コンピュータセキュリティ　18−39
(2002. 7. 19)

効率的なモンゴメリ型楕円曲線のスカラ倍演算 (2)

布田　裕一†　　大森　基司†

† 松下電器産業株式会社 マルチメディア開発センター
〒571-8501 大阪府門真市門真 1006
E-mail: †{futa,ohmori}@isl.mei.co.jp

あらまし　モンゴメリ型楕円曲線は, ワイヤーシュトラス型楕円曲線より, その楕円曲線演算が高速であることが知られている. 楕円曲線暗号の処理では, 楕円曲線のベース点 (固定点) のスカラ倍演算が含まれている. ワイヤーシュトラス型楕円曲線には, ベース点のスカラ倍点演算テーブルを予め用意することにより高速化する方法が考えられている. しかし, 従来考えられていたモンゴメリ型楕円曲線におけるスカラ倍演算に対しては, スカラ倍点演算テーブルを効率よく利用できない. そのため, モンゴメリ型楕円曲線における ベース点のスカラ倍点を高速に計算する方法は知られていなかった. SCIS2002 で筆者らは, 上記のベース点のスカラ倍演算方法を提案した (方式 1 とよぶ). 本発表では, 方式 1 の改良方法について提案する. この提案方法は, 方式 1 より 1.25 倍高速である.
キーワード　楕円曲線, モンゴメリ型楕円曲線, スカラ倍演算, 固定点

Efficient exponentiation of Montgomery-type elliptic curves (2)

Yuichi FUTA† and Motoji OHMORI†

† Multimedia Development Center, Matsushita Electric Industrial Co., Ltd.,
1006 Kadoma Kadoma City Osaka 571-8501, Japan
E-mail: †{futa,ohmori}@isl.mei.co.jp

Abstract　The Montgomery-type elliptic curve is known for their faster arithmetic than the Weierstrass-type elliptic curve. The dominant operation of Elliptic Curve Cryptosystem(ECC) is scalar multiplication of points on an elliptic curve, and it usually includes scalar multiplication of a fixed base point of ECC. For Weierstrass-type elliptic curves, accelerating methods, using the pre-computed table of scalar multiplication of the fixed point, are widely studied. However, such a scheme does not naturally expand to the Montgomery-type elliptic curve. In this paper, we propose fast scalar multiplication methods on Montgomery-type elliptic curves using pre-computed table for the first time. Our method is 1.6 times as fast as the known method for Montgomery-type elliptic curves.
Key words　elliptic curve, Montgomery-type elliptic curve, exponentiation, fixed point

1. はじめに

近年, 楕円曲線上の離散対数問題 (ECDLP) を利用した暗号方式である楕円曲線暗号 [4], [6] が注目されている. 楕円曲線暗号は, 楕円曲線を適当に選ぶと準指数時間攻撃アルゴリズムが現時点では存在しない. このため, 鍵サイズを小さくできる利点をもつ.

楕円曲線暗号で使用されている楕円曲線の多くは, ワイヤーシュトラス型楕円曲線と呼ばれるもので, 方程式が $y^2 = x^3 + ax + b$ と表される. それに対して, 1987 年に Montgomery によって提案されたモンゴメリ型楕円曲線 [8] が最近, 注目されてきている [3], [10]. その理由は, モンゴメリ型楕円曲線は, ワイヤーシュトラス型楕円曲線より, 楕円曲線加算や 2 倍算が高速であるからである.

モンゴメリ型楕円曲線は, 方程式が $By^2 = x^3 + Ax^2 + x$ で表される. モンゴメリ型楕円曲線は, 一部のワイヤーシュトラス型楕円曲線から変換可能である [3]. 一方で, モンゴメリ型楕円曲線には以下の特徴がある:

（1） Y 座標を計算するための公式がなく, 簡単には計算できない

（2） 点 P, Q の加算において $P - Q$ の座標が必要である

上記の (1) は, モンゴメリ型楕円曲線を用いた署名生成アルゴリズムを適用する時に欠点となる. これに対しては, いくつかの方法で改善されている [1], [9], [10].

また, 楕円曲線暗号の主要な計算処理の一つに, 楕円曲線のベース点のスカラ倍演算がある. ベース点は楕円曲線暗号のシ

ステムごとに定まっている場合が多く, その場合, 暗号の計算処理で常に同じものを使用する. このことを利用して, ワイヤーシュトラス型楕円曲線においては, ベース点のスカラ倍点を予め計算し, テーブルに記憶しておくことにより, スカラ倍演算を高速化する方法が考えられている. しかし, モンゴメリ型楕円曲線においては, 予備計算テーブルを用いてベース点のスカラ倍点を高速に計算する方法は知られていなかった.

筆者らは 以前に, 予備計算テーブルを用いた効率的なモンゴメリ型楕円曲線のスカラ倍演算方法を提案した [2]. その方法は, スカラ倍のスカラを順次小さい値に分解する際, 事前計算されたテーブルに記憶された値を利用する頻度が高くなるように分解するものである. スカラ倍演算においては分解されて小さくなった値から順次大きいスカラ倍を計算する. 第 1 のアプローチでは, 2^iG をテーブルに持ち, G のスカラ倍を計算する時に, 必ず 2^iG が出現するように分解し, 予備計算テーブルの効率的利用を可能としている. さらに, このアプローチ 1 においては, P と Q の加算のために必要となる差分点 $P-Q$ の値が, 順次計算を行う途中段階で必ず現れるため, 差分点の計算を別途行うことが必要ではなく, 加算の計算回数が膨らまないようになっている.

しかし, アプローチ 1 では, 従来のテーブルを用いない計算方法と計算量がほとんど変わらないことが判明した. その原因は, 分解後の値の一つが分解前に比べて小さくならない場合があるためである. そこで, 分解後の値が必ず小さくなり, かつ, 2^iG を用いることのできる第 2 のアプローチを考案した. その結果, 従来のテーブルを用いない方法より, 高速にスカラ倍演算を行うことが可能となった. ただ, アプローチ 2 でも例外的な処理が発生し, 処理が大きくなることが判明した.

本稿では, 上記アプローチ 2 より高速な第 3 のアプローチと, それに基づいた予備計算テーブルを用いた効率的なモンゴメリ型楕円曲線のスカラ倍演算方法を提案する. アプローチ 3 は, テーブルとして記憶する点を $2^iG+2^{i-1}G$ にしている. その結果, アプローチ 2 における例外的な処理を無くすことができ, 全体の計算量を削減することができた.

本稿では, アプローチ 3 に基づくアルゴリズムを提案する. 提案法は テーブルの計算を除いた場合, 従来のテーブルを使用しないモンゴメリ型スカラ倍演算方法の平均 1.6 倍, [2] の方法の 1.25 倍程度高速である.

2. 従 来 法

本節では, 本稿で使用するモンゴメリ型楕円曲線及びそのスカラ倍演算方法について説明する.

2.1 モンゴメリ型楕円曲線

ここでは, 本稿で使用するモンゴメリ型楕円曲線について説明する.

[定義 1] (モンゴメリ型楕円曲線)

$By^2 = x^3 + Ax^2 + x (A, B \in GF(p), (A^2-4)B \neq 0, p$: 素数) の方程式で定義される楕円曲線を**モンゴメリ型楕円曲線**とよぶ.

ワイヤーシュトラス型楕円曲線は, すべての楕円曲線を表現可能であり, モンゴメリ型楕円曲線は, 一部のワイヤーシュトラス型楕円曲線を変換することによって得ることができる [3].

モンゴメリ型楕円曲線上の射影座標の点の加算は以下のようになる.

$P=(X_1,Y_1,Z_1), Q=(X_2,Y_2,Z_2), P+Q=(X_3,Y_3,Z_3)$, $P-Q=(X_3',Y_3',Z_3')$ とする.

[**加算**] $(P \neq Q)$

$$X_3 = Z_3'[(X_1-Z_1)(X_2+Z_2)+(X_1+Z_1)(X_2-Z_2)]^2$$
$$Z_3 = X_3'[(X_1-Z_1)(X_2+Z_2)-(X_1+Z_1)(X_2-Z_2)]^2$$

[**2 倍算**] $(P=Q)$

$$X_3 = (X_1+Z_1)^2(X_1-Z_1)^2$$
$$Z_3 = (4X_1Z_1)\left[(X_1-Z_1)^2+\frac{A+2}{4}(4X_1Z_1)\right]$$

Z_3 を求める際の $4X_1Z_1$ は以下のように計算する.

$$4X_1Z_1 = (X_1+Z_1)^2-(X_1-Z_1)^2$$

有限体 $GF(p)$ の乗算, 2 乗算の計算量をそれぞれ, M, S とするとき, 上記加算, 2 倍算の計算量は, $4M+2S$ ($Z_3'=1$ の場合は $3M+2S$), $3M+2S$ である.

ワイヤーシュトラス型楕円曲線では, ヤコビアン座標を用いた場合, 加算が $12M+4S$, 2 倍算は $4M+6S$ である [7]. $S=0.8M$ と仮定するとき, モンゴメリ型楕円曲線は, ワイヤーシュトラス型楕円曲線に比べて, 加算で 2.7 倍 ($Z_3'=1$ の場合は, 3.3 倍), 2 倍算で 1.9 倍高速である.

2.2 モンゴメリ型楕円曲線のスカラ倍演算方法

以下に従来のモンゴメリ型楕円曲線のスカラ倍演算アルゴリズムについて示す [8]. 下記のアルゴリズムは, スカラ倍演算 sG を計算するのに, $((s-1)/2)G$ と $((s-1)/2+1)G$ とその差である G の座標を用いて実行する.

[アルゴリズム 1] **(従来のスカラ倍演算アルゴリズム)**

[入力] k: n ビットの整数, 楕円曲線上の点 G

[出力] スカラ倍点 kG

Step 1 : $k=s\cdot 2^e$ を満たす最大の e と, s を求める.

Step 2 : $c \leftarrow n-e-2$, $P \leftarrow G, Q \leftarrow 2G$

Step 3 : s_c の値により以下の処理を行う. ここで, $s = s_{n-e-1}2^{n-e-1}+s_{n-e-2}2^{n-e-2}+\cdots+s_12+s_0$ である.

(1) $s_c=0$ の場合: $Q \leftarrow P+Q, P \leftarrow 2P$

(2) $s_c=1$ の場合: $P \leftarrow P+Q, Q \leftarrow 2Q$

Step 4 : $c \leftarrow c-1$

Step 5 : $c<0$ であるか判定. $c<0$ の場合, 次のステップへ. それ以外は, Step 3 へ.

Step 6 : $c \leftarrow e$

Step 7 : $c=0$ であるか判定. $c=0$ の場合, P を出力し終了. それ以外は, 次のステップへ.

Step 8 : $P \leftarrow 2P$.

Step 9 : $c \leftarrow c-1$. Step 7 へ.

[**従来法の計算量**]

筆者らが計算量を考察した結果, この計算量は, $k=s\cdot 2^e$ としたとき,

$$\left[(n-2)+\left\lfloor\frac{s}{2^{n-1}}-\frac{1}{2}\right\rfloor\right]ECD+(n-e-1)ECA$$

であることがわかった．ここで，ECD, ECAはそれぞれ楕円曲線2倍算，加算の計算量を示している．$ECD=3M+2S, ECA=3M+2S$(GのZ座標が1であるため) を代入すると，

$$\left[(2n-e-3)+\left\lfloor\frac{s}{2^{n-1}}-\frac{1}{2}\right\rfloor\right](3M+2S)$$

である．kをランダムに与えた場合の計算量の平均は

$$\left(2n-\frac{7}{2}\right)(3M+2S)$$

であり，$n=160$の場合，$S=0.8M$とすると$1455.9M$になる．

3. ベース点のスカラ倍演算

楕円曲線暗号では，通常，楕円曲線が定義される体の情報 ($GF(p)$の場合は素数p)，方程式の情報 ($By^2=x^3+Ax^2+x$のA, B)，及び楕円曲線上のベース点 (G) が，システム全体に共通の値として与られる．ベース点Gのスカラ倍演算は，暗号や署名の処理の中で，必ず用いられる主要な演算であるため，高速な演算が望まれる．ワイヤーシュトラス型楕円曲線においては，ベース点のスカラ倍点演算テーブルを予備計算テーブルとして予め，計算し求めることがよく行われる．

以下にワイヤーシュトラス型楕円曲線における，予備計算テーブルを用いたスカラ倍演算方法を示す．以下の方法は，最も単純な方法である．

予め，2^iG $(i=1,2,\cdots)$を計算しておき，テーブルに保存しているとする．$k=k_{n-1}2^{n-1}+k_{n-2}2^{n-2}+\cdots+k_12+k_0$とするとき，

$$kG=\sum_{i=0}^{n-1}k_i(2^iG)$$

であり，k_i $(i=1,2,\cdots)$は0か1であり，予備計算テーブルより，2^iGが得られるため，最大$n-1$回の楕円曲線の加算でkGを求めることができる．テーブルがなければ，各2^iGを計算する必要があるため，テーブルを持つことにより，$n-1$回の2^iGの計算量を削減できたことになる．ワイヤーシュトラス型楕円曲線では，上記の方法より高速な方法が提案されている [5]．

モンゴメリ型楕円曲線では，点P, Qの加算において$P-Q$の座標が必要であるという性質があるため，上記のようなワイヤーシュトラス型楕円曲線のスカラ倍演算方法を単純に適用できない．そのため，モンゴメリ型楕円曲線では，効率的な予備計算テーブルを用いたスカラ倍演算方法が構成しにくくなっている．

4. 筆者らの従来の方法

筆者らはSCIS2002において，予備計算テーブルを用いたモンゴメリ型楕円曲線のスカラ倍演算方法についての検討結果を発表した [2]．筆者らが検討したアプローチは，全部で2つある．ここでは，それらのアプローチを順に説明する．なお，ここでの方法はすべて，スカラ倍のスカラを順次分解する際に，予備計算テーブル値を利用する頻度が高くなるように分解するものである．スカラ倍計算においては，分解されて小さくなったスカラ倍から順に大きなスカラ倍を計算する．

4.1 アプローチ1

モンゴメリ型楕円曲線スカラ倍演算の従来法では，sG (s:奇数) を$((s-1)/2)G$と$((s-1)/2+1)G$に，常に差がGになるように加算を行うことでスカラ倍演算を実行している．ワイヤーシュトラス型楕円曲線の予備計算テーブルを用いた場合のように$2^iG(i=1,2,\cdots,n-1)$をテーブル化する場合を考えると，従来法では，$((s-1)/2)G$または，$((s-1)/2+1)G$が$2^iG(i=1,2,\cdots,n-1)$のいずれかになる確率が非常に小さいため，テーブルが効率良く使用できない．

そこで，アプローチ1として$s=2^i+u$ $(2^{j-1}\leq u<2^j,\ j<i)$とするとき，$sG$を$(2^{i-1}+u)G$と$2^{i-1}G$と$uG$から求めることを考える (図1参照：図1において最初の“←”は，$(2^i+u)G$を$(2^{i-1}+u)G$と$2^{i-1}G$とその差uGの座標を用いて求めることを意味している)．

この場合，分解された側に必ず$2^{i-1}G$が出現するため，これを予め計算してテーブルにしておくことにより計算の省略が可能となる．さらに，図1のようにuGの結果は，$(2^{i-1}+u)G$の分解の中に存在するので，新たに計算する必要はなく，$(2^{i-1}+u)G$の計算の過程で求めた結果を使用することができる．そのため，$(2^{i-1}+u)G$の座標の計算のみ必要になる．

アプローチ1は，正整数を$2^i+u(2^{j-1}\leq u<2^j,\ j<i)$のように，2のベキ数とそのベキ数未満の正整数の和として考える．これをプラス表現と呼ぶ．

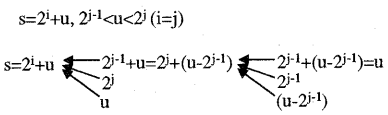

図1 2^i+uの分解例

アプローチ1では，従来法からほとんど高速化されていないという問題がある．以下では，その原因について述べる．図2は，アプローチ1を用いた場合の$87G$の分解例の一部を示している．2^5+23を分解すると，その分解後の値は$2^4+23=2^5+7$となる．2^5+23と2^5+7は同じビット数であり，分解してもビット数が小さくなっていないことがわかる．このため，アプローチ1では分解，すなわち，加算の回数が多くなり，そのため，全体の計算量が大きくなっている．

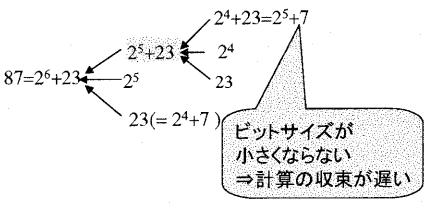

図2 アプローチ1の問題点

4.2 アプローチ 2

前節のアプローチ 1 では，予備計算テーブルを用いてもほとんど高速化されていないことがわかった．これはスカラをより小さなスカラに分解できない場合があるためである．そこでスカラを分解する時に，プラス表現のみならず，マイナス表現の分解も許すことを考え，分解した時に必ず値が小さくなるようにする．以下では，この考え方に基づく予備計算テーブルを使用した方法 (アプローチ 2) について説明する．

4.2.1 アプローチ 2 の概要

アプローチ 2 では，分解後のスカラ値のビット数が，必ず分解前に比べて小さくなるように $(2^i+u)G$ を u の値に応じて，以下の計算により求める (図 3 参照)．

プラス表現 $[(u<2^{i-1})$ **の時]**

$(2^{i-1}+u)G$ と $2^{i-1}G$ の加算 [差は uG] を行う．

マイナス表現 $[(u \geq 2^{i-1})$ **の時]**

$2^i+u=2^{i+1}-v$ として，2^iG と $(2^i-v)G$ の加算 [差は vG] を行う．

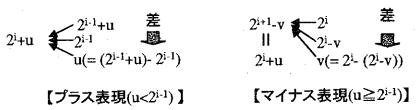

図 3 プラス表現とマイナス表現

アプローチ 2 は，アプローチ 1 の場合と異なり，差の計算を例外的に別途行わなければならない場合がある．それは，マイナス表現からプラス表現に切り替わる箇所で起こる．具体的には，図 4 のように $(2^{i+1}-v)G$(マイナス表現) から $(2^{i-1}+u)G$(プラス表現, $u=2^{i-1}-v$) に切り替わるときに，派生的に計算することになる差 vG を，$2^{i-1}G$ と $-(2^{i-1}-v)G=-uG$ の加算 [差は $(2^{i-1}+2^{i-1}-v)G=(2^{i-1}+u)G$] により求める．

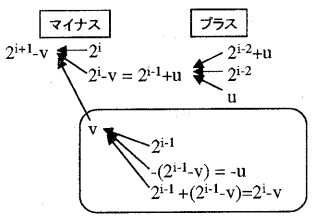

図 4 マイナス→プラス切替時の計算

4.2.2 アプローチ 2 に基づくアルゴリズムの計算量

アプローチ 2 に基づくスカラ倍演算アルゴリズム (従来法 B) については，文献 [2] に詳しく述べているため，ここでは割愛する．

従来法 B の計算量は，

$$(5n-12)M+\left(\frac{5}{2}n-6\right)S$$

である．$n=160$ のとき，$S=0.8M$ より，この計算量は，$1103.2M$ である．モンゴメリ型楕円曲線の従来法 A の計算量は $1455.9M$ であるため，従来法 B は従来法 A の 1.32 倍高速である．一般に，n が大きい場合は，従来法 B は従来法 A の 1.3 倍程度高速である．

5. 提 案 法

5.1 アプローチ 3

アプローチ 2 では，アプローチ 1 の繰返回数を削減することができた．しかし，アプローチ 2 では，差の計算を別途行うような例外的な処理 (マイナスからプラスへの切替) が起こるため，計算回数が増加している．これは，予備計算する点が 2^iG である場合には，避けられないように思われる．そこで，テーブルとして記憶する点を 2^iG から $2^iG+2^{i-1}G$ に変更し，その場合に適した第 3 のアプローチを考案した．

5.1.1 アプローチ 3 の概要

以下では，アプローチ 3 について説明する．アプローチ 3 では，$(2^i+u)G$ を u の値に応じて，以下の計算により求める (図 5 参照)．

プラス表現 2$[(u<2^{i-1})$ **の時]**

$(2^{i-1}+2^{i-2})G$ と $(2^{i-2}+u)G$ の加算 [差は $(2^{i-1}-u)G$] を行う．

マイナス表現 2$[(u \geq 2^{i-1})$ **の時]**

$2^i+u=2^{i+1}-v$ として，$(2^i+2^{i-1})G$ と $(2^{i-1}-v)G$ の加算 [差は $(2^i+v)G$] を行う．

プラス表現 2 では，$2^{i-2}+u$ と $2^{i-1}-u$，マイナス表現 2 では，$2^{i-1}-v$ と 2^i+v の大きい方を次に分解する．大きい方の分解の中に，もう一方 (小さい方) のスカラ倍点が必ず現れるため，小さい方の計算は行う必要がない．

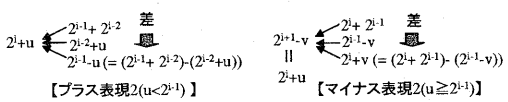

図 5 プラス表現 2 とマイナス表現 2

$87G$ の計算例を図 6 に示す．

この図から，必ず，大きい方の数の分解の中に，小さい方の数が現れることがわかる．アプローチ 2 のときのように，差の計算を別途行うような例外的な処理がないため，その分，高速化される．アプローチ 3 を用いた場合，$87G$ は 4 回の加算により計算できるので，その計算量は，$4ECA_{Z_3'=1}=16M+8S=22.4M$ である．文献 [2] より，従来法 B(アプローチ 2) と比べて，1.7 倍，従来法 A と比べて 2.3 倍程度高速になっている．

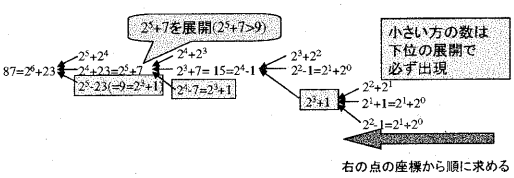

図 6 87G の数値例 (アプローチ 3)

5.1.2 提 案 法

ここでは, アプローチ 3 に基づくスカラ倍演算アルゴリズム (提案法) を示す. アプローチ 3 に基づくアルゴリズムは, 以下のアルゴリズム 3, 4 からなる. 最初にアルゴリズム 3 によって, 計算手順を示す配列 $\{S_i\}$ を求め, それを用いてアルゴリズム 4 によって, スカラ倍点 kG を計算する.

[アルゴリズム 2] **(提案法の計算手順生成アルゴリズム)**

[入力] k: n ビットの整数

[出力] 配列 $\{S_i\}$

Step 1 : $k = s \cdot 2^e$ を満たす最大の e と, s を求める.

Step 2 : $c \leftarrow n-e-1, c_1 \leftarrow 3$

Step 3 : ($k = 2^e$ であるか?)$s = 1$ であるか判定, $s = 1$ の場合, Step 4 へ. それ以外は, Step5 へ.

Step 4 : ($k = 2^e$ の場合)$S_1 \leftarrow 1, S_2 \leftarrow 3, S_3 \leftarrow 2, S_4 \leftarrow 2, \cdots, S_e \leftarrow 2$ として, $\{S_i\}$ を出力し終了.

Step 5 : (kG がテーブルにあるか?) $s = 3$ であるか判定. $s = 3$ の場合, $S_1 \leftarrow e, S_2 \leftarrow 4$ として, $\{S_i\}$ を出力し終了.

Step 6 : s の値により, 以下の処理を行う.

(1) $2^{c-3} < s - 2^c < 2^{c-2}$ の場合,
$s \leftarrow s - 3 \cdot 2^{c-2}, S_{c_1} \leftarrow 1, c \leftarrow c-2$

(2) $2^{c-2} < s - 2^c < 2^{c-1}$ の場合,
$s \leftarrow s - 3 \cdot 2^{c-2}, Sc_1 \leftarrow 1, c \leftarrow c-1$

(3) $s - 2^c < 2^{c-3}$ の場合,
$s \leftarrow 3 \cdot 2^{c-1} - s, S_{c_1} \leftarrow 2, c \leftarrow c-2$

(4) $2^{c-3} < s - 2^c < 2^{c-2}$ の場合,
$s \leftarrow 3 \cdot 2^c - s, S_{c_1} \leftarrow 2$

Step 7 : $c_1 \leftarrow c_1 + 1$.

Step 8 : $s = 5$ または, $s = 9$ であるか判定. $s = 5$ の場合, 次のステップへ. $s = 9$ の場合, Step10 へ. それ以外は, Step6 へ.

Step 9 : $S_1 \leftarrow e, S_2 \leftarrow 1$ として, $\{S_i\}$ を出力し終了.

Step 10 : $S_1 \leftarrow e, S_2 \leftarrow 2$ として, $\{S_i\}$ を出力し終了.

[アルゴリズム 3] **(提案法のスカラ倍演算アルゴリズム)**

[入力] k: n ビットの整数, $\{S_i\}$: 配列

[出力] スカラ倍点 kG

Step 1 : S_2 を判定. $S_2 = 4$ の場合, kG をテーブルから参照して, 出力し終了.

Step 2 : $k_{n-2} = 1$ であるか判定. 成り立つ場合, $c_1 \leftarrow n-e$, それ以外は, $c_1 \leftarrow n-e-1$ とする.

Step 3 : $S_2 = 3$ であるか判定. 成り立つ場合, $U \leftarrow G, V \leftarrow 2G, c_2 \leftarrow 1$ として Step5 へ. それ以外は, Step 4 へ.

Step 4 : $S_2 = 1$ であるか判定. 成り立つ場合, $U \leftarrow 2^{e+1}G, V \leftarrow 2^eG, c_2 \leftarrow 1$, それ以外は, $U \leftarrow 3 \cdot 2^eG, V \leftarrow 3 \cdot 2^eG, c_2 \leftarrow 2, c_1 \leftarrow c_1 - 1$ とする.

Step 5 : $W \leftarrow (2^{c_2} + 2^{c_2-1})G$

Step 6 : $S_{c_1} = 1$ であるか判定. 成り立つ場合, 次のステップへ. それ以外は, Step 8 へ.

Step 7 : $U \leftarrow W + U$(差は V)

Step 8 : $V \leftarrow W + U$ (差は V)

Step 9 : $c_1 = 3$ であるか判定. 成り立つ場合は, Step11 へ.

Step 10 : $c_1 \leftarrow c_1 - 1, c_2 \leftarrow c_2 + 1$ として, Step 5 へ.

Step 11 : $W \leftarrow (2^{c_2+1} + 2^{c_2})G, U \leftarrow W + U$(差は V) として, U を出力し終了.

5.1.3 提案法の計算量

前節で述べた提案法の計算量は, 以下のようになる :

n ビットのスカラ k に対する kG の計算量は,

$$\left(n - \frac{53}{18}\right)(4M + 2S) \approx (5.6n - 16.5)M$$

ただし, M, S はそれぞれ 有限体 $GF(p)$ の乗算, 2 乗算の計算量である. 特に, $n = 160$ のとき, この計算量は $879.5M$ である.

なお計算量の導出については, 付録に述べている.

6. 実 測 結 果

ここでは, モンゴメリ型楕円曲線のスカラ倍計算方法の従来法と提案法のソフトウェア実装結果を示す.

計算機環境は, CPU: PentiumIII 800MHz, メモリ: 128MB, OS: Linux 2.2.17 であり, 多倍長演算ライブラリは GNU MP 4.0 を使用した. 使用した楕円曲線パラメータは以下のとおりである. ここで, $\#E$ は. 楕円曲線の位数, m はベース点の位数を示している. 表中の値はすべて 16 進数で表示している.

[パラメータ 1]

p	ffffffff ffffffff ffffffff ffffff61
A	7a693a13 c9fc98ad 622cb5af 4eea1e31
B	5
$\#E$	1 00000000 00000001 d1c7b863 ae3378f4
G	(d, 201f818a 09592fa4 41ebc2fa 0dc0e549)
m	40000000 00000000 7471ee18 eb8cde3d

[パラメータ 2]

p	ffffffff ffffffff ffffffff ffffffff ffff6415
A	68482a17 1faebe9c 4553eb1d f907c191 4ca92b72
B	2
$\#E$	1 00000000 00000000 00006b06 c1a8c848 df328c5c
G	(6,2c84746e cf5da102 8bde9684 1a81eb43 f2d30a3e)
m	40000000 00000000 00001ac1 b06a3212 37cca317

従来法と提案法の実装結果 (計算時間) を表 1 に示す. この表は, k(パラメータ 1 では 128bits, パラメータ 2 では 160bits) を 1000 回ランダムに発生したときの kG の計算時間を示している. 計算時間の単位は msec である. k のサイズの違いによらず, 提案法 B は従来法 A に比べ 1.6 倍, 従来法 B に比べ 1.25 倍高速になっていることがわかる.

表 1 モンゴメリ型楕円曲線のスカラ倍演算の実測結果

	パラメータ 1	パラメータ 2
(1) 従来法 A	2.30	3.51
(2) 従来法 B	1.74	2.66
(3) 提案法	1.44	2.20
(4) 提案法／ A	0.63	0.63
(5) 提案法／ B	0.83	0.83

7. ワイヤーシュトラス型楕円曲線の方法との比較

ここでは, 提案法とワイヤーシュトラス型楕円曲線で使用される方法と比較した結果を示す. 具体的には, 定義体のサイズが 160 ビットの場合で比較した結果を示す.

5.1.3 節より, 提案法の計算量は, $879.5M$ である. また, 提案法 では, $(2^1+2^0)G=3G$, $(2^2+2^1)G, \cdots, (2^{159}+2^{158})G$ の 159 点をテーブルに記憶する. モンゴメリ型楕円曲線では, Y 座標が必要でなく, また, テーブルに記憶する点は Z 座標を 1 とすることができるため, X 座標をメモリに格納するだけでよい. X 座標は, 160 ビットであるので、提案法 でテーブルとして使用するメモリ量は, $159 \times 160 = 25440$ ビットである.

3. 節で述べたワイヤーシュトラス型楕円曲線の方法についても同様に見積もった. 提案法 も含めた計算量, メモリ使用量 (テーブルによるもののみ) を表 2 に示す. 単純法は, 3. 節で述べた最も単純な方法であり, 最速法は, 文献 [5] でブロックを 5 つとした場合の方法である. ワイヤーシュトラス型楕円曲線では, ヤコビアン座標を用いた.

表 2 より, 提案法は, ワイヤーシュトラス型楕円曲線の単純法とほぼ同等の速度であるが, メモリ使用量が 1/2 に削減できている. しかし, 最速法と比較すると, 提案法は, 1.67 倍低速であり, メモリ使用量も 2.56 倍である. しかし, 本提案法はモンゴメリ型楕円曲線に対する予備計算テーブルを用いたスカラ倍演算として初のアプローチによる方法であり, 今後の展開が期待される.

表 2 提案法とワイヤーシュトラス型楕円曲線の方法の比較

	計算量	メモリ使用量 (bits)
提案法	879.5 M	25440
単純法	832.0 M	50880
最速法	526.4 M	9920

8. ま と め

本稿では, 予備計算テーブルを用いたモンゴメリ型楕円曲線のスカラ倍演算方法を提案した. 楕円曲線の定義体のサイズが大きい場合, 提案法はテーブルを使用しないモンゴメリ型楕円曲線スカラ倍演算の従来法の 1.6 倍, 筆者らの従来法の 1.25 程度高速である. また予備計算テーブルを用いたワイヤーシュトラス型楕円曲線のスカラ倍演算と比べると, 単純な方法との比較では優位性はあるものの, 筆者の知る最速のものとの比較では高速性, メモリ容量の両方で優位性はない. しかし, モンゴメリ型楕円曲線スカラ倍演算を行うための予備計算テーブルを用いて高速化を図る初のアプローチによる方法であることには意義がある.

今後は, 本提案法のアイデアを基にさらなる改良を行うことにより, 最速のワイヤーシュトラス型楕円曲線より高速な計算方法を発見したい.

参 考 文 献

[1] 秋下 徹, "Montgomery 型楕円曲線における高速なスカラー倍同時計算法", 信学技報, ISEC2001-32, pp. 97-103, 2001

[2] 布田 裕一, 大森 基司, "効率的なモンゴメリ型楕円曲線のスカラ倍演算", SCIS2002, 8B-4, pp. , 2002

[3] 伊豆 哲也, "楕円曲線暗号演算の計算法について", SCIS'99, pp. 275-280, 1999

[4] N. Koblitz, "Elliptic curve cryptosystems" , Mathematics of Computation, 48, pp.203-209, 1987

[5] C. Lim and P.Lee, "More flexible exponentiation with precomputation", CRYPTO'94, LNCS 839, Springer-Verlag, pp.95–107, 1994

[6] V.S. Miller, "Use of elliptic curves in cryptography", Advances in Cryptology - Proceedings of CRYPTO'85, LNCS 218, Springer-verlag, pp. 417-426, 1986

[7] H. Cohen, A. Miyaji and T. Ono: "Efficient elliptic curve exponentiation using mixed coordinates " , ASIACRYPT'98, LNCS 1514, Springer-Verlag, pp.51-65, 1998

[8] P.L. Montgomery, "Speeding the Pollar and Elliptic Curve Methods for Factorizations", Math. of Comp. 48, pp. 243-264, 1987

[9] 大岸 聖史, 境 隆一, 笠原 正雄, "y 座標を必要としない楕円型署名の演算法", SCIS'99, pp. 285-287, 1999

[10] 桶屋 勝幸, 櫻井 幸一, "モンゴメリ型楕円曲線上のスカラー倍計算方法の一拡張", SCIS'2001, pp. 305-310, 2001

付 録

付録 A 提案法の計算量導出

ここでは, 5.1.2 節で述べた提案法の計算量を考察し, その導出を行う.

以下に, s は奇数であるとして, 計算量を求めていく. $s = 2^i + u_1$ とする ($u_1 < 2^i$).

[Case1-1(a)] ($2^{i-3} < u_1 < 2^{i-2}$ の時)

$$
\begin{array}{lcll}
 & \swarrow & 2^{i-1}+2^{i-2} & \swarrow \quad 2^{i-2}+2^{i-3} \\
s = 2^i + u_1 & \leftarrow & 2^{i-2}+u_1 (= 2^{i-1}-v_1) & \leftarrow \quad 2^{i-3}-v_1 \\
 & \nwarrow & 2^{i-1}-u_1 (= 2^{i-2}+v_1) & \nwarrow \quad 2^{i-2}+v_1
\end{array}
$$

ただし, $v_1 = 2^{i-2} - u_1$ である. このケースは, アルゴリズム 2 の Step6(1) に相当する. この時, 1 回の加算により, $(i+1)$ ビットデータの $2^i + u_1$ が, $(i-1)$ ビットデータの $2^{i-1} - v_1$ に帰着しているので, データ幅として 2 ビット減っている. ここで, $2^{i-2} + v_1$ は, $2^{i-1} - v_1$ の計算時に計算できるので, ここでは計算しなくてもよいことに注意. また $2^{i-1} - v_1$ は [Case2] で計算される.

[Case1-1(b)] ($2^{i-2} < u_1 < 2^{i-1}$ の時)

$$
\begin{array}{lcll}
 & \swarrow & 2^{i-1}+2^{i-2} & \swarrow \quad 2^{i-2}+2^{i-3} \\
s = 2^i + u_1 & \leftarrow & 2^{i-2}+u_1 (= 2^{i-1}+u_2) & \leftarrow \quad 2^{i-3}+u_2 \\
 & \nwarrow & 2^{i-1}-u_1 (= 2^{i-2}-u_2) & \nwarrow \quad 2^{i-2}-u_2
\end{array}
$$

ただし, $u_2 = u_1 - 2^{i-2}$ である. このケースは, アルゴリズム 2 の Step6(2) に相当する. この時, 1 回の加算により, $(i+1)$ ビットデータの $2^i + u_1$ が, i ビットデータの $2^{i-1} + u_2$ に帰着しているので, データ幅として 1 ビット減っている. ここで, $2^{i-2} - u_2$ は, $2^{i-1} + u_2$ の計算時に計算できるので, ここでは計算しなくてもよいことに注意. また $2^{i-1} + u_2$ は [Case1] で

計算される.

[Case1-2] ($u_1 < 2^{i-3}$ の時)

$$
\begin{array}{lll}
 & \swarrow & 2^{i-1}+2^{i-2} \\
s=2^i+u_1 & \leftarrow & 2^{i-2}+u_1 \quad \swarrow \quad 2^{i-2}+2^{i-3} \\
 & \nwarrow & 2^{i-1}-u_1 \quad \leftarrow \quad 2^{i-3}-u_1 \\
 & & \nwarrow \quad 2^{i-2}+u_1
\end{array}
$$

このケースは, アルゴリズム 2 の Step6(3) に相当する. この時, 1 回の加算により, $(i+1)$ ビットデータの 2^i+u_1 が, $(i-1)$ ビットデータの $2^{i-1}-u_1$ に帰着しているので, データ幅として 2 ビット減っている. ここで, $2^{i-2}+u_1$ は, $2^{i-1}-u_1$ の計算時に計算できるので, ここでは計算しなくてもよいことに注意. また $2^{i-1}-u_1$ は [Case2] で計算される.

[Case2] ($u_1 > 2^{i-1}$ の時)

$$
\begin{array}{lll}
 & \swarrow & 2^i+2^{i-1} \\
s=2^i+u_1(=2^{i+1}-v_1) & \leftarrow & 2^{i-1}-v_1 \quad \swarrow \quad 2^{i-1}+2^{i-2} \\
 & \nwarrow & 2^i+v_1 \quad \leftarrow \quad 2^{i-2}+v_1 \\
 & & \nwarrow \quad 2^{i-1}-v_1
\end{array}
$$

ただし, $v_1=2^i-u_1$ である. このケースは, アルゴリズム 2 の Step6(4) に相当する. この時, 1 回の加算により, $(i+1)$ ビットデータの 2^i+u_1 が, $(i+1)$ ビットデータの 2^i+v_1 に帰着しているので, データ幅として減っていない. ここで, $2^{i-1}-v_1$ は, 2^i+v_1 の計算時に計算できるので, ここでは計算しなくてもよいことに注意. また 2^i+v_1 は [Case1] で計算される.

以上のような遷移で状態が移動するので, 各 Case の発生確率を見ていく. 初期状態において, 入力 s がどの Case に該当するかをビット表現にして見ていく:

[Case1-1(a)]	$1001**\cdots$
[Case1-1(b)]	$101***\cdots$
[Case1-2]	$1000**\cdots$
[Case2]	$11*****\cdots$

上記より, 発生確率は以下のようになる:

[Case1-1(a)]	$\frac{1}{2}\times\frac{1}{2}\times\frac{1}{2}=\frac{1}{8}$	(Case1 の $\frac{1}{4}$)
[Case1-1(b)]	$\frac{1}{2}\times\frac{1}{2}=\frac{1}{4}$	(Case1 の $\frac{1}{2}$)
[Case1-2]	$\frac{1}{2}\times\frac{1}{2}\times\frac{1}{2}=\frac{1}{8}$	(Case1 の $\frac{1}{4}$)
[Case2]	$\frac{1}{2}$	

ここで, 各 Case の計算量を求めるために, 記号を以下のように定義する:

$T_1(n)$	:	Case1 の場合の 楕円曲線加算回数
$T_{11a}(n)$	:	Case1-1(a) の場合の 楕円曲線加算回数
$T_{11b}(n)$	:	Case1-1(b) の場合の 楕円曲線加算回数
$T_{12}(n)$	:	Case1-2 の場合の 楕円曲線加算回数
$T_2(n)$	:	Case2 の場合の 楕円曲線加算回数

ただし, 上記楕円曲線加算回数は, n ビットデータに対するものである.

先に見てきたことから,

[Case1-1(a)]	$T_{11a}(n)=1+T_2(n-2)$
[Case1-1(b)]	$T_{11b}(n)=1+T_1(n-1)$
[Case1-2]	$T_{12}(n)=1+T_2(n-2)$
[Case2]	$T_2(n)=1+T_1(n)$

であり, また $T_1(n)=\frac{1}{4}T_{11a}(n)+\frac{1}{2}T_{11b}(n)+\frac{1}{4}T_{12}(n)$ であるので,

$$
\begin{aligned}
T_1(n) &= \frac{1}{4}\{1+T_2(n-2)\}+\frac{1}{2}\{1+T_1(n-1)\} \\
&\quad +\frac{1}{4}\{1+T_2(n-2)\} \\
&= \cdots = \frac{1}{2}T_1(n-1)+\frac{1}{2}T_1(n-2)+\frac{3}{2}
\end{aligned}
$$

となる. これより, $T_1(n)$ の一般項を求めると,

$$
\begin{aligned}
T_1(n) &= T_1(4)+\sum_{k=5}^{n}\left\{1-\frac{2}{3}\left(-\frac{1}{2}\right)^{k-2}\right\}=\cdots \\
&= n-\frac{22}{9}+\frac{1}{9}\left(-\frac{1}{2}\right)^{n-3}
\end{aligned}
$$

従って, s : 奇数, n ビットの場合の楕円曲線加算回数 $T_{\mathrm{odd}}(n)$ は,

$$
\begin{aligned}
T_{\mathrm{odd}}(n) &= \frac{1}{2}T_1(n)+\frac{1}{2}T_2(n)=\cdots \\
&= n-\frac{35}{8}+\frac{1}{9}\left(-\frac{1}{2}\right)^{n-3}
\end{aligned}
$$

また偶数の場合も含めた k : n ビットに対する, kG の楕円曲線加算回数 $T(n)$ は,

$$
\begin{aligned}
T(n) &= \frac{1}{2}T_{\mathrm{odd}}(n)+\left(\frac{1}{2}\right)^2 T_{\mathrm{odd}}(n-1)+\cdots \\
&= \sum_{k=0}^{n-4}\left(\frac{1}{2}\right)^{k+1} T_{\mathrm{odd}}(n-k)=\cdots \\
&= n-\frac{53}{18}-\left(\frac{1}{2}\right)^{n-2}\left\{-3n+\frac{87}{18}+\frac{1}{18}(-1)^{n-3}\right\} \\
&\approx n-\frac{53}{18}
\end{aligned}
$$

となる.

楕円曲線加算の計算量は, $ECA_{Z_3'+1}=4M+2S$ であるので, $S=0.8M$ とおくと, kG の計算量は,

$$
\left(n-\frac{53}{18}\right)(4M+2S)\approx(5.6n-16.5)M
$$

$n=160$ のとき, この計算量は $879.5M$ である.