Unspecified Journal
Volume 00, Number 0, Pages 000–000
S ????-????(XX)0000-0

# FURTHER REFINEMENT OF PAIRING COMPUTATION BASED ON MILLER'S ALGORITHM

CHAO-LIANG LIU, GWOBOA HORNG, AND TE-YU CHEN

Abstract. In 2006, Blake, Murty and Xu proposed three refinements to Miller's algorithm for computing Weil/Tate Pairings. In this paper we extend their work and propose a generalized algorithm, which integrates their first two algorithms. Our approach is to pre-organize the binary representation of the involved integer to the best cases of Blake's algorithms. Further, our refinement is more suitable for Solinas numbers than theirs. We analyze our algorithm and show that our refinement can perform better than the original algorithms.

## 1. Introduction

The Weil/Tate pairing is a mapping with nondegenerate and bilinear properties, which will map a special pair of points on elliptic curves to a certain multiplicative subgroup of a finite field. In 1993, Menezes, Okamoto and Vanstone [9] found that the Weil pairing could be applied to reduce the elliptic curves discrete logarithm problem on supersingular elliptic curves into a discrete logarithm problem of the multiplicative subgroup of a finite field. Their result shows that supersingular elliptic curves are unsuitable for many cryptographic schemes.

Since then, The Weil/Tate pairing exactly provide a constructive tool for cryptography. Indeed, many cryptographic applications based on pairings have been proposed, such as identity-based encryption system [2], digital signature [1, 3, 13], signcryption [8, 12], key agreement [7, 15], and so on. As a result, the application of pairings plays an important role in modern cryptography. Therefore, the computation of pairings is a critical issue for those applications based on pairings. The first efficient algorithm for computing pairing was proposed by Miller in 1986 [11]. The main idea of Miller's algorithm is to use lines to integrate the divisors, which the algorithm has processed (see section 2, for details). Many researches are directed in many different aspects in order to enhance the efficiency of the computation [5, 6].

In 2006 Blake, Murty and Xu [4] proposed a brand new concept based on the conjugate of a line, to reduce the total number of lines in Miller's algorithm. Though this concept does not dramatically decrease the cost of points adding, but it is novel and can be applied to decrease the number of field multiplications. They proposed three different algorithms for three cases: when there are relatively more

2000 *Mathematics Subject Classification.* Primary .

*Key words and phrases.* Algorithm, Elliptic curve, Cryptography, Weil pairing, Tate pairing, Miller's algorithm.

This research was supported by NSC94-2213-E-005-028.

©0000 (copyright holder)

zero bits (or average cases) of the integer $n$ (see section 2, for details), when there are relatively more one bits, and when the characteristic of the field is three.

In this paper, we continue their work and suggest a generalized algorithm, which can reduce more lines than the first two algorithms in average cases. Even in the extreme cases, our algorithm still performs as well as the best one of Blake's algorithms. In our algorithm, we use the same technique as Blake et al., but we consider the bits globally. We divide the binary representation integer $n$, which is a constant in every pairing base cryptosystem, into fragments. In accordance with these fragments, we design an algorithm to further reduce lines.

The rest of the paper is organized as follows. We briefly describe the mathematical preliminaries of Miller and Blake et al.'s algorithms in section 2. In section 3, we describe our proposed algorithm. Its analysis is given in section 4. Finally, some concluding remarks are given in section 5.

## 2. Mathematical preliminaries

2.1. **Weil/Tate pairing and Miller's algorithm.** Let $q = p^k$ with a prime $p$, then $\mathbb{F}_q$ is a finite field with $q$ elements and $p$ is the *characteristic* of $\mathbb{F}_q$. An elliptic curve $E$ define over $\mathbb{F}_q$, can be described as the set of points $(x, y)$ satisfying the *Weierstrass* equation $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where $a_i \in \mathbb{F}_q$. If $K$ is an extension of $\mathbb{F}_q$, the set of $K$-rational points of $E$ together with with an additional *point at infinity*, denoted as $\infty$. There exists an abelian group law on E. Explicit formulas for computing the coordinates of a point $R = P + Q$ from the coordinates of $P$ and $Q$ are well know[10, 14].

A *divisor* $D$ is a formal sum with symbols on $E$ and with integer coefficients, denote as $D = \sum_{P\in E} n_P(P)$. The set of all *divisors*, denoted by $\mathrm{Div}(E)$, is a free abelian group generated by $E(\overline{K})$. The *degree* and the *sum* of a *divisor* $D$ are defined as $\deg(D) = \sum_{P\in E} n_P$ and $sum(D) = \sum_{P\in E} n_P P$, respectively. Let $\mathrm{Div}^0(E) = \{D \in Div(E)|deg(D) = 0\}$. Then the *sum* function is a surjective homomorphism from $\mathrm{Div}^0(E)$ to $E$.

Define the divisor of a nonzero rational function $f$ as $\mathrm{div}(f) = \sum_{P\in E}\mathrm{ord}_P(f)(P)$, where $\mathrm{ord}_P(f)$ is the order of $f$ at $P$. Then $\mathrm{div}(f) \in \mathrm{Div}^0(E)$ and it is called a *principle divisor*. It is well known that the set of *principle divisors* is the kernel of *sum*. The quotient group $\mathrm{Div}^0(E)/(principle\ divisors)$ is called the *Picard group* of $E$. Moreover, two divisors $D_1$ and $D_2$ are said to be equivalent, denoted as $D_1 \sim D_2$, *iff* $D_1 - D_2$ is principle, i.e. there exists a nonzero rational function $f$ such that $D_1 = D_2 + \mathrm{div}(f)$. The *support* of a divisor $D$ is the set of points with nonzero coefficients, that is, $supp(D) = \{P \in E|n_P \neq 0\}$. If $\mathrm{div}(f)$ and $D$ have disjoint support, then we can evaluate $f(D) = \prod_{P\in E} f(P)^{n_P}$.

Let $n$ be an integer relatively prime to $q$. Points $P, Q \in E[n]$, where $E[n]$ is the $n$-torsion subgroup of $E(K)$. Then there exist divisors $D_P$ and $D_Q$ such that $D_P \sim (P) - (\infty)$ and $D_Q \sim (Q) - (\infty)$. Further, there exist functions $f_P$ and $f_Q$ such that $\mathrm{div}(f_P) = nD_P$ and $\mathrm{div}(f_Q) = nD_Q$. If $D_P$ and $D_Q$ have disjoint supports, then we can define the Weil pairing as

$$e_n(P, Q) = \frac{f_P(D_Q)}{f_Q(D_P)}.$$

And the Tate pairing of order $n$ is the map

$$\tau_n : E(F_q)[n] \times E(F_{q^k})/nE(F_{q^k}) \to F_{q^k}$$

defined as

$$\tau_n(P,Q) = f_P(D_Q)^{(q^k-1)/n}.$$

Hence, computing the Weil/Tate pairing can be reduced to the evaluation of $f_P(S)$, for each point $S$ in the support of $D_Q$. In 1986, an unpublished manuscript by Miller [11] showed how to do this efficiently. The main idea of Miller's algorithm is using lines to integrate the divisors which has been processed. We briefly describe it as follows:

Let $D_P = (P+R) - (R)$ with an auxiliary point $R$, and define $D_P^j = j(P+R) - j(R) - (jP) + (\infty)$ then there is a rational function $f_j$ such that $\mathrm{div}(f_j) = D_P^j$, for each integer $j$, in particular, $f_n = f_P$. Let $L_{S,T}$ be the line through points $S$ and $T$, if $T \notin \{S, -S\}$ then $L_{S,T}$ is a chord, else if $T = S$ then $L_{S,S}$ is a tangent, else if $T = -S$ then $L_{S,-S}$ a vertical line through points $S$ and $-S$. We will denote $L_{S,-S}$ as $L_S$. Then we have

$$\begin{aligned}
&\mathrm{div}(L_{jP,kP}) = (jP) + (kP) + (-(j+k)P) - 3(\infty) \text{ and}\\
&\mathrm{div}(L_{(j+k)P}) = ((j+k)P) + (-(j+k)P) - 2(\infty). \text{ Hence}\\
&\mathrm{div}(f_{j+k}) = \mathrm{div}(f_j) + \mathrm{div}(f_k) + \mathrm{div}(L_{jP,kP}) - \mathrm{div}(L_{(j+k)P}), \text{ and}\\
&f_{j+k} = f_j f_k \frac{L_{jP,kP}}{L_{(j+k)P}}.
\end{aligned}$$

We can compute $f_n(S)$ recursively with initial values[1] $f_0 = 1$ and $f_1 = L_{P+R}/L_{P,R}$. We describe the following algorithm, which is similar to the algorithm proposed in [4, 5]. Note that we can perform Miller's algorithm to compute Tate pairing by changing the initial value $f_1 = 1$, see [5] for details.

**Algorithm 1** (*Miller's algorithm*).
INPUT: Elliptic curve $E$, integer $n = \sum_{i=0}^{t} b_i 2^i$ with $b_i \in \{0,1\}$ and $b_t = 1$, and points $P, S \in E$ where $P$ has order $n$.
OUTPUT: $f = f_n(S)$.

$f \leftarrow f_1$; $Z \leftarrow P$;
**for** $j \leftarrow t-1$ **down to** $0$ **do**
  $f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}$; $Z \leftarrow 2Z$;
  **if** $b_j = 1$ **then**
    $f \leftarrow f_1 f \frac{L_{Z,P}(S)}{L_{Z+P}(S)}$; $Z \leftarrow Z + P$;
**return** $f$

2.2. **Blake's algorithms.** As we have seen above, the double-and-add method is used in Miller's algorithm. Two lines or four lines must be used if the bit is zero, or one respectively, in the binary expansion of $n$. From the analysis in [4], we know that the more lines we use, the more field multiplications are required in Miller's algorithm. For this reason, Blake et al. propose three algorithms to reduce the number of lines. The first algorithm, referred to as *BMX*-1, is suitable for every case. The second algorithm, referred to as *BMX*-2, can work well if the Hamming weight of $n$ (the number of one bits in the binary expansion of $n$, denote as $H(n)$) is high. The third algorithm is proposed for field of characteristic 3. We are interested in the first two algorithms since it is possible to combine them to

[1]There is a typo in [4] where $f_1 = h_{P,R}/h_{P+R}$ should be $f_1 = h_{P+R}/h_{P,R}$.

further minimize the number of lines. Therefore, we will only describe Blake et al.'s basic idea and their first two algorithms in this section[2]. Their algorithms are based on the following two lemmas proved in [4]:

**Lemma 2.1.** *If the line $L(x,y)=0$ intersects with $E$ at points $P=(a,b), Q=(c,d)$ and $-(P+Q)=(\alpha,\beta)$, then $L(x,y)\overline{L}(x,y)=-(x-a)(x-c)(x-\alpha)$, where $\overline{L}(x,y)$ is the conjugate of $L$ with $L(R)=\overline{L}(-R)$ for $R\in E$.*

**Lemma 2.2.** *Let $Q\in E[n]$ and $S\neq Q,2Q,\ldots,nQ$, then*

(1)
$$\frac{L_{Q,Q}(S)}{L_Q^2(S)L_{2Q}(S)}=-\frac{1}{L_{Q,Q}(-S)}$$

(2)
$$\forall k\in\mathbb{Z}\Rightarrow\frac{L_{(k+1)Q,kQ}(S)}{L_{(k+1)Q}(S)L_{(2k+1)Q}(S)}=-\frac{L_{kQ}(S)}{L_{(k+1)Q,kQ}(-S)}$$

(3)
$$\frac{L_{Q,Q}(S)L_{2Q,Q}(S)}{L_{2Q}(S)L_{3Q}(S)}=-\frac{L_{Q,Q}(S)L_Q(S)}{L_{2Q,Q}(-S)}.$$

Notice that only the vertical lines can be reduced by lemma 2. Thus, all of the vertical lines can be eliminated in the best cases. In other word, one can only replace the tangents and chords by their conjugate to the denominator. We call the tangents and chords as necessary components of Miller's algorithm. There are totally 50% of lines are necessary, these components will appear in denominator or numerator depend on whether Lemma 2 is applied or not. Two of the applications of Lemma 2 are algorithms *BMX*-1 and *BMX*-2 which are given with initial value $f_1=L_{P+R}/L_{P,R}$ or $f_1=1$.

**Algorithm 2** (*BMX*-1).
INPUT: Elliptic curve $E$, integer $n=\sum_{i=0}^{r}q_i4^i$ with $q_i\in\{0,1,2,3\}$ and $q_r\neq 0$, and points $P,S\in E$ where $P$ has order $n$.
OUTPUT: $f=f_n(S)$.

$f\leftarrow f_1$; $Z\leftarrow P$;
**if** $q_r=2$ **then**
  $f\leftarrow f^2\frac{L_{P,P}(S)}{L_{2P}(S)}$; $Z\leftarrow 2P$;
**if** $q_r=3$ **then**
  $f\leftarrow f^3\frac{L_{P,P}^2(S)L_P(S)}{L_{2P,P}(-S)}$; $Z\leftarrow 3P$;
**for** $j\leftarrow r-1$ **down to** $0$ **do**
  **if** $q_j=0$ **then**
    $f\leftarrow f^4\frac{L_{Z,Z}^2(S)}{L_{2Z,2Z}(-S)}$; $Z\leftarrow 4Z$;
  **if** $q_j=1$ **then**
    $f\leftarrow f_1f^4\frac{L_{Z,Z}^2(S)L_{4Z,P}(S)}{L_{2Z,2Z}(-S)L_{4Z+P}(S)}$; $Z\leftarrow 4Z+P$;
  **if** $q_j=2$ **then**
    $f\leftarrow f_1^2f^4\frac{L_{Z,Z}^2(S)L_{2Z,P}^2(S)}{L_{2Z}^2(S)L_{2Z+P,2Z+P}(-S)}$; $Z\leftarrow 4Z+2P$;
  **if** $q_j=3$ **then**

[2]There is another typo in the case "$b_{i-1}=0, b_{i-2}=1$", at the end of page 141 in [4].

$f \leftarrow f_1^3 f^4 \frac{L_{Z,Z}^2(S) L_{2Z,P}^2(S) L_{4Z+2P,P}(S)}{L_{2Z}^2(S) L_{2Z+P,2Z+P}(-S) L_{4Z+3P}(S)}$; $Z \leftarrow 4Z + 3P$;
**return** $f$

*BMX*-1 reduces the lines by processing a pair of consecutive bits together. The best case happened when the pair of bits is "00", which can reduce 50% of lines. The worst case happened when the pair of bits is "11", which can only reduce 25% of lines. Therefore, they suggest using this algorithm for lower $H(n)$ or average cases. However, we observe that *BMX*-1 is more suitable for cases with consecutive zeros. Similarly, *BMX*-2 described in the following, is more suitable for cases with consecutive ones.

**Algorithm 3** (*BMX*-2).
INPUT: Elliptic curve $E$, integer $n = \sum_{i=0}^{t} b_i 2^i$ with $b_i \in \{0, 1\}$ and $b_t = 1$, and points $P, S \in E$ where $P$ has order $n$.
OUTPUT: $f = f_n(S)$.

**if** $b_{t-1} = 0$ **then**
  $f \leftarrow f_1^2 L_{P,P}(S)$; $Z \leftarrow 2P$;
**else**
  $f \leftarrow f_1^3 \frac{L_{P,P}(S) L_{2P,P}(S)}{L_{2P}(S)}$; $Z \leftarrow 3P$;
**for** $j \leftarrow t-2$ **down to** $0$ **do**
  **if** $b_j = 0$ **then**
    $f \leftarrow f^2 \frac{L_{2Z}(S)}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z$;
  **else**
    $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z + P$;
**return** $f$

*BMX*-2 always delays the computation of the value of $L_{2Z}(S)$(or $L_{Z+P}(S)$, depending on the bit $b_j$) in Miller's algorithm at each iteration of the **for** loop. It is easy to see that *BMX*-2 can reduce 50% of lines if the bit is one. For this propose, every vertical line of zero bits must be reused. Unfortunately, the line reused at the cost of no line can be reduced when the bit is zero. This is why they suggest performing *BMX*-2 for higher $H(n)$.

## 3. Refinement of *BMX* algorithms

In this section, we propose a refinement to the *BMX* algorithms. We segment $n$ into different bit sequences. For each sequence, we carefully design the algorithm to reduce more lines. From Lemma 2(a), there are two different kinds of bit sequences which can reduce all of the vertical lines, and they are the best two cases presented in [4]. One is a sequence of even number of zero bits, denoted by $0^{2r}$, and the other is a sequence of ones between two zeros, denoted $01^m0$. In the following, we give two simple examples to explain why these two cases can reduce 50% of lines (all of the vertical lines can be reduced). Suppose the Miller's algorithm has performed up to a point where $f = \lambda$ and $Z = Q$ before processing these two cases.

The first example is the case of $q_j = 0$ in *BMX*-1. Miller's algorithm performs the following two steps.

(a)

$$f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}; Z \leftarrow 2Z; \text{ and}$$

(b)

$$f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}; Z \leftarrow 2Z; .$$

Putting together, we have

$$f \leftarrow (\lambda^2 \frac{L_{Q,Q}(S)}{L_{2Q}(S)})^2 \frac{L_{2Q,2Q}(S)}{L_{4Q}(S)} = \lambda^4 \frac{L^2_{Q,Q}(S)}{L^2_{2Q}(S)} \frac{L_{2Q,2Q}(S)}{L_{4Q}(S)} = \lambda^4 \frac{L^2_{Q,Q}(S)}{L_{2Q,2Q}(-S)}.$$

To simplify the notations, we use "$\circ$" to denote a line, which belongs to a zero bit, use "$\bullet$" to denote a line of a one bit, and use "$*$" to denote a conjugate of a tangent or chord by Lemma 2. Such a representation of lines is called the corresponding dot notation of the lines. Note that, "$\circ$" or "$\bullet$" which appear in denominator is a vertical line. It can be reduced by some suitable conjugate lines. And the others are necessary components, which are tangents or chords in numerator or the conjugate of them. Omitting the exponents and $\lambda$ will not affect the reductions because Miller's algorithm always performs squaring operations in each step. We can denote the deduction of "00" in terms of its corresponding dot notation as follows.

$$\frac{\circ}{\circ}\frac{\circ}{\circ} = \frac{\circ}{*},$$

and the reduction ratio of the lines of "00" is $(4-2)/4 = 50\%$.

The second example is the case of processing "0110".
Miller's algorithm performs:

(a)

$$f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}; Z \leftarrow 2Z;$$

(b)

$$f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}; Z \leftarrow 2Z; \text{ and } f \leftarrow f_1 f \frac{L_{Z,P}(S)}{L_{Z+P}(S)}; Z \leftarrow Z + P;$$

(c)

$$f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}; Z \leftarrow 2Z; \text{ and } f \leftarrow f_1 f \frac{L_{Z,P}(S)}{L_{Z+P}(S)}; Z \leftarrow Z + P;$$

(d)

$$f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}; Z \leftarrow 2Z; .$$

And we have

(a)

$$f \leftarrow \lambda^2 \frac{L_{Q,Q}(S)}{L_{2Q}(S)},$$

(b)

$$f \leftarrow \lambda^4 \frac{L^2_{Q,Q}(S)}{L^2_{2Q}(S)} \frac{L_{2Q,2Q}(S)}{L_{4Q}(S)} \text{ and } f \leftarrow f_1 \lambda^4 \frac{L^2_{Q,Q}(S)}{L^2_{2Q}(S)} \frac{L_{2Q,2Q}(S) L_{4Q,P}(S)}{L_{4Q}(S) L_{4Q+P}(S)},$$

(c)

$$\begin{aligned} f &\leftarrow f_1^2 \lambda^8 \frac{L^4_{Q,Q}(S)}{L^4_{2Q}(S)} \frac{L^2_{2Q,2Q}(S) L^2_{4Q,P}(S)}{L^2_{4Q}(S) L^2_{4Q+P}(S)} \frac{L_{4Q+P,4Q+P}(S)}{L_{8Q+2P}(S)} \text{ and} \\ f &\leftarrow f_1^3 \lambda^8 \frac{L^4_{Q,Q}(S)}{L^4_{2Q}(S)} \frac{L^2_{2Q,2Q}(S) L^2_{4Q,P}(S)}{L^2_{4Q}(S) L^2_{4Q+P}(S)} \frac{L_{4Q+P,4Q+P}(S) L_{8Q+2P,P}(S)}{L_{8Q+2P}(S) L_{8Q+3P}(S)}, \end{aligned}$$

(d)

$$f \leftarrow f_1^6 \lambda^{16} \frac{L^8_{Q,Q}(S)}{L^8_{2Q}(S)} \frac{L^4_{2Q,2Q}(S) L^4_{4Q,P}(S)}{L^4_{4Q}(S) L^4_{4Q+P}(S)} \frac{L^2_{4Q+P,4Q+P}(S) L^2_{8Q+2P,P}(S)}{L^2_{8Q+2P}(S) L^2_{8Q+3P}(S)} \frac{L_{8Q+3P,8Q+3P}(S)}{L_{16Q+6P}(S)}.$$

Reducing $f$ by Lemma 2(a), we have:

$$f \leftarrow f_1^6 \lambda^{16} L^8_{Q,Q}(S) \frac{L^4_{4Q,P}(S)}{L^4_{2Q,2Q}(-S)} \frac{L^2_{8Q+2P,P}(S)}{L^2_{4Q+P,4Q+P}(-S)} \frac{1}{L_{8Q+3P,8Q+3P}(-S)}.$$

We can denote the corresponding dot notation of the reduction of "0110" as follows

$$\frac{\circ}{\circ}\frac{\bullet\bullet}{\bullet\bullet}\frac{\bullet\bullet}{\bullet\bullet}\frac{\circ}{\circ} = \circ\frac{\bullet}{*}\frac{\bullet}{*}\frac{1}{*},$$

and the reduction ratio of the lines is $(12-6)/(12) = 50\%$.

Inductively, the reduction of $0^{2r} = (00)^r$ is

$$\left(\frac{\circ}{\circ}\frac{\circ}{\circ}\right)^r = \left(\frac{\circ}{*}\right)^r,$$

with reduction ratio of the lines is $2r/4r = 50\%$, and the reduction of the other case "$01^m0$" is

$$\frac{\circ}{\circ}\underbrace{\frac{\bullet\bullet}{\bullet\bullet}\frac{\bullet\bullet}{\bullet\bullet}\cdots\frac{\bullet\bullet}{\bullet\bullet}}_{m}\frac{\circ}{\circ} = \circ\underbrace{\frac{\bullet}{*}\frac{\bullet}{*}\cdots\frac{\bullet}{*}}_{m}\frac{1}{*},$$

with the reduction ratio $(2m+2)/(4m+4) = 50\%$. As above, the two lines of bit zero can be reduced only with the lines of the previous or the next bit. But, the four lines of one bit can be reduced with both of the lines of the previous and next bit, if available. That is why *BMX*-2 can "cascading-cut" the lines of one bit by coercing the last line at denominator to be shifted to the next step, but the reduction ratio of zero bits is always 0%. To summarize, there are at most 50% of lines, which can be reduced in Miller's algorithm by the concept of "*conjugate*".

In order to further reducing the number of lines in *BMX*-1 and *BMX*-2, we propose an algorithm [see Appendix] to segment $n$ from right to left into various patterns. It tries to find the longest chain of bit 1s and distribute the bit 0s carefully. There are four cases in this algorithm when scanning the binary expansion of $n$. There are described as follows.

(1) If the current two bits are "00" then find the first 1 bit, $b_k$, from right to left. The algorithm will collect all of the zero bits but the leftmost zero into a block. It produces a block of "$0^m$". The current two bits will become "10".

(2) If the current two bits are "10" then the algorithm will find the first 0 bit from right to left. It will collect all of them and produce a block of "$01^m0$". Finally, the algorithm will move the pointer to the next two bits.
(3) If the current two bits are "01" then the algorithm will produce a block of "$(01)^m$" and move the pointer to the next two bits.
(4) If the current two bits are "11" then the algorithm will find the first 0 bit from right to left. It will collect all of them to produce a block as "$01^w$" and move the pointer to the next two bits.

In case 2, if no zero bit can be found then it will produce a bloke of "$1^m0$", and in case 4, if no zero bit can be found then it will produce a block of "$1^m$". Therefore, we have six kinds of patterns $l_0 = 0^m, l_1 = 1^m, l_2 = (01)^m, l_3 = 01^w, l_4 = 1^m0$ and $l_5 = 01^m0$, where $m \geq 1$ and $w \geq 2$. There are certain restrictions among the patterns, we state them as basic properties.

(1) $1^m$ and $1^m0$ can only appear in the leftmost block. Every pattern can follow $1^m0$, but only $(01)^m$, $01^w$ and $01^m0$ can follow $1^m$.
(2) If $i \geq 1$ and $B_i \in \{0^m, (01)^m, 01^w\}$ then $B_{i-1} \in \{(01)^m, 01^w, 01^m0\}$. But $(01)^m$ can never follow $(01)^m$.
(3) $(01)^m$, $01^w$ and $01^m0$ can follow any pattern.

The following is a simple example of the *Segmentation* algorithm.
Let $n = (1111100111000101010111)_2$. Then the output of *Segmentation* algorithm is "$\{(l_4, 4), (l_5, 3), (l_0, 1), (l_2, 3), (l_3, 3)\}$".

A bit pattern is said to be "*perfect*" if we can reduce 50% of lines of the corresponding lines in Miller's algorithm. It is said to be "*standard*" if we can reduce more than 25% of lines, in the other words, 50% of vertical lines are reduced. If we can not reduce any lines of a bit, then the bit is said to be "*isolate*". Hence $01^m0$ and $0^{2r}$ are two perfect patterns, both of $0^{2r-1}$ and $1^1$ always have an isolate bit, and the others are standard.

Before describing the algorithm we illustrate the reduction rule and compute the reduction ratio for those imperfect patterns as follows.Note that, the last vertical line in Miller's algorithm is $L_{nP} = 1$, therefore, we do not have to count it.

(1)

$$l_0 = 0^{2r-1} : \underbrace{(\frac{\circ\,\circ}{\circ\,\circ}) \dots (\frac{\circ\,\circ}{\circ\,\circ})}_{r-1} \frac{\circ}{\circ} = \underbrace{\frac{\circ\,\circ}{*\,*} \dots \frac{\circ}{*}}_{r-1} \frac{\circ}{\circ}, \text{ we need } 2r \text{ lines.}$$

However, if it appears in the rightmost block then we need $2r - 1$ lines. Note that, the last zero bit is an isolate bit.

(2)

$$l_1 = 1^m : \underbrace{\frac{\bullet\bullet}{\bullet\bullet}\frac{\bullet\bullet}{\bullet\bullet} \dots \frac{\bullet\bullet}{\bullet\bullet}}_{m} = \frac{\bullet\bullet}{\bullet} \underbrace{\frac{\bullet}{*} \dots \frac{\bullet}{*}}_{m-2} \frac{\bullet}{*\bullet}, \text{ we need } 2m + 2 \text{ lines.}$$

However, if the *Segmentation* algorithm produces of only one block "$1^m$" then we need only $2m + 1$ lines. If $m = 1$ then there is an isolated bit and the corresponding dot notation is $\frac{\bullet\bullet}{\bullet\bullet}$.

(3)

$$l_2 = (01)^m : \underbrace{(\frac{\circ}{\circ}\frac{\bullet\bullet}{\bullet\bullet})\dots(\frac{\circ}{\circ}\frac{\bullet\bullet}{\bullet\bullet})}_{m} = \underbrace{(\frac{\circ\bullet}{*\bullet})\dots(\frac{\circ\bullet}{*\bullet})}_{m},\ \text{we need } 4m \text{ lines.}$$

However, if it appears in the rightmost block then we need $4m-1$ lines. Note that it is similar to the cases $q_r = 1$ or $q_r = 2$ in *BMX*-1.

(4)

$$l_3 = 01^w : \frac{\circ}{\circ}\underbrace{\frac{\bullet\bullet}{\bullet\bullet}\frac{\bullet\bullet}{\bullet\bullet}\dots\frac{\bullet\bullet}{\bullet\bullet}}_{w} = \circ\underbrace{\frac{\bullet}{*}\dots\frac{\bullet}{*}}_{w-1}\frac{\bullet}{*\bullet},\ \text{we need } 2w+2 \text{ lines.}$$

However, if it appears in the rightmost block then we need $2w+1$ lines.

(5)

$$l_4 = 1^m0 : \underbrace{\frac{\bullet\bullet}{\bullet\bullet}\frac{\bullet\bullet}{\bullet\bullet}\dots\frac{\bullet\bullet}{\bullet\bullet}}_{m}\frac{\circ}{\circ} = \frac{\bullet\bullet}{\bullet}\underbrace{\frac{\bullet}{*}\dots\frac{\bullet}{*}}_{m-2}\frac{\bullet}{**},\ \text{we need } 2m+2 \text{ lines.}$$

As we have seen above, cases (2) and (5) always appear in the leftmost part of $n$, as a result, there is a vertical line which cannot be reduced in the leftmost bit in Miller's algorithm. All of the cases from (1) to (4) have a vertical line at the final bit. There are $m$ vertical lines in total which cannot be reduced in case (3). This is the worst case. Fortunately, in practically interesting cases, such as, when $n$ is Solinas number of the form $2^a \pm 2^b \pm 1$[4, 11], the cases $(01)^m$ are rare. Therefore, our refinement is more suitable for Solinas numbers than *BMX* algorithms. The following algorithm is designed based on these rules to minimize the number of lines in Miller's algorithm. As usual, the initial value is $f_1 = \frac{L_{P+R}}{L_{P,R}}$ or 1.

**Algorithm 4** (*Refinement of BMX algorithm).*
INPUT: Elliptic curve $E$, integer $n = \{B_t, B_{t-1}, \dots, B_1, B_0\}$ where $B_j \in \{(l_0, m_0), (l_1, m_1), (l_2, m_2)(l_3, w), (l_4, m_4), (l_5, m_5)\}$ and $m_i, w \in \mathbb{N}$, $w \geq 2$ and points $P, S \in E$ where $P$ has order $n$.
OUTPUT: $f = f_n(S)$.

$f \leftarrow f_1$; $Z \leftarrow P$;
**for** $j \leftarrow t$ **down to** 0 **do**
  **if** $B_j = (l_0, m)$ **then** $\triangleright$ **case 1.**
    **if** $m$ is even **then**
      $m \leftarrow m/2$;
      **for** $k \leftarrow 1$ **to** $m$ **do**
        $f \leftarrow f^4 \frac{L^2_{Z,Z}(S)}{L_{2Z,2Z}(-S)}$; $Z \leftarrow 4Z$;
    **else**
      $m \leftarrow (m-1)/2$;
      **for** $k \leftarrow 1$ **to** $m$ **do**
        $f \leftarrow f^4 \frac{L^2_{Z,Z}(S)}{L_{2Z,2Z}(-S)}$; $Z \leftarrow 4Z$;
      $f \leftarrow f^2 \frac{L_{Z,Z}(S)}{L_{2Z}(S)}$; $Z \leftarrow 2Z$;

  **if** $B_j = (l_1, m)$ **then** $\triangleright$ **case 2.**

**if** $m = 1$ **then**
  $f \leftarrow f_1^3 \frac{L_{P,P}(S)L_{2P,P}(S)}{L_{2P}(S)L_{3P}(S)}$; $Z \leftarrow 3P$;
**else if** $m = 2$ **then**
  $f \leftarrow f_1^7 \frac{L_{P,P}^2(S)L_{2P,P}^2(S)L_{6P,P}(S)}{L_{2P}^2(S)L_{3P,3P}(-S)L_{7P}(S)}$; $Z \leftarrow 7P$;
**else**
  $f \leftarrow f_1^3 \frac{L_{P,P}(S)L_{2P,P}(S)}{L_{2P}(S)}$; $Z \leftarrow 3P$;
  **for** $k \leftarrow 1$ **to** $m-2$ **do**
    $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z + P$;
  $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)L_{2Z+P}(S)}$; $Z \leftarrow 2Z + P$;

**if** $B_j = (l_2, m)$ **then** ▷ **case 3.**
  **for** $k \leftarrow 1$ **to** $m$ **do**
    $f \leftarrow f_1 f^4 \frac{L_{Z,Z}^2(S)L_{4Z,P}(S)}{L_{2Z,2Z}(-S)L_{4Z+P}(S)}$; $Z \leftarrow 4Z + P$;

**if** $B_j = (l_3, w)$ **then** ▷ **case 4.**
  $f \leftarrow f^2 L_{Z,Z}(S)$; $Z \leftarrow 2Z$;
  **for** $k \leftarrow 1$ **to** $w-1$ **do**
    $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z + P$;
  $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)L_{2Z+P}(S)}$; $Z \leftarrow 2Z + P$;

**if** $B_j = (l_4, m)$ **then** ▷ **case 5.**
  **if** $m = 1$ **then**
    $f \leftarrow f_1^6 \frac{L_{P,P}^2(S)L_{2P,P}^2(S)}{L_{2P}^2(S)L_{3P,3P}(-S)}$; $Z \leftarrow 6P$;
  **else**
    $f \leftarrow f_1^3 \frac{L_{P,P}(S)L_{2P,P}(S)}{L_{2P}(S)}$; $Z \leftarrow 3P$;
    **for** $k \leftarrow 1$ **to** $m-1$ **do**
      $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z + P$;
    $f \leftarrow f^2 \frac{1}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z$;

**if** $B_j = (l_5, m)$ **then** ▷ **case 6.**
  $f \leftarrow f^2 L_{Z,Z}(S)$; $Z \leftarrow 2Z$;
  **while** $m \neq 1$ **do**
    $f \leftarrow f_1 f^2 \frac{L_{2Z,P}(S)}{L_{Z,Z}(-S)}$; $Z \leftarrow 2Z + P$; $m \leftarrow m - 1$;
  $f \leftarrow f_1^2 f^4 \frac{L_{2Z,P}^2(S)}{L_{Z,Z}^2(S)L_{2Z+P,2Z+P}(-S)}$; $Z \leftarrow 4Z + 2P$;

**return** $f$

## 4. Analysis

We denote $N_{Alg}(n)$ to be the number of lines used by the algorithm $Alg$ with input integer $n$. For example, $N_{Ref}(n)$ is the number of lines used in our refinement. We will show $N_{Ref}(n) \leq N_{BMX-1}(n)$ in section 4.1 and $N_{Ref}(n) \leq N_{BMX-2}(n)$ in section 4.2. That is, our algorithm can reduce more lines than both of the $BMX$ algorithms. To simplify the analysis, we count the number of lines based on the corresponding dot notations introduced in section 3. Suppose $n$ is a $k+1$ bits integer, that is $n = \sum_{i=0}^{k} b_i 2^i$ with $b_k \neq 0$, and we can segment $n$ into $t+1$ blocks, denoted as $\{B_t, B_{t-1}, \dots, B_0\}$ where $B_i \in \{(01)^m, 01^w, 01^m0, 0^m, 1^m0, 1^m\}$, with $t \geq 0, m \geq 1$ and $w \geq 2$. Then $N_{Ref}(n) = \sum_{i=0}^{t} N_{Ref}(B_i)$ and $N_{BMX-2}(n) = \sum_{i=0}^{t} N_{BMX-2}(B_i)$.

4.1. **Comparison of *BMX*-1 and the Refinement.** There are four cases performed in *BMX*-1, namely $q_j = 0$, $q_j = 1$, $q_j = 2$ and $q_j = 3$. We will show that our refinement requires no more lines than *BMX*-1 in each case. Hence $N_{Ref}(n) \leq N_{BMX-1}(n)$.

(1) The case $q_j = (3)_4 = (11)_2$ is obvious. There are six lines needed in *BMX*-1. In our refinement, the best location of $(11)_2$ is when it is part of $01^m0$, $1^m$, $1^m0$ and $01^w$, in these cases only four lines are needed. And the worst case is when $(11)_2$ located in the first block, which needs six lines. Note that our refinement and Miller's algorithm always neglect the leftmost bit, but *BMX*-1 must consider this bit. Therefore, *BMX*-1 needs 6 lines in this case.

(2) Four lines are required in the case $q_j = (2)_4 = (10)_2$ when we perform *BMX*-1. The zero bit in $(10)_2$ can never be an isolate bit in our algorithm, and it will be reduced with the previous or the next bit, so that four lines are needed in our algorithm.

(3) Similarly, four lines are required in the case $q_j = (1)_4 = (01)_2$ when we perform *BMX*-1. In our algorithm, it is part of the pattern $(01)^m$. We do it better when $(01)_2$ interlaces with $(10)_2$. For example, when $(01)_2$ is part of the pattern $(0110)_2$. In *BMX*-1, $(0110)_2$ will be reduced as

$$\frac{\circ}{\circ}\frac{\bullet\bullet}{\bullet\bullet}\frac{\bullet\bullet}{\bullet\bullet}\frac{\circ}{\circ} = \frac{\circ\bullet}{*\bullet}\frac{\bullet\bullet}{\bullet *},$$

whereas it is reduced as

$$\circ\frac{\bullet}{*}\frac{\bullet}{**}$$

in our refinement.

(4) The case $q_j = (0)_4 = (00)_2$ is the most complicated one. We need to show that the isolate zero bits do not affect the total number of lines. By the way the bit sequence is divided into patterns, the isolate zero bits will appear in the block $(0^{2r-1})_2$ between $(\dots 10)_2$ and $(01\dots)_2$, by the properties of the patterns we have described in session 3, except that it is in the front or tail. Because there are odd number of zeros in $(0^{2r-1})_2$. If we encode the binary notation $[\dots 10)(0^{2r-1})(01\dots]_2$ in tetral, then there are two possible cases to deal with the zero chain. One is to aggregate the leftmost zero bit to another zero bit which in the previous block, the other is to aggregate the rightmost zero bit to another zero bit which in the next block. In both cases, there are even number of zero bits remained. As we know, all the vertical lines of the even number zero bits can be reduced by *BMX*-1 and our algorithm. Therefore, we can simplify $[\dots 10)(0^{2r-1})(01\dots]_2$ to $[\dots 10)(0)(01\dots]_2$. And it is easy to see that, we can do better than *BMX*-1 in the case of $[\dots 110)(0)(011\dots]_2$. Therefore, we consider the two sub-cases

when dividing the sequence $[\ldots 010)(0)(010\ldots]_2$.

(a)

$$[\ldots\underline{10}(0)0\underline{10}\ldots]_2 = [\ldots\underline{2}0\underline{2}\ldots]_4 \Rightarrow \ldots(\frac{\bullet\bullet\ \circ}{\bullet\bullet\ \circ})(\frac{\circ\ \circ}{\circ\ \circ})(\frac{\bullet\bullet\ \circ}{\bullet\bullet\ \circ})\ldots$$

*BMX*-1 reduces it to " $\ldots(\frac{\bullet\bullet}{\bullet *})(\frac{\circ}{*})(\frac{\bullet\bullet}{\bullet *})\ldots$ ", and the number of lines for the corresponding bits is ten.

The dot notation for the corresponding bits of our refinement is

" $\ldots\frac{\bullet}{**})(\frac{\circ}{\circ})(\circ\frac{\bullet}{**})\ldots,$ " which has only nine lines.

Note that, if the location of the first one bit is $k-1$, then we need ten lines, too.

(b)

$$(\ldots\underline{01}0(0)\underline{01}\ldots)_2 = (\ldots\underline{1}0\underline{1}\ldots)_4 = \ldots(\frac{\circ\ \bullet\bullet}{\circ\ \bullet\bullet})(\frac{\circ\ \circ}{\circ\ \circ})(\frac{\circ\ \bullet\bullet}{\circ\ \bullet\bullet})\ldots$$

The dot notation for the corresponding bits of *BMX*-1 is

" $\ldots(\frac{\circ\bullet}{*\bullet})(\frac{\circ}{*})(\frac{\circ\bullet}{*\bullet})\ldots$ ",

and the number of lines of the corresponding bits is ten.

The dot notation for the corresponding bits of our refinement is

" $\ldots(\frac{\circ\bullet}{**})(\frac{\circ}{\circ})(\circ\frac{\bullet}{*}\ldots$ " or " $\ldots(\frac{\circ\bullet}{**})(\frac{\circ}{\circ})(\frac{\circ\bullet}{*\bullet})\ldots$ ".

Therefore, we need nine lines in the general cases and ten lines in the worst cases. Note that if the last one bit is the right most bit then the number of lines required in our refinement and *BMX*-1 is nine.

Since our refinement requires no more lines in each case, we have $N_{Ref}(n) \leq N_{BMX-1}(n)$. ∎

4.2. **Comparison of *BMX*-2 and the Refinement.** Before the comparison, we describe the requirement of the number of lines for each bit of $n$ in *BMX*-2. It is easy to see that, every bit, $b_i$, needs two lines in the **for** loop. Before the iteration, *BMX*-2 needs only one line if the first bit, $b_{k-1}$, is zero, otherwise it needs three lines. And if the last bit, $b_0$, is zero, it needs one line. Therefore, we can compute the total number of lines of a sequence of bits. Based on the number of lines of our refinement by the corresponding dot notations in section 3, we summarize them in the following table.

**Table 1.** $N_{Ref}(B_i)$ and $N_{BMX-2}(B_i)$

| $B_i$ | $N_{BMX-2}(B_i)$ | $N_{BMX-2}(B_t)$ | $N_{Ref}(B_i)$ | $N_{Ref}(B_0)$ |
|---|---|---|---|---|
| $(01)^m$ | $4m$ | $4m-1$ | $4m$ | $4m-1$ |
| $01^w$ | $2w+2$ | $\underline{2w+1}$ | $\underline{2w+2}$ | $2w+1$ |
| $01^m0$ | $2m+4$ | $2m+3$ | $2m+2$ | $2m+2$ |
| $0^m$ | $2m$ | $\underline{2m-1}$ | $\lceil m/2\rceil \times 2$ | $m$ |
| $1^m0$ | — | $\underline{2m+2}$ | $\underline{2m+3}$ | $2m+2$ |
| $1^m$ | — | $\underline{2m+1}$ | $\underline{2m+2}$ | $2m+1$ |

From those underlined items in Table 1, we have $N_{Ref}(B_t) = N_{BMX-2}(B_t) + 1$, when $B_t \in \{01^w, 0, 1^m0, 1^m\}$ and $1 \leq t$. For the remaining blocks, we have $N_{Ref}(B_i) \leq N_{BMX-2}(B_i)$ for $0 \leq i < t$. Therefore,

$$\sum_{i=0}^{t-1} N_{Ref}(B_i) \leq \sum_{i=0}^{t-1} N_{BMX-2}(B_i).$$

If $B_t \notin \{01^w, 0, 1^m0, 1^m\}$ or $t = 0$ then we have

$$N_{Ref}(n) = \sum_{i=0}^{t} N_{Ref}(B_i) \leq \sum_{i=0}^{t} N_{BMX-2}(B_i) = N_{BMX-2}(n).$$

Therefore, to show $N_{Ref}(n) \leq N_{BMX-2}(n)$ with integer $n$ segmented into $t+1$ blocks $\{B_t, B_{t-1}, \ldots, B_0\}$ where $B_t \in \{01^w, 0, 1^m0, 1^m\}$ and $1 \leq t$, we only need to prove $\sum_{i=0}^{t-1} N_{Ref}(B_i) < \sum_{i=0}^{t-1} N_{BMX-2}(B_i)$ since $N_{Ref}(B_t) = N_{BMX-2}(B_t) + 1$. First we show the following two properties.

(1) If $B_k = 01^m0$ for some $k$ and $0 \leq k \leq t$ then $N_{Ref}(n) \leq N_{BMX-2}(n)$.

(2) If $B_k \in \{(01)^m, 01^w\}$ for some $k$ and $0 \leq k < t$ then $\sum_{i=0}^{t-1} N_{Ref}(B_i) < \sum_{i=0}^{t-1} N_{BMX-2}(B_i)$.

Property (1) can be verified by a straightforward calculation. We prove the case for $B_k = (01)^m$ in (2) as follows. The other case, $B_k = 01^w$, can be proved similarly.

As we have seen in Table 1, if $1 \leq t$ and $B_0 \in \{(01)^m, 01^w, 01^m0, 0^m\}$ then we have $N_{Ref}(B_0) < N_{BMX-2}(B_0)$. Therefore, suppose $j$ is the minimal index such that $B_j = (01)^m$. If $j = 0$ then we are done. Otherwise, by the properties stated in section 3, $B_{j-1} \in \{01^w, 01^m0\}$. If $j - 1 = 0$ or $B_{j-1} = 01^m0$ then we have $\sum_{i=0}^{t-1} N_{Ref}(B_i) < \sum_{i=0}^{t-1} N_{BMX-2}(B_i)$. Otherwise, $B_{j-1} = 01^w$, and $B_{j-2} \in \{01^w, (01)^m, 01^m0\}$. Since $j$ is the minimal index for $B_j = (01)^m$, $B_{j-2} \in \{01^w, 01^m0\}$. Therefore, we are left with only two cases $B_0 = 01^w$ and $B_l = 01^m0$ for some $l$ and $0 \leq l \leq j-2$. Both of them we have $\sum_{i=0}^{t-1} N_{Ref}(B_i) < \sum_{i=0}^{t-1} N_{BMX-2}(B_i)$, if there exists $B_k = (01)^m$ for some $k$ and $0 \leq k < t$.

Now we are ready to show $N_{Ref}(n) \leq N_{BMX-2}(n)$ when $n$ is segmented into $t+1$ blocks $\{B_t, B_{t-1}, \ldots, B_0\}$ with $1 \leq t$ and $B_t \in \{01^w, 0, 1^m0, 1^m\}$. As we have known, if $B_t \in \{01^w, 0, 1^m\}$ then $B_{t-1} \in \{(01)^m, 01^w, 01^m0\}$. From properties (1) and (2), we have $N_{Ref}(n) \leq N_{BMX-2}(n)$. And if $B_t = (01)^m$ then $B_{t-1} \in \{01^w, 01^m0\}$. Based on the same argument we have $N_{Ref}(n) \leq N_{BMX-2}(n)$. ∎

4.3. **Examples.** We give three examples to show the strength of our refinement. The first two examples come from [4]. *BMX*-1 (*BMX*-2) has a good result in the first example (respectively, the second example). We will use the best of *BMX*-1 and *BMX*-2 to compare with our refinement. The third example shows that our refinement is better than both of *BMX*-1 and *BMX*-2. We shorten the symbols $L_{aP,bP}$, $L_{aP}$ as $L_{a,b}$, $L_a$ respectively, and denote $L_{aP,bP}(-S)$ as $\overline{L}_{a,b}$. In the following tables, "o" stands for Miller's algorithm, "i" stands for *BMX*-1, "ii" stands for *BMX*-2, "iii" stands for our refinement and "NL" stands for the number of lines required in the corresponding algorithm.

**Example 4.1.** Compute $f_{257}$:

| $n$: | $257 = (100000001)_2 = (10001)_4 \Rightarrow \{(l_0, 6), (l_2, 1)\}$ | NL |
|---|---|---|
| o: | $f_1^{257} \frac{L_{1,1}^{128}}{L_2^{128}} \frac{L_{2,2}^{64}}{L_4^{64}} \frac{L_{4,4}^{32}}{L_8^{32}} \frac{L_{8,8}^{16}}{L_{16}^{16}} \frac{L_{16,16}^{8}}{L_{32}^{8}} \frac{L_{32,32}^{4}}{L_{64}^{4}} \frac{L_{64,64}^{2}}{L_{128}^{2}} \frac{L_{128,128}}{L_{256}} \frac{L_{256,1}}{L_{257}}$ | 17 |
| i: | $f_1^{257} \frac{L_{1,1}^{128}}{\overline{L}_{2,2}^{64}} \frac{L_{4,4}^{32}}{\overline{L}_{8,8}^{16}} \frac{L_{16,16}^{8}}{\overline{L}_{32,32}^{4}} \frac{L_{64,64}^{2}}{\overline{L}_{128,128}} \frac{L_{256,1}}{L_{257}}$ | 9 |
| iii: | $f_1^{257} \frac{L_{1,1}^{128}}{\overline{L}_{2,2}^{64}} \frac{L_{4,4}^{32}}{\overline{L}_{8,8}^{16}} \frac{L_{16,16}^{8}}{\overline{L}_{32,32}^{4}} \frac{L_{64,64}^{2}}{\overline{L}_{128,128}} \frac{L_{256,1}}{L_{257}}$ | 9 |

**Example 4.2.** Compute $f_{191}$:

| $n$: | $191 = (10111111)_2 \Rightarrow \{(l_3, 6)\}$ | NL |
|---|---|---|
| o: | $f_1^{191} \frac{L_{1,1}^{64}}{L_2^{64}} \frac{L_{2,2}^{32}}{L_4^{32}} \frac{L_{4,1}^{32}}{L_5^{32}} \frac{L_{5,5}^{16}}{L_{10}^{16}} \frac{L_{10,1}^{16}}{L_{11}^{16}} \frac{L_{11,11}^{8}}{L_{22}^{8}} \frac{L_{22,1}^{8}}{L_{23}^{8}} \frac{L_{23,23}^{4}}{L_{46}^{4}} \frac{L_{46,1}^{4}}{L_{47}^{4}} \frac{L_{47,47}^{2}}{L_{94}^{2}} \frac{L_{94,1}^{2}}{L_{95}^{2}} \frac{L_{95,95}}{L_{190}} \frac{L_{190,1}}{L_{191}}$ | 25 |
| ii: | $f_1^{191} L_{1,1}^{64} \frac{L_{4,1}^{32}}{\overline{L}_{2,2}^{32}} \frac{L_{10,1}^{16}}{\overline{L}_{5,5}^{16}} \frac{L_{22,1}^{8}}{\overline{L}_{11,11}^{8}} \frac{L_{46,1}^{4}}{\overline{L}_{23,23}^{4}} \frac{L_{94,1}^{2}}{\overline{L}_{47,47}^{2}} \frac{L_{190,1}}{\overline{L}_{95,95}}$ | 13 |
| iii: | $f_1^{191} L_{1,1}^{64} \frac{L_{4,1}^{32}}{\overline{L}_{2,2}^{32}} \frac{L_{10,1}^{16}}{\overline{L}_{5,5}^{16}} \frac{L_{22,1}^{8}}{\overline{L}_{11,11}^{8}} \frac{L_{46,1}^{4}}{\overline{L}_{23,23}^{4}} \frac{L_{94,1}^{2}}{\overline{L}_{47,47}^{2}} \frac{L_{190,1}}{\overline{L}_{95,95} L_{191}}$ | 13 |

**Example 4.3.** Compute $f_{1567}$:

| $n$: | $1567 = (11000011111)_2 = (120133)_4 \Rightarrow \{(l_3, 1), (l_0, 2)(l_3, 5)\}$ | NL |
|---|---|---|
| o: | $f_1^{1567} \frac{L_{1,1}^{512}}{L_2^{512}} \frac{L_{2,1}^{512}}{L_3^{512}} \frac{L_{3,3}^{256}}{L_6^{256}} \frac{L_{6,6}^{128}}{L_{12}^{128}} \frac{L_{12,12}^{64}}{L_{24}^{64}} \frac{L_{24,24}^{32}}{L_{48}^{32}} \frac{L_{48,48}^{16}}{L_{96}^{16}} \frac{L_{96,1}^{16}}{L_{97}^{16}} \frac{L_{97,97}^{8}}{L_{194}^{8}} \frac{L_{194,1}^{8}}{L_{195}^{8}} \frac{L_{195,195}^{4}}{L_{390}^{4}} \frac{L_{390,1}^{4}}{L_{391}^{4}}$ $\frac{L_{391,391}^{2}}{L_{782}^{2}} \frac{L_{782,1}^{2}}{L_{783}^{2}} \frac{L_{783,783}}{L_{1566}} \frac{L_{1566,1}}{L_{1567}}$ | 31 |
| i: | $f_1^{1567} \frac{L_{1,1}^{512}}{L_2^{512}} \frac{L_{2,1}^{512}}{\overline{L}_{3,3}^{256}} \frac{L_{6,6}^{128}}{\overline{L}_{12,12}^{64}} \frac{L_{24,24}^{32}}{\overline{L}_{48,48}^{16}} \frac{L_{96,1}^{16}}{L_{97}^{16}} \frac{L_{97,97}^{8}}{L_{194}^{8}} \frac{L_{194,1}^{8}}{\overline{L}_{195,195}^{4}} \frac{L_{390,1}^{4}}{L_{391}^{4}} \frac{L_{391,391}^{2}}{L_{782}^{2}} \frac{L_{782,1}^{2}}{\overline{L}_{783,783}} \frac{L_{1566,1}}{L_{1567}}$ | 21 |
| ii: | $f_1^{1567} \frac{L_{1,1}^{512} L_{2,1}^{512}}{L_2^{512}} \frac{L_6^{256}}{\overline{L}_{3,3}^{256}} \frac{L_{12}^{128}}{\overline{L}_{6,6}^{128}} \frac{L_{24}^{64}}{\overline{L}_{12,12}^{64}} \frac{L_{48}^{32}}{\overline{L}_{24,24}^{32}} \frac{L_{96,1}^{16}}{\overline{L}_{48,48}^{16}} \frac{L_{194,1}^{8}}{\overline{L}_{97,97}^{8}} \frac{L_{390,1}^{4}}{\overline{L}_{195,195}^{4}} \frac{L_{782,1}^{2}}{\overline{L}_{391,391}^{2}} \frac{L_{1566,1}}{\overline{L}_{783,783}}$ | 21 |
| iii: | $f_1^{1567} \frac{L_{1,1}^{512}}{L_2^{512}} \frac{L_{2,1}^{512}}{\overline{L}_{3,3}^{256}} \frac{L_{6,6}^{128}}{\overline{L}_{12,12}^{64}} L_{24,24}^{32} \frac{L_{96,1}^{16}}{\overline{L}_{48,48}^{16}} \frac{L_{194,1}^{8}}{\overline{L}_{97,97}^{8}} \frac{L_{390,1}^{4}}{\overline{L}_{195,195}^{4}} \frac{L_{782,1}^{2}}{\overline{L}_{391,391}^{2}} \frac{L_{1566,1}}{\overline{L}_{783,783} L_{1567}}$ | 17 |

There is a typo of "Compute $f_{191}$" in [4] where $f_1^{191} h_{1,1}^{64} \frac{h_{4,1}^{32}}{\overline{h}_{2,2}^{32}} \frac{h_{5,5}^{16}}{h_{10}^{16}} \frac{h_{10,1}^{16}}{\overline{h}_{5,5}^{16}} \frac{h_{22,1}^{8}}{\overline{h}_{11,11}^{8}} \frac{h_{46,1}^{4}}{\overline{h}_{23,23}^{4}} \frac{h_{94,1}^{2}}{\overline{h}_{47,47}^{2}} \frac{h_{190,1}}{\overline{h}_{95,95}}$ should be $f_1^{191} h_{1,1}^{64} \frac{h_{4,1}^{32}}{\overline{h}_{2,2}^{32}} \frac{h_{10,1}^{16}}{\overline{h}_{5,5}^{16}} \frac{h_{22,1}^{8}}{\overline{h}_{11,11}^{8}} \frac{h_{46,1}^{4}}{\overline{h}_{23,23}^{4}} \frac{h_{94,1}^{2}}{\overline{h}_{47,47}^{2}} \frac{h_{190,1}}{\overline{h}_{95,95}}$.

## 5. Concluding remarks

We have proposed a refinement to the *BMX* algorithms. Roughly speaking, our refinement can reduce more vertical lines than *BMX*-1 (*BMX*-2) by an amount of $H(n)$ ($H_0(n)$, respectively) in the best cases. Therefore, the saving in the number of multiplications of our algorithm is more than that of the *BMX* algorithms. Moreover, if $n$ is a Solinas number then our refinement has better performance than

*BMX* algorithms. We believe that our refinement is optimal in the sense that no other segment of the order $n$ can reduce more lines.

## References


1. D. Boneh, X. Boyen, and H. Shacham, *Short Group Signatures*, *CRYPTO 2004,* LNCS 3152, pp.41-55, Springer-Verlag, 2004.
2. D. Boneh, and M. Franklin, *Identity-base encryption from the Weil pairing*, *CRYPTO 2001,* LNCS 2139, pp.213-239, Springer-Verlag, 2001.
3. D. Boneh, B. Lynn, and H. Shacham, *Short signature from the Weil pairing*, *ASIACRYPT 2001,* LNCS 2248, pp.514-532, Springer-Verlag, 2001.
4. I.F. Blake, V.K.Murty, and G. Xu, *Refinement of Miller's algorithm for computing the Weil/Tate pairing*, Journal of Algorithms. Vol.58, pp.134-149, 2006.
5. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott, *Efficient algorithms for pairing-based cryptosystems*, *CRYPTO 2002,* LNCS 2442, pp.354-368, Springer-Verlag, 2002.
6. K. Eisenträger, K.Lauter, and P.L. Montgomery, *Fast elliptic curve arithmetic and improved Weil pairing evaluation*, *CT-RSA 2003,* LNCS 2612, pp. 343-354, Springer-Verlag, 2003.
7. A. Joux, *A one round protocol for tripartite Diffie-Hellman*, *ANTS IV,* LNCS 1838, pp.385-393, Springer-Verlag, 2000.
8. X. Li, and K. Chen, *Identity based proxy-signcryption scheme from pairings*, *IEEE-SCC 2004,* Sept. pp.494-497, 2004.
9. A.J. Menezes, T. Okamoto, and S.A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Tran. Inform. Theory, vol.39 pp.1639-1646, 1993.
10. A.J. Menezes, *Elliptic curve cryptosystems*, Kluwer Academic Publishers, 1993.
11. V. Miller, *Short programs for functions on curve," unpublished manuscript*, 1986.
12. D. Nalla and K.C. Reddy, *Signcryption scheme for identity-based cryptosystems*, Cryptology ePrint Archive, Report 2003/066, 2003.
13. R. Sakai, K. Ohgishi, and M. Kasahara, *Cryptosystems based on pairing*, *SCIS 2000* Okinawa, Japan, Jan. pp.26-28, 2000.
14. J.H. Silverman, *The Arithmetic of Elliptic Curves*, Grad. Texts in Math., vol.106, Springer, New York, 1986.
15. N.P. Smart, *An identity based authenticated key agreement protocol based on Weil pairing*, *Electronics Letters* Vol.38, pp.630-632, 2002.


## 6. Appendix A

**Algorithm 5** (*Segmentation*).
INPUT: Integer $n = \sum_{i=0}^{m} b_i 2^i$ with $b_i \in \{0, 1\}$ and $b_m = 1$.
OUTPUT: $\{B_i, B_{i-1}, \ldots, B_0\}$, $B_t \in \{l_0, l_1 \ldots, l_5\} \times \mathbb{N}$.

**Function** *Tracker*$(x, y)$
  **while** $x \neq m$ **and** $b_x \neq y$
    **do** $x \leftarrow x + 1$**;**
  **if** $x = m$ **then return -1**
  **else return** $x$

  $i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$;
  **while** $j < m - 1$ **do**
    **if** $(b_{j+1}, b_j) = (0, 0)$ **then**  $\triangleright$ **case 1.**
      $k \leftarrow Tracker(j + 2, 1)$;
      **if** $k = -1$ **then**
        $B_i \leftarrow (l_0, m - j)$; $j \leftarrow m$;
      **else**
        $B_i \leftarrow (l_0, k - j - 1)$; $j \leftarrow k - 1$; $i \leftarrow i + 1$;

```
    if (b_{j+1}, b_j) = (0,1) then      ▷ case 2.
        if i − 1 > 0 and B_{j−1} = (l_2, α) then
            B_{i−1} ← (l_2, α + 1); j ← j + 2;
            if j = m then i ← i − 1;
        else
            B_i ← (l_2, 1); j ← j + 2;
            if j ≠ m then i ← i + 1;

    if (b_{j+1}, b_j) = (1,0) then      ▷ case 3.
        k ← Tracker(j + 2, 0);
        if k = −1 then
            B_i ← (l_4, m − j − 1); j ← m;
        else
            B_i ← (l_5, k − j − 1); j ← k + 1;
            if j ≠ m then i ← i + 1;

    if (b_{j+1}, b_j) = (1,1) then      ▷ case 4.
        k ← Tracker(j + 2, 0);
        if k = −1 then
            B_i ← (l_1, m − j); j ← m;
        else
            B_i ← (l_3, k − j); j ← k + 1;
            if j ≠ t then i ← i + 1;
end-while;
if j = m then return {B_i, B_{i−1}, ..., B_0}
else if b_{m−1} = 1 then B_i ← (l_1, 1);
    else B_i ← (l_0, 1);
return {B_i, B_{i−1}, ..., B_0}
```

Department of Computer Science, National Chung-Hsing University, 250 Kuo-Kuang Road, Taichung 402, Taiwan, ROC

*E-mail address*: s9056001@cs.nchu.edu.tw

Department of Computer Science, National Chung-Hsing University, 250 Kuo-Kuang Road, Taichung 402, Taiwan, ROC

*Current address*: Department of Computer Science, National Chung-Hsing University, 250 Kuo-Kuang Road, Taichung 402, Taiwan, ROC, Tel.: +886-422840497-912; fax: +886-422853869.

*E-mail address*: gbhorng@cs.nchu.edu.tw

Department of Information Management, Hsiuping Institute of Technology,, No.11, Gungye Rd., Dali City, Taichung 412, Taiwan, ROC

*E-mail address*: chendy@mail.hit.edu.tw