Compressed Jacobian Coordinates for OEF

Fumitaka Hoshino, Tetsutaro Kobayashi, and Kazumaro Aoki

NTT Information Sharing Platform Laboratories, NTT Corporation
{fhoshino,kotetsu,maro}@isl.ntt.co.jp

Abstract. This paper presents a new coordinate system for elliptic curves that accelerates the elliptic curve addition and doubling over an optimal extension field (OEF). Many coordinate systems for elliptic curves have been proposed to accelerate elliptic curve cryptosystems. This paper is a natural extension of these papers and the new coordinates are much faster when the elliptic curve is defined over an OEF. This paper also shows that the total computational cost is reduced by 28% when the elliptic curve is defined over $\mathbb{F}_{q^m}$, $q = 2^{61} - 1$ for $m = 5$ and the speed of a scalar multiplication on an elliptic curve becomes 41.9 μsec per operation on a 2.82-GHz Athlon 64 FX PC.

Keywords: elliptic curve cryptosystem, coordinate system, OEF, fast software implementation.

1 Introduction

Elliptic curve schemes such as EC-DSA and EC-ElGamal have been the focus of much attention since they provide smaller key sizes and faster operations compared to RSA. In particular, the use of an optimal extension field (OEF) [1] for software implementation has determined that an elliptic curve cryptosystem is faster than a public key cryptosystem based on modular exponentiations [1]. Moreover, elliptic curves over extension fields become much important because they are used in pairing based cryptosystems.

Points on an elliptic curve can be represented using different coordinate systems. Many coordinate systems have been studied to find a way to accelerate the computation of elliptic curve addition (ECADD) and doubling (ECDBL) calculations because the computational costs of ECADD and ECDBL are mainly determined by the coordinate systems. The most popular coordinates are affine coordinates, Jacobian coordinates, and its variation as shown in [2] and Appendix B. The best coordinate for fast software implementation depends on the computational environment. If an inversion is slower than 7.6 multiplications, Jacobian coordinates are faster than affine coordinates, for example. Previous implementations [1,3] used the Jacobian coordinates because multiplication algorithms are more efficient than inversion algorithms in many environments for an OEF.

Recently, there has been an increased availability of 64-bit word size processors. In OEF for these processors, inversion is slower than in the 32-bit environment; however, *pseudo-inversion* can be computed at a low cost. The basic

P.Q. Nguyen (Ed.): VIETCRYPT 2006, LNCS 4341, pp. 147–156, 2006.

concept of pseudo-inversion over $\mathbb{F}_{q^m}$ was first proposed in [4] to compute efficiently a pairing on elliptic curves. In order to apply the pseudo-inversion to ECADD and ECDBL, we propose new coordinates suitable for OEFs. These new coordinates, *compressed Jacobian coordinates*, represent an intermediate concept of the affine and Jacobian coordinates and make the elliptic curve addition faster than that for either type of coordinates.

The compressed Jacobian coordinates are a natural extension of conventional coordinates because the coordinates over $\mathbb{F}_q$ become equivalent to Jacobian coordinates and the coordinates over $\mathbb{F}_{2^m}$, where q is prime, become equivalent to affine coordinates.

We also show the efficiency of the proposed algorithm based on theoretical estimation and actual implementation using the assembly language on a PC. In our implementation, the speed of random scalar multiplications over an elliptic curve, which is almost equal to the speed of EC-DSA signature generation, is 41.9 μsec on a 2.82-GHz Athlon 64 FX PC.

This paper is organized as follows. Section 2 defines the concepts used in the paper. Section 3 describes the proposed algorithm. Section 4 presents an efficiency comparison of the proposed algorithm and the conventional one. Section 5 concludes this paper.

2 Preliminaries

2.1 OEF

Let an OEF [1] be finite field $\mathbb{F}_{q^m}$ that satisfies the following:

- q is a prime less than but close to the maximum integer word of the processor,
- $q = 2^n \pm c$ where $\log_2 c \leq n/2$ and
- an irreducible binomial $f(x) = x^m - \omega$ exists.

We consider the following polynomial based representation of element $A \in \mathbb{F}_{q^m}$,

$$A = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0$$

where $a_i \in \mathbb{F}_q$ and $\alpha \in \mathbb{F}_{q^m}$ is a primitive root of $f(x)$. Since we choose q to be less than the maximum integer word of the processor, we can represent A using m registers.

2.2 Computational Cost

Here, we define the following notation.

`M` is the computational cost of multiplication in $\mathbb{F}_{q^m}$
`S` is the computational cost of squaring in $\mathbb{F}_{q^m}$
`I` is the computational cost of inversion in $\mathbb{F}_{q^m}$
`P` is the computational cost of pseudo-inversion in $\mathbb{F}_{q^m}$
`A` is the computational cost of addition/subtraction in $\mathbb{F}_{q^m}$
`v` is the computational cost of multiplication $\mathbb{F}_q \times \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$

`m` is the computational cost of multiplication in $\mathbb{F}_q$
`s` is the computational cost of squaring in $\mathbb{F}_q$
`i` is the computational cost of inversion in $\mathbb{F}_q$
`a` is the computational cost of addition in $\mathbb{F}_q$
`f` is the computational cost of the Frobenius map in $\mathbb{F}_{q^m}/\mathbb{F}_q$

2.3 Representation of Extension Field Element

This section discusses the representations of extension field elements and the cost of arithmetics using the representations.

Consider an algorithm that consists of a series of arithmetic operations, which include inversion(s), in $\mathbb{F}_{q^m}$. For most implementations, inversion is a costly operation. So, rational representation $\frac{N}{D}$ $(N, D \in \mathbb{F}_{q^m}^{\times})$ can be used. Using the representation, division is free,

$$\left(\frac{N}{D}\right)^{-1} = \frac{D}{N}$$

On the other hand, multiplication and addition require additional multiplications,

$$\frac{N_1}{D_1}\frac{N_2}{D_2} = \frac{N_1 N_2}{D_1 D_2}$$
$$\frac{N_1}{D_1} + \frac{N_2}{D_2} = \frac{N_1 D_2 + N_2 D_1}{D_1 D_2}$$

That is, rational representation does not require an inversion in the algorithm, except for the final inversion(s). If the inversion is a very costly operation, the rational representation is an effective way to reduce the complexity of the algorithm. In some $\mathbb{F}_{q^m}$, when adopting a rational representation in the context of elliptic curve arithmetics, we observed that "complete inversion" is not always necessary. We can use pseudo-inversion, which is easy to compute compared to complete inversion, in some finite fields, $\mathbb{F}_{q^m}$. The definition of the pseudo-inversion algorithm is given hereafter.

Definition 1 (Pseudo-Inversion Algorithm)

Input: $X \in \mathbb{F}_{q^m}$
Output: $(\iota(X), \mathbf{N}(X)) \in \mathbb{F}_{q^m} \times \mathbb{F}_q^{\times}$ *such that* $\iota(X)X = \mathbf{N}(X)$

Moreover, we call $\iota(X)$ a pseudo-inversion of X and $\mathbf{N}(X)$ a co-pseudo-inversion of X. When the pseudo-inversion algorithm is light, the *compressed* rational representation, $\frac{N}{d}$ $(N \in \mathbb{F}_{q^m}, d \in \mathbb{F}_q^{\times})$, is superior to the common and the rational representation. The arithmetics for the compressed rational representation are

$$\left(\frac{N}{d}\right)^{-1} = \frac{\iota(N)d}{\mathbf{N}(N)}$$

$$\frac{N_1}{d_1}\frac{N_2}{d_2} = \frac{N_1 N_2}{d_1 d_2}$$
$$\frac{N_1}{d_1} + \frac{N_2}{d_2} = \frac{N_1 d_2 + N_2 d_1}{d_1 d_2}$$

When $m > 1$, the cost of multiplication between $\mathbb{F}_{q^m}$ and $\mathbb{F}_q$ is less than the cost of multiplication in $\mathbb{F}_{q^m}$. Utilizing the pseudo-inversion algorithm can reduce the cost of an algorithm that consists of a series of arithmetics. This paper applies this idea to elliptic curve scalar multiplication.

Two examples of pseudo-inversion algorithm implementations are given in Sections 2.4 and 2.5.

2.4 Pseudo-inversion by Norm

To compute the pseudo-inversion, we can use exponentiation by q^i.

Given $X \in \mathbb{F}_{q^m}$, we call

$$z = \prod_{i=0}^{m-1} X^{q^i}$$

the norm. The norm, z, is known to be $z \in \mathbb{F}_q$. By this definition,

$$z = X \left(\prod_{i=1}^{m-1} X^{q^i} \right)$$

and

$$Y = \prod_{i=1}^{m-1} X^{q^i}.$$

Output Y can be considered as the pseudo-inversion and z is the co-pseudo-inversion. We can reduce the computational cost of the above algorithm by using the Ito-Tsujii algorithm [5].

For example when $m = 5$,

$$Y \leftarrow X \cdot X^q$$
$$Y \leftarrow Y \cdot Y^{q^2}$$
$$Y \leftarrow Y^q$$
$$z \leftarrow X \cdot Y$$

The computational cost of the q^i-th power in $\mathbb{F}_{q^m}$ can be estimated as

$$\mathtt{f} \approx \mathtt{v} \cdot (m-1)/m = 0.8\mathtt{v},$$

if we use the polynomial basis using a binomial as an irreducible polynomial.

In addition, the computational cost of the last multiplication, $z \leftarrow X \cdot Y$, can be estimated as $\mathtt{v}$ because $z \in \mathbb{F}_q$. Therefore, the total computational cost is estimated as

$$\mathtt{P} = 2\mathtt{M} + 3\mathtt{f} + 1\mathtt{v} \approx 2\mathtt{M} + 3.4\mathtt{v}.$$

2.5 Pseudo-inversion by Euclidean Method

Let `deg(X)` be the degree of polynomial `X`, `lead(X)` be the coefficient of polynomial `X`'s leading term, and `swap(X,Y)` be the procedure that exchanges the values of `X` and `Y`. The following algorithm computes a pseudo-inversion.

```
A = f(x) ; // irreducible polynomial.
B = g(x) ; // input polynomial (where deg(A) > deg(B)) ≠ 0

C =  0  ;
D =  1  ;

while(deg(B) > 0){
  b = lead(B) ;            // coefficient of B's leading term
  while(deg(A) >= deg(B)){
    a = lead(A) ;          // coefficient of A's leading term
    n = deg(A) - deg(B) ;  // difference between degrees A and B

    A *= b ; C *= b ;
    A -= a*B*x^n ;
    C -= a*D*x^n ;
  } ;
  swap(A,B) ;
  swap(C,D) ;
} ;

// output D as pseudo-inversion
// output B as co-pseudo-inversion
```

Roughly speeking, the computational cost of this algorithm is $4m\mathtt{v}$.

3 Compressed Jacobian Coordinates

In the Jacobian coordinate system, a point on an elliptic curve over an extension field is represented as an element of $\mathbb{F}_{q^m}^3$. We can decrease the redundancy of the Jacobian coordinates by using pseudo-inversion. In our new coordinates, a point is represented as an element of $\mathbb{F}_{q^m}^2 \times \mathbb{F}_q$. We call this the compressed Jacobian coordinate system. The compressed Jacobian coordinates can be used for both Koblitz curves[1] and random curves over OEF. Addition formulas in the compressed Jacobian coordinates are given hereafter.

Compressed Jacobian coordinates
$(X, Y, z) \in \mathbb{F}_{q^m}^2 \times \mathbb{F}_q$, where $Y^2 = X^3 + az^4X + bz^6$

[1] Usually, Koblitz curve means elliptic curves over $\mathbb{F}_{2^m}$. In this paper, we use the word "Koblitz curve" as a general elliptic curve over $\mathbb{F}_{p^m}$ that the cofactor is in $\mathbb{F}_p$.

Addition formulas in compressed Jacobian coordinates
Input: $(X_1, Y_1, z_1), (X_2, Y_2, z_2)$, Output: (X_3, Y_3, z_3).

$$\begin{aligned}
z_3' &\leftarrow z_1 z_2 \\
(X_1', Y_1') &\leftarrow (X_1 z_2^2, Y_1 z_2^3) \\
(X_2', Y_2') &\leftarrow (X_2 z_1^2, Y_2 z_1^3) \\
\Lambda_n &\leftarrow \iota(X_2' - X_1') \cdot (Y_2' - Y_1') \\
\lambda_d &\leftarrow \mathbf{N}(X_2' - X_1') \\
z_3 &\leftarrow \lambda_d z_3' \\
(X_1'', Y_1'') &\leftarrow (X_1' \lambda_d^2, Y_1' \lambda_d^3) \\
(X_2'', Y_2'') &\leftarrow (X_2' \lambda_d^2, Y_2' \lambda_d^3) \\
X_3 &\leftarrow \Lambda_n^2 - X_1'' - X_2'' \\
Y_3 &\leftarrow \Lambda_n (X_1'' - X_3) - Y_1''
\end{aligned}$$

We don't compute Y_2'' in real implementation because Y_2'' is not used to compute X_3, Y_3 and z_3. We estimate the computational cost of ECADD without computing Y_2'' hereafter.

Doubling formulas in compressed Jacobian coordinates
Input: (X_1, Y_1, z_1), Output: (X_3, Y_3, z_3).

$$\begin{aligned}
\Lambda_n &\leftarrow \iota(2Y_1) \cdot (3X_1^2 + a z_1^4) \\
\lambda_d &\leftarrow \mathbf{N}(2Y_1) \\
z_3 &\leftarrow \lambda_d z_1 \\
(X_1'', Y_1'') &\leftarrow (X_1 \lambda_d^2, Y_1 \lambda_d^3) \\
X_3 &\leftarrow \Lambda_n^2 - 2X_1'' \\
Y_3 &\leftarrow \Lambda_n (X_1'' - X_3) - Y_1''
\end{aligned}$$

4 Evaluation and Implementation

In this section, we evaluate the performance of addition and doubling in the compressed Jacobian coordinates. For simplicity, we neglect the cost of addition and subtraction in $\mathbb{F}_{q^m}$, and multiplication in $\mathbb{F}_q$. In addition, if $m = 5$, we can estimate that

$$\begin{aligned}
\mathtt{v} &\approx 0.2\mathtt{M}. \\
\mathtt{P} &\approx 2\mathtt{M} + 3.4\mathtt{v} \approx 2.68\mathtt{M}.
\end{aligned}$$

See Section 2.4 for details.

If $\mathtt{I} > 3.68\mathtt{M}$ and $\mathtt{S} = 0.8\mathtt{M}$, the compressed Jacobian coordinate system is the fastest when $m = 5$.

Implementation Result
We chose the parameters in Table 3 and the environment as given in Table 4. We chose the cyclic window method [3] as the scalar multiplication algorithm and the results are given in Table 7. In Table 8, we refer to the previously

Table 1. Computational Cost of ECADD

Coordinate system	Cost	$m = 5$
Compressed Jacobian	$2\mathtt{M} + 1\mathtt{S} + 1\mathtt{P} + 7\mathtt{v}$	$\approx 6.08\mathtt{M} + 1\mathtt{S}$
Compressed Jacobian ($z_2 = 1$)	$2\mathtt{M} + 1\mathtt{S} + 1\mathtt{P} + 5\mathtt{v}$	$\approx 5.68\mathtt{M} + 1\mathtt{S}$
Affine	$2\mathtt{M} + 1\mathtt{S} + 1\mathtt{I}$	
Jacobian	$12\mathtt{M} + 4\mathtt{S}$	
Jacobian ($Z_2 = 1$)	$8\mathtt{M} + 3\mathtt{S}$	
Modified Jacobian	$13\mathtt{M} + 6\mathtt{S}$	
Modified Jacobian ($Z_2 = 1$)	$9\mathtt{M} + 5\mathtt{S}$	
Modified Jacobian ($Z_2 = 1$) ($a = -3$)	$8\mathtt{M} + 5\mathtt{S}$	

Table 2. Computational Cost of ECDBL

Coordinate system	Cost	$m = 5$
Compressed Jacobian	$2\mathtt{M} + 2\mathtt{S} + 1\mathtt{P} + 3\mathtt{v}$	$\approx 5.28\mathtt{M} + 2\mathtt{S}$
Compressed Jacobian ($a \in \mathbb{F}_q$)	$2\mathtt{M} + 2\mathtt{S} + 1\mathtt{P} + 2\mathtt{v}$	$\approx 5.08\mathtt{M} + 2\mathtt{S}$
Affine	$2\mathtt{M} + 2\mathtt{S} + 1\mathtt{I}$	
Jacobian	$4\mathtt{M} + 6\mathtt{S}$	
Jacobian ($a = -3$)	$4\mathtt{M} + 4\mathtt{S}$	
Modified Jacobian	$4\mathtt{M} + 4\mathtt{S}$	

Table 3. Parameters

q	$2^{61} - 1$ (prime)
$\mathbb{F}_{q^m}$	$\mathbb{F}_q[\alpha]/(\alpha^5 - 3)$ (polynomial basis)
Elliptic curve	$Y^2 = X^3 - 3X + 2023176626027320614$
Trace	$t = 2713676959$
Generator	$X = 1225397330577448427\ \alpha^4 + 110313758532384199\ \alpha^3 + 1639881413522258503\ \alpha^2 + 547643109538786165\ \alpha + 2214931762811684809$ $Y = 74533231004088031\ \alpha^4 + 1705584686783011420\ \alpha^3 + 2159424991416008329\ \alpha^2 + 509248187364731537\ \alpha + 570065311020511817$
Order	28269553069723731963330948928353289444455373120300688657015697428589796171 (244-bit prime)

Table 4. Environment

CPU	AMD Athlon 64 FX-57
Clock Frequency	2.82 GHz
OS	FreeBSD 5.4R
Compiler	gcc version 4.0.3 20051229 (prerelease)

reported fastest implementation results. We cannot find any previous work that uses similar parameter we used. We decided to show the references:

1. The security parameter is similar to ours. (122 bit)
2. The field construction is similar to ours. (OEF)

Table 5. Implementation Results of ECADD

Coordinate system	Cycles	μsec
Compressed Jacobian	939	0.33
Compressed Jacobian ($z_2 = 1$)	889	0.32
Affine	2216	0.79
Jacobian	1996	0.71
Jacobian ($Z_2 = 1$)	1481	0.53
Modified Jacobian ($a = -3$)	2213	0.79
Modified Jacobian ($a = -3, Z_2 = 1$)	1702	0.60

Table 6. Implementation Results of ECDBL

Coordinate system ($a = -3$)	Cycles	μsec
Compressed Jacobian	942	0.33
Affine	2337	0.83
Jacobian	1086	0.39
Modified Jacobian	1031	0.37

Table 7. Implementation Results of EC Scalar Multiplication

Coordinate system ($a = -3$)	kcycles	μsec
Compressed Jacobian	118	41.9
Affine	276	97.9
Jacobian	165	58.6
Modified Jacobian	174	61.9

Table 8. Comparison to Known Results

Curve		Security	kcycles	μsec	CPU	Clock
$\mathbb{F}_{q^m}$ Koblitz (This paper)	$q = 2^{61} - 1, m = 5$	122 bits	118	41.9	Athlon 64 FX	2.82 GHz
$\mathbb{F}_q$ Montgomery form [6]	$q = 2^{255} - 19$	126 bits	625	–	Athlon	–
$\mathbb{F}_{2^m}$ Koblitz [7]	$m = 233$	116 bits	897	2243	Pentium II	400 MHz
$\mathbb{F}_{q^m}$ Koblitz [3]	$q = 2^{29} - 3, m = 7$	87 bits	127	254	21264	500 MHz
$\mathbb{F}_{q^m}$ Koblitz, Hessian form [8]	$q = 2^{29} - 3, m = 7$	87 bits	213	66.6	Pentium 4	3.2 GHz

Our current results are faster than these both in terms of speed in CPU cycles and μsec. These numbers were obtained using various parameters and/or platforms. Please take note of the parameters when comparing these numbers.

5 Conclusion

In this paper, we proposed a new coordinate system, which we call the compressed Jacobian coordinates, to accelerate elliptic curve cryptosystems over extension fields. The main idea of the proposed coordinates is to utilize pseudo-inversion, and the coordinates are a natural extension of traditional affine and Jacobian coordinates. We estimated the computational cost of ECADD and ECDBL for the coordinates. We also implemented scalar multiplication as soft-

ware on the x86-64 platform and confirmed that 244-bit OEF scalar multiplication can be computed in 41.9 μsec on an Athlon 64 FX-57 (2.82 GHz) using the compressed Jacobian coordinates. To the best knowledge of the authors, this is the fastest software implementation of elliptic curve scalar multiplications for the security level of 122 bits or higher. This means that we can exchange more than 10,000 ECDH keys per second on an inexpensive personal computer (if the communication cost is negligible).

The idea of the compressed Jacobian coordinates can be easily applied to the other coordinates, e.g., projective coordinates and Montgomery coordinates, and the other algebraic curves, e.g., the hyper-elliptic curves and C_{ab} curves.

References

1. Bailey, D.V., Paar, C.: Optimal extension fields for fast arithmetic in public-key algorithms. In Krawczyk, H., ed.: Advances in Cryptology — CRYPTO'98. Volume 1462 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1998) 472–485
2. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In Ohta, K., Pei, D., eds.: Advances in Cryptology — ASIACRYPT'98. Volume 1514 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1998) 51–65
3. K.Aoki, F.Hoshino, T.: A cyclic window algorithm for ECC defined over extension fields. In: ICICS 2001. Volume 2229 of Lecture Notes in Computer Science., Springer-Verlag (2001) 62–73
4. Kobayashi, T., Aoki, K., Imai, H.: Efficient algorithms for Tate pairing. IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan) **E89-A** (2006) 134–143
5. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. In: Information and Computation. Volume 78. (1988) 171–177
6. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. Lecture Notes in Computer Science **3958** (2006) 207–228
7. Hankerson, D., Hernandez, J.L., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. Lecture Notes in Computer Science **1965** (2001) 1–24
8. Kumagai, M.: Efficient implementation of Hessian-form elliptic curve cryptosystem with SIMD instructions. In: The 2005 Symposium on Cryptography and Information Security (SCIS2005), Maiko, Kobe, Japan, The Institute of Electronics, Information and Communication Engineers (2005) 1651–1656 (in Japanese).

A Appendix: Frobenius Map

In this section, we recall the Frobenius map. Let $E/\mathbb{F}_q$ denote a non-supersingular elliptic curve defined over finite field $\mathbb{F}_q$ where q is a prime or any power of a prime. $P = (X, Y)$ is an $\mathbb{F}_{q^m}$-rational point of elliptic curve E defined over $\mathbb{F}_q$. The Frobenius map, ϕ, is defined as

$$\phi : (X, Y) \to (X^q, Y^q).$$

The Frobenius map is an endomorphism over $E(\mathbb{F}_{q^m})$. It satisfies the equation

$$\phi^2 - t\phi + q = 0, \qquad -2\sqrt{q} \leq t \leq 2\sqrt{q} \tag{1}$$

It takes a negligible amount of time to compute the Frobenius map in an endomorphism ring where t is the trace of $E/\mathbb{F}_q$ provided that element in $\mathbb{F}_{q^m}$ is represented using the polynomial basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ using a binomial as a definition polynomial.

B Appendix: Coordinates

Let

$$E : Y^2 = X^3 + aX + b \qquad (a, b \in \mathbb{F}_{q^m},\ 4a^3 + 27b^2 \neq 0)$$

be the equation of elliptic curve E over $\mathbb{F}_{q^m}$.

For Jacobian coordinates, with $X = X_J/Z_J^2$ and $Y = Y_J/Z_J^3$, a point on elliptic curve P is represented as $P = (X_J, Y_J, Z_J) = (X, Y)$. In order to accelerate ECADD, the Chudnovsky Jacobian coordinates [2] represent a Jacobian point as the quintuple $(X_J, Y_J, Z_J, Z_J^2, Z_J^3)$. On the other hand, in order to accelerate ECDBL, the modified Jacobian coordinates [2] represent a Jacobian point as the quadruple (X_J, Y_J, Z_J, aZ_J^4).

The number of operations required to compute ECDBL and ECADD is shown in Table 9.

Table 9. Number of Operations for Each Coordinate Systems

Coordinates	ECDBL	ECADD	ECADD with $Z = 1$
Affine	2 M+ 2 S+ I	2 M+ 1 S+ I	—
Chudnovsky Jacobian	5 M+ 6 S	12 M+ 4 S	8 M+ 3 S
Modified Jacobian	4 M+ 4 S	13 M+ 6 S	9 M+ 5 S