Bs in Physic Engineering, Universidad del Cauca, Colombia. M.Sc. in Electronics Engineering, Universidad del Valle, Colombia. Affiliation: Universidad del Valle, Colombia. E-mail: paulo.realpe@correounivalle.edu.co
[2] Vladimir Trujillo Olaya. Bs in Electronic Engineering, Universidad del Valle, Colombia. M. Sc. in Electronics Engineering, Universidad del Valle, Colombia. Affiliation: Universidad del Valle, Colombia. E-mail: vladimir.trujillo@correounivalle.edu.co
[3] Jaime Velasco Medina. B.S in Electrical Engineering, University del Valle, Colombia. Ph.D in Microelectronics, TIMA-INPG, France. Universidad del Valle, Colombia. E-mail: jaime.velasco@correounivalle.edu.co

How to cite: Realpe-Muñoz, P. C., Trujillo-Olaya, V., & Velasco-Medina, J. (2014). Design of elliptic curve cryptoprocessors over GF(2^{163}) using the Gaussian normal basis. Ingeniería e Investigación, 34(2), 55-65.

better security, efficiency and memory usage. The hardware implementations of ECCs have many advantages and are used in equipment such as ATMs, smart cards, telephones, and cell phones.

In elliptic curve cryptography, it is known that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point, that is, the elliptic curve discrete logarithm problem or ECDLP, has high hardness. The entire security of the ECC depends on the ability to compute the scalar multiplication and the inability to compute the multiplicand given the original and product points. Furthermore, the finite-field size of the elliptic curve determines the computational complexity of the above problem.

Several works regarding scalar multiplication over a finite field *GF*(2^m) have been proposed and implemented efficiently in hardware.

C. Rebeiro and D. Mukhopadhyay (Rebeiro and Mukhopadhyay, 2008) presented a cryptoprocessor with novel multiplication and inversion algorithms. J.Y. Lai, T.Y. Hung, K.H. Yang and C.T. Huang (Lai *et al.*, 2010) proposed an architecture for elliptic curves along with the operation scheduling for the Montgomery scalar multiplication algorithm. B. Muthukumar and S. Jeevananthan (Muthukumar and Jeevanahthan, 2010) implemented an elliptic curve coprocessor, which is a dual-field processor with a projective coordinate. A.K. Rahuman and G. Athisha (Rahuman and Athisha, 2010) presented an architecture using the Lopez-Dahab algorithm for the elliptic curve point multiplication and Gaussian normal basis (GNB) for field arithmetic over *GF*(2^{163}). M. Amara and A. Siad (Amara and Siad, 2011) proposed an EC point multiplication processor intended for cryptographic applications such as digital signatures and key agreement protocols. X. Cui and J. Yang (Cui and Yang, 2012) implemented a processor that parallelizes the computations of the ECC at the bit-level and gains a considerable speedup. The processor is fully implemented in hardware and supports key lengths of 113 bits, 163 bits and 193 bits.

In this context, we present in this work efficient hardware implementations of cryptoprocessors over *GF*(2^{163}) using a GNB representation and the Lopez-Dahab algorithm, 2-NAF halve-and-add algorithm and *w*-τNAF method for Koblitz curves (Anomalous Binary Curves or ABC) with window sizes of 2, 4, 8 and 16 to perform the scalar multiplication *kP*.

The main contributions of this work are: (i) the hardware design of cryptoprocessors using the GNB over GF(2^{163}) and three scalar multiplication algorithms (Lopez-Dahab, halve-and-add and w-τNAF method for Koblitz curves) to determine the best cryptoprocessor for embedded cryptographic applications. (ii) an efficient hardware implementation of cryptoprocessors based on the *w*-τNAF method with different window sizes for the Koblitz curves. They present the best trade-off between the computation time and area, obtaining a higher performance than the other cryptoprocessors reported in the literature. Additionally, they are very suitable for hardware cryptosystems.

Mathematical background

GNB representation

ANSI X9.62 (ANSI, 1999) describes the detailed specifications of the ECC protocols and uses the GNB to represent the finite field elements (NIST, 2000). An element over GF(2^m) has the computational advantage of performing squaring very efficiently. However, multiplying distinct elements can be cumbersome. In this case, there are multiplication algorithms that make this operation both simpler and more efficient.

A normal basis over GF(2^m) is as follows:

$$\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\} \tag{1}$$

where $\beta \in$ GF(2^m) and any element $A \in$ GF(2^m) can be written as follows:

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} \quad a_i \in \{0,1\} \tag{2}$$

The type *T* of a GNB is a positive integer and measures the complexity of the multiplication operation with respect to that basis. Generally, the type *T* of a smaller value provides a more efficient multiplication. For a given *m* and *T*, the field *GF*(2^m) can have at most one GNB of type *T*. A GNB exists whenever *m* is not divisible by 8. Let *m* and *T* be two positive integers. Then, the type *T* of a GNB over *GF*(2^m) exists if and only if $p = Tm+1$ is prime.

If $\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\}$ is a GNB over GF(2^m), then the element $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ is represented by the binary string $(a_0 a_1 a_2 \dots a_{m-1})$, where $a_i \in \{0, 1\}$. In this case, the multiplicative identity element is represented by the bit string of all ones.

The additive identity element is represented by the bit string of all zeros. An important result for the GNB arithmetic is Fermat's Theorem. For all $\beta \in$ GF(2^m), then

$$\beta^{2^m} = \beta \tag{3}$$

This theorem is important for performing the squaring of an element over *GF*(2^m).

Finite field arithmetic operations

The following arithmetic operations can be performed over GF(2^m) when using a normal basis of type *T*.

Addition: If $A = (a_0 a_1 a_2 \dots a_{m-1})$ and $B = (b_0 b_1 b_2 \dots b_{m-1})$ are elements over GF(2^m), then $A + B = C = (c_0 c_1 c_2 \dots c_{m-1})$, where $c_i = (a_i + b_i)$ mod 2.

Squaring: Let $A = (a_0 a_1 a_2 \dots a_{m-1}) \in$ GF(2^m), then

$$A^2 = \left(\sum_{i=0}^{m-1} a_i \beta^{2^i}\right)^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{m-1} \beta^{2^i} \tag{4}$$

Based on Fermat's Theorem, $\beta^{2^m} = \beta$, then

$$A^2 = (a_{m-1} a_0 a_1 \dots a_{m-2}) \tag{5}$$

In this case, squaring is a simple rotation of the vector representation.

Multiplication: The multiplication $C = A \cdot B$ is based on the multiplication matrix $R_{(m-1)\times T}$ (Masoleh, 2006). If $A = (a_0 a_1 a_2 \dots a_{m-1})$ and $B = (b_0 b_1 b_2 \dots b_{m-1})$ are elements over GF(2^m) and are represented using a GNB, then $A \cdot B = C = (c_0 c_1 c_2 \dots c_{m-1})$, where the coefficient c_0 is given by equation (6)

$$c_0 = a_0 b_1 + \sum_{i=0}^{m-1} a_i \left(\sum_{j=1}^{T} b_{R(i,j)}\right) \tag{6}$$

and $R(i,j), 0 \le R(i,j) \le m-1,\ 1 \le i \le m-1,\ 1 \le j \le T$ denotes the (*i*, *j*)th element of the $R_{(m-1)\times T}$ matrix. To obtain the *i*th coefficient of *C*, i.e., c_i, add "*i* mod *m*" to all indices in (6).

Inversion: If $A \neq 0$ and $A \in GF(2^m)$, the inverse of A is $C \in GF(2^m)$, and C is the only element of $GF(2^m)$ such that $A \cdot C = 1$, i.e., $C = A^{-1}$. The algorithm used to calculate the inversion is based on equation (7):

$$A^{-1} = A^{2^m-2} = \left(A^{2^m-1}\right)^2 \tag{7}$$

Itoh and Tsujii (Itoh and Tsujii, 1998) proposed a method that reduces the number of multiplications to calculate the inversion, and it is based on the following:

$$A^{2^m-2} = \begin{cases} A \cdot \left(A^{2^{m-2}-1}\right)^2 & if \;\; m \;\; even \\ \left(\left(A^{2^{\frac{m-1}{2}}-1}\right)^{2^{\frac{m-1}{2}}} \cdot A^{2^{\frac{m-1}{2}}-1}\right)^2 & if \;\; m \;\; odd \end{cases} \tag{8}$$

Trace: If A is an element over $GF(2^m)$, the trace of A is:

$$Tr(A) = A + A^2 + A^{2^2} + \ldots + A^{2^{m-1}} \tag{9}$$

If $A = (a_0 a_1 a_2 \ldots a_{m-1})$ and it is represented in a normal basis, then the trace can be computed efficiently as follows:

$$Tr(A) = a_0 \oplus a_1 \oplus \ldots \oplus a_{m-1} \tag{10}$$

The trace of the element A has two possible values (0 or 1). Quadratic equation solving over $GF(2^m)$: If A is an element of $GF(2^m)$ represented in a normal basis, then the quadratic equation:

$$z^2 + z = A \tag{11}$$

has $2 - 2T$ solutions over $GF(2^m)$, where $T = Tr(A)$. Therefore, if $T = 1$, there is no solution, and if $T = 0$, there are two solutions. If z is one solution, then the other solution is z + 1. For example, if $A = 0$, the solutions are z = 0 and z = 1 (IEEE std 1363, 2000). The algorithm 1 calculates the quadratic equation over $GF(2^m)$ for a normal basis representation.

Algorithm 1: Quadratic equation solving over $GF(2^m)$
Input: An element $A \neq 0$
Output: An element z for which $z^2 + z = A$
1. Let $(a_0 a_1 \ldots a_{m-1})$ be the representation of A
2. Set $z_0 \leftarrow 0$
3. For i from 0 to $m - 1$ do
3.1 Set $z_i \leftarrow z_{i-1} \oplus a_i$
4. Return $z \leftarrow (z_0 z_1 \ldots z_{m-1})$

Square root: Let $A = (a_0 a_1 a_2 \ldots a_{m-1}) \in GF(2^m)$, then

$$\sqrt{A} = \left(a_1 a_2 \ldots a_{m-1} a_0\right) \tag{12}$$

In this case, the square root in a normal basis is a simple rotation of the vector representation (IEEE std 1363, 2000).

Elliptic curve arithmetic

A non-supersingular elliptic curve $E(F_q)$ is defined as a set of points $(x, y) \in GF(2^m) \times GF(2^m)$ that satisfies the affine coordinates equation,

$$y^2 + xy = x^3 + ax^2 + b \tag{13}$$

where a and $b \in F_q$ and are constants with $b \neq 0$ together with the point at infinity denoted by O. The group operations for the elliptic curve arithmetic in affine coordinates are defined as follows. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points that belong to the curve, and let the addition inverse of P be defined as $-P = (x_1, x_1 + y_1)$. Then, if $Q \neq -P$, the point $P + Q = (x_3, y_3)$ can be computed as:

$$x_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a & P \neq Q \\ x_1^2 + \dfrac{b}{x_1^2} & P = Q \end{cases} \tag{14}$$

$$y_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1 & P \neq Q \\ x_1^2 + \left(x_1 + \dfrac{y_1}{x_1}\right)x_3 + x_3 & P = Q \end{cases} \tag{15}$$

Using the group operations above, the elliptic curve scalar multiplication can be defined as follows. Let E be an elliptic curve over $GF(2^m)$, let Q and $P \in E$ be two arbitrary elliptic points satisfying equation (13), and let k be an arbitrary positive integer. Then, the elliptic curve scalar multiplication $Q = kP$ is defined as:

$$kP = \underbrace{P + P + \ldots + P}_{k \; times} \tag{16}$$

Considering the group operations described in equations (14) and (15) using the finite field arithmetic in affine coordinates, three main elliptic curve operations can be defined: point addition, point doubling and point halving. In the group operations, the inversion is the arithmetic operation that is most expensive over $GF(2^m)$, and this operation can be avoided with a projective coordinate representation. In this case, the inversion is avoided by using the finite field multiplication.

A point P in the projective coordinates is represented using three coordinates (X, Y and Z). For the *Lopez-Dahab* (LD) projective coordinates (Lopez and Dahab, 1999), the projective point $(X : Y : Z)$ with $Z \neq 0$ corresponds to the affine coordinates $x = X/Z$ and $y = Y/Z^2$. Then, equation (13) can be mapped from the affine coordinates to the LD projective coordinates as:

$$Y^2 + XYZ = X^3 Z + aX^2 Z^2 + bZ^4 \tag{17}$$

The three group operations for the elliptic curve arithmetic in the projective and affine coordinates can be computed as (Menezes *et al*., 2003):

1. Point doubling $Q = 2P$, where $Q = (X_3 : Y_3 : Z_3)$ and $P = (X_1 : Y_1 : Z_1)$ in the projective coordinates, can be performed using 4 finite field multiplications, such as

$$\begin{gathered} Z_3 = X_1^2 Z_1^2 \qquad X_3 = X_1^4 + bZ_1^4 \\ Y_3 = bZ_1^4 Z_3 + X_3\left(aZ_3 + Y_1^2 + bZ_1^4\right) \end{gathered} \tag{18}$$

2. Point addition $Q + P$, where $Q = (X_1 : Y_1 : Z_1)$ in the projective coordinates and $P = (x_2, y_2)$ in the affine coordinates, can be performed using 8 finite field multiplications, such as

$$\begin{array}{ll} A = y_2 Z_1^2 + Y_1 & B = x_2 Z_1 + X_1 \\ C = Z_1 B & D = B^2\left(C + aZ_1^2\right) \\ Z_3 = C^2 & E = AC \\ X_3 = A^2 + D + E & F = X_3 + X_2 Z_3 \\ G = (x_2 + y_2)Z_3^2 & Y_3 = (E + Z_3)F + G \end{array} \tag{19}$$

3. Point halving $Q/2$ is the inverse operation of point doubling. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be the points over the curve (13) in the affine coordinates. The point halving operation is performed by computing P such that $Q = 2P$ by solving the following equations:

$$\lambda^2 + \lambda = x_2 + a \tag{20}$$

$$x_1 = \sqrt{y_2 + x_2(\lambda + 1)} \tag{21}$$

$$y_1 = \lambda x_1 + x_1^2 \tag{22}$$

Let the λ-representation of a point $Q = (x_2, y_2)$ be $Q = (x, \lambda_Q)$, where

$$\lambda_Q = x_2 + y_2 / x_2 \tag{23}$$

If Q in the λ-representation is the input of the point halving algorithm, then it is possible to compute point halving without using the affine coordinates. In scalar multiplication, repeated point halving operations can be performed directly on the λ-representation. However, when a point addition is required, a conversion to the affine coordinates must be performed. Algorithm 2 computes the point halving operation.

Algorithm 2: Point Halving
Input: λ-representation (x_2, λ_Q) or affine representation (x_2, y_2) of Q
Output: λ-representation (x_1, λ_P) of $P = (x_1, y_1)$
1. Find a solution $\hat{\lambda}$ of $\widehat{\lambda^2} + \hat{\lambda} = x_2 + a$
2. If the input is a λ-representation, then $t = x_2(x_2 + \lambda_Q + \hat{\lambda})$
Else $t = x_2 + x_1\hat{\lambda}$
3. If Tr(t)=0 then $\lambda_P \leftarrow \hat{\lambda},\ x_1 \leftarrow \sqrt{t + x_1}$
Else $\lambda_P \leftarrow \hat{\lambda} + 1,\ x_1 \leftarrow \sqrt{t}$
4. Return (x_1, λ_P)

Koblitz Curves

Koblitz curves, or anomalous binary curves, are elliptic curves defined over GF(2^m). The main advantage of these curves is that the scalar multiplication operation can be performed without the use of point doubling operations.

An algorithm for scalar multiplication on Koblitz curves is presented by Solinas (Solinas, 2000). The Solinas algorithm or the τ–adic window method computes a special τ–adic expansion of an integer number in $\mathbb{Z}[\tau]$. For example, a special τ–adic expansion is the window τ–adic non-adjacent form (τNAF).

The Koblitz curves are curves defined over $GF(2^m)$ by:

$$E_a = y^2 + xy = x^3 + ax + 1 \tag{24}$$

where $a \in \{0,1\}$, that is, curves E_0 and E_1.

These curves present the following property: If $P(x, y)$ is a point on the curve E_a, then the point (x^2, y^2) is also a point on E_a. In addition, they satisfy $(x^4, y^4) + 2(x, y) = \mu(x^2, y^2)$ for each point (x, y) on E_a, where $\mu = (-1)^{1-a}$. In $GF(2^m)$, the Frobenius map τ is an endomorphism that raises every element to its power of two, i.e., $\tau : x \rightarrow x^2$. Then, the Frobenius endomorphism is performed efficiently (cost-free) when the elements of the finite field are represented in a normal basis (Cui and Yang, 2012). Koblitz shows that the point doubling operation can be performed efficiently by using the Frobenius endomorphism, if the binary curve is defined over $GF(2^m)$ and $a \in \{0, 1\}$. Then, the Frobenius map can be defined as τ: $(x, y) \rightarrow (x^2, y^2)$. In this case, if the scalar k is represented in τNAF, then

$$\sum_{i=0}^{l-1} k_i \tau^i \quad \text{for } k_i \in \{0,1,-1\} \tag{25}$$

The τ–adic representation can be obtained by repeatedly dividing k by τ, where the remainders of each division step are named digits u_i. This procedure is also used to obtain the representation's NAF of the scalar k, namely, k is repeatedly divided by 2. To decrease the number of point additions for the scalar multiplication, it is necessary to obtain a τNAF representation of k that achieves a smaller number of nonzero digits. The scalar multiplication can be computed as:

$$kP = \sum_{i=0}^{l-1} k_i \tau^i (P) \tag{26}$$

The result corresponds to the Hamming weight of the τNAF, and it is equal to the binary NAF representation, i.e., the Hamming weight ≈ ($\log_2 k$)/3, and the length of the τ–adic representation of k is approximately $2m$, which is twice the length of the binary NAF representation. However, Solinas presents a method that reduces the length of the τ–adic representation to approximately m. Thus, the Koblitz curves' arithmetic is based on the point addition and Frobenius map τ.

Hardware architectures for elliptic curve cryptoprocessors

In this section, we present the hardware architectures for elliptic curve cryptoprocessors over $GF(2^{163})$ using a Gaussian normal basis. Each cryptoprocessor is designed using one algorithm for the scalar multiplication, namely, the Lopez-Dahab algorithm (Lopez and Dahab, 1999), the halve-and-add 2-NAF algorithm (Menezes *et al*., 2000) and the w-τNAF method for Koblitz curves with w = 2, 4, 8 and 16 (Solinas, 2000).

Digit-level multiplier

The finite field multiplication over $GF(2^m)$ is an operation that is more important for performing the scalar multiplication. Thus, this operation must be implemented efficiently in hardware. There are several algorithms for performing the finite field multiplication that are presented in Azarderakhsh and Masoleh (2010), Huang et al. (2011,), Wang and Fan (2012) Lee and Chiou (2012).

Azarderakhsh and Masoleh (Azarderakhsh and Masoleh, 2010) proposed a serial or parallel digit-level multiplier with a digit-size d, where $1 \leq d \leq m$. In this case, if $d = m$, the multiplier is parallel and if $d < m$, it is serial and requires $M = \lceil m/d \rceil$, $1 \leq M \leq m$, clock cycles to generate all the m coefficients of $C = A{\cdot}B = (c_0c_1c_2\ldots c_{m-1})$, where $A = (a_0a_1a_2\ldots a_{m-1})$ and $B = (b_0b_1b_2\ldots b_{m-1})$ are elements represented in a GNB over $GF(2^m)$. Figure 1 shows the digit-level $GF(2^m)$ multiplier for T = 4, where A, B and C are registers for storing the input and output elements.

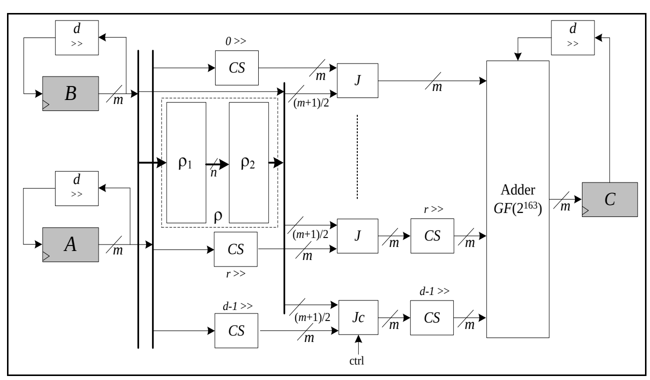

Figure 1. Digit-level Multiplier

The block ρ is formed by the blocks ρ_1 and ρ_2, and its structure depends on type T of the GBN with $T \geq 2$ and the multiplication matrix R. The block J is a set of m, two-input AND gates. The block *CS* is a *d*-fold cyclic shift and an adder GF (2^{163}), which is a set of two-input XOR gates.

The block ρ_1 is an optimal set of XOR gates that are obtained using (27), and ρ_2 is a set of XOR gates that are obtained from the main matrix ρ:

$$P_k(B) = (b_{1-k}, s_0^{'}(1,\ B<<k), s_0^{'}(2,\ B<<k), \ldots$$
$$, \ldots, s_0^{'}((m-1)/2,\ B<<k),\ 0 \le k \le d-1$$

$$s_0^{'}(k,B) = \sum_{i \in R_k} b_{i-k} \qquad (27)$$

The time complexity of the digit-level multiplier is $T_A + (2 + \lceil \log_2 m \rceil) T_X$, where T_X and T_A are the delay time of a two-input XOR gate and a two-input AND gate, respectively. The area complexity of this multiplier is m^2 ANDs and $\le 2\underline{m}^2 - 2\underline{m}$ XORs (Azarderakhsh and Masoleh, 2010).

To implement the digit-level multiplier with a digit-size d = 55 in hardware, that is M = 3 clock cycles, a Matlab code is written to generate the equations of the blocks ρ_1 and ρ_2, which are synthesized using VHDL.

Hardware architecture using the Lopez-Dahab algorithm

The scalar multiplication *k*P for non-supersingular elliptic curves over binary fields using the Lopez-Dahab algorithm is shown in Algorithm 3, which is a modified version of the Montgomery algorithm, where the same operations are performed during each iteration of the main loop (D. Hankerson *et al*., 2003).

Algorithm 3: Montgomery point multiplication

Input: $k = (k_{t-1},\ldots,k_1k_0)$ with $k_{t-1} = 1$, $P = (x, y) \in GF(2^m)$
Output: *kP*
1. $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2$
2. For i from t - 2 downto 0 do
3. if $k_i = 1$ then
   3.1 $T \leftarrow Z_1, Z_1 \leftarrow (X_1Z_2 + X_2Z_1)^2, X_1 \leftarrow xZ_1 + X_1X_2TZ_2$
   3.2 $T \leftarrow X_2, X_2 \leftarrow X_2^4 + bZ_2^4, Z_2 \leftarrow T^2Z_2^2$
4. else
   4.1 $T \leftarrow Z_2, Z_2 \leftarrow (X_1Z_2 + X_2Z_1)^2, X_2 \leftarrow xZ_2 + X_1X_2TZ_1$
   4.2 $T \leftarrow X_1, X_1 \leftarrow X_1^4 + bZ_1^4, Z_1 \leftarrow T^2Z_1^2$
5. $x_3 \leftarrow X_1/Z_1$
   $y_3 \leftarrow \left(x + \frac{X_1}{Z_1}\right)[(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1Z_2)](xZ_1Z_2)^{-1} + y$
6. Return (x_3, y_3)

In this case, the scalar multiplication is performed in three steps: 1) conversion of *P* from affine to projective coordinates; 2) compute *Q* = *kP* by addition and doubling; and 3) conversion of *Q* from projective to affine coordinates.

To implement the above algorithm in hardware, we initially define three functions: $M_{add}()$ performs the point addition, $M_{double}()$ performs the point doubling and $M_{xy}()$ performs the conversion from projective to affine coordinates. These functions are defined as follows:

$$\begin{array}{ll} M_{add}(X_1, Z_1, X_2, Z_2) & M_{double}(X_1, Z_1) \\ \{X_1 \leftarrow X_1Z_2X_2Z_1 + x(X_1Z_2 + X_2Z_1)^2 & \{X_2 \leftarrow X_2^4 + bZ_2^4 \\ Z_1 \leftarrow (X_1Z_2 + X_2Z_1)^2 & Z_2 \leftarrow X_2^2 Z_1^2 \\ \text{Return}\,(X_1, Z_1)\} & \text{Return}\,(X_2, Z_2)\} \end{array}$$

where, (x, y) and (x_3, y_3) are the coordinates of points *P* and *Q* = *kP*, respectively.

Point addition and point doubling are implemented in hardware using the data dependence graph shown in Figure 2, and the conversion from the projective to affine coordinates is implemented using two digit-level multipliers for the data dependence graph shown in Figure 3. The inversion operation is implemented using the Itoh-Tsujii algorithm (Itoh and Tsujii, 1998).

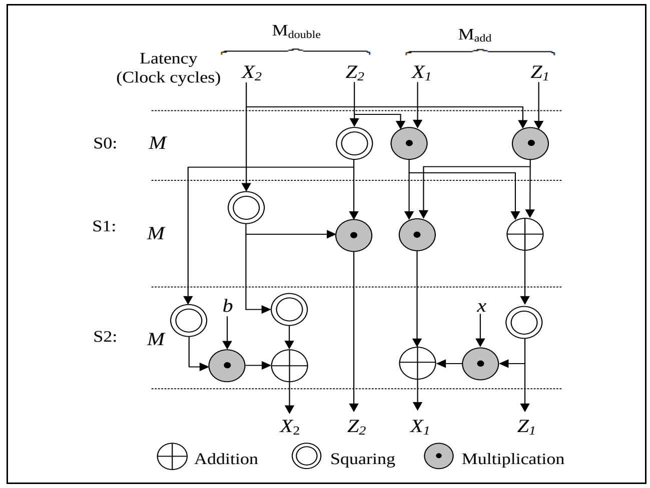

Figure 2. Data dependence graph for M_{add} and M_{double}

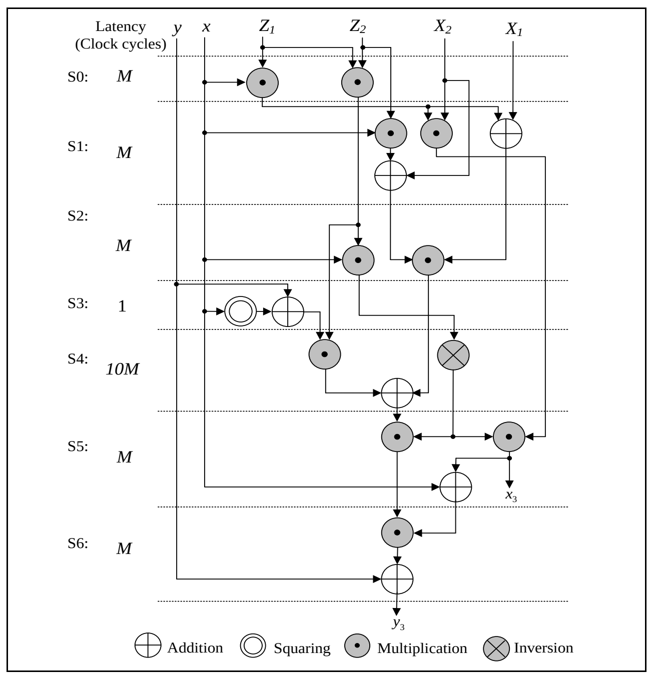

Figure 3. Data dependence graph for conversion of the projective to affine coordinates

According to Figures 2 and 3, the latencies for M_{add} and M_{double} and the projective to affine conversion are $3\underline{M}$ and $15\underline{M}$ +1, respectively, where *M* is the latency for a finite field multiplication.

In step 4 of Figure 3, two multipliers are used, and one of them with the block of rotation performs the inversion of an element *A* $\in GF(2^{163})$. In this case, the latency of the inversion is $10\underline{M}$ because it needs 10 finite field multiplications for m = 163. In step 6, a multiplier is only used because the last operation of the coordinate conversion requires a multiplication.

The architecture of the cryptoprocessor over GF(2^{163}) using the Lopez-Dahab algorithm is shown in Figure 4. It uses two register files, two parallel digit-level multipliers, one inversion block, several squaring and adder blocks, a main control and an FSM to perform the point addition, point double and conversion from the projective to affine coordinates.

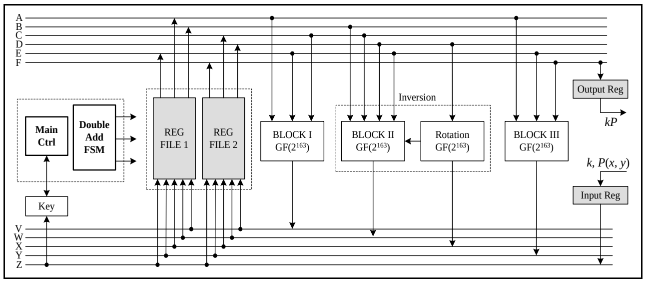

Figure 4. Elliptic curve cryptoprocessor using the Lopez-Dahab algorithm

The functional blocks that perform the finite field arithmetic operations over GF(2^{163}) for the Lopez-Dahab cryptoprocessor are shown in Figure 5. It is important to mention that the performance of any cryptoprocessor depends on the efficient implementation of the hardware for the finite field arithmetic.

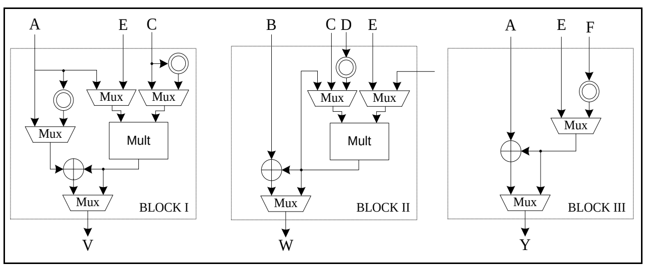

Figure 5. Functional blocks of the finite field arithmetic

The main control is an FSM that generates the control signals to perform the scalar multiplication, process the key, initialize the cryptoprocessor and control the I/O registers. The second FSM performs the point addition, point doubling and conversion from the projective to the affine coordinates.

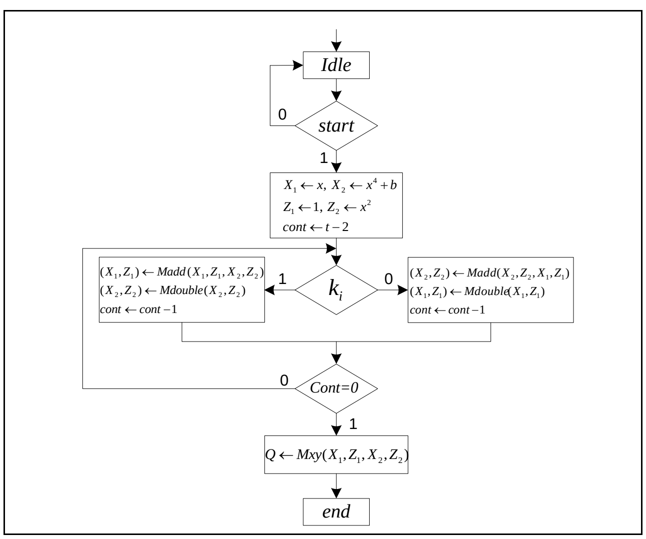

Figure 6. ASM chart of the main control

In Figure 6, the ASM chart of the main control is shown, where the variables X_1, Z_1, X_2 and Z_2 are initialized and stored in the register files. Each bit of the scalar k is evaluated from left to right to perform the operations M_{add} and M_{double} using the data dependence graph shown in Figure 2. If the bit k_i is '1', then $M_{add}(X_1,Z_1,X_2,Z_2)$, $M_{double}(X_2,Z_2)$ are computed. Else, $M_{add}(X_2,Z_2,X_1,Z_1)$, $M_{double}(X_1,Z_1)$. When all bits of the scalar k are evaluated, the conversion from the projective to affine coordinates is executed using the data dependence graph shown in Figure 3, and kP in the affine coordinates is stored in the output register.

Algorithm 3 is more resistant against simple power analysis and timing attacks. This is because the computation cost does not depend on the specific bit of the scalar k. For each bit of the scalar k, one point addition and one point doubling are performed. The proposed scheme has two different execution paths depending on the current bit of the scalar k. Both execution paths have the same complexity and require the same number of clock cycles.

Hardware architecture using the halve-and-add algorithm

Schroeppel (Schroeppel, 2000) and Knudsen (Knudsen, 1999) independently proposed the halve-and-add algorithm to accelerate the scalar multiplication on the elliptic curves defined over the binary extension fields. This algorithm uses an elliptic curve primitive called point halving as shown in algorithm 2.

Because, theoretically, the point halving operation is three times faster than the point doubling operation, it is possible to accelerate the scalar multiplication $Q = kP$ by replacing the double-and-add algorithm with the halve-and-add algorithm, which uses an expansion of the scalar k in terms of negative powers of 2 (Mercurio *et al.*, 2006).

In the halve-and-add algorithm, it is necessary to transform the integer $k = (k_{m-1},\ldots,k_0)_2$. If k' is defined by

$$k \equiv k'_{t-1}/2^{t-1}+,\ldots,+k'_1/2+k'_0 (\bmod n) \qquad (28)$$

where n represents the order of the base point P, then

$$kP = \sum_{i=0}^{t-1} k'_i / 2^i P \qquad (29)$$

Equation (29) can be generalized to a window-NAF. The NAF$_w$ of a positive integer k and $w \geq 2$ is represented by the expression $k = \sum_{i=0}^{l-1} k_i 2^i$, where each nonzero coefficient k_i is odd and at most, one of any w consecutive digits is nonzero. In this case, the NAF$_w$ of k can be computed using algorithm 4.

Algorithm 4: NAF$_w$ of a positive integer
Input: Window width w, positive integer k.
Output: NAF$_w$(k)
1. $i \leftarrow 0$
2. While $k \geq 1$ do
2.1 If k is odd then $k_i \leftarrow k$ mods 2^w, $k \leftarrow k - k_i$
2.2 Else $k_i \leftarrow 0$
2.3 $k \leftarrow k/2$, $i \leftarrow i+1$
3. Return ($k_{i-1},\ldots,k_1, k_0$)

In this work, a Maple code is written to obtain the expansion coefficients NAF$_w$ with $w = 2$, namely, the coefficients NAF$_w$(2^{t-1} k mod n), which are represented by 2-bits.

The halve-and-add algorithm is shown in algorithm 5. Step 3 of the algorithm performs the point addition $Q_i + P$ in the Lopez-Dahab mixed coordinates (Q_i and P are represented in LD projective and affine coordinates, respectively) using equation (14) and the halving point $P/2$ in the affine coordinates or λ-representation, if bit $k_i' \neq 0$; else, compute point halving. In this case, it is important to mention that if the results of the first two operations A and B of equation (19) are equal to zero, the point doubling $2P$ is performed in the LD projective coordinates using equation (18) with $X_1 = x_2$, $Y_1 = y_2$ and $Z_1 = 1$.

Algorithm 5: Halve-and-add w-NAF point multiplication

Input: Window width w, $\mathrm{NAF_w}(2^{t-1}\,\mathrm{k} \bmod \mathrm{n}) = \sum_{i=0}^{t} k'_i 2^i$, $P \in GF(2^m)$
Output: kP
1. Set $Q_i \leftarrow \infty$ for $i \in I = \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
2. if $k'_t = 1$ then $Q_1 = 2P$
3. For i from t - 1 downto 0 do
 3.1 if $k'_i > 0$ then $Q_{k'_i} \leftarrow Q_{k'_i} + P$
 3.2 if $k'_i < 0$ then $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
 3.3 $P \leftarrow P/2$
4. $Q \leftarrow \sum_{i \in I} iQ_i$
5. Return (Q)

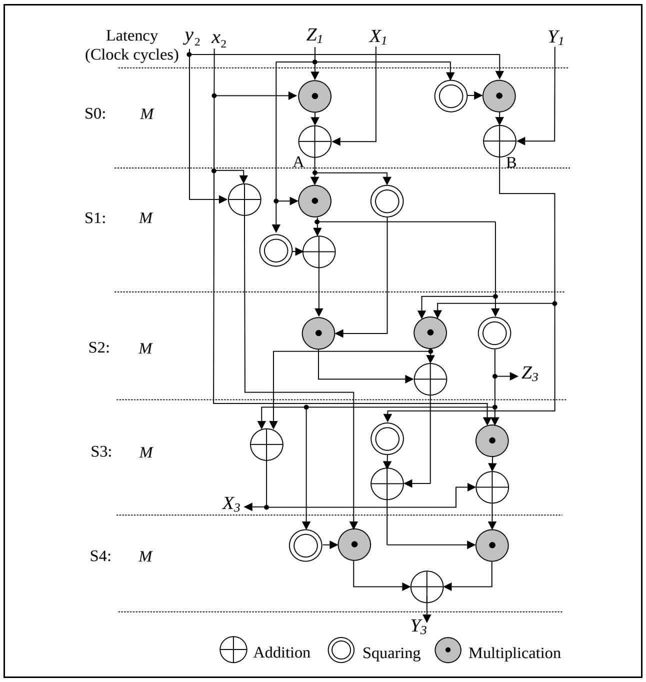

Figure 7. Data dependence graph for point addition

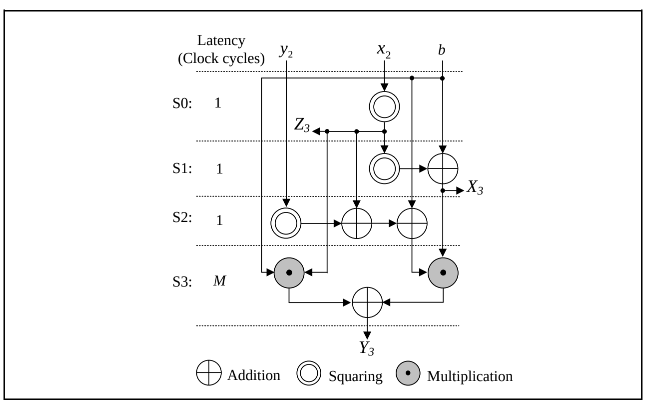

Figure 8. Data dependence graph for point doubling

The point addition in the LD mixed coordinates and the point doubling in the LD projective coordinates are implemented in hardware using the data dependence graphs shown in Figure 7 and Figure 8, respectively. According to Figures 7 and 8, the latencies for the point addition and point doubling are 5M and M + 3, respectively.

The architecture of the cryptoprocessor over GF(2163) using the halve-and-add algorithm is shown in Figure 9, and it uses two register files, two digit-level finite multipliers, one solving quadratic equation block, one point halving block, several squaring and adder blocks, a main control and an FSM to perform the point addition, point doubling and point halving.

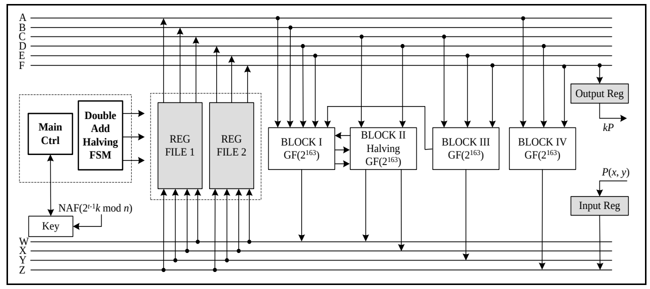

Figure 9. Elliptic curve cryptoprocessor using the halve-and-add algorithm

The functional blocks that perform the finite field arithmetic operations over $GF(2^{163})$ for the halve-and-add cryptoprocessor are shown in Figure 10. In this case, finite field arithmetic operations are the addition, squarer, square root, trace, half trace (quadratic equation solving in a normal basis) and multiplication.

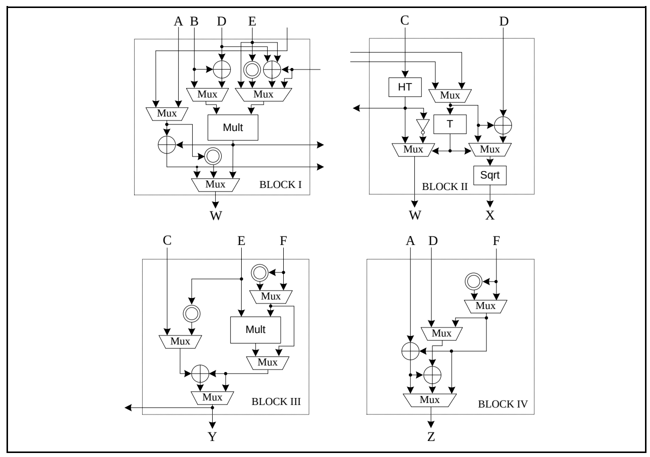

Figure 10. Blocks of the finite field arithmetic

The main control is an FSM that generates the control signals to perform the scalar multiplication, process the key, initialize the cryptoprocessor and control the I/O registers. The second FSM performs the point addition, point doubling and point halving.

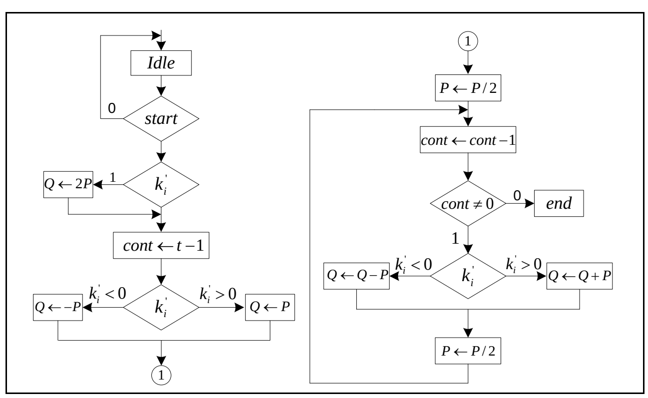

Figure 11. ASM chart of the main control

In Figure 11, the ASM chart of the main control is shown, where the sequence processing is as follows: initialize coordinate Q according to the sign of the bit k'_{t-1}; perform the point halving operation on P; evaluate the bit k'_i for $i > t-1$; compute the point addition in the LD mixed coordinates and point halving on P if $k'_i \neq 0$, else compute point halving; and perform the conversion of the point P in the λ-representation to the affine coordinates only when a point addition is required. Finally, $Q = kP$ is obtained in the LD projective coordinates.

Algorithm 7 performs the rounding of a complex number $\lambda_0 + \lambda_1\tau$ with λ_0 and $\lambda_1 \in \mathbb{Q}$ to obtain an element $\in \mathbb{Z}[\tau]$.

Hardware architecture using the w-τNAF algorithm

The length of the τ–*adic* representation for $k = d_0 + d_1\tau \in \mathbb{Z}[\tau]$ is roughly twice $log_2(\max(d_0, d_1))$. Solinas (Solinas, 2000) presents a method that reduces the length of the τ–adic representation. The objective is to find $\rho \in \mathbb{Z}[\tau]$ of small norm with $\rho \equiv k \pmod{\delta}$, where $\delta = (\tau^m - 1)/(\tau - 1)$, and use τNAF($\rho$) to calculate ρP.

Algorithm 6 calculates an element $\rho' \equiv k \pmod{\delta}$, which is also written as $\rho' = k$ partmod δ. Solinas proved that $l(\rho) \leq m + a$ and if $C \geq 2$, then $l(\rho') \leq m + a + 3$.

Algorithm 6: Partial reduction modulo $\delta = (\tau^m - 1)/(\tau - 1)$

Input: $k \in [n - 1]$, $C \geq 2$, $s_0 = d_0 - \mu d_1$, V_m, $s_1 = -d_1$
Output: $\rho' = k$ partmod δ
1. $k' = \lfloor k/2^{a-C+(m-9)/2} \rfloor$
2. For i from 0 to 1 do
   2.1 $g' \leftarrow s_i.k'$ $j' \leftarrow V_m.\lfloor g'/2^m \rfloor$
   2.2 $\lambda_i \leftarrow \lfloor (g'+j')/2^{(m+5)/2} + 1/2 \rfloor /2^C$
3. Use Algorithm 7 to compute $(q_0, q_1) \leftarrow$ Round(λ_0, λ_1)
4. $r_0 \leftarrow k-(s_0 + \mu s_1)q_0 - 2s_1q_1$ $r_1 \leftarrow s_1q_0 - s_0q_1$
5. Return $r_0 + r_1\tau$

Algorithm 7: Rounding off

Input: Rational numbers λ_0 and λ_1
Output: Integers q_0 and q_1
1. For i from 0 to 1 do
   $f_i \leftarrow \lfloor \lambda_i + 1/2 \rfloor$ $n_i \leftarrow \lambda_i - f_i$ $h_i \leftarrow 0$
2. $n \leftarrow 2n_0 + \mu n_1$
3. If $n \geq 1$ then
   3.1 If $n_0 - 3\underline{\mu} n_1 < -1$ then $h_1 \leftarrow \mu$; else $h_0 \leftarrow -1$
   Else
   3.2 If $n_0 - 4\underline{\mu}n1 \geq 2$ then $h_1 \leftarrow \mu$
4. If $n < -1$ then
   4.1 If $n_0 - 3\underline{\mu}n_1 \geq 1$ then $h_1 \leftarrow -\mu$; else $h_0 \leftarrow -1$
   Else
   4.2 If $n_0 - 4\underline{\mu}n_1 < -2$ then $h_1 \leftarrow -\mu$
5. $q_0 \leftarrow f_0 + h_0$, $q_1 \leftarrow f_1 + h_1$
6. Return (q_0, q_1)

Let $w \geq 2$ be a positive integer, and $\alpha_i = i \bmod \tau^w$ for $i \in \{1, 3, 5,\ldots, 2^{w-1} - 1\}$. A w–τNAF expansion of an nonzero element $\kappa \in Z[\tau]$ is an expression:

$$\rho = \sum_{i=0}^{l-1} u_i \tau^i \qquad (30)$$

where $u_i \in \{0, \pm\alpha_1, \pm\alpha_3,\ldots, \pm\alpha_{2^{w-1}-1}\}$, $u_{l-1} \neq 0$ and at most, one of any w consecutive digits is nonzero. Then, $kP = \alpha_{u0}P + \tau\alpha_{u1}P + \ldots + \tau^{l-1}\alpha_{l-1}P$, when the scalar k is represented in w–τNAF.

The w-τNAF expansion can be efficiently computed using algorithm 8, which can be viewed as an approach similar to the general NAF algorithm. In this work, a Maple code is written to obtain the expansion w–τNAF of the scalar k with w = 2, 4 and 8, generating 8-bit expansion coefficients and w = 16, generating 16-bit expansion coefficients.

Algorithm 8: Computing a w–τNAF of an element in $\mathbb{Z}[\tau]$

Input: w, t_ω, $\rho = r_0 + r_1\tau \in \mathbb{Z}[\tau]$, $\alpha_u = \beta_u + \gamma_u$ for $u \in \{1, 3, 5,\ldots,2^{\omega-1} - 1\}$
Output: w–τNAF(ρ)
1. $i \leftarrow 0$
2. While $r_0 \neq 0$ or $r_1 \neq 0$ do
   2.1 If r_0 is odd then
      $u \leftarrow r_0 + r_1t_\omega$ mods 2^w
      If $u > 0$ then $s \leftarrow 1$; else $s \leftarrow -1$, $u \leftarrow -u$
      $r_0 \leftarrow r_0 - s\beta_u$, $r_1 \leftarrow r_1 - s\gamma_u$, $u_i \leftarrow s\alpha_u$.
   2.2 Else: $u_i \leftarrow 0$
   2.3 $t \leftarrow r_0$, $r_0 \leftarrow r_1 + \mu r_0/2$, $r_1 \leftarrow -t/2$, $i \leftarrow i + 1$
3. Return $(u_{i-1}, u_{i-2},\ldots,u_1, u_0)$

Solinas proposed algorithms to compute kP using the window τNAF method for the scalar k, namely, kP is calculated using the w-τNAF method and Horner's rule (Solinas, 2000). An efficient scalar multiplication algorithm that uses the w-τNAF method is presented in algorithm 9, where step 1 calculates the w-τNAF of the scalar k with the partial reduction modulo $\delta = (\tau^m-1)/(\tau-1)$, namely, w-τNAF($\rho \equiv k \bmod(\delta)$), where $\rho \equiv k \bmod(\delta)$ is obtained from algorithms 6 and 7; step 2 generates the multiples of the point P and step 4.2 performs the point addition $Q + P_u$, when the bit $u_i \neq 0$, and point doubling $2Q$, when the results of the two first operations A and B of equation (19) are equal to zero.

Algorithm 9: w–τNAF point multiplication method for Koblitz curves

---

Input: Window width w, integer $k \in [1, n-1]$, $P \in$ GF(2^m) of order n
Output: kP
1. Compute $w{-}\tau\mathrm{NAF}(\rho \equiv k \bmod(\delta)) = \sum_{i=0}^{l-1} u_i\tau^i$
2. Compute $P_u = \alpha_u P$, for $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
3. $Q \leftarrow \infty$
4. For i from l – 1 downto 0 do
   4.1 $Q \leftarrow \tau Q$
   4.2 if $u_i \neq 0$ then
      Let u be such that $\alpha_u = u_i$ or $\alpha_{-u} = -u_i$
      If $u > 0$ then $Q \leftarrow Q + P_u$
      Else $Q \leftarrow Q - P_{-u}$
5. Return Q

The point addition in the LD mixed coordinates and the point doubling in the LD projective coordinates with $b = 1$ are implemented in hardware using the data dependence graphs shown in Figure 7 and Figure 8, respectively.

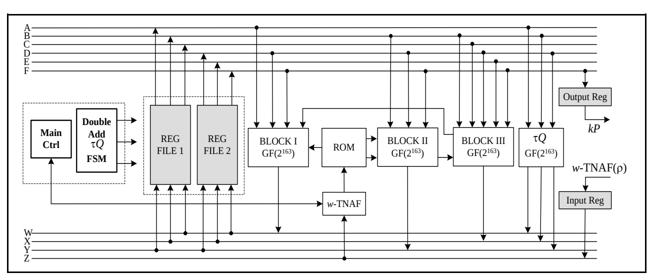

Figure 12. Elliptic curve cryptoprocessor for Koblitz curves

The architecture of the cryptoprocessor over GF(2^{163}) using the w-τNAF algorithm for Koblitz curves is shown in Figure 12, and it uses two register files, two digit-level finite multipliers, one Frobenius map block, one RAM that stores the expansion coefficients w-τNAF of the scalar k, two ROMs that store the pre-computed points P_u in the affine coordinates, which were obtained from Matlab for w = 2, 4, 8 and 16, several squaring and adder blocks, a main control and an FSM to perform the point addition, point doubling and τQ.

The functional blocks that perform the finite field arithmetic operations over GF(2^{163}) for the w–τNAF cryptoprocessor for Koblitz curves are shown in Figure 13.

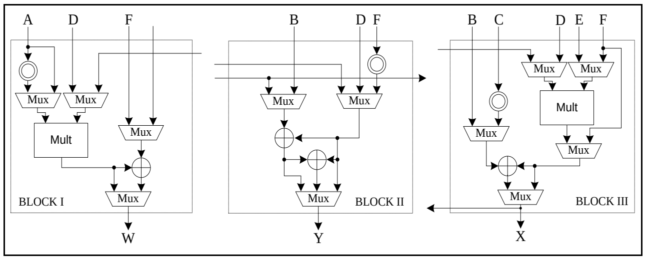

Figure 13. Blocks of the finite field arithmetic

The main control is an FSM that generates the control signals to perform the scalar multiplication, process the key, initialize the cryptoprocessor and control the I/O registers. The second FSM performs the point addition, point doubling and τQ.

In Figure 14, the ASM chart of the main control is shown, where the sequence processing is as follows: initialize the Q coordinate according to the sign of the bit u_i of the w-τNAF expansion; evaluate the bits u_i for $i > t-1$; and compute the point addition in the LD mixed coordinates and the Frobenius map τ on Q, if $u_i \neq 0$. Else, compute τQ. Finally, $Q = kP$ is obtained in the LD projective coordinates. In Figure 14, the ASM of the FSM is shown. One important remark is that the Koblitz curves are resistant to simple power analysis and to all the known special attacks (T. Juhas, 2007).

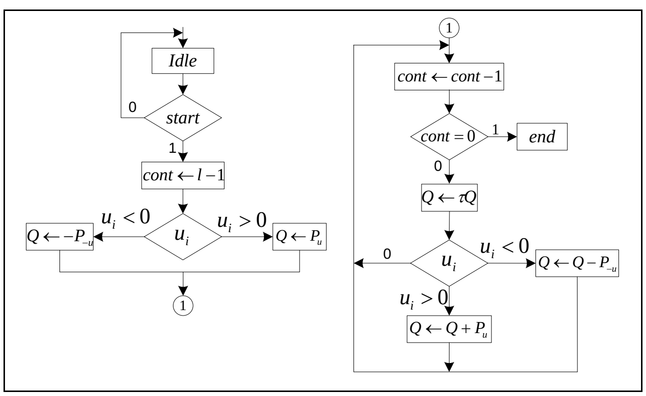

Figure 14. ASM chart of the main FSM

Hardware verification and synthesis results

The López-Dahab, halve-and-add and w-τNAF cryptoprocessors are described using generic structural VHDL, are synthesized for a digit-size of d = 55 on the Stratix-IV FPGA (EP4SGX180HF35C2) using the Altera Quartus II version 12 design software for the implementation and are verified using SignalTap II and Matlab.

Hardware verification of the cryptoprocessors

To verify the synthesis and simulation results of the cryptoprocessors, the following parameters for a pseudo-random elliptic curve are used according to the National Institute of Standards and Technology (NIST, 2000):

1. Random elliptic curves B–163:
The form of the curve is: $y^2 + xy = x^3 + x + b$
Gx = 3F0EBA16286A2D57EA0991168D4994637E8343E36
Gy = 0D51FBC6C71A0094FA2CDD545B11C5C0C7973244F1
b = 20A601907B8C953CA1481EB10512F78744A3205FD

2. Koblitz elliptic curves K–163
The form of the curve is: $y^2 + xy = x^3 + x + 1$
Gx = 2FE13C0537BBC11ACAA07D793DE4E6D5E5C94EEE8
Gy = 289070FB05D38FF58321F2E800536D538CCDAA3D9
n = 4000000000000000000020108A2E0CC0D99F8A5EF

In Figures 15 through 17, the simulation results for the cryptoprocessors over GF(2^{163}) in a GNB using SignalTAP II and Matlab are shown.

Figure 15. Simulation results for the Lopez-Dahab cryptoprocessor. (a) Results from SignalTAP II (b) Results from Matlab

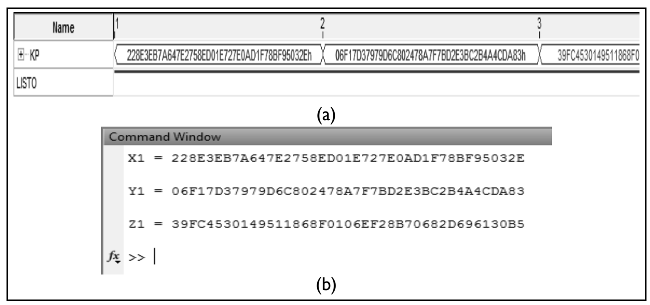

Figure 16. Simulation results for the halve-and-add cryptoprocessor. (a) Results from SignalTAP II (b) Results from Matlab

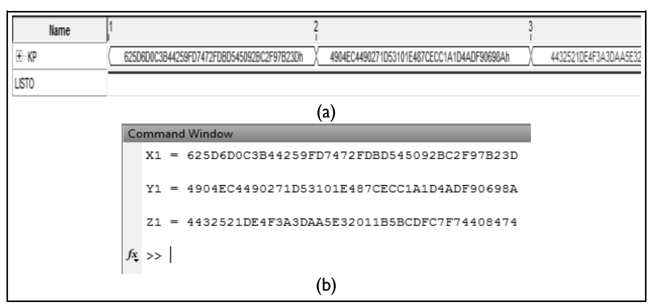

Figure 17. Simulation results for the Koblitz curves cryptoprocessor. (a) Results from SignalTAP II (b) Results from Matlab

From Figures 15 through 17, we can see that the results obtained from Matlab are the same as the results from SignalTAP II. Then, the hardware results verify the correct functionality of the designed cryptoprocessors.

Synthesis results for the cryptoprocessors

The synthesis results of the cryptoprocessors over GF(2^{163}) are shown in Table 1. Additionally, some of the data presented in Table I are plotted in Figure 18.

Table 1. Synthesis results for the cryptoprocessors

Cryptoprocessor	Area (ALUTs)	Fmax (MHz)	Registers	kP (μs)	T×A
Lopez-Dahab	24882	215.3	2608	13.37	0.33
Halving 2-NAF	22670	158.2	2572	16.90	0.38
Koblitz 2-τNAF	24223	226.6	2046	9.88	0.23
Koblitz 4-τNAF	24257	226.7	2050	7.37	0.17
Koblitz 8-τNAF	24249	211.6	2108	6.17	0.14
Koblitz 16-τNAF	24270	177.1	2135	5.05	0.12

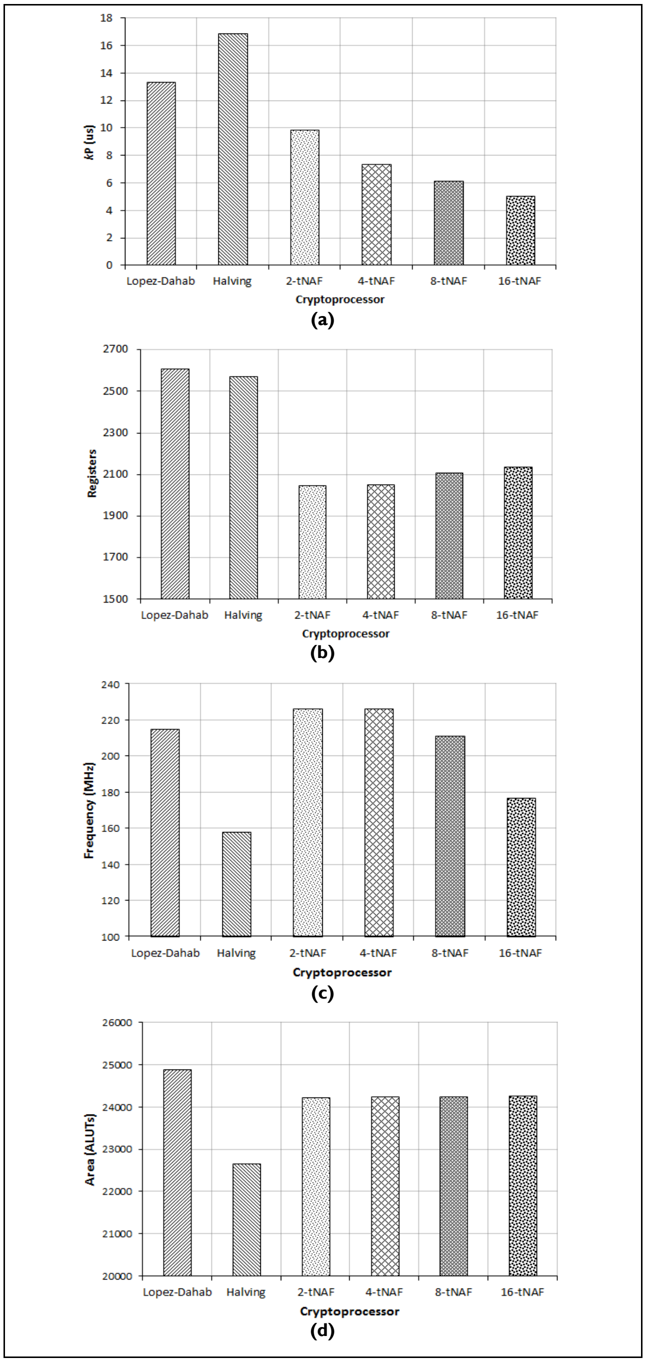

Figure 18. (a) Area resources. (b) Frequency. (c) Registers resources. (d) Time to perform the scalar multiplication of each cryptoprocessor.

From Figure 18, we can see that the *w*–τNAF cryptoprocessorwith *w* = 16 performs the scalar multiplication at a faster time (5.05 μs), and the halve-and-add processor with *w* = 2 uses fewer area resources than the other processors.

Comparison of the results with other works

To compare the performance of the designed cryptoprocessors with respect to the cryptoprocessors presented in the literature, Table 2 shows several design parameters and processing times, such as area resources, frequency, kP time and time-area product. However, it is important to mention that performing a fair comparison in hardware design is very difficult because there are other technical considerations, including the technologies, hardware platforms, software tools, scalar multiplication algorithms, finite field representations, and size of the fields.

From Table 2, it is possible to observe that the GF(2^{163}) cryptoprocessor presented in Mahadizadeh et al (2013) requires less time to perform the scalar multiplication than our processor based on the Lopez-Dahab algorithm because the first processor uses three digit-level multipliers, and our design uses two digit-level multipliers, and the latency to compute M_{add} and M_{double} is 3$\underline{M}$. However, the first processor requires more area than our processor. Mercurio et al (2006) computes kP by using the half-and-add algorithm, m=163, polynomial bases representation and one parallel multiplier. Our processor requires more area than the mentioned processor because it uses two digit-level multipliers, but our design requires less time to perform the scalar multiplication, and the latency to compute the point addition is 5$\underline{M}$. Finally, our processor is based on the Koblitz curves and has a higher performance (area and time) than the processor presented in Azarderakhsh (2013) because our design has a latency of 5$\underline{M}$ to compute the point addition, and it uses two digit-level multipliers and a window method that allows us to reduce the amount of point addition operations.

Table 2. Performance comparison results

Design	FPGA	Area	Freq. (MHz)	kP (μs)	T×A
(Trujillo and Velasco, 2010)	Stratix III	18567 ALUTs	97.51	60	1.11
(Malik, 2010)	TMS320	-	160	63400	-
(Sandoval et al., 2011)	Virtex 4	4586 LUTs	100	1070	4.90
(Nabil et al., 2012)	Xilinx	2502 LUTs	213	254	0.63
(Mercurio et al., 2006)	Virtex E	11616 LUTs	41	25.0	0.29
(Azarderakhsh *et al.*, 2013)	Stratix II	46168 ALUTs	188.7	9.15	0.42
(Mahdizadeh *et al.*, 2013)	Virtex 4	33414 LUTs	250	9.60	0.32
This work Lopez-Dahab	Stratix IV	24882 ALUTs	215.3	13.37	0.33
This work Halving	Stratix IV	22670 ALUTs	158.2	16.90	0.38
This work 2-τNAF	Stratix IV	24223 ALUTs	226.6	9.88	0.23
This work 4-τNAF	Stratix IV	24257 ALUTs	226.7	7.37	0.17
This work 8-τNAF	Stratix IV	24249 ALUTs	211.6	6.17	0.14
This work 16-τNAF	Stratix IV	24270 ALUTs	177.1	5.05	0.12

From Table 2, it is possible to observe that the GF(2^{163}) cryptoprocessor presented in Mahadizadeh et al (2013) requires less time to perform the scalar multiplication than our processor based on the Lopez-Dahab algorithm because the first processor uses three digit-level multipliers, and our design uses two digit-level multipliers, and the latency to compute M_{add} and M_{double} is 3$\underline{M}$. However, the first processor requires more area than our processor. Mercurio et al (2006) computes kP by using the half-and-add algorithm, m=163, polynomial bases representation and one parallel multiplier. Our processor requires more area than the mentioned processor because it uses two digit-level multipliers, but our design requires less time to perform the scalar multiplication, and the latency to compute the point addition is 5$\underline{M}$. Finally,

our processor is based on the Koblitz curves and has a higher performance (area and time) than the processor presented in Azarderakhsh (2013) because our design has a latency of 5M to compute the point addition, and it uses two digit-level multipliers and a window method that allows us to reduce the amount of point addition operations.

Conclusions

This work presents the design of elliptic curve cryptoprocessors to compute the scalar multiplication over GF(2^{163}) using the GNB.

The Lopez-Dahab, halve-and-add and w–τNAF algorithms are used to design the cryptoprocessors, which are described using generic structural VHDL, synthesized on the Stratix IV FPGA (EP4SGX180HF35C2).

Considering the hardware verification results, the 16-τNAF cryptoprocessor performs the scalar multiplication in less time (5.05 μs), and the 2-NAF halve-and-add cryptoprocessor uses fewer area resources than the other processors, in this case, 22670 ALUTs. All the cryptoprocessors use roughly 17% of the ALUTs of the FPGA.

Additionally, it is important to mention that the algorithms are synthetized on the same hardware platform using Quartus II, are simulated in Modelsim, and are verified using SignalTAP and Matlab; the cryptoprocessors use two digit-level finite field multipliers over GF(2^{163}) in the GNB; the expansion coefficients for the private key k are obtained using the software Maple; and the FSMs use a data dependence graph to perform kP to achieve the minimal states.

Future work will be oriented to increase the performance of the designed cryptoprocessors and the hardware implementation of the GF(2^{233}) processors. Additionally, new cryptoprocessors will be designed based on elliptic curves that are not included in the National Institute of Standards and Technology (NIST), such as the Hessian and Edwards curves that perform the scalar multiplication kP.

References

Amara, M., & Siad, A. (2011). Hardware implementation of arithmetic for elliptic curve cryptosystems over GF(2^m). In *World Congress on Internet Security (WorldCIS)* (pp. 73-78). London: IEEE.

Azarderakhsh, R., & Masoleh, R. (2010). *A Modified Low Complexity Digit-Level Gaussian Normal Basis Multiplier. Arithmetic of Finite Fields* (pp. 25-40). Turkey: Springer.

Azarderakhsh, R., & Masoleh, R. (2013). High-performance implementation of point multiplication on Koblitz Curves. *IEEE Transactions on Circuits and Systems*, *60*(1), 41 - 45.

Chester, R., & Mukhopadhyay, D. (2008). *Progress in Cryptology - INDOCRYPT 2008*. Kharagpur: Springer.

Cui, X.-N., & Yang, J. (2012). An FPGA based processor for Elliptic Curve Cryptography. In *International Conference on Computer Science and Information Processing (CSIP)* (pp. 343-349). Shaanxi: IEEE.

Ghanmy, N., Khlif, N., Fourati, L., & Kamoun, L. (2012). Hardware implementation of elliptic curve digital signature algorithm ECDSA on Koblitz curves. In *International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)* (pp. 1 - 6). Poznan: IEEE.

Hankerson, D., Menezes, A., & Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer.

Huang, T., Chang, C., Chiou, C., & Tan, S. (2011). Non-XOR approach for low-cost bit-parallel polynomial basis multiplier over GF(2^m). *Information Security, IET*, *5*(3) 152-162.

IEEE std 1363. (2000). *1363-2000 IEEE Standard Specifications for Public-Key Cryptography*. IEEE Computer Society.

Itoh, T., & Tsujii, S. (1988). A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computation*, *78*(3), 171-177.

Jeevananthan, S., & Muthukumar, B. (2010). High speed hardware implementation of an elliptic curve cryptography (ECC) co-processor. In Trendz in Information Sciences & Computing (TISC) (pp. 176-180). Chennai: IEEE.

Johnson, D., Menezes, A., & Vastone, S. (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, *1*(1), 36-63.

Juhas, T. (2007). *The Use of Elliptic Curves in Cryptography*. Retrieved from: http://munin.uit.no/bitstream/handle/10037/1091/thesis.pdf?sequence=5

Knudsen, W. (1999). *Elliptic Scalar Multiplication Using Point Halving. Advances in Cryptology - ASIACRYPT* (pp. 135-149). Berlin: Springer.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, *48*(1987), 203-209.

Lai, J.-Y., Hung, T.-Y., Yang, K.-H., & Huang, C.-T. (2010). *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 3933 - 3936). Paris: IEEE.

Lee, C., & Chiou, C. (2012). Scalable Gaussian Normal Basis Multipliers over GF(2^m) Using Hankel Matrix-Vector Representation. *Journal of Signal Processing Systems*, *69*(2), 197-211.

Lopez, J., & Dahab, R. (1999). Fast Multiplication on Elliptic Curves Over GF(2^m) without precomputation. *Cryptographic Hardware and Embedded Systems*, *1717*, 316-327.

Mahdizadeh, H., & Masoumi, M. (2013). Novel Architecture for efficient FPGA implementation of elliptic curve cryptographic processor over GF(2^163). *IEEE transactions on very large scale integration (VLSI) systems*, *21*(12), 1-4.

Malik, M. (2010). Efficient implementation of Elliptic Curve Cryptography using low-power Digital Signal Processor. In International Conference on Advanced Communication Technology (ICACT) (pp. 1464-1468). Phoenix Park: IEEE.

Masoleh, R. (2006). Efficient algorithms and architectures for field multiplication using Gaussian normal basis. *IEEE Transactions on Computers*, *55*(1), 34-47.

Mercurio, S., & Rodriguez, F. (2006). *Elliptic Curve Scalar Multiplication using Point Halving on Reconfigurable Hardware Platforms* (pp. 1-5). Mexico: CiteSeerX.

Miller, V. (1986). Advances in Cryptology. In *CRYPTO '85 Proceedings*. Santa Barbara: Springer.

Morales, S., Uribe, F., & Badillo, A. (2011). A reconfigurable GF(2^m) elliptic curve cryptographic coprocessor. In *Southern Conference on Programmable Logic (SPL)* (pp. 209 - 214). Cordoba: IEEE.

NIST. (2013). *Digital Signature Standard*. Gaithersburg: Federal Information Processing Standards.

Rahuman, A., & Athisha, G. (2010). Reconfigurable architecture for elliptic curve cryptography. In *International Conference on Communication and Computational Intelligence (INCOCCI)* (pp. 461-466). Erode: IEEE.

Schroeppel, R. (2000). United States Patent No. EP1232602.

Solinas, J. (2000). Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, *19*(2-3), 195-249.

Trujillo, V., & Velasco, J. (2010). Hardware Architectures for Elliptic Curve Cryptoprocessors Using Polynomial and Gaussian Normal Basis over GF(2^233). *Lecture Notes in Computer Science*, *6480*, 79-103.

Wang, Z., & Fan, S. (2012). Efficient Montgomery-Based Semi-Systolic Multiplier for Even-Type GNB of GF(2^m). *IEEE Transactions on Computers*, *61*(3), 415-419.