マルチメディア通信と分散処理　101－7
コンピュータセキュリティ　12－7
（２００１．２．２０）

マルチスレッドによる楕円k倍算の高速化法

奥野 琢人* 若山 公威* 村瀬 晋二** 鈴木 春洋** 岩田 彰*
* 名古屋工業大学 電気情報工学科
** 株式会社 シーティーアイ SI事業部

本稿では、素数p>2を法とする楕円曲線上でのk倍算を2分割し、並列に計算を行なわせる事で高速に処理する手法について述べる。この手法では、IEEE P1363等に記載されている標準的なk倍算、すなわち、Jacobian座標系やNAF-Signed Binary表現、Window Methodによる高速化をそのまま活用でき、さらに乗数を2分割する事で、従来の手法に比べ理論的に1.5倍の高速化を達成する事が出来た。また、マルチスレッドを使った共有メモリ型並列処理による実装実験の結果、スレッドのオーバヘッドを多少含みながらも、ほぼ理論通りの1.45倍の高速化を実験値として得ることが出来た。

A Fast Computation of the Elliptic Curve Exponentiation by using Multithread

Takuto OKUNO* Kimitake WAKAYAMA* Shinji MURASE** Shunyo SUZUKI** Akira IWATA*
* Department of Electrical and Computer Engineering, Nagoya Institute of Technology
** CTI Co. Ltd.

In this paper, we propose a fast processing method for the computation of the elliptic curve exponentiation by parting a multiplier factor and computing it in parallel. Indeed, this method uses obvious elliptic curve exponentiation mentioned on the IEEE P1363 and other papers, such as Jacobian coodinate, NAF-Signed Binary, and Window Method. Then, it is possible to part a multiplier factor and accomplish that its processing speed be theoretically 1.5 times faster than standard method. Moreover, we could get a result of processing time same as our theory at the implimentation of this method by using multithread.

1. はじめに

近年、インターネットにおける暗号技術の利用が非常に活発である。ここでの暗号技術は総称して、PKI（Public Key Infrastructure：公開鍵基盤）と呼ばれ、通信技術や電子証明書技術など多くの暗号の応用技術を含んでおり、文字通り公開鍵暗号をベースとしている。

現在、PKIで主に使われている公開鍵暗号はRSA暗号であるが、その鍵長の多くが1024bit, 2048bitといった巨大ものになりつつあり、更に安全性を高めるために大きな鍵長を利用する可能性がある。し

かし、鍵長が大きくなればなるほど、それを利用する電子証明書のサイズの増大や、通信時における署名・検証にかかる処理時間の増大をまねく事となる。

これに対し楕円曲線暗号は、安全性を保ちながら鍵長を小さくでき、署名・検証に関する処理時間がRSA暗号よりも短いため、近年では多くの研究、実装が行なわれている。また、基本的な楕円曲線上の点の演算については、IEEE P1363 [1] によってアルゴリズムが記載されており、すでに多くの実装に用いられている。

本稿では、P1363 や伊藤ら[2] による汎用的な楕円曲線上の点の k 倍算アルゴリズムを用いながら、更に k 倍算を 2 分割し並列に計算する事で高速に演算を処理する方法を述べる。さらに理論的に求められた高速化率に対し、Solaris2.7 上で POSIX Thread を使い、共有メモリ型並列処理による実装実験を行なった。その結果、ほぼ理論値通りの値を得ることが出来た。なお本稿では、素数 p>2 を法とする楕円曲線を中心に扱うものとする。

2. 準備

2.1. 標準的な楕円曲線上の演算

ここでの標準的な楕円曲線とは、素数 p>2 を法とする体 F_p 上での曲線である。その式は、下記のように表される。

$$E:\ y^2 = x^3 + ax + b$$
$$(a,b \in F_p,\ 4a^3 + 27b^2 \neq 0)$$

この楕円曲線上での点 $P_0(x_0,y_0)$, $P_1(x_1,y_1)$ の加算$\mathcal{A}$や倍算$\mathcal{D}$は下記のように定義されている。

$$x_2 = \lambda^2 - x_0 - x_1$$
$$y_2 = \lambda(x_0 - x_2) - y_0$$
$$\lambda = \begin{cases} \dfrac{y_1 - y_0}{x_1 - x_0} & \text{if} \quad P_0 \neq P_1 \\ \dfrac{3x_0^{\,2} + a}{2y_1} & \text{if} \quad P_0 = P_1 \end{cases}$$

この形の楕円曲線点の演算をアフィン座標系での加算・倍算といい、その計算時間を巨大数の乗算をもとに表わすことが出来る。

すなわち、加算$\mathcal{A}$ にかかる計算時間を $t(\mathcal{A})$とすると、

$$t(\mathcal{A}) = I + M + S$$

であり、倍算にかかる計算時間は、

$$t(\mathcal{D}) = I + M + 2S$$

のように表わされる。ここで M, S, I はそれぞれ、巨大数の乗算にかかる時間、巨大数の 2 乗にかかる時間、法を p とする巨大数の逆元演算にかかる時間である。一般的に巨大数の乗算は、加算や減算に比べ多くの処理時間を必要とするため、乗算を起点として計算時間を評価し、その他の加算や減算、小さな定数の乗算（乗数が 1CPU ワードやシフトによる倍算が可能な場合）による計算時間は誤差として評価しない。

また、巨大数の 2 乗と乗算にかかる計算時間の関係は、S = 0.8M 程度と仮定される事が多く、一方で素数 p を法とする巨大数の逆元演算では、I = 10M ~ 30M 程度の演算時間がかかるといわれている。

以降、本稿では I = 10M、S = 0.7M として計算を行なっていく。特に、2 乗にかかる時間を S=0.7M としたのは、実装に使用した暗号ライブラリ AiCrypto [3] の実測値がこの値に近かったためである。

よって、先の楕円曲線上での点の加算$\mathcal{A}$と倍算$\mathcal{D}$にかかる計算時間は、

$$t(\mathcal{A}) = I + M + S = 11.7M$$
$$t(\mathcal{D}) = I + M + 2S = 12.4M$$

のように表わす事が出来る。

2.2. Jacobian 座標系での演算

2.1 で示されている座標系では、処理時間が大きな逆元演算が含まれるため、全体の処理時間が遅くなる傾向がある。そこで、逆元演算を行なわずに点の乗算や倍算を行なうのがこの座標変換を利用する手法である。ここで述べる手法は、P1363 に記載されている標準的な手法であるが、より高速化を目指した座標変換も提案されている。[4]

Jacobian 座標系は、$x = X/Z^2$, $y = Y/Z^3$ となる (X,Y,Z) よって表わされ、その曲線は、

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

と表わされる。

この座標系での点 $P_0(X_0,Y_0,Z_0)$, $P_1(X_1,Y_1,Z_1)$ の加

算$\mathcal{A}$は、

$$U_0 = X_0 Z_1^2$$
$$S_0 = Y_0 Z_1^3$$
$$U_1 = X_1 Z_0^2$$
$$S_1 = Y_1 Z_0^3$$
$$W = U_0 - U_1$$
$$R = S_0 - S_1$$
$$T = U_0 + U_1$$
$$M = S_0 + S_1$$
$$Z_2 = Z_0 Z_1 W$$
$$X_2 = R^2 - TW^2$$
$$V = TW^2 - 2X_2$$
$$Y_2 = (VR - M W^3)$$

のように求められる。また、倍算$\mathcal{D}$も同様に、

$$M = 3X_1^2 + a Z_1^4$$
$$S = 4X_1 Y_1^2$$
$$T = 8Y_1^4$$
$$X_2 = M^2 - 2S$$
$$Y_2 = M(S - X_2) - T$$
$$Z_2 = 2Y_1 Z_1$$

のようにして求められる。これらJacobian座標系での加算、倍算にかかる計算時間は、

$$t(\mathcal{A}) = 12M + 4S = 14.8M$$
$$t(\mathcal{D}) = 4M + 6S = 8.2M$$

となり、アフィン座標系よりも点の加算に関しては低速であるが、点の倍算については大幅に高速である事が分かる。

3. 曲線上のk倍算

楕円曲線上の点のk倍算とは、ある曲線上の点G(x,y)をk(=整数)倍し、点Pを求める演算である。この演算を行なうためには、kを2進数で表現し各bit毎に加算や倍算を繰り返して結果を得るバイナリ法が基本的に使われる。さらにP1363には、このバイナリ法を高速化する手法として、NAF-Signedバイナリ表現によるk倍算の高速化手法が記載されている。

ここではk倍算の一般的な高速化手法である、バイナリ法、NAF-Signedバイナリ表現、Window Methodに関して簡単に述べ、その計算時間の評価を行なっていく。

3.1. バイナリ法

バイナリ法は、楕円曲線上の点のk倍算において基本となる手法である。例として、点Pの26319998倍算を考えると、

$$26319998P = 2^2(2(2(2(2^4(2(2(2^3(2(2^4(2^3(2P+P)+P)+P)+P)+P)+P)+P)+P)+P)+P)+P)$$

となる。これは即ち、

26319998 =1100100011001110001111110 (2進数)

を左から1bit毎にトレースし順に倍算を行い、bitが立っていればPを加算する事を繰り返す方法である。この方法でk倍算$\mathcal{K}$にかかる処理時間$t(\mathcal{K})$は、

$$t(\mathcal{K}) = 1/2\ m\ t(\mathcal{A}) + m\ t(\mathcal{D})$$
$$= 15.6mM$$

となる。ここで、mはkのbit数であり、$2^{m-1} \leqq k < 2^m$を満たす整数である。また、kの2進数表現では1, 0の出現率は同じであると考えられるため、点の加算は、m/2回程度行なわれると仮定している。

3.2. Window Method

バイナリ法では、1bit毎に演算を行なっていたが、これを数bitのブロック毎に行う手法がWindow Methodである。この手法では先に加算の基となるテーブルを作成し、それを演算に利用することでバイナリ法よりも大幅に速度が向上する。先の点Pの26319998倍に5bitのWindowを適用した場合、

$$26319998P = 2^2(2^5(2^5(2^8(25P)+25P)+24P)+31P)+2P$$

のように演算が行なわれ、m/6回程度の加算と、テーブル作成のために、$15\ t(\mathcal{A}) + 3\ t(\mathcal{D})$ (= 246.6M)の計算時間がかかる。よって$t(\mathcal{K})$には、

$$t(\mathcal{K}) = 1/6\ m\ t(\mathcal{A}) + m\ t(\mathcal{D}) + 246.6M$$
$$= 10.66mM + 246.6M$$

の計算時間がかかる事になる。

3.3. NAF-Signedバイナリ表現

NAF-Signedバイナリ表現は、2進数の0,1を0,1,-1の数値に置き換え、バイナリ値の0の値を増やし、点の加算の回数を減らす手法である。この手法では、負の数を利用するため、Window Methodと併用した場合、テーブル作成にかかる時間を半減する事が出来る。

すなわち、点 P の 26319998 倍を 4bit の Window と併用した場合、

$26319998P = 2^7(2^5(2^5(2^5(6P)+9P)-6P)-7P)-2P$

となり、明らかにテーブル作成にかけられる処理時間を減少する事が出来る。さらに、ビット値が 0 である割合も 2/3 程度になるため、点の加算回数も m/6 回程となる。

その結果、3.1、3.2、3.3 を全て併用した場合の計算時間は、

$t(\mathcal{K}) = 1/6\ m\ t(\mathcal{A}) + m\ t(\mathcal{D}) + 5\ t(\mathcal{A}) + 2\ t(\mathcal{D})$

$= 10.66mM + 90.4M$

と計算される。

4. k 倍算の並列化

ここでは、前節で解説した従来の k 倍算の高速化法を全て活用しながら並列に k 倍算を行なう手法を示す。

4.1. k 倍算の分割

ある点 G を k 倍する場合、乗数 k を $k=k_1+k_2$ を満たす k_1, k_2 に分割する事で、$kG= k_1G + k_2G$ とする事が出来る。

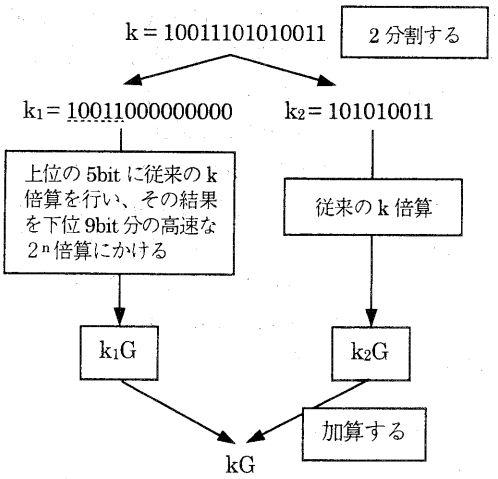

図 1　並列 k 倍算の流れ

そこで、k_1 には k の上位の桁を、k_2 には k の下位の桁を割り当て、それぞれ個別に解答となる点 k_1G, k_2G を求める。ここで、k_2G には従来の k 倍算のみを適用するが、k_1G を実行する場合は上位の桁に従来の k 倍算を適用し、その結果を、下位の 0 が連続する桁数分だけ倍算を連続して行なう演算（以降、高速な 2^n 倍算と呼ぶ）で計算する。

そして最後に、k_1G, k_2G を加算する事で kG を得ることが出来る。この流れを表わしたものが図 1 である。

この並列処理では、処理の途中に同期変数などは全く必要とせず、完全に並列に実行する事ができ、最後の点の加算にのみ、待ち状態が発生する。

よって、k_1G と k_2G の処理時間が近いほど全体の処理速度が向上するため、最適な bit 数で乗数 k を分割する必要がある。この最適な分割 bit 数は理論的に求めることができ、後の節ではその数値を実験でも求める。

4.2. 高速な 2^n 倍算

2.2 節で示されている Jacobian 座標系での倍算$\mathcal{D}$は、連続で行なう事により、前回のループで計算された値を再利用する事が出来る。この加算鎖によって、2^n 倍算の 1 回分のループである倍算$\mathcal{D}'$にかかる計算時間は、

$t(\mathcal{D}') = 4M + 4S = 6.8M$

となり、通常の倍算よりも高速に演算が行なわれる。

4.3. 最適分割 bit の導出

最適な分割 bit は、k_1G と k_2G の処理時間がほぼ等しくなる bit 数だと考えられる。よって、m を k の bit 数、x を最適な分割 bit 数とすると、

$10.66(m-x)M + 6.8xM + 90.4M$

$= 10.66xM + 90.4M$

$x= 0.734m$

となり、x と m の関係が求められる。

この結果より、最適な分割 bit で分割して並列に

表 1　k 倍算の計算時間と速度比

	k 倍算 Binary + NAF + Window	並列 k 倍算 Binary + NAF + Window	速度比
m=160 (x=118)	1796M	1355.3M	75.5 %
m=192 (x=141)	2137.1M	1607.7M	75.2 %
m=256 (x=188)	2819.4M	2108.5M	74.8 %

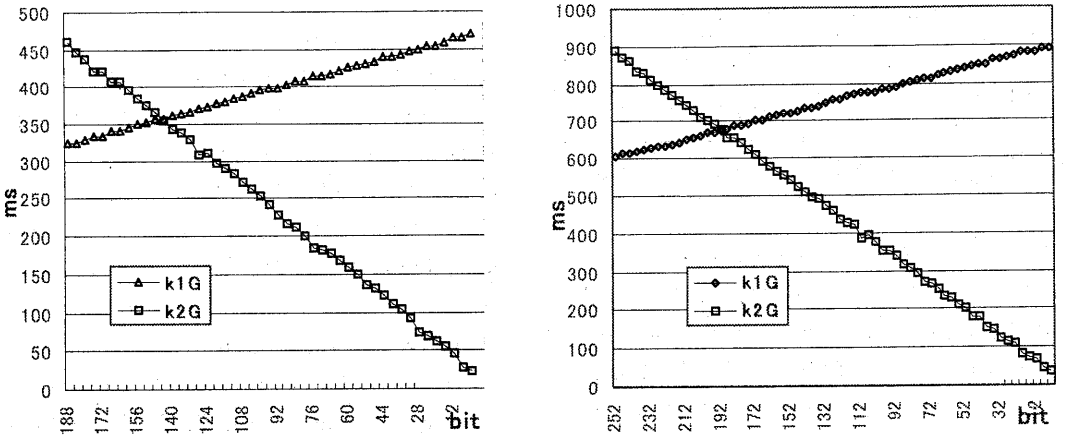

図2　実装実験による最適分割 bit の導出

計算を行なう k 倍算と、従来の k 倍算の計算時間の比較を行なう事ができ、表としてまとめたものを表1に示した。

この表に示されるとおり、k の bit 数ごとに一意の分割点が存在し、それぞれの並列 k 倍算は従来の k 倍算に比べて、計算時間において約 1.5 倍高速である事が分かる。

5.　マルチスレッドによる実装実験

前節での計算結果により、並列 k 倍算は従来の k 倍算より 1.5 倍高速である事が示された。ここではこの結果を踏まえて、Solaris 2.7 の POSIX Thread を使用して並列 k 倍算の実装を行い、その理論値の検証を実験によって行なう。

5.1.　実験環境

実験に使用した環境は表 2 の通りである。ここでは、POSIX Thread を利用したが、Solaris2.7 では使用する LWP (Light Weight Process) の個数を指定しないと並列に処理を行なえないため、先に thr_setconcurrency 関数により 2 個の LWP を確保している。

表 2　実験環境

マシン:	Sparc Server 1000
MPU :	SuperSPARC 50MHz (×8)
OS :	Solaris 2.7
コンパイラ:	Gcc 2.7.1

また、楕円曲線点の演算に関しては、本研究室で作成した暗号ライブラリ AiCrypto をベースとして使用している。残念ながら、現在のところ AiCrypto には、巨大数演算のアセンブラ記述によるルーチンが無く、剰余算アルゴリズムの改善を必要とするため、楕円曲線演算によく利用される GNU multiple precision arithmetic library [5] の演算速度には及ばない。これは、後の改良が望まれるところである。

5.2.　最適分割 bit の導出実験

5.1 に示される実験環境を使い、最初に最適分割 bit の導出実験を行なった。ここでは、192bit と 256bit の 2 通りの楕円曲線を利用し、分割点を 4bit づつスライドしながら、k_1G と k_2G の処理時間を出力した。この結果が図 2 である。

左のグラフに示されるとおり、192bit の乗数では 142bit が実験での最適分割 bit であり、その処理時間は 358ms (192bit 楕円曲線)であった。一方、256bit の乗数では、190bit が最適分割 bit であり、処理時間は 668ms (256bit 楕円曲線)であった。

この実験の結果、4.3 節で求められた理論的な最適分割 bit と実装による分割 bit 数は、ほぼ同じである事が示された。また、従来の k 倍算の実行速度がそれぞれ 480ms, 910ms であることを考えると、スレッドのオーバヘッドを含まない並列 k 倍算は従来の k 倍算に比べ、ほぼ理論通りに 1.5 倍の高速化を達成していることが分かる。

表 3　実装における処理時間

	k 倍算	並列 k 倍算	速度比
192bit (142bit)	480ms	371ms	77.3 %
256bit (190bit)	910ms	695ms	76.3 %

5.3.　マルチスレッドによる実装実験

k を実験で導出された最適分割 bit によって分割し、POSIX Thread を使って k_1G と k_2G の演算を完全に並列実行する。両者の演算が終了後、各 Thread を join して求められた k_1G, k_2G から最終的な答えを得ることが出来る。また、このマルチスレッドによる並列 k 倍算の処理時間と従来の k 倍算による処理時間を表 3 に示した。

実際にマルチスレッドの実装によって得られた処理時間は幾らかのスレッドの処理によるオーバヘッドを含んでいるため、理論値による速度比よりも幾らか遅くなっている。しかし、そのオーバヘッドを考慮しても、k 倍算を 2 分割し並列に処理する事で、逐次に行なうよりも、1.45 倍高速である事が示された。

6.　まとめ

本稿では、$p>2$ の素数を法とする楕円曲線に対し、k 倍算を並列化する手法を新たに提案した。すなわち、k 倍算の乗数を最適 bit 数によって 2 分割し、2 つの k 倍算を並列実行して最後に加算を行なう事で完全並列に処理する手法を提示した。

また、乗数の分割に関して最適分割 bit 数を理論的に導出し、ここで示された値が実装実験で示された値とほぼ同じになる事を示した。

この並列 k 倍算では、2 分割して並列に k 倍算を実行するため、従来の逐次的な k 倍算よりも理論的には 1.5 倍、マルチスレッドによる実装では 1.45 倍の高速化を行なう事が可能である。

参考文献

[1] IEEE P1363/D13 (Draft Version 8) Standard Sepecifications for Public Key Cryptography (November 12, 1999)

[2] 伊藤孝一, 武仲正彦, 鳥居直哉, 天満尚二, 栗原靖, "DSP を用いた楕円曲線暗号の高速実装," SCIS'99 pp591-596, Jan.1999

[3] 若山公威, 奥野琢人, 岩田彰, 村瀬晋二, 鈴木春洋, "暗号ライブラリと認証局パッケージの開発," 第 59 回(平成 11 年後期)情報処理学会全国大会, 情報処理学会

[4] 宮地充子, 小野貴敏, Henri Cohen, "効率的な楕円曲線の冪演算(II)," SCIS'98-7.1.D, Jan.1998

[5] Torbjorn Granlund, The GNU MP LIBRARY, ftp://prep.ai.mit.edu/pub/gnu/gmp-2.0.2.tar.gz

A ライブラリ入手経路

本論文で用いられた暗号ライブラリの紙媒体（ライブラリコードを紙に印刷した物）や光・磁気記憶媒体を入手したい場合は、以下の住所に問い合わせを行なってください。

〒466-8555
名古屋市昭和区御器所町
名古屋工業大学 電気情報工学科 岩田研究室

また、本論文の著作者である奥野に対し電子メールを送る事で問い合わせる事も可能です。

okuno@mars.elcom.nitech.ac.jp

さらに、何らかの媒体ではなく電磁気的な記録だけを入手したい場合は、以下の Web サイト

http://mars.elcom.nitech.ac.jp/security/

にアクセスする事によりダウンロードを行なう事が可能です。