Journal of Computing and Security

Research Article, 2021, Volume 8, Number 1 (pp. 1–18)

http://www.jcomsec.org

Bit-Parallel ECC Coprocessor resistant to Differential Power Analysis Attacks in GF(2^m)

Hashem Rezaei [a], Alireza Shafieinejad [a,*]

[a] *Departement of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran.*

ARTICLE INFO.

Article history:
Received: 26 June 2020
Revised: 22 March 2021
Accepted: 18 April 2021
Published Online: 23 May 2021

Keywords:
Differential PowerAttacks, Bit-Parallel Architecture, ECC Processor, Randomized Montgomery Algorithm.

ABSTRACT

Elliptic curve cryptography (ECC) is one of the most popular public key systems in recent years due to its both high security and low resource consumption. Thus, ECC is more appropriate for Internet applications of Things, which are mainly involved with limited resources. However, non-invasive side channel attacks (SCAs) are considered a major threat to ECC systems. In this paper, we design a processor for the ECC in the binary field, resistant to Differential Power Attacks (DPA). The main operations in this architecture are randomized Montgomery multiplication and division units, which make it impossible to create differential power attacks by involving a random number in the calculation process. The goal is to accelerate the operation by opening the loops in the Montgomery randomized multiplication/division units, and accordingly, a bit-parallel design instead of bit-serial design. The results show that, despite the complexity of the logic in the two/three-bit processing versions, the speed is significantly improved by accepting a slight increase in the area resource. Further, our design is flexible wherein the top-level module, depending on the area-speed trade-off, a variety of multiplier and divisor units can be selected. The FPGA evaluations show that in terms of Time×Slice metric, the 2-bit divider/3-bit multiplier version of our architecture leads to a 40% improvement over the best previous work. Further, by duplicating the divider and multiplier modules along with the bit-parallel architecture this gain can reach to 50%. In terms of operation speed, our design versions are faster than previous work by a factor of 1.87 and 3.29. Furthermore, ASIC evaluations in terms of the Time×Area metric, indicate that deploying a 2-bit multiplier leads to a 19% gain relative to previous well-known work. Moreover, duplication of modules along with bit-paralleling amplifies the overall gain up to 36%.

1 Introduction

Elliptic curve cryptography (ECC) is a widely used public key cryptosystem [1] since it offers an equivalent security margin along with shorter key length relative to other public key cryptosystems such as RSA and discrete logarithm [2].

* Corresponding author.

Email addresses: `h.rezaei@modares.ac.ir` (H. Rezaei), `shafieinejad@modares.ac.ir` (A. Shafieinejad)

https://dx.doi.org/10.22108/JCS.2021.123698.1051
ISSN: 2322-4460 © Research Article, 2021 JComSec. All rights reserved.

As security is implemented in various network levels, such as IP security (IPsec), secure socket layer (SSL), and Application, a low-latency implementation of an asymmetric cryptosystem is crucial and accordingly is a main subject of research. Further, for the resource-limited device that is used in embedded systems low-complexity implementation is important. Scalar point multiplication is the basic operation in ECC which most of the upper layer protocols use as the basic primitive. Thus, the entire performance of most ECC protocol is dependent on the performance of scalar point multiplication. Some of the works such as [3] and [4] tried to adjust the ECC processor according to the requirements of various Internet of Things environments such as low hardware complexity and processing time.

Beyond the performance from the security point of view, the private data of an unprotected hardware device can be extracted by physical attacks due to side-channel leakage. Based on the methods selected by an adversary, the attacks can be categorized as simple/ differential power-analysis (SPA/DPA) proposed by Kocher [5], timing analysis (TA), and correlation power analysis(CPA) [6].

The basic countermeasure method for avoiding simple power/time analysis is a *double-and-add-always* method which is primarily deployed by the Montgomery Ladder algorithm.

The authors in [4] defined an emerging family of lightweight ECC in the prime field such as MoTE curves to meet the development requirement on resource-constrained devices. The parameterized implementation has two optimized-specific designs: the high-speed version (HS) and the memory-efficient (ME) version. Some efforts are taken to harden the deployed library against some basic side-channel attacks, *e.g.*, timing attacks and simple power analysis attacks.

The authors, in [7], proposed an ECC processor based on the Globally Asynchronous Locally Synchronous (GALS) with focusing on the resistances of design against side-channel attacks (SCAs). The pausible clocking scheme, with random hopping of clock frequencies, is applied as a countermeasure of SCAs with low overhead to inject timing uncertainty on the cryptographic operations.

On the other side, the DPA and CPA attacks which investigate the correlation between target power traces and power model can reveal the key-value due to the existence of key-dependent operations in every round of calculation. The countermeasure methods for avoiding such attacks involve *hiding techniques* with algorithm-independent dedicated circuit [8] and *masking* the processed data at the algorithm level [9]. However, most of these methods practically fail due to the additional cost of area and time.

In a prominent work, Lee et al. [10] have proposed a novel DPA resistant algorithm in GF(p) performing all field operations in a *randomized Montgomery domain* to eliminate the correlation between the measured target power traces and the power reference model. The basic idea of the algorithm is a transformation of the operands into a randomized Montgomery domain $A \equiv a.2^{\lambda}$ (mod p), where the domain value λ equals the Hamming weight (HW) of an n-bit random value. The random value is selected at the start of scalar point multiplication, and accordingly transforming the x/y coordinates of base-point into a random domain. All of the subsequent point addition and doubling are done in the randomized domain by the means of RMM, Randomized Montgomery Multiplier, and RMD, Randomized Montgomery Divider algorithm. Finally, the result is transformed back into the original domain by two RMM operations with identity operand 1.

Liao et al. [11] extended the work of Lee from two aspects. First, it represents a randomized Montgomery Ladder Algorithm in GF(2m) with the aims of Randomized Montgomery Addition, Multiplication, and Division modules. Second, it comes up with a division algorithm whose iterations are constant and are independent of the input value. This causes the system time constant combined with the Montgomery Ladder algorithm for scalar multiplication. It is worth noting that the work of Liao is done in affine coordinates. The main shortcoming of Liao architecture is the bit-serial implementation which shows a large gap between the speed of the proposed coprocessor relative to other designs in projective coordinates such as [12].

In this paper, we propose an efficient ECC coprocessor in $GF(2^m)$ in affine coordinates resistant to passive SCA attacks. Similar to [10] and [11], the main resistant technique is the operation in the randomized Montgomery domain. In comparison to [10], our ECC coprocessor operates in the binary field rather than a prime field. Further, compared to Liao design [11], we present a bit-parallel architecture for both multiplication and division modules. The goal is to speed up the scalar point multiplication and accordingly entire ECC operation relative to bit-serial architecture. Further, this is a long step to reduce the large gap between the performance of the ECC coprocessor in affine coordinates relative to ECC in projective coordinates. The evaluation results show that our ECC coprocessor, by reducing the iteration time of the multiplication and division, outperforms in both time and time-area objectives, the previous related works.

1.1 Contributions of the Work

The main contributions of this paper can be summarized as follows:

- We propose a bit-parallel algorithm for $\acute{k}$-bit randomized modular multiplier ($\acute{k}\leqslant 4$), as well as a bit-parallel algorithm for k-bit randomized modular divider (k⩽3), that reduces the required cycles for multiplication/division by a factor of $\acute{k}$/k.
- Based on these modules, we present a noninvasive SCAs resistant coprocessor, namely RMD-k/RMM-$\acute{k}$, that performs scalar point multiplication in GF(2m) based on randomized Montgomery Ladder.
- Further, by accepting redundant modules, we present a flexible coprocessor in GF(2m), namely RMD2-k/RMM2-$\acute{k}$, that consists of two multipliers and two divider modules.
- We discuss the trade-off between time and area objectives to find the optimized value for both k and $\acute{k}$ in RMD-k/RMM-$\acute{k}$ and RMD2-k/RMM2-$\acute{k}$ regarding time, area, and time-area optimized design.

The rest of this paper is outlined as follows. Section 2 gives a brief background of ECC for GF(2m) along with a brief introduction of the Liao work [5] for the Randomized Montgomery Ladder algorithm for scalar multiplication. The bit-parallel randomized Montgomery multiplier and divider is introduced and explained in Section 3. The overall architecture of the ECC coprocessor is discussed in Section 4, including two designs; resource-optimized and time-optimized based on allowing/disallowing redundant RMM and RMD modules. In Section 5, the evaluation results are presented which gives the performance analysis of the proposed coprocessor and the comparison with other related works based on the synthesis results. Finally, Section 6 concludes the paper.

2 Preliminarily

2.1 ECC Over GF(2^m)

A non-supersingular elliptic curve E is defined over either a binary finite field GF(2^m) or GF(p). For binary field, the set of points along with a point at infinity satisfies the reduced Weierstrass equation:

$$E : y^2 + xy = x^3 + ax^2 + b \qquad (1)$$

where a,b∈ GF(2^m) and b≠0. This curve over binary field is suitable for hardware implementation. While the curve $E : y^2 + xy = x^3 + ax^2 + b$ over GF(p), prime field, is proper for software implementation. The addition of two points is a primary operation on the group constructed by an elliptic curve. The other primary operation includes the addition of two similar points which refers to point doubling. Assume $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two points on the curve defined by Equation (1). We can show that the addition of these points, $P_3 = (x_3, y_3) = P_1 + P_2$, is obtained by Equation (2). Further, the doubling of P_1, denoted by $P_4 = (x_4, y_4) = 2P_1$, is given by Equation (3).

$$\begin{cases} x_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + a \\ y_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1 + x_3) + x_3 + y_1 \end{cases} \qquad (2)$$

$$\begin{cases} x_4 = \left(x_1 + \frac{y_1}{x_1}\right)^2 + x_1 + \frac{y_1}{x_1} + a \\ y_4 = x_1^2 + x_4\left(x_1 + \frac{y_1}{x_1}\right) + x_4 \end{cases} \qquad (3)$$

It is worth noting that both the equations are defined in affine coordinates which consists of computationally intensive modular divisions. In many prior works such as [12], researchers used projective coordinates to avoid divisions by accepting a large number of extra multiplications and squarings. However, in projective coordinates a single inversion operation for the projective to affine coordinates conversion is unavoidable which can be achieved with multiplicative inversion based on Fermat's little theorem. The Itoh-Tsujii [13] algorithm requires only $\log_2 m$ multiplications and m-1 repeated squaring operations. In summary, overall performance of projective coordinate highly depends on the efficiency of the field multipliers.

However, if the division is well designed, the computation in affine coordinate also becomes a suitable option for high-performance ECC design. This work investigates the bit-parallel architecture for the divider module in addition to the multiplier module. Bit-parallel processing is a crucial technique for achieving better performance if the time-area tradeoff is well discussed and resolved.

2.2 Liao Randomized Montgomery Ladder

In this section, we describe the Randomized Montgomery Ladder Algorithm, namely RMLA, proposed by Liao [11]. This is a typical extension of the Montgomery Ladder algorithm (MLA) which involves a random parameter to make the algorithm resistant against differential power attack [10]. MLA is one of the most widely used algorithms for fast scalar multiplication, which efficiently completes the operation with m iterations for m-bit ECC. The MLA constant runtime provides an inherent way to resist simple SCAs (such as SPA) since the type of operation is independent of the current bit value of scalar k.

Algorithm 1 Randomized Montgomery Ladder (RMLA).

```
INPUT: scalar $k = k_{m-1 1}k_0$, where $k_{m-1} = 1$, base point $P$, $f(x)$,
OUTPUT: $k.P$
 1: set $\alpha \leftarrow$ a $m$-bit random number;
 2: set $X \leftarrow RMD(P.x, 1, f, \alpha)$
 3: set $Y \leftarrow RMD(P.y, 1, f, \alpha)$
 4: set $P_T \leftarrow (X, Y)$
 5: set $P_1 \leftarrow P_T, P_2 \leftarrow 2P_T$
 6: for $i \leftarrow m-2$ downto 0 do
 7:     if ($k_i$ =10) then
 8:         $P_1 \leftarrow P_1 + P_2, P_2 \leftarrow 2P_2$
 9:     else
10:         $P_1 \leftarrow 2P_1, P_2 \leftarrow P_1 + P_2$
11:     end if
12: end for
13: set $x \leftarrow RMM(P1.x, 1, f, \alpha)$
14: set $y \leftarrow RMM(P1.y, 1, f, \alpha)$
15: return $(x, y)$;
```

Algorithm 2 Bit-Serial Randomized Montgomery Multiplier (RMM) [11].

```
INPUT: $A(x), B(x), f(x), \alpha$
OUTPUT: $C(x) = A(x).B(x).x^{-\beta}$ mod $f(x)$
 1: set $R \leftarrow A, S \leftarrow B, Q \leftarrow 0, T \leftarrow \alpha$,
 2: for $i \leftarrow 0$ downto $m-1$ do
 3:     if ($R[0] == 1$ && $T[0] == 0$) then
 4:         $Q \leftarrow Q$
 5:     else
 6:         if ($R[0] == 1$ && $T[0] == 0$) then
 7:             $Q \leftarrow Q + S$;
 8:         else
 9:             if ($R[0] == 0$ && $T[0] == 1$) then
10:                 $Q \leftarrow Q.x^{-1}$ mod $f$;
11:             else
12:                 $Q \leftarrow (Q + S).x^{-1}$ mod $f$;
13:             end if
14:         end if
15:     end if
16:     if ($T[0] == 0$) then
17:         $S \leftarrow S.x$ mod $f$;
18:     else$S \leftarrow S$
19:     end if
20: end for
21: return $Q$
```

The defined randomized Montgomery arithmetic are utilized to prevent the first-order noninvasive SCAs. It realizes the randomness based on a randomly generated number α for every scalar multiplication, which breaks the statistical dependence of the side-channel leakages on the real-time operating data. Thus, the adversaries are incapable of utilizing the leakages to conduct all kinds of first-order noninvasive SCAs, including differential PA attacks, correlation PA, and template attacks.

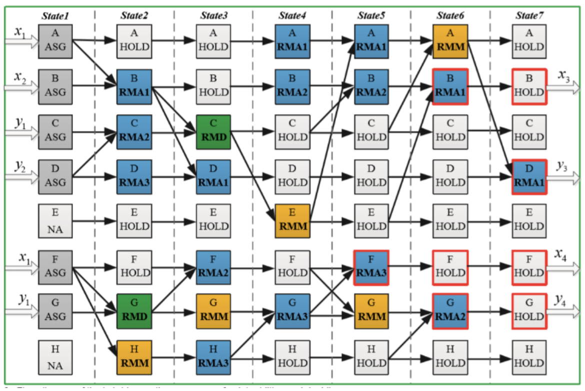


Figure 1. Operation Sequences for RMLA (Liao ECC Coprocessor [11]).

Let $f(x)$ denotes an irreducible polynomial of degree m in GF(2^m. For the desired polynomial, $a(x)$ of degree m-1, the corresponding representation in randomized Montgomery domain will be $A(x) = a(x).x^{\beta}$, where β is the Hamming Weight of a random m-bit binary number α. The binary finite field arithmetic in the randomized Montgomery domain is defined as follows:

Randomized Montgomery Addition (**RMA**): $C(x) = RMA(A(x), B(x)) = (a(x) + b(x)).x^{\beta} modf(x) = c(x).x^{\beta} modf(x)$

Randomized Montgomery Multiplication (**RMM**): $C(x) = RMM(A(x), B(x)) = A(x).B(x).x^{-\beta} = a(x).b(x).x^{\beta} modf(x) = c(x).x^{\beta} modf(x)$

Randomized Montgomery Division (**RMD**): $C(x) = RMD(A(x), B(x)) = A(x)/B(x).x^{\beta} = a(x)/b(x).x^{\beta} modf(x) = c(x).x^{\beta} modf(x)$

The details of RMLA are described in Algorithm 1. In addition to the base point P and the integer k, a random number α and $f(x)$ are given as inputs. For conversion to the randomized domain, a single RMD operation with identity polynomial 1 is sufficient. As we see in Algorithm 1, the base point is transformed to the randomized domain by two RMD operations at the beginning of the algorithm. The main loop of the algorithm repeats the point addition and doubling according to each bit of scalar k. The operation of point addition and doubling, depicted in Figure 1, is accomplished by a sequence of RMA, RMM, and RMD modules. Ultimately, the final result is converted back to the original domain by two RMMs with identity polynomial 1.

Randomized Montgomery multiplication and division, namely RMM and RMD, are respectively shown in Algorithms 2 and 3 [11]. Both the algorithms have

Algorithm 3 Liao Bit-Serial Randomized Montgomery Divider (RMD) [11].

INPUT: $A(x), B(x), f(x), \alpha$
OUTPUT: $C(x) = A(x)/B(x).x^{\beta}$ mod $f(x)$

```
1: set $R \leftarrow B, S \leftarrow f, U \leftarrow A, V \leftarrow 0, T \leftarrow, count \leftarrow 0$
2: for $i \leftarrow 0$ to $2m - 1$ do
3:     if ($R[0] = 0$) then
4:         $R \leftarrow R/x, S \leftarrow S, count \leftarrow count + 1$
5:         if ($T[0] = 0$) then
6:             $U \leftarrow U, V \leftarrow V.x \bmod f;$
7:         else
8:             $U \leftarrow U/x \bmod f, V \leftarrow V;$
9:         end if
10:    else
11:        if ($R[0] = 1$ && count $\geqslant 0$) then
12:            $R \leftarrow (R + S)/x;\ S = R,\ count \leftarrow -(count + 1);$
13:            if ($T[0] = 1$) then
14:                $U \leftarrow U + V; V \leftarrow V.x \bmod f;$
15:            else
16:                $U \leftarrow (U + V)/x \bmod f; V \leftarrow U;$
17:            end if
18:        else
19:            $R \leftarrow (R + S)/x; S \leftarrow S, count \leftarrow count + 1;$
20:            if ($T[0] = 1$) then
21:                $U \leftarrow U + V; V \leftarrow V.x \bmod f;$
22:            else
23:                $U \leftarrow (U + V)/x \bmod f; V \leftarrow V;$
24:            end if
25:        end if
26:    end if
27:    if ($i \leqslant m - 2$) then
28:        $T \leftarrow T >> 1;$
29:    else$T \leftarrow 0$
30:    end if
31: end for
32: return $C = V$
```

bit-serial implementation. The former is fulfilled by m iterations while the latter is accomplished across $2m$ iterations. This algorithm at each clock process a single bit of both registers R and T and correspondingly repeats for 2m times.

The formula which is used for point addition and point doubling is respectively denoted by the Equations (4) and 5. These are an extended version of the basic formula, (2) and (3), wherein the x-coordinate of the result, x_3 in point addition and x_4 point doubling, is eliminated in the evaluation of the y-coordinate [11].

$$\begin{cases} x_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + a \\ y_3 = \left[x_1 + a + 1 + \left(\frac{y_1+y_2}{x_1+x_2}\right)^2\right]\left(\frac{y_1+y_2}{x_1+x_2}\right) \\ +x_1 + x_2 + y_2 + a \end{cases} \quad (4)$$

$$\begin{cases} x_4 = x_1^2\left(\frac{y_1}{x_1}\right)^2 + x_1 + \frac{y_1}{x_1} + a \\ y_4 = x_1^2 + a + \left[x_1^2 + \left(\frac{y_1}{x_1}\right)^2 + a + 1\right]\left(x_1 + \frac{y_1}{x_1}\right) \end{cases} \quad (5)$$

The flow diagram of the operation sequence for both point addition and doubling is shown in Figure 2. The inputs of point addition are denoted by (x_1, y_1) and (x_2, y_2). The point doubling has a single point which is assumed to be the first point, *i.e.*, (x_1, y_1). The goal is the computation of point-addition and doubling according to Equations (4) and 5. The outputs of the unit are respectively shown by (x_3, y_3) for addition and (x_4, y_4) for doubling.

The unit has eight m-bit register chains (denoted by A to H) and seven states (denoted by state1 to state7). The four input coordinates are latched to the corresponding register chains in State1, and the results of point addition and doubling are finally ob-

Table 1. The Different States of 2-Bits Randomized Divider (“X” Denotes Don’T Care a Bit).

$R_1R_0T_1T_0$	After processing R_0T_0		After processing $R_0T_0R_1T_1$	
	Q	S	Q	S
0000	Q	$S.x$	Q	$S.x^2$
0001	$Q.x^{-1}$	S	$Q.x^{-1}$	$S.x$
0010	Q	$S.x$	$Q.x^{-1}$	$S.x$
0011	$Q.x^{-1}$	S	$Q.x^{-2}$	S
0100	$Q+S$	$S.x$	$Q+S$	$S.x^2$
0101	$(Q{+}S).x^{-1}$	S	$Q.x^{-1}+S.x^{-1}$	S.x
0110	$Q+S$	$S.x$	$Q.x^{-1}+S.x^{-1}$	S.x
0111	$(Q{+}S).x^{-1}$	S	$Q.x^{-2}+S.x^{-2}$	S
1000	Q	$S.x$	$Q+S.x$	$S.x^2$
1001	$Q.x^{-1}$	S	$Q.x^{-1}+S$	$S.x$
1010	Q	$S.x$	$Q.x^{-1}+S$	S.x
1011	$Q.x^{-1}$	S	$Q.x^{-2}+S.x^{-1}$	S
1100	$Q+S$	$S.x$	$Q+S+S.x$	$S.x^2$
1101	$(Q{+}S).x^{-1}$	S	$Q.x^{-1}+S.x^{-1}+S$	$S.x$
1110	$Q+S$	$S.x$	$Q.x^{-1}+S.x^{-1}+S$	$S.x$
1111	$(Q{+}S).x^{-1}$	S	$Q.x^{-2}{+}S.x^{-2}{+}S.x^{-1}$	S

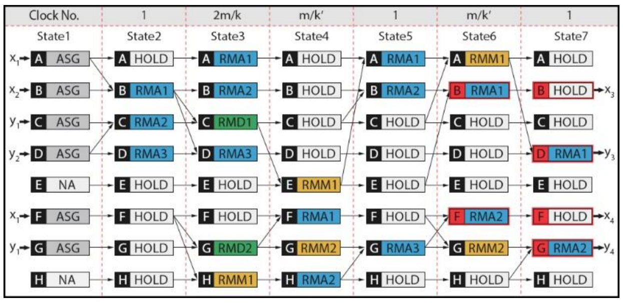


Figure 2. Operation Sequences For RMD^2-k/RMM^2-$\acute{k}$.

tained in State7. In each state, different operations are executed to generate fresh data and update the relevant register chains. The operations in each state are done by a subset of one RMD, one RMM, and three RMA modules (RMA1, RMA2, and RMA3).

The delay of both state2 and state3 is identified by the clocks required to complete the RMD computations, *i.e.* equals $2m$, while the delay of three subsequent states (4, 5, and 6) are identified by the clock numbers consumed by the RMM module, *i.e.*, m. Thus, the total clock number required for one round of point-addition/doubling equals $7m$+1.

3 Parallel-Bit Processing Multiplier and Divider

The basic idea is to replace of bit-serial multiplier and divider with multi-bit versions. For multiplier, we propose 2-bit, 3-bit, and 4-bit processing versions. Due to the complexity of the divider algorithm, only 2-bit and 3-bit versions are discussed in this paper.

3.1 2-Bit Randomized Multiplier

The Liao bit-serial randomized multiplier, namely RMM, is shown in Algorithm 2. It uses four registers R, S, Q, and T which are initially assigned by A (first operand), B(second operand), zero, and α (random number). At each clock, the value of Q and S is updated corresponding to the least significant bit of registers R and T. The value of Q is dependent on both $R(0)$ and $T(0)$ while S depends only on $T(0)$. Thus, Q gets one of the expressions Q, $Q+S$, $Q.x^{-1}$, or $(Q+S)x^{-1}$ while S is assigned either by S or $S.x$. The loop repeats for m times, the size of the field, and finally Q is returned as the result of the product.

The formulation can be simplified respectively to only 12 distinct states for evaluation of Q and 3 states for evaluation of S. According to Table 1, we see that Q is equal to the sum of a subset of $\{Q$, S, $Q.x^{-1}$, $S.x^{-1}$, $Q.x^{-2}$, S.$x^{-2}\}$ while S has simpler form and

Algorithm 4 Randomized 2-Bit Multiplier (RMM-2).

```
INPUT: $A(x), B(x), f(x), \alpha$
OUTPUT: $C(x) = A(x).B(x).x^{-\beta}\ mod\ f(x)$
1: $setR \leftarrow A, S \leftarrow B, Q \leftarrow 0, T \leftarrow \alpha$;
2: $n \leftarrow (m+1)/2$;
3: for $i \leftarrow 1\ to\ n$ do
4:     switch $R(1)R(0)T(1)T(0)$
5:     case "0000" : $Q \leftarrow Q$;
6:     case "0001","0010": $Q \leftarrow Q.x^{-1}\ mod\ f$;
7:     case "0011" : $Q \leftarrow Q.x^{-2}\ mod\ f$;
8:     case "0100" : $Q \leftarrow Q + S$;
9:     case "0101","0110": $Q \leftarrow (Q+S).x^{-1}\ mod\ f$;
10:    case "0111" : $Q \leftarrow (Q+S).x^{-2}\ mod\ f$;
11:    case "1000" : $Q \leftarrow Q + (S.x\ mod\ f)$;
12:    case "1001","1010": $Q \leftarrow (Q.x^{-1}\ mod\ f)+S$;
13:    case "1011" : $Q \leftarrow Q.x^{-2} + S.x^{-1}\ mod\ f$;
14:    case "1100" : $Q \leftarrow (Q+S) + S.x\ mod\ f$;
15:    case "1101","1110": $Q \leftarrow (Q+S).x^{-1}\ mod\ f + S$;
16:    case "1111": $Q \leftarrow (Q+S).x^{-2} + S.x^{-1}\ mod\ f$;
17:    end switch;
18:    switch $T(1)T(0)$
19:    case "00" :$S \leftarrow S.x^2\ mod\ f$;
20:    case "11" : $S \leftarrow S$;
21:    case "10", "01" : $S \leftarrow S.x\ mod\ f$;
22:    end switch;
23:    $R \leftarrow R >> 2$;
24:    $T \leftarrow T >> 2$;
25: end for
26: return $Q$
```

is equal to one of the S, $S.x$, and $S.x^2$. The above-mentioned formulation can be used to deploy a 2-bit randomized modular multiplier, 2-bit RMM. This algorithm at each clock process 2-bits of registers R and T and correspondingly repeats only for $M/2$ times. The detail of 2-bit RMM (RMM-2) is shown in Algorithm 4.

3.2 3/4-Bit Randomized Multiplier (RMM-3 and RMM-4)

We can extend multi-bit processing to a higher bit similar to a 2-bit multiplier. For a 3-bit multiplier (RMM-3), the three least significant bits of both R and T registers are used for computation of Q and S. Thus, we will have at most 64 states. By simplification, 33 distinct states have remained for computation of Q. The value of S is equal to one of the S, $S.x$, $S.x^2$ and $S.x^3$. For the lack of space, we ignore bringing the details for computation of Q.

Further, for the 4-bit multiplier (RMM-4), the 4 least significant of R and T are included in computation which leads to 256 states. After simplification, there are only 111 distinct states for computation of Q. The value of S is equal to one of the S, $S.x$, $S.x^2$ and $S.x^3$, and $S.x^4$.

3.3 2-Bit Randomized Divider (RMD-2)

The Liao bit-serial randomized modular divider, namely RMD, is shown in Algorithm 3. It uses five registers R, S, U, V, and T which are initially assigned by A (second operand), irreducible polynomial (f), B(first operand), zero, and α (random number), respectively. At each clock, the value of U, V, R, and S is updated corresponding to the least significant bit of registers R, T, and an internal variable Count.

The value of R, S, and $Count$ depends on $R(0)$ and the sign of $Count$. While U, V is dependent on $T(0)$ as well as $R(0)$ and the sign of Count. Register U is assigned by one of the expressions U, $U.x^{-1}$, U +V, and $(U{+}V).x^{-1}$ while V is assigned by V, U, $V.x$, or $U.x$. Further, R is changed by a type of modular shift, *i.e.*, based on the value of $R(0)$ is assigned either by R or $R+S$ shifted 1-bit to right.

The loop repeats 2m times, and finally, V is returned as the result of division. Our idea is the opening of the loop and processing $R(1)T(1)$ in addition to $R(0)T(0)$. For similar but simple work, we can point to [14]. We will have 32 states at all which are shown in Table 2. Further, we need a new variable $Count2$ in addition to $Count$ to complete the division state. We can see that U is not dependent on the sign of $Count2$. $Count2$ is only used in few cases for computation of V. Further, Instead of using $R(1)$ directly, we use a modular shift of R, shifting 1-bit to right either R or $R+S$ based on the value of $R(0)$, and correspondingly using the least significant bit of the result, namely $R_1(0)$. For simplicity, we omit the mod f from the expressions. After simplification, there are 15 distinct states left for evaluating U and 10 states for V. We see that both U and V are equal to the sum of a subset of V, $V.x^{-1}$, V.x^{-2}, U, $U.x^{\pm 1}$, $U.x^{\pm 2}$. the value of R, S, and Count are respectively updated based on the value of R_1, S_1, and $Count2$.

We can use this formulation to deploy a 2-bit randomized modular divider, namely RMD-2. This algorithm at each clock process 2-bits of registers R and T and correspondingly repeats only for m times. The detail of RMD-2 is shown in Algorithm 5.

3.4 3-Bit Randomized Modular Divider (RMD-3)

Similar to a randomized multiplier, we can extend multi-bit processing for a divider. For the complexity

Table 2. The Different States of 2-Bits Randomized Divider (“X” Denotes Don’T Care a Bit).

$R_1(0)R(0)T(1)T(0)$	Sign (Count)	$Sign(Count_2)$	After Processing R(0)T(0)		After Processing R(0)T(0)R1(0)T(1)	
			V	U	V	U
0000	X	X	V	$U.x^-1$	V	$U.x^{-2}$
0001	X	X	V.x	U	$V.x^2$	U
0010	X	X	V	$U.x^-1$	$V.x$	$U.x^{-1}$
0011	X	X	$V.x$	U	$V.x$	$U.x^{-1}$
0100	+	X	U	$(U+V).x^{-1}$	U	$(U+V).x^{-2}$
0101	-	X	V	$(U+V).x^{-1}$	V	$(U+V).x^{-2}$
0110	+	X	U.x	$U+V$	$U.x^2$	$U+V$
0111	-	X	$V.x$	$U+V$	$V.x^2$	$U+V$
1000	+	X	U.x	$U+V$	$U.x$	$(U+V).x^{-1}$
1001	-	X	$V.x$	$U+V$	V.x	$(U+V).x^{-1}$
1010	+	X	U	$(U+V).x^{-1}$	$U.x$	$(U+V).x^{-1}$
1011	-	X	V	$(U+V).x^{-1}$	V.x	$(U+V).x^{-1}$
1100	+	X	V	$U.x^{-1}$	$U.x^{-1}$	$U.x-2+V.x^{-1}$
1101	-	X	V	$U.x^{-1}$	V	$U.x-2+V.x^{-1}$
1110	+	X	$V.x$	U	$U.x$	$U+V.x$
1111	-	X	$V.x$	U	$V.x^2$	$U+V.x$
0000	+	X	$V.x$	U	U	$U.x^{-1}+V$
0001	-	X	$V.x$	U	$V.x$	$U.x^{-1}+V$
0010	+	X	V	$U.x^{-1}$	U	$U.x^{-1}+V$
0011	-	X	V	$U.x^{-1}$	$V.x$	$U.x^{-1}+V$
0100	+	X	U	$(U+V).x^{-1}$	U	(U+V)x-2+U.x-1
0101	-	+	V	$(U+V).x^{-1}$	(U+V).x-1	(U+V).x-2+V.x-1
0110	-	-	V	$(U+V).x^{-1}$	V	$(U+V).x^{-2}+V.x^{-1}$
0111	+	X	$U.x$	$U+V$	$U.x^2$	$U+V+U.x$
1000	-	-	$V.x$	$U+V$	$(U+V).x$	$U+V+V.x$
1001	-	X	$V.x$	$U+V$	$V.x^2$	$U+V+V.x$
1010	+	X	$U.x$	$U+V$	$U.x$	$(U+V).x^{-1}+U$
1011	-	+	$V.x$	$U+V$	$U+V$	$(U+V).x^{-1}+V$
1100	-	-	$V.x$	$U+V$	$V.x$	$(U+V).x^{-1}+V$
1101	+	X	U	$(U+V).x^{-1}$	$U.x$	$(U+V).x^{-1}+U$
1110	-	+	V	$(U+V).x^{-1}$	$U+V$	$(U+V).x^{-1}+V$
1111	-	-	V	$(U+V).x^{-1}$	$V.x$	$(U+V).x^{-1}+V$

of division, we only present an algorithm processing 3-bit in each step. Opening the loop by a factor of 3 leads to the process of three bits of R and Three bits of T along with a sign of three counters which results in 2_9=512 different states at all. A better approach is to combine the operation of RMD-2 with a bit-serial RMD which leads to 8×15 states for U and 8×10 states for V. The lack of space prevents us to bring all the details of computations. Our evaluations show that the corresponding states further can be reduced to 50 distinct states for U and 27 states for V. Table 3 summarizes the complexity of randomized multiplier, 1-bit RMM through 4-bit RMM, along with randomized divider, RMD-1bit through RMD-3.

4 Overall Architecture

In this section, the primitive operations of the processor will be discussed. Next, we propose two architectures for the ECC processor. The first is RMD-$\acute{k}$/RMM-k which is the Liao architecture with a little difference that RMM and RMD modules are replaced by RMM-k and RMD-$\acute{k}$, respectively. The second which is denoted by RMD2-k/RMM2-$\acute{k}$, is a new architecture with two multipliers and two divider modules of type RMM-k and RMD-$\acute{k}$.

4.1 Primitive Operations

The basic operations for the proposed modular multiplier/divider are classified into shifting right, bitwise XOR, and modular multiplication of $x^{\pm k}$. The first and second ones are trivial operations that are simply implemented in hardware, while the last operation needs to be represented in a convenient non-algebraic formula. Let $f(x)$ be the irreducible polynomial of degree m, represented by:

$$f(x) = f_0 + f_1x + .. + f_{m-1}x^{m-1} + x^m$$

It is implied that the coefficient of both x^m and x is 1 (f_0=1). Thus, we will have the following formulas for x^m and x^{-1}:

$$\begin{cases} x_m = f_0 + f_1x + .. + f_{m-2}x^{m-2} + f_{m-1}x^{m-1}(modf) \\ x_{-1} = f_1 + f_2x + .. + f_{m-1}x^{m-2} + f_0x^{m-1}(modf) \end{cases} \tag{6}$$

For the desired polynomial Q(x) of degree m, we can write Q(x).x as:

$$\begin{aligned} Q(x).x &= (q_0 + q_1x + .. + q_{m-1}x^{m-1})x \\ &= q_0x + q_1x^2 + .. + q_{m-2}x^{m-1} + q_{m-1}x^m \\ &= q_0x + q_1x^2 + .. + q_{m-2}x^{m-1} \\ &+ q_{m-1}(f_0 + f_1x + .. + f_{m-1}x^{m-1}) \\ &= q_{m-1}f_0 + (q_1 + q_{m-1}f_1) \\ &+ .. + (q_{m-2} + q_{m-1}f_{m-1})x^{m-1} \end{aligned} \tag{7}$$

$$\begin{aligned} Q(x).x^{-1} &= q_0x^{-1} + q_1 + q_2x.. + q_{m-1}x^{m-2} \\ &= q_0(f_1 + f_2x + .. + f_{m-1}x^{m-2} \\ &+ f_0x^{m-1}) + q_1 + q_2x.. + q_{m-1}x^{m-2} \\ &= (q_1 + q_0f_1) \\ &+ (q_2 + q_0f_2)x + .. + (q_{m-1} + q_0f_{m-1})x \\ &+ q_0f_0x^{m-1} \end{aligned} \tag{8}$$

According to Equations (7) and 8, both $Q(x).x$ and $Q(x).\ x^{-1}$ can be implemented in hardware by a single AND layer followed by a single XOR layer consisting of m gates AND plus m gates XOR. For higher degree of modular multiplication, we can use recursive multiplication, *i.e.*, $Q(x).x^{-2}$= $(Q(x).x).x$. Equations (9) and 11 respectively represent the evaluation for $Q(x).x^{-1}$, $Q(x).x^{-k}$, $Q(x).x$ and Q(x).x^k. This formulation implies that $Q(x).x^{\pm k}$ is implemented by subsequent of k modules, each of them consists of a AND layer followed by a XOR layer. The required elements include km gates XOR as well as km gates AND.

$$X(m,n) = \begin{cases} q_0f_0 & i = m-1 \\ q_{i+1} \oplus q_0f_{i+1} & 0 \le i \le m-2 \end{cases} \tag{9}$$

$$q_i^{-k} = \begin{cases} q_0^{-(k-1)}f_0 & i = m-1 \\ q_{i+1}^{-(k-1)} \oplus q_0^{-(k-1)}f_{i+1} & 0 \le i \le m-2 \end{cases} \tag{10}$$

$$q_i^1 = \begin{cases} q_{i-1} \oplus q_{m-1}f_i & 1 \le i \le m-1 \\ q_{m-1}f_0 & i = 0 \end{cases} \tag{11}$$

$$q_i^k = \begin{cases} q_{i-1}^{k-1} \oplus q_{m-1}^{k-1}f_i & 1 \le i \le m-1 \\ q_{m-1}^{k-1}f_0 & i = 0 \end{cases} \tag{12}$$

4.2 Operation Sequences

In this section, the sequence of operations for the ECC processor is discussed. The operations include both point addition and point doubling which are parallelly executed in the Montgomery Ladder algorithm. We will study two architectures denoted by RMD-k/RMM-$\acute{k}$ and RMD2-k/RMM2-$\acute{k}$, respectively. It is worth noting that both k and $\acute{k}$ are small integers which is not greater than 4.

The former consists of three addition modules, one multiplier of type RMM-$\acute{k}$ and one divider of type RMD-k. The latter benefits from replication of the multiplier and divider modules. It consists of three addition modules along with two multipliers of type

Algorithm 5 Randomized 2-Bit Divider(“X” Denotes Don’T Care Bit).

INPUT: $A(x), B(x), f(x), \alpha$

OUTPUT: $C(x) = A(x)/B(x).x^{-\beta} \bmod f(x)$

```
1: $n \leftarrow (m{+}1)$ when ($m$ is odd) else $m$;
2: $R, S, R_1, S_1, Count, Count2$ : $n{+}1$-bit Register;
3: $U, V, T$ : n-bit Register;
4: set $R \leftarrow B, S \leftarrow f, U \leftarrow A, V \leftarrow 0, T \leftarrow \alpha, Count \leftarrow 0$;
5: for $i \leftarrow 1$ to $n$ do
6:     $R_1 \leftarrow (R >> 1)$ when $R(0) =$ ‘0′ else $(R + S) >> 1$
7:     $S_1 \leftarrow R$ when ($R(0)$=’1’ and $Count \geqslant 0$) else $S$
8:     $Count2 \leftarrow -(Count + 1)$ when ($R(0)$=’1’ and $Count \geqslant 0$)
9:     else $(Count{+}1)$
10:    switch $R-1(0)R(0)T(1)T(0)sign(Count)$
11:    case ”0000X” : $U \leftarrow U.x^{-2}$ mod $f$ ;
12:    case ”0011X” : $U \leftarrow U$;
13:    case ”0010X” , ”0001X” : $U \leftarrow U.x^{-1}$ mod $f$;
14:    case ”0100X” : $U \leftarrow (U + V).x^{-2}$ mod $f$;
15:    case ”0111X” : $U \leftarrow U + V$;
16:    case ”0101X” , ”0110X” : $U \leftarrow (U + V).x^{-1}$ mod $f$;
17:    case ”0011X” : $U \leftarrow U.x^{-2} + V.x^{-1}$ mod $f$ ;
18:    case ”1011X” : $U \leftarrow U + (U + V).x$ mod $f$;
19:    case ”1001X” , ”1010X”: $U \leftarrow U.x^{-1}$ mod $f + V$;
20:    case ”11000” : $U \leftarrow (U + V).x^{-2} + U.x^{-1}$ mod $f$;
21:    case ”11001” : $U \leftarrow (U + V).x^{-2} + V.x^{-1}$ mod $f$;
22:    case ”11110” : $U \leftarrow U + V + U.x$ mod $f$;
23:    case ”111111” : $U \leftarrow U + V + V.x^{-2}$ mod $f$;
24:    case ”11010” , ”11100” : $U \leftarrow (U + V).x^{-1}$ mod $f + U$;
25:    case ”11011” , ”11101” : $U \leftarrow (U + V).x^{-1}$ mod $f + V$;
26:    end switch;
27:    switch $R_1(0)R(0)T(1)T(0)$ sign($Count$)sign($Count2$)
28:    case ”0000XX” , ”01001X” , ”10001X”,
29:    ”110011” : $V \leftarrow V$ ;
30:    case ”0011XX” , ”01111X” , ”10111X”,
31:    ”111111”: $V \leftarrow V.x^2$ mod $f$;
32:    case ”0010XX” , ”0001XX” , ”01011X” , ”01101X”,
33:    ”10011X” , ”10101X” , ”110111”,
34:    ”111011” : $V \leftarrow V.x$ mod $f$;
35:    case ”01000X” , ”10010X” , ”11000X” ,
36:    ”10100X” : $V \leftarrow U$;
37:    case ”01110X” , ”11110X” : $V \leftarrow U.x^2$ mod $f$
38:    case ”01010X” , ”01100X” , ”10110X” , ”11010X” , ”11100X” : $V \leftarrow U.x$ mod $f$
39:    case ”10000X” : $V \leftarrow U.x^{-1}$ mod $f$;
40:    case ”110010” : $V \leftarrow (U + V).x^{-1}$ mod $f$;
41:    case ”111110” : $V \leftarrow U.x + V.x$ mod $f$;
42:    case ”110110” , ”111010” : $V \leftarrow U + V$;
43:    end switch;
44:    $T \leftarrow T >> 2$ ;
45:    $R \leftarrow (R_1 >> 1)$ when $R_1(0) =$ ‘0′ else $(R_1 + S_1) >> 1$
46:    $S \leftarrow R_1$ when ($R_1(0) =$ ‘1′ and $Count2 \geqslant 0$) else $S_1$
47:    $Count \leftarrow -(Count2 + 1)$ when ($R_1(0) =$ ‘1′ and $Count2 \geqslant 0$)
48:    else $(Count2{+}1)$
49: end for
50: return $V$
```

Table 3. The Specifications of Bit-Parallel RMM and RMD.

Module	Distinct states				Combinational units	XOR fan-in
	Q	S	U	V		
RMM-1	4	2	-	-	$Q.x^{-1},\ S.x^{\pm1}$	2
RMM-2	12	3	-	-	$Q.x^{-1}, Q.x^{-2}, S.x^{\pm1}, S.x^{\pm2}$	3
RMM-3	33	4	-	-	$Q.x^{-1}, Q.x^{-2}, Q.x^{-3}, S.x^{\pm1}, S.x^{\pm2}, S.x^{\pm3}$	4
RMM-4	111	5	-	-	$Q.x^{-1}, Q.x^{-2}, Q.x^{-3}, Q.x^{-4}, S.x^{\pm1}, S.x^{\pm2}, S.x^{\pm3}, S.x^{\pm4}$	5
RMM-1	-	2	4	4	$V.x^{-1}, U.x^{\pm1}$	2
RMM-2	-	4	15	10	$V.x^{-1}, V.x^{-2}, U.x^{\pm1}, U.x^{\pm2}$	3
RMM-3	-	8	50	27	$V.x^{-1}, V.x^{-2}, V.x^{-3}, U.x^{\pm1}, U.x^{\pm2}, U.x^{\pm3}$	4

RMM-$\acute{k}$ and two dividers of type RMD-k.

4.2.1 RMD-k/RMM-$\acute{k}$

The flow diagram of the operation sequence for both point addition and doubling is shown in Figure 1. The sequences are the same as Liao architecture except that the RMD/RMM module is replaced by RMD-k/RMD-$\acute{k}$. The inputs of point addition are denoted by (x_1, y_1) and (x_2, y_2). The point doubling has a single point which is assumed to be the first point, *i.e.*, (x_1, y_1). The goal is the computation of point-addition and doubling according to Equations (4) and 5. The outputs of the unit are respectively shown by (x_3, y_3) for addition and (x_4, y_4) for doubling.

The unit has eight m-bit register chains (denoted by A to H) and seven states (denoted by state1 to state7). The four input coordinates are latched to the corresponding register chains in State1, and the results of point addition and doubling are finally obtained in State7. In each state, different operations are executed to generate fresh data and update the relevant register chains. The operations in each state are done by a subset of one RMD, one RMM, and three RMA modules (RMA1, RMA2, and RMA3).

The delay of both state2 and state3 is identified by the clocks required to complete the RMD computations, *i.e.* equals to $2m/k$. While the delay of three subsequent states (4, 5, and 6) are identified by the clock numbers consumed by the RMM module, *i.e.*, $m/\acute{k}$. Thus, the total clock number required for one round of point-addition/doubling equals $4m/k + 3m/\acute{k}$+1.

Since the RMLA scalar point multiplication needs (m-1) iterations, the total number of clock cycles required for one scalar multiplication can be calculated as:

$$\begin{aligned} N_{RMD-k/RMM-\acute{k}} &= \\ (m-1) \times (4m/k + 3m/\acute{k} + 1) \\ +10m/k + 5m/\acute{k} + 2 \end{aligned} \tag{13}$$

where the first part is the cost of the main iterations of RMLA. The second part is the clock cycles required for coordinates transformations and the initialization of RMLA including:

1. Two RMD before the main loop for transforming point to the randomized domain $\frac{4m}{k}$
2. Two RMD for transforming constant a and 1 to the randomized domain $\frac{4m}{k}$
3. Two RMM after the main loop $\frac{2m}{\acute{k}}$
4. A single point doubling before the main loop ($\frac{2m}{k}$ + $\frac{3m}{\acute{k}}$ + 1)
5. Assignment of the inputs to register chains (1)

By simplification of the above formula, we will have:

$$\begin{aligned} N_{RMD-k/RMM-\acute{k}} &= \\ (4/k + 3/\acute{k})m^2 \\ +(6/k + 2/\acute{k} + 1)m + 1 \end{aligned} \tag{14}$$

Setting $\acute{k}$=k=1 leads to delay of Liao architecture which equals $7m^2$+9m+1. For example, RMD-2/RMM-3 has a runtime equal to $3m^2$+4.7m+1 clock cycles, *i.e.*, approximately 57% reduction relative to Liao design (($3m^2$+4.7m+1)/($7m^2$+9m+1)≈3/7).

4.2.2 RMD^2-k/RMD^2-$\acute{k}$

The same as the previous section, the unit has eight m-bit register chains (denoted by A to H) and seven states (denoted by state1 to state7).

The registers A through E are used for point addition while remained registers (F, G, and H) are used for point doubling. The four input coordinates are latched to the corresponding register chains in State1, and the results of point addition and doubling are finally obtained in State7. The operations are done by two RMD modules (RMD1 and RMD2), two RMM modules (RMM1 and RMM2), and three RMA modules (RMA1, RMA2, and RMA3). In each state, some of the modules are active and thus generate fresh data updating the corresponding register chains. Both RMD modules are only active in state3. While RMM modules are active during state2 through state5. Table 4 shows the sequence of operations for each active module in each state. The flow diagram of operation sequences is depicted in Figure 2.

The delay of both state2 and state7 is the delay of the RMA module, *i.e.*, 1 clock cycle. Further, for the state3, delay of RMD is the major, *i.e.* equals to 2m/k. Moreover, the delay of three subsequent states (4, 5, and 6) are identified by the clock numbers consumed by the RMM module, *i.e.*, $m/\acute{k}$.

Thus, the total clock number required for one round of point-addition/doubling equals $2m/\acute{k}$+$2m/\acute{k}$+3. Since the RMLA scalar point multiplication needs (m-1) iterations, the total number of clock cycles required for one scalar multiplication can be calculated as

$$\begin{aligned} N_{RMD^2-k/RMM^2-\acute{k}} &= \\ (m-1)\times(2m/k+2m/\acute{k}+3) \\ +6m/k+3m/\acute{k}+4 \end{aligned} \qquad (15)$$

where the first part is the cost of the main iterations of RMLA. The second part is the clock cycles required for coordinates transformations and the initialization of RMLA including:

1. Two RMD before the main loop, parallel execution with two RMD modules $\frac{4m}{k}$
2. Two RMD for transforming constant a and 1 to the randomized domain, parallel execution $\frac{4m}{k}$
3. Two RMM after the main loop, parallel execution with two RMM modules $\frac{2m}{\acute{k}}$
4. A single point doubling before the main loop ($\frac{2m}{k}$ + $\frac{2m}{\acute{k}}$ + 1)
5. Assignment of the inputs to register chains (1)

By simplification of the above formula, we will have:

$$\begin{aligned} N_{RMD^2-k/RMM^2-\acute{k}} &= \\ (2/k+2/\acute{k})m^2 \\ +(4/k+1/\acute{k}+3)m+1 \end{aligned} \qquad (16)$$

For example, the required clock pulses for RMD^2-2/RMD^2-4 becomes $5m^2$+5.5m+1. Compared with the Liao design, it is approximately equal to a 79% reduction in the number of clock pulses.

Table 4. The Data Flow In RMD^2-k/RMM^2-$\acute{k}$.

state	module	Output result
$1\rightarrow 2$	RMA1	x_1+x_2
	RMA2	y_1+y_2
	RMA3	y_2+a
$2\rightarrow 3$	RMA1	x_1+a
	RMA2	x_1+x_2+a
	RMA3	$x_1+x_2+y_2+a$
	RMM1	x_1^2
	RMD1	$((y_1+y_2))/((x_1+x_2))$
	RMD2	y_1/x_1
$3\rightarrow 4$	RMA1	$x_1+y_1\ /x_1$
	RMA2	x_1^2+a
	RMM1	$[((y_1+y_2))/((x_1+x_2))]^2$
	RMM2	$(y_1/x_1)^2$
$4\rightarrow 5$	RMA1	$x_1+a+[((y_1+y_2))/((x_1+x_2))]^2$
	RMA2	$((y_1+y_2))/((x_1+x_2)+x_1+x_2+a)$
	RMA3	x_1^2+$(y_1/x_1)^2+a$
$5\rightarrow 6$	RMA1	x_3
	RMA2	x_4
	RMM1	$[x_1+a+1+[((y_1+y_2))/((x_1+x_2))]^2].((y_1+y_2))/((x_1+x_2))$
	RMM2	$[x_1^2+(y_1/x_1)^2+a+1].(x_1+y_1/x_1)$
$6\rightarrow 7$	RMA1	y_3
	RMA2	y_4

4.3 Putting It All Together

Based on the operation sequence mentioned above, the overall architecture of the proposed ECC coprocessor in the case of redundant multiplier and divider modules are given in Figure 3. The main parts consist of the following components:

- Arithmetic modules including RMAs, RMMs, and RMDs.
- Control and support modules
- Storage modules
- Load and Store modules.

The storage modules include the scalar registers for parameter k, the register for irreducible polynomial f, curve coefficients a and b, and the register chains A through H for maintaining the internal values of the point addition/doubling iterations. All of these registers have m-bit or a slightly higher width (*e.g.* $m+1$). For RMD-k/RMM-$\acute{k}$ design, the arithmetic part consists of one RMD, one RMM module, and three RMA modules. While in RMD^2-k/RMD^2-$\acute{k}$ we have two RMD and two RMM in addition to three RMA modules. These modules fetch the corresponding data from the data buses and complete the intended arithmetic operations.

The Load/Store module is responsible either for reading the required data including point P and scalar k from or writing the multiplication result outside the processor. The data are fed into the system in chunks of 64 bits and controlled by the signal Data_in indicating a 64-bit portion of input data, Data_Out indicating a 64-bit portion of the output result, RegNo indicating the source/destination register, Part64 indicating the chunk counter, and Load/Store indicating the direction of data flow.

The Load/Store module has a key role to fix the required primary input/output pins of the module independent of the size of the field (m). Further, it makes the synthesis possible on a device with a limited number of in/out pins.

Further, the control unit generates control signals at different operation stages. It is activated by the external enable signal Start, reset to initial state, and then run automatically based on the built-in finite state machine, loop counters, and the feedback from the least significant bit of the scalar k. The PRNG module is responsible for the generation of m-bit random α. It is simply implemented by an m-bit linear feedback shift register (LFSR) with the following recurves equation:

$$z_m = z_{m-1} \oplus z_0 \tag{17}$$

This parameter is always fed into the arithmetic modules as a required input. The data exchange module is a shared bus that provides the flow of data from a single register to either another register or an arithmetic module. Further, as mentioned earlier, it facilitates the exchange of data outside the coprocessor via the Load/Store module.

The Load signals enable the arithmetic modules to start the demanded arithmetic operations, the selection signal Sel instructing the management module of α to stop randomization and outputs the current value for α. On the other hand, output control signal EReady indicates the finishing of MLA and the readiness of output result for locating on the data bus and accordingly for transferring to outside of the coprocessor via Store module. Note that the state machines for RMD-k/RMM-$\acute{k}$ and $/RMD^2$-$k//RMM^2$-$\acute{k}$ are different.

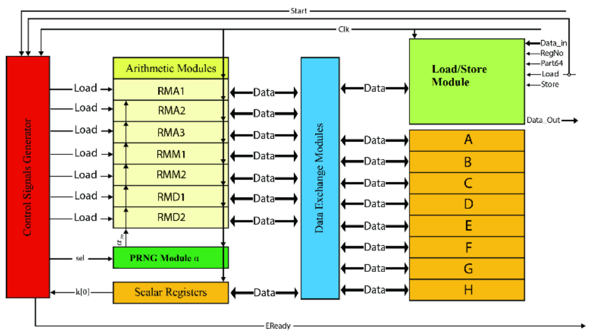


Figure 3. The Architecture of ECC Coprocessor With Replicative Modules ($RMD^2 - k/RMM^2$-$\acute{k}$).

The operation sequence for RMD-mathalphak/RMM-$\acute{k}$ design is similar to the Liao ECC coprocessor [11]. While for RMD2-k/RMM2-$\acute{k}$ design, the operation sequence corresponds to data flow depicted in Figure 2 and identified by more details in Table 4.

It is worth noting when the system is idle, most parts of the coprocessor are inactive to reduce the power consumption. When the input enables signal Start to become high, the system is activated to execute the demanding process including coordinate transformation and accordingly beginning of MLA loop. Subsequently, the coprocessor puts the output result on the data bus and makes the signal Eready valid to inform the unit requests. Finally, the system returns to an idle state waiting for a new command.

5 Evaluations

5.1 Implementation Details

For comparison, we simulate and synthesis the proposed ECC processor in Xilinx Vivado. The target device selected for synthesis is Zynq-7000. We use the 163-bit binary elliptic curve recommended by NIST, namely B-163, over binary extension field GF(2^{163}). Although our ECC coprocessor is flexible to implement the desired curve on any binary field, we choose NIST 163-bit ECC just for comparison with other work. The runtime for one scalar multiplication is calculated by the number of clock cycles from Equations (14) and 16. Further, the clock time was obtained from the synthesis of the design.

5.2 Evaluation Results on FPGA

Table 5 summarizes the synthesis results and makes a comparison with the Liao coprocessor on corresponding metrics. The columns of the table, for both design RMD-k/RMM-$\acute{k}$ and RMD^2-k/RMM^2-$\acute{k}$, respectively show:

- the number of slices used for each architecture,

Table 5. FPGA Implementation Results of ECC Coprocessor Over GF(2^{163}) on Zynq-7000.

DESIGN	RMD-k/RMM-$\acute{k}$						$RMD^2 - k/RMM^2$-$\acute{k}$					
$k/\acute{k}$	Number of Slices	Clock Period (ns)	Clock cycles	Runtime (ms)	Speed (KP)/s	Time× Slice	Number of Slices	Clock Period (ns)	Clock cycles	Runtime (ms)	Speed (KP)/s	Time× Slice
1/1 (Liao)	4987	3.97	187,451	0.74	1344	3.71	7508	4.10	107,579	0.44	2267	3.31
1/2	5151	4.00	147,761	0.59	1692	3.04	8023	4.16	81,173	0.34	2961	2.71
1/3	5281	4.46	134,531	0.60	1667	3.17	7905	4.65	72,371	0.34	2972	2.66
1/4	6794	5.59	127,671	0.71	1401	4.85	10589	5.78	67,807	0.39	2552	4.15
2/1	5523	4.87	134,153	0.65	1531	3.61	7713	4.80	80,849	0.39	2577	2.99
2/2	5623	4.91	94,463	0.46	2156	2.61	7859	4.89	54,443	0.27	3756	2.09
2/3	5725	4.90	81,233	0.40	2512	2.28	8254	4.95	45,641	0.23	4426	1.86
2/4	7104	4.92	74,373	0.37	2733	2.60	12285	5.58	41,077	0.23	4363	2.82
3/1	6367	5.37	116,387	0.62	1600	3.98	9715	5.84	71,939	0.42	2380	4.08
3/2	6631	5.64	76,697	0.43	2312	2.87	9516	6.29	45,533	0.29	3492	2.73
3/3	6735	5.31	63,467	0.34	2967	2.27	10304	5.97	36,731	0.22	4560	2.26
3/4	9149	6.18	56,607	0.35	2859	3.20	13992	6.15	32,167	0.20	5055	2.77

- the clock period in nano-seconds,
- the number of clock cycles for a scalar multiplication acquired by formula (15),
- the runtime of one scalar multiplication in milliseconds,
- speed of operation in several operations per second,
- production of time and slices count, *i.e.*, runtime×slices

Further, Figures 4 and 5 respectively visualize this comparison based on runtime and time×slice metrics.

We can see that along with the increase in complexity of the circuit basically in combinational parts, the clock time is increased up to 56% relative to Liao architecture. The maximum clock which is equal to 6.18 ns belongs to RMD-3/RMM-4. This is due to the critical path that passes through the combinational part of either RMD or RMM modules. Those have primitive operations such as $U.x^{\pm 3}$ and $S.x^{\pm 4}$ which impose 3 and 4 consecutive layers of AND+XOR to the critical path.

On the other hand, the area has more variation than the clock period for a different architecture. The range of variations is between 3% to 83%. The maximum slices (9149) are used for RMD-3/RMM-4 design which shows 83% of resource increment relative to the Liao design, while the minimum difference happens for RMD-1/RMM-2. More specifically by fixing k, there is no significant difference between the area consumed for RMD-k/RMM-1 and RMD-k/RMM2.

Column 5, namely runtime, shows the execution time for a single scalar multiplication ($k.P$) obtained by the production of corresponding required clock cycles and the clock period. From this point of view, a single scalar point multiplication is done in 0.35 up to 0.74 milliseconds for different configurations. We can see in Figure 4 that increasing of either k or $\acute{k}$ leads to reducing runtime. The maximum reduction, about 55%, holds for RMD-3/RMM-3. Despite RMD-3/RMM-4 requires fewer clock cycles, the resulting runtime for RMD-3/RMM-3 is better than RMD-3/RMM-4 due to its shorter clock time.

The preceding column shows the speed of operation, *i.e.*, the number of multiplications per second. It varies from 1344 up to 2859 operations per second. Further, we can see that operation speedup for RMD-1/RMM-2 is equal to 26% without any significant increase in the area. Moreover, the speedup reaches to max-value of 120% for RMD-3/RMM-3 by accepting about 34% increasing of the area.

The last column compares the different designs based on the main optimization criteria, *i.e.*, Time×Slice, which criteria considers both time and area objectives. Further, Figure 5 shows the comparison of different designs based on this criterion. We see that RMD-3/RMM-3 design reduces the Time×Area by a factor of 55% to Liao design. Moreover, we can see a repetitive pattern by fixing k in RMD-k, that is the Time×Area objective is gradually reduced from RMD-k/RMM-1 to RMD-k/RMM-3 while suddenly

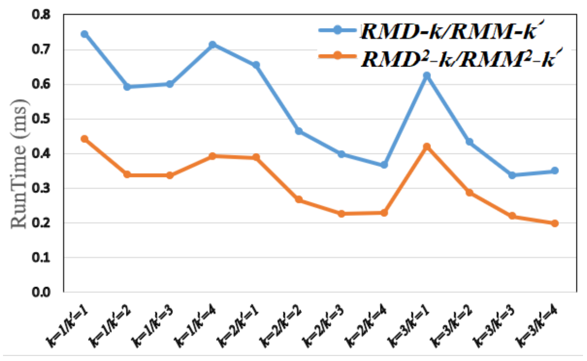


Figure 4. Comparison in Terms of Time Metric.

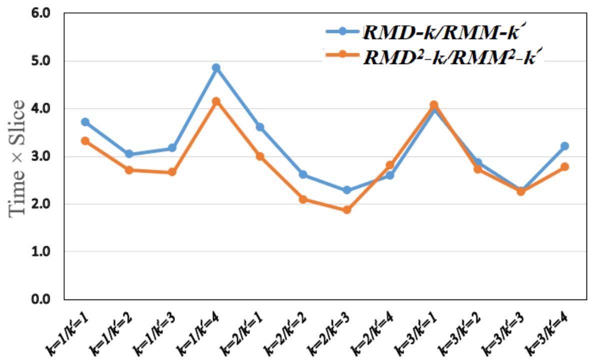


Figure 5. Comparison in Terms of Time×Slice Metric.

increased in RMD-k/RMM-4. Thus, the optimum result for a specific k is obtained for RMD-k/RMM-3 design. This is due to the complexity of RMM-4 relative to RMM-3 in terms of enlarging multiplexers (*e.g.*, 111 different Q inputs relative to 33 ones) and modular multiplication of $x^{\pm 4}$.

The above-mentioned experiments are repeated for $RMD^2 - k/RMM^2$- $\acute{k}$ designs. We can see that increasing of either k or $\acute{k}$ leads to increasing slices and operation speed as well as reducing runtime and Time×Slice objective relative to $RMD^2 - 1/RMM^2$-1. For a better, we focus on the comparison of each RMD^2-k/RMM^2- $\acute{k}$ with corresponding RMD-k/RMM- $\acute{k}$. First of all, we see that there is no significant difference between the clock period for $RMD^2 - k/RMM^2$- $\acute{k}$ and RMD-k/RMM- $\acute{k}$ since replication does not have a direct effect on the critical path.

Secondly, the number of slices is increased by a factor of 40% to 70% relative to non-replicative design. Despite the resource increasing, we can see an average of 40% improvement in runtime due to fewer clock cycles.

From the operation speed point of view, the number of scalar multiplications per second is increased by a factor between 50% to 82%. Since the average gain of operation speedup (68%) is larger than the average of resource increasing (50%), the replicative design outperforms non-replicative ones regarding the Time×Area objective. As we see in Table 5, the improvement of design based on replication is up to 20% over the corresponding non-replication design. The maximum gain equals 20%, is obtained by $RMD^2 - 2/RMM^2$-2 relative to RMD-2/RMM-2. The gain resulted for the optimum design, *i.e.*, $RMD^2 - 2/RMM^2$-3, is equal to 18% relative to RMD-2/RMM-3 which is the best non-replicative design.

In summary, we can argue that increasing the parallelism factor over 3-bits does not lead to better Time×Slice performance. This is happened due to the logic complexity of k-bit multiplier and divider which in its turn increased both the clock time and the number of slices. As mentioned above, the optimum performance is derived by both RMD-2/RMM-3 and RMD-3/RMM-3. As the latter requires fewer resources than the former, we prefer RMD-2/RMM-3. Further, the same thing is true for replicative design and we got the best performance by $RMD^2 - 2/RMM^2$-3.

5.3 Evaluation Results on ASIC

Table 6. summarizes the synthesis results and makes a comparison with the Liao coprocessor on corresponding metrics. A different version of our proposed ECC coprocessor was implemented by UMC 0.18-μm CMOS technology. Moreover, to compare with known previous works, we focused on a 163-bit version of ECC coprocessor in GF(2^{163}). The layout and synthesis results along with the comparisons are given in Table 5. The first column of the table indicates the design and the preceding ones are respectively shown:

- the CMOS Technology used,
- the Field wherein implementation is done,
- the core area,
- the equivalent gates which is approximated by dividing the design core area to the area of 2-inputs NAND gate.
- the clock period in nano-seconds,
- the number of clock cycles for a scalar multiplication acquired by formula (15),
- the runtime of one scalar multiplication in milliseconds,
- the normalized Area Time Product. First, computation is done as ATP = Equivalent gates × (Time× λ), where λ is the technology ratio. Then, the results are normalized to ATP obtained for RMD-1/RMM-1.

Note that RMD-1/RMM-1 is our implementation of Liao bit-serial ECC coprocessor which is slightly better than the basic implementation reported in [11]. In

Table 6. ASIC Comparison of Our ECC Coprocessor With Previous Well-Known Work.

DESIGN	Tech.	Field	Core Area (mm^2)	Equivalent Gates (k)	Clock Period (ns)	Clock cycles	Runtime (ms)	Normalized ATP	security
RMD-1/RMM-1	0.18 μ m	GF(2^{163})	1.98	82.7	3.9	187,451	0.73	1	SPA and DPA resistant
RMD-1/RMM-2			2.15	89.5	3.89	147,761	0.57	0.85	
RMD-2/RMM-2			2.31	96.5	6.29	94,463	0.59	0.94	
RMD-2/RMM-3			2.64	109.9	6.29	81,233	0.51	0.93	
RMD^2-1/RMM^2-1			2.7	112.4	3.9	107,579	0.42	0.78	
RMD^2-1/RMM^2-2			3.02	126.0	3.89	81,173	0.32	0.67	
RMD^2-2/RMM^2-2			3.4	141.5	6.29	54,443	0.34	0.80	
RMD^2-2/ RMM^2-3			4.04	168.3	6.29	45,641	0.29	0.81	
Liao [11]	0.13 μm	GF(2163)	0.28	57.4	4.25	187,451	0.80	1.05	Noninvasive SCAs resistant
Lee [15]	90 nm	GF(2^{160})	0.21	61.3	3.61	168970	0.61	1.24	SPA and CPA resistant
Lee [16]	90 nm	GF(2^{160})	0.55	170	5.32	216,200	1.15	6.48	SPA and DPA resistant
Jyu [17]	0.13 μm	GF(2^{160})	1.44	169	6.85	54,319	0.37	1.43	SPA resistant

terms of ATP, all of our coprocessor variants outperform other approaches. In contrast to FPGA designs, we consider only four variants for non-replicative designs (RMD-k/RMM-$\acute{k}$) as well as four versions of replicative ones (RMD^2-k/RMM^2-$\acute{k}$). The optimum result in terms of ATP is obtained by RMD-1/RMM-2 with 19% for non-replicative designs. While, for replicative design, the best gain is obtained by RMD^2-1/RMM^2-2 with 36%, both relative to Liao coprocessor.

Other designs have less ATP performance. Although the runtime is gradually reduced by increasing either k or $\acute{k}$, the effect of increasing both clock period and core area causes the overall ATP is not to decrease as well. More specifically, for RMD-2/RMM-3, we can see a 69% increase in clock time relative to RMD-1/RMM-1 which bounds the runtime gain to 30% and accordingly the overall ATP gain to only 7%. The same thing happens for replicative designs. We can observe about 69% of clock time increasing for RMD^2-2/RMM^2-3 which cuts the runtime gain to 29% relative to $RMD^2 - 1/RMM^2$-1. Finally, the overall ATP performance is 4% worse than RMD2-1/RMM2-1 since the core area of RMD2-2/RMM2-3 shows about 50% growth (4.04 vs 2.7). The resulted gain for larger k and $\acute{k}$ such as RMD-3/RMM-3 is even worse which prohibits us to bring them to the table.

In addition to the higher ATP performance results, the major fact is the security margin of our coprocessor. It is resistant to simple and differential power attacks, *i.e.* the intruder is not capable of revealing the private key by the methods such as TA attacks, SPA, DPA, and CPA attacks.

6 Discussions and Conclusions

In this paper, a bit-parallel ECC coprocessor resistant to differential power attacks was proposed. It acts in the binary field along with the operations done in the randomized Montgomery domain which makes it impossible to create differential power attacks by involving a random number in the calculation process. We proposed 2-bit and 3-bit randomized divider as well as 2 to 4-bit randomized multiplier modules. Despite the complexity of the logic in the multi-bit modules, the speed was significantly improved by accepting overhead in the area resource. We evaluated our design variants both in FPGA and ASIC implementations.

In FPGA we got a better result than ASIC mainly because the critical path in FPGA includes net delay rather than logic delay in ASIC. However, ASIC evaluations in term of Time×Area metric showed a gain of about 19% over the previous well-known work. A version of our design with duplication improved the overall gain up to 36%.

Further, the FPGA evaluations showed that the 2-bit divider/3-bit multiplier version of our architecture could lead to 40% improvement over the best previous work in terms of the Time×Slice metric. Further, by duplicating the divider and multiplier modules along with the bit-parallel architecture this gain could reach 50%.

References

[1] A. H. Koblitz, N. Koblitz, and A. Menezes. Elliptic curve cryptography: The serpentine course of a paradigm shift. *Journal of Number theory*, 131 (5):781–814, 2011. doi:10.1016/j.jnt.2009.01.006.

[2] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[3] P. Choi, M. Lee, J. Kim, and D. Kim. Low-complexity elliptic curve cryptography processor based on configurable partial modular reduction over nist prime fields. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(11):1703–1707, 2017. ISSN 1558-3791. doi:10.1109/TCSII.2017.2756680.

[4] Z. Liu, X. Huang, Z. Hu, M. K. Khan, H. Seo, and L. Zhou. On emerging family of elliptic curves to secure internet of things: Ecc comes of age. *IEEE Transactions on Dependable and Secure Computing*, 14(3):237–248, 2016. ISSN 1941-0018. doi:10.1109/TDSC.2016.2577022.

[5] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999. ISBN 978-3-540-66347-8. doi:10.1007/3-540-48405-1_25.

[6] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004. ISBN 978-3-540-22666-6. doi:10.1007/978-3-540-28632-5_2.

[7] X. Fan, S. Peter, and M. Krstic. Gals design of ecc against side-channel attacks—a comparative study. In *2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–6. IEEE, 2014. ISBN 978-1-4799-5412-4. doi:10.1109/PATMOS.2014.6951905.

[8] P. C. Liu, H. C. Chang, and C. Y. Lee. A true random-based differential power analysis countermeasure circuit for an aes engine. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(2):103 – 107, 2012. ISSN 1549-7747. doi:10.1109/TCSII.2011.2180094.

[9] M. Joye and C. Tymen. Protections against differential analysis for elliptic curve cryptography. In *international workshop on cryptographic hardware and embedded systems*, pages 377–390. Springer, 2001. ISBN 978-3-540-42521-2. doi:10.1007/3-540-44709-1_31.

[10] J. Lee, J. Hsiao, H. Chang, and C. Lee. An efficient dpa countermeasure with randomized montgomery operations for df-ecc processor. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(5):287 – 291, 2012. ISSN 1549-7747. doi:10.1109/TCSII.2012.2190857.

[11] K. Liao, X. Cui, N. Liao, T. Wang, D. Yu, and X. Cui. High-performance noninvasive side-channel attack resistant ecc coprocessor for gf (2m). *IEEE Transactions on Industrial Electronics*, 64(1):727 – 738, 2016. ISSN 0278-0046. doi:10.1109/TIE.2016.2610402.

[12] Z. Khan and M. Benaissa. High speed and low latency ecc implementation over gf(2m) on fpga. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1):165 – 176, 2017. ISSN 1063-8210. doi:10.1109/TVLSI.2016.2574620.

[13] W. N. Chelton and M. Benaissa. Fast elliptic curve cryptography on fpga. *IEEE transactions on very large scale integration (VLSI) systems*, 16(2):198 – 205, 2008. ISSN 1063-8210. doi:10.1109/TVLSI.2007.912228.

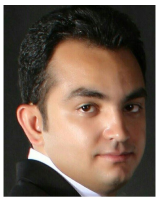
[14] K. Liao, X. Cui, N. Liao, T. Wang, X. Zhang, Y. Huang, and D. Yu. High-speed constant-time division module for elliptic curve cryptography based on gf(2m). In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 818–821. IEEE, 2014. ISBN 978-1-4799-3432-4. doi:10.1109/ISCAS.2014.6865261.

[15] J. Lee, S. Chung, H. Chang, and C. Lee. An efficient countermeasure against correlation power-analysis attacks with randomized montgomery operations for df-ecc processor. In *International Workshop on Cryptographic Hardware and Em-*

bedded Systems, pages 548–564. Springer, 2012. ISBN 978-3-642-33026-1. doi:10.1007/978-3-642-33027-8_32.

[16] J. Lee, Y. Chen, C. Tseng, H. Chang, and C. Lee. A 521-bit dual field elliptic curve cryptographic processor with power analysis resistance. In *2010 Proceedings of ESSCIRC*, pages 206–209. IEEE, 2010. ISBN 978-3-642-33026-1. doi:10.1109/ESSCIRC.2010.5619893.

[17] J. Lai and C. Huang. A highly efficient cipher processor for dual-field elliptic curve cryptography. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 56(5):394 – 398, 2009. ISSN 1549-7747. doi:10.1109/TCSII.2009.2019327.



Hashem Rezaei was born in Torbat-Jam, Khorasan, Iran in 1987. He received the B.S. degree in computer engineering from Payam-Noor University, Torbat-Jam branch, in 2015. He was also received M.Sc degree in computer engineering from Tarbiat Modares University, Tehran, Iran, in 2019. Since 2019, he is a researcher in SE-Lab (Security Evaluation Laboratory) at Tarbiat Modares University. His research area includes Network Security and Hardware implementation of Cryptography Algorithms.

Alireza Shafieinejad was born in Isfahan, Iran in 1977. He received the B.S. degree in computer engineering from the Sharif University of Technology, Tehran, in 1999. He was also received M.Sc and Ph.D degrees in electrical engineering from Isfahan University of Technology, Isfahan, Iran, respectively in 2002 and 2013. Since 2015, he has been an Assistant Professor with the Electrical and Computer Engineering Department, Tarbiat Modares University, Tehran, Iran. His research interests are in the area of Wireless Network Coding, Network Security, Design, and Analysis of Cryptography Algorithms. At Tarbiat Modares University, he leads research in network security and penetration test in SE-Lab (Security Evaluation Laboratory).