applied sciences

MDPI

*Article*

# Large Field-Size Elliptic Curve Processor for Area-Constrained Applications

**Muhammad Rashid [1,*], Omar S. Sonbul [1], Muhammad Yousuf Irfan Zia [2,3,*], Nadeem Kafi [4], Mohammed H. Sinky [1] and Muhammad Arif [5]**

1 Computer Engineering Department, Umm Al Qura University, Makkah 21955, Saudi Arabia
2 Department of Electrical Engineering, Ziauddin University, Karachi 74600, Pakistan
3 Telecommunications Engineering School, University of Malaga, 29010 Málaga, Spain
4 Department of Computer Science, National University of Computer and Emerging Sciences, Karachi 75030, Pakistan
5 Computer Science Department, Umm Al Qura University, Makkah 21955, Saudi Arabia
* Correspondence: mfelahi@uqu.edu.sa (M.R.); yousuf.irfan@zu.edu.pk or yirfanzia@uma.es (M.Y.I.Z.)

**Abstract:** This article has proposed an efficient area-optimized elliptic curve cryptographic processor architecture over $GF(2^{409})$ and $GF(2^{571})$. The proposed architecture employs Lopez-Dahab projective point arithmetic operations. To do this, a hybrid Karatsuba multiplier of 4-split polynomials is proposed. The proposed multiplier uses general Karatsuba and traditional schoolbook multiplication approaches. Moreover, the multiplier resources are reused to implement the modular squares and addition chains of the Itoh-Tsujii algorithm for inverse computations. The reuse of resources reduces the overall area requirements. The implementation is performed in Verilog (HDL). The achieved results are provided on Xilinx Virtex 7 device. In addition, the performance of the proposed design is evaluated on ASIC 65 nm process technology. Consequently, a figure-of-merit is constructed to compare the FPGA and ASIC implementations. An exhaustive comparison to existing designs in the literature shows that the proposed architecture utilizes less area. Therefore, the proposed design is the right choice for area-constrained cryptographic applications.

**Keywords:** hardware accelerator; elliptic curve cryptography; crypto processor; FPGA; ASIC



**Citation:** Rashid, M.; Sonbul, O.S.; Zia, M.Y.I.; Kafi, N.; Sinky, M.H.; Arif, M. Large Field-Size Elliptic Curve Processor for Area-Constrained Applications. *Appl. Sci.* **2023**, *13*, 1240. https://doi.org/10.3390/app13031240

Academic Editor: Arcangelo Castiglione

Received: 27 November 2022
Revised: 9 January 2023
Accepted: 13 January 2023
Published: 17 January 2023



## 1. Introduction

The public-key cryptographic standards (ECC [1] and RSA [2]) are vulnerable to quantum computers when running a Shor's algorithm [3,4]. In 2022, the National Institute of Standards and Technology (NIST) has standardized new quantum-secure key encapsulation and digital signature algorithms [5]. However, the NIST-recommended quantum algorithms are not reasonable for securing area constrained cryptographic applications such as wireless sensor nodes (WSNs) [6,7], radio-frequency-identification-networks (RFID) [8,9], automotive vehicles [10], secure robotics communication [11], etc. The reason is the huge hardware resource requirements due to several hash computations, binomial and gaussian samplers for introducing noise in the algorithm(s), and many more [3]. Many researchers are predicting that 10–15 years are needed for the widespread deployment of quantum computers [3,12]. Based on this compliance, this article realizes a low-complexity hardware accelerator of present public-key algorithms over large key sizes for area-constrained cryptographic applications.

As compared to RSA, ECC ensures equivalent security with shorter key lengths. For example, 233 bits of ECC are required to achieve identical security to a 2048-bit RSA [2]. The shorter key length assures additional benefits such as low area, low power consumption, a lower channel bandwidth requirement for data transmitting over the unsecured channel, etc. Therefore, ECC is an appropriate public-key cryptographic algorithm for area-constraint devices and applications. The NIST has standardized elliptic curve parameters for prime, i.e., $GF(P)$, and binary, i.e., $GF(2^m)$, fields [13]. The NIST-specified lengths are

different for the prime field (192, 224, 256, 384, and 521) and binary field (163, 233, 283, 409 and 571). Furthermore, the prime field has been frequently targeted for software-based implementations [14]. On the other hand, the binary field is preferred for field-programmable gate arrays (FPGA) and application-specific integrated circuits (ASIC) [15–25].

In addition to key length and the selection of a particular field, the choice between polynomial and normal basis representations is also important. These representations are required to implement the finite field modular operations (addition, multiplication, square, and inversion). The polynomial basis provides efficient modular multiplications while the normal basis is more suitable for modular square implementations [14,26]. Furthermore, ECC offers two coordinate representations (affine and projective) to execute the required cryptographic operations. In comparison, the affine coordinate representation is more expensive in throughput as it needs an inversion operation [14,17,21,25]. Therefore, we have preferred projective coordinates to implement $GF(2^{409})$ and $GF(2^{571})$ fields with polynomial basis representations for our computations in this work. The reason behind the selection of large field lengths is to consider higher security as the higher key lengths specify the higher security.

### 1.1. Related Work

Elliptic curve point multiplication (ECPM) is the most critical part (from a computation point of view) in elliptic curves. The EPCM further depends on the computation of two more operations. These operations are point addition (PA) and point double (PD). The PA and PD computations depend on the execution of finite field addition, square, multiplication and inverse operations. Different hardware accelerators have been described in the literature to optimize the ECPM implementation. Nevertheless, the most recent architectures are described in [15–25].

A highly efficient architecture for ECPM computation on Koblitz curves is proposed in [15] over $GF(2^{163})$ to $GF(2^{571})$. The selected PM algorithm allows the parallel processing of Frobenius maps and point additions. Moreover, to minimize the computational time and improve clock frequency, the scheduling of point addition operations is performed. A pipelined multiplier accumulator unit is used. Finally, the inversion is based on efficient addition chains for the reduction of clock cycles. In [16], a 6CC-6CC (clock cycle) dual-field ECPM architecture is proposed. A Karatsuba multiplier is utilized to compute efficient polynomial multiplications. The scheduling of addition operations minimizes the total number of required cycles. The results are provided on Xilinx Virtex 5 and Virtex 7 FPGA devices.

An efficient throughput over area ECC processor over binary elliptic curves is proposed in [17]. A segmented digit-serial finite field multiplier is utilized to reduce hardware resources. Moreover, the scheduling of addition operations reduces latency (i.e., the computational time for PM execution). The results are synthesized on Virtex 7 FPGA device over $GF(2^{163})$ to $GF(2^{571})$ fields. Another throughput over area efficient ECC processor is presented in [18]. The area is reduced by employing only a singular finite field multiplier. A 41-bit digit-parallel multiplier is used to compute finite field squares and multiplication operations. Throughput is optimized using a 2-stage pipelining and rescheduling of PA and PD computations. Over $GF(2^{163})$ to $GF(2^{571})$ fields, the implementations are provided on Virtex 7 and modern 16nm ASIC technology.

The authentication of IoT (Internet of Things) devices is generally performed using a Physical Unclonable Function (PUF). Therefore, the leakage of PUF responses in an authentication system reduces its privacy and security. To improve its security, an ECC-based PM algorithm with different key lengths is used in [19]. The hardware resources are reduced by reusing the underlying finite field computing units. Their circuit achieves a maximum of 135 MHz frequency on Virtex 6 device. The single ECPM computation takes 19.33, 22.36, 41.36 and 56.5 µ for 233, 283, 409 and 571 key lengths respectively.

The Weierstrass model of ECC is implemented in [15–19]. Nevertheless, these designs are vulnerable to side-channel attacks. Consequently, in order to prevent these side-channel

attacks, Edward and Huff curve-based PM architectures have been described in [20–25]. The Edward and Huff curves (special models of ECC) provide unified PA and PD operations to execute the PM operation in ECC. On Xilinx FPGA devices, Edward curve architecture over $GF(P)$ with $P = 256$ is implemented in [20,21]. Similarly, the FPGA implemented Edward curve designs over $GF(2^{233})$ is described in [22,23]. The 4-stage pipelined architectures of Huff curves over $GF(2^{233})$ on FPGA is implemented in [24,25].

### *1.2. Research Gap*

Various architectural techniques such as pipelining, parallelism, and different flavor of finite field multipliers, i.e., digit-serial, digit-parallel, and bit-parallel, etc., have been utilized to optimize the performance of ECPM operation in [15–25]. However, due to their higher area complexity, these architectures are not suitable for applications related to secure robotics communication, WSNs, automotive vehicles, RFID, etc. Therefore, the objective of this work is to provide a low-complexity hardware accelerator over a large ECC field size.

### *1.3. Contributions*

The contributions are given as follows:

- We have presented an area-optimized hardware accelerator of a large field-size elliptic curve point multiplication processor over binary $GF(2^{409})$ and $GF(2^{571})$ fields. The design details are described in Section 3.
- Towards the area reduction, we have first proposed a hybrid polynomial multiplier architecture for efficient polynomial multiplications. Subsequently, we have utilized the proposed polynomial multiplier to operate modular square and addition chains of the Itoh-Tsujii algorithm [27] for modular inverse computation. A hybrid approach in our polynomial multiplier design is achieved in two steps: (i) 4-level polynomial splits for multiplying large polynomial operands using a general Karatsuba multiplier and (ii) traditional schoolbook multiplication for smaller polynomial multiplications of length 35 and 36 bits. The details of the hybrid approach are presented in Section 3.3.
- An efficient finite-state-machine (FSM) based controller is implemented to provide read/write operations from/to utilized memory block. Moreover, the implemented FSM (also) generates the corresponding control signals for the routing networks (i.e., multiplexer(s) and demultiplexer(s)). We have provided the relevant details in Section 3.4.

For validation, the Verilog model of the proposed architecture is simulated and synthesized Vivado IDE tool. We have provided implementation results on Xilinx Virtex 7 FPGA and 65 nm ASIC technologies. The hardware resource utilization of the proposed architecture on ASIC is 30,561 $\mu m^2$ and 36,857 $\mu m^2$ over $GF(2^{409})$ and $GF(2^{571})$, respectively. Similarly, the FPGA slicesares 4439 and 5683 over $GF(2^{409})$ and $GF(2^{571})$, respectively. For $GF(2^{409})$ and $GF(2^{571})$, the clock cycle requirements of the proposed design are 13,903 and 20,551, respectively. The proposed architecture over $GF(2^{409})$ operates on a maximum frequency of 1800 MHz (for ASIC) and 357 MHz (for FPGA). Similarly, the proposed architecture over $GF(2^{571})$ operates on a maximum frequency of 1450 MHz (for ASIC) and 317 MHz (for FPGA). The time for one PM computation on ASIC is 7.72 µs and 14.17 µs over $GF(2^{409})$ and $GF(2^{571})$, respectively. Similarly, the time for one PM computation on FPGA is 38.94 µs and 64.82 µs over $GF(2^{409})$ and $GF(2^{571})$, respectively.

This article is organized as: Section 2 provides the required mathematical knowledge to implement the ECC PM operation. The proposed crypto processor architecture design is described in Section 3. The implementation results and comparisons are given in Section 4. Finally, Section 5 concludes the article.

## 2. ECC Background

In the case of $GF(2^m)$ field, an affine form of ECC can be represented as a set of points, i.e., $x$, and $y$, which obeys the Equation (1) as follows:

$$E : y^2 + xy = x^3 + ax^2 + b \tag{1}$$

In Equation (1), $x$ and $y$ are the base elements of the $GF(2^m)$ field. Points $a$ and $b$ represent curve parameters with $b \neq 0$. The representation of $x$ and $y$ elements of the $GF(2^m)$ field in a triplet, i.e., $X$, $Y$, and $Z$, is projective coordinates. Thus, for $GF(2^m)$ field, a Lopez Dahab projective form of Equation (1) is represented in Equation (2). It can be seen in Equation (2) that there are three variables ($X$, $Y$ and $Z$). These variables are the projective elements of point $P(X : Y : Z)$, such that $Z \neq 0$, $a$ and $b$ are the curve constants with $b \neq 0$.

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

The points on the elliptic curve form an additive group. According to the additive group definition, the addition of two elements in a group defines another element in the group. For instance, we have two points, i.e., $P$ and $Q$, on an elliptic curve. Then a point addition will be $R = P + Q$, where $R$ will be the resultant point on the curve. Similarly, when both the points on the curve are the same ($P + P = 2P$) then this is called a point double. These point addition and double operations are needed to compute a point multiplication operation in ECC. Consequently, the addition of $k$ copies of point addition and double determines the ECPM and can be calculated using Equation (3).

$$Q = kP \tag{3}$$

The initial point (in Equation (3)) on the curve is $P$. The resultant point on the curve is $Q$. Similarly, $k$ is the scalar multiplier. To implement Equation (3), we have several algorithms in the literature such as Double and Add, Lopez Dahab, Montgomery, etc. Comparatively, the Double and Add algorithm is more suitable for the unified models of ECC, such as Edwards, Huff, Twisted Edwards, etc. Similarly, the Lopez Dahab algorithm is a better choice for achieving instruction-level parallelism (i.e., for performance improvements). However, similar instructions for the computation of point addition and double make the Montgomery point multiplication algorithm suitable for the side-channel resistant implementation of ECC. A detailed comparison of various ECPM algorithms is presented in [28]. To summarize, we have used the Montgomery point multiplication algorithm (given in Algorithm 1) in our implementation. It is important to mention that we are not considering the side channel attacks at the design level. In other words, we have implemented only the side channel attack-protected point multiplication algorithm on FPGA and ASIC platforms.

The inputs to Algorithm 1 are an initial point $P$ with $x$ and $y$ coordinates (i.e., $x_p$, and $y_p$), and a scalar multiplier $k$. The bit stream $k_{n-1}, \ldots, k_1, k_0$ presents the scalar multiplier. The outputs of the Algorithm 1 are the $x$ and $y$ coordinates of the final point on the ECC curve. For $if$ and $else$ statements, the corresponding sequence of instructions for point addition and double operations are given in Table 1. Columns one and column two show the point addition information in terms of the sequence of instructions (i.e., $Inst_i$) as well as the corresponding operations. Similarly, columns three and column four show the number of instructions (i.e., $Inst_i$) as well as the respective finite field operations for point double computations. It can be observed from Table 1 that there are 14 instructions (in total) for point addition and double computations. 6 of these 14 instructions perform finite field multiplications. Similarly, 3 of these 14 instructions perform finite field additions. Finally, the remaining 5 instructions perform finite field squaring.

**Algorithm 1:** Montgomery PM Algorithm [17,26]

**Input:** $k = (k_{n-1}, \ldots, k_1, k_0)$ with $k_{n-1} = 1$, $P = (x_p, y_p) \in GF(2^m)$
**Output:** $Q = (x_q, y_q) = k \cdot P$
**Affine to projective conversions:** $X_1 = x_p$, $Z_1 = 1$, $Z_2 = x_p^2$ and $X_2 = x_p^4 + b$
**PM in projective coordinates:**
**for** *(i from* $m-2$ *down to 0)* **do**
  **if** $(k_i = 1)$ **then**
    $PA(X_1, Z_1) = (X_1, Z_1, X_2, Z_2)$ and $PD(X_2, Z_2) = (X_2, Z_2)$
  **else**
    $PA(X_2, Z_2) = (X_2, Z_2, X_1, Z_1)$ and $PD(X_1, Z_1) = (X_1, Z_1)$
    **end if**
  **end for**
**Reconversion from projective to affine:** $x_q = \frac{X_1}{Z_1}$, $y_q = x_p + (\frac{X_1}{Z_1})[(X_1 + x_p \times Z_1)(X_2 + x_p \times Z_2) + (x_p^2 + y_p)(Z_1 \times Z_2)](x_p \times Z_1 \times Z_2)^{-1} + y_p$

**Table 1.** Point addition and double instructions.

| $\mathbf{Inst_i}$ | **Point Addition (PA)** | $\mathbf{Inst_i}$ | **Point Double (PD)** |
|---|---|---|---|
| $Inst_1$ | $Z_1 = X_2 \times Z_1$ | $Inst_1$ | $Z_2 = Z_2^2$ |
| $Inst_2$ | $X_1 = X_1 \times Z_2$ | $Inst_2$ | $T_1 = Z_2^2$ |
| $Inst_3$ | $T_1 = X_1 + Z_1$ | $Inst_3$ | $T_1 = b \times T_1$ |
| $Inst_4$ | $X_1 = X_1 \times Z_1$ | $Inst_4$ | $X_2 = X_2^2$ |
| $Inst_5$ | $Z_1 = T_1^2$ | $Inst_5$ | $Z_2 = X_2 \times Z_2$ |
| $Inst_6$ | $T_1 = x_p \times Z_1$ | $Inst_6$ | $X_2 = X_2^2$ |
| $Inst_7$ | $X_1 = X_1 + T_1$ | $Inst_7$ | $X_2 = X_2 + T_1$ |

## 3. Proposed Crypto Processor Architecture

Figure 1 shows the block diagram of our proposed elliptic curve point multiplication architecture. It includes data buffers, a memory block, an arithmetic unit and a control unit. These essential components of our architecture are highlighted with different colors in Figure 1. The data buffers contain input parameters loaded from the outside of the accelerator circuit. The memory block stores the intermediate and the final results after the computation. The arithmetic unit of the proposed accelerator executes the modular operations (addition, square, multiplication and inversion) of layer one of ECC. The control unit generates the corresponding control signals to implement the elliptic-curve PM operation. More insight details are given in the succeeding sections.

### *3.1. Data Buffers*

The orange color in Figure 1 contains four $m$-bit buffers and a $3 \times 1$ multiplexer $M1$. The $x_p$ and $y_p$ buffers contain the $x$ and $y$ coordinate of initial point $P$. A buffer $b$ holds an $m$-bit elliptic curve constant. The *KeyReg* buffer keeps an $m$-bit scalar multiplier. Whereas $m$ determines the scalar multiplier length (i.e., $k$). All these parameters ($x_p$, $y_p$, $b$ and *KeyReg*) are input to our proposed architecture. The corresponding values for $x_p$, $y_p$ and $b$ are selected from a standardized document by NIST.

### *3.2. RegFile*

The yellow color in Figure 1 shows the memory unit. It consists of an $8 \times m$ register array (RegFile). The purpose of this unit is to store the initial, intermediate and final output. It contains two $8 \times 1$ multiplexers and one $1 \times 8$ demultiplexer. The objective of (RegFile) multiplexers is to read two $m$-bit operands for further computations. The demultiplexer is utilized to update the value of the specified register. Finally, for read/write operations, the control signals (shown with red color lines in Figure 1) are generated by the control unit.

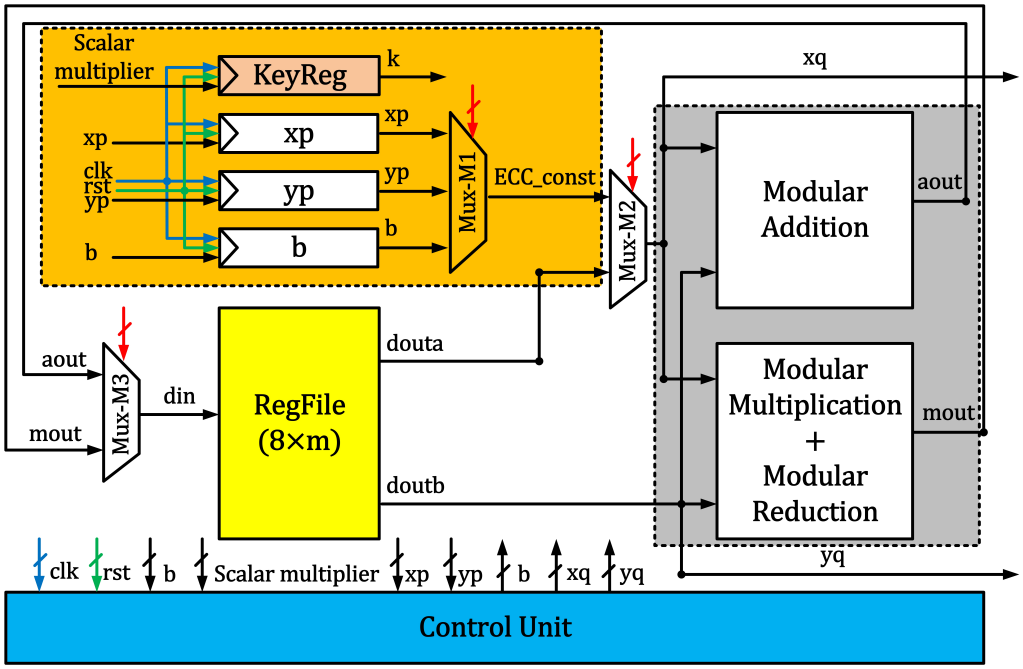


**Figure 1.** Block diagram of our large field-size elliptic curve cryptographic processor in $GF(2^m)$.

### *3.3. Arithmetic Unit*

The gray highlighted portion in Figure 1 shows the arithmetic unit. It contains an $m$-bit adder and an $m$-bit multiplier. The adder unit is simply implemented by the bitwise exclusive (OR) operation. For two $m$-bit operands, $m$ exclusive (OR) gates are needed.

#### Proposed Hybrid Karatsuba Modular Multiplier

We have performed modular multiplication using a hybrid Karatsuba multiplier. A hybrid approach is achieved by combining Karatsuba and traditional school book multipliers. In order to completely describe our proposed multiplier architecture, we first present the polynomial splits over $GF(2^{571})$ for the Karatsuba multiplier in Figure 2. It can be observed from Figure 2, the input polynomials of length 571-bits are partitioned into two pieces, i.e., 285 and 286. Then, in level two, the split 285-bit polynomial is further divided into two more parts of lengths 142 and 143. We repeat the splitting process until to reach the 35-bit and 36-bit polynomials. Afterward, the multiplication over polynomial lengths of 35-bit and 36-bit is acquired using a traditional schoolbook multiplier. The proposed hybrid modular multiplier architecture is shown in Figure 3.

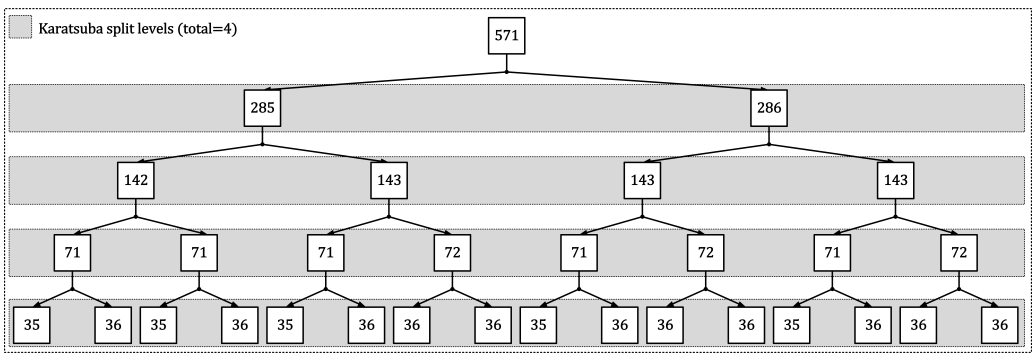


**Figure 2.** Polynomial splits of a Karatsuba modular multiplier over $GF(2^{571})$.

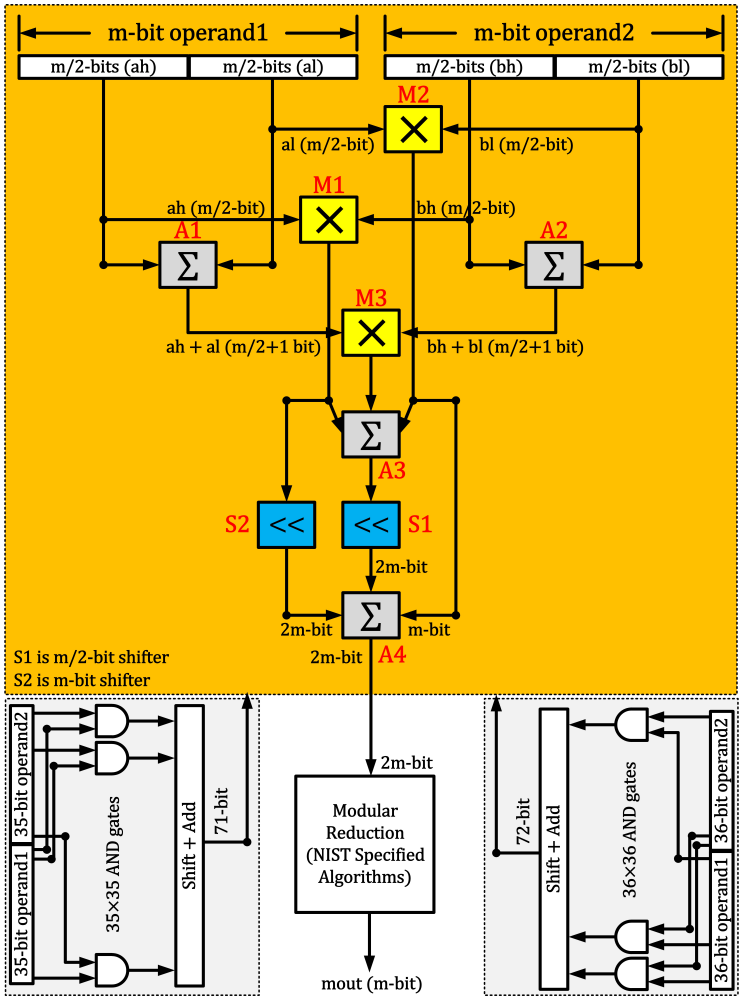


**Figure 3.** Proposed Karatsuba modular multiplier.

The orange color portion in Figure 3 determines the Karatsuba multiplication while the gray highlighted regions are for traditional schoolbook multiplication. The proposed hybrid Karatsuba multiplier operates in a way that the schoolbook multiplier performs first to generate the polynomial multiplication result over smaller polynomial lengths. Then, the outputs of the schoolbook multiplier will be transferred to the Karatsuba multiplier to accomplish the required polynomial multiplications over 409 and 571 bits. The area cost of the schoolbook multiplier is 35 AND gates to generate a partial product over 35-bit polynomials. The overall cost includes $35 \times 35$ (meaning that 35 partial products are needed and the length of each partial product is 35-bits). Some OR gates are needed to generate the resultant product of 71-bits length. Similarly, the area cost of the schoolbook multiplier is 36 AND gates to generate a partial product over 36-bit polynomials. The overall cost will be $36 \times 36$. It implies that a total of 36 partial products are needed and each partial product contains a 36-bits length. The final product of 72-bit length is generated using some OR gates. The outputs of schoolbook multipliers is 71 (for $35 \times 35$) and 72 (for $36 \times 36$) bits. These 71 and 72-bit outputs are input to the Karatsuba multiplier for generating multiplications over longer polynomials than 35 and 36 bits.

The Karatsuba multiplier takes two $m$-bit operands $a$ and $b$ as input and divides them into two $\frac{m}{2}$ parts. The split parts of polynomial $a$ are $ah$ and $al$. Similarly, the split parts of polynomial $b$ are $bh$ and $bl$. Three multipliers are needed to execute the inner multiplications [29]. The M1 multiplies $ah$ and $bh$ whereas M2 multiplies $al$ and $bl$. The multiplier M3 is responsible to multiply the outputs generated after adders, i.e., A1, and A2. Apart from the inner multiplications, the addition of split polynomials is performed. The first adder (i.e., A1) is used to add $ah$ and $al$. Similarly, the second adder (i.e., A2) adds the

*bh* and *bl* polynomials. As seen in Figure 3, the adder A3 adds the multiplication results produced by multipliers M1, M2 and M3. Two shifters, i.e., S1, and S2, are needed to shift polynomials by $\frac{m}{2}$ and $m$ bit towards the right. Finally, an adder A4 generates the resultant polynomial of length $2 \times m$ and it goes to the reduction block as input to generate an $m$-bit output. To perform polynomial reduction over $GF(2^{409})$ and $GF(2^{571})$, we have implemented NIST-specified algorithms as described in Algorithms 2 and 3, respectively. In our work, these reduction algorithms are implemented like combinational circuit because these require only shift and exclusive (OR) operations.

**Algorithm 2:** NIST reduction over $GF(2^{409})$ [14]

**Input:** Polynomial, $D(x)$ with $2 \times m - 1$-bit length
**Output:** Polynomial, $D(x)$ with $m$-bit length
1. *for* ($i$ *from* 25 *down to* 13) *do*
   - 1.1 $T \longleftarrow D[i]$
   - 1.2 $D[i-13] \longleftarrow D[i-13] \oplus (T \ll 7)$
   - 1.3 $D[i-12] \longleftarrow D[i-12] \oplus (T \gg 25)$
   - 1.4 $D[i-11] \longleftarrow D[i-11] \oplus (T \ll 30)$
   - 1.5 $D[i-10] \longleftarrow D[i-10] \oplus (T \gg 2)$
2. $T \longleftarrow D[12] \gg 25$
3. $D[0] \longleftarrow D[0] \oplus T$
4. $D[2] \longleftarrow D[2] \oplus (T \text{ !! } 23)$
5. $D[12] \longleftarrow D[12]$ & 0x1FFFFFF
6. *Return* $(D[12], \ldots, D[3], D[2], D[1], D[0])$

**Algorithm 3:** NIST reduction over $GF(2^{571})$ [14]

**Input:** Polynomial, $D(x)$ with $2 \times m - 1$-bit length
**Output:** Polynomial, $D(x)$ with $m$-bit length
1. *for* ($i$ *from* 35 *down to* 18) *do*
   - 1.1 $T \longleftarrow D[i]$
   - 1.2 $D[i-18] \longleftarrow D[i-18] \oplus (T \ll 5) \oplus (T \ll 7) \oplus (T \ll 10) \oplus (T \ll 15)$
   - 1.3 $D[i-17] \longleftarrow D[i-17] \oplus (T \gg 27) \oplus (T \gg 25) \oplus (T \gg 22) \oplus (T \gg 17)$
2. $T \longleftarrow D[17] \gg 27$
3. $D[0] \longleftarrow D[0] \oplus T \oplus (T \ll 2) \oplus (T \ll 5) \oplus (T \ll 10)$
4. $D[17] \longleftarrow D[17]$ & 0x7FFFFFFF
5. *Return* $(D[17], \ldots, D[3], D[2], D[1], D[0])$

The reconversion in Algorithm 1 requires modular inversion operation. In the existing literature, we have several algorithms to execute the modular inversion over $GF(2^{409})$ and $GF(2^{571})$. Our implementation executes the inversion operation using the square version of Itoh-Tsujii algorithm [27]. It requires $m-1$ squares followed by 10 & 11 multiplications over $GF(2^{409})$ and $GF(2^{571})$, where $m$ is the operand length. From a hardware implementation perspective, there are two possibilities to implement modular inversion: (1) using a separate square and a multiplier block and (2) using existing hardware resources of a modular multiplier. The first method is inconvenient as it utilizes more hardware resources and consumes more power. The second approach is appealing to save the implementation resources and keep low power consumption of the circuit [15,25,26].

The instructions in Table 1 contain square, multiplication and addition operations. It is important to note that we have used only a single modular multiplier to perform both square and modular multiplications. The square is computed by providing two similar operands as inputs to the multiplier, as described in [26]. Similarly, the Itoh-Tsujii inversion

algorithm is implemented using the existing hardware resources of our proposed modular multiplier, as implemented using hardware resources of different multipliers in [25,26]. The proposed multiplier architecture is implemented as combinational logic and it requires only one clock cycle (including reduction) to perform one polynomial square and multiplication.

### 3.4. Control Unit and Clock Cycles Calculation

An FSM-based dedicated controller is used to implement Algorithm 1 for PM computation. Therefore, during each stage of the FSM, the corresponding read/write signals are generated for the RegFile unit. Moreover, the dedicated controller is (also) responsible to generate the corresponding control signals for routing multiplexers. Using the proposed architecture, the affine to projective conversions of Algorithm 1 take 4 clock cycles. The $for$ loop (or point multiplication in projective coordinates) in Algorithm 1 requires 14 instructions of Table 1. Out of these 14 instructions, 7 are for point addition and the other 7 are for point double. All these 14 instructions require addition, multiplication and square operations. Therefore, the computational cost of the $for$ loop of Algorithm 1 is $14(m-2)$ clock cycles, where $m$ is the implemented key length.

For $GF(2^{409})$ and $GF(2^{571})$, the clock cycle requirement is 5698 and 7966, respectively. The reconversions from projective to affine of Algorithm 1 require two modular inversion operations. As we have mentioned before, the inversion needs $m-1$ squares followed by 10 and 11 field multiplications. In addition, the cycle count for each square and multiplication computation is one. Therefore, the total cycle count for each inversion over $GF(2^{409})$ and $GF(2^{571})$ is 4080 and 6270 respectively. The total cost for inversion (in terms of clock cycles) over $GF(2^{409})$ and $GF(2^{571})$ is 8160 and 12,540 respectively. The 41 cycles are needed to calculate the remaining computations of reconversion in Algorithm 1. Consequently, the total number of clock cycles (to implement Algorithm 1 over $GF(2^{409})$ and $GF(2^{571})$) are 13,903 and 20,551 respectively.

## 4. Implementation Results and Performance Comparison

The register-transfer-level (RTL) implementation of our proposed crypto processor architecture over $GF(2^{409})$ and $GF(2^{571})$ is implemented in Verilog HDL using a Vivado IDE. For targeted binary field lengths, i.e., 409 and 571, the initial point $P$ and a curve constant value $b$ as inputs are selected from NIST standardized document [13]. Therefore, we describe implementation results and our comparison to state-of-the-art in Sections 4.1 and 4.2, respectively.

### 4.1. Results

The implementation results on FPGA and ASIC devices are shown in Table 2 and Table 3, respectively. For FPGA implementations, we have provided results on Xilinx Virtex 7 (a modern 28 nm technology) device. Moreover, the synthesis results for ASIC is provided on a commercial 65 nm process technology. Concerning Table 2, the targeted field length $m$ is shown in column one. Columns two to four provide the area results in slices, look-up-tables (LUTs) and flip-flops (FFs). The timing information in terms of clock cycles, frequency (Freq), latency and throughput (Thrpt) are given in columns five to eight. A figure-of-merit (FoM) is defined in terms of a ratio of throughput over slices to evaluate the performance of the proposed crypto processor. Similarly, considering Table 3, the targeted field length $m$ is shown in column one. Column two shows the area (in $mm^2$) results. The timing information in terms of clock cycles, frequency (Freq), latency and throughput (Thrpt) are given in columns three to six. A FoM is also defined in terms of a ratio of throughput over area. For FPGA and ASIC implementations, the area (slices, LUTs, FFs for FPGA) and frequency values are collected from the synthesis tool (Vivado IDE for FPGA and Cadence Genus for ASIC). The clock cycles calculation is already described in Section 3.4. Similarly, the latency and throughput values are calculated using Equation (4)

and Equation (5), respectively. Finally, the figure-of-merit values are calculated using Equation (6).

$$Latency\ (\mu s) = \frac{Clock\ cycles}{Clock\ frequency\ (\text{MHz})} \tag{4}$$

$$Throughput\ (Thrpt) = \frac{1}{Latency\ (\mu s)} = \frac{10^6}{Latency} \tag{5}$$

$$FoM = \frac{Throughput\ (Thrpt)}{Area\ (slices\ for\ FPGA)} \tag{6}$$

**Table 2.** Implementation results on Xilinx (Virtex 7) FPGA.

| *m* | Slices | LUTs | FFs | Clock Cycles | Freq (MHz) | Latency (µs) | Thrpt | FoM |
|---|---|---|---|---|---|---|---|---|
| 409 | 4439 | 12,568 | 4129 | 13,903 | 357 | 38.94 | 25.68 kbps | 5.78 |
| 571 | 5683 | 14,356 | 5961 | 20,551 | 317 | 64.82 | 15.42 kbps | 1.07 |

**Table 3.** Synthesis results on (65 nm) ASIC.

| *m* | Area ($\mu m^2$) | Clock Cycles | Freq (MHz) | Latency (µs) | Thrpt | FoM |
|---|---|---|---|---|---|---|
| 409 | 30,561 | 13,903 | 1800 | 7.72 | 129.53 kbps | 4238.40 |
| 571 | 36,857 | 20,551 | 1450 | 14.17 | 70.57 kbps | 1914.69 |

*Comparison of $GF(2^{409})$ & $GF(2^{571})$ on FPGA.* Table 2 demonstrates that the increase in the field length increases hardware resources (slices, LUTs and FFs). For example, the slices, LUTs and FFs for $GF(2^{409})$ field are 0.78 (ratio of 4439 over 5683), 0.87 (ratio of 12,568 over 14,356) and 0.69 (ratio of 4129 over 5961) times lower than $GF(2^{571})$ field. Similarly, the computational cost to execute one PM operation on $GF(2^{409})$ field is 0.67 (ratio of 13,903 over 20,551) times lower than $GF(2^{571})$ field. Moreover, on a modern Virtex 7 FPGA device, our proposed architecture can operate on a maximum 357 MHz and 317 MHz clock frequency over $GF(2^{409})$ and $GF(2^{571})$ fields, respectively. As for as the computational cost for one PM execution is concerned, our proposed architecture over $GF(2^{409})$ and $GF(2^{571})$ require 38.94 µs and 64.82 µs, respectively. This indicates that the 0.71 (ratio of 409 over 571) times increase in the field length (i.e., from $GF(2^{409})$ to $GF(2^{571})$) results in 0.60 (ratio of 38.94 over 64.82) times increase in computation time. If we calculate the ratio of increase in the field length (0.71) with the increase in computation time (0.60), the proposed accelerator architecture over a larger $GF(2^{571})$ field length is more efficient than the architecture over $GF(2^{409})$ field. As shown in column eight of Table 2, the throughput of our proposed accelerator design over $GF(2^{409})$ and $GF(2^{571})$ is 25.68 kbps and 15.42 kbs, respectively. Concerning our defined FoM for comparison, the last column of Table 2 shows that the proposed accelerator architecture over $GF(2^{409})$ is more efficient than $GF(2^{571})$ field. One of the reasons is lower field length (i.e., 409). The lower field length utilizes lower hardware resources and takes lower computational time. Thus, the ratio of throughput over slices results in a higher value for lower field length.

*Comparison of $GF(2^{409})$ & $GF(2^{571})$ on ASIC.* Similar to FPGA results, Table 3 shows that the increase in the field length increases area. For instance, the area for $GF(2^{409})$ field are 0.82 (ratio of 30,561 over 36,857) times lower than $GF(2^{571})$ field. Moreover, on 65 nm process technology, our proposed architecture can operate on a maximum 1800 MHz and 1450 MHz clock frequency over $GF(2^{409})$ and $GF(2^{571})$ fields, respectively. As for as the computational cost for one PM execution is concerned, our proposed architecture over $GF(2^{409})$ and $GF(2^{571})$ require 7.72 µs and 14.17 µs, respectively. This indicates that the 0.71 (ratio of 409 over 571) times increase in the field length (i.e., from $GF(2^{409})$ to

$GF(2^{571})$) results in 0.54 (ratio of 7.72 over 14.17) times increase in computation time. As shown in column six of Table 3, the throughput of our proposed accelerator design over $GF(2^{409})$ and $GF(2^{571})$ is 129.53 kbps and 70.57 kbs, respectively. When we consider the defined FoM for comparison, the last column of Table 3 shows that the proposed accelerator architecture over $GF(2^{409})$ is more efficient than $GF(2^{571})$ field. The reason is lower field length (i.e., 409). The lower field length utilizes lower hardware resources and takes lower computational time. Hence, the ratio of throughput over area results in a higher value for lower field length.

*Comparison to FPGA and ASIC results.* As expected, the performance of our proposed architecture on ASIC is more efficient as compared to the FPGA implementations. The area comparison is not possible to provide. However, the frequency achieved on ASIC over $GF(2^{409})$ and $GF(2^{571})$ is 5.02 (ratio of 1800 with 357) and 4.57 (ratio of 1450 with 317) times higher as compared to FPGA. Similarly, the latency calculated for ASIC over $GF(2^{409})$ and $GF(2^{571})$ is 5.04 (ratio of 38.94 with 7.72) and 4.57 (ratio of 64.82 with 14.17) times lower than compared FPGA. The computational cost of our proposed architecture in latency can be reduced by using pipeline registers after each Karatsuba split, shown in Figure 2. If we consider the throughput and FoM values, Tables 2 and 3 reveal that the ASIC implementations on 65 nm technology are more efficient as compared to the modern 28 nm technology on FPGA.

### 4.2. Comparison to State of the Art

The comparison to state-of-the-art architectures is shown in Table 4. Column one gives the reference design and publication year (in form of Ref/Year). The utilized field length $m$ is illustrated in column two. The implementation device is presented in column three. Columns four and five show the hardware resources in FPGA slices and LUTs. The last three columns provide the timing information in clock cycles, operational frequency and latency.

**Table 4.** Comparison to related PM architectures.

| Ref/Year | $m$ | Device | Slices | LUTs | CCs | Freq (MHz) | Latency (µs) |
|---|---|---|---|---|---|---|---|
| **Implementations of Weierstrass model of ECC** | | | | | | | |
| [15]/2018 | 409 | Virtex 7 | 12,290 | NA | NA | NA | 9.50 |
| [17]/2015 | 409 | Virtex 7 | 6888 | 20,881 | NA | 316 | 32.72 |
| [19]/2021 | 409 | Virtex 6 | NA | 116,241 | 5784 | 135 | 41.36 |
| [15]/2018 | 571 | Virtex 7 | 20,291 | NA | NA | NA | 18.51 |
| [17]/2015 | 571 | Virtex 7 | 12,965 | 38,547 | NA | 250 | 57.61 |
| [19]/2021 | 571 | Virtex 6 | NA | 116,241 | 7628 | 135 | 56.50 |
| [16]/2021 | 571 | Virtex 7 | NA | 80,970 | NA | 274 | 12.55 |
| **Implementations of Edward model of ECC** | | | | | | | |
| [20]/2020 | 256 | Virtex 6 | 6600 | NA | NA | 93 | 2130 |
| [21]/2019 | 256 | Virtex 7 | 8873 | NA | 262,650 | 178 | 1480 |
| [22]/2020 | 233 | Virtex 6 | 1245 | 3878 | NA | 107 | 6720 |
| [23]/2021 | 233 | Virtex 7 | 2662 | 24,727 | 3244 | 179 | 18.10 |
| [30]/2019 | CURVE25519 | ZYNQ 7000 | NA | 12.95k | NA | 137 | 280.64 |
| **Implementations of Huff model of ECC** | | | | | | | |
| [25]/2021 | 233 | Virtex 7 | 7123 | NA | 15,495 | 188 | 82.4 |
| [24]/2020 | 233 | Virtex 7 | 7017 | NA | 13,057 | 434 | 30.08 |
| **This work** | 409 | Virtex 6 | 6439 | 14,578 | 13,903 | 210 | 66.20 |
| **This work** | 571 | Virtex 6 | 7237 | 16,856 | 20,551 | 170 | 120.88 |
| **This work** | 409 | Virtex 7 | 4596 | 12,845 | 13,903 | 349 | 39.83 |
| **This work** | 571 | Virtex 7 | 5797 | 14,687 | 20,551 | 311 | 66.08 |

*Comparison to Weierstrass model of ECC.* As compared to [15], our proposed architecture utilizes 2.67 and 3.50 times lower slices on Virtex 7 FPGA device over $GF(2^{409})$ and $GF(2^{571})$, respectively. The cause to achieve higher resource utilization in [15] is the parallel computation of Frobenius maps and point additions while in our work, we use a non-pipelined Karatsuba multiplier. Note that the clock cycles and frequency comparison are not feasible as the particular information is not given. On the other hand, the operation rescheduling for PA and PD computations result in lower latency values (9.50 over $GF(2^{409})$ and 18.51 over $GF(2^{571})$) as compared to our work (39.83 over $GF(2^{409})$ and 66.08 over $GF(2^{571})$). There is always a tradeoff between hardware resources and computational time (i.e., latency).

Concerning [23,29,31], the bit-serial and digit-serial multipliers are practical to reduce hardware resources while bit-parallel and digit-parallel multipliers are more convenient to reduce computational cost with area and power overheads. Therefore, over $GF(2^{409})$ and $GF(2^{571})$, the use of a segmented digit-serial multiplier in [17] results in 1.49 and 2.23 times higher FPGA slices as compared to our proposed architecture where we employed a hybrid bit-parallel Karatsuba multiplier architecture of Figure 3. The clock cycle comparison is not possible as the relevant information is not described in the work of [17]. If we compare the circuit frequency, our architecture is 1.10 and 1.24 times more efficient. The computational cost of our proposed architecture is comparatively 0.82 and 0.87 times higher. Again, we remind the reader that there is always a tradeoff between the area and processing time.

On Virtex 6 FPGA, we provide the area comparison of our work in terms of LUTs with the unified architecture of [19] as the information of the FPGA slices is not reported. Comparatively, our architecture utilizes 7.97 and 6.89 times lower FPGA LUTs over $GF(2^{409})$ and $GF(2^{571})$. The reason for the use of lower resources in our work is that we proposed dedicated designs for $GF(2^{409})$ and $GF(2^{571})$ fields while in [19] a unified ECC architecture is proposed to implement binary field lengths from $GF(2^{233})$ to $GF(2^{571})$. As shown in Table 4, the clock cycles utilized in [19] are lower as compared to our work but we achieved higher operational frequency (210 for $GF(2^{409})$ and 170 for $GF(2^{571})$). The average sum of frequency achieved for $GF(2^{409})$ and $GF(2^{571})$ is 190 MHz (210 MHz + 170 MHz then divided by 2). Comparatively, our average frequency of two dedicated designs is 1.40 times higher as compared to the frequency (135 MHz) achieved for the unified design of [19]. As we are utilizing the lower hardware resources but our design takes more processing time.

As compared to the 6CC-6CC architecture of [16], our hybrid Karatsuba multiplier-based ECPM design takes 5.51 (ratio of 80,970 over 14,687) times lower FPGA LUTs. Similarly, as shown in Table 4, our architecture can operate on maximum 311 MHz while the design of [16] can operate on 274 MHz which is comparatively 1.13 (ratio of 311 over 274) times lower than our work. But, the latency values calculated for [16] are lower than our work (shown in Table 4). There is always a tradeoff.

*Comparison to Edward model of ECC.* As shown in Table 4, the Edward model of ECC on $GF(p)$ and $GF(2^m)$ field lengths are implemented in [20–23,30]. These Edward curve implementations utilize 256-bit and 233-bit field lengths. But in our work, we utilize only the higher field lengths of 409 and 571. In short, due to different field lengths, a fair comparison is impossible to provide. As seen in Table 4, for higher field lengths of 409-bit and 571-bit, our architectures on Virtex 6 and Virtex 7 achieves higher operational frequency as compared to Edward designs of [20–23,30]. Similarly, as compared to [20–22,30], our architectures take lower computational time (i.e., latency) on Virtex 6 and Virtex 7 FPGA devices. The design of [23] is more efficient as compared to our work in terms of latency as their architecture utilizes 233-bit field length while in our work we use 409 and 571 bits field length. The area comparison is non-trivial as lower field lengths are targeted in designs of [20–23,30] when compared to our work. Consequently, the proposed large field length designs of this work are competent for recent Edward-based FPGA solutions.

*Comparison to Huff model of ECC.* Table 4 provides that the Huff model of ECC on $GF(2^{233})$ field length is implemented in [24,25]. Due to higher field lengths (409-bit and 571-bit) in our work, a reasonable comparison is impossible to provide. On Virtex 7 FPGA,

the 4-stage pipeline designs of [24,25] over $GF(2^{233})$ takes 1.22 (ratio of 7123 over 5797) and 1.21 (ratio of 7017 over 5797) times higher FPGA slices as compared to our non-pipelined architecture over $GF(2^{571})$. On a similar Virtex 7 device, the comparison to latency reveals that the proposed architecture over $GF(2^{571})$ is more efficient than the design of [25] over $GF(2^{233})$. As the field lengths are different, the design of [24] over $GF(2^{233})$ takes 2.19 (ratio of 66.08 over 30.08) times lower computational time as compared to our Virtex 7 implementation of $GF(2^{571})$.

## 5. Conclusions

This article has presented a novel elliptic curve cryptographic processor over $GF(2^{409})$ and $GF(2^{571})$ using Lopez-Dahab projective point arithmetic operations. The critical feature of this work is a hybrid Karatsuba multiplier of 4-split polynomials to multiply two polynomial multiplications. A hybrid multiplier is implemented using a general Karatsuba and traditional schoolbook multiplication approaches. For low hardware resource utilizations, the proposed multiplier design is employed to implement the modular squares and addition chains of the Itoh-Tsujii algorithm for inverse computations. On Xilinx Virtex 7 FPGA device, the number of the required slices is 4439 and 5683 over $(2^{409})$ and $GF(2^{571})$, respectively. Similarly, on ASIC 65nm process technology, the core size is 30,561 $\mu m^2$ and 36,857 $\mu m^2$ for $GF(2^{409})$ and $GF(2^{571})$, respectively. The comparison to existing designs in the literature shows that the proposed architecture utilizes less area and provides competent performance for PM computation. Therefore, the proposed design is an alternate for area-constrained cryptosystems.

**Author Contributions:** Conceptualization, M.R. and M.A.; methodology, M.R. and M.Y.I.Z.; software, N.K. and M.A.; validation, M.R. and M.A.; formal analysis, M.R. and N.K.; investigation, M.A. and M.R.; resources, M.R. and N.K.; data curation, O.S.S.; writing—original draft preparation, M.R. and M.H.S.; writing—review and editing, M.R. and M.Y.I.Z.; visualization, O.S.S.; supervision, M.R. and M.Y.I.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors would like to thank the Deanship of Scientific Research at Umm Al-Qura University for supporting this work by Grant Code: (22UQU4320020DSR01).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare that they have no conflict of interest.

## References

1. Miller, V.S. Use of Elliptic Curves in Cryptography. In *Proceedings of the Advances in Cryptology—CRYPTO'85 Proceedings*; Williams, H.C., Ed.; Springer: Berlin/Heidelberg, Germany, 1986; pp. 417–426.
2. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
3. Sinha Roy, S.; Basso, A. High-speed Instruction-set Coprocessor for Lattice-based Key Encapsulation Mechanism: Saber in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 443–466. [CrossRef]
4. Imran, M.; Almeida, F.; Raik, J.; Basso, A.; Roy, S.S.; Pagliarini, S. Design Space Exploration of SABER in 65 nm ASIC. In Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security, Virtual Event, 19 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 85–90. [CrossRef]
5. NIST. PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates. Available online: https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4 (accessed on 11 August 2022).
6. Kumar, K.A.; Krishna, A.V.N.; Chatrapati, K.S. New secure routing protocol with elliptic curve cryptography for military heterogeneous wireless sensor networks. *J. Inf. Optim. Sci.* **2017**, *38*, 341–365. [CrossRef]
7. Gulen, U.; Baktir, S. Elliptic Curve Cryptography for Wireless Sensor Networks Using the Number Theoretic Transform. *Sensors* **2020**, *20*, 1507. [CrossRef] [PubMed]
8. Noori, D.; Shakeri, H.; Niazi, T, M. Scalable, efficient, and secure RFID with elliptic curve cryptosystem for Internet of Things in healthcare environment. *EURASIP J. Inf. Secur.* **2020**, *2020*, 13. [CrossRef]

9. Calderoni, L.; Maio, D. Lightweight Security Settings in RFID Technology for Smart Agri-Food Certification. In Proceedings of the 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, 22–25 June 2020; pp. 226–231. [CrossRef]
10. Singh, R.; Miglani, S. Efficient and secure message transfer in VANET. In Proceedings of the 2016 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 26–27 August 2016; Volume 2, pp. 1–5. [CrossRef]
11. Chavhan, S.; Doriya, R. Secured Map Building using Elliptic Curve Integrated Encryption Scheme and Kerberos for Cloud-based Robots. In Proceedings of the 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 11–13 March 2020; pp. 157–164. [CrossRef]
12. Malina, L.; Dzurenda, P.; Ricci, S.; Hajny, J.; Srivastava, G.; Matulevičius, R.; Affia, A.A.O.; Laurent, M.; Sultan, N.H.; Tang, Q. Post-Quantum Era Privacy Protection for Intelligent Infrastructures. *IEEE Access* **2021**, *9*, 36038–36077. [CrossRef]
13. NIST. Recommended Elliptic Curves for Federal Government Use (1999). Available online: https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf (accessed on 5 January 2023 ).
14. Hankerson, D.; Menezes, A.J.; Vanstone, S. Guide to Elliptic Curve Cryptography; Springer: New York, NY, USA, 2004; pp. 1–311. Available online: https://link.springer.com/book/10.1007/b97644 (accessed on 5 January 2023).
15. Li, L.; Li, S. High-Performance Pipelined Architecture of Point Multiplication on Koblitz Curves. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 1723–1727. [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. Khan, Z.U.A.; Benaissa, M. Throughput/Area-efficient ECC Processor Using Montgomery Point Multiplication on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2015**, *62*, 1078–1082. [CrossRef]
18. Imran, M.; Pagliarini, S.; Rashid, M. An Area Aware Accelerator for Elliptic Curve Point Multiplication. In Proceedings of the 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 23–25 November 2020; pp. 1–4. [CrossRef]
19. Zhao, X.; Li, B.; Zhang, L.; Wang, Y.; Zhang, Y.; Chen, R. FPGA Implementation of High-Efficiency ECC Point Multiplication Circuit. *Electronics* **2021**, *10*, 1252. [CrossRef]
20. Islam, M.M.; Hossain, M.S.; Hasan, M.K.; Shahjalal, M.; Jang, Y.M. Design and Implementation of High-Performance ECC Processor with Unified Point Addition on Twisted Edwards Curve. *Sensors* **2020**, *20*, 5148. [CrossRef]
21. Islam, M.M.; Hossain, M.S.; Hasan, M.K.; Shahjalal, M.; Jang, Y. FPGA Implementation of High-Speed Area-Efficient Processor for Elliptic Curve Point Multiplication Over Prime Field. *IEEE Access* **2019**, *7*, 178811–178826. [CrossRef]
22. Lara-Nino, C.A.; Diaz-Perez, A.; Morales-Sandoval, M. Lightweight elliptic curve cryptography accelerator for internet of things applications. *Ad Hoc Netw.* **2020**, *103*, 102159. [CrossRef]
23. Sajid, A.; Rashid, M.; Imran, M.; Jafri, A.R. A Low-Complexity Edward-Curve Point Multiplication Architecture. *Electronics* **2021**, *10*, 1080. [CrossRef]
24. Rashid, M.; Imran, M.; Jafri, A.R.; Mehmood, Z. A 4-Stage Pipelined Architecture for Point Multiplication of Binary Huff Curves. *J. Circuits Syst. Comput.* **2020**, *29*, 2050179. [CrossRef]
25. Rashid, M.; Imran, M.; Kashif, M.; Sajid, A. An Optimized Architecture for Binary Huff Curves With Improved Security. *IEEE Access* **2021**, *9*, 88498–88511. [CrossRef]
26. Imran, M.; Rashid, M.; Jafri, A.R.; Kashif, M. Throughput/area optimised pipelined architecture for elliptic curve crypto processor. *IET Comput. Digit. Tech.* **2019**, *13*, 361–368. [CrossRef]
27. Itoh, T.; Tsujii, S. A fast algorithm for computing multiplicative inverses in GF (2m) using normal bases. *Inf. Comput.* **1988**, *78*, 171–177. [CrossRef]
28. Rashid, M.; Imran, M.; Jafri, A.R.; Al-Somani, T.F. Flexible Architectures for Cryptographic Algorithms — A Systematic Literature Review. *J. Circuits, Syst. Comput.* **2019**, *28*, 1930003. [CrossRef]
29. Imran, M.; Abideen, Z.U.; Pagliarini, S. An Open-source Library of Large Integer Polynomial Multipliers. In Proceedings of the 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Vienna, Austria, 7–9 April 2021; pp. 145–150. [CrossRef]
30. Mehrabi, M.A.; Doche, C. Low-Cost, Low-Power FPGA Implementation of ED25519 and CURVE25519 Point Multiplication. *Information* **2019**, *10*, 285. [CrossRef]
31. Imran, M.; Rashid, M. Architectural review of polynomial bases finite field multipliers over GF(2m). In Proceedings of the 2017 International Conference on Communication, Computing and Digital Systems (C-CODE), Islamabad, Pakistan, 8–9 March 2017; pp. 331–336. [CrossRef]