cryptography

MDPI

Article

Lookup Table-Based Design of Scalar Multiplication for Elliptic Curve Cryptography

Yan-Duan Ning [1], Yan-Haw Chen [2], Cheng-Sin Shih [1] and Shao-I Chu [1,*]

[1] Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Kaohsiung 807618, Taiwan; m0915787182@gmail.com (Y.-D.N.); f110152145@nkust.edu.tw (C.-S.S.)
[2] Department of Information Engineering, I-Shou University, Kaohsiung 840301, Taiwan; yanchen@isu.edu.tw
* Correspondence: erwinchu@nkust.edu.tw

Abstract: This paper is aimed at using a lookup table method to improve the scalar multiplication performance of elliptic curve cryptography. The lookup table must be divided into two polynomials and requires two iterations of point doubling operation, for which negation operations are needed. It is well known that an inversion operation requires a lot of multiplication time. The advantage of this paper is that we are able to reduce one inverse element calculation for this problem and also improve the basic operations of finite fields through segmentation methods. If the normal basis method is used in the design of the inverse element operation, it must be converted to the normal basis through the standard basis. However, the conversion process requires a lot of matrix operations. Though the anti-element operation has good speed performance, it also increases the computational complexity. Using number theory and grouping methods will greatly improve the performance of inverse element operations. With application of the two-time point doubling operation in the hardware implementation, the developed approach reduces the computing time by 48% as compared with the conventional approach. The computational time of the scalar multiplication using the presented method is further improved by 67% over the traditional algorithm with only an area increase of 12%. Finally, the proposed lookup table-based technique can be utilized for software and hardware implementation, as the developed arithmetic operations are simple and are consistent in their execution.

Keywords: elliptic curve cryptography; scalar multiplication; finite field; point doubling

Citation: Ning, Y.-D.; Chen, Y.-H.; Shih, C.-S.; Chu, S.-I. Lookup Table-Based Design of Scalar Multiplication for Elliptic Curve Cryptography. *Cryptography* **2024**, *8*, 11. https://doi.org/10.3390/cryptography8010011

Academic Editor: Josef Pieprzyk

Received: 7 February 2024
Revised: 13 March 2024
Accepted: 14 March 2024
Published: 18 March 2024

1. Introduction

Elliptic curve cryptography (ECC) was presented by N. Koblitz and V. Miller [1,2] in 1986. The ECC is an encryption technique based on the discrete logarithm problem. The public key cryptographic primitives can be implemented by using the elliptic curves over finite fields to generate finite abelian groups. The ECC provides similar security to existing public key cryptography but via application of a smaller key. The ECC with a 160-bit key has the equivalent security level of Rivest–Shamir–Adleman (RSA) and digital signature algorithms (DSA) that require a 1024-bit key. The ECC is defined over prime finite fields GF(p) or binary finite fields GF(2^m). The ECC design over prime fields generally provides more robust resistance to side-channel attacks as compared with those over binary fields. However, the ECC design over binary fields has the carry-free feature and makes arithmetic operations more suitable for hardware implementation.

The authors of [3] presented a method for constructing dynamic lookup tables in order to realize multiplication over a finite field. The Euclidian algorithm can be used for the inverse operation at the expense of computation time. Algorithms for the point multiplication over the ECC have been investigated in many studies. The method of non-adjacent forms (NAF) was presented by Morain et al. [4] to record the scalar in a point multiplication with an aim of reducing nonzero bits and thus the number of point additions.

The authors of [5] used the representation of integers in binary form for simplification. Nowadays, most ECC techniques are performed over Koblitz curves [1,6]. Solinas [7] presented the Frobenius map method for data encryption. The projective and affine coordinates were combined in [8] to implement high-efficiency point addition and point doubling operations. There are no inverse operations involved in the exploitation of the coordinate transform techniques. Guo et al. developed [9] a scalar multiplication algorithm based on the step multi-base representation via point halving and the septuple formula to significantly reduce computational cost. The triple-based chain method was proposed in [10] to optimize the time usage in the elliptic curve cryptosystem. The authors of [11] presented a configurable ECC crypto-processor. The ECC operates over the prime field and is defined by the Weierstrass equation. This crypto-processor was verified on a Xilinx FPGA board.

The existing work on modular multipliers over binary finite fields GF(2^m) can be found in [12–14]. In [15], a speed-oriented architecture over the finite fields GF(2^{163}) and GF(2^{571}) were implemented by using a balanced quadratic multiplier with high operating frequency. Li et al. [16] exploited the Karatsuba–Ofman multiplier to a scalar multiplication over the finite fields over GF(2^{571}) and GF(2^{283}). To minimize the number of point additions without increasing the number of point doubling, the Radix-2^w arithmetic for scalar multiplication was proposed in [17]. The authors of [18] developed a new Montgomery point multiplication algorithm for large field-size ECC over GF(2^{571}) and GF(2^{409}) to optimize resource utilization efficiency. Recently, Zhang et al. [19] introduced a high-performance scalar multiplication architecture over binary fields. A low-latency window (LLW) algorithm was presented for hardware implementation to enhance security, as was an enhanced comb method for point addition and point doubling. The above methods in [15–19] transformed the process of scalar multiplication from an affine coordinate system to a projective coordinate system. These designs were implemented and verified on the FPGA boards.

Different from operations over the projective coordinate system, this paper focuses on the scalar multiplication over the affine system. The hardware architecture and circuit are synthesized for future ASIC implementation. In the affine coordinate system, point addition and point doubling require one modular inversion operation each time. Because the inversion over the finite field is the most time-consuming operation among all of the basic operations. Reducing the number of inverse operations is an important objective of our design. The main idea of this paper can be described as follows: The Fermat's little theorem [20] is flexibly used for the inverse operation, which controls the critical path easily, as hardware implementation is required. The Horner's rule is also innovatively exploited to improve the method in [5]. By the binary representation and grouping techniques for constructing lookup tables, the operations of point addition and point doubling are improved. A high-speed scalar multiplication is therefore achieved. The presented approach is applicable to the digital signature [21] for real-time mutual authentication. Finally, the proposed lookup table-based algorithms can be utilized for software and hardware implementations as the developed arithmetic operations are simple and consistent in their execution. From the perspective of the hardware design, the computational time of the scalar multiplication by the proposed method is reduced by 67% over the conventional algorithm. This is because the presented two-time point doubling is superior to the conventional method.

The rest of the paper is organized as follows: The finite field arithmetic is introduced in Section 2. Section 3 briefs the concepts of point addition and point doubling in ECC. The proposed algorithms for the finite field arithmetic is described in Section 4. Section 5 presents the new algorithm for scalar multiplication, which combines the methodology in Section 4. Section 6 concludes this paper.

2. Finite Field Arithmetic

The finite field is a set of finite elements of the field, also known as the Galois field (GF). Multiplication, addition, subtraction and division are defined and certain basic rules are satisfied.

The finite field GF(2) has two elements, with values of 0 and 1. The finite field GF(2) can be extended to GF(2^m) by a primitive polynomial. The finite field GF(2^m) has 2^melements with the values from 0 to $2^m - 1$. The primitive polynomial is an irreducible polynomial denoted as $F(x)$. The national institute of standards and technology (NIST) recommended the primitive polynomials that are applicable to different GF(2^m) finite fields in the public document FIPS 186-4 [22] issued in 2013, as shown in Table 1.

Table 1. NIST-recommended primitive polynomial.

Finite Field	Primitive Polynomial
$GF(2^{163})$	$F(x) = x^{163} + x^7 + x^6 + x^3 + 1$
$GF(2^{233})$	$F(x) = x^{233} + x^{74} + 1$
$GF(2^{283})$	$F(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
$GF(2^{409})$	$F(x) = x^{409} + x^{87} + 1$
$GF(2^{571})$	$F(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

2.1. Addition and Subtraction

The addition operation over the finite field is $\delta = \gamma + \beta$, where $\beta, \gamma, \delta \in GF(2^m)$. The element has m-bit representation in a vector form. The addition is based on the bitwise exclusive OR (XOR) operation. Consider an example of GF(2^3). Let $\gamma = (011)$ and $\beta = (110)$. The result of $\gamma \oplus \beta$ is (101). The subtraction operation is the same as the addition operation.

2.2. Finite Field Multiplication

Multiplication over the finite field is defined as $\delta = \gamma \cdot \beta$, where $\beta, \gamma, \delta \in GF(2^m)$. The element γ can be represented by the polynomial $A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0$, where $a_i \in \{0,\ 1\}$. The element β is expressed by the polynomial $B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_2x^2 + b_1x + b_0$, where $b_i \in \{0,\ 1\}$. The element δ is represented by $C(x) = c_{m-1}x^{m-1} + c_{m-2}x^{m-2} + \cdots + c_2x^2 + c_1x + c_0$, where $c_i \in \{0, 1\}$. The multiplication result $C(x)$ is the polynomial multiplication of $A(x)$ and $B(x)$ modulo $F(x)$, where $F(x)$ is the primitive polynomial. That is,

$$C(x) \equiv A(x)B(x)\ (mod\ F(x)) \tag{1}$$

Take the field GF(2^3) as an example. Let the elements γ and β be (011) and (110), respectively. The primitive polynomial is $F(x) = x^3 + x + 1$. The multiplication result of γ and β is obtained as (001).

2.3. Finite Field Division

The division operation is $\delta = \gamma/\beta$. It can be viewed as $\delta = \gamma \cdot \beta^{-1}$, where β^{-1} is the inverse element of β. The inverse element of an element can be obtained quickly by Fermat's little theorem. Let $A(x)$ represent the element of a finite field, briefly denoted as A. Using the fact that $A^{2^m-1} \equiv 1(mod\ F(x))$, we have $A^{2^m-2} \equiv A^{-1}(mod\ F(x))$. Furthermore,

$$\begin{aligned} A^{2^m-2} &= A^{2^1+2^2+\cdots+2^{m-2}+2^{m-1}} \\ &= A^{2^1} \cdot A^{2^2} \cdot \cdots \cdot A^{2^{m-2}} \cdot A^{2^{m-1}} \end{aligned} \tag{2}$$

The inverse element can be calculated efficiently.

3. Arithmetic in ECC

In 1985, Koblitz and Miller proposed the elliptic curve for public key cryptography, elliptic curve cryptography is used in data encryption and decryption, key agreement,

digital signature, etc. The definition of the elliptic curve E in the finite field $\mathrm{GF}(2^m)$ is expressed as $E(\mathrm{GF}(2^m))$ where m is a positive integer, as shown in (3).

$$E(\mathrm{GF}(2^m)) : y^2 + xy = x^3 + ax^2 + b, \tag{3}$$

where $a, b \in \mathrm{GF}(2^m)$ and $b \neq 0$. The point P of the elliptic curve E is defined as $P = (x_1, y_1)$, where $x_1 \in \mathrm{GF}(2^m)$ and $y_1 \in \mathrm{GF}(2^m)$. The inverse of P is $-P = (x_1, x_1 + y_1)$.

Koblitz proposed the elliptic curve E in 1991 as

$$E(\mathrm{GF}(2)) : y^2 + xy = x^3 + ax^2 + 1. \tag{4}$$

The cases of $a = 0$ or $a = 1$ refer to a Koblitz curve.

3.1. Point Addition

The point addition of P and Q is defined as $R = P \boxplus Q$, where $P \neq Q$. Let the points P and Q be $P = (x_1, y_1)$, and $Q = (x_2, y_2)$, respectively, where $x_1 \neq x_2$. The result of point addition R is $R = (x_3, y_3)$. The relationships among these are

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}, \tag{5}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \tag{6}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \tag{7}$$

where the symbol "+" in (5)–(7) is the addition over the finite field.

3.2. Point Doubling

The point doubling of P is defined as $R = 2P$, where $P = (x_1, y_1)$, $R = (x_3, y_3)$ and $x_1 \neq 0$. The operations of point doubling are

$$\lambda = x_1 + \frac{y_1}{x_1}, \tag{8}$$

$$x_3 = \lambda^2 + \lambda + a, \tag{9}$$

$$y_3 = x_1^2 + (\lambda + 1)x_3. \tag{10}$$

3.3. The Points at Infinite (O Points)

The elliptic curve E contains a O point in the finite field $\mathrm{GF}(2^m)$. The O point must comply with the following rules:

$$O = O + O \tag{11}$$

$$P = P + O = O + P \tag{12}$$

$$O = P + (-P) = (-P) + P \tag{13}$$

3.4. Elliptic Curve Scalar Multiplication

The elliptic curve scalar multiplication is to add a point P on elliptic curve K times. That is,

$$KP = \underbrace{P + P + P + \cdots + P + P}_{K}, \tag{14}$$

where K is a positive integer.

4. Proposed Lookup Table-Based Techniques for Arithmetic on GF(2^{163})

This paper focuses on the finite field GF(2^{163}).

4.1. Improved Multiplication Algorithm

Let $H(a_i, a_j) = a_i x + a_j$. By Horner's rule, $A(x)$ is expressed as

$$A(x) = \left(\begin{pmatrix} \cdots \begin{pmatrix} (a_{162}x + a_{161})x^2 \\ +(a_{160}x + a_{159}) \end{pmatrix} x^2 \\ +\cdots+(a_4 x + a_3) \end{pmatrix} x^2 \right) x + a_0. \quad (15)$$
$$+(a_2 x + a_1)$$

Furthermore, the multiplication of the polynomials $A(x)$ and $B(x)$ is re-written as

$$A(x)B(x) = \left(\begin{pmatrix} \cdots \begin{pmatrix} (a_{162}xB(x) + a_{161}B(x))x^2 \\ +(a_{160}xB(x) + a_{159}B(x)) \end{pmatrix} x^2 \\ +\cdots+(a_4 xB(x) + a_3 B(x)) \end{pmatrix} x^2 \right) x + a_0 B(x). \quad (16)$$
$$+(a_2 xB(x) + a_1 B(x))$$

Define d as the number of input items of H.

As $d = 1$, $H(a_i) = a_i$. In the case of $d = 2$, $H(a_i, a_j) = a_i x + a_j$. According to (16), one needs only $\left\lfloor \frac{163}{d} \right\rfloor$ executions to create an H table. The table of $H(a_i, a_j)$ for $d = 2$ is shown in Table 2.

Table 2. H Table.

Input		**Output**
$\{a_i, a_j\}$	$H(a_i, a_j) = a_i x + a_j$	$H(x) \cdot B(x) \bmod F(x)$
$\{0,\ 0\}$	0	0
$\{0,\ 1\}$	1	$B(x)$
$\{1,\ 0\}$	x	$x \cdot B(x) \bmod F(x)$
$\{1,\ 1\}$	$x+1$	$x \cdot B(x) \bmod F(x) + B(x)$

4.2. Lookup Table-Based Modulo Operation

Consider the multiplication of $A(x) \cdot x^q$. We have

$$A(x) \cdot x^q = \begin{pmatrix} a_{162}x^{162+q} + a_{161}x^{161+q} + \cdots \\ +a_2 x^{2+q} + a_1 x^{1+q} + a_0 x^q \end{pmatrix} mod\ F(x), \quad (17)$$

where $F(x) = x^{163} + x^7 + x^6 + x^3 + 1$. In (17), there are q terms ($a_{162}x^{162}, a_{161}x^{161}, \cdots, a_{163-q}x^{163-q}$) to perform a modulo $F(x)$ operation, because of the degree. As a result, we can establish a table with which to manage these q terms. The table is used to store the results of $\left(a_{162}x^{162} + a_{161}x^{161} + \cdots + a_{163-q}x^{163-q}\right)x^q\ mod\ F(x)$. Such a table is required when $q < 156$. The inputs of the table with q terms are $a_{162}, a_{161}, \cdots, a_{163-q}$. The output is $a_{162}x^{q-1} \cdot f + a_{161}x^{q-2} \cdot f + \cdots + a_{163-q} \cdot f$, where $f = x^7 + x^6 + x^3 + 1$. For $q = 2$, the M table is constructed as Table 3.

Table 3. M Table.

Input	**Output**
$M(a_{162}, a_{161})$	
$M(0,\ 0)$	0
$M(0,\ 1)$	f
$M(1,\ 0)$	$x \cdot f$
$M(1,\ 1)$	$x \cdot f + f$

The circuit of the finite field multiplication is depicted in Figure 1, where the H and M tables are included. It can also be observed that the H table dominates the critical path of the whole architecture. Note that two XOR elements and one multiplexer are required for constructing the H table as depicted in Figure 2. The M table includes one XOR logic and one multiplexer in Figure 3.

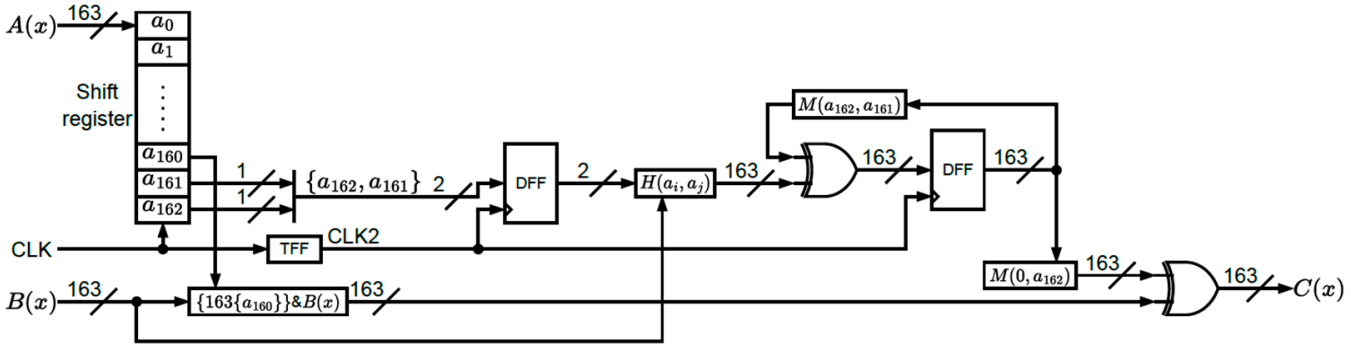

Figure 1. Multiplication circuit over GF(2^{163}) when $d = 2$.

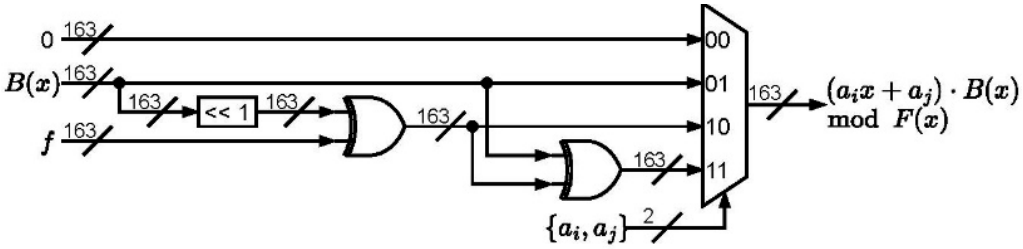

Figure 2. H table.

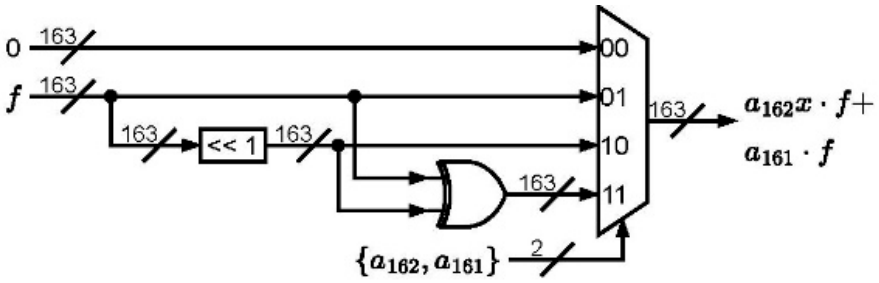

Figure 3. M table.

For the software implementation, there is more time to establish these tables and store the corresponding values. However, the building and searching time for the tables will increase as the tables become large. Additionally, in terms of hardware implementation, more logic elements are required.

4.3. Improved Squaring Operation

The squaring operation of $A(x)$ is

$$C(x) \equiv a_{162}x^{324} + a_{161}x^{322} + \cdots + a_1x^2 + a_0 \ mod \ F(x). \tag{18}$$

Consider modulo operations of each term as follows:

$$\begin{array}{c} A_{162}(x) \equiv a_{162}x^{324} \ mod \ F(x) \\ A_{161}(x) \equiv a_{161}x^{322} \ mod \ F(x) \\ \cdots \\ A_1(x) \equiv a_1x^2 \ mod \ F(x) \\ A_0(x) \equiv a_0 \ mod \ F(x). \end{array} \tag{19}$$

Let $A_i(x) = \sum_{j=0}^{162} \rho_{ij}x^j$, where $0 \leq i \leq 162$. In light of (19), we have

$$C(x) = \sum_{i=0}^{162} A_i(x) = \sum_{i=0}^{162} c_i(x), \tag{20}$$

where $c_i(x) = \theta_i x^i$.

4.4. Improved Inverse Operation

As shown in (12), we have

$$2^{163} - 2 = 2^{162} + 2^{161} + \cdots + 2^2 + 2^1 \tag{21}$$

(21) can be re-written as

$$2^{163} - 2 = \left(\begin{pmatrix} \cdots \begin{pmatrix} \left(\sum_{i=1}^{w} 2^i\right)2^w \\ +\left(\sum_{i=1}^{w} 2^i\right) \end{pmatrix} 2^w + \cdots \\ +\left(\sum_{i=1}^{w} 2^i\right) \end{pmatrix} 2^w \right) 2^l + sgn(l)\left(\sum_{i=1}^{l} 2^i\right), \tag{22}$$

where $l \equiv 162 \ mod \ w$ and $sgn(l)$ are sign function defined as follows:

$$sgn(l) = \begin{cases} 0 & , \ l = 0 \\ 1 & , \ l > 0 \end{cases} \tag{23}$$

Simplification of $\sum_{i=1}^{w} 2^i$ and $\sum_{i=1}^{l} 2^i$ in (22) is required.
As w is odd, we have

$$\sum_{i=1}^{w} 2^i = 2^w + \left(2^{w-1} + \cdots + 2^2 + 2^1\right). \tag{24}$$

As w is even, it is derived that

$$\sum_{i=1}^{w} 2^i = \left(2^{w/2} + 2^{w/2-1} + \cdots + 2^2 + 2^1\right)2^{w/2} + \left(2^{w/2} + 2^{w/2-1} + \cdots + 2^2 + 2^1\right) \tag{25}$$

According to (24) and (25), (22) is represented as

$$\begin{aligned} A^{2^m-2} &= A^{2^1+2^2+\cdots+2^{m-2}+2^{m-1}} \\ &= A^{\left(\begin{pmatrix} \cdots \begin{pmatrix} \left(\sum_{i=1}^{w} 2^i\right)2^w \\ +\left(\sum_{i=1}^{w} 2^i\right) \end{pmatrix} 2^w + \cdots \\ +\left(\sum_{i=1}^{w} 2^i\right) \end{pmatrix} 2^w \right)^{2^l + sgn(l)\left(\sum_{i=1}^{l} 2^i\right)}} \end{aligned} \tag{26}$$

The original Fermat's little theorem for the inverse element operation needs to go through 162 cycles of power operation and 161 cycles of multiplication operation. The main purpose of the above simplification process is to reduce the number of multiplication operations, but there is also a need to increase the number of different 2^n power operations. We investigate the number of multiplications for the inverse operation on the field GF(2^{163}) for various w. The results are depicted in Figure 4 for $w = 61$ to $w = 70$.

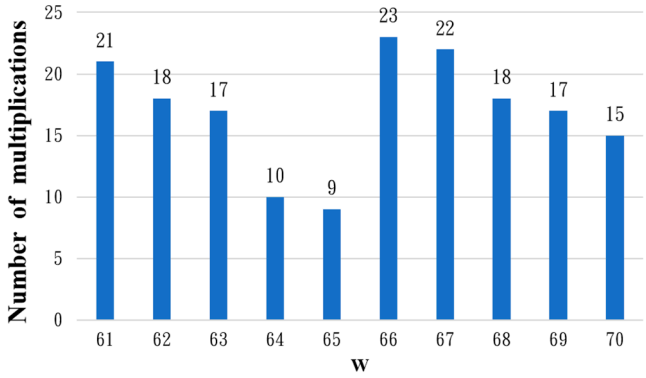

Figure 4. Number of multiplications for different w value groupings.

It can be seen in Figure 1 that grouping by $w = 65$ requires the lowest number of multiplications. In the following, the inverse operation of $GF(2^{163})$ and grouping by $w = 65$ are simplified, where $l = 32$, as shown in (27).

$$2^{163} - 2 = \left(\left(\sum_{i=1}^{65} 2^i\right)2^{65} + \left(\sum_{i=1}^{65} 2^i\right)\right)2^{32} + \left(\sum_{i=1}^{32} 2^i\right) \tag{27}$$

Simplify $\sum_{i=1}^{w} 2^i$ and $\sum_{i=1}^{l} 2^i$ in (27) and let $Z = 2^{64} + 2^{63} + \cdots + 2^2 + 2^1$. We have

$$\begin{aligned} \sum_{i=1}^{65} 2^i &= 2^{65} + \left(2^{64} + 2^{63} + \cdots + 2^2 + 2^1\right) \\ &= 2^{65} + Z. \end{aligned} \tag{28}$$

Using the idea, it is derived as follows:

$$\begin{aligned} Z = &\left(2^{32} + 2^{31} + \cdots + 2^2 + 2^1\right)2^{32} \\ &+\left(2^{32} + 2^{31} + \cdots + 2^2 + 2^1\right) \\ &= (Y)2^{32} + Y, \end{aligned} \tag{29}$$

where $Y = 2^{32} + 2^{31} + \cdots + 2^2 + 2^1$.

$$\begin{aligned} Y = &\left(2^{16} + 2^{15} + \cdots + 2^2 + 2^1\right)2^{16} \\ &+\left(2^{16} + 2^{15} + \cdots + 2^2 + 2^1\right) \\ &= (X)2^{16} + X, \end{aligned} \tag{30}$$

where $X = 2^{16} + 2^{15} + \cdots + 2^2 + 2^1$.

$$\begin{aligned} X = &\left(2^{8} + 2^{7} + \cdots + 2^2 + 2^1\right)2^{8} \\ &+\left(2^{8} + 2^{7} + \cdots + 2^2 + 2^1\right) \\ &= (W)2^{8} + W, \end{aligned} \tag{31}$$

where $W = 2^8 + 2^7 + \cdots + 2^2 + 2^1$.

$$\begin{aligned} W = &\left(2^{4} + 2^{3} + 2^2 + 2^1\right)2^{4} \\ &+\left(2^{4} + 2^{3} + 2^2 + 2^1\right) \\ &= (V)2^{4} + V, \end{aligned} \tag{32}$$

where $V = 2^4 + 2^3 + 2^2 + 2^1$.

$$\begin{aligned} V &= \left(2^2 + 2^1\right)2^2 + \left(2^2 + 2^1\right) \\ &= (U)2^2 + U, \end{aligned} \tag{33}$$

where $U = 2^2 + 2^1$.

Among these, Y is the same as the residue $\sum_{i=1}^{32} 2^i$, so there is no need to simplify the residue further, though it does need to store the value of Y after the operation. In this

case, where $w = 65$, the inverse operation over the field GF(2^{163}) requires 9 multiplication operations, 1 squaring operation and $2^2, 2^4, 2^8, 2^{16}, 2^{32}, 2^{65}$ power operations.

The circuit for inverse operation over GF(2^{163}) is shown in Figure 5. The GFM is the Galois field multiplication unit and the input Ci in the multiplexer means the ith cycle.

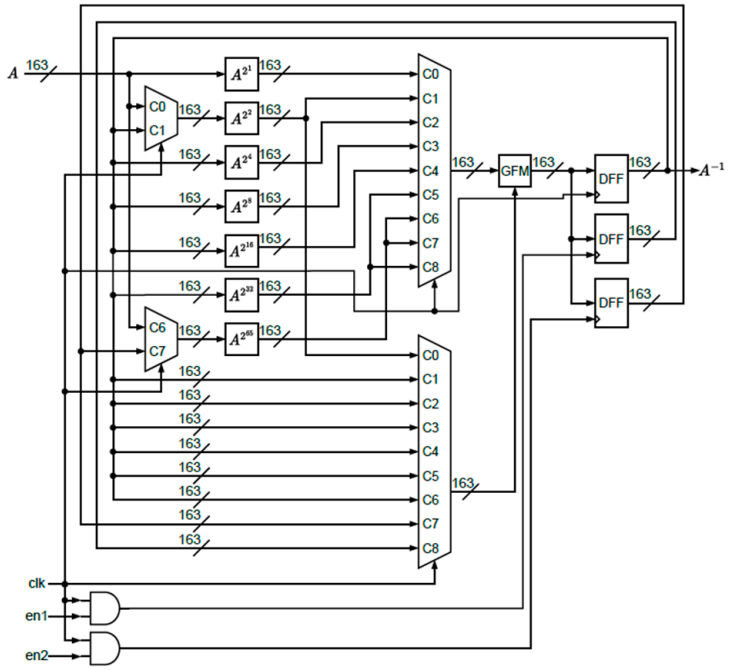

Figure 5. Proposed inversion circuit over GF(2^{163}) with $w = 65$.

5. Proposed Elliptic Curve Scalar Multiplication Operation

By building a table of elliptic curve points and using a table lookup method and an inverse operation that reduces the number of iterations of point doubling by two, the speed of scalar multiplication of elliptic curves is increased.

5.1. Conventional Multiple Time Point Doubling Algorithm

Consider Algorithm 1, called Point_Doubling_Repeating (P, r), where P is the point of the elliptic curve and r is a positive integer. If $r = 2$, the point of P will be subjected to a point doubling operation twice. The coordinate of P is (x_1, y_1) and the result of point doubling is (x_3, y_3).

Algorithm 1 Point Doubling Repeating

1: Function Point_Doubling_Repeating (P, r)
2: for i from 0 to $r - 1$ do
3: $\lambda = x_1 + \frac{y_1}{x_1}$
4: $x_3 = \lambda^2 + \lambda + a$
5: $y_3 = x_1^2 + (\lambda + 1)x_3$
6: $P \leftarrow P(x_3, y_3)$
7: $x_1 = x_3$
8: $y_1 = y_3$
9: end for
10: end function

The computational complexity of Algorithm 1 is listed in Table 4 with $r = 1$ and $r = 2$. It is observed that point doubling two times ($r = 2$) requires the inverse operation two times. In fact, k inverse operations are needed as $r = k$.

Table 4. Number of finite field operations required for each of the one-time and two-time point doubling operations.

r	$r = 1$	$r = 2$
Addition	5	10
Multiplication	2	4
Square	2	4
Inverse	1	2

5.2. Proposed Two-Time Point Doubling Algorithm

Algorithm 2, Point_Doubling_Modify (P, Sel), is presented, where the input of P point is $P(x_1, y_1)$. As $Sel = 0$, the point doubling is performed once and the output is $P(x_3, y_3)$. When $Sel = 1$, the improved algorithm for point doubling twice is executed and the output is $P(x_5, y_5)$.

Algorithm 2 Point Doubling Modify

Function Point_Doubling_Modify (P, Sel)
  if $Sel = 0$ then
    $\lambda = x_1 + \frac{y_1}{x_1}$
    $x_3 = \lambda^2 + \lambda + a$
    $y_3 = x_1^2 + (\lambda + 1)x_3$
    $P \leftarrow P(x_3, y_3)$
  else if $Sel = 1$ then
    $t_0 = x_1^2$
    $u_0 = t_0 + y_1$
    $t_1 = a \cdot t_0$
    $t_2 = (u_0 + x_1) \cdot u_0$
    $u_1 = t_1 + t_2$
    $t_3 = u_1^2$
    $t_4 = u_1 \cdot t_0$
    $t_5 = t_4^{-1}$
    $x_3 = t_3 \cdot t_5$
    $t_6 = (a + 1) \cdot t_4$
    $t_7 = u_0^2$
    $t_8 = t_7 \cdot u_1$
    $t_9 = t_0^2$
    $t_{10} = t_0 \cdot t_9$
    $u_2 = t_6 + t_8 + t_{10}$
    $\lambda = u_2 \cdot t_5$
    $t_{11} = \lambda^2$
    $x_5 = t_{11} + \lambda + a$
    $t_{12} = x_3^2$
    $t_{13} = (\lambda + 1) \cdot x_5$
    $y_5 = t_{12} + t_{13}$
    $P \leftarrow P(x_5, y_5)$
  end if
  return P
end function

The number of finite field operations required for improved two-time point doubling is shown in Table 5. It can be found that, after simplification, only one inverse operation is required.

Comparison with the conventional two-time point doubling is revealed in Table 6. Although the proposed two-time point doubling algorithm reduces the number of inverse operations by 1, it increases the number of multiplication operations by 5 and the number of squaring operations by 2.

Table 5. Number of finite field operations for improved two-time point doubling.

	Number of Operations
Addition	10
Multiplication	9
Square	6
Inverse	1

Table 6. Comparison of two-time point doubling.

	Conventional	Proposed
Addition	10	10
Multiplication	4	9
Square	4	6
Inverse	2	1

The proposed architecture of two-time point doubling is presented in Figure 6. The yellow block is used to perform one-time point doubling and the red block is for computing two-time point doubling.

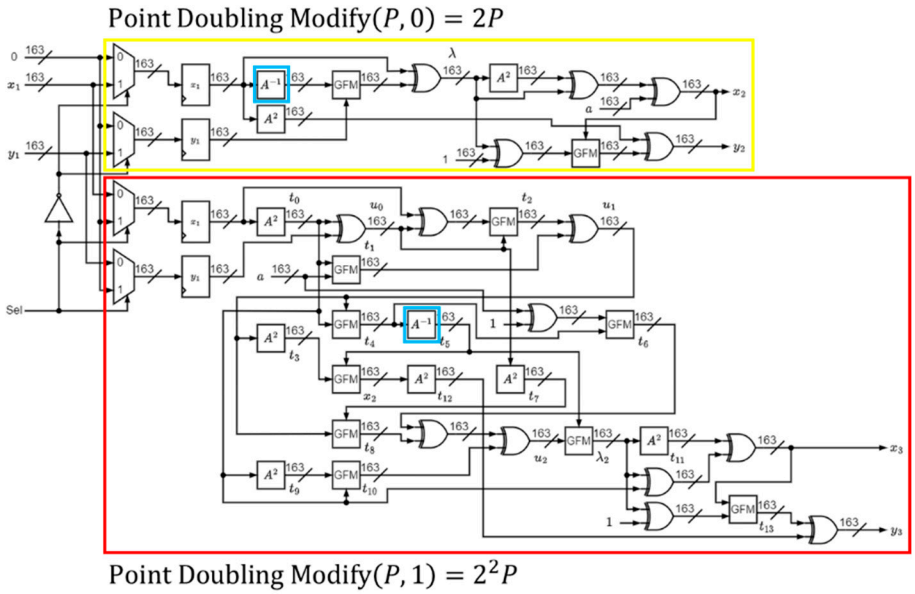

Figure 6. Proposed circuit for two-time point doubling.

The inverse operation utilizes the framework in Figure 5.

5.3. Proposed Elliptic Curve Scalar Multiplication Algorithm

The elliptic curve scalar multiplication operation is defined in (14). The positive integer K is expressed as follows in the binary form.

$$K = k_{m-1}2^{m-1} + k_{m-2}2^{m-2} + \cdots + k_2 2^2 + k_1 2^1 + k_0 2^0, \tag{34}$$

where $k_i \in \{0,\ 1\}$.

If m is odd, K is re-written as

$$K = \left(\cdots \begin{pmatrix} (k_{m-1}2 + k_{m-2})2^2 \\ +(k_{m-3}2 + k_{m-4}) \end{pmatrix} 2^2 + \cdots \right) 2 + k_0 \tag{35}$$

by Horner's rule. Moreover,

$$KP = \left(\cdots \left(\begin{matrix} (k_{m-1}2P + k_{m-2}P)2^2 \\ +(k_{m-3}2P + k_{m-4}P) \end{matrix} \right) 2^2 + \cdots \begin{matrix} \\ +(k_2 2P + k_1 P) \end{matrix} \right) 2 + k_0 P. \tag{36}$$

It is observed in (36) that the last term is k_0P and that the common term is expressed as $k_i2P + k_jP$. As a result, the L table is motivated and constructed. The L table is listed as Table 7, where the input is (k_i, k_j). As when one builds the L table, the operations for point addition and doubling once need to be performed first. There are an additional four 163-bit registers required for the table.

Table 7. L Table.

$L(k_i, k_j)$	$k_i2P + k_jP$
$L(0,\ 0)$	0
$L(0,\ 1)$	P
$L(1,\ 0)$	$2P$
$L(1,\ 1)$	$2P + P$

The term of $(k_i2P + k_jP)$ can be viewed as a point of the elliptic curve. This term requires point doubling once, as $k_i = 1$. Furthermore, $(k_i2P + k_jP)2^2$ is the operation of two-time point doubling. According to (36), a new scalar multiplication technique is proposed in Algorithm 3.

Algorithm 3 New Scalar Multiplication 1

1: Function New_ScalarM (K, P)
2: $Q = (0, 0)$
3: To make a lookup table $L(k_i, k_j)$ as shown in Table 7
4: for i from $\lfloor \frac{m}{2} \rfloor - 1$ to 0 do
5: $Q \leftarrow$ Point_Doubling_Repeating$(Q, 2)$
6: $Q \leftarrow$ Point_Addition $(Q, L(k_{2i+2}, k_{2i+1}))$
7: end for
8: $Q \leftarrow$ Point_Doubling_Repeating$(Q, 1)$
9: $Q \leftarrow$ Point_Addition$(Q, L(0, k_0))$
10: return Q
11: end function

We replace Point_Doubling_Repeating(P, r) with Point_Doubling_Modify(P, Sel) in Algorithm 3. Algorithm 4 is therefore obtained.

Algorithm 4 New Scalar Multiplication 2

1: Function New_ScalarM (K, P)
2: $Q = (0, 0)$
3: To make a lookup table $L(k_i, k_j)$ as shown in Table 7
4: for i from $\lfloor \frac{m}{2} \rfloor - 1$ to 0 do
5: $Q \leftarrow$ Point_Doubling_Modify$(Q, 1)$
6: $Q \leftarrow$ Point_Addition$(Q, L(k_{2i+2}, k_{2i+1}))$
7: end for
8: $Q \leftarrow$ Point_Doubling_Modify$(Q, 0)$
9: $Q \leftarrow$ Point_Addition$(Q, L(0, k_0))$
10: return Q
11: end function

The numbers of arithmetic operations required in Algorithms 3 and 4 are demonstrated in Table 8. The proposed framework of the scalar multiplication by Algorithm 4 is depicted in Figure 7, where Algorithm 2 is applied for point doubling. Note that using the improved two-time point doubling reduces the number of inverse operations by 81. However, the numbers of multiplication and squaring operations will increase by 405 and 162, respectively.

Table 8. Number of arithmetic operations in Algorithms 3 and 4.

Operation	Number
Number of Point Addition	83
Number of Point Doubling	2
Two-time Point Doubling	81

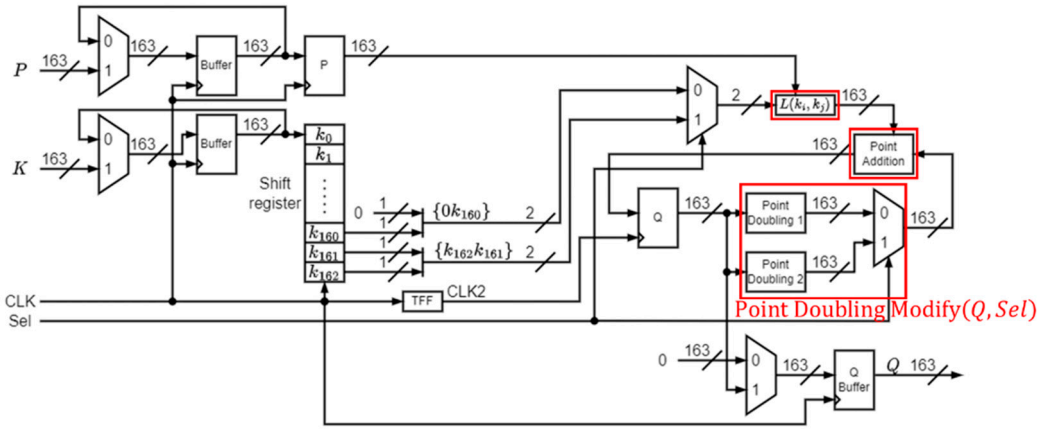

Figure 7. Proposed circuit for scalar multiplication by Algorithm 4.

5.4. Performance Evaluation

The results of multiplication over the finite field GF(2^{163}) are obtained after computing 163 coefficients. Utilization of the H table by Horner's rule only needs computation of $\left\lfloor \frac{163}{d} \right\rfloor$ coefficients. The M table is exploited in order to speed up the modulo operation by the XOR operation. The inverse operation using Fermat's little theorem is obtained by the simplifications of $\sum_{i=1}^{w} 2^i$ and $\sum_{i=1}^{l} 2^i$. The results of $A^{\sum_{i=1}^{w} 2^i}$ and $A^{\sum_{i=1}^{l} 2^i}$ are stored in advance. Thus, this leads to improvements in performance of the inverse operation.

The most time-consuming operation over the finite field is the inverse operation. Exploiting Horner's rule to compute the inverse requires nine multiplication and one squaring operation. The conventional two-time point doubling needs two inverse operations. However, the proposed method requires only one inverse operation at the expense of five multiplication and two squaring operations. Finally, the proposed algorithm reduces four multiplications and increase two squaring operations. Elliptic curve point multiplication KP is improved by using the idea that the integer K is expressed in the binary form with Horner's rule-based grouping technique. The L table is established to further enhance the efficiency of the scalar multiplication.

The efficiency evaluations for the proposed hardware design are listed in Tables 9–12. All circuits are synthesized with Taiwan Semiconductor Manufacturing Company (TSMC) 40 nm standard cell library by Synopsys Design Compiler

Table 9. Multiplication designs with various *d*.

d	Critical Path (ns)	Frequency (MHz)	Area (μm^2)	Total Power (mW)	Cycle
1	1.44	694.44	4761.44	3.39	164
2	0.86	1162.79	7705.08	4.53	84
3	0.95	1052.63	8750.62	5.03	57
4	0.95	1052.63	10482.24	6.00	43

Table 10. Squaring circuits of 2^n.

2^n	Critical Path (ns)	Area (μm^2)	Total Power (mW)	Cycle
2^1	0.18	355.6	0.38	1
2^2	0.4	709.2	1.09	1
2^4	0.82	1980.6	4.00	1
2^8	1.11	5736.1	14.06	1
2^{16}	1.22	6569.3	16.44	1
2^{32}	1.22	6652.3	16.63	1
2^{64}	1.25	6624.6	16.65	1

Table 11. Architecture comparison for two-time point doubling.

Two-Time Doubling	Critical Path (ns)	Area (μm^2)	Total Power (mW)	Cycle	Delay (ns)
Conventional	2.64	123,197.08	62.15	1795	4738.8
Proposed	2.65	147,547.68	75.11	924	2448.6

Table 12. Architecture comparison for scalar multiplication.

	Critical Path (ns)	Area (μm^2)	Total Power (mW)	Cycle	Delay (ns)
Algorithm 3	2.76	393,205.19	194.23	464,476	1,281,953.76
Algorithm 4	2.78	441,985.79	219.44	150,925	419,571.5

Table 9 indicates the synthesized results of the proposed multiplication method, with various d values. It is observed that the shortest critical path happens as $d = 2$. For the proposed squaring circuits, Table 10 shows that the critical path becomes short with the increase of n. However, the improvements are not significant from the cases of 2^{16} to 2^{65}. Table 11 reveals that the proposed two-time point doubling has a 48% shorter computing time when compared with the conventional one, and with low area–time complexity. Take the architectures of scalar multiplication into consideration in Table 12. The delay of Algorithm 4 is improved by 67% over Algorithm 3, with an area increase of only 12%.

6. Conclusions

This paper proposed a methodology and hardware architecture for the scalar multiplication in the affine coordinate system over ECC. Our main idea is to exploit Horner's rule to efficiently construct the lookup tables for arithmetic operations over the finite field. The time-consuming part in such a system is the inverse operation. The new algorithm for two-time point doubling was therefore developed by reducing the number of inverse operations. Results of the hardware implementations show that the computing time of two-time point doubling is reduced by 48% over the conventional one, with an area increase of only 18%. For the scalar multiplication, including the point addition and doubling operations, the presented hardware architecture has a reduction of delay by 68% as compared

with the traditional algorithm. The total power and circuit area increase by only 13% and 12%, respectively. Such results reveal that the proposed lookup table-based design indeed achieves a high-speed performance by reducing the computational complexity of point doubling. However, the current reported hardware designs are still involved in plenty of registers. The size of the lookup tables may be reduced by some merging techniques. These issues will be investigated and addressed for compact design in the future.

Author Contributions: Conceptualization, Y.-D.N. and Y.-H.C.; methodology, Y.-H.C.; software, C.-S.S.; validation, Y.-H.C. and C.-S.S.; formal analysis, Y.-D.N.; investigation, Y.-H.C. and C.-S.S.; writing—original draft preparation, Y.-D.N. and S.-I.C.; writing—review and editing, Y.-D.N. and S.-I.C.; visualization, C.-S.S.; supervision, Y.-H.C. and S.-I.C.; project administration, Y.-H.C.; funding acquisition, Y.-H.C. All authors have read and agreed to the published version of the manuscript.

Funding: The funding sources are the Ministry of Science and Technology, Taiwan (under Grant no. MOST 111-2221-E-214-014).

Data Availability Statement: The data is available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Koblitz, N. Elliptic curve crypto systems. *Math. Comput.* **1987**, *48*, 203–309. [CrossRef]
2. Miller, V. Uses of elliptic curves in cryptography. In *Advances in Cryptography-CRYPTO*; Springer: Berlin/Heidelberg, Germany, 1986; pp. 417–426.
3. Chen, Y.H.; Huang, C.H. Efficient Operations in Large Finite Fields for Elliptic Curve Cryptographic. *Int. J. Eng. Technol. Manag. Res.* **2020**, *7*, 141–151. [CrossRef]
4. Morain, F.; Olivos, J. Speeding Up the Computations on An Elliptic Curve Using Addition-Subtraction Chains. *RAIRO Theor. Inform. Appl.* **1990**, *24*, 531–543. [CrossRef]
5. Solinas, J.A. *Low-Weight Binary Representation for Pairs of Integers*; Combinatorics and Optimization Research Report CORR 2001-41; Technical Report; Centre for Applied Cryptographic Research, University of Waterloo: Waterloo, ON, Canada, 2001.
6. Koblitz, N. CM-Curves with Good Cryptographic Properties. In *Advances in Cryptology-CRYPTO '91*; Feigenbaum, J., Ed.; Springer: Berlin/Heidelberg, Germany, 1992; pp. 279–287.
7. Solinas, J.A. Efficient Arithmetic on Koblitz Curves. In *Towards a Quarter-Century of Public Key Cryptography: A Special Issue of Designs, Codes and Cryptography an International Journal*; Koblitz, N., Ed.; Springer: New York, NY, USA, 2000; Volume 19, pp. 125–179.
8. Lange, T. *A Note On López-Dahab Coordinates*; IACR Cryptology ePrint Archive, Paper 2004/323; 2004.
9. Guo, C.; Gong, B. Efficient Scalar Multiplication of ECC Using SMBR and Fast Septuple Formula for IoT. *EURASIP J. Wirel. Commun. Netw.* **2021**, *2021*, 82. [CrossRef]
10. Cho, S.M.; Gwak, S.G.; Kim, C.H.; Hong, S. Faster Elliptic Curve Arithmetic for Triple-base Chain by Reordering Sequences of Field Operations. *Multimed. Tools Appl.* **2016**, *75*, 14819–14831. [CrossRef]
11. Di Matteo, S.; Baldanzi, L.; Crocetti, L.; Nannipieri, P.; Fanucci, L.; Saponara, S. Secure Elliptic Curve Crypto-Processor for Real-Time IoT Applications. *Energies* **2021**, *14*, 4676. [CrossRef]
12. Imran, M.; Rashid, M. Architectural review of polynomial bases finite field multipliers over GF(2 m). In Proceedings of the International Conference on Communication, Computing and Digital Systems (C-CODE), Islamabad, Pakistan, 8–9 March 2017; pp. 331–336.
13. Khan, S.; Javeed, K.; Shah, Y.A. High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications. *Microprocess. Microsyst.* **2018**, *62*, 91–101. [CrossRef]
14. SPillutla, R.; Boppana, L. A high-throughput fully digit-serial polynomial basis finite field GF(2 m) multiplier for IoT applications. In Proceedings of the TENCON 2019—2019 IEEE Region 10 Conference (TENCON), Kochi, India, 17–20 October 2019; pp. 920–924.
15. Li, J.; Zhong, S.; Li, Z.; Cao, S.; Zhang, J.; Wang, W. Speed-Oriented Architecture for Binary Field Point Multiplication on Elliptic Curves. *IEEE Access* **2019**, *7*, 32048–32060. [CrossRef]
16. Li, J.; Wang, W.; Zhang, J.; Luo, Y.; Ren, S. Innovative Dual-Binary-Field Architecture for Point Multiplication of Elliptic Curve Cryptography. *IEEE Access* **2021**, *9*, 12405–12419. [CrossRef]
17. Oudjida, A.K.; Liacha, A. Radix-2w Arithmetic for Scalar Multiplication in Elliptic Curve Cryptography. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 1979–1989. [CrossRef]
18. Lee, C.-Y.; Zeghid, M.; Sghaier, A.; Ahmed, H.Y.; Xie, J. Efficient Hardware Implementation of Large Field-Size Elliptic Curve Cryptographic Processor. *IEEE Access* **2022**, *10*, 7926–7936. [CrossRef]
19. Zhang, J.; Chen, Z.; Ma, M.; Jiang, R.; Li, H.; Wang, W. High-Performance ECC Scalar Multiplication Architecture Based on Comb Method and Low-Latency Window Recoding Algorithm. *IEEE Trans. Very Large Scale Integr. Syst.* **2024**, *32*, 382–395. [CrossRef]
20. Wang, C.C.; Truong, T.K.; Shao, H.M.; Deutsch, L.J. VLSI Architectures for Computing Multiplications and Inverses in GF(2 m). *IEEE Trans. Comput.* **1985**, *100*, 709–717. [CrossRef] [PubMed]

21. National Institute of Standards and Technology. *Digital Signature Standard (DSS)*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023.
22. National Institute of Standards and Technology. *Digital Signature Standard (DSS)*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2013. [CrossRef]