Low-Resource and Fast Binary Edwards Curves Cryptography

Brian Koziel[1], Reza Azarderakhsh[1], and Mehran Mozaffari-Kermani[2]

[1] Computer Engineering Department
[2] Electrical and Microelectronic Engineering Department
Rochester Institute of Technology
Rochester, NY 14623, USA
{bck6520, rxaeec, mmkeme}@rit.edu.

Abstract. Elliptic curve cryptography (ECC) is an ideal choice for low-resource applications because it provides the same level of security with smaller key sizes than other existing public key encryption schemes. For low-resource applications, designing efficient functional units for elliptic curve computations over binary fields results in an effective platform for an embedded co-processor. This paper proposes such a co-processor designed for area-constrained devices by utilizing state of the art binary Edwards curve equations over mixed point addition and doubling. The binary Edwards curve offers the security advantage that it is complete and is, therefore, immune to the exceptional points attack. In conjunction with Montgomery Ladder, such a curve is naturally immune to most types of simple power and timing attacks. The recently presented formulas for mixed point addition in [1] were found to be invalid, but were corrected such that the speed and register usage were maintained. We utilize corrected mixed point addition and doubling formulas to achieve a secure, but still fast implementation of a point multiplication on binary Edwards curves. Our synthesis results over NIST recommended fields for ECC indicate that the proposed co-processor requires about 50% fewer clock cycles for point multiplication and occupies a similar silicon area when compared to the most recent in literature.

Key Words: Crypto-processor, Binary Edwards curves, Gaussian Normal Basis, Point Multiplication, Low-resource Devices.

1 Introduction

Deeply-embedded computing systems, nowadays, are essential parts of emerging, sensitive applications. With the transition to the Internet of Things (IoTs), where all tools and electronics will be linked wirelessly, there is a need to secure these devices from malicious intent. However, these devices are mainly designed in such a way that the functionality and connectivity monopolize the device's area and power. Little power and area are allocated for the establishment of security. Therefore, a secure co-processor that can fill this niche in the current technological world is necessary for the evolution of achieving security for IoTs in the near future.

Elliptic Curve Cryptography (ECC) is the ideal implementation for this application because it provides a secure application for far fewer bits than RSA and other public key encryption schemes. ECC provides key exchange ECDH, authentication ECDSA, and encryption ECIES protocols. An elliptic curve is composed of all points that satisfy an elliptic curve equation as well as a point at infinity. This forms an Abelian group, E, over addition, where the point at infinity represents the zero element or identity of the group. The most basic operations over this Abelian group are point addition and point doubling. Using a double-and-add method, a point multiplication, $Q = kP$, where $k \in \mathbb{Z}$ and $Q, P \in E$, can be computed quickly and efficiently. Protocols implemented over ECC rely on the difficulty to solve the elliptic curve discrete logarithm problem (ECDLP), that given Q and P in $Q = kP$, it is infeasible to solve for k [2]. For the computations of ECC, several parameters should be considered including representation of field elements and underlying curve, choosing point addition and doubling method, selecting coordinate systems such as affine, projective, Jacobian, and mixed, and finally arithmetic

(addition, inversion, multiplication, squaring) on finite field. Field multiplication determines the efficiency of point multiplication on elliptic curves as its computation is complex and point multiplication requires many field multiplications. IEEE and NIST recommended the usage of both binary and prime fields for the computation of ECC [3,4]. However, in hardware implementations and more specifically for area-constrained applications, binary fields outperform prime fields, as shown in [5]. Therefore, a lot of research in the literature has been focused on investigating the efficiency of computing point multiplication on elliptic curves over binary fields. For instance, one can refer to [6], [7], [8] and [9] to name a few, covering a wide variety of cases including different curve forms, e.g., generic and Edwards, and different coordinate systems, e.g., affine, projective, and mixed. The formulas for point addition and point addition can be determined by using geometric properties. In [10], binary Edwards curves are presented for the first time for ECC and their low-resource implementations appeared in [9]. It has been shown that a binary Edwards curves (BEC) is isomorphic to a general elliptic curve if the singularities are resolved [10]. Based on the implementations provided in [9], it has been observed that their implementations are not as efficient as other standardized curves. Recently, in [1], the authors revisited the original equations for point addition and doubling and provided competitive formulas. We observed that the revisited formulas for mixed point addition in [1] are invalid. After modifying their formulas, we employed them for the computation of point multiplication using a mixed coordinate system and proposed an efficient crypto-processor for low-resource devices. The main contributions of this paper can be summarized as follows:

- We propose an efficient hardware architecture for point multiplication on binary Edwards curves. We employed Gaussian normal basis (GNB) for representing field elements and curves as the computation of squaring, inversion, and trace function can be done very efficiently over GNB in hardware.
- We modified and corrected the w-coordinates differential point addition formulas presented in [1]. We provide explicit formulas over binary Edwards curves that maintain the speed and register usage provided in [1] and employed the formulas in steps on the Montgomery Ladder [11]. This is the first time this double-and-add algorithm has appeared in literature. This implementation was competitive with many of the area-efficient elliptic curve crypto-processors found in literature, but adds the additional security benefit of completeness.
- We implemented and synthesized our proposed algorithms and architectures for the computation of point multiplication on binary Edwards curves and compared our results to the leading ones available in the literature.

This paper is organized as follows. In Section 2, the binary Edwards curve is introduced and proper mixed coordinate addition formulas are presented. Section 3 details the area-efficient architecture used for this ECC co-processor. Section 4 compares this work to other ECC crypto-processors in terms of area, latency, computation time, and innate security. Section 5 concludes the paper with takeaways and the future of area efficient implementations of point multiplication.

2 Point Multiplication on Binary Edwards Curves

ECC cryptosystems can be implemented over a variety of curves. Some curves have more inherent properties than others. Table 1 contains a comparison of point addition and doubling formulas presented in literature. Completeness means that there are no exceptional cases to addition or doubling (e.g., adding the neutral point). From this table, the choice was to apply the new mixed coordinate addition and doubling formulas over new binary Edwards curves presented in [1].

Table 1. Cost of point operations on binary generic curves (BGCs) [12], binary Edwards curves (BECs) [10], binary Edwards curves revisited [1], and generalized Hessian curves (GHC) [13] over $GF(2^m)$.

Curve	Coordinate System	Differential PA and PD	Completeness
BGC	Projective	$6M + 1D + 5S$	×
	Mixed	$5M + 1D + 4S$	×
BEC ($d_1 = d_2$)	Projective	$7M + 2D + 4S$	✓
	Mixed	$5M + 2D + 4S$	✓
BEC-R ($d_1 = d_2$)	Projective	$7M + 2D + 4S$	✓
	Mixed	$5M + 1D + 4S$	✓
GHC	Projective	$7M + 2D + 4S$	✓
	Mixed	$5M + 2D + 4S$	✓

2.1 Binary Edwards Curve

Definition 1. *Consider a finite field of characteristic two, K. Let $d_1, d_2 \in K$ such that $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. Then the binary Edwards curve with coefficients d_1 and d_2 is the affine curve [10]:*

$$E_{\mathbb{F}_{2^m},d_1,d_2} : d_1(x + y) + d_2(x^2 + y^2) = xy + xy(x + y) + x^2y^2 \tag{1}$$

This curve is symmetric in that if (x, y) is on the curve, then (y, x) is also on the curve. In fact, these points are additive inverses over the Edwards addition law. The point $(0, 0)$ is isomorphic to the point at infinity in a binary generic curve. This represents the neutral point in the binary Edwards curve. The point (1,1) is also on every binary Edwards curve, and has order 2. The curve is complete if there is no element $t \in K$ that satisfies the relation $t^2 + t + d_2 = 0$ [10]. Alternatively, this means that if $\mathrm{Tr}(d_2) = 1$, then the curve is complete [14].

Point addition and point doubling do not have the same representation or equations as standard generic curves. The Edwards Addition Law is presented below. The sum of any two points $(x_1, y_1), (x_2, y_2)$ on the curve defined by $E_{\mathbb{F}_{2^m},d_1,d_2}$ to (x_3,y_3) is defined as [10]

$$\begin{aligned} x_3 &= \frac{d_1(x_1+x_2)+d_2(x_1+y_1)(x_2+y_2)+(x_1+x_1^2)(x_2(y_1+y_2+1)+y_1y_2)}{d_1+(x_1+x_1^2)(x_2+y_2)} \\ y_3 &= \frac{d_1(y_1+y_2)+d_2(x_1+y_1)(x_2+y_2)+(y_1+y_1^2)(y_2(x_1+x_2+1)+x_1x_2)}{d_1+(y_1+y_1^2)(x_2+y_2)} \end{aligned} \tag{2}$$

We note that [10] uses this addition law to prove that ordinary elliptic curves over binary fields are birationally equivalent to binary Edwards curves.

2.2 Revised Differential Addition and Doubling Formulas

Point multiplication utilizes point doubling and point addition to quickly generate large multiples of a point. As a deterrent to timing and other side channel attacks, the Montgomery Ladder [11] is used as a method to generate multiplications efficiently and securely. Montgomery Ladder is shown in Algorithm 1. At each step of the ladder, there is an addition and doubling. The current bit of a key determines which point is doubled and where the point addition and doubling reside. For standard point additions and point doublings, the finite

Algorithm 1 Montgomery algorithm [11] for point multiplication using w-coordinates.

Inputs: A point $P = (x_0, y_0) \in E(\mathbb{F}_{2^m})$ on a binary curve and an integer $k = (k_{l-1}, \cdots, k_1, k_0)_2$.
Output: $w(Q) = w(kP) \in E(\mathbb{F}_{2^m})$.
1: **set** : $w_0 \leftarrow x_0 + y_0$ and initialize
  a: $W_1 \leftarrow w_0$ and $Z_1 \leftarrow 1$ and $c = \frac{1}{w_0}$ (inversion)
  b: $(W_2, Z_2) =$ `DiffDBL`(W_1, Z_1)
2: **for** i **from** $l-2$ down to 0 **do**
  a: **if** $k_i = 1$ **then**
    i): $(W_1, Z_1) =$ `MDiffADD`(W_1, Z_1, W_2, Z_2, c)
    ii): $(W_2, Z_2) =$ `DiffDBL`(W_2, Z_2)
  b: **else**
    i): $(W_1, Z_1) =$ `DiffDBL`(W_1, Z_1)
    ii): $(W_2, Z_2) =$ `MDiffADD`(W_1, Z_1, W_2, Z_2, c)
  end if
  end for
3: **return** $w(kP) \leftarrow (W_1, Z_1)$ and $w((k+1)P) \leftarrow (W_2, Z_2)$

field inversion dominates the computation. To reduce this impact, the typical convention is to use projective coordinates, $(x, y) \rightarrow (X, Y, Z)$, where $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$. X, Y, and Z are updated at each step of the ladder and a single inversion is performed at the end. An additional improvement to this convention is to use w-coordinates, $(x, y) \rightarrow (w)$, where $w = x + y$. Mixed coordinates is the combination of w-coordinates and the projective coordinates. Hence, $(x, y) \rightarrow w \rightarrow (W, Z)$, where $w = x + y = \frac{W}{Z}$. For computing point multiplication, let P be a point on a binary Edwards curve $E_{\mathbb{F}_{2^m}, d_1, d_2}$ and let us assume $w(nP)$ and $w((n+1)P)$, $0 < n < k$ are known. Therefore, one can use the w-coordinate differential addition and doubling formulas to compute their sum as $w((2n+1)P)$ and double of $w(nP)$ as $w(2nP)$ [10].

2.3 Fixed w-Coordinate Differential Addition

In [1], the authors present faster equations for w-coordinates and mixed coordinates addition than those presented in [10]. This equation makes the assumption that $d_1 = d_2$. An analysis of the formula, however, shows that they do not properly produce the correct w-coordinates. The authors correctly identify the relation, $\frac{w_3 w_0}{d_1(w_1^2 + w_2^2)} = \frac{w_3 + w_0 + 1}{d_1}$, but incorrectly solve for w_3. We observe that the final equation for differential point addition that is presented in subsection (3.19) of [1] is faulty. Therefore, we wrote a sage script to verify this claim[1]. This algebra was performed correctly and here we present the revised formulas. The incorrect formula presented in [1] is in (3) and the corrected formula is shown in (4). This formula defines the addition of $w_1 + w_2 = w_3$, given that $w_i = x_i + y_i$ and $w_0 = w_2 - w_1$.

Proposition 1. *The w-coordinate differential addition formula over binary Edwards curves with $d_1 = d_2$ proposed in [1] does not provide correct formulation based on the following equation:*

$$w_3 = 1 + \frac{\frac{1}{w_0}(w_1^2 + w_2^2)}{\frac{1}{w_0}(w_1^2 + w_2^2) + 1} \tag{3}$$

[1] http://github.com/briankoziel/BEC_Small

Proof. In the following equations, the correct w-coordinate differential addition formula over binary Edwards curves with $d_1 = d_2$ is discovered from the starting relation in [1].

$$\frac{w_3 w_0}{d_1(w_1^2 + w_2^2)} = \frac{w_3 + w_0 + 1}{d_1}$$

$$\frac{w_3 w_0}{(w_1^2 + w_2^2)} = w_3 + w_0 + 1$$

$$\frac{w_3 w_0}{(w_1^2 + w_2^2)} + w_3 = w_0 + 1$$

$$w_3(\frac{w_0}{(w_1^2 + w_2^2)} + 1) = w_0 + 1$$

$$w_3(w_0 + w_1^2 + w_2^2) = (w_0 + 1)(w_1^2 + w_2^2)$$

$$w_3 = \frac{(w_0 + 1)(w_1^2 + w_2^2)}{w_0 + w_1^2 + w_2^2}$$

$$w_3 = \frac{(1 + \frac{1}{w_0})(w_1^2 + w_2^2)}{\frac{1}{w_0}(w_1^2 + w_2^2) + 1}$$

Corrected w-Coordinate Differential Addition

$$w_3 = \frac{w_1^2 + w_2^2 + \frac{1}{w_0}(w_1^2 + w_2^2)}{\frac{1}{w_0}(w_1^2 + w_2^2) + 1}. \tag{4}$$

The explicit affine w-coordinate differential addition is

$$\begin{aligned} A = (w_1 + w_2)^2, B = A \cdot \tfrac{1}{w_0}, N = A + B, \\ D = B + 1, \quad E = \tfrac{1}{D}, \quad w_3 = N \cdot E. \end{aligned} \tag{5}$$

The total cost of this corrected formula is still $1I + 1M + 1D + 1S$, but now the differential addition functions as intended. Assuming that inversion requires at least two registers, a total of three registers are required. $\frac{1}{w_0}$ is the inverse of the difference between the points and will not be updated in each step of the point multiplication algorithm. For the application in Montgomery Ladder [11], the difference between the two points is always P (specifically $w(P)$). Therefore, this value can be determined at the start of the ladder and used throughout to cut down on each step.

[1] uses the faulty formula (3) for determining explicit formulas in mixed w-coordinates, but also gives a faster and correct formula for affine w-coordinate differential addition which requires $1I + 1M + 2S$, so long as the values $\frac{1}{w_0 + w_0^2}$ and w_0 are known. This formula is shown below.

$$w_3 = w_0 + 1 + \frac{1}{\frac{1}{w_0 + w_0^2}(w_1^2 + w_2^2 + w_0)} \tag{6}$$

The explicit affine w-coordinate differential addition is

$$A = (w_1 + w_2)^2, \quad B = A + w_0, \quad D = B \cdot \frac{1}{w_0 + w_0^2}, \tag{7}$$
$$E = \frac{1}{D}, w_3 = E + 1 + w_0$$

Assuming that w_0 and $\frac{1}{w_0+w_0^2}$ are known, the actual cost for w-coordinate differential addition can be reduced down to $1I + 1D + 1S$. This method requires two registers for w_1 and w_2, and the storage of w_0 and $\frac{1}{w_0+w_0^2}$.

Mixed w-Coordinate Differential Addition and Doubling Equation (4) can be applied to mixed w-coordinate differential addition and doubling. The general formula and explicit formula are shown below. This formula defines the addition of $\frac{W_1}{Z_1} + \frac{W_2}{Z_2} = \frac{W_3}{Z_3}$, given that $w_0 = w_2 - w_1$.

$$\frac{W_3}{Z_3} = \frac{(W_1Z_2 + W_2Z_1)^2 + \frac{1}{w_0}(W_1Z_2 + W_2Z_1)^2}{Z_1^2Z_2^2 + \frac{1}{w_0}(W_1Z_2 + W_2Z_1)^2} \tag{8}$$

$$C = (W_1Z_2 + W_2Z_1)^2, D = (Z_1Z_2)^2, E = \frac{1}{w_0} \cdot C, \tag{9}$$
$$W_3 = E + C, \ Z_3 = E + D$$

Thus, mixed w-coordinate differential addition requires $3M + 1D + 2S$. From a simple analysis of the formula, four registers are needed.

For mixed w-coordinate differential addition and doubling, the doubling formula from [10] can be used in conjunction with this corrected differential addition formula, with the assumption that $d_1 = d_2$. This formula defines the addition of $\frac{W_1}{Z_1} + \frac{W_2}{Z_2} = \frac{W_3}{Z_3}$ and doubling of $2 \times \frac{W_1}{Z_1} = \frac{W_4}{Z_4}$ given that $w_0 = w_2 - w_1$.

$$\frac{W_4}{Z_4} = \frac{(W_1(W_1 + Z_1))^2}{d_1 \centerdot Z_1^4 + (W_1(W_1 + Z_1))^2} \tag{10}$$

$$C = (W_1Z_2 + W_2Z_1)^2, \quad D = (Z_1Z_2)^2, \quad E = \frac{1}{w_0} \cdot C, \tag{11}$$
$$W_3 = E + C, \quad Z_3 = E + D \quad W_4 = (W_1(W_1 + Z_1))^2,$$
$$Z_4 = W_4 + d_1 \cdot Z_1^4$$

Thus, mixed w-coordinate differential addition and doubling requires $5M + 1D + 5S$. From an analysis of the formula, five registers are needed.

Mixed w-Coordinate Differential Addition and Doubling with the Co-Z Trick We note that in [15] the common-Z trick is proposed. This method to reduces the number of registers required per step of the Montgomery Ladder [11] and simplifies the number of operations per step. Each step of the Montgomery Ladder is a point doubling and addition. By using a common-Z coordinate system, one less register is required for a step on the ladder, and the method becomes more efficient, requiring one less squaring operation. The doubling formula

was obtained from [10] and it is assumed that $d_1 = d_2$. The general formula and explicit formulas are shown below. This formula defines the addition of $\frac{W_1}{Z} + \frac{W_2}{Z} = \frac{W_3}{Z'}$ and doubling of $2 \times \frac{W_1}{Z} = \frac{W_4}{Z'}$ given that $w_0 = w_2 - w_1$.

$$\frac{W_3}{Z'} = \frac{(W_1 + W_2)^2 + \frac{1}{w_0}(W_1 + W_2)^2}{Z^2 + \frac{1}{w_0}(W_1 + W_2)^2} \tag{12}$$

$$\frac{W_4}{Z'} = \frac{(W_1(W_1 + Z))^2}{d_1 \centerdot Z^4 + (W_1(W_1 + Z))^2} \tag{13}$$

$$\begin{gathered} C = (W_1 + W_2)^2, \quad D = Z^2, \quad E = \frac{1}{w_0} \cdot C, \\ U = E + C, V = E + D, S = (W_1(W_1 + Z))^2, \\ T = S + d_1 \cdot D^2, W_3 = U \cdot T, W_4 = V \cdot S, \\ Z' = V \cdot T \end{gathered} \tag{14}$$

Thus, the mixed w-coordinate differential addition and doubling formula requires $5M + 1D + 4S$. An analysis of this formula shows that it requires only four registers. As will be discussed later, this implementation incorporates shifting for the multiplication within the register file, forcing the need for an additional register. This formula requires one less squaring than that provided in [10], and also uses registers much more efficiently. Table 2 shows a comparison of differential point addition schemes for BEC with $d_1 = d_2$.

Table 2. Comparison of Differential Point Addition Schemes for BEC with $d_1 = d_2$.

Operation	Formula	Complexity	#Registers
Affine w-coordinate Differential Addition	(5)	$1I + 1M + 1D + 2S$	3
Affine w-coordinate Differential Addition	(7)	$1I + 1D + 1S$	2
Mixed Differential Addition	(9)	$3M + 1D + 1S$	4
Mixed Differential Addition and Doubling	(11)	$5M + 1D + 5S$	5
Mixed Differential Addition and Doubling w/ Co-Z	(14)	$5M + 1D + 4S$	4

Retrieving x and y from w-Coordinates The formula to retrieve the x-coordinate from w-coordinates is presented in [10]. This formula requires P, $w(kP)$, and $w(kP+1)$. Again, relating back to the application of Montgomery Ladder [11], each consecutive step produces $w(mP)$ and $w(mP + 1)$, where m represents the scalar multiplication over each steps. The formula to solve for the x-coordinate of mP is shown below [10]. In this formula, $P = (x_1, y_1)$,$w_0 = x_1 + y_1$, $w_2 = w(kP)$, and $w_3 = w(kP + 1)$.

$$x_2^2 + x_2 = \frac{w_3(d_1 + w_0 w_2(1 + w_0 + w_2) + \frac{d_2}{d_1} w_0^2 w_2^2) + d_1(w_0 + w_2) + (y_1^2 + y_1)(w_0^2 + w_2)}{w_0^2 + w_0} \tag{15}$$

This formula requires $1I + 4M + 4S$ if $d_2 = d_1$. After solving for $x_2^2 + x_2 = A$, if $\mathrm{Tr}(A) = 0$, then the value of x_2 or $x_2 + 1$ can be recovered by using the half-trace.

After the value of x_2 has been found, y_2 can be retrieved by solving the curve equation for $y_2^2 + y_2$ (16) and also using the half-trace to solve for y_2 or $y_2 + 1$.

Algorithm 2 Retrieving x and y from w-coordinates

Inputs: A point $P = (x_0, y_0) \in E(\mathbb{F}_{2^m})$ on a
binary curve and an integer $k = (k_{l-1}, \cdots, k_1, k_0)_2$.
Output: $Q = kP \in E(\mathbb{F}_{2^m})$.
1: set: $w_0 \leftarrow x_0 + y_0$ and initialize
2: compute: $w_2 \leftarrow w(kP)$, $w_3 \leftarrow w(kP + 1)$
3: solve (15) for $x_2 + x_2^2$
4: **if** $\mathrm{Tr}(x_2 + x_2^2) = 0$ **then**
a: $x_2 =$**half-trace**$(x_2 + x_2^2)$
end if
3: solve (16) for $y_2 + y_2^2$
4: **if** $\mathrm{Tr}(y_2 + y_2^2) = 0$ **then**
a: $y_2 =$**half-trace**$(y_2 + y_2^2)$
end if
5: **return** $Q = (x_2, y_2) = kP \in E(\mathbb{F}_{2^m})$

$$y_2^2 + y_2 = \frac{d(x_2 + x_2^2)}{d + x_2 + x_2^2} \tag{16}$$

Therefore, recovering y_2 requires $1I + 2M + S$, and the total cost of recovering points from w-coordinates is $2I + 6M + 5S$. Even though the point $(x_2 + 1,\ y_2 + 1)$ is not the same as $(x_2,\ y_2)$, both points will produce the same value in standard ECC applications. Algorithm 2 summarizes how to retrieve the x and y-coordinates.

The implementation of the algorithms noted in this section require a binary Edwards curve with $d_1 = d_2$. The standardized NIST curves over binary generic curves [3] could be converted to binary Edwards curves. However, there is no guarantee that these isomorphic binary Edwards curves would satisfy $d_1 = d_2$. Therefore, values for x and d were randomly picked and used in conjunction with (16) to solve for y. If the point (x,y) was on the curve, then the point and corresponding binary Edwards curve were valid and could be used with the above algorithms. It can also be noted that there are no restrictions on d, so it could be chosen to be small for faster arithmetic.

2.4 Resistance Against Side-Channel Attacks

The binary Edwards curve features the unique properties that its addition formula is unified and complete. Unified implies that the addition and doubling formulas are the same. This gives the advantage that no checking is required for the points to differentiate if an addition or doubling needs to take place. Complete implies that the addition formula works for any two input points, including the neutral point. Therefore, as long as two points are on the curve, no checking is needed for the addition formula, as it will always produce a point for a complete binary Edwards Curve [10].

One common attack to reveal bits of an ECC system's key is to use the exceptional points attack [16]. This attacks the common projective coordinate system. For the point at infinity in a non-binary Edwards curve system, the point is often represented as $(X_k, Y_k, 0)$. Hence, a conversion back to the (x_k, y_k) coordinate system would attempt to divide by zero, causing an error or revealing a point that is not on the curve [16]. In either case, an adversary could detect that the point at infinity was attempted to be retrieved. The attack relies on picking

different base points, which after multiplied by the hidden key, reveal that the point of infinity was retrieved.

The binary Edwards curve's completeness property and coordinate system make the curve immune to this form of attack. For a complete binary Edwards curve, the projective coordinate system representation for the neutral point, which is isomorphic to the point at infinity of other curves, is $(X_k, Y_k, 1)$. Furthermore, the completeness also ensures that no other sets of points can be used to break the system and reveal critical information about the key. The mixed w-coordinates that are used for their speed in the binary Edwards curve are also invulnerable to this attack as long as $w_0 \neq 0,1$, since the denominator will never be 0 [1]. With the Montgomery Ladder [11], a proper curve and starting point will never violate this condition.

Montgomery Ladder [11] is a secure way to perform repeated point addition and point doublings to thwart side channel attacks. The ladder provides a point addition and point doubling for each step, with each step taking the same amount of time. Therefore, this application provides an extremely powerful defense against power analysis attacks and timing attacks. Power analysis attacks identify characteristics of the power consumption of a device to reveal bits of the key and timing attacks identify characteristics of the timing as the point multiplication is performed. By application of the binary Edwards curve with Montgomery Ladder, the binary Edwards curve features an innate defense against many of the most common attacks on ECC systems today.

It should be noted that this work does not investigate resistance against differential power analysis (DPA) [17] or electromagnetic (EM) radiation leaks. These will be investigated in detail in a future work.

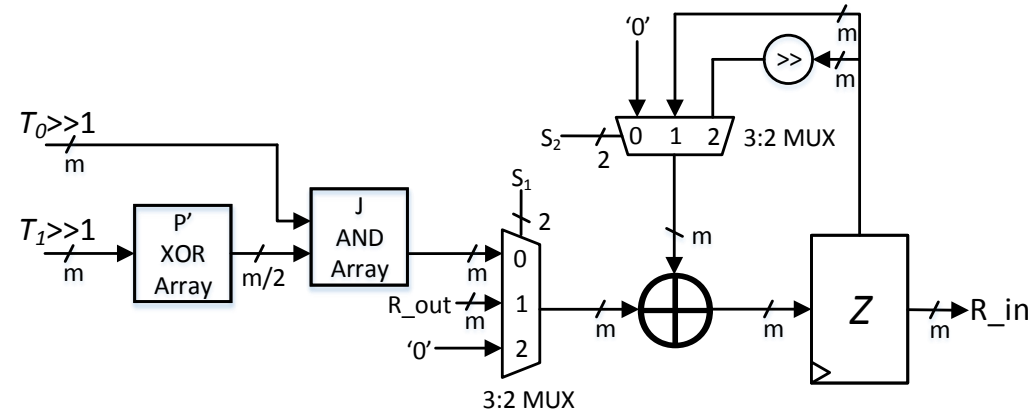
3 Architecture

The architecture of the ECC co-processor that was implemented resembles that of [6]. However, there are several major differences. An analysis of the explicit formula presented for mixed w-coordinate addition and doubling revealed that five registers $(T_0, T_1, R_0, R_1, R_2)$ and four constants $(\frac{1}{w_0}, d_1, x_1, y_1)$ were required. Additionally, it was deemed that the neutral element in GNB multiplication (all '1's) was not required for any part of the multiplication, which reduced the size of the 4:2 output multiplexer to a 3:2 multiplexer. These following sections will explain the design in more detail. Architectures for each of the components can be found in Fig. 1.

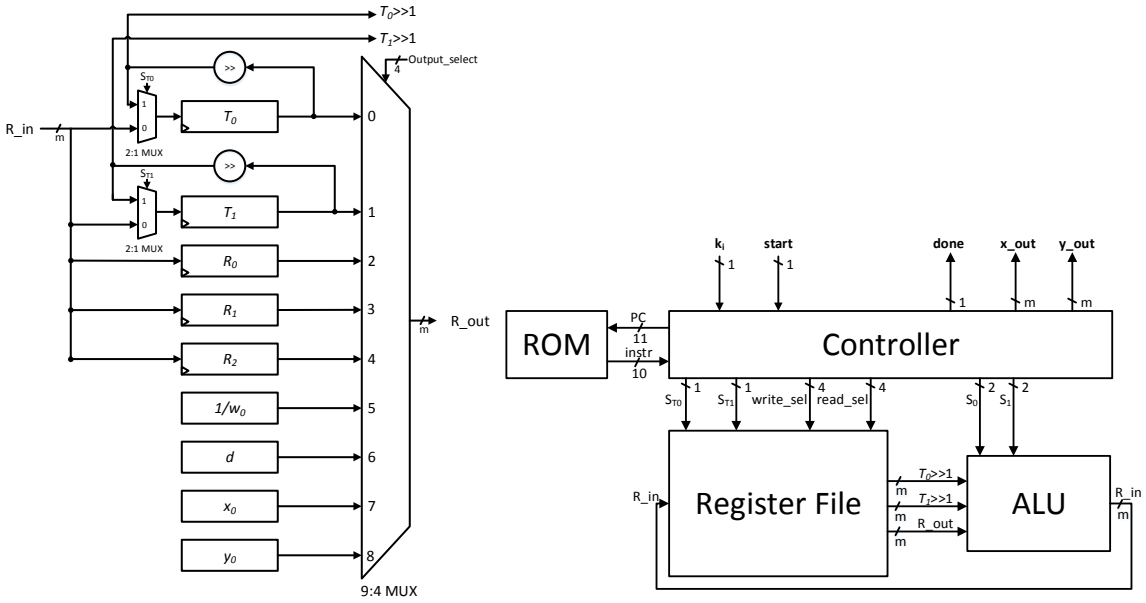
3.1 Field Arithmetic Unit

The field arithmetic unit is designed to incorporate the critical finite field operations in as small of a place as possible. In particular, this requires multiplication, squaring, and addition. The XOR gate to add two elements was reused in the multiplication and addition to reduce the total size of the FAU. Since the neutral element was not necessary for this point multiplier, the neutral element select from the output multiplexer in [6] was removed to save area. Swap functionality was added to incorporate quick register file swap operations. The field arithmetic unit incorporates the GNB multiplier from [18]. The operations are as follows:

- **Addition** $C = A + B$: Addition is a simple XOR of two inputs. The first input is loaded to Z by selecting the first input in the register file, and setting $s_1 =$ "01" and $s_2 =$ "00". The next cycle, the second operand is selected from the register file, and $s_2 =$ "01" so that the output register has the addition of the two input elements. The output is written on the third cycle. This operation requires three clock cycles.



(a) The FAU.



(b) The register file and top-level control unit.

Fig. 1. Architecture of the Proposed Co-processor for Point Multiplication on Binary Edwards Curve. This includes (a) Field Arithmetic Unit, (b) Register File, and Top-Level Control Unit.

- **Squaring** $C = A \gg 1$: Squaring is a right circular shift of the input. The input is loaded to Z by selecting the input in the register file, and setting $s_1 =$ "01" and $s_2 =$ "00". The next cycle, $s_1 =$ "10" and $s_2 =$ "10" so that the output register has been shifted. The output is written on the third cycle. This operation requires three clock cycles.
- **Multiplication** $C = T_0 \times T_1$: Multiplication is a series of shifted additions. For the first cycle, $s_1 =$ "00",$s_2 =$ "00", and $s_{T_0} = s_{T_1} =$ "1". The next cycle, $s_2 =$ "01". After m cycles of shifts and addition, $s_{T_0} = s_{T_1} =$ "0",and the output is ready. The output is written on the mth cycle. This operation requires m clock cycles.
- **Swapping** $A, B = B, A$: Swapping is a switch of two registers within the register file. The first register is loaded to Z by selecting the input in the register file, and setting $s_1 =$ "01" and $s_2 =$ "00". The next cycle, the first register is written to the second register's location as it is being loaded to Z. The second register's value is written to the first register's place on the third cycle. This operation requires three clock cycles.

3.2 Register File

Similar to [6] and [9], the register file was designed to contain registers, with two particular registers that perform special shifting for the finite field multiplication. An analysis of the formulas used in this ECC unit revealed that four registers and four constants were required. However, with two registers being designated as multiplication registers, an extra register is needed for swapping in the value of d_1 for a multiplication with D^2. The other three registers would be holding (U, V, S). Thus, the formulas require five registers with the Co-Z trick implementation.

For unified access to constants and not impact the retrieval, the registers and constants are co-located in the register file. However, since this implementation targets a future standardization of a binary Edwards curve, the idea was that a starting point and curve parameters would be strictly defined. Therefore, there is no reason to add flexibility to the parameters of the base point or d_1. Hardwiring these coordinates to the register file provides the advantage that they can be used on-the-fly and that no extra control is necessary to bring these into the register file. For instance, [9] uses a small and external RAM chip to hold these constants. Such a design requires extra interfacing and extra cycles to load the value into the register file. After NIST standards for ECC are revised, hardwiring the constants in a place close to the register file is the best solution to save power and area.

The register file is random access to values including the constants. A register is written to when write is enabled and the multiplexer for writing selects that register.

3.3 Control Unit

The control unit handles the multiplexers for reading, writing, and performing operations. The four operations are ADD, SQ, MULT, and SWAP. The control unit uses a Finite State Machine to switch between these operations. A program counter is sent to an external ROM device that feeds in the current instruction. Instructions are ten bits long. The first two bits indicate which instruction is being used. The next four bits indicate the input register. This value does not matter for multiplication. The last four bits indicate the output register.

The key is never stored in the control unit, such as how it was in [6]. The controller signals the master device to provide the next bit as the Montgomery Ladder [11] is being performed. Special SWAP instructions that depend on the key were left inside the controller to handle each step of the ladder, depending on the provided bit of the key. The subroutine for a step

on the Montgomery Ladder with the corresponding register usage is shown below in Table 3. Table 3 shows the registers after each instruction. Six multiplications are required for each step.

Table 3. Point Addition and Doubling Register Usage

#	Op	T_0	T_1	R_0	R_1	R_2
1	ADD T0 T1	W_1	$W_1 + W_2$	Z		
2	SQ T1 R1	W_1	$W_1 + W_2$	Z	C	
3	SWAP T1 R0	W_1	Z	$W_1 + W_2$	C	
4	SQ T1 R0	W_1	Z	D	C	
5	ADD T0 T1	W_1	$W_1 + Z$	D	C	
6	MULT T1 T0	$W_1 \cdot (W_1 + Z)$	$W_1 + Z$	D	C	
7	SQ T0 R2	$W_1 \cdot (W_1 + Z)$	$W_1 + Z$	D	C	S
8	SWAP R1 T1	$W_1 \cdot (W_1 + Z)$	C	D	$W_1 + Z$	S
9	SWAP R3 T0	$\frac{1}{w_0}$	C	D	$W_1 + Z$	S
10	MULT T0 R1	$\frac{1}{w_0}$	C	D	E	S
11	ADD R1 T1	$\frac{1}{w_0}$	U	D	E	S
12	ADD R0 R1	$\frac{1}{w_0}$	U	D	E	S
13	SQ R0 R0	$\frac{1}{w_0}$	U	D^2	V	S
14	SWAP R0 T1	$\frac{1}{w_0}$	D^2	U	V	S
15	SWAP R4 T0	d	D^2	U	V	S
16	MULT T1 T0	$d \cdot D^2$	D^2	U	V	S
17	ADD R2 T0	T	D^2	U	V	S
18	SWAP R0 T1	T	U	D^2	V	S
19	MULT T0 T1	T	W_3	D^2	V	S
20	SWAP T1 R1	T	V	D^2	W_3	S
21	MULT T0 T0	Z'	V	D^2	W_3	S
22	SWAP T0 R2	S	V	D^2	W_3	Z'
23	MULT T0 T1	S	W_4	D^2	W_3	Z'
24	SWAP R0 R2	S	W_4	Z'	W_3	D^2
25	SWAP T0 R1	W_3	W_4	Z'	S	D^2

To save area, the half-trace functionality was left as a series of squarings and additions. Adding additional area to handle the half-trace saves a relatively small fraction of instructions but adds an additional multiplexer select in the FAU.

Inversion and the half-trace were implemented as subroutines within the ROM for instructions. The half traces uses a repetitive combination of double SQ then ADD. This was used to recover the x and y-coordinates of the final point. Inversion was used to obtain $w_i = \frac{W_i}{Z_i}$, recover the x-coordinate, and recover the y-coordinate. Itoh-Tsujii inversion algorithm [19] was used to reduce the number of multiplications. For $\mathbb{F}_{2^{283}}$, the addition chain (1,2,4,8,16,17,34,35,70,140,141,282) was used. By implementing these repeated functionalities as subroutines, the number of instructions in the ROM is dramatically reduced. The main program is shown in Figure 2. The subroutines for inversion in $\mathbb{F}_{2^{283}}$ and the half-trace are shown in Figure 3. The total instruction count of the point multiplier for $\mathbb{F}_{2^{283}}$ is shown in Table 4. Approximately 132, 10-bit instructions were needed.

Initialization
1: $T_0 \leftarrow x_1$
2: $T_0 \leftarrow T_0 + y_1$
3: $T_1 \leftarrow T_0^2$
4: $T_1 \leftarrow T_1 + T_0$
5: $R_0 \leftarrow T_1^2$
6: $T_1 \leftarrow d$
7: $T_1 \leftarrow T_1 + R_0$
8: $T_0 \leftarrow T_0 \times T_1$
9: $T_1, R_0 \leftarrow R_0, T_1$
Reg: $W_1, W_2, Z, 0, 0$
Mont. Ladder
Reg: $W_3, W_4, Z', 0, 0$
Recover w_2 **and** w_3
1: $R_1, T_0 \leftarrow T_0, R_1$
2: $T_1, R_2 \leftarrow R_2, T_1$
3: $T_0, R_0 \leftarrow R_0, T_0$
4: $T_0 \leftarrow T_0^{-1}$
5: $T_1, R_2 \leftarrow R_2, T_1$
6: $R_2 \leftarrow T_0 \times T_1$
7: $T_1, R_1 \leftarrow R_1, T_1$
8: $T_1 \leftarrow T_0 \times T_1$
9: $T_0 \leftarrow x_0$
10: $T_0 \leftarrow T_0 + y_0$
Reg: $w_0, w_2, 0, 0, w_3$
Recover x_2
1: $R_0 \leftarrow T_0 \times T_1$
2: $T_0 \leftarrow T_0 + R_0$
3: $T_0 \leftarrow T_0 + T_1$
4: $T_1, R_0 \leftarrow R_0, T_1$
5: $T_0 \leftarrow T_0 \times T_1$
6: $T_0 \leftarrow T_0 + d$
7: $T_1, R_2 \leftarrow R_2, T_1$
8: $R_2 \leftarrow T_0 \times T_1$
9: $T_0 \leftarrow x_0$
10: $T_0 \leftarrow T_0 + y_0$
11:$T_0 \leftarrow T_0 + R_0$
12:$T_0 \leftarrow T_0 + R_0$
13:$T_1 \leftarrow d$
14:$T_0 \leftarrow T_0 \times T_1$
15:$R_2 \leftarrow R_2 + T_0$
16:$T_0 \leftarrow y_1^2$
17:$T_0 \leftarrow T_0 + y_1$
18:$T_1 \leftarrow R_0^2$
19:$T_1 \leftarrow T_1 + R_0$
20:$T_0 \leftarrow T_0 \times T_1$
21:$R_2 \leftarrow R_2 + T_0$
22:$T_0 \leftarrow x_0$
23:$T_0 \leftarrow T_0 + y_0$
24:$T_1 \leftarrow T_0^2$
25:$T_0 \leftarrow T_0 + T_1$
26:$T_0 \leftarrow T_0^{-1}$
27:$T_1, R_2 \leftarrow R_2, T_1$
28:$T_0 \leftarrow T_0 \times T_1$
29:$T_0 \leftarrow halfTr(T_0)$
Reg: $x_2, 0, 0, 0, 0$
Recover y_2
1: $R_2 \leftarrow T_0^2$
2: $R_2 \leftarrow R_2 + T_0$
3: $R_1, T_0 \leftarrow T_0, R_1$
5: $T_0 \leftarrow T_0 + R_2$
6: $T_0 \leftarrow T_0^{-1}$
7: $T_1, R_2 \leftarrow R_2, T_1$
8: $T_0 \leftarrow T_0 \times T_1$
9: $T_1 \leftarrow d$
10:$T_0 \leftarrow T_0 \times T_1$
11:$T_0 \leftarrow halfTr(T_0)$
12:$T_2 \leftarrow Z$
13:$T_0, T_1 \leftarrow T_1, T_0$
14:$T_0, R_1 \leftarrow R_1, T_0$
Reg: $x_2, y_2, 0, 0, 0$

Fig. 2. Main Program Listing for Point Multiplication using Binary Edwards Curves [10]

Inversion
1: $T_1 \leftarrow T_0^2$
2: $T_1 \leftarrow T_0 \times T_1$
3: $T_0, R_0 \leftarrow R_0, T_0$
4: $T_0 \leftarrow T_1^2$
5: $T_0 \leftarrow T_0^2$
6: $T_0 \leftarrow T_0 \times T_1$
7: $T_0 \leftarrow T_1^2$
8: $T_0 \leftarrow T_0^{2^3}$
9: $T_0 \leftarrow T_0 \times T_1$
10:$T_0 \leftarrow T_1^2$
11:$T_0 \leftarrow T_0^{2^7}$
12: $T_0 \leftarrow T_0 \times T_1$
13: $T_1, R_0 \leftarrow R_0, T_1$
14: $T_0 \leftarrow T_0^2$
15: $T_0 \leftarrow T_0 \times T_1$
16: $T_1, R_0 \leftarrow R_0, T_1$
17: $T_0 \leftarrow T_1^2$
18: $T_0 \leftarrow T_0^{2^{16}}$
19: $T_0 \leftarrow T_0 \times T_1$
20: $T_1, R_0 \leftarrow R_0, T_1$
21: $T_0 \leftarrow T_0^2$
22: $T_0 \leftarrow T_0 \times T_1$
23: $T_1, R_0 \leftarrow R_0, T_1$
24:$T_0 \leftarrow T_1^2$
25:$T_0 \leftarrow T_0^{2^{34}}$
26:$T_0 \leftarrow T_0 \times T_1$
27:$T_0 \leftarrow T_1^2$
28:$T_0 \leftarrow T_0^{2^{69}}$
29:$T_0 \leftarrow T_0 \times T_1$
30:$T_1, R_0 \leftarrow R_0, T_1$
31:$T_0 \leftarrow T_0^2$
32:$T_0 \leftarrow T_0 \times T_1$
33:$T_0 \leftarrow T_1^2$
34:$T_0 \leftarrow T_0^{2^{140}}$
35:$T_0 \leftarrow T_0 \times T_1$
36:$T_0 \leftarrow T_0^2$
Half-Trace
1: $T_1 \leftarrow T_0^2$
2: $T_1 \leftarrow T_1^2$
3: $T_0 \leftarrow T_0 + T_1$
4: $T_1 \leftarrow T_1^2$
5: $T_1 \leftarrow T_1^2$
6: $T_0 \leftarrow T_0 + T_1$
*Repeat steps 4-6 $\frac{m-2}{2}$ times

Fig. 3. Itoh-Tsujii [19] Inversion ($\mathbb{F}_{2^{283}}$) and Half-Trace Subroutines

Table 4. Necessary Subroutines.

Subroutine	Iterations	#ADD	#SQ	#MULT	#SWAP	Latency (cycles)
Init	1	3	2	1	4	310
Step	281	5	4	6	10+2[1]	494,841
x Recovery, no HT and Inv	1	16	5	9	13	2,649
y Recovery, no HT and Inv	1	3	2	3	6	882
Half Trace	2×141	1	2	0	0	2×1,269
Inversion	3×1	0	282	11	6	3×3,977
Total		1,705	2,540	1,730	3,410	512,555

1. Special SWAP's that the controller handles.

4 Comparison and Discussion

This design was synthesized using Synopsys Design Compiler in $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{233}}$, and $\mathbb{F}_{2^{163}}$, each a different standardized binary field size by NIST [3]. The TSMC 65-nm CMOS standard technology and CORE65LPSVT standard cell library were used for results. This implementation was optimized for area.

The area was converted to Gate Equivalent (GE), where the size of a single NAND gate is considered 1 GE. For our particular technology library, the size of a synthesized NAND gate was 1.4 μm^2, so this was used as the conversion factor. Latency reports the total number of cycles to compute the final coordinates of a point multiplication. Parameters such as the type of curve used and if Montgomery Ladder were used to indicate some innate security properties of the curve. Power and energy results were not included as a comparison because they are dependent on the underlying technology, frequency of the processor, and testing methodology. The comparison results are shown in Table 5.

Table 5. Comparison of Different Point Bit-Level Multiplications Targeted for ASIC

Work	Curve	Ladder?	Field Size	Tech (nm)	Mult.	# of clock Cycles	Coord.	Area (GE)
[15], 2007	BGC	✓	$\mathbb{F}_{2^{163}}$	180	Bit-serial	313,901	Projective	13,182
[8], 2008	BGC	✓	$\mathbb{F}_{2^{163}}$	130	Bit-serial	275,816	Mixed	12,506
[9], 2010	BEC	✓	$\mathbb{F}_{2^{163}}$	130	Bit-serial	219,148	Mixed	11,720
[20], 2011	BGC	✓	$\mathbb{F}_{2^{163}}$	130	Comb-serial	286,000	Projective	8,958
[6], 2014	BKC	×	$\mathbb{F}_{2^{163}}$	65	Bit-serial	106,700	Affine	10,299
[21], 2014	BGC	×	$\mathbb{F}_{p^{160}}$	130	Comb-serial	139,930	Projective	12,448[1]
[7], 2015	BKC	×	$\mathbb{F}_{2^{283}}$	130	Comb-serial	1,566,000	Projective	10,204[2] (4,323)[3]
This work	BEC	✓	$\mathbb{F}_{2^{163}}$	65	Bit-serial	177,707	Mixed	10,945[4]
		✓	$\mathbb{F}_{2^{233}}$			351,856		14,903[4]
		✓	$\mathbb{F}_{2^{283}}$			512,555		19,058[4]

1. Includes a Keccak module to perform ECDSA
2. RAM results were not synthesized, but extrapolated from a different implementation.
3. Area excluding RAM
4. Area excluding ROM. Approximately 274 GE more with ROM.

This ECC implementation over BEC does make a few assumptions that not necessarily each of these other implementations make. This architecture's area does not include the ROM to hold the instructions. The ROM was not synthesized, but approximately 165 bytes of ROM were required. By the estimate that 1,426 bytes is equivalent to 2,635 GE in [22], 165 bytes of ROM is roughly equivalent to 274 GE. This architecture assumes that each bit of the key will be fed into the co-processor. These assumptions are explained in previous sections. The areas of the implementation for $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, and $\mathbb{F}_{2^{283}}$ excluding the register file and program ROM are 3,248 GE, 3,788 GE, and 5,566 GE, respectively.

Looking at timing for these implementation, the number of clock cycles appears to rise quadratically when comparing $\mathbb{F}_{2^{163}}$ to $\mathbb{F}_{2^{283}}$. This is to be expected, as the Montgomery Ladder performs 6 multiplications each step. A multiplication takes m clock cycles and there are $m-2$ steps.

The area appears to have a linear relationship. This is also to be expected, as the register file's size increases linearly. The area of the FAU depends on the underlying finite field and the area of the controller is fairly constant. The area of the FAU and controller for $\mathbb{F}_{2^{233}}$ is only a slight increase over the area of the FAU and controller for $\mathbb{F}_{2^{163}}$ because the $\mathbb{F}_{2^{233}}$ is type II

GNB, in contrast to $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{283}}$ are type IV GNB. Therefore the p' block in $\mathbb{F}_{2^{233}}$ requires much fewer XOR gates.

The underlying architecture of this implementation was similar to [6]. This implementation uses more area because an additional register and two additional constants were used in the register file. However, one less multiplexer was required in the FAU since the neutral element in GNB was not required in any formulas. Other than that, the implementation in [6] does not use the Montgomery Ladder and performs over Koblitz curves, which speeds up the point multiplication at the cost of some security.

The only other light-weight implementation of BEC point multiplication is found in [9]. Many of the internals of our point multiplier are different. For instance, this implementation uses a circular register structure, and also a different bit-serial multiplier in Polynomial Basis. A Polynomial Basis parallel squaring unit was used in this implementation, which is costly when compared to the GNB. This implementation uses Common-Z differential coordinate system for the Montgomery Ladder, but each step requires 8 multiplications. Our implementation requires only 6 multiplications, representing a reduction of latency in the Montgomery Ladder by approximately 25%. Lastly, this implementation requires a register file to hold 6 registers, whereas our register file only requires 5 registers. Hence, our implementation features a smaller and faster point multiplication scheme than that in [9].

The introduction of extremely area-efficient crypto-processors with comb-serial multiplication schemes [23] like the one proposed in [7] indicates that there is a need for new trade-off for future implementations of these ECC targeted at RFID chips. Bit-parallel multiplication architectures are among the fastest approaches to perform finite field multiplications, but this requires a tremendous amount of area. Digit-serial schemes require a factor more of cycles, but use less area. The most popular scheme for RFID chip point multiplication is bit-serial, which requires a fraction of the area of digit-serial and requires m cycles to perform a multiplication. Comb-serial multiplication takes this a step further by performing small multiplications over many small combs. Depending on the multiplication scheme, this could require more than m cycles but holds new records for area-efficiency. The work presented in [7] is among the smallest ECC co-processors, even in $\mathbb{F}_{2^{283}}$. It was designed as a drop-in concept, such that the co-processor can share RAM blocks with a microcontroller. This implementation utilizes a comb-serial multiplicationscheme in polynomial basis over Koblitz curves. As such, the latency of each operation is larger than that of this work. Field addition, squaring, and multiplication require 60, 200, and 829 cycles, respectively. This implementation needs space to hold 14 intermediate elements throughout the point multiplication operation. Including the constants, our implementation requires 9 intermediate values. The area of the co-processor without the RAM for the register file is 4,323 GE. Moreover, in [7], the RAM results that were included were extrapolated from a different implementation of ECC appeared in [22]. With these extrapolated results, the total area of the co-processor would be 10,204 GE. Our crypto-processor with the register file uses 87% more area, but performs the point multiplication approximately three times faster, reducing the need to run at higher speeds to meet timing requirements in a device. Further, [7] utilizes zero-free tau-adic expansion to enforce a constant pattern of operations, similar to the Montgomery ladder [11], to protect against timing and power analysis attacks. However, this new technique has not been thoroughly explored like the Montgomery ladder. Furthermore, the co-processor does not have any protection against exceptional points attacks such as the ones presented in [16]. In summary, for higher levels of security as was implemented in [7], the time complexity was several factors higher, but the area was comparable to an implementation of a smaller finite field. As there is a push for larger field sizes for

higher security levels, the time complexity of the comb-serial method of multiplication and other operations becomes inefficient.

5 Conclusion

In this paper, it is shown that new mixed w-coordinate differential addition and doubling formulas for binary Edwards curve produce a fast, small, and secure implementation of point multiplication. Corrected formulas for addition in this coordinate system have been provided and proven. Binary Edwards curves feature a complete and unified addition formula. The future of point multipliers targeted at RFID technology depends on the trade-offs among area, latency, and security. The binary Edwards curves implementation presented in this paper has demonstrated that BEC is highly-competitive with the dominant elliptic curve systems standardized by NIST and IEEE. As such, new standardizations that include binary Edwards curves are necessary for the future of elliptic curve cryptography. The detailed analysis in this paper also suggests that binary Edwards curves are among the fastest and most secure curves for point multiplication targeting resource-constrained devices.

6 Acknowledgment

The authors would like to thank the reviewers for their constructive comments. This material is based upon work supported by the National Science Foundation under Award No. CNS-1464118 to Reza Azarderakhsh.

References

1. K. Kim, C. Lee, and C. Negre: Binary edwards curves revisited. In Meier, W., Mukhopadhyay, D., eds.: Progress in Cryptology – INDOCRYPT 2014. Springer International Publishing (2014) 393–408
2. Hankerson, D.R., Vanstone, S.A., Menezes, A.J.: Guide to Elliptic Curve Cryptography. Springer-Verlag New York Inc (2004)
3. U.S. Department of Commerce/NIST: National Institute of Standards and Technology. Digital Signature Standard, FIPS Publications 186-2, (January 2000)
4. IEEE Std 1363-2000: IEEE Standard Specifications for Public-Key Cryptography, (Jan. 2000)
5. E. Wenger and M. Hutter: Exploring the design space of prime field vs. binary field ecc-hardware implementations. In Laud, P., ed.: Information Security Technology for Applications. Volume 7161 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 256–271
6. R. Azarderakhsh, K. U. Jarvinen, and M. Mozaffari Kermani: Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications. IEEE Trans. on Circuits and Systems 61-I(4), 1144–1155 (2014)
7. S. S. Roy, K. Jarvinen, and I. Verbauwhede: Lightweight coprocessor for Koblitz curves: 283-bit ECC including scalar conversion with only 4300 gates. Cryptology ePrint Archive, Report 2015/556 (2015) http://eprint.iacr.org/.
8. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede: Elliptic-Curve-Based Security Processor for RFID. IEEE Transactions on Computers 57(11), 1514–1527 (2008)
9. Kocabas, U., Fan, J., Verbauwhede, I.: Implementation of Binary Edwards Curves for Very-Constrained Devices. In: Proceedings of 21st International Conference on Application-specific Systems Architectures and Processors (ASAP 2010). 185–191 (2010)
10. Bernstein, D.J., Lange, T., Farashahi, R.R.: Binary Edwards Curves. In: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008). Volume 5154. 244–265 (2008)
11. Montgomery, P.L.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Mathematics of computation, 243–264 (1987)
12. Lopez, J., Dahab, R.: Fast Multiplication on Elliptic Curves Over $GF(2^m)$ Without Precomputation. In: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999). 316–327 (1999)

13. Farashahi, R., Joye, M.: Efficient Arithmetic on Hessian Curves. In: Proceedings of The 13th International Conference on Practice and Theory of Public Key Cryptography (PKC 2010). 243–260 (2010)

14. R. Azarderakhsh A. Reyhani-Masoleh: Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis. IEEE Trans. Very Large Scale Integr. Syst. 20(8), 1453–1466 (August 2012)

15. Y. Lee and I. Verbauwhede: A compact architecture for montgomery elliptic curve scalar multiplication processor. In Kim, S., Yung, M., Lee, H.W., eds.: Information Security Applications. Volume 4867 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 115–127

16. T. Izu and T. Takagi: Exceptional procedure attack on elliptic curve cryptosystems. In Desmedt, Y., ed.: Public Key Cryptography, PKC 2003. Volume 2567 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 224–239

17. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M., ed.: Advances in Cryptology - CRYPTO' 99. Volume 1666 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1999) 388–397

18. R. Azarderakhsh, D. Jao, and H. Lee: Common subexpression algorithms for space-complexity reduction of Gaussian normal basis multiplication. IEEE Transactions on Information Theory 61(5), 2357–2369 (2015)

19. Itoh, T., Tsujii, S.: A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. Information Computing 78(3), 171–177 (1988)

20. Wenger, E., Hutter, M.: A hardware processor supporting elliptic curve cryptography for less than 9 kges. In Prouff, E., ed.: Smart Card Research and Advanced Applications. Volume 7079 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 182–198

21. Pessl, P., Hutter, M.: Curved tags - a low-resource ecdsa implementation tailored for rfid. In Saxena, N., Sadeghi, A.R., eds.: Radio Frequency Identification: Security and Privacy Issues. Volume 8651 of Lecture Notes in Computer Science. Springer International Publishing (2014) 156–172

22. E. Wenger: Hardware architectures for MSP430-based wireless sensor nodes performing elliptic curve cryptography. In Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R., eds.: Applied Cryptography and Network Security. Volume 7954 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 290–306

23. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. 1st edn. CRC Press, Inc., Boca Raton, FL, USA (1996)

7 Appendix

7.1 Subroutines

This contains a code listing of the program in assembly.

Algorithm 3 shows the Itoh-Tsujii [19] inversion subroutine for $\mathbb{F}_{2^{283}}$. This follows the addition chain (1,2,4,8,16,17,34,35,70,140,141,282). Eleven multiplications are required for this binary field. A similar approach was done for $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$.

Algorithm 3 Itoh-Tsujii [19] Inversion Subroutine for $GF(2^{283})$

SQ T0 T1
MULT T1 T1 –2^2-1
SWAP T0 R0
SQ T1 T0
SQ T0 T0
MULT T1 T0 –2^4-1
SQ T0 T1
SQ T1 T1 3 Times
MULT T1 T0 –2^8-1
SQ T0 T1
SQ T1 T1 7 Times
MULT T1 T0 –2^16-1
SWAP T1 R0
SQ T0 T0
MULT T1 T0 –2^17-1
SWAP T1 R0
SQ T0 T1
SQ T1 T1 16 Times
MULT T1 T0 –2^34-1
SWAP T1 R0
SQ T0 T0
MULT T1 T0 –2^35-1
SWAP T1 R0
SQ T0 T1
SQ T1 T1 34 Times
MULT T1 T0 –2^70-1
SQ T0 T1
SQ T1 T1 69 Times
MULT T1 T0 –2^140-1
SWAP T1 R0
SQ T0 T0
MULT T1 T0 –2^141-1
SQ T0 T1
SQ T1 T1 140 Times
MULT T1 T0 –2^282-1
SQ T0 T0

Algorithm 4 shows the half-trace subroutine. This is a simple double square and add routine that produces the result after $\frac{m-1}{2}$ iterations.

Algorithm 4 Half-Trace Subroutine

SQ T0 T1
SQ T1 T1
ADD T1 T0
{SQ T1 T1
SQ T1 T1
ADD T1 T0} loop for $\frac{m-2}{2}$ times

Algorithm 5 shows the beginning of the main program that was used. This includes the initialization of the point and the repeated step of the Montgomery ladder [11].

Algorithm 5 General Program Flow

INIT
SWAP R5 T0
ADD R6 T0
SQ T0 T1
ADD T0 T1
SQ T1 R0 –W4
SWAP R4 T1
ADD R0 T1 –Z4
MULT T1 T0 –W1 revised
SWAP T1 R0 –W1 W4 Z4
STEP
SWAP T0 T0 –OUTPUT register selected by k bit
ADD T0 T1
SQ T1 R1
SWAP T1 R0
SQ T1 R0
ADD T0 T1
MULT T1 T0
SQ T0 R2 –S
SWAP R1 T1
SWAP R3 T0 –1/w0
MULT T0 R1 –E
ADD R1 T1 –U
ADD R0 R1 –V
SQ R0 R0
SWAP R0 T1
SWAP R4 T0 –d1
MULT T1 T0
ADD R2 T0 –T
SWAP R0 T1
MULT T0 T1 –W3
SWAP T1 R1
MULT T0 T0 –Z'
SWAP T0 R2
MULT T0 T1 –W4
SWAP R0 R2
SWAP T0 R1
SWAP T0 T0 –Output register selected by k bit. Repeat for every step

Algorithm 6 shows the end of the main program that was used. This includes the recovery of w_2, w_3, x_2, y_2.

Algorithm 6 General Program Flow (cont.)

RECOVER X
SWAP T0 R1
SWAP T1 R2
SWAP R0 T0
Invert T0 –1/Z
SWAP R2 T1
MULT T1 R2 –w3
SWAP R1 T1
MULT T1 T1 –w2
SWAP R5 T0
ADD R6 T0
MULT T1 R0
ADD R0 T0
ADD T1 T0
SWAP R0 T1 –(w1w2+w1+w2) w1w2 w2 0 w3
MULT T1 T0
ADD T1 T0
ADD R4 T0 –d1+w1w2+w1w2*(w1+w2+w1w2) w1w2 w2 0 w3
SWAP T1 R2
MULT T1 R2 –1st part of the numerator – 0 0 w2 0 1st
SWAP R5 T0
ADD R6 T0
ADD R0 T0 –w1+w2
SWAP R4 T1
MULT T1 T0
ADD T0 R2 –0 0 w2 0 1st+2nd
SQ R6 T0
ADD R6 T0
SQ R0 T1
ADD R0 T1
MULT T1 T0
ADD T0 R2 –Numerator complete, compute inversion now
SWAP R5 T0
ADD R6 T0 –w1 0 0 0 Numerator
SQ T0 T1
ADD T1 T0 –w1^2+w1 0 0 0 Numerator, now inversion
Invert T0 –1/(w0^2+w0)
SWAP R2 T1
MULT T1 T0
T0 = HalfTrace(T0) –x2 or x2+1
RECOVER Y
SQ T0 R2
ADD T0 R2
SWAP T0 R1
SWAP R4 T0
ADD R2 T0
Invert T0 –1/(d+x+x^2)
SWAP T1 R2
MULT T1 T0
SWAP R4 T1
MULT T1 T0
T0 = HalfTrace(T0) –y2 or y2+1
SWAP T0 T1
SWAP R1 T0 –Solution is x, y in T0 T1