

# Efficient Computation of Tate Pairing in Projective Coordinate Over General Characteristic Fields

Sanjit Chatterjee, Palash Sarkar and Rana Barua

Cryptology Research Group  
Applied Statistics Unit  
Indian Statistical Institute  
203, B.T. Road  
Kolkata 700 108, India  
{sanjit.t,palash,rana}@isical.ac.in

**Abstract.** We consider the use of Jacobian coordinates for Tate pairing over general characteristics. The idea of encapsulated double-and-line computation and add-and-line computation has been introduced. We also describe the encapsulated version of iterated doubling. Detailed algorithms are presented in each case and memory requirement has been considered. The inherent parallelism in each of the algorithms have been identified leading to optimal two-multiplier algorithm. The cost comparison of our algorithm with previously best known algorithms shows an efficiency improvement of around 33% in the general case and an efficiency improvement of 20% for the case of the curve parameter  $a = -3$ .

**Keywords :** Tate pairing, Jacobian coordinate, efficient implementation.

## 1 Introduction

Pairing based cryptography is a new way of constructing public key protocols. Initially, bilinear maps were used for attacking the discrete logarithm problem on elliptic curve groups [14, 5]. Starting with the initial work of Joux [10], Boneh-Franklin [2], etc. design of pairing based public key protocols have received a great deal of attention from the research community. See [3] for a recent survey of such protocols.

Implementation of pairing based protocols require efficient algorithms for computing pairings. An initial breakthrough in this direction has been made in [1] and [6], which introduced some nice optimisation ideas leading to dramatic improvement in pairing computation time. Since then, there have been quite a few papers on implementation aspects. Almost all of the implementation work have focussed on Tate pairing as it is faster than Weil pairing.

**OUR CONTRIBUTIONS:** We consider elliptic curves over large prime fields having embedding degree 2. For such fields, we consider the use of Jacobian coordinates for Tate pairing computation. The new idea that we introduce is encapsulated double-and-line computation and encapsulate add-and-line computation. We also describe encapsulated version of iterated double and line computation.

From an implementation point of view, we divide curves of the form  $y^2 = x^3 + ax + b$  into three cases – small  $a$ ,  $a = -3$  and the case where  $a$  is a general element of the field. In each case, we present detailed algorithm for pairing computation. To the best of our knowledge, such a detailed presentation of projective coordinate algorithms have not been reported earlier. We consider the memory requirement for our algorithms, a feature which has not been seriously considered earlier.

For hardware applications having special purpose crypto co-processors, it might be desirable to consider parallel versions of the algorithms. We identify the inherent parallelism in our algorithms and two-multiplier parallel version of our algorithms are optimal with respect to the number of parallel rounds.

In comparison with earlier work, we are able to obtain approximately 33% speed-up over the best known algorithm [8] for the case of general  $a$ . In the case  $a = -3$  and for non-supersingular curves, the recent work by Scott [16] provides the most efficient algorithm. In comparison, for the case  $a = -3$ , we are able to obtain approximately 20% speed-up over the algorithm in [16].

RELATED WORK: The important work in [1] and [6] has been mentioned before. Projective coordinates were seriously considered by Izu and Takagi [8], where mainly non-supersingular curves with large embedding degrees were considered. Further, projective coordinates in conjunction with non-supersingular curves with embedding degree 2 were also considered in the work by Scott [16] mentioned earlier. A recent work by Scott and Baretto [17] considers the issue of computing trace of pairings. This paper also describes a laddering algorithm for exponentiation in  $\mathbb{F}_{p^2}$  based on Lucas sequences. For general characteristics, this exponentiation algorithm is the fastest known and has to be used with the algorithm that we develop. The algorithm proposed by Eisenträger et. al. [4] uses the double-add trick with parabolas for fast computation of pairing in affine coordinates. There are several other works on Tate pairing computation. However, most of these work with affine coordinates and over characteristic three. Hence, they are not much relevant in the present context and therefore are not discussed here.

## 2 Preliminaries

We discuss background material for Tate pairing, choice of curves and NAF representation.

### 2.1 The Tate pairing

We first discuss how to compute the (modified) Tate pairing. Let  $p$  be an odd prime,  $\mathbb{F}_p$  the corresponding finite field with  $p$  elements,  $E$  is an elliptic curve over  $\mathbb{F}_p$ . Let  $r$  be a large prime divisor of  $(p + 1)$ , such that  $r$  is coprime to  $p$  and for some  $k > 0$ ,  $r | p^k - 1$  but  $r \nmid p^s - 1$  for any  $1 \leq s < k$ ;  $k$  is called the embedding degree (MOV degree). Suppose  $P$  is a point of order  $r$  on the elliptic curve  $E(\mathbb{F}_p)$  and  $Q$  is a point of same order on the elliptic curve  $E(\mathbb{F}_{p^k})$ ,

linearly independent of  $P$ . We denote the (modified) Tate pairing of order  $r$  as  $e_r(P, Q) \in \mathbb{F}_{p^k}$ .  $e_r(P, Q)$  is defined in terms of divisors of a rational function. A divisor is a formal sum:  $D = \sum_{P \in E} a_P \langle P \rangle$ , where  $P \in E(\mathbb{F}_p)$ . The degree of a divisor  $D$  is  $\deg(D) = \sum_{P \in E} a_P$ . The set of divisors forms an abelian group by the addition of their coefficients in the formal sum. Let  $f$  be a (rational) function on  $E$ , then the divisor of  $f$ ,  $\langle f \rangle = \sum_P \text{ord}_P(f) \langle P \rangle$ , where  $\text{ord}_P(f)$  is the order of the zero or pole of  $f$  at  $P$ . A divisor  $D = \sum_{P \in E} a_P \langle P \rangle$  is called a principal divisor if and only if it is a divisor of degree 0 (zero divisor) and  $\sum_{P \in E} a_P P = \mathcal{O}$ . If  $D$  is principal then there is some function  $f$  such that  $D = \langle f \rangle$ . Two divisors  $D_1$  and  $D_2$  are said to be equivalent if  $D_1 - D_2$  is a principal divisor. Let  $\mathcal{A}_P$  be a divisor equivalent to  $\langle P \rangle - \langle \mathcal{O} \rangle$  (similarly  $\mathcal{A}_Q$ ). Then it is easy to see that  $r\mathcal{A}_P$  is principal; thus there is a rational function  $f_P$  with  $\langle f_P \rangle = r\mathcal{A}_P = r\langle P \rangle - r\langle \mathcal{O} \rangle$ . The (modified) Tate pairing of order  $r$  is defined as –

$$e_r(P, Q) = f_P(\mathcal{A}_Q)^{(p^k - 1)/r}.$$

To compute  $f_P(\mathcal{A}_Q)$ ,  $Q \neq \mathcal{O}$  one uses Miller's algorithm [12]. Let  $f_a$  be a (rational) function with divisor  $\langle f_a \rangle = a\langle P \rangle - \langle aP \rangle - (a - 1)\langle \mathcal{O} \rangle$ ,  $a \in \mathbb{Z}$ . It can be shown that  $f_{2a}(Q) = f_a(Q)^2 \cdot h_{aP, aP}(Q) / h_{2aP}(Q)$  where,  $h_{aP, aP}$  is the line tangent to  $E(\mathbb{F}_p)$  at  $aP$ , it intersects  $E(\mathbb{F}_p)$  at the point  $-2aP$ , and  $h_{2aP}$  is the (vertical) line that intersects  $E(\mathbb{F}_p)$  at  $2aP$  and  $-2aP$ . Now,  $\langle f_r \rangle = r\langle P \rangle - \langle rP \rangle - (r - 1)\langle \mathcal{O} \rangle = \langle f_P \rangle$ , since  $rP = \mathcal{O}$ . Given  $P$  and the binary representation of  $r$ , Miller's algorithm computes  $f_P(Q) = f_r(Q)$  in  $\lg r$  steps by the standard double-and-add through line-and-tangent method for elliptic curve scalar multiplication. Under the condition,  $r \nmid (p - 1)$  we can further have  $e_r(P, Q) = f_P(Q)^{(p^k - 1)/r}$  for  $Q \neq \mathcal{O}$ , as long as  $k > 1$ .

In the implementation of Tate pairing over  $E(\mathbb{F}_p)$ , the usual practice is to take  $Q \in E(\mathbb{F}_p)$  of order  $r$  and then use a distortion map  $\phi()$ , to get a point  $\phi(Q) \in E(\mathbb{F}_{p^k})$  of order  $r$  which is linearly independent of  $P$ . A major finding in [1] is that, for particular choices of the curve parameters and distortion map  $\phi()$  we can freely multiply or divide the intermediate result of pairing computation by any  $\mathbb{F}_p$  element and consequently completely ignore the denominator part in the computation of Tate pairing.

## 2.2 Choice of Curves

Let  $E_1$  be the elliptic curve given by the equation

$$y^2 = x^3 + ax$$

over  $\mathbb{F}_p$ .  $E_1$  is super-singular if  $p \equiv 3 \pmod{4}$ . For these curves, the curve order is  $\#E_1(\mathbb{F}_p) = p + 1$  and embedding degree is  $k = 2$ . For such curves, a distortion map [1] is  $\phi(x, y) = (-x, iy) \in \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$  with  $i^2 = -1$ . Let  $r$  be the integer, for which we wish to compute  $e_r(\cdot, \cdot)$ . Then  $r | (p + 1)$  and the final

powering in Tate pairing computation is of the form  $(p^2 - 1)/r$ . As observed in [1], this implies  $(p - 1)$  divides the final powering exponent.

Let,  $E_2$  be the elliptic curve given by the equation

$$y^2 = x^3 - 3x + B$$

over  $\mathbb{F}_p$ ,  $p \equiv 3 \pmod{4}$ . Scott in his recent paper [16] considered this form of non super-singular EC with embedding degree,  $k = 2$  with  $\#E_2(\mathbb{F}_p) = p + 1 - t$ , where  $t$  is the trace of Frobenius [11]. It is shown in [16] that the  $r$  for which  $e_r(\cdot, \cdot)$  is computed over  $E_2$  also satisfies  $r|(p + 1)$  and hence again  $p - 1$  divides the final powering exponent of Tate pairing.

Thus for both  $E_1$  and  $E_2$  the following observation holds. Since  $x^{p-1} = 1$  for any  $x \in \mathbb{F}_p$ , this implies that we can freely multiply or divide any intermediate Tate pairing result by any nonzero element of  $\mathbb{F}_p$ . This has been previously used to improve the efficiency of Tate pairing computation in affine coordinates. In Section 3, we point out the importance of this observation in the context of projective coordinates.

Note that, for  $E_1$  as well as  $E_2$ , the embedding degree is  $k = 2$  and the elements of the quadratic extension field ( $\mathbb{F}_{p^2}$ ) is represented as  $a + ib$ , where  $a, b \in \mathbb{F}_p$  and  $i$  is the *square root* of a quadratic non-residue. For  $p \equiv 3 \pmod{4}$  we can further have  $i^2 = -1$ . Essentially, the same algorithm that we develop for  $E_1$  also applies to  $E_2$  by evaluating  $e(\cdot, \cdot)$  at  $P$  and  $(-x_Q, iy_Q)$ .

### 2.3 Non-Adjacent Form Representation

The Non-Adjacent Form (NAF) representation of an integer has been suggested for elliptic curve scalar multiplication. In this representation, the digits  $\{0, \pm 1\}$  are used to represent an integer with the property that no two adjacent digits are non-zero. The advantage is that, on an average, the number of non-zero digits is one-third of the length of the representation, while it is one-half in the case of binary representation. For details of NAF representation we refer the reader to [7].

For Tate pairing applications, the representation of  $r$  should be sparse, i.e., the number of non-zero digits should be small. The NAF representation is sparser than the corresponding binary representation. Hence in our algorithms, we work entirely with the NAF representation.

## 3 Encapsulated Computation

In the computation of Tate pairing one needs to perform an implicit scalar multiplication of the EC point  $P$ . For this, as well as for computation of the line function  $h_{aP, aP}(\cdot)$  one requires to perform base field inversion. But inversion for large characteristic is quite costly. The standard method to avoid inversion is to move from affine to projective coordinate system. Among the different available projective coordinate systems, the Jacobian gives the best result. In [8] the

authors suggested to take the so called *simplified Jacobian-Chudnovsky coordinate*  $J^s$  as they store  $(X, Y, Z, Z^2)$  instead of  $(X, Y, Z)$ . However, we have found out that if one encapsulates EC addition/doubling with line computation then there is no need to additionally store  $Z^2$  – one can simply work in the Jacobian coordinate. Here we give the explicit formulae required for the encapsulated computation of double/add-and-line computation. In what follows, by [M] and [S], we respectively denote the cost of one multiplication and one squaring in  $\mathbb{F}_p$ .

### 3.1 Encapsulated point doubling and line computation

Here  $P = (X_1, Y_1, Z_1)$  correspond to  $(X_1/Z_1^2, Y_1/Z_1^3)$  in affine coordinate. We encapsulate the computation of  $2P$  given  $P$  together with the computation corresponding to the associated line.

**Point Doubling** : From the EC point doubling rule we have the following formula:

$$\begin{aligned} X'_3 &= \frac{(3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2}{4Y_1^2Z_1^2} \\ Y'_3 &= \frac{3X_1^2 + aZ_1^4}{2Y_1Z_1} \left( \frac{X_1}{Z_1^2} - X'_3 \right) - \frac{Y_1}{Z_1^3} \\ X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned}$$

Using temporary variables, we compute:

$$\begin{aligned} 1. t_1 &= Y_1^2; & 2. t_2 &= 4X_1t_1; & 3. t_3 &= 8t_1^2; \\ 4. t_4 &= Z_1^2; & 5. t_5 &= 3X_1^2 + aZ_1^4; & 6. X_3 &= t_5^2 - 2t_2; \\ 7. Y_3 &= t_5(t_2 - X_3) - t_3; & 8. Z_3 &= 2Y_1Z_1. \end{aligned}$$

So, we require  $6[S] + 4[M]$  for EC doubling. Now consider  $t_5$ . If  $a$  is a general element of  $\mathbb{F}_p$ , then we have to count the multiplication  $a \times (Z_1^4)$ . However, if  $a$  is small, i.e., it can be represented using only a few (say  $\leq 8$ ) bits, then we do not count this multiplication. In this case,  $aZ_1^4$  can be obtained summing  $Z_1^4$  a total of  $a$  times. This reduces the operation count to  $6[S] + 3[M]$ . Further, if  $a = -3$ , then  $t_5 = 3(X_1 - Z_1^2)(X_1 + Z_1^2) = 3(X_1 - t_4)(X_1 + t_4)$  and the operation count reduces to  $4[S] + 4[M]$ . These facts are known and can be found in [7].

**Line Computation:** Note that, the slope  $\mu$  of  $h_{P,P}$ , the line through  $P$  and  $-2P$ , is  $\mu = t_5/Z_3$ . So,

$$h_{P,P}(x, y) = \left( y - \frac{Y_1}{Z_1^3} \right) - \mu \left( x - \frac{X_1}{Z_1^2} \right).$$

Hence,  $h_{P,P}(-x_Q, iy_Q) = (y_Q i - \frac{Y_1}{Z_1^3}) + \mu(x_Q + \frac{X_1}{Z_1^2})$ .

By defining  $g_{P,P}(x, y) = (2Y_1Z_1^3)h_{P,P}(x, y)$ , we get,

$$\begin{aligned} g_{P,P}(-x_Q, iy_Q) &= (2Y_1Z_1)Z_1^2 y_Q i - 2Y_1^2 + (3X_1^2 + aZ_1^4)(Z_1^2 x_Q + X_1) \\ &= Z_3 t_4 y_Q i - (2t_1 - t_5(t_4 x_Q + X_1)) \end{aligned}$$

The ultimate result is raised to the power  $(p^2 - 1)/r$ , where  $r|(p + 1)$  (see Section 2.1). Thus we have to compute

$$\begin{aligned} (h_{P,P}(-x_Q, iy_Q))^{(p^2-1)/r} &= ((h_{P,P}(-x_Q, iy_Q))^{(p-1)})^{(p+1)/r} \\ &= ((g_{P,P}(-x_Q, iy_Q)/(2Y_1Z_1^3))^{(p-1)})^{(p+1)/r} \\ &= ((g_{P,P}(-x_Q, iy_Q))^{(p-1)})^{(p+1)/r}. \end{aligned}$$

The last equation holds since  $(2Y_1Z_1^3) \in \mathbb{F}_p$  and consequently  $(2Y_1Z_1^3)^{p-1} = 1$ . Thus, we can work entirely with  $g_{P,P}(-x_Q, iy_Q)$  instead of  $h_{P,P}(-x_Q, iy_Q)$ . Since  $t_1, t_4$  and  $t_5$  have already been computed,  $g_{P,P}(-x_Q, iy_Q)$  can be obtained using 4 additional multiplications.

Hence, *encapsulated point doubling and line computation* requires  $8[M] + 6[S]$ . In the case  $a$  is small, this cost is  $7[M]+6[S]$  and for the case  $a = -3$ , this cost is  $8[M]+4[S]$ .

### 3.2 Encapsulated (mixed) point addition and line computation

We encapsulate the computation of  $P + R$  given  $P$  in affine and  $R$  in Jacobian together with the computation corresponding to the associated line.

**Mixed Addition:** Given  $R = (X_1, Y_1, Z_1)$  and  $P = (X, Y, 1)$  we compute  $R + P = (X_3, Y_3, Z_3)$  as follows.

$$\begin{aligned} X'_3 &= \left( \frac{Y - \frac{Y_1}{Z_1^3}}{X - \frac{X_1}{Z_1^2}} \right)^2 - \frac{X_1}{Z_1^2} - X \\ &= \left( \frac{YZ_1^3 - Y_1}{(XZ_1^2 - X_1)Z_1} \right)^2 - \frac{X_1}{Z_1^2} - X \\ Y'_3 &= \left( \frac{YZ_1^3 - Y_1}{(XZ_1^2 - X_1)Z_1} \right) \left( \frac{X_1}{Z_1^2} - X'_3 \right) - \frac{Y_1}{Z_1^3} \\ X_3 &= X'_3 Z_3 \\ &= (YZ_1^3 - Y_1)^2 - X_1(XZ_1^2 - X_1)^2 - X(XZ_1^2 - X_1)^2 Z_1^2 \\ &= (YZ_1^3 - Y_1)^2 - (XZ_1^2 - X_1)^2 (X_1 + XZ_1^2) \\ Y_3 &= Y'_3 Z_3 \\ &= (YZ_1^3 - Y_1)((XZ_1^2 - X_1)^2 X_1 - X_3) - Y_1(XZ_1^2 - X_1)^3 \\ Z_3 &= (XZ_1^2 - X_1)Z_1 \end{aligned}$$

Using temporary variables we compute:

- |                                   |                                       |                       |
|-----------------------------------|---------------------------------------|-----------------------|
| 1. $t_1 = Z_1^2;$                 | 2. $t_2 = Z_1 t_1;$                   | 3. $t_3 = X t_1;$     |
| 4. $t_4 = Y t_2;$                 | 5. $t_5 = t_3 - X_1;$                 | 6. $t_6 = t_4 - Y_1;$ |
| 7. $t_7 = t_5^2;$                 | 8. $t_8 = t_5 t_7;$                   | 9. $t_9 = X_1 t_7;$   |
| 10. $X_3 = t_6^2 - (t_8 + 2t_9);$ | 11. $Y_3 = t_6(t_9 - X_3) - Y_1 t_8;$ | 12. $Z_3 = Z_1 t_5.$  |

Hence, we require  $3[S] + 8[M]$  for EC point addition. See [7] for details.

**Line Computation:** Note that, the slope  $\mu$  of  $h_{R,P}$ , the line through  $R$  and  $P$  is  $\mu = t_6/Z_3$ . So,

$$h_{R,P}(x, y) = (y - Y) - \mu(x - X).$$

Hence,  $h_{R,P}(-x_Q, iy_Q) = (y_Q i - Y) + \mu(x_Q + X)$ . Define  $g(x, y)$  as  $g(x, y) = Z_3 h_{R,P}(x, y)$ . Thus, we get

$$g_{R,P}(-x_Q, iy_Q) = Z_3 y_Q i - (Z_3 Y - t_6(x_Q + X))$$

As explained in the case of doubling, we can simply work with  $g_{R,P}$  instead of  $h_{R,P}$ . Since we have already computed  $t_6$  and  $Z_3$  during point addition,  $g_{R,P}(-x_Q, iy_Q)$  can be computed using additional three multiplications. Hence, *encapsulated point addition and line computation* requires  $11[M] + 3[S]$ .

## 4 Algorithm

We consider three situations:  $a = -3$ ;  $a$  small (i.e., multiplication by  $a$  need not be counted); and the case where  $a$  is a general element of  $\mathbb{F}_p$ . For the first two cases, double-and-add algorithm is considered. For the general case, we adopt an iterated doubling technique used by Izu and Takagi [8].

### 4.1 Double-and-Add

We slightly modify the Miller's algorithm as improved in [1]. We will call this algorithm the modified BKLS algorithm. In the algorithm the NAF representation of  $r$  is taken to be  $r_t = 1, r_{t-1}, \dots, r_0$ .

**Algorithm 1 (Modified BKLS Algorithm):**

1. set  $f = 1$  and  $V = P$
2. for  $i = t - 1$  downto 0 do
3.      $(u, V) = \text{EncDL}(V)$ ;
4.     set  $f = f^2 \times u$ ;
5.     if  $r_i \neq 0$ , then
6.          $(u, V) = \text{EncAL}(V, r_i)$ ;
7.         set  $f = f \times u$ ;
8.     end if;
9. end for;
10. return  $f$ ;

**end Algorithm 1.**

The subroutine  $\text{EncDL}(V)$  performs the computation of Section 3.1 and returns  $(g_{V,V}(\phi(Q)), 2V)$ . The subroutine  $\text{EncAL}(V, r_i)$  takes  $V$  and  $r_i$  as input. If  $r_i = 1$ , it performs the computation of Section 3.2 and returns  $(g_{V,P}(\phi(Q)), V + P)$ ; if  $r_i = -1$ , it first negates the point  $P = (\alpha, \beta)$  to obtain  $P' = -P = (\alpha, -\beta)$ , then it performs the computation of Section 3.2 with  $P'$  instead of  $P$  and returns  $(g_{V,-P}(\phi(Q)), V - P)$ . The correctness of the algorithm follows easily from the correctness of the original BKLS algorithm.

We consider the cost. The subroutine `EncDL` is invoked a total of  $t$  times while `EncAL` is invoked a total of  $s$  times where  $s$  is the Hamming weight of  $r_{t-1} \dots r_0$ . The cost of updation in Line 4 is one  $\mathbb{F}_{p^2}$  squaring and one  $\mathbb{F}_{p^2}$  multiplication. These operations can be completed in five  $\mathbb{F}_p$  multiplications (see [17]). The cost of updation in Line 7 is three  $\mathbb{F}_p$  multiplications. The cost of `EncDL` depends upon the value of the curve parameter  $a$ . We analyse the total cost for the following two cases.

*Case  $a = -3$ :*

- Cost of `EncDL` is  $8[M]+4[S]$ .
- Cost of update in line 4 is  $5[M]$ .
- Cost of `EncAL` is  $11[M]+3[S]$ .
- Cost of update in line 7 is  $3[M]$ .
- Total cost is  $t(13[M]+4[S]) + s(14[M]+3[S])$ .

*Case  $a$  is small:*

- Cost of `EncDL` is  $7[M]+6[S]$ .
- Cost of update in line 4 is  $5[M]$ .
- Cost of `EncAL` is  $11[M]+3[S]$ .
- Cost of update in line 7 is  $3[M]$ .
- Total cost is  $t(12[M]+6[S]) + s(14[M]+3[S])$ .

## 4.2 Iterated Doubling

In the case where we have to consider the multiplication by the curve parameter  $a$ , we employ the technique of iterated doubling to reduce the total number of operations. As before we consider the NAF representation of  $r$ . We write the NAF representation of  $r$  as

$$(r_t = 1, r_{t-1}, \dots, r_0) = (l_s = 1, 0^{w_{s-1}}, l_{s-1}, \dots, 0^{w_0}, l_0)$$

where the  $l_i$ 's are  $\pm 1$ . The following algorithm is an iterated doubling version of the modified BKLS algorithm described in Section 4.1. The points  $P = (\alpha, \beta)$  and  $Q = (x_Q, y_Q)$  are globally available.

### Algorithm 2 (iterated doubling):

**Input:**  $P = (\alpha, \beta, 1)$  in Jacobian coordinates;  $Q = (x_Q, y_Q)$ .

**Output:**  $f_P(\phi(Q))$ .

1. Set  $f = 1$ ;  $g = 1$ ;
2.  $X = \alpha$ ;  $Y = \beta$ ;  $Z = 1$ ; set  $R = (X, Y, Z)$ ;
3. for  $j = s - 1$  down to 0
4.      $(f, R) = \text{Encdbl}(f, R, w_j)$ ;
5.      $(g, R) = \text{EncAL}(R, l_j)$ ;
6.      $f = f \times g$ ;
7. end for;
8. return  $f$ ;

**end Algorithm 2.**

The Subroutine EncAL has already been discussed in Section 4.1. We now describe Subroutine Encldbl.

**Subroutine Encldbl**

**Input:**  $R = (X, Y, Z)$ ,  $f$  and  $w$ .

**Output:** updated  $f$  and  $2^{w+1}R$ .

1.  $t_1 = Y^2; t_2 = 4Xt_1; t_3 = 8t_1^2; t_4 = Z^2; w = aZ^4; t_5 = 3X^2 + w;$
2.  $A = -(2t_1 + t_5(t_4x_Q - X)); X = t_5^2 - 2t_2;$   
 $Y = t_5(t_2 - X) - t_3; Z = 2YZ; B = Zt_4y_Q;$
3.  $f = f^2 \times (A + iB);$
4. for  $j = 1$  to  $w$  do
5.  $w = 2t_3w; t_1 = Y^2; t_2 = 4Xt_1; t_3 = 8t_1^2; t_4 = Z^2; t_5 = 3X^2 + w;$
6.  $A = -(2t_1 + t_5(t_4x_Q - X)); X = t_5^2 - 2t_2;$   
 $Y = t_5(t_2 - X) - t_3; Z = 2YZ; B = Zt_4y_Q;$
7.  $f = f^2 \times (A + iB);$
8. end for;
9.  $R = (X, Y, Z);$
9. return  $(f, R);$

**end Subroutine Encldbl.**

Algorithm 2 is essentially the same as Algorithm 1 except for the use of iterated doubling. The technique of iterated doubling is considered to reduce computation cost but does not affect the correctness of the algorithm. We consider the cost of the algorithm. As before let the Hamming weight of  $r_{t-1}, \dots, r_0$  be  $s$ .

- Steps 5 and 6 of Algorithm 2 are invoked  $s$  times. The total cost of these two steps is  $s(14[M]+3[S])$ .
- Step 4 of Algorithm 2 is invoked a total of  $s$  times. The cost of the  $j$ th invocation of Step 4 is computed as follows:
  - Cost of Steps 3 and 7 in Encldbl is  $5[M]$ .
  - Cost of Steps 1 and 2 in Encldbl is  $8[M]+6[S]$ .
  - Cost of Steps 5 and 6 in Encldbl is  $8[M]+5[S]$ .
  - Total cost of  $j$ th invocation of Encldbl is  $13[M]+6[S]+w_j(13[M]+5[S])=1[S]+(w_j + 1)(13[M]+5[S])$ .
- Total cost of Algorithm 2 is  $s(14[M]+3[S])+\sum_{j=0}^{s-1}(1[S]+(w_j + 1)(13[M]+5[S]))$   
 $=s(14[M]+4[S])+t(13[M]+5[S])$ .

### 4.3 Memory Requirement

The memory requirement of Algorithm 1 and Algorithm 2 are similar with Algorithm 2 requiring slightly more memory. We consider the memory requirement of Algorithm 2. To find the minimum memory requirement, first note that in Algorithm 2 we have to store and update  $f \in \mathbb{F}_{p^2}$  and  $X, Y, Z \in \mathbb{F}_p$  – they require  $5 \mathbb{F}_p$  storage space. We also need to store  $Q = (x_Q, y_Q)$ . In addition, we require some temporary variables to keep the intermediate results produced in the subroutines Encldbl and EncAL. These subroutines are called one after

another – we first call `Encdbl` and update  $f$  together with  $X, Y, Z$ , release the temporary variables and then call `EncAL` where these temporary variables can be reused. The maximum of the number of temporary variables required by the two subroutines determines the number of temporary variables required in Algorithm 2. We ran a computer program to separately find these requirements. Given a straight line code what the program essentially does is to exhaustively search (with some optimisations) for all possible execution paths and output the path pertaining to minimum number of temporary variables. This turns out to be 9 for `Encdbl`, while it is 7 for `EncAL`. So, at most we require to store 16  $\mathbb{F}_p$  elements.

#### 4.4 Parallelism

We first consider the parallelism in the encapsulated computations of Sections 3.1 and 3.2. While considering parallelism, we assume that a multiplier is used to perform squaring.

First we consider the case of encapsulated double and line computation. The situation in Section 3.1 has three cases – small  $a$ ,  $a = -3$  and general  $a$ . For the last case we use the iterated doubling technique of Section 4.2. We separately describe parallelism for the three cases. In each case, we first need to identify the multiplications which can be performed together. This then easily leads to parallel algorithms with a fixed number of multipliers.

**Small  $a$**  In this case, multiplication by  $a$  will be performed as additions. The multiplication levels are as follows.

Level 1 :  $t_1 = Y_1^2$ ;  $t_4 = Z_1^2$ ;  $X_1^2$ ;  $Z_3 = 2Y_1Z_1$ ; square  $f$ ;  
 Level 2 :  $t_2 = 4X_1t_1$ ;  $t_3 = 8t_1^2$ ;  $t_5 = 3X_1^2 + aZ_1^4$ ;  $t_6 = t_4x_Q$ ;  $t_7 = t_4y_Q$ ;  
 Level 3 :  $-(2t_1 + t_5(t_6 - X_1))$ ;  $X_3 = t_5^2 - 2t_2$ ;  $Y_3 = t_5(t_2 - X_3) - t_3$ ;  $Z_3t_7$ ;  
 Level 4 : update  $f$ .

**Case  $a = -3$**  In this case,  $t_5 = 3(X_1^2 - Z_1^4) = 3(X_1 - Z_1^2)(X_1 + Z_1^2)$ . The multiplication levels are as follows.

Level 1 :  $t_1 = Y_1^2$ ;  $t_4 = Z_1^2$ ;  $Z_3 = 2Y_1Z_1$ ; square  $f$ ;  
 Level 2 :  $t_2 = 4X_1t_1$ ;  $t_3 = 8t_1^2$ ;  $t_5 = 3(X_1 - t_4)(X_1 + t_4)$ ;  $t_6 = t_4x_Q$ ;  $t_7 = t_4y_Q$ ;  
 Level 3 :  $-(2t_1 + t_5(t_6 - X_1))$ ;  $X_3 = t_5^2 - 2t_2$ ;  $Y_3 = t_5(t_2 - X_3) - t_3$ ;  $Z_3t_7$ ;  
 Level 4 : update  $f$ .

**General  $a$**  In this case, Subroutine `Encdbl` is used. This consists of an initial part plus computation inside the *for* loop. The parallel version of both these parts are similar and we describe the parallel version of the loop computation. The multiplication levels are as follows.

Level 1 :  $w = 2t_3w$ ;  $t_1 = Y^2$ ;  $t_4 = Z^2$ ;  $X^2$ ;  $Z_3 = 2YZ$ ; square  $f$ ;  
 Level 2 :  $t_2 = 4Xt_1$ ;  $t_3 = 8t_1^2$ ;  $t_5 = 3X^2 + w$ ;  $t_6 = t_4x_Q$ ;  $t_7 = t_4y_Q$ ;  
 Level 3 :  $A = -(2t_1 + t_5(t_6 - X))$ ;  $X = t_5^2 - 2t_2$ ;  $Y = t_5(t_2 - X) - t_3$ ;  $Z_3t_7$ ;  
 Level 4 : update  $f$ .

In each of the above cases, with two multipliers the entire operation can be performed in 9 rounds and with four multipliers it can be performed in 5 rounds. Since the total number of operations is either 17 or 18 squarings and multiplications, the number of rounds is optimal for the given number of operations and given number of multipliers.

**Addition** We now consider the case of encapsulated add-and-line computation. See Section 3.2 for the details of the temporary variables and the operations. Here we mainly list the multiplication/squaring operations.

Level 1 :  $t_1 = Z_1^2$ ;  
 Level 2 :  $t_2 = Z_1t_1$ ;  $t_3 = Xt_1$ ;  
 Level 3 :  $t_4 = Yt_2$ ;  $t_7 = t_5^2$ ;  $Z_3 = Z_1t_5$ ;  
 Level 4 :  $t_8 = t_5t_7$ ;  $t_9 = X_1t_7$ ;  $X_3 = t_6^2 - (t_8 + 2t_9)$ ;  $Z_3y_Q$ ;  $Z_3Y$ ;  $t_6(x_Q - X)$ ;  
 Level 5 :  $Y_3 = t_6(t_9 - X_3) - Y_1t_8$ ; update  $f$ ;

There are a total of 17 multiplications including the update operation. Using two multipliers, these can be performed in 9 rounds. On the other hand, the four multiplier algorithm is sub-optimal in the number of rounds.

Thus, for parallel version of pairing computation algorithm, one obtains optimal two-multiplier algorithms for both doubling and addition. For doubling, the four multiplier algorithm is optimal, while for addition, the four multiplier algorithm is sub-optimal. However, the Hamming weight of  $r$  will be small and hence if we use four multipliers then the sub-optimal performance will be amortized over the length of the representation of  $r$  and will not be significantly reflected in the final cost analysis.

## 5 Comparison

For the purpose of comparison, we assume that  $r = (r_t = 1, r_{t-1}, \dots, r_0)$  is represented in NAF having length  $t$  and Hamming weight  $s$ .

The irrelevant denominator optimisation was introduced in [1]. Further, [1] uses affine representation. The total cost including point/line computation and updation is  $t(1[\text{I}]+8[\text{M}]+2[\text{S}])+s(1[\text{I}]+6[\text{M}]+1[\text{S}])$ , where  $[\text{I}]$  is the cost of inversion over  $\mathbb{F}_p$  and is at least  $30[\text{M}]$ , see [15].

Izu-Takagi [8] uses projective coordinates for pairing computation in general characteristics for large embedding degree  $k$ . They also consider the BKLS optimisations for supersingular curves with embedding degree  $k = 2$  for general  $a$ . They assume that one  $\mathbb{F}_{p^k}$  multiplication takes  $k^2[\text{M}]$ . For  $k = 2$ , this can be improved to  $3[\text{M}]$ . In the following calculation, we use this fact. Their cost for  $w$ -iterated doubling is  $6w[\text{M}]+4w[\text{S}]+13w[\text{M}]+(5w+1)[\text{S}]$  and addition is

$6[M]+16[M]+3[S]$ . Summing over  $w$ 's, the total cost comes to  $t(19[M]+9[S])+s(22[M]+4[S])$ .

The recent paper by Scott [16], also proposes the use of projective coordinates in the case  $a = -3$  for certain non-supersingular curves. The paper does not distinguish between multiplication and squaring. The total cost is  $21t[M]+22s[M]$ . In Table 1, we summarize the above costs along with the costs obtained by our algorithms for the various cases for the curve parameter  $a$ . The best case

**Table 1.** Cost Comparison. Note  $1[I] \geq 30[M]$  [15].

Method	Cost
BKLS [1] (affine)	$t(1[I]+8[M]+2[S])+s(1[I]+6[M]+1[S])$
Izu-Takagi [8] (general $a$ )	$t(19[M]+9[S])+s(22[M]+4[S])$
Scott [16] ( $a = -3$ )	$21t[M]+22s[M]$
Algorithm 1 ( $a$ small)	$t(12[M]+6[S])+s(14[M]+3[S])$
Algorithm 1 ( $a = -3$ )	$t(13[M]+4[S])+s(14[M]+3[S])$
Algorithm 2 (general $a$ )	$t(13[M]+5[S])+s(14[M]+4[S])$

occurs for Algorithm 1 with  $a = -3$ . Also the cases for Algorithm 1 for small  $a$  and Algorithm 2 are marginally slower than the best case. However, all three of these cases are much more efficient than any of the previous algorithms. The algorithms of Izu-Takagi [8] and Scott [16] are more efficient than the basic BKLS algorithm with affine coordinates.

For Tate pairing applications,  $r$  is generally chosen so that the Hamming weight  $s$  is small. On the other hand, for a general  $r$ , the Hamming weight  $s$  is approximately  $s = t/3$ . In either of these two situations, we summarize the superiority of our method as follows.

- Algorithm 1 with  $a = -3$  is approximately 20% faster compared to the algorithm by Scott.
- Algorithm 2 is approximately 33% faster compared to the algorithm by Izu and Takagi.

We consider the cost comparison to EC scalar multiplication. For the purpose of security, scalar multiplication has to be resistant to side channel attacks. One simple method of attaining resistance to simple power analysis is to use Coron's dummy addition using binary representation of multiplier. Under the (realistic) assumption that the length of the binary representation of the multiplier is equal to the length of the NAF representation of  $r$  for Tate pairing, the cost of dummy-addition countermeasure is  $t(2[M]+7[S])$  for the case of  $a = -3$ . This cost is comparable to the cost of Algorithm 1 for  $a = -3$  when  $s$  is at most around  $t/8$ . Again for practical situation  $r$  can usually be chosen so that  $s \leq t/8$ . Thus the efficiency of our algorithm is almost comparable to the efficiency of simple SPA resistant EC scalar multiplication. On the other hand, there is a wide

variety of techniques for EC scalar multiplication providing very efficient algorithms. Whether the cost of Tate pairing computation can be made comparable to the most efficient EC scalar multiplication is currently a challenging research problem.

## 6 Conclusion

In this paper, we have considered the use of Jacobian coordinates for Tate pairing computation in general characteristics. The main idea that we have introduced is encapsulated double-and-line computation and encapsulated add-and-line computation. We have also developed encapsulated version of iterated double algorithm. The algorithms are presented in details and memory requirement has been considered. Inherent parallelism in these algorithms have been identified leading to optimal two-multiplier parallel algorithms. Our algorithms lead to an improvement of around 33% over best known algorithm for the general case where the curve parameter  $a$  is an arbitrary field element. In the special case where  $a = -3$ , our techniques provide an efficiency improvement of around 20% over the previously best known algorithm.

## References

1. P. S. L. M. Barreto, H. Y. Kim, B. Lynn and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology - Crypto'2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354-368. Springer-Verlag, 2002.
2. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586-615, 2003.
3. R. Dutta, R. Barua and P. Sarkar. Pairing-based cryptography : A survey. Cryptology ePrint Archive, Report 2004/064, 2004. Available from <http://eprint.iacr.org/2004/064>.
4. K. Eisenträger, K. Lauter and P. L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. CT-RSA 2003, *Lecture Notes in Computer Science*, Vol. 2612, pp. 343-354, Springer-Verlag, (2003). (Also see <http://eprint.iacr.org/2002/112>).
5. G. Frey, M. Muller and H. Ruck, The Tate Pairing and the Discrete Logarithm applied to Elliptic Curve Cryptosystems, *IEEE Trans, Inf. Theory*, 45, No. 5(1999) pp. 1717-1719.
6. S. Galbraith, K. Harrison and D. Soldera. Implementing the Tate pairing. In *Algorithmic Number Theory Symposium - ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, Pages 324-337. Springer-Verlag, 2002.
7. D. R. Hankerson, A. J. Menezes and S. A. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004.
8. T. Izu and T. Takagi, Efficient computation of the Tate pairing for the Large MOV degree. 5th International Conference on Information Security and Cryptology, ICISC 2002, *Lecture Notes in Computer Science*, Vol. 2587, pp. 283-297, Springer-Verlag, (2003).
9. K. Itoh, M. Takenaka, N. Torii, S. Temma and Y. Kurihara, Fast implementation of public key cryptography on DSP TMS320C6201, CHES'99, LNCS 1717, pp 61-72, Springer-Verlag, 1999.

10. A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *Algorithmic Number Theory Symposium – ANTS IV*, volume 1838 of *Lecture Notes in Computer Science*, Pages 385-394. Springer-Verlag, 2000.
11. A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
12. V. S. Miller. Short programs for functions on curves. Unpublished manuscript, 1986. Available from <http://crypto.stanford.edu/miller/miller.pdf>
13. W. Mao and K. Harrison. Divisors, bilinear pairings and pairing enabled cryptographic applications, 2003. <http://hplbwww.hpl.hp.com/people/wm/research/pairing.pdf>.
14. A. Menezes, T. Okamoto and S. A. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, *IEEE Trans, Inf. Theory*, 39, No. 5(1993) 1639-1646.
15. A. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptology*. CRC Press, 1997
16. M. Scott. Computing the Tate Pairing, manuscript accepted for presentation in *CT-RSA 2005*.
17. M. Scott and P. S. L. M. Barreto. Compressed pairings. In *Advances in Cryptology - Crypto'2004*, volume 3152 of *Lecture Notes in Computer Science*, Spriger-Verlag, 2004. Also available in cryptology ePrint archive : Report 2004/032.