A New Double Point Multiplication Algorithm and Its Application to Binary Elliptic Curves with Endomorphisms

Reza Azarderakhsh and Koray Karabina

Abstract—We present a new double point multiplication algorithm based on differential addition chains. Our proposed scheme has a uniform structure and has some degree of built-in resistance against side channel analysis attacks. We discuss deploying our scheme in a hardware implementation of single point multiplication on binary elliptic curves with efficiently computable endomorphisms. Based on operation counts, we expect to gain accelerations of 30% and 18% for computing single point multiplication with and without availability of parallel multipliers, respectively, and these results are verified in our implementations.

Index Terms—Elliptic curve cryptosystems, endomorphism, differential addition chains, double point multiplication

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. The principal operation in elliptic curve cryptographic systems is point multiplication. Several effort in the literature have focused on developing efficient techniques to compute point multiplication on various forms of elliptic curves.

Let elliptic curve E, 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 positive 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 as a naive method. 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 [3]) and interleaving [4]. Other alternatives for computing simultaneous double point multiplication are based on differential addition chains; see for instance [5]–[8]. 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. For instance, Montgomery's continued fraction algorithm CFRC [5] has a uniform pattern whereas its optimized version PRAC [5] 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 $k = k_1 + k_2\lambda \bmod r$, 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) [9] 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 [10] generalized this technique to larger classes of curves (known as the GLS curves). Recently, Hankerson, Karabina and Menezes [11] 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 [11], 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 [12] and [13]). These formulas are generalization of Montgomery's formulas [14] 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 [13] and Bernstein [8]. In [13], Stam adapted Montgomery's PRAC algorithm [5] and proposed a double point multiplication algorithm in elliptic curves over fields of characteristic two. Stam's method costs 1.5 additions and 0.5 doublings per scalar bit and outperforms the previous results available in the literature. However, the proposed algorithm in [13] does not have a uniform structure, and it is concluded in [13] that protection against timing and power analysis attacks has not been supported. More recently, Bernstein [8] proposed two double point multiplication algorithms based on new differential addition-subtraction chains. The first algorithm in [8] has a uniform structure and costs two additions and one doubling per scalar bit. Bernstein's second algorithm in [8] 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 [8].

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 the previous work. In Section 3, we employ double point multiplication to compute single point multiplication on elliptic curves with efficiently computable endomorphisms. In

- *R. Azarderakhsh is with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, New York. E-mail: rxaeec@rit.edu.*
- *K. Karabina is with the Department of Mathematical Sciences at Florida Atlantic University, Boca Raton, FL, 33431. E-mail: kkarabina@fau.edu.*

Manuscript received 27 Dec. 2012; revised 12 Apr. 2013; accepted 24 Apr. 2013. Date of publication 05 May 2013; date of current version 12 Sep. 2014. Recommended for acceptance by J. Beuchat.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.112

TABLE 1
Update Rules for Double Point Multiplication

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

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

Section 4, we discuss the implementation of the proposed scheme for single point multiplication on binary elliptic curves. We conclude the paper in Section 5.

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 [5].

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

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 $aP + bQ = d(R_u + R_v) = R_u + R_v$.

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 $d(R_u + R_v)$

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.

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.

The main issue to determine the efficiency of Algorithm 1 is to estimate the number of iterations, say $L(a, b)$, 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 $L(a, b) = 2\ m$.

In our experiments we observed that $L(a, b)$ 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 1 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 3.2). Our experimental evidence

TABLE 3
Practical Behavior of Algorithm 1 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 6	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

suggests that, on average, $L(a, b) = 1.4 \log_2 a$ and $d = 6$ in step 6 of Algorithm 1. Moreover, each R1, R2, R1′, R2′ is used in about 25% of the total number of iterations. The variance is very high for the size of d in step 6 of Algorithm 1 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, $aP + bQ$ can on average be computed in about 1.4ℓ additions and 1.4ℓ doublings in G.*

Remark 1. Algorithm 1 is basically performing the binary Euclidean algorithm during the execution of its while loop. 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 [15]. The number of subtractions in the binary Euclidean algorithm corresponds to the number of executions of the rules R1 and R1′ in Algorithm 1. 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 6 of Algorithm 1 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 in step 6 of Algorithm 1. For example, a legitimate user with her secret key k may compose $k = k_1 + k_2\lambda \bmod r$ as described in Section 3, and use Algorithm 1 to compute $kP = aP + bQ$, where $a = k_1$, $b = k_2$, and $Q = \lambda P$. Then the value of d in step 6 of Algorithm 1 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 [15]. 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 3) from the distribution of the randomly chosen pair of integers (a, b). Under this heuristic, we expect that $d = 1$ in step 6 of Algorithm 1 with probability at least $p(1) \approx 0.6$. Similarly, we can argue that $d > 3$ in step 6 of Algorithm 1 with probability at most $1 - (p(1) + p(2) + p(3)) \approx 0.17$; $d > 7$ with probability at most $1 - (p(1) + \cdots + p(7)) \approx 0.08$, and so on. In order to lower the attacker's success probability, one needs to 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 1, compute $kP = (k + c)P - cP$. Note that for a set S of size $|S| \geq 2$ we expect that $d_{\min} = 1$.

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 1 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 proposed algorithm which make it attractive for practical implementations.

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 discuss in the next section, double point multiplication can be employed 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.

We note that our proposed algorithm is an adaptation of Montgomery's PRAC algorithm [5] which was originally proposed to compute the n-th term of a second-degree recursive sequence. Montgomery's PRAC algorithm was previously adapted by Stam [13] to obtain a double point multiplication algorithm which on average requires 1.5 additions and 0.5 doublings per scalar bit[2], 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 to have protection against side channel analysis attacks which was also noted in [13]. More recently, Bernstein [8] 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 gcd 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

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.

2. The per-bit cost is reported to be 1.49A+0.33D in [7], Conjecture 3.29].

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, and D, Respectively

Algorithm	Per-bit cost	Uniform
Schoenmakers [7, Section 3.2.3]	$2.25A + 1.25D$	No
Akishita [6]	$2.25A + 0.75D$	No
Stam [13]	$1.5A + 0.5D$	No
New binary [8]	$2A + 1D$	Yes
New extended gcd [8]	$1.43A/S + 0.347D$	No
Algorithm 1	$1.4A + 1.4D$	Yes

multiplication algorithms proposed by Akishita [6], and Schoenmakers (see [7], Section 3.2.3]). Even though there are many other techniques that yield double point multiplication algorithms such as the interleaving technique [4], 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 its execution.

3 Point Multiplication on Elliptic Curves with Endomorphisms

Based on the information provided in the previous section, here we propose a new scheme to compute single point multiplication on elliptic curves with endomorphism.

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 [11] 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 $s \in \mathbb{F}_{q^4} \backslash \mathbb{F}_{q^2}$ 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 $\lambda^2 + 1 \equiv 0 (\mathrm{mod} r)$. It can be shown that $\lambda = t^{-1}(q-1) (\mathrm{mod} r)$. More interestingly, if k is an integer, then one can efficiently find two integers k_1 and k_2 such that

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

as follows [10]: 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., $r \approx q^2$), then for a $\log_2(r)$-bit integer k, one would expect that the bitlengths of k_1 and k_2 are half that of k.

Hence, Algorithm 1 described in Section 2 can be used to simultaneously compute $k_1P + k_2Q$ given k_1, k_2, P and Q.

3.1 Our Choice of Elliptic Curves

In this section, we briefly discuss the elliptic curves that can be employed in the crypto-processors to compute point multiplication.

3.1.1 A Binary Generic Curve

Let $n = 257, q = 2^n, \mathbb{F}_q = \mathbb{F}_2[x]/(x^{257} + x^{41} + 1)$. We choose $b \in \mathbb{F}_q$, and set

$$E/\mathbb{F}_q : Y^2 + XY = X^3 + X^2 + b, \quad (1)$$

where $\#E(\mathbb{F}_q) = (q + 1 - t) = 2r$, and r is a 256-bit prime.

3.1.2 A GLS Curve

Let $\ell = 127, q = 2^\ell, \mathbb{F}_q = \mathbb{F}_2[x]/(x^{127} + x^{63} + 1), \mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2 + u + 1)$. We choose $b \in \mathbb{F}_q$, and set

$$E'/\mathbb{F}_{q^2} : Y^2 + XY = X^3 + uX^2 + b, \quad (2)$$

where $\#E'(\mathbb{F}_{q^2}) = (q-1)^2 + t^2 = 2r$, and r is a 253-bit prime. The endomorphism is defined by $\Psi : E' \to E', (x, y) \mapsto (x^q, y^q + (u+1)x^q)$.

Remark 2. From our discussion in Section 3, an integer k can be decomposed as $k = k_1 + k_2\lambda \bmod r$ with $\log_2 |k_1|, \log_2 |k_2| \leq 127$. Our experiments with 10^6 random choices of $k \in [1, r)$, where $\log_2 k = 252$ on average, indicate that the decomposition method specified in Section 3 yields on average $\log_2 |k_1| = 125, \log_2 |k_2| = 124.99$. Moreover, if $a = |k_1|$ and $b = |k_2|$ are given as input to Algorithm 1, then the average length of the while loop is $L(a, b) = 175.40$. Note that these results are in agreement with our findings in Table 3 as $176/125 \approx 1.4$. As a result, the following conjecture can be stated.

Conjecture 2. *Let $E'/\mathbb{F}_{q^2}$ be the elliptic curve as in (2) with $P, Q \in E'(\mathbb{F}_{q^2})[r]$, and $a, b \in \mathbb{Z}$. Using Algorithm 1, $aP + bQ$ 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

$$E''/\mathbb{F}_{q^2} : Y^2 + a_1XY = X^3 + a_1^2uX^2 + 1/a_1^2, \quad (3)$$

that is isomorphic to E' over $\mathbb{F}_{q^2}$. The isomorphism and its inverse are given by $\Phi : E' \to E'', (x, y) \mapsto (a_1^2x, a_1^3y)$ and $\Phi^{-1} : E'' \to E', (x, y) \mapsto (a_1^{-2}x, a_1^{-3}y)$, respectively. The curve equation (3) is constructed in [13], and is better suited than (2) for implementing Algorithm 1; see [13]. Deploying Stam's differential addition and doubling formulae for E'' (see Algorithm 3) results to have the differences of two points given in projective coordinates. Contrarily to a Montgomery ladder, point differences in our proposed double point multiplication algorithm are not fixed (i.e., cannot be precomputed). Therefore,

Algorithm 2, which requires these differences to be in affine coordinates, cannot be used.

3.2 Security

Weil descent attacks [16], [17] 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 [11] 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 a polynomial time check that can be performed to rule out this possibility. This explicit check would take about 1300 days of CPU time on a 1 GHz Sun V440, and it can be easily parallelized; see [11]. Index calculus attacks are also not effective for solving DLP in our above choices of elliptic curves; see the two recent papers [18], [19] and references therein. Hence, we conclude that the curves that we are considering in this paper can be selected so that Pollard's rho method [20] is the fastest attack known for solving DLP, which has running time approximately $\sqrt{r}$.

Algorithm 2 Parallel computation of mixed differential PA and PD on generic curves employing three multipliers (Cost: $6\mathbf{M} + 6\mathbf{S}$ [21]). Note that $\mathbf{M}$, and $\mathbf{S}$ are the costs of a field multiplication and a squaring, 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.

Algorithm 3 Parallel computation of projective differential PA and PD on binary generic curves employing four multipliers (Cost: $6\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ [13]). Note that $\mathbf{M}, \mathbf{S}$, and $\mathbf{D}$, 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.

4 Computation of Single Point Multiplication Using Our Proposed Scheme

Based on the information provided in the previous sections, we investigate the efficiency of the proposed scheme for computing single point multiplication over binary elliptic curves. The cost of computing single point multiplication using mixed differential point addition and doubling [21] and Montgomery's ladder is $6\mathbf{M} + 6\mathbf{S}$, where $\mathbf{M}$ and $\mathbf{S}$ are the costs of a field multiplication and a squaring, respectively. Also, the cost of single point multiplication over GLS curves based on using projective differential point addition and doubling is $7\mathbf{M} + 4\mathbf{S}$ [13].

As indicated in Section 3, the latency of point multiplication on GLS curves using double point multiplication reduces the number of iterations in computing the main loop of point multiplication. In the main loop, finite field multiplications play an important role in determining the efficiency of an elliptic curve based crypto-processor. Note that addition is a simple bit-wise XOR operation and squaring can be implemented very efficiently. Therefore, we investigate the performance of our proposed scheme for single point multiplication in terms of the number of finite field multiplications and consider two scenarios in terms of availability of parallel multipliers.

4.1 Cost of Single Point Multiplication with One Multiplier

In this subsection, we assume only one finite field multiplier is available for the computation of single point multiplication. Therefore, the latency of computing a single point multiplication based on mixed differential point addition and doubling algorithm given in [21] using Montgomery's ladder is $6 \times 2k = 12k$, where k is the security level in bits. On the other hand, the latency of computing of a single point multiplication based our proposed scheme and projective differential addition and doubling formulas and the algorithm presented in [13] is $1.4 \times 7 \times 2k/2 = 9.8k$. Therefore, applying our proposed scheme reduces the latency of single point multiplication about 18% in comparison to the traditional scheme whenever we have only one field multiplier available. Note that this acceleration is independent of the choice of the security level k.

4.2 Cost of Single Point Multiplication with Several Parallel Multipliers

In this subsection, we assume that we have several field multipliers available and the target is high-performance applications. In Algorithms 2 ([21]) and 3 ([13]), scheduling of parallel computation of differential PA and PD on binary generic curves and GLS curves are illustrated, respectively. We can employ three and four parallel multipliers as illustrated in Algorithms 2 and 3, respectively, to reduce the latency of one differential point addition and doubling to only two multiplications. The latencies of single point multiplications for binary generic curves using Montgomery's ladder and our proposed scheme are $2k \times 2 = 4k$ and $(2k/2) \times 1.4 \times 2 = 2.8k$, respectively. Therefore, our proposed scheme reduces the latency of single point multiplication on binary curves about 30% in comparison to the traditional scheme.

To be more concrete, let us consider the implementation of single point multiplication using the binary generic curve $E(\mathbb{F}_{2^{257}})$ given by (1), and the GLS curve $E''(\mathbb{F}_{2^{254}})$ given by (3). The total latency in terms of the number of multiplications in the case of using E is $(256 \times 2)M_{257} = 512M_{257}$, where M_{257} is the latency of a field multiplication over $\mathbb{F}_{2^{257}}$. The total latency in terms of the number of multiplications in the case of using E'' can be estimated as $(176 \times 2)M_{254} = 352M_{254}$ (see Conjecture 2), where M_{254} is the

latency of a field multiplication over $\mathbb{F}_{2^{254}}$. Assuming that M_{257} and M_{254} are comparable, our technique provides an acceleration of $(\frac{512-352}{512}) \times 100 = 31\%$, which has been verified in our implementations. It is worth mentioning that in hardware, finite field multipliers can be implemented in the form of digit-level, bit-parallel, or Karatsuba based multipliers with some time-area trade-offs. In this paper, we do not consider such low level arithmetic computations since we focus on the curve level optimizations and investigate the efficiency of our proposed algorithm only.

The proposed scheme is suitable for high-performance applications and one can easily map its architecture (described in any hardware design language) to hardware (FPGA or ASIC) and achieve required performance. For brevity, we do not include the details of the hardware implementations in this paper.

5 Conclusion

We have proposed a new algorithm 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 efficiently computable endomorphisms. We have achieved accelerations of 30% and 18% for single point multiplication with and without availability of parallel multipliers. It would be interesting to apply the proposed technique on binary Edwards curves [22] as they offer unified and complete point addition formulas but they do not provide fast results in comparison to the binary generic curves. It would also be interesting to compare our algorithm to simultaneous multi-exponentiation algorithms, where the exponents are represented using 2^w-ary digits, and they are recoded to protect against side-channel attacks; see [23].

Acknowledgment

The authors would like to thank D. Jao for his valuable comments on an early draft of this paper. The authors also thank the referees for their detailed comments that improved the presentation of our paper. The work by Reza Azarderakhsh has been supported by NSERC CRD Grant CRDPJ 405857-10. A part of this work had been completed when Koray Karabina was working in the Department of Combinatorics and Optimization at the University of Waterloo. A part of this work has been completed when R. Azarderakhsh and K. Karabina was working in the Department of Combinatorics and Optimization at the University of Waterloo. Also, R. Azarderakhsh would like to thank H. Bahloul-Azarderakhsh for his kind support (Dualar).

References

[1] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Adv. Cryptol. (CRYPTO)*, 1986, pp. 417–426.
[2] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, pp. 203–209, 1987.
[3] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. New York, NY, CRC Press, 1996.
[4] B. Möller, "Algorithms for multi-exponentiation," in *Proc. 8th Annu. Int. Workshop Sel. Areas Comput. Sci. (SAC)*, 2001, vol. 2259, pp. 165–180.
[5] P. Montgomery. (Jan. 1992). *Evaluating Recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas Chains* [Online]. Available: www.cwi.nl/ftp/pmontgom/Lucas.ps.gz
[6] T. Akishita, "Fast simultaneous scalar multiplication on elliptic curve with montgomery form," in *Proc. 8th Annu. Int. Workshop Sel. Areas Comput. Sci.*, 2001, vol. 2259, pp. 225–267.
[7] M. Stam, "Speeding up subgroup cryptosystems," Ph.D. dissertation, Dept. Math., Tech. Univ. Eindhoven, Eindhoven, The Netherlands, 2003.
[8] D. Bernstein, "Differential addition chains," Tech. Rep., 2006, http://cr.yp.to/ecdh/diffchain-20060219.pdf
[9] R. Gallant, R. Lambert, and S. Vanstone, "Faster point multiplication on elliptic curves with efficient endomorphisms," in *Proc. 21st Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, 2001, vol. 2139, pp. 190–200.
[10] D. Galbraith, X. Lin, and M. Scott, "Endomorphisms for faster elliptic curve cryptography on a large class of curves," *J. Cryptol.*, vol. 24, pp. 446–469, 2011.
[11] D. Hankerson, K. Karabina, and A. Menezes, "Analyzing the Galbraith-Lin-Scott point multiplication method for elliptic curves over binary fields," *IEEE Trans. Comput.*, vol. 58, no. 10, pp. 1411–1420, Oct. 2009.
[12] R. Dahab, D. Hankerson, F. Hu, M. Long, and M. Lopez, "Software multiplication using Gaussian normal bases," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 974–984, Aug. 2006.
[13] M. Stam, "On Montgomery-like representations for elliptic curves over $GF(2^k)$," in *Proc. 6th Int. Workshop Pract. Theory Public Key Cryptogr. (PKC)*, 2003, pp. 240–253.
[14] P. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, pp. 243–264, 1987.
[15] D. Knuth, "The art of computer programming," *Seminumerical Algorithms*. vol. 2, 3rd ed. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1997.
[16] P. Gaudry, F. Hess, and N. Smart, "Constructive and destructive facets of Weil descent on elliptitc curves," *J. Cryptol.*, vol. 15, pp. 19–46, 2002.
[17] F. Hess, "Generalizing the GHS attack on the elliptic curve discrete logarithm problem," *J. Comput. Math.*, vol. 7, pp. 167–192, 2004.
[18] J. Faugere, L. Perret, C. Petit, and G. Renault, "Improving the complexity of index calculus algorithms in elliptic curves over binary fields," in *Proc. 31st Annu. Int. Conf. Theory Appl. Cryptogr. Tech. (EUROCRYPT)*, 2012, pp. 27–44.
[19] C. Petit and J. Quisquater, "On polynomial systems arising from a Weil descent," *Cryptology ePrint Archive, Report 2012/146*, 2012, http://eprint.iacr.org/
[20] J. Pollard, "Monte Carlo methods for index computation (mod p)," *Math. Comput.*, vol. 32, no. 143, pp. 918–924, 1978.
[21] J. López and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," in *Proc. Workshop Cryptogr. Hardware Embedded Syst. (CHES)*, 1999, pp. 316–327.
[22] D. J. Bernstein, T. Lange, and R. R. Farashahi, "Binary Edwards curves," in *Proc. Workshop Cryptogr. Hardware. Embedded Syst. (CHES)*, 2008, pp. 244–265.
[23] M. Joye and M. Tunstall, "Exponent recoding and regular exponentiation algorithms," *Proc. 2nd Int. Conf. Theory Appl. Cryptol. (AFRICACRYPT)*, 2009, vol. 5580, pp. 334–349.