A New Double Point Multiplication Method and its Implementation on Binary Elliptic Curves with Endomorphisms

Reza Azarderakhsh and Koray Karabina

Abstract

Efficient and high-performance implementation of point multiplication is crucial for elliptic curve cryptosystems. In this paper, we present a new double point multiplication algorithm based on differential addition chains. We use our scheme to implement single point multiplication on binary elliptic curves with efficiently computable endomorphisms. Our proposed scheme has a uniform structure and has some degree of built-in resistance against side channel analysis attacks. We design a crypto-processor based on the proposed algorithm for double point multiplication and evaluate its area and time efficiency on FPGA. To the best of the authors' knowledge, this is the first hardware implementation of single point multiplication (using double point multiplication) on elliptic curves with efficiently computable endomorphisms. Our analysis and timing results show that the expected acceleration in point multiplication is considerable. Prototypes of the proposed architectures are implemented and experimental results are presented.

Index Terms

Elliptic curve cryptosystems, endomorphism, differential addition chains, double point multiplication, digit-level multiplier.

1 Introduction

In 1985, Miller [1] and Koblitz [2] independently showed that the group of rational points on elliptic curves over finite fields can be used for public-key cryptography. Since then, elliptic curve cryptography (ECC) has been identified and employed as an efficient and suitable scheme for public key cryptographic systems. ECC offers similar security with smaller key size and its security relies on the difficulty of solving the elliptic curve discrete logarithm problem (ECDLP) [3]. The principal operation in elliptic curve cryptographic systems is point multiplication. Therefore, several effort in the literature have focused on developing efficient and high-performance techniques to compute point multiplication on various forms of elliptic curves.

R. Azarderakhsh is with the Center for Applied Cryptographic Research (CACR), Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. E-mail address: razarder@math.uwaterloo.ca.

K. Karabina is with the Department of Mathematics, Bilkent University, Bilkent, Ankara, Turkey, 06800. E-mail address: karabina@fen.bilkent.edu.tr.

Let $E:\ Y^2+XY=X^3+aX^2+b$, together with a point at infinity, be an ordinary non-supersingular elliptic curve defined over $\mathbb{F}_{2^n}$. Given an integer k and a point $P \in E(\mathbb{F}_{2^n})$, a *(single) point multiplication* algorithm computes $kP \in E(\mathbb{F}_{2^n})$. Given two integers k_1, k_2 and two points P, $Q \in E(\mathbb{F}_{2^n})$, a *double point multiplication* algorithm computes $k_1P + k_2Q \in E(\mathbb{F}_{2^n})$. Elliptic curve based cryptographic schemes rely heavily on efficient point multiplication and double point multiplication algorithms. For example, in an elliptic curve digital signature scheme, the signer has to compute kP where k is randomly chosen by the signer, and $P \in E$ is a domain parameter. For verifying a signature, the verifier obtains the public key $Q \in E$ of the signer and computes $k_1P + k_2Q$ for certain integers k_1 and k_2. One can obviously perform double point multiplication at a cost of performing two single point multiplications. A more efficient method is to compute k_1P+k_2Q simultaneously. Two such methods are Straus-Shamir's trick (see Algorithm 14.88 in [4]) and interleaving [5]. Other alternatives for computing simultaneous double point multiplication are based on differential addition chains; see for instance [6], [7], [8], and [9]. One advantage of using differential addition chains is that the resulting algorithms are potentially resistant against side-channel analysis attacks due to the uniform pattern of operations executed. On the other hand, algorithms based on differential addition chains with uniform pattern suffer from being less efficient in comparison to the traditional methods; and if such algorithms are optimized for efficiency then the uniform property is sacrificed in general. As an example, Montgomery's continued fraction algorithm CFRC [6] has a uniform pattern whereas its optimized version PRAC [6] is not uniform.

Double point multiplication can be used to obtain fast (single) point multiplication in certain cases. To be more concrete, suppose that Ψ is an efficiently computable endomorphism of E such that $\Psi(P) = \lambda P$, where $P \in E(\mathbb{F}_{2^n})$ is of prime order r and $\lambda \in [2, r-2]$ is an integer. If a scalar k can be written as $k = k_1 + k_2\lambda \mod r$, where k_1, $k_2 \approx \sqrt{r}$, then $kP = k_1P + k_2\Psi(P)$ can be computed using a simultaneous double point multiplication algorithm. Computational speed-up is achieved if the cost of performing a double point multiplication (plus the cost of evaluating Ψ) is less than twice the cost of performing a single point multiplication. Gallant, Lambert and Vanstone (GLV) [10] introduced this technique and obtained fast point multiplication on certain ordinary elliptic curves (the so-called GLV curves) defined over finite fields of characteristic greater than three. Later, Galbraith, Lin and Scott [11] generalized this technique to larger classes of curves (known as the GLS curves). Recently, Hankerson, Karabina and Menezes [12] analyzed GLS curves over binary fields and showed that the GLV technique is effective in a large class of elliptic curves in the sense that GLV point multiplication is faster than the traditional point multiplication methods. In [12], the interleaving technique was used in the double point multiplication (GLV point multiplication) algorithm.

In this paper, we focus on a double point multiplication algorithm which has a uniform structure as opposed to Straus-Shamir's trick or interleaving techniques. This suggests using a differential addition chain based multiplication. Differential addition chains yield an extra advantage in an elliptic curve setting because there exist differential point addition and doubling formulas (see [13], [14]). These formulas are generalization of Montgomery's formulas [15] using only the x-coordinates of points, and are more efficient than the traditional point addition and doubling formulas. Differential addition chain based elliptic curve double point multiplication algorithms have

been previously studied by Stam [14] and Bernstein [9]. In [14], Stam adapted Montgomery's PRAC algorithm [6] and proposed a double point multiplication algorithm in elliptic curves over fields of characteristic two. Stam's method costs 1.49 additions and 0.43 doublings per scalar bit and improves on previous results. However, the proposed algorithm in [14] does not have a uniform structure, and it is concluded in [14] that protection against timing and power analysis attacks has not been supported. More recently, Bernstein [9] proposed two double point multiplication algorithms based on new differential addition-subtraction chains. The first algorithm in [9] has a uniform structure and costs two additions and one doubling per scalar bit. Bernstein's second algorithm in [9] is more efficient (1.43 additions/subtractions and 0.347 doublings per scalar bit) but does not have a uniform structure.

We propose and analyze a new double point multiplication algorithm based on an adaptation of Montgomery's PRAC algorithm. Our algorithm has a uniform structure that yields some degree of built-in resistance against side-channel analysis attacks. Our algorithm requires 1.4 additions and 1.4 doublings per scalar bit on average. Hence, our proposal can be seen as an alternative to Bernstein's first algorithm proposed in [9]. To evaluate the practical performance of the proposed scheme, we design a new crypto-processor architecture based on our proposed double point multiplication algorithm and code it using VHDL and implement it on Xilinx Virtex$^{\text{TM}}$-4 FPGA. We employ a digit-level polynomial basis multiplier and investigate the efficiency of the proposed hardware architecture based on different digit sizes. Moreover, we implement point multiplication on a binary generic curve and compare the FPGA implementation results with the ones obtained for GLS curves using our double point multiplication algorithm.

The rest of the paper is organized as follows. In Section 2, we present our new double point multiplication algorithm, its analysis, and a comparison with previous work. We briefly review preliminaries and point multiplication on elliptic curves with endomorphisms in Section 3. Section 4 specifies the elliptic curves that we use in our implementation. In Section 5, a parallelization technique is described to speed up point multiplication on binary elliptic curves. In Section 6, our proposed architecture for point multiplication using double point multiplication algorithm is presented. In Section 7, we implement single point multiplication using the proposed architecture for double point multiplication, and compare its area and timing results with a traditional single point multiplication. Finally, we conclude the paper in Section 8.

2 A New Double Point Multiplication Algorithm

Let G be an abelian additive group. We describe a new double point multiplication algorithm to compute $aP + bQ$ where $a, b \in \mathbb{Z}$ and $P, Q \in G$. We may assume without loss of generality that a and b are positive because $aP = -a(-P)$. Our algorithm is an adaptation of Montgomery's PRAC algorithm [6].

First we introduce some notation. Let $\vec{u} = (u_0, u_1)$ and $\vec{v} = (v_0, v_1)$ be two-dimensional vectors with integer components, and let $\vec{R} = (P, Q)$ denote a two-dimensional vector where the components are group elements. For an integer i, we define $i\vec{u} = (iu_0, iu_1)$ where the component-wise scalar multiplication is performed over integers; and $i\vec{R} = (iP, iQ)$ where the component-wise scalar multiplication is performed in G. We define $\vec{u} \cdot \vec{R} = u_0P + u_1Q$, $\vec{v} \cdot \vec{R} = v_0P + v_1Q$, and $\Delta_{\vec{u},\vec{v}} = \vec{u} - \vec{v} = (u_0 - v_0, u_1 - v_1)$. When it is clear from the context, we use $\vec{\Delta}$ for $\Delta_{\vec{u},\vec{v}}$ to

Algorithm 1 Double point multiplication algorithm

Inputs: $a > 0, b > 0, P, Q$
Output: $aP + bQ$
1: $d \leftarrow a$, $e \leftarrow b$, $\vec{u} \leftarrow (1,0)$, $\vec{v} \leftarrow (0,1)$, $\vec{\Delta} \leftarrow (1,-1)$
2: $R_u \leftarrow P$, $R_v \leftarrow Q$, $R_\Delta \leftarrow P - Q$
3: **While** $d \neq e$ **do**
4: Execute the first applicable rule in Table 1
5: **end While**
6: Using single point multiplication with input d and $(R_u + R_v)$, compute and return $d(R_u + R_v)$

Table 1. Update rules for double point multiplication

Rule	Condition	d	e	$\vec{u}$	$\vec{v}$	$\vec{\Delta}$	R_u	R_v	R_Δ
if $d > e$									
R1	$d \equiv e \pmod 2$	$(d-e)/2$	e	$2\vec{u}$	$\vec{u}+\vec{v}$	$\vec{\Delta}$	$2R_u$	$R_u + R_v$	R_Δ
R2	$d \equiv 0 \pmod 2$	$d/2$	e	$2\vec{u}$	$\vec{v}$	$\vec{u}+\vec{\Delta}$	$2R_u$	R_v	$R_u + R_\Delta$
R2′	$e \equiv 0 \pmod 2$	d	$e/2$	$\vec{u}$	$2\vec{v}$	$\vec{\Delta}+(-\vec{v})$	R_u	$2R_v$	$R_\Delta + (-R_v)$
else									
R1′	$d \equiv e \pmod 2$	d	$(e-d)/2$	$\vec{u}+\vec{v}$	$2\vec{v}$	$\vec{\Delta}$	$R_u + R_v$	$2R_v$	R_Δ
R2	$d \equiv 0 \pmod 2$	$d/2$	e	$2\vec{u}$	$\vec{v}$	$\vec{u}+\vec{\Delta}$	$2R_u$	R_v	$R_u + R_\Delta$
R2′	$e \equiv 0 \pmod 2$	d	$e/2$	$\vec{u}$	$2\vec{v}$	$\vec{\Delta}+(-\vec{v})$	R_u	$2R_v$	$R_\Delta + (-R_v)$

simplify the notation. Finally, we set $R_u = \vec{u} \cdot \vec{R}$, $R_v = \vec{v} \cdot \vec{R}$, and $R_\Delta = \overrightarrow{\Delta} \cdot \vec{R}$.

Algorithm 1 starts with $d = a$, $e = b$, $\vec{R} = (P, Q)$, $\vec{u} = (1,0)$, $\vec{v} = (0,1)$, and $\vec{\Delta} = (1,-1)$. These initial values yield $R_u = P$, $R_v = Q$, $R_\Delta = R_u - R_v = P - Q$, and $dR_u + eR_v = aP + bQ$. During the execution of the algorithm, $d, e, \vec{u}, \vec{v}, \vec{\Delta}, R_u, R_v, R_\Delta$ are updated so that $dR_u + eR_v = aP + bQ$ and $R_\Delta = R_u - R_v$ hold, $d, e > 0$, and $(d + e)$ decreases until $d = e$. When $d = e$, we will have $aP + bQ = dR_u + eR_v = d(R_u + R_v)$ which can be computed using a single point multiplication algorithm with base $R_u + R_v$ and scalar d. We should note that when $\gcd(a, b) = 1$, $(d + e)$ in the algorithm will decrease until $d = e = 1$ and we will have $aP + bQ = d(R_u + R_v) = R_u + R_v$.

Note that each rule in Table 1 requires an addition and a doubling in G. R2′ requires an extra negation of a group element. Moreover addition and doubling operations can be performed using differential addition and differential doubling formulas as the difference of the group elements to be added are known by construction. We give an example in Table 2 to show intermediate values of Algorithm 1 with input $a = 35, b = 17, P, Q$ and $P - Q$. Note that in step 6 of Algorithm 1, we have $d = 1$, $R_u = 24P + 8Q$, $R_v = 11P + 9Q$, and the output is $R_u + R_v = 35P + 17Q$, as required.

Algorithm 1 is simultaneously processing the scalars d, e, and the points R_u, R_v, R_Δ. Alternatively, we can use Algorithm 2 which is essentially the same as Algorithm 1 except that Algorithm 2 first precomputes a sequence S via processing the scalars d, e, and then computes $aP + bQ$ via processing the points R_u, R_v, R_Δ. The S-sequence will have entries $\mathrm{R1}, \mathrm{R2}, \mathrm{R1}'$, or $\mathrm{R2}'$ which determine how to update R_u, R_v and R_Δ. For the example in Table 2, the corresponding S-sequence is $S = \{\mathrm{R1}, \mathrm{R1}', \mathrm{R2}, \mathrm{R2}', \mathrm{R1}, \mathrm{R2}, \mathrm{R2}\}$.

Table 2. An example for Algorithm 1. The input is $a = 35, b = 17, P, Q$

Rule	d	e	$\vec{u}$	$\vec{v}$	$\vec{\Delta}$	R_u	R_v	R_Δ
	35	17	$(1,0)$	$(0,1)$	$(1,-1)$	P	Q	P-Q
R1	9	17	$(2,0)$	$(1,1)$	$(1,-1)$	2P	P+Q	P-Q
R1′	9	4	$(3,1)$	$(2,2)$	$(1,-1)$	3P+Q	2P+2Q	P-Q
R2′	9	2	$(3,1)$	$(4,4)$	$(-1,-3)$	3P+Q	4P+4Q	-P-3Q
R2′	9	1	$(3,1)$	$(8,8)$	$(-5,-7)$	3P+Q	8P+8Q	-5P-7Q
R1	4	1	$(6,2)$	$(11,9)$	$(-5,-7)$	6P+2Q	11P+9Q	-5P-7Q
R2	2	1	$(12,4)$	$(11,9)$	$(1,-5)$	12P+4Q	11P+9Q	P-5Q
R2	1	1	$(24,8)$	$(11,9)$	$(13,-1)$	24P+8Q	11P+9Q	13P-Q

Algorithm 2 Double point multiplication algorithm with precomputation

Inputs: $a > 0,\ b > 0,\ P,\ Q$
Output: $aP + bQ$
1: $d \leftarrow a$, $e \leftarrow b$, $L \leftarrow 0$
2: $R_u \leftarrow P,\ R_v \leftarrow Q,\ R_\Delta \leftarrow P - Q$
3: **While** $d \neq e$ **do**
4: Update d and e by executing the first applicable rule in Table 1
5: $L \leftarrow L + 1,\ S[L] \leftarrow R$, where $R \in \{\mathrm{R1}, \mathrm{R2}, \mathrm{R1}', \mathrm{R2}'\}$ is the rule applied in the previous step
5: **end While**
6: **For** i **from** 1 to L do
7: **If** $S[i] = \mathrm{R1}$ **then**
8: $(R_u, R_v, R_\Delta) \leftarrow (2R_u, R_u + R_v, R_\Delta)$
9: **else if** $S[i] = \mathrm{R2}$ **then**
10: $(R_u, R_v, R_\Delta) \leftarrow (2R_u, R_v, R_u + R_\Delta)$
11: **else if** $S[i] = \mathrm{R1}'$ **then**
12: $(R_u, R_v, R_\Delta) \leftarrow (R_u + R_v, 2R_v, R_\Delta)$
13: **else if** $S[i] = \mathrm{R2}'$ **then**
15: $(R_u, R_v, R_\Delta) \leftarrow (R_u, 2R_v, R_\Delta + (-R_v))$
16: **end If**
17: **end For**
18: Using single point multiplication with input d and $(R_u + R_v)$, compute and return $d(R_u + R_v)$

2.1 Correctness and Analysis

Let d, e, R_u, R_v be as defined before, and assume that $a > 0, b > 0, P, Q, P - Q$ is an input to Algorithm 1. In Algorithm 1, d and e are updated so that d, e are always positive, and $(d+e)$ strictly decreases. Therefore, the while loop in the algorithm must finish after finitely many steps with $d = e$. When $d = e$, since $d\vec{u} + e\vec{v} = (a, b)$ is kept invariant while applying the rules in Table 1, we must have

$$\begin{aligned} d\vec{u} + e\vec{v} &= d(\vec{u} + \vec{v}) = d((u_0, u_1) + (v_0, v_1)) \\ &= (d(u_0 + u_1), d(v_0 + v_1)) = (a, b), \end{aligned}$$

and so Algorithm 1 outputs $aP + bQ$. Moreover, we deduce from the above equality that d must divide both a and b which implies $d | \gcd(a, b)$. In particular, if $\gcd(a, b) = 1$, then we will have $d = e = 1$ right after the while loop in Algorithm 1.

Table 3. Practical behavior of Algorithm 2 at the 128-bit security level. 10^6 pairs (a, b) were randomly chosen with $a, b \in [2^{127}, 2^{128})$ and $a, b \in [2^{255}, 2^{256})$.

	Average		Standard deviation	
	$a, b \in [2^{127}, 2^{128})$	$a, b \in [2^{255}, 2^{256})$	$a, b \in [2^{127}, 2^{128})$	$a, b \in [2^{255}, 2^{256})$
$\log_2 a$, $\log_2 b$	128	256	0	0
$L(a, b)$	$1.401 \cdot \log_2 a$	$1.406 \cdot \log_2 a$	$0.054 \cdot \log_2 a$	$0.037 \cdot \log_2 a$
d in step 18	5.741	5.648	365.4	286.9
Rule	Average usage		Standard deviation	
R1	0.2507	0.2503	0.029	0.020
R2	0.2493	0.2497	0.031	0.022
R1′	0.2507	0.2503	0.029	0.020
R2′	0.2493	0.2497	0.031	0.022

The main issue to determine the efficiency of Algorithm 1 (and so of Algorithm 2) is to estimate the length $L(a, b)$ of the S-sequence in Algorithm 2 (the number of iterations in the while loop in Algorithm 1). It is easy to show that $L(a, b) \leq \log_2 a + \log_2 b$. In fact, this bound is tight because if $a = 2^m$ and $b = 2^m - 1$, then $L(a, b) = 2m$.

In our experiments we observed that the length $L(a, b)$ of the S-sequence in Algorithm 2 in practice is remarkably smaller than the upper bound $\log_2 a + \log_2 b$. Moreover, the behavior of Algorithm 1 becomes very stable as $(a+b)$ gets larger. We tested the performance of Algorithm 2 with 10^6 randomly chosen pairs (a, b) such that, $a, b \in [2^{127}, 2^{128})$ and $a, b \in [2^{255}, 2^{256})$. The intervals $[2^{127}, 2^{128}), [2^{255}, 2^{256})$ are relevant for obtaining fast single point multiplication and double point multiplication algorithms at the 128-bit security level (see Section 4.3). Our experimental evidence suggests that, on average, $L(a, b) = 1.4 \log_2 a$ and $d = 6$ in step 18 of Algorithm 2. Moreover, each R1,R2,R1′,R2′ is used in around 25% of the total number of iterations. The variance is very high for the size of d in step 18 of Algorithm 2 because, as one might expect, $\gcd(a, b) = 1$ most of the time. Hence, we may have the following conjecture on the expected running time of our proposed algorithm.

Conjecture 1. *Let G be an additive group with $P, Q \in G$, and $a, b \in \mathbb{Z}$ with $a, b \in [2^{\ell-1}, 2^{\ell}), \ell \in \{128, 256\}$. Using Algorithm 1 or Algorithm 2, $aP + bQ$ can on average be computed in about 1.4ℓ additions and 1.4ℓ doublings in G.*

Remark 2. Algorithm 2 is basically performing the binary Euclidean algorithm while constructing the S-sequence. It has been proved under certain assumptions that if the input to the binary Euclidean algorithm is an odd positive pair of integers of each ℓ-bit length and uniformly chosen at random, then the average number of subtractions in the binary Euclidean algorithm is asymptotically 0.7ℓ; see [16]. The number of subtractions in the binary Euclidean algorithm corresponds to the number of executions of the rules R1 and R1′in Algorithm 2. Furthermore, assuming that, after executing R1 (R1′), the updated value of d (e) is an odd multiple of 2^k with probability 2^{-k}, we deduce that $L(a, b) \approx 0.7\ell \sum_{k=0}^{\infty} k/2^k = 1.4\ell$ that agrees with our empirical results.

Note that the value of d in step 18 of Algorithm 2 is not necessarily 1, and in the case that $d > 1$ an attacker might collect some useful information on a and b because $d | \gcd(a, b)$, and d is used in a single point multiplication algorithm after step 18 of Algorithm 2. For example, a legitimate user with her secret key k may compose $k = k_1 + k_2\lambda \mod r$ as described in Section 2, and use Algorithm 2 to compute $kP = aP + bQ$, where $a = k_1$, $b = k_2$, and $Q = \lambda P$.

Then the value of d in step 18 of Algorithm 2 would be equal to the odd part of $\gcd(a, b)$ which can be recovered by an attacker via side channel attacks. We argue that the attacker cannot learn much about the secret k by adapting this strategy. First, we note that if a and b are integers chosen at random, then the probability that $\gcd(a, b) = d$ is $p(d) = (6/(\pi d)^2)$; see [16]. It is plausible to assume that it is hard to distinguish the distribution of the pair of integers (k_1, k_2) obtained from the decomposition of randomly chosen integers k (using the decomposition method in Section 2) from the distribution of the randomly chosen pair of integers (a, b). Under this heuristic, we expect that $d = 1$ in step 18 of Algorithm 2 with probability at least $p(1) \approx 0.6$. Similarly, we can argue that $d > 3$ in step 18 of Algorithm 2 with probability at most $1 - (p(1) + p(2) + p(3)) \approx 0.17$; $d > 7$ with probability at most $1 - (p(1) + \cdots + p(7)) \approx 0.08$, etc.. In order to lower the attacker's success probability, the user might proceed as follows: First, precompute a small set of points cP for a set of small integers c. Instead of decomposing only k, decompose all the integers in a (small) set $S = \{k + c : \ c \text{ is small}\}$ and choose the one, say $(k + c)$, at random that yields the smallest $\gcd(k_1, k_2)$, say d_{min}. After computing $(k + c)P$ using Algorithm 2, compute $kP = (k + c)P - cP$. Note that for a set S of size $|S| \geq 2$ we expect that $d_{\text{min}} = 1$.

An attacker might also try to recover the secret exponent of a legitimate user by using the variance in $L(a, b)$. First of all, the standard deviation is very small (see Table 3) and also it is not clear to the authors how to manipulate this variance in an attack. In any case, our suggestion of using the set S at the end of the previous paragraph can be applied to run Algorithm 2 with ℓ-bit integers a and b such that $L(a, b)$ does not deviate much from its expected value 1.4ℓ (see Conjecture 1).

2.2 Comparison with Previous Work

There are three crucial aspects of our algorithm which make it attractive for implementing.

First of all, assuming the inversion operation $P \mapsto -P$ can be performed at a negligible cost [1] and ignoring the cost of updating scalars d and e, the cost of applying each rule in Table 1 is dominated by an addition and a doubling operation. Therefore, the same types of operations are executed in Algorithm 1 which yields some degree of built-in resistance against side-channel analysis attacks. Second, the addition and the doubling operations in Algorithm 2 can be implemented using differential addition-doubling formulas which are in general more efficient than traditional addition-doubling formulas. Finally, as we already discussed in Section 3.2, double point multiplication can be used to speed-up single point multiplication in certain groups such as in the group of points on an elliptic curve with an efficiently computable endomorphism.

Our proposed algorithm is an adaptation of Montgomery's PRAC algorithm [6] which was originally proposed to compute the nth term of a second-degree recursive sequence. Montgomery's PRAC algorithm was previously adapted by Stam (see [8], [14]) to obtain a double point multiplication algorithm which on average requires 1.49 additions and 0.43 doublings per scalar bit, and which can benefit from differential addition-doubling formulas. However, the operations executed in Stam's adaptation do not have the uniform structure and hence are not likely

1. This is indeed the case in elliptic curves setting. Moreover, if differential addition and doubling formulas are deployed then the cost is literally zero.

Table 4. Comparison of our algorithm with some of the previously-known algorithms. The costs of addition, subtraction, and doubling are denoted by A, S, D, respectively.

Algorithm	Per bit cost	Uniform
Schoenmakers [8, Section 3.2.3]	$2.25A + 1.25D$	No
Akishita [7]	$2.25A + 0.75D$	No
Stam [14]	$1.49A + 0.43D$	No
New binary [9]	$2A + 1D$	Yes
New extended gcd [9]	$1.43A/S + 0.347D$	No
Algorithm 1	$1.4A + 1.4D$	Yes

to have protection against side channel analysis attacks which was also noted in [14]. More recently, Bernstein [9] proposed two methods, for constructing differential addition-subtraction chains, the *new binary chain* and the *new extended gcd chain*. Bernstein's new binary chain method yields a double point multiplication algorithm which requires to compute 2 additions and 1 doubling per scalar bit in a uniform add-double-add pattern. Bernstein's new extended chain method yields a double point multiplication algorithm which on average requires 1.43 addition/subtractions and 0.347 doublings per scalar bit, however, the operations do not follow a uniform pattern. Differential addition-doubling formulas can be utilized in both of these algorithms.

In Table 4, we compare our proposed algorithm with the above mentioned algorithms, and with two other double point multiplication algorithms proposed by Schoenmakers (see [8, Section 3.2.3]), and Akishita [7]. Even though there are many other techniques that yield double point multiplication algorithms such as the interleaving technique [5], we do not include them in our comparisons because differential addition-doubling formulas cannot be utilized in these algorithms, and the sequence of addition/doubling operations does not follow a uniform pattern during their execution.

3 PRELIMINARIES

In this section, we provide preliminaries of finite fields represented by polynomial basis and arithmetic over binary generic curves and GLS curves.

3.1 Binary Extension Fields

The finite field of characteristic two or binary extension field $\mathbb{F}_{2^m}$ is a finite field that contains 2^m different elements [17]. This finite field is an extension of $\mathbb{F}_2$ which contains 0 and 1. $\mathbb{F}_{2^m}$ is associated with an irreducible polynomial of degree m over $\mathbb{F}_2$ as

$$F(z) = f_m z^m + f_{m-1} z^{m-1} + \cdots + f_1 z + f_0, \tag{1}$$

where $f_i \in \mathbb{F}_2$. The field elements in $\mathbb{F}_{2^m}$ can be represented using different representation bases such as normal basis and polynomial basis [3] and [18]. The latter provides efficient implementation results and hence is considered in this paper.

Let $\alpha \in \mathbb{F}_{2^m}$ be a root of the irreducible polynomial $F(z)$, i.e., $F(\alpha) = 0$. Then the set $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ is known as the polynomial basis and an element $A \in \mathbb{F}_{2^m}$ can be represented as linear combinations of this set with a polynomial of degree $m-1$ over $\mathbb{F}_2$, as $A = \sum_{i=0}^{m-1} a_i \alpha^i$, where $a_i \in \mathbb{F}_2$. For simplicity, a bit-vector representation is commonly used and so that $A = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0)$, where a_{m-1} and a_0 are the most significant bit (MSB) and least significant bit (LSB), respectively.

3.2 Point Multiplication on Elliptic Curves with Endomorphisms

Let $q = 2^\ell$ and let

$$E/\mathbb{F}_q \ : \ Y^2 + XY = X^3 + aX^2 + b$$

be an elliptic curve defined over $\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q + 1 - t$. Let $a' \in \mathbb{F}_{q^2}$ be an element with $\mathrm{Tr}(a') = 1$, where $\mathrm{Tr} : \mathbb{F}_{q^2} \to \mathbb{F}_2$ denotes the trace function. Then the elliptic curve

$$E'/\mathbb{F}_{q^2} \ : \ Y^2 + XY = X^3 + a'X^2 + b$$

is the quadratic twist of E over $\mathbb{F}_{q^2}$, and $\#E'(\mathbb{F}_{q^2}) = (q-1)^2 + t^2$. It was shown in [12] that there exists an efficiently-computable endomorphism Ψ of E' defined over $\mathbb{F}_{q^2}$ such that

$$\begin{aligned} \Psi \ &: \ E' \to E' \\ &(x, y) \mapsto (x^q, y^q + (s^q + s)x^q), \end{aligned}$$

where $s \in \mathbb{F}_{q^4} \setminus \mathbb{F}_{q^2}$ satisfies $s^2 + s = a + a'$. Moreover, if $\#E'(\mathbb{F}_{q^2}) = hr$, where r is an odd prime and h is a (small) cofactor, then for any $P \in E'(\mathbb{F}_{q^2})[r]$, we have $\Psi(P) = \lambda P$ for some integer λ satisfying $\lambda^2 + 1 \equiv 0 \pmod{r}$. It can be shown that $\lambda = t^{-1}(q-1) \pmod{r}$. More interestingly, if k is an integer, then one can efficiently find two integers k_1 and k_2 such that

$$k = k_1 + k_2\lambda \mod r, \ |k_1|, |k_2| \leq (q+1)/\sqrt{2}$$

as follows [11]: First write

$$(k, 0) = \beta_1(1 - q, t) + \beta_2(t, q - 1)$$

for some rationals β_1, β_2. Note that $\beta_1 = ((1-q)/(t^2 + (q-1)^2))k$ and $\beta_2 = (t/(t^2 + (q-1)^2))k$. Then, let $b_1 = \lfloor \beta_1 \rceil$ and $b_2 = \lfloor \beta_2 \rceil$, where $\lfloor x \rceil$ is the nearest integer to x, and set

$$(k_1, k_2) = (k, 0) - (b_1(1 - q, t) + b_2(t, q - 1)).$$

It is clear that computing kP on E' is the same as computing

$$(k_1 + k_2\lambda)P = k_1P + k_2\Psi(P) = k_1P + k_2Q,$$

where $Q = \Psi(P)$. Moreover, if the cofactor h is small (i.e., $r \approx q$), then for an r-bit integer k, one would expect that

the bitlengths of k_1 and k_2 are half that of k. In Section 2, we describe an algorithm to simultaneously compute $k_1P + k_2Q$ given k_1, k_2, P and Q.

4 Our Choice of Elliptic Curves

In this section, we specify the elliptic curves to be used in our implementations in Section 7.

4.1 A Binary Generic Curve

Let $n = 257$, $q = 2^n$, $\mathbb{F}_q = \mathbb{F}_2[x]/(x^{257} + x^{41} + 1)$. A binary vector $\vec{b} = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)$ naturally corresponds to the finite field element $b = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ldots + b_1x + b_0 \in \mathbb{F}_q$. We denote the hexadecimal representation of $\vec{b}$ by $\vec{b}_H$. We choose

$\vec{b}_H = 026233F3E9C84A3A2BF5D662EF99575AF1BE3333D2F21D236B1BDA63E30C3475E$,

and set

$$E/\mathbb{F}_q : \; Y^2 + XY = X^3 + X^2 + b. \tag{2}$$

Then, $\#E(\mathbb{F}_q) = (q + 1 - t) = 2r$, where

$t = 215684806268585900188390555655618328127$, and

$r = 115792089237316195423570985008687907853162142262506271089363388730085320475873$

is an 256-bit prime.

4.2 A GLS Curve

Let $\ell = 127$, $q = 2^\ell$, $\mathbb{F}_q = \mathbb{F}_2[x]/(x^{127} + x^{63} + 1)$, $\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2 + u + 1)$. A binary vector $\vec{b} = (b_{\ell-1}, b_{\ell-2}, \ldots, b_1, b_0)$ naturally corresponds to the finite field element $b = b_{\ell-1}x^{\ell-1} + b_{\ell-2}x^{\ell-2} + \ldots + b_1x + b_0 \in \mathbb{F}_q$. We denote the hexadecimal representation of $\vec{b}$ by $\vec{b}_H$. We choose

$\vec{b}_H = 2BACF997126F185C3E67CB944EEB1168$,

and set

$$E'/\mathbb{F}_{q^2} : \; Y^2 + XY = X^3 + uX^2 + b. \tag{3}$$

Then, $\#E'(\mathbb{F}_{q^2}) = (q-1)^2 + t^2 = 2r$, where

$t = -880902903216571407$, and

$r = 144740111546645244279463731260859884814889948947070489652861672433814220790 89$

is an 253-bit prime. The endomorphism is defined by $\Psi : E' \to E'$, $(x, y) \mapsto (x^q, y^q + (u+1)x^q)$. If $P \in E'(\mathbb{F}_{q^2})[r]$, then $\Psi(P) = \lambda P$, where

$\lambda = 800802114842106653132700569325720912715596993263102454696425871443122201 8403$.

Remark 3. From our discussion in Section 3.2, an integer k can be decomposed as $k = k_1 + k_2\lambda \mod r$ with $\log_2 |k_1|, \log_2 |k_2| \leq 127$. Our experiments with 10^6 random choices of $k \in [(r-1)/2, r)$, indicate that the decomposition method specified in Section 3.2 yields on average $\log_2 |k_1| = \log_2 |k_2| = 124.56$. Moreover, if $a = |k_1|$ and $b = |k_2|$ are given as input to Algorithm 2, then the average length of the resulting S-sequence is $L(a, b) = 175.41$. Note that these results are in agreement with our findings in Table 3 as $176/125 \approx 1.4$. As a result, we may state the following conjecture.

Conjecture 4. *Let $E'/\mathbb{F}_{q^2}$ be the elliptic curve as in (3) with $P, Q \in E'(\mathbb{F}_{q^2})[r]$, and $a, b \in \mathbb{Z}$. Using Algorithm 1 or Algorithm 2, $aP + bQ$ can on average be computed in about 176 additions and 176 doublings in E'.*

Now, let $a_1 = b^{-1/8}$, and consider the elliptic curve

$$E''/\mathbb{F}_{q^2} : \; Y^2 + a_1 XY = X^3 + a_1^2 u X^2 + 1/a_1^2, \tag{4}$$

that is isomorphic to E' over $\mathbb{F}_{q^2}$. The isomorphism and its inverse are given by $\Phi : \; E' \to E'', \; (x, y) \mapsto (a_1^2 x, a_1^3 y)$ and $\Phi^{-1} : \; E'' \to E', \; (x, y) \mapsto (a_1^{-2} x, a_1^{-3} y)$, respectively. The curve equation (4) is better suited than (3) for implementing Algorithm 1 because deploying Stam's differential addition and doubling formulae for E'' (see Algorithm 4) yields a more optimized point multiplication algorithm than deploying López-Dahab's differential addition and doubling formulae (see Algorithm 3) for E'. Contrarily to a Montgomery ladder, point differences in double point multiplication are not fixed (i.e., cannot be precomputed). Therefore, Algorithm 4, which requires these differences to be in affine coordinates, cannot be used.

4.3 Security

Weil descent attacks [19], [20] have been shown to be effective for solving the discrete logarithm problem (DLP) in some elliptic curves defined over characteristic two finite fields of composite extension degrees. It has been shown in [12] that the probability that a randomly selected GLS curve defined over $\mathbb{F}_{2^{254}}$ is vulnerable to Weil descent attacks is negligibly small ($\approx 1/2^{99}$), and there is an efficient check that can be performed to rule out this possibility. This explicit check would take about 1300 days of CPU time on a 1GHz Sun V440, and it can be easily parallelized; see [12]. Index calculus attacks are also not effective for solving DLP in our above choices of elliptic curves; see the two recent papers [21], [22] and references therein. Hence, we can conclude that the curves that we are considering in this paper can be easily selected so that Pollard's rho method [23] is the fastest attack known for solving DLP, which has running time approximately $\sqrt{r}$.

5 Parallel Point Multiplication on Binary Elliptic Curves

Parallelism in point multiplication is an approach to reduce the number of field arithmetic operations, in the critical-path by using multiple arithmetic operators (mainly multipliers) concurrently [24]. It is important to eliminate the data dependencies to perform a careful scheduling of the field operations for the computation of point addition

Algorithm 3 Parallel computation of mixed differential PA and PD on generic curves employing three multipliers (Cost: $5\mathbf{M} + 6\mathbf{S} + 1\mathbf{D}$ [28]). Note that **M**, **S**, and **D**, are the costs of a field multiplication, a squaring, and a multiplication by curve parameter, respectively.

Input: $P_1 = (X_1, Z_1)$, $P_2 = (X_2, Z_2)$, and
$P_1 - P_2 = (x_0, y_0)$ in affine coordinates
Output: $P_1 + P_2 = (X_3, Z_3)$ and $2P_2 = (X_4, Z_4)$
Step 1: $T_1 \leftarrow X_1 \times Z_2$, $T_2 \leftarrow X_2 \times Z_1$, $T_3 \leftarrow X_2 \times Z_2$,
$T_4 \leftarrow (X_2)^4$, $T_5 \leftarrow (Z_2)^4$ $(1M)$
Step 2: $T_1 \leftarrow T_1 + T_2$ $(1A)$
Step 3: $T_1 \leftarrow T_1^2$, $T_3 \leftarrow T_3^2$ $(1A)$
Step 4: $Z_3 \leftarrow T_1$, $T_1 \leftarrow x_0 \times T_1$, $T_2 \leftarrow T_2 \times T_3$,
$Z_4 \leftarrow T_3$, $T_5 \leftarrow T_5 \times b$. $(1M)$
Step 5: $T_1 \leftarrow T_1 + T_2$, $T_5 \leftarrow T_5 + T_4$. $(1A)$
Step 6: $X_3 \leftarrow T_1$, $X_4 \leftarrow T_5$. $(1A)$
return X_3, Z_3, X_4, Z_4.

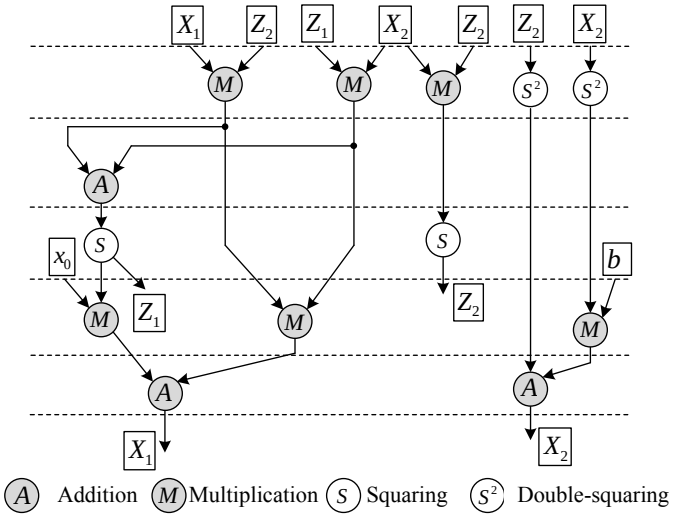


Figure 1. Data dependency graph for differential point addition and doubling on binary elliptic curves using three parallel multipliers [29].

(PA) and point doubling (PD) as lower level curve arithmetic computations. Parallel computation of combined PA and PD on binary generic curves has been recently investigated in [24], [25], [26], and [27]. Employing widely used Montgomery's ladder for point multiplication, it has been concluded that the fastest computation can be achieved by using three finite field multipliers on binary generic curves. The reason is that the data dependency does not allow to reduce the cost of point multiplication to less than two multiplications.

In Algorithms 3 and 4, scheduling of parallel computation of differential PA and PD on binary generic curves and GLS curves are illustrated, respectively. As we target fast and high performance applications, three parallel multipliers over $\mathbb{F}_{2^m}$ are employed to reduce the cost (latency) of combined PA and PD to two multiplications as shown in Figure 1. As one can see, the latency in terms of number of multipliers in the critical-path is $2M$, where M is the latency of a field multiplication over $\mathbb{F}_{2^m}$. In Algorithm 4, we employ differential PA and PD formula given in projective coordinates for GLS curves using four parallel multipliers [14]. As seen the latency is similar to the binary generic curves, i.e., $2M$. Also, the difference of two points is given in projective coordinates as we need to update

Algorithm 4 Parallel computation of projective differential PA and PD on binary generic curves employing four multipliers (Cost: $6\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ [14]). Note that **M**, **S**, and **D**, are the costs of a field multiplication, a squaring, and a multiplication by curve parameter, respectively.

Input: $P_1 = (X_1, Z_1)$, $P_2 = (X_2, Z_2)$, and
$P_1 - P_2 = (X_0, Z_0)$ in projective coordinates
Output: $P_1 + P_2 = (X_3, Z_3)$ and $2P_2 = (X_4, Z_4)$
Step 1: $T_1 \leftarrow X_1 + Z_1$, $T_2 \leftarrow X_2 + Z_2$ $(1A)$
Step 2: $T_2 \leftarrow T_1 \times T_2$, $T_3 \leftarrow X_1 \times X_2$,
$T_4 \leftarrow Z_1 \times Z_2$, $T_5 \leftarrow X_1 \times Z_1$ $(1M)$
Step 3: $T_3 \leftarrow T_3 + T_4$ $(1A)$
Step 4: $T_2 \leftarrow T_2 + T_3$ $(1A)$
Step 5: $T_2 \leftarrow T_2^2$, $T_3 \leftarrow T_3^2$, $T_1 \leftarrow T_1^2$ $(1S)$
Step 6: $Z_3 \leftarrow T_2 \times X_0$, $X_3 \leftarrow T_3 \times Z_0$,
$Z_4 \leftarrow T_5 \times a_1$, $X_4 \leftarrow T_1^2$. $(1M)$
return X_3, Z_3, X_4, Z_4.

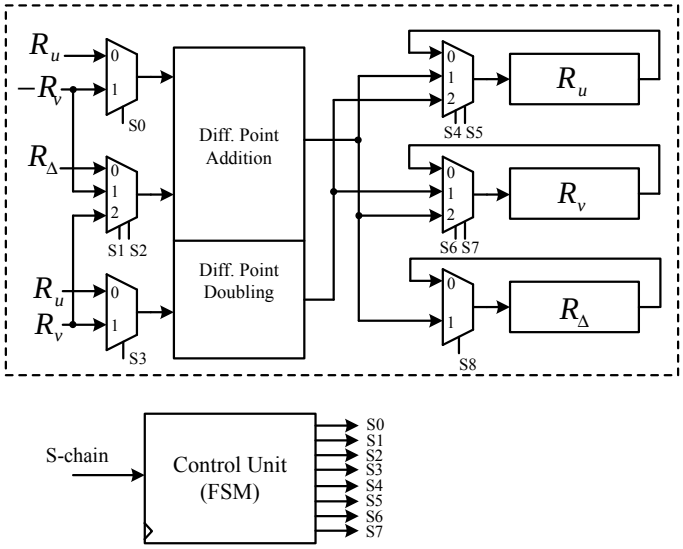


Figure 2. The simplified architecture of proposed point multiplication scheme using double point multiplication and efficient endomorphism.

them in each iteration based on the conditions given in our proposed scheme in Algorithm 2. As indicated in the previous section, the latency of point multiplication on GLS curves using double point multiplication considerably reduces the number of iterations in computing the main loop of point multiplication. Therefore, in the following section we describe the implement of point multiplication on GLS curves in details.

Table 5. Control signals generated by the simplified control unit based on the S-sequence

S-Seq.	Operation			S0	S1	S2	S3	S4	S5	S6	S7	S8
	R_u	R_v	$R_\triangle$									
Initialize	P	Q	$P - Q$	0	0	1	0	0	0	0	0	1
$R1$	$2R_u$	$R_u + R_v$	$R_\triangle$	0	1	0	0	1	0	1	0	0
$R2$	$2R_u$	R_v	$R_u + R_\triangle$	0	0	0	0	1	0	0	0	1
$R1'$	$R_u + R_v$	$2R_v$	$R_\triangle$	0	1	0	1	0	1	0	1	0
$R2'$	R_u	$2R_v$	$R_\triangle + (-R_v)$	1	0	0	1	0	0	0	1	1

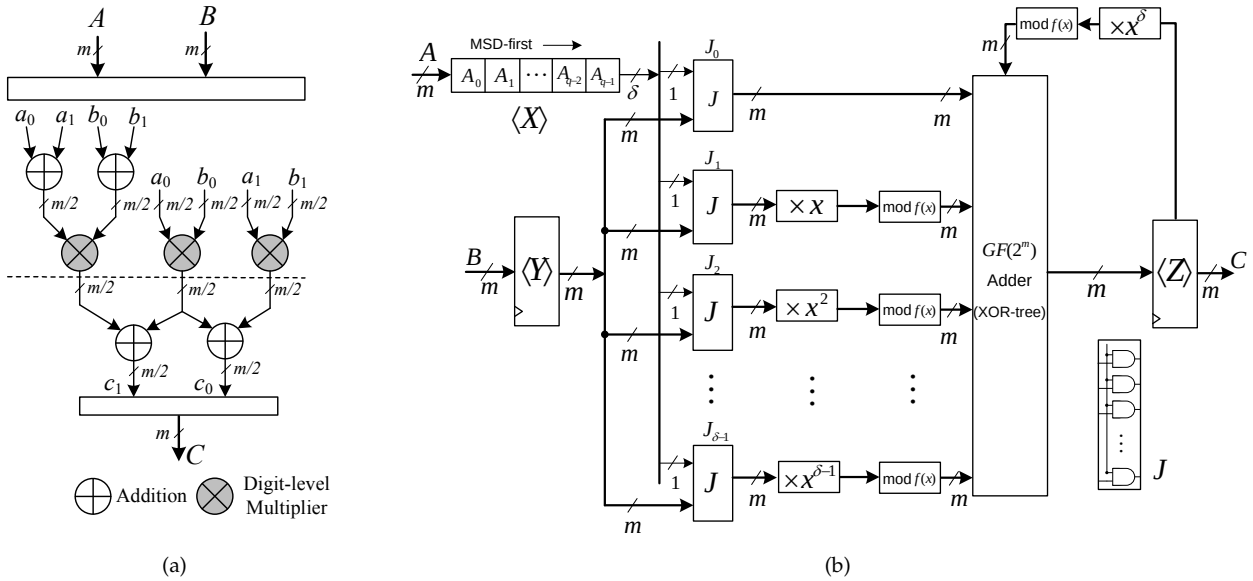


Figure 3. (a) Then architecture of $\mathbb{F}_{2^{2\ell}}$ Karatsuba multiplier (b) The architecture of employed digit-level SIPO polynomial basis multiplier for trinomial irreducible polynomials [30].

6 Proposed Architecture for Point Multiplication Using Double Point Multiplication

Based on the information provided in the previous section, in this section we propose a hardware architecture for computing point multiplication using double point multiplication. Based on the point multiplication scheme given in Algorithm 1, we design the proposed crypto-processor as depicted in Fig. 2. As one can see, this architecture implements the Algorithm 1 using the S-chain as the input of control unit. The proposed architecture is composed of an arithmetic unit, a control unit, and memory (register file) as one can has for the traditional ECC crypto-processors. The point addition and point doubling are composed of finite field arithmetic blocks to perform field operations including additions, multiplications, squarings, and inversion. In what follows we explain these blocks in details.

6.1 Arithmetic Unit

The arithmetic unit of the proposed architecture for point multiplication using double point multiplication (based on Algorithm 4 on GLS curves) is composed of two adders, two squarers, and four finite field multipliers as described in the following.

6.1.1 Addition and Squaring

Addition of two field elements, say, $A = \sum_{i=0}^{m-1} a_i \alpha^i = (a_{m-1}, \cdots, a_1, a_0)$ and $B = \sum_{i=0}^{m-1} b_i \alpha^i = (b_{m-1}, \cdots, b_1, b_0)$ in $\mathbb{F}_{2^m}$ represented by polynomial basis is $C = A + B$ and can be obtained by pair-wise addition of the coordinates of A and B over $\mathbb{F}_2$ (i.e., modulo 2 addition) as $c_i = a_i \oplus b_i$. Addition requires only one clock cycle to store the results in the registers.

For squaring an element $A \in \mathbb{F}_{2^m}$, we first simply insert zeros between each bit in the bit-vector representing A which must be followed by a reduction operation as $A^2 = \sum_{i=0}^{m-1} a_i \alpha^{2i} \bmod f(x)$. The reduction, mod $f(x)$ ($f(x)$ is a degree-m irreducible polynomial) is computed using XOR and shift operations only.

6.1.2 Multiplication

Finite field multipliers are available in bit-level (with area complexity of $O(m)$ and time complexity of $O(m)$), digit-level (with area complexity of $O(m\delta)$ and time complexity of $O(m/\delta)$), and bit-parallel (with area complexity of $O(m^2)$ for quadratic and $O(m^{log_2 3})$ for subquadratic with time complexity of $O(1)$) architectures depending on the available resources. To perform multiplication in the extension field $\mathbb{F}_{2^{2\ell}}$, we employ the Karatsuba technique which requires three multiplications and four additions over $\mathbb{F}_{2^\ell}$. In Fig. 3a, the architecture of the $\mathbb{F}_{2^{2\ell}}$ Karatsuba multiplier is depicted for more explanation. Therefore, first, we devise a digit-level polynomial basis multiplier for computing the product of two elements over $\mathbb{F}_{2^\ell}$ using a degree-ℓ polynomial $f(x)$ irreducible over $\mathbb{F}_{2^\ell}$, e.g., $\mathbb{F}_{2^{127}} : \mathbb{F}_2[x]/(x^{127} + x^{63} + 1)$. Then, we compute the multiplication over $\mathbb{F}_{2^{2\ell}}$ using Karatsuba's multiplier based on the architecture given in Fig. 3a. The critical-path through this multiplier includes two $\mathbb{F}_{2^\ell}$ additions and one $\mathbb{F}_{2^\ell}$ multiplication.

In [31], an efficient digit-level polynomial basis multiplier is proposed which offers a good trade-off between computation time and required area. We employ the MSD-first multiplier architecture proposed in [30] which is the most suitable one for FPGA implementations when the irreducible polynomial, $f(x)$ is a trinomial [32]. Note that we could have used Karatsuba method for multiplication over $\mathbb{F}_{2^\ell}$ as well, but as indicated in [33], for smaller field sizes digit-serial polynomial basis multipliers can operate in higher clock frequencies occupying same area in comparison to the Karatsuba multiplier.

In Fig. 3, the polynomial basis digit-level multiplier with serial-in parallel-out (SIPO) architecture is depicted. As one can see, in each clock cycle δ coefficients of the operand A are processed having all bits of operand B available through multiplication process. In this architecture, the J blocks perform bit-wise AND operation as $a_i \odot B$. The $\times x^i$ blocks perform corresponding shift operations and are only wiring. Once the δ partial products are computed at the output of J blocks, they are multiplied by x^i, $1 \leq i \leq \delta$ and then reduced using mod $f(x)$ blocks. The $\mathbb{F}_{2^m}$ adder block performs addition (XOR) over $\delta + 1$ m-bit field elements. Therefore, the critical-path delay of the $\mathbb{F}_{2^m}$ adder is $(\lceil \log_2(\delta + 1) \rceil) T_X$. For multiplier operation, first the registers $\langle Y \rangle$ and $\langle Z \rangle$ are preloaded with the operand B and zero ($0 \in \mathbb{F}_{2^m}$), respectively. The register $\langle X \rangle$ provides in each clock cycle d bits of operand A. Then, the results of the multiplication are available after $M_q = \lceil \frac{m}{\delta} \rceil$, $1 \leq \delta \leq m$ clock cycles in the register $\langle Z \rangle$. The main advantage of this multiplier is that it operates in higher clock frequencies in comparison to the counterparts available in the literature.

6.1.3 Inversion

Inversion is the most expensive operation and can be computed using the Extended Euclidean Algorithm (EEA) or Fermat's Little Theorem (FLT) [34]. Base on FLT, one can write $A^{2^m-2} = A^{-1}$ whose computation requires $m - 1$

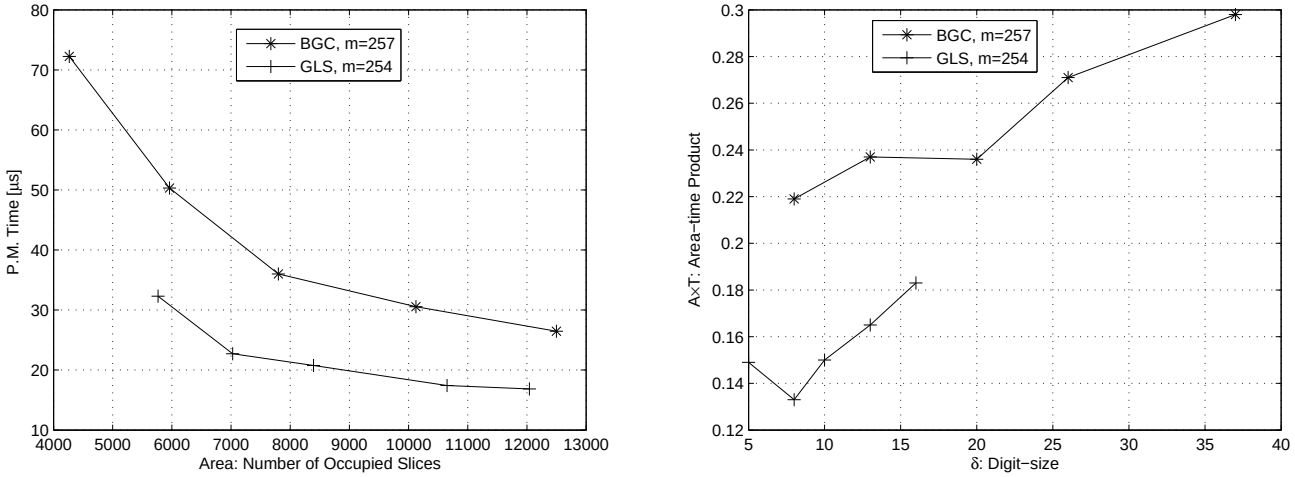
Table 6. Implementation results of point multiplication on binary generic and GLS curves on Xilinx Virtex-4 FPGA.

Point multiplication on a BGC over $\mathbb{F}_{2^{257}}$						
δ	M_q	Latency [# Clock Cycles]	f [MHz]	Area [# Slices]	P.M. Time [μs]	Area-Time [A$\times T$]
8	33	18,813	260.4	4,266	72.24	0.219
13	20	12,157	241.5	5,961	50.32	0.237
20	13	8,573	238.1	7,800	36.01	0.236
26	10	7,037	230.4	10,125	30.54	0.271
37	7	5,501	207.9	12,498	26.45	0.298
Point multiplication on a GLS curve over $\mathbb{F}_{2^{254}}$ using double point multiplication						
δ	M_q	Latency [# Clock Cycles]	f [MHz]	Area [# Slices]	P.M. Time [μs]	Area-Time [A$\times T$]
5	26	11,296	349.6	5,767	32.31	0.149
8	16	7,756	341.3	7,027	22.72	0.133
10	13	6,694	322.5	8,395	20.75	0.150
13	10	5,632	323.3	10,561	17.42	0.165
16	8	4,924	292.4	12,043	16.85	0.183

squarings and $m-2$ multiplications as $2^m-2=(11\cdots110)_2$. However, Itoh and Tsujii (IT) [35] proposed an efficient algorithm for computing inversion over $\mathbb{F}_{2^m}$. The IT scheme requires $\lfloor \log_2(m-1) \rfloor + HW(m-1)-1$ multiplications and $m-1$ squarings, where $HW(m-1)$ is the Hamming weight (number of ones) of the binary representation of $m-1$. Inversion of an element in $\mathbb{F}_{2^{2\ell}}$ can be performed by an inversion, a squaring, three multiplications, and two additions over $\mathbb{F}_{2^\ell}$. We employ the IT scheme [35] to perform inversions. Inversion over $\mathbb{F}_{2^{257}}$ costs $8M+256S$, where M and S are the costs of a multiplication and a squaring, respectively. For inversion over $\mathbb{F}_{2^{127}}$ the IT scheme dose not provide optimum number of multiplications and there is a shorter addition chain with only 9 multiplications. The shortest addition chain to compute $m-1=126$ has length 9 ($U=1\ 2\ 3\ 4\ 7\ 14\ 21\ 42\ 63\ 126$), so the complexity of the inversion over $\mathbb{F}_{2^{127}}$ is $9M+126S$. It is worth mentioning that the inversion can also be computed using EEA scheme in a slightly faster time and dedicating a specific hardware. EEA is used more in software implementations of inversion but supplementing an extra hardware is not efficient for our implementations. Note that our scheme based on IT method computes inversion by using available resources without adding any extra hardware cost.

6.2 Control Unit and Memory

The control unit is designed with a finite state machine (FSM) based on the point multiplication algorithm given in the previous sections. It schedules the computation tasks by generating the signals and switching the operands for arithmetic unit. The intermediate results are stored in a register file. The S-sequence is stored in the control unit. Therefore, based on the values of S-sequence appropriate signals are provided to select the arithmetic operations in the arithmetic unit to perform differential addition and doubling as shown in Fig. 2. First, the registers R_u and R_v are initialized with P and Q, respectively. Then, simplified control signals based on the S-sequence are generated as given in Table 5. We note that the control unit is simpler and requires smaller area than the other units in the data path. Since it is implemented as a FSM, it can easily mapped into the FPGA by the synthesis tools.



(a) (b)

Figure 4. Comparison of the point multiplication on BGCs and GLS curves over $\mathbb{F}_{2^{257}}$ and $\mathbb{F}_{2^{254}}$ in terms of (a) time of computation and (b) area-time products.

To store the input and output points, the S-sequence, and the intermediate results we employed a register file instead of using RAM blocks of the FPGA. This eliminates the overhead of communication between memory and arithmetic unit. Also, several multiplexers are employed to chose appropriate registers and connect to the arithmetic unit.

7 FPGA Implementation

In this section, we focus on the FPGA implementations of the proposed architectures in the previous sections. One of the advantages of hardware implementation is that one can achieve parallel computations and provide high speed results if multiple operations can be performed at the same time. As explained before, we have employed parallelism to increase the speed of point multiplication. The proposed architecture for point multiplication on binary generic and GLS curves are implemented on FPGA. The proposed architectures are modeled in VHDL and are synthesized using XST™ of Xilinx ISE™ version 12.1 design software and are implemented on Xilinx Virtex™-4 XC4VLX200 FPGA. In terms of available resources it contains 89,088 Slices (178,176 LUTs and 178,176 FFs) and 960 input/output (I/O) pins. Each slice contains 2 flip-flops (FFs) and 2 look-up tables (LUTs) [36].

7.1 Implementation Results

The implementation results of point multiplication over $\mathbb{F}_{2^{257}}$ for BGCs and $\mathbb{F}_{2^{254}}$ for GLS curves are reported in Table 6 for different digit sizes after place-and-route (PAR). We choose digit sizes from the sets $\delta \in \{8, 13, 20, 26, 37\}$ and $\delta \in \{5, 8, 10, 13, 16\}$ for BGCs and GLS curves, respectively. We choose smaller digit sizes for GLS curves as we need to employ four parallel multipliers to perform a point multiplication. As one can see, increasing the digit-size results in the reduction of the latency of the point multiplication, i.e., L_{Total}, at the cost of increase in the area and

decrease in the operating clock frequency. The point multiplication time is provided by dividing the total number of clock cycles (L_{Total}) by the maximum operating clock frequency (f_{max}).

Based on the scheduling scheme presented in Algorithm 3, the point multiplication on BGCs over $\mathbb{F}_{2^{257}}$ provides fastest computation time in 26.45 μs employing 12,498 Slices (with $\delta = 37$). As one can see in Table 6, for GLS curves the fastest time of point multiplication over $\mathbb{F}_{2^{254}}$ is achieved by choosing $\delta = 16$ which is 16.85 μs and occupies 12,043 Slices. As a result, point multiplication on GLS curves provides 37% faster computation time in comparison to the BGCs. To consider resource-constrained applications we investigate the area-time requirements of the point multiplication on both curves. For GLS curves, we observe that $\delta = 8$ provides the best results in terms of area-time trade-offs and a point multiplication is computed in 22.72 μs.

In Fig. 4a, we plot the point multiplication times for BGCs and GLS curves and compare them in terms of number of occupied slices. As illustrated, point multiplication on GLS curves is always faster than the BGCs. Moreover, we plot the area-time products in terms of digit-size δ in Fig. 4b. As one can see in this figure, GLS curves provides the best results in terms of time-area requirements in comparison to the BGCs. It is worth mentioning that the proposed scheme and architecture for point multiplication are platform independent and one can achieve high performance and faster results by implementing them on faster FPGA devices such as Virtex-5 and Virtex-6.

7.2 Comparison and Discussions

Previously proposed schemes for hardware implementation of point multiplication on binary elliptic curves, have considered finite field arithmetic optimizations such as parallelization and pipelining techniques and few works have considered curve level optimizations in the literature. For instance, one can refer to [24], [37], [38], [39], [26], [25], and [27] to name a few. Therefore, for comparison purposes we have implemented the point multiplication on binary elliptic curves using Montgomery's ladder. We note that our proposed technique for computing single point multiplication using double point multiplication yields 37% faster results in comparison to the traditional scheme of computing point multiplication. We further stress that we slightly gained on the speed of computing point multiplication due to employing Karatsuba's method for computing multiplication in the extension field and also using smaller field size which is only 7% of our improvements. It is worth mentioning that for the point multiplication on BGCs, if a multiplier over $GF(2^{257})$ is replaced by three 129×129-bit multipliers (similar to the one employed for GLS curves) the performance gain would be 7% for the timing results.

The implementation results are reported in Table 7 and are compared with the results for generic and Koblitz curves available in the literature. We note that because different field sizes and different FPGA technologies are used to implement different crypto-processors, meaningful quantitative comparisons of the area and time results are difficult. However, in comparison to the previous work over $\mathbb{F}_{2^{233}}$, the proposed scheme here provides faster computation results.

Table 7. Comparison of FPGA implementation results of proposed point multiplication architecture with the most recent works available in the literature over $GF(2^{233})$. Note that BGC: binary generic curve, BKC: binary Koblitz curve, GLS: Galbraith-Lin-Scott curve.

Work	FPGA Device	Curve	Field*	# of Clock Cycles	f [MHz]	Area	P.M. Time [μs]
[40]	Virtex-5*	BGC	$\mathbb{F}_{2^{233}}$	3825	192.3	6,487 Slices	19.89
[41]	Virtex-4	BKC	$\mathbb{F}_{2^{233}}$	5890	190.0	7,130 Slices	31.00
[42]	Virtex-4	BGC	$\mathbb{F}_{2^{233}}$	–	60.0	19,647 Slices	31.00
This work	Virtex-4	GLS	$\mathbb{F}_{2^{254}}$	4,924	292.4	12,043 Slices	16.85

*. Note that different field sizes and different FPGA technologies are used to implement point multiplication in the previous work available in the literature.

8 Conclusion

In this paper, high-performance implementation results of point multiplication on binary elliptic curves are presented. We have proposed a new scheme to compute double point multiplication employing differential addition chains. Moreover, we have demonstrated how double point multiplication can be employed to speed up the computation of single point multiplication on elliptic curves with endomorphisms. Finally, we have implemented single point multiplication using our proposed scheme on FPGA. We have compared the implementation results in terms of time and area with the traditional methods of computing point multiplication. To the best of our knowledge, our implementation results are about 30% faster than the fastest results of point multiplication on binary generic curves.

The approach proposed in this paper to design a point multiplication based on double point multiplication using efficiently computable endomorphisms offers several further research topics to be investigated. For instance, it would be interesting to apply the proposed technique on binary Edwards curves [43] as they offer unified and complete point addition formulation but they do not provide fast results in comparison to the binary generic curves. Another interesting research topic is to employ the proposed scheme for computing double point multiplication in cryptographic applications such as Schnorr- and ElGamal-like digital signature schemes, and compare the resulting schemes' performance with the performance of traditionally implemented schemes.

9 Acknowledgment

The authors of the paper would like to thank David Jao for his valuable comments on an early draft of this paper. The work by Reza Azarderakhsh has been supported by NSERC CRD Grant CRDPJ 405857-10. A part of this work was completed when Koray Karabina was working in the Department of Combinatorics and Optimization at the University of Waterloo.

References

[1] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *LNCS 218 as Proceedings of Crypto '85*. Springer Verlag, 1986, pp. 417–426.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[3] "National Institute of Standards and Technology," *Digital Signature Standard*, 186-2, January 2000.

[4] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, New York, 1996.

[5] B. Möller, "Algorithms for Multi-exponentiation," *Selected Areas in Computer Science SAC 2001, Lecture Notes in Computer Science*, vol. 2259, pp. 165–180, 2001.

[6] P. Montgomery, "Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains," December 13, 1983; Revised March, 1991 and January, 1992, www.cwi.nl/ftp/pmontgom/Lucas.ps.gz.

[7] T. Akishita, "Fast Simultaneous Scalar Multiplication on Elliptic Curve with Montgomery Form," *Selected Areas in Computer Science SAC 2001, Lecture Notes in Computer Science*, vol. 2259, pp. 225–267, 2001.

[8] M. Stam, "Speeding up subgroup cryptosystems," Ph.D. dissertation, Technische Universiteit Eindhoven, 2003.

[9] D. Bernstein, "Differential addition chains," Tech. Rep., 2006, available at http://cr.yp.to/ecdh/diffchain-20060219.pdf.

[10] R. Gallant, R. Lambert, and S. Vanstone, "Faster point multiplication on elliptic curves with efficient endomorphisms," *Advances in Cryptology - CRYPTO 2011, Lecture Notes in Computer Science*, vol. 2139, pp. 190–200, 2001.

[11] D. Galbraith, X. Lin, and M. Scott, "Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves," *Journal of Cryptology*, vol. 24, pp. 446–469, 2011.

[12] D. Hankerson, K. Karabina, and A. Menezes, "Analyzing the Galbraith-Lin-Scott point multiplication method for elliptic curves over binary fields," *IEEE Transactions on Computers*, vol. 58, pp. 1411–1420, 2009.

[13] R. Dahab, D. Hankerson, F. Hu, M. Long, and M. Lopez, "Software Multiplication Using Gaussian Normal Bases," *IEEE Transactions on Computers*, pp. 974–984, 2006.

[14] M. Stam, "On Montgomery-Like Representations for Elliptic Curves over $GF(2^k)$," *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography*, pp. 240–253.

[15] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of computation*, pp. 243–264, 1987.

[16] D. Knuth, *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[17] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. Cambridge University Press, NY, USA, 1986.

[18] IEEE Std 1363-2000, "IEEE Standard Specifications for Public-Key Cryptography," January 2000.

[19] P. Gaudry, F. Hess, and N. Smart, "Constructive and destructive facets of Weil descent on elliptitc curves," *Journal of Cryptology*, vol. 15, pp. 19–46, 2002.

[20] F. Hess, "Generalizing the GHS attack on the elliptic curve discrete logarithm problem," *Journal of Computation and Mathematics*, vol. 7, pp. 167–192, 2004.

[21] J. Faugere, L. Perret, C. Petit, and G. Renault, "Improving the complexity of index calculus algorithms in elliptic curves over binary fields," *Accepted at EUROCRYPT2012*, 2012.

[22] C. Petit and J. Quisquater, "On polynomial systems arising from a weil descent," *Cryptology ePrint Archive, Report 2012/146*, 2012, http://eprint.iacr.org/.

[23] J. Pollard, "Monte Carlo Methods for Index Computation (mod p)," *Mathematics of computation*, vol. 32, no. 143, pp. 918–924, 1978.

[24] R. Cheung, N. Telle, W. Luk, and P. Cheung, "Customizable Elliptic Curve Cryptosystems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 9, pp. 1048–1059, 2005.

[25] C. H. Kim, S. Kwon, and C. P. Hong, "FPGA Implementation of High Performance Elliptic Curve Cryptographic Processor over $GF(2^{163})$," *Journal of System Architcture*, vol. 54, no. 10, pp. 893–900, 2008.

[26] O. Ahmadi, D. Hankerson, and F. Rodríguez-Henríquez, "Parallel Formulations of Scalar Multiplication on Koblitz Curves," *Journal of Univers. Computing Sci.*, vol. 14, no. 3, pp. 481–504, 2008.

[27] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA Implementation of Point Multiplication on Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1453–1466, 2012.

[28] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation," *Cryptographic Hardware and Embedded Systems: First International Workshop, CHES'99, Worcester, MA, USA, August 1999: Proceedings*, 1999.

[29] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves Over $GF(2^m)$ Without Precomputation," in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999)*, 1999, pp. 316–327.

[30] C. Shu, S. Kwon, and K. Gaj, "Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1221–1237, 2009.

[31] L. Song and K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *The Journal of VLSI Signal Processing*, vol. 19, no. 2, pp. 149–166, 1998.

[32] J.-L. Beuchat, T. Miyoshi, Y. Oyama, and E. Okamoto, "Multiplication over F_{pm} on FPGA: A Survey," in *Third International Workshop of Reconfigurable Computing: Architectures, Tools and Applications (ARC 2007)*, vol. 4419, 2007, pp. 214–225.

[33] M. Morales-Sandoval, C. F. Uribe, R. Cumplido, and I. Algredo-Badillo, "An area/performance trade-off analysis of a GF(2^m) multiplier architecture for elliptic curve cryptography," *Computers & Electrical Engineering*, vol. 35, no. 1, pp. 54–58, 2009.

[34] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited," *IEEE Transactions on Computers*, pp. 1047–1059, 2004.

[35] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases," *Information Computing*, vol. 78, no. 3, pp. 171–177, 1988.

[36] Xilinx, "Xilinx Virtex-4 device data sheet," *www.xilinx.com/support/documentation/virtex-4.htm*, vol. ver5.0, Febraury 2011.

[37] W. N. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*, vol. 16, no. 2, pp. 198–205, 2008.

[38] V. S. Dimitrov, K. U. Järvinen, M. J. Jacobson, W. F. Chan, and Z. Huang, "Provably Sublinear Point Multiplication on Koblitz Curves and its Hardware Implementation," *IEEE Transaction on Computers*, vol. 57, no. 11, pp. 1469–1481, 2008.

[39] R. Azarderakhsh and A. Reyhani-Masoleh, "A Low Complexity Hybrid Architecture for Double-MUltiplication Using Gaussian Normal Basis," *IEEE Transactions on Computers*,, 2011.

[40] G. Sutter, J. Deschamps, and J. Imana, "Efficient Elliptic Curve Point Multiplication using Digit Serial Binary Field Operations," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, 2013.

[41] B. Ansari and M. Hasan, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1443–1453, 2008.

[42] C. Rebeiro and D. Mukhopadhyay, "High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms," in *9th International Conference on Cryptology in India (INDOCRYPT 2008)*, ser. Lecture Notes in Computer Science, D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365. Springer, 2008, pp. 376–388.

[43] D. J. Bernstein, T. Lange, and R. R. Farashahi, "Binary Edwards Curves," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2008, pp. 244–265.