

## 楕円曲線暗号のソフトウェア並列実装

### Software Implementation of Parallel Elliptic Curve Cryptosystem

小林 鉄太郎\*

Tetsutaro Kobayashi

[kotetsu@isl.ntt.co.jp](mailto:kotetsu@isl.ntt.co.jp)

青木 和麻呂 \*

Kazumaro Aoki

[maro@isl.ntt.co.jp](mailto:maro@isl.ntt.co.jp)

星野 文学 \*

Fumitaka Hoshino

[fhoshino@isl.ntt.co.jp](mailto:fhoshino@isl.ntt.co.jp)

小黒 博昭 \*

Hiroaki Oguro

[oguro@isl.ntt.co.jp](mailto:oguro@isl.ntt.co.jp)

**Abstract**— A bitslice implementation method is proposed for scalar multiplications of an elliptic curve over  $\mathbf{F}_{2^m}$ . This method is suitable for software implementation. The number of parallel computation is the word size of CPUs. (e.g., 32, 64, etc.) Using the method, we can efficiently perform bitwise operation. In theory, it is about twice faster than traditional implementation. We discuss the theoretical efficiency and an actual implementation on Pentium III PC of the bitslice implementation and traditional one.

**Keywords:** elliptic curve cryptosystem, scalar multiplication, binary field, finite field, parallel implementation, bitslice implementation

#### 1 はじめに

楕円DSA署名や楕円ElGamal暗号を実装する場合、楕円曲線上の点のスカラー倍演算が主な処理となる。この部分の高速化手法について様々な研究が行なわれてきた。また、暗号や署名に用いられる楕円曲線としては  $\mathbf{F}_p$  ( $p$  は素数) や  $\mathbf{F}_{2^m}$  の上の表現が IEEE P1363 や ISO/IEC JTC1/SC27 で標準化され、多くの企業や団体で実装が行なわれてきた。

とくに、 $\mathbf{F}_{2^m}$  上の楕円曲線暗号は、専用のハードウェアを使用する場合に非常に効率が良く、ICカードなどで公開鍵暗号を実現する場合に多く用いられている。

一方ソフトウェアでの実装では、大きなテーブルを使う実装法などがあったが十分に効率が良いとはいえない。楕円暗号の機能をもつICカードが大量に流通した場合、一定の時間内にできるだけ多くの暗号や署名処理を行うことが必要となる。例えば電子投票や電子行政などのサーバー系のシステムにおいて多くの暗号や署名処理を一度に行なうことが予想される。この問題を解決するには、専用のハードウェアで処理する方法もあるが、今日では Pentium などの高性能なプロセッサが安価になっており、汎用のコンピュータで演算するほうがコストパフォーマンスが良い。すなわち、大量の暗号や署名処理を一度に行なわなければ

ならない場合に高速なソフトウェアでの処理が必要となる。

本論文は、DESなどの実装で用いられたビットスライス実装 [1] に着目し、 $\mathbf{F}_{2^m}$  上の楕円曲線暗号に適用する方法について述べる。

この方法は、計算機CPUのワード長(32, 64, など)を  $w$  とし、 $w$  個の独立な楕円曲線の演算を同時に行なう。これにより、ビット単位の演算が多い  $\mathbf{F}_{2^m}$  の演算を効率良く行なうことができる。ビットスライス実装では全ての演算がビット単位で行なえることから、多くの高速化技法が仮定する理想的な計算機モデルに近く、オーバーヘッドを減らすことができることも利点である。逆に、1回の暗号署名処理が必要な場合は  $w$  個の独立な演算を行なう意味が失われるため、ほとんど利点がない。

本論文では電子投票など、十分に多量の暗号署名処理を行なうことを見定し、通常の実装とビットスライス実装の比較を行なう。

この後の構成は次の通りである。第2章で、ビットスライス実装の基本的な概念を述べる。第3章で、通常の実装法との理論的な計算量の比較を行ない、第4章で、NIST推奨パラメータを Pentium III プロセッサ上で実装した場合の効果を予想する。第5章はビットスライス効果のまとめを行なう。

\* 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 Information Sharing Platform Laboratories, 1-1 Hikarinooka Yokosuka-shi Kanagawa 239-0847 Japan

## 2 ビットスライス実装法

本章では、ビットスライス実装法の基本的アイデアを述べる。

演算は、ワード長が  $w$  ビットのプロセッサを用いて行なう。1ワード ( $w$  ビット) の **AND**, **XOR**, **SHIFT** 演算のコストをそれぞれ  $a$ ,  $x$ ,  $s$  とする。ただし、**SHIFT** 演算は任意のビットシフトを  $s$  のコストで行なうことができ、 $n$  ワードのシフトを  $ns$  のコストで行なうことができると仮定する。

$\mathbf{F}_{2^m}$  の上の非超特異な橙円曲線

$$E : y^2 + xy = x^3 + ax^2 + b \quad (a, b \in \mathbf{F}_{2^m})$$

を考える。また、 $\mathbf{F}_{2^m}$  の元は  $m$  次既約多項式  $p(t)$  を定義多項式として  $\mathbf{F}_2$  上の  $m$  次多項式の剩余環として表現する。今後、 $p(t)$  は  $u$  項多項式とする。また、 $p(t)$  の最上位項を除いた多項式  $p(t) - t^m$  は十分に次数が低いと仮定する。

本稿では、同時に  $w$  個の独立な  $\mathbf{F}_{2^m}$  上の演算を行なう場合を考え通常の実装とビットスライス実装を比べる。

通常の実装では、 $\mathbf{F}_{2^m}$  上の 1 個元を  $\lceil \frac{m}{w} \rceil$  ワードで表現する。したがって、 $w$  個の元は  $w \lceil \frac{m}{w} \rceil$  ワードで表現される（図 1 参照）。

$\mathbf{F}_{2^m}$  の 2 つの元の加算は、**XOR** 演算を  $\lceil \frac{m}{w} \rceil$  回行なうだけで良い。

一方  $m$  次多項式の自乗を行なう場合、ハードウェアでの実装では、 $m$  個のデータの並びかえを行なうだけで良いが、ソフトウェア実装では、データの並びかえのために演算を行なう必要があり、1回あたり  $\lceil \frac{m}{w} \rceil (\log w)(a+x+s)$  程度のコストで実現できることが明らかである。

ビットスライス実装は、この問題をデータ構造を工夫することによって解決する。通常の実装では、図 1 のように表現されていた  $w$  個の  $\mathbf{F}_{2^m}$  の元を図 2 のように表現する。

これは、 $j$  番目の  $\mathbf{F}_{2^m}$  の元の  $i$  ビット目を  $e_{(i,j)}$  とすると、

$$e_i = \sum_{j=0}^{w-1} e_{(i,j)} 2^j \quad (0 \leq i < m)$$

の  $m$  ワードによって表すことに相当する。これをビットスライス表現と呼ぶ。

ビットスライス表現では、各ビットの情報が別々のワードに格納されるため、ビットごとの順番の入れかえなどを簡単に行なうことが出来る。たとえば  $m$  次多項式の自乗を行なう場合は、ハードウェアでの実装と同様にワードの入れかえのみで行なうことができ、コスト 0 で演算できると見なせる。

ビットスライス表現は  $\mathbf{F}_{2^m}$  の自乗演算を行なう場合だけでなく、 $\mathbf{F}_{2^m}$  の乗算演算などの場合もシフト演算など

$m$ ビット									
1	1	0	1	1	0	1	1	0	1
1	1	0	0	1	0	1	1	1	1
1	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	1	1

図 1: 通常の実装 (非ビットスライス) のデータ表現例

$m$ ビット										↑ $w$ ビット ↓
1	1	0	1	1	0	1	1	1	0	
1	1	0	0	1	0	1	1	1	1	
1	1	0	0	0	0	1	1	0	0	
0	0	0	0	1	0	1	1	1	1	

図 2: ビットスライス法のデータ表現例

を省略できるため効率が良い。その理論的な効率の比較を第 3 章で行なう。

通常のデータ表現からビットスライス表現に変換するコストは  $\lceil \frac{m}{w} \rceil \frac{w}{2} (\log w)(2s + 3x + a)$  で実現でき、乗算一回のコストと比較して少ない。

## 3 計算量

本章では、理論的計算量によってビットスライス実装と通常の実装法による実装を比較する。

代表的な  $\mathbf{F}_{2^m}$  の表現には多項式基底に基づくものと、正規基底に基づくものがあるが、ここでは Karatsuba 法 [2] による乗算の高速化効果が大きいことを考慮して多項式基底を選択する。

CPU のワード長を  $w$  ビットとしたとき、 $\mathbf{F}_{2^m}$  の加算、乗算、自乗、逆元演算のコストを表 1 に、橙円曲線上の加算、 $k$  倍算のコストを表 2 に示す。ただし、表を見やすくするため、 $m$  の次数が一番大きい項のみを示している。また、ビットスライス実装における演算は 1 回の実行で  $w$  個の独立した演算を並列実行していると見なせるため、1 回の実行時間を  $w$  で割った時間（1 回当たり実行時間）を表示している。

$\mathbf{F}_{2^m}$  の乗算については、教科書法と Karatsuba 法の両方の計算量を考察する。Karatsuba 法は、再帰的に適用することによって効率を高めることができる。理論的には分割すればするほど乗算の回数を減らすことができるが、実際には  $\mathbf{F}_2$  の乗算と加算のコストが同程度と考えられるので、最小分割数を 1 ビットにしない方が効率が良い。

例として、ビットスライス実装で  $\mathbf{F}_2$  上 1 次多項式の乗算を考える。この演算は図 3 や図 4 に示すように行なうことができる。教科書法のコストが  $4a + x$  なのに對し、Karatsuba 法のコストは  $3a + 4x$  であり、 $a$  と  $x$  が同程度のコストであると考えると教科書法のほうが効率が良い。

表 1:  $\mathbf{F}_{2^m}$  上の演算コスト

		ビットスライス実装	通常の実装
加算 $A$	$s$	0	0
	$x$	$\frac{m}{w}$	$\lceil \frac{m}{w} \rceil$
	$a$	0	0
乗算 (教科書法) $M_S$	$s$	0	$w \lceil \frac{m}{w} \rceil + 2m$
	$x$	$\frac{m^2}{w}$	$m \lceil \frac{m}{w} \rceil$
	$a$	$\frac{m^2}{w}$	$m \lceil \frac{m}{w} \rceil$
乗算 (Karatsuba 法) $M_K$	$s$	0	$w \cdot \left(\frac{m}{w}\right)^{\log 3}$
	$x$	$\frac{13}{3w} \cdot m^{\log 3}$	$(9w - 2) \cdot \left(\frac{m}{w}\right)^{\log 3}$
	$a$	$\frac{16}{9w} \cdot m^{\log 3}$	$w \cdot \left(\frac{m}{w}\right)^{\log 3}$
自乗 $S$	$s$	0	$2 \lceil \frac{m}{w} \rceil \log w$
	$x$	$\frac{um}{2w}$	$2 \lceil \frac{m}{w} \rceil \log w$
	$a$	0	$2 \lceil \frac{m}{w} \rceil \log w$
逆元 $I$	$s$	0	$4m \lceil \frac{m}{w} \rceil$
	$x$	$\frac{13}{3w} \cdot w_H(m-1) \cdot m^{\log 3}$	$4m \lceil \frac{m}{w} \rceil$
	$a$	$\frac{16}{9w} \cdot w_H(m-1) \cdot m^{\log 3}$	0

注:  $u$  は定義多項式の項数、 $w_H(x)$  は  $x$  のハミング重みを表す。 $\log$  の底は 2 とする。

表 2: 楕円曲線演算コスト

	座標	演算	ビットスライス実装	通常の実装
楕円加算	Affine	$s$	$2M_K + S + I$	
		$x$	0	$4m \lceil \frac{m}{w} \rceil$
		$a$	$\frac{13}{3w} w_H(m-1) m^{\log 3}$	$4m \lceil \frac{m}{w} \rceil$
	Standard projective $(x/z, y/z)$	$s$	$12M_K + S$	
		$x$	0	$12w \cdot \left(\frac{m}{w}\right)^{\log 3}$
		$a$	$\frac{52}{w} m^{\log 3}$	$12w(9w-2) \cdot \left(\frac{m}{w}\right)^{\log 3}$
	Projective [5] $(x/z, y/z^2)$	$s$	$9M_K + 4S$	
		$x$	0	$9w \cdot \left(\frac{m}{w}\right)^{\log 3}$
		$a$	$\frac{39}{w} m^{\log 3}$	$9w(9w-2) \cdot \left(\frac{m}{w}\right)^{\log 3}$

×)	a	b
	c	d
	ad	bd
	ac	bc
	ac	bc + ad
		bd

$$M_S(2) = 4a + x$$

図 3: 教科書法 2 ビット乗算のコスト

×)	a	b
	c	d
	-bd	bd
	a'c'	
	ac	-ac
	ac	bc + ad
		bd

ただし、 $a' = (a + b)$ ,  $c' = (c + d)$  とする。

$$M_K(2) = 3a + 4x$$

図 4: Karatsuba 法 2 ビット乗算のコスト

したがって、本稿では  $a \simeq x$  と仮定した場合に最適な最小分割数を考える。まず、 $m$  ビットの多項式の乗算の計算量を  $M(m)$  と定義する。教科書法、Karatsuba 法を用いた場合の計算量  $M_S(m)$ ,  $M_K(m)$  は、以下の式で表すことが出来る。

$$M_S(m) = 4M\left(\frac{m}{2}\right) + (2m - 3)x \quad (1)$$

$$M_K(m) = 3M\left(\frac{m}{2}\right) + (4m - 4)x \quad (2)$$

式 (1), (2) をグラフにすると図 5 となり、最小分割単位を 4 ビットとするのがもっとも効率が良いことがわかる。

また、通常の実装でも Karatsuba 法を用いて乗算を高速化することができるが、 $w$  ビット以下の分割はオーバー

ヘッドが大きいため、最小分割数を  $w$  とする。  
逆元演算については、ビットスライス実装ではそのデータ構造の特性から、条件分岐を伴う方法は利用することが難しい。Almost Inverse 法 [3] のようにあるビットが 1 かどうかによって行なう処理が決まるような方法は、 $w$  個のデータの該当ビットのうち一部が 1 で一部が 0 である場合に行なう処理を決定できない。よって、ビットスライス実装では条件分岐をせずに逆元演算を行なえる伊東-辻井 [4] 法を、通常の実装では Almost Inverse 法を逆元演算として用いることとする。

楕円曲線演算については、アフィン座標、標準的な射影楕円曲線演算について、アフィン座標、標準的な射影楕円曲線演算について、アフィン座標 ( $x/z, y/z$ )、そして文献 [5] で提案された射影座標 ( $x/z, y/z^2$ ) の各々について考察する。楕円  $k$  倍算については、 $\phi$  進 NAF 法 [6, 7] を利用する。この演算法を用いれば、楕円曲線のスカラー倍演算を  $\frac{m}{3}$  回の楕円加算と  $2m$  回の  $\mathbf{F}_{2^m}$  上の自乗によって行なうことができる。楕円  $2m$  回の  $\mathbf{F}_{2^m}$  上の自乗によって行なう必要はないため、本稿では比較しない。

#### 4 実装による比較

この章では、Pentium III PC 上で実際に楕円演算を実装する場合にビットスライス実装と通常の実装の演算速度を比較する。

楕円曲線は NIST が推奨している  $\mathbf{F}_{2^m}$  上のパラメータ [8] を使用する。以下、typewriter font の数値は 16 進表現とする。

##### [NIST-recommended elliptic curve K-163]

$$E : y^2 + xy = x^3 + ax^2 + b, \quad a = 1, b = 1$$

$$\text{最小多項式} : p(t) = t^{163} + t^7 + t^6 + t^3 + 1$$

ベースポイントの位数 :

$$r = 4\ 00000000\ 00000000\ 00020108\ a2e0cc0d\ 99f8a5ef$$

$$\text{ベースポイント} : G = (G_x, G_y)$$

$$G_x = 2\ fe13c053\ 7bbc11ac\ aa07d793\ de4e6d5e\ 5c94eee8$$

$$G_y = 2\ 89070fb0\ 5d38ff58\ 321f2e80\ 0536d538\ ccdcaa3d9$$

想定するマシンの環境は以下の通りである。

##### [マシン環境]

CPU : Pentium III 700MHz

メモリ : 128MB

上記の条件で実装する場合、Pentium III では 64 ビットの MMX レジスタを使用することができるため、 $m = 163$ ,  $w = 64$  ということになり、表 3 に示すような結果が予想される。ただし、 $x = a = s = \frac{1}{700 \cdot 10^6}$  秒とし、任意の命令を 2 並列で演算することができ、依存関係を考慮しない。また、 $x, a, s$  以外のオーバーヘッド (メモリ読み書きなど) を無視する。

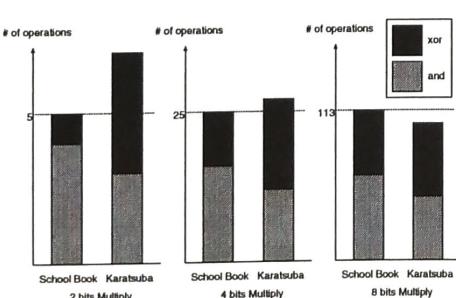


図 5: 教科書法と Karatsuba 法の比較

表 3: 楕円曲線演算コスト ( $m = 163$ , Pentium III の場合, 単位は  $\mu$  秒)

	座標	ビットスライス実装	通常の実装
楕円加算	Affine	3.073	3.746
	Projective [5] ( $x/z, y/z^2$ )	1.921	11.990
楕円 $k$ 倍算	Affine	168.400	233.600
	Projective [5] ( $x/z, y/z^2$ )	105.900	681.700

## 5まとめ

$F_{2^m}$  上の楕円曲線演算をビットスライス実装を用いて行なう方法を提案した。

この方法は、 $w$  個の楕円曲線演算を並列に行なう方法である ( $w$  は、プロセッサのワード長 (32, 64, など))。ビット単位の処理を効率良く行なうことが出来るという利点があり、プロセッサのワード長が大きいほど  $F_{2^m}$  乗算や自乗を効率良く行なうことが出来る。例えば  $w = 64$  と仮定すると、ビットスライス法は約 6 倍高速に  $F_{2^m}$  乗算を行なうことが出来る。

通常の実装では  $F_{2^m}$  上の楕円演算は Affine 座標で実現するほうが効率が良いが、ビットスライス実装では Projective 座標で表現する方が効率が良い。Pentium III プロセッサ上で、163 ビット楕円曲線暗号を実装する場合、理論的には通常の実装法 (Affine 座標) による実装よりもビットスライス実装 (Projective 座標) で実現する方が 2 倍以上高速に楕円スカラー倍演算を行なえる。

この結果から、同時に大量の楕円曲線暗号や署名の処理を行なう場合にはビットスライス法が有効であるといえる。

また、本手法は、楕円曲線以外にも超楕円曲線など  $F_2$  の拡大体を用いる演算全般に適用することもできる。

## 参考文献

- [1] E. Biham, "A Fast New DES Implementation in Software," Fast Software Encryption – FSE '97, Lecture Notes in Computer Science 1267, pp.260-271, Springer-Verlag, 1997.
- [2] D. E. Knuth, The Art of Computer Programming 第4分冊 準数値算法 / 算術演算, pp.112, サイエンス社, 1986
- [3] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast Key Exchange with Elliptic Curve System," Advances in Cryptology – CRYPTO '95, Lecture Notes in Computer Science 963, pp.43-56, Springer-Verlag, 1995.
- [4] T. Itoh and S. Tsujii, "A Fast algorithm for Computing Multiplicative Inverses in Finite Fields Using Normal Basis," IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan), Vol.J 70-A No.11, 1987.
- [5] J. López and R. Dahab, "Improved Algorithms for Elliptic Curves Arithmetic in  $GF(2^n)$ ," Selected Areas in Cryptography – SAC '98, Lecture Notes in Computer Science 1556, pp.201-212, Springer-Verlag, 1999.
- [6] N. Koblitz, "CM-Curves with Good Cryptographic Properties," Advances in Cryptology – CRYPTO'91, Lecture Notes in Computer Science 576, pp.279-287, Springer-Verlag, 1992.
- [7] J. A. Solinas, "An Improved Algorithm for Arithmetic on a Family of Elliptic Curves," Advances in Cryptology – CRYPTO '97, Lecture Notes in Computer Science 1294, pp.357-371, Springer-Verlag, 1997.
- [8] National Institute of Standards and Technology, Digital Signature Standard, FIPS Publication 186-2, February 2000.  
<http://csrc.ncsl.nist.gov/fips/fips186-2.pdf>