

楕円曲線暗号の最高速実装

The Fastest ECC Implementations

青木和麻呂*

Kazumaro AOKI

小林鉄太郎*

Tetsutaro KOBAYASHI

星野文学*

Fumitaka HOSHINO

Abstract— We optimize the Karatsuba-Ofman multiplication for polynomials whose degree is 2, and applied the result to GF(p^3)-type OEF. As a result, we achieved an elliptic curve encryption in $816\mu s$ on 21164A (500MHz) which is about 6% faster than the Lim-Hwang's result presented at PKC2000, and in $354\mu s$ on 21264 (500MHz). Moreover, we present elliptic curve arithmetics using parallel multiplications, which are well suited to Intel's MMX technology, and applied them to GF(p^{13})-type OEF. As a result, elliptic curve addition and doubling speed up about 14% faster than the Lim-Hwang's results.

Keywords: Elliptic curve cryptosystem (ECC), Optimal extension field (OEF), Pentium II, 21164A, 21264

1 はじめに

近年、電子商取引の発展など公開鍵暗号の需要がますます高まって来ている。公開鍵暗号方式として従来 RSA 暗号が事実上の標準であったが、最近は楕円曲線暗号が高速処理可能なことや、いくつかの例外を除いて効率的な攻撃が見つかっていないことから RSA と同程度の安全性を保ったまま鍵長を短くできるので注目されている。

楕円曲線暗号は、そのままでも RSA 暗号に比べてひと桁程度高速である。さらにソフトウェアでひと桁程度高速となる最適拡大体 (OEF; Optimal Extension Field) を利用した実装法が CRYPTO'98 で Bailey と Paar によって提案された [BP98]。OEF は CPU のワードサイズに特化した演算を有限体乗算に活用することにより最大限 CPU の性能を引き出す方法と考えられる。この方法は、既に共通鍵暗号では RC5 や RC6、ハッシュ関数では MD5 や SHA-1 などで取り入れられている。

本稿では、現在世界で最も使われていると思われる Pentium II ファミリを含む CPU で、OEF の実装を最適化した結果を報告する。

* NTT 情報流通プラットフォーム研究所
〒239-0847 神奈川県横須賀市光の丘 1-1

NTT Laboratories.
1-1 Hikarinooka, Yokosuka-shi, Kanagawa-ken, 239-0847
Japan.

2 楕円曲線

楕円曲線暗号は楕円曲線上定義される群の演算を用いる。楕円曲線上の群の元をどう表現するかによって楕円曲線の定義体上での四則演算回数がどの程度必要かが変化する。表現方法は大きく分けて、群演算に逆元を必要としない projective 系の座標か、逆元を必要とする affine 座標に分類される。楕円曲線暗号を実現する上で、定義体の逆元演算が乗算に比べて 3 ~ 5 倍の範囲より時間がかかる場合は projective 系の座標、少ない時間でできる場合は affine 座標を用いる方が高速に楕円曲線暗号を実行できる。それぞれの座標を用いた場合の、群演算やそれぞれの座標の優位性については文献 [CMO98] が詳しい。

ここでは、例として projective 系の座標である Jacobi 座標を用いた場合を説明する。楕円曲線は a, b をパラメータとし

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

で定義され $P = (X_1, Y_1, Z_1)$ 、 $Q = (X_2, Y_2, Z_2)$ 、 $P + Q = R = (X_3, Y_3, Z_3)$ とおいたときに群演算は以下のように計算される。

加算

$$\begin{cases} X_3 \leftarrow -H^3 - 2U_1H^2 + r^2 \\ Y_3 \leftarrow -S_1H^3 + r(U_1H^2 - X_3) \\ Z_3 \leftarrow Z_1Z_2H \end{cases} \quad (1)$$

但し、 $U_1 \leftarrow X_1Z_2^2$ 、 $U_2 \leftarrow X_2Z_1^2$ 、 $S_1 \leftarrow Y_1Z_2^3$ 、 $S_2 \leftarrow Y_2Z_1^3$ 、 $H \leftarrow U_2 - U_1$ 、 $r \leftarrow S_2 - S_1$ とする。

2 倍算

$$\begin{cases} X_3 \leftarrow -2S + M^2 \\ Y_3 \leftarrow -8Y_1^4 + M(S - X_3) \\ Z_3 \leftarrow 2Y_1Z_1 \end{cases}$$

但し、 $S \leftarrow 4X_1Y_1^2$ 、 $M \leftarrow 3X_1^2 + aZ_1^4$ とする。

3 従来技術

この節では OEF 上の楕円曲線暗号高速化関連技術について歴史順に説明する。

OEF そのものは Bailey と Paar により CRYPTO'98 で発表された [BP98]。OEF とは、素数 p を $2^n \pm c$ (ただし c は小さな整数) とし、 $GF(p)$ 上の既約多項式 $f(x)$ が $x^m - \omega$ という 2 項式を用いて、 $GF(p)/(f(x)) \cong GF(p^m)$ と表される有限体である。CPU に応じて n や m を適切に選ぶことにより、 p^m と同程度の位数を持つ素体上の楕円曲線暗号に比べ OEF 上の楕円曲線暗号は数倍高速に実装できる。

標数 2 の有限体上で定義される楕円曲線上の群の k 倍演算の高速化の一手法として、Frobenius 写像を用いる方法が知られている。小林らは EUROCRYPT'99 で、この手法を標数 p である OEF で実現できる方法を提案した [KMKH99]。この方法により楕円曲線暗号は 2 倍程度高速に実装できるようになった。

正規基底を用いた表現で、Fermat の小定理に基づく冪乗演算を用いて有限体上の逆元を高速に求める方法が知られている [IT87]。この方法は、Frobenius 写像が巡回シフトを用いることにより高速に計算できることを利用している。OEF では正規基底を用いず多項式基底を用いた場合でも十分高速に Frobenius 写像を計算できることが既に小林らの方法で用いられている。Bailey らは Mathematics of Public Key Cryptography で、伊東らの方法を OEF 上に応用し、高速に逆元を計算する方法を提案した [BP99]。逆元が高速に計算できるようになったことから、従来 OEF 上の楕円曲線の群演算は projective 系の座標で計算されてきたが、affine 座標を用いる方が優位になり、全体として楕円曲線暗号が数割高速化された。

楕円曲線暗号を実現する上で高速化のためには様々な方法が知られている。その中で従来 OEF 上の楕円曲線暗号を実現する上で利用が報告されていなかった方法として、[K97, Section 4.3.3] で紹介されている Karatsuba-Ofman 乗算 (以後「KO 法」と呼ぶ)、[K97, Section 4.6.1] で紹介されている多項式除算、projective 系の座標で k 倍演算をする場合に楕円加算は片方の元が $Z = 1$ と仮定できることの利用、さらに、OEF 乗算を計算する上で計算途中の和を加算する度に $\text{mod } p$ するのではなく 2 倍長加算を用いた方が速い場合があることを利用した方法が、PKC2000 で Lim らにより報告された [LH00]。この方法により、従来法より約 2 倍の高速化が達成された。

4 議論

我々の目的はソフトウェアで最高速の楕円曲線暗号を実装することにある。基本的には Lim らの方法 [LH00] に従って実装する。

この節では実装する楕円曲線暗号の様々なパラメータ及び利用アルゴリズムを順に決めていく。

4.1 安全性の下限

一般によくいわれている [LV00] ように、楕円曲線上の部分群の素数位数の最大値を最低 160 ビットとなるようにした。

4.2 CPU

実装環境として世界で最も使われていると思われる Pentium ファミリーのうち Pentium II (400MHz) を選択した。また、高速化追求の目的からワード長 64 ビットの CPU として 21264 (500MHz) も選択した。また、他の実装と比較のため 21164A (500MHz) での実装も参考のため評価した。

表 1 にそれぞれの CPU の乗算のおおよその特性を示す。latency とは計算を開始してから結果が利用可能になるまでの時間、throughput とは計算を開始してから乗算を含む次の命令を実行できるまでに要する時間である。乗算以外では、加減算を含むほとんどの命令で全て latency 1、throughput 1 となっている。また、ここにあげた CPU はどれも乗算器は一つしか備えていない。

今回選択した CPU も含めて、近年の CPU では整数乗算が桁違いに速くなっていることに注意されたい。例えば、Intel の 80386 では加減算では 2、乗算では 9 ~ 38 の latency かつ throughput であった。従って、今回の実装のように漸近的な議論ではなく 160 ビット固定の議論をする場合は乗算だけではなく加減算のコストも比較検討することは重要なことである。

表 1 中の数字は条件によって大きく変わるので、詳しくは各社 (Intel と COMPAQ) の WWWなどを参照して欲しい。

4.3 定義体パラメータ

それぞれのプロセッサについて最適なパラメータを探索した。ここで、OEF $GF(p^m)$ の活用は用いない場合に比べて明らかな優位性があるので、利用を大前提とした。

また、Frobenius 写像を用いた k 倍演算は、用いない場合に比べてかなり優位ではある。しかし楕円曲線上の群の部分群の最大素数位数は、 m が素数だとしても p^{m-1} 程度にしかならず、 m が合成数の場合は、体の位数 p^m に比べて極端に小さくなり Frobenius 写像を用いた場合に得られる優位性が発揮されないことに注意しなければならない。

表 1: 乗算速度 (cycles/ 乗算)

	Pentium II (2 ALUs)		21164A (2 ALUs)		21264 (4 ALUs)	
	latency	throughput	latency	throughput	latency	throughput
(32,32)	4	1	8 ~ 10	4 ~ 6	7	1 ~ 2
(32,64)	5	2				
(64,64)			12 ~ 14	8 ~ 10	7	1 ~ 2
(64,128) の上位 64 ビット			14 ~ 16	8 ~ 10	7	1 ~ 2
(16,16) × 4*	3	1				
(16,32) の上位 16 ビット × 4*	3	1				
[(16, 32) + (16, 32)] × 2*	3	1				

- 表中 (a, b) は a ビット同士の乗算で結果が b ビットを表す
- 「*」は MMX 命令を表す

4.3.1 ワード長 64 ビット

表 1 より、21164A 及び 21264 では 64 ビットの乗算を効率的に行なうことができる。そこで、 p として 64 ビット程度の数を選択すべきである。

拡大次数 安全性上の理由から拡大次数は $\lceil \frac{160}{64} \rceil = 3$ 以上必要である。Frobenius 写像を用いることを考慮すると、 m として 3 より大きな素数 5 を選ぶ必要がある。

Frobenius 写像を用いる場合 $m = 5$ と、用いない場合 $m = 3$ について、 $GF(p^m)$ 乗算がどの程度異なるか、 $GF(p)$ での乗算回数を用いて大雑把に比較する。また、乗算は KO 法を使うものとする。

$m = 3$ であれば $GF(p)$ 乗算 6 回、一方 $m = 5$ の場合は(詳しくは検討していないが) 最低でも $GF(p)$ 乗算 $\lceil 5 \log_2 3 \rceil = 13$ 回は必要となることが予想される。従って、 $GF(p^3)$ の乗算は $GF(p^5)$ の乗算に比べ約 2.2 倍高速と考えられる。一方、楕円 k 倍算には、大雑把には signed window 法を使った場合約 183 回の群演算 ([LH00] の評価)、Frobenius 写像を用いる場合は約 86 回の群演算 ([KMKH99] の評価) が必要なので、 $GF(p^3)$ での k 倍算は $GF(p^5)$ での k 倍算に比べ約 2.1 倍低速と考えられる。

それぞれの優位性はほとんど変わらないので、利用レジスタ数が少なくて済む $m = 3$ を採用することとした。

標数と 2 項式 安全性上の制約から $\log_2 p \geq \lceil \frac{160}{3} \rceil = 54$ とする。この範囲で、 $p = 2^n - 1$ となる n は唯一存在し $n = 61$ である。 $x^3 - \omega$ が既約となる最小の ω は 5 である。他の素数については、 $p = 2^n - 3$ は存在せず $p = 2^n - 5$ の形では $n = 56$ のみ存在したが、 $GF(2^{56} - 5)$ 上 $X^3 - \omega$ は全て可約である。

従って、 $p = 2^{61} - 1$ 、 $f(x) = x^3 - 5$ が最適なパラメータと考えられるので、我々はこのパラメータを採用した。

4.3.2 ワード長 32 ビット

表 1 より、Pentium II は 32 ビットの乗算を効率的に行なうことができる。そこで、 p として 32 ビット程度の

素数を採用する。

拡大次数 安全性上の理由から、拡大次数 m は $\frac{160}{32} = 5$ 以上必要である。実際には、 2^{32} は素数ではないので、6 次以上が必要となる。Frobenius 写像の利用を考えると $m \geq 7$ となる素数 m が必要となるが幸いにして $m = 7$ は素数であるので、これを採用する。

標数と 2 項式 安全性上の理由から $n \geq \lceil \frac{160}{7-1} \rceil = 27$ が必要となる。また、 $GF(p^m)$ の乗算で $\text{mod } f(x)$ を計算する前に最大 $m(p-1)^2$ を扱うことになるが、この値が 2 倍長(64 ビット)を越えると極端に効率が低下する。この理由から $\left\lfloor \log_2 \left(\sqrt{\frac{2^{2 \times 32}}{7}} + 1 \right) \right\rfloor = 30 \geq n$ という条件も満たすことが望ましい。

この条件を満たす $p = 2^n - 1$ は存在せず、 $p = 2^n - 3$ は $n = 29$ のときのみ存在した。この p に対して、 $x^7 - \omega$ が既約となる最小の ω は 2 であった。

従って、 $p = 2^{29} - 3$ 、 $f(x) = x^3 - 2$ が最適なパラメータと考えられるので、我々はこのパラメータを採用した。

4.3.3 ワード長 16 ビット

表 1 より、Pentium II は MMX 命令に含まれる 16 ビットの乗算を効率的に行なうことができる。そこで、 p として 16 ビット程度の素数を採用した場合について考察する。

拡大次数 安全性上の理由から $m \geq \frac{160}{16} = 10$ を満たす必要がある。実際には $p = 2^{16}$ は素数ではないので、 $m \geq 11$ でなければならないが、Frobenius 写像の利用を考えると $m \geq 12$ である必要がある。また、 m は素数でもなければならぬので、 $m = 13$ を採用する。

標数と 2 項式 安全性上の理由から $n \geq \lceil \frac{160}{13-1} \rceil = 14$ でなければならない。 $p = 2^n - 1$ となる n は存在せず、 $p = 2^n - 3$ となる n は 14 のみ存在する。このとき、 $f(x) = x^{14} - \omega$ が既約となる最小の ω は 2 である。

従って、 $p = 2^{14} - 3$ 、 $f(x) = x^{14} - 2$ が最適なパラメータと考えられるので我々はこれを採用した。

4.3.4 パラメータのまとめ

表 2 に定義体のパラメータをまとめた。

表 2: 定義体のパラメータ

ワード長	標数 (p)	既約 2 項式 ($f(x)$)	部分群位数
64	$2^{61} - 1$	$x^3 - 5$	183 ビット
32	$2^{29} - 3$	$x^7 - 2$	174 ビット
16	$2^{14} - 3$	$x^{13} - 2$	168 ビット

どのパラメータも $\text{mod } f(x)$ するまえに 2 倍長演算で取まる範囲である

$$m(p-1)^2 \leq 2^{(\text{ワード長})}$$

を満たす。さらにワード長が 16 ビット以外の場合は、 $\text{mod } f(x)$ の前に各係数の $\text{mod } p$ を行なわずに済む

$$(1 + \omega(m-1))(p-1)^2 \leq 2^{(\text{ワード長})}$$

も満たす。

4.4 KO 法に対する考察

KO 法の基本は

$$\begin{aligned} & (a_0 + a_1x) \times (b_0 + b_1x) \\ &= a_0b_0 + (a_0b_1 + a_1b_0)x + a_1b_1x^2 \\ &= a_0b_0 + \{(a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1\}x \\ &\quad + a_1b_1x^2 \end{aligned}$$

という変換を行ない、係数の乗算の結果を再利用することにより乗算回数を減らすことにある。

4.4.1 1 次式乗算

多項式の次数が小さい場合は、

- 係数の乗算が高速に計算可能
- 加算回数が多い

という理由によりかえって教科書方式より遅くなる可能性がある。しかし、次数が増加したとき漸近的には乗算回数だけでなく加算回数も少なくなる。

実際 Pentium II で、 $\text{GF}((2^{29} - 3)^7)$ の乗算の中で使われそうな形で実装実験してみたところ 1 次式の乗算では、教科書方式では 20 cycles で、KO 法では 18 cycles となり有意な差が確認されなかった。それぞれの計算量は表 3 に示す通りなので、計算量について

Time(単精度乗算)

$$\approx 2\text{Time}(单精度加算) + \text{Time}(倍精度加算) \quad (2)$$

となることがわかる。

KO 法は、乗算が遅ければ遅いほど効果を発揮することから $\text{GF}((2^{61} - 1)^3)$ では KO 法を利用するのがよい。

表 3: 1 次式乗算の計算量

教科書方式	KO 法
単精度乗算	4 回
単精度加算	0 回
倍精度加減算	1 回

4.4.2 2 次式乗算

2 次式乗算を KO 法で行なう場合、文献 [LH00] に次の方法が記載されている。

$$\begin{aligned} & (a_0 + a_1x + a_2x^2)(b_0 + b_1x + b_2x^2) \\ &= a_0b_0 + \{(a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1\}x \\ &\quad + \{(a_0 + a_2)(b_0 + b_2) + a_1b_1 - a_2b_2 - a_0b_0\}x^2 \\ &\quad + \{(a_1 + a_2)(b_1 + b_2) - a_2b_2 - a_1b_1\}x^3 + a_2b_2x^4 \end{aligned}$$

これに対し我々は以下の方法を発見した。

$$\begin{aligned} & (a_0 + a_1x + a_2x^2)(b_0 + b_1x + b_2x^2) \\ &= a_0b_0 + \{[(a_0 + a_1)(b_0 + b_1) - a_1b_1] - a_0b_0\}x \\ &\quad + [(a_0 + a_1 + a_2)(b_0 + b_1 + b_2) \\ &\quad - \{(a_0 + a_1)(b_0 + b_1) - a_1b_1\} \\ &\quad - \{(a_1 + a_2)(b_1 + b_2) - a_1b_1\}]x^2 \\ &\quad + [\{(a_1 + a_2)(b_1 + b_2) - a_1b_1\} - a_2b_2]x^3 \\ &\quad + a_2b_2x^4 \end{aligned}$$

それぞれの方法の計算量は表 4 に示す通りである。Pentium II 上では、式 (2) より、教科書方式と Lim らの方法ではほとんど差がないことがわかるが、提案方式では若干 KO 法が有利になることがわかった。

表 4: 2 次式乗算の計算量

教科書方式	Lim らの方法	提案方式
単精度乗算	9 回	6 回
単精度加算	0 回	6 回
倍精度加減算	4 回	7 回

よって、 $\text{GF}((2^{29} - 3)^7)$ では 2 次以上の乗算を行なう場合 KO 法を用いて計算する。

4.4.3 3 次式乗算

定義体 $\text{GF}((2^{14} - 3)^{13})$ で利用を想定している Pentium II の MMX 命令では乗算が Pentium II の通常の整数乗算よりさらに速い、かつ加算まで 1 命令で同時に行なえることから明らかに 2 次式乗算までは教科書方式が有利である。3 次式については、微妙ではあるが項数が偶数であるので、3 回の 1 次式乗算を教科書方式で行なった結果を KO 法でまとめた。

4.5 楕円曲線パラメータ

Frobenius写像を用いるワード長16ビット及び32ビットでは、Weil予想を用いて構成した[KMKH99]。なお、 α はOEFの定義多項式 $f(x)$ の根の一つとする。

4.5.1 ワード長16ビット

$$y^2 = x^3 - 3x - 172$$

部分群の素数位数の最大値 3 7735447064 0784663733
8580162818 7749646114 9221530761 (168ビット超)

生成元 $(10869\alpha^{12} + 3898\alpha^{11} + 15358\alpha^{10} + 3782\alpha^9 + 4242\alpha^8 + 7589\alpha^7 + 5310\alpha^6 + 12599\alpha^5 + 10370\alpha^4 + 9316\alpha^3 + 8340\alpha^2 + 184\alpha + 9573, 8924\alpha^{12} + 9141\alpha^{11} + 9472\alpha^{10} + 8964\alpha^9 + 14633\alpha^8 + 4204\alpha^7 + 5379\alpha^6 + 13644\alpha^5 + 11470\alpha^4 + 15042\alpha^3 + 6518\alpha^2 + 15906\alpha + 7391)$

4.5.2 ワード長32ビット

$$y^2 = x^3 - 3x - 85$$

部分群の素数位数の最大値 239 4696831448 0862150279
8948628438 5174133848 4034750169 (174ビット超)

生成元 $(200472906\alpha^6 + 172723217\alpha^5 + 174386879\alpha^4 + 403718784\alpha^3 + 23043362\alpha^2 + 525400877\alpha + 17252111, 523133120\alpha^6 + 178522781\alpha^5 + 357710308\alpha^4 + 10611891\alpha^3 + 423928020\alpha^2 + 2135201\alpha + 535095305)$

4.6 並列楕円群演算

MMX命令はSIMD(Single Instruction Multiple Data)型の命令である。従って、表1に示すように同時に2回または、4回の乗算が実行できたとしても、次のような問題が発生する。

- データの依存性のある乗算は同時に実行できない。
- ひと組みのデータセットの中同士の演算を実行するためには、データの位置変換のためのコストが馬鹿にならない。

上記問題を解決するためには、同時に独立な2回または4回のOEF乗算を行なえばよい。楕円曲線上の群演算を注意深く観察した結果、我々はprojective系の座標では、独立な乗算を無駄なくできることを発見した。一例として図1に式(1)を並列乗算を用いた計算する方法を示す。図中、加減算は一部省略されていることに注意。

5 結果

4節の議論に従い楕円曲線暗号を実装した結果を表5に示す。群演算の(A)と(P)は、それぞれaffine座標と、projective系の座標を示す。また参考のため、従来法で

Step	演算器1	演算器2
1*	Z_1^2	Z_2^2
2	$U_1 \leftarrow X_1 Z_2^2$	$U_2 \leftarrow X_2 Z_1^2$
3	Z_1^3	Z_2^3
4	$S_1 \leftarrow Y_1 Z_2^3$	$S_2 \leftarrow Y_2 Z_1^3$
5*	H^2	r^2
6	H^3	$U_1 H^2$
7	$S_1 H^3$	$Z_1 Z_2$
8	$r(U_1 H^2 - X_3)$	$Z_3 \leftarrow Z_1 Z_2 H$

「*」は2乗を示す

図1: Jacobi座標での楕円加算(並列乗算版)

もっとも速いと思われるLimらの方法[LH00]をクロック比で補正した結果も併記する。

表中の実装は全てprojective系の座標を用いた結果である。GF($(2^{61} - 1)^3$)については、Limらが指摘したように楕円加算で加算の片方の元が $Z = 1$ となると比較的高速に実装できるので、 k 倍算で利用したsigned window法での事前演算でのみaffine座標を利用した。その場合のOEF逆元はCramérの公式に基づく方法[KMKH99]を用いた。

なおGF($(2^{61} - 1)^3$)上の数値は、時間的な制約から安全な楕円曲線を構成できなかったので、ランダムなパラメータを使った場合の計測値である。この場合でも演算速度はパラメータに依存しないことに注意せよ。

6まとめ

本稿では、まず2次多項式KO法の改善を行なった。この結果をGF(p^3)型のOEFに適用したところ、楕円 k 倍算が従来法[LH00]より約6%高速になった。

また、Pentium IIで利用可能なMMX命令で有効に働く並列楕円群演算を提案した。Pentium IIで実装実験した結果、同様のOEFパラメータでは従来法[LH00]より楕円群演算で約14%の高速化が計られた。

今回報告した実装中、まだまだ最適化の可能性があるにも関わらず未検討の部分がある。今後、さらに最適化を進め、その結果を報告したい。

参考文献

- [BP98] D. V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In H. Krawczyk, editor, *Advances in Cryptology — CRYPTO'98*, Volume 1462 of *Lecture Notes in Computer Science*, pp. 472–485. Springer-Verlag, Berlin, Heidelberg, New York, 1998.

- [BP99] D. V. Bailey and C. Paar. Inversion in Optimal Extension Fields. In A. Odlyzko, G. Walsh, and

表 5: 楕円曲線上の群の k 倍算 (cycles)

CPU	提案法				[LH00]		
	Pentium II	21164A	21264		Pentium II	21164	
p	$2^{14} - 3$	$2^{29} - 3$	$2^{61} - 1$	$2^{61} - 1$	$2^{14} - 3$	$2^{28} - 57$	$2^{57} - 13$
$f(x)$	$x^{13} - 2$	$x^7 - 2$	$x^3 - 5$	$x^3 - 5$	$x^{13} - 2$	$x^7 - 2$	$x^3 - 2$
部分群位数	168	174	183	183	168	168	171
k 倍算 (μ 秒)	1420	1240	816	354	904	778	870
k 倍算 (10^3 cycles)	568	496	408	177	362	311	435
楕円加算 (A)	NA	NA	1921	1290	6091	4628	2596
楕円 2 倍算 (A)	NA	NA	2137	1382	6863	5107	3006
楕円加算 (P)	5280	4595	2902	1133	6171	4256	2697
楕円 2 倍算 (P)	3815	4146	1836	794	4442	3086	2041
OEF 逆元	NA	NA	1140	963	4200	3259	1504
OEF 乗算	827/2	376	202	75	543	379	225
OEF 自乗	NA	310	174	62	404	301	204
OEF 加算	33	39	22	15	91	42	34
OEF 減算	28	38	22	16			
素体逆元	NA	NA	806	861	19	457	848

- NA は未実装であることを示す
- Pentium II のみ周波数 400MHz、他は周波数 500MHz の CPU を利用した

H. Williams, editors, *Conference on The Mathematics of Public Key Cryptography*. The Fields Institute for Research in the Mathematical Sciences, Toronto, Ontario, 1999.

[CMO98] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, Volume 1514 of *Lecture Notes in Computer Science*, pp. 51–65. Springer-Verlag, Berlin, Heidelberg, New York, 1998.

[IT87] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in Finite Fields Using Normal Bases. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. J70-A, No. 11, pp. 1637–1645, 1987. (in-Japanese).

[K97] D. E. Knuth. *Seminumerical Algorithms*, Volume 2 of *The Art of Computer Programming*. Addison Wesley, third edition, 1997.

[KMKH99] T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino. Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT'99*, Volume 1592 of *Lecture Notes in Computer Science*,

pp. 176–189. Springer-Verlag, Berlin, Heidelberg, New York, 1999. (A preliminary version was written in Japanese and presented at SCIS'99-W4-1.4).

[LH00] C. H. Lim and H. S. Hwang. Fast Implementation of Elliptic Curve Arithmetic in $GF(p^n)$. In H. Imai and Y. Zheng, editors, *Public Key Cryptography — Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000*, Volume 1751 of *Lecture Notes in Computer Science*, pp. 405–421. Springer-Verlag, Berlin, Heidelberg, New York, 2000.

[LV00] A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. In H. Imai and Y. Zheng, editors, *Public Key Cryptography — Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000*, Volume 1751 of *Lecture Notes in Computer Science*, pp. 446–465. Springer-Verlag, Berlin, Heidelberg, New York, 2000.