アルゴリズム 78－6
（2001．5．18）

楕円曲線上のスカラー倍点計算法の改良

安達 大亮　　平田 富夫

名古屋大学大学院工学研究科
〒464-8603 名古屋市千種区不老町
Tel: 052-789-3440,　Fax: 052-789-3089
E-mail: {adachi,hirata}@hirata.nuee.nagoya-u.ac.jp

概 要

楕円曲線上のスカラー倍点計算は，楕円曲線暗号方式において鍵生成や暗号化・復号化処理で使用される重要な計算である．この計算について加算連鎖を用いた方法や巾乗計算に基づいたWindow Method と呼ばれる方法など効率の良い計算法が提案されている．本論文では，Window Method の改良版を提案する．このアルゴリズムでは完全重み付き有向グラフ上での spanning arborescence を用い，より効率良くスカラー倍点を求める．また既存の計算法との性能比較を行う．

Improved Scalar Exponentiation Method on Elliptic Curve

Daisuke Adachi　　Tomio Hirata

School of Engineering, Nagoya University
Furou, Chikusa, Nagoya, Aichi, 464-8603, Japan
Tel: 052-789-3440,　Fax: 052-789-3089
E-mail: {adachi,hirata}@hirata.nuee.nagoya-u.ac.jp

Abstract

In the Elliptic Curve Cryptosystem(ECC), scalar exponentiation plays an important role in the process of key generation, encryption, and decryption. So, various algorithms have been proposed such as Window Method and a method using Addition-Chain. In this paper, we improve Window Method algorithm. Using weighted spanning arborescence, our algorithm executes scalar exponentiation efficiently. We also present experimental results that show the superiority of our algorithm to the existing ones.

1　はじめに

近年，市民生活においても計算機ネットワークを介して情報を暗号化して送ることが増えている．このような暗号通信では，受信者は不特定多数からメッセージを受け取ることが多い．このような利用形態に適しているのが公開鍵暗号である．有名な公開鍵暗号としては RSA 暗号，ElGamal 暗号が挙げられる．ElGamal 暗号を代表とする離散対数問題に基づく暗号方式を楕円離散対数問題を用いて再構成したものが楕円曲線暗号方式と呼ばれるものである．

楕円曲線暗号方式の特徴として，同等の安全性を確保する為の鍵の長さが RSA 暗号など従来の公開鍵暗号方式の数分の一で済むことがある．通常，鍵は数値データまたは数値データの組として

与えられ, 鍵以外のパラメータの bit 長は鍵の長さとほぼ同じである. 従って暗号処理に必要な計算量が, RSA 暗号など従来の公開鍵暗号方式と比較して少なくなる. このため, 楕円曲線暗号方式は, 実装サイズと製作コストの制約から演算装置を低性能なものにせざるをえない携帯端末や IC カード等への実装により適している.

楕円曲線暗号方式の各種処理 (公開鍵の生成, 暗号化, および復号化) では, 加算および 2 倍算という 2 種類の演算を基本演算とする. しかし, 両演算共に 1 回あたり, 長 bit 整数どうしの乗算だけでも 10 数回を必要とするため, 処理全体の計算量は依然大きい. そのため, これらの処理に関して様々な効率化手法が考えられている. 本論文では, 公開鍵の生成, 暗号化, および復号化処理の中で使用されるスカラー倍点計算に注目し, 根付き有向グラフの一種である spanning arborescence を用いた計算法を提案する. そして, 既存法との計算時間および演算回数に関する比較を行う.

2 背景

affine 座標系の場合, 楕円曲線は $y^2 = x^3 + ax + b$ $(4a^3 + 27b^2 \neq 0)$ という式で表現される. このような曲線上の任意の 2 点 P, Q に関して, $P \oplus Q$ もまた曲線上の点である二項演算 $\oplus$ が定義される [5]. $P \neq Q$ の場合を加算 (addition), $P = Q$ の場合を 2 倍算 (doubling) と呼び, 総称して点と点の加法と呼ぶ. affine 座標系における P, Q と $P \oplus Q$, $2P$ の幾何学的な関係を図 1 に示す.

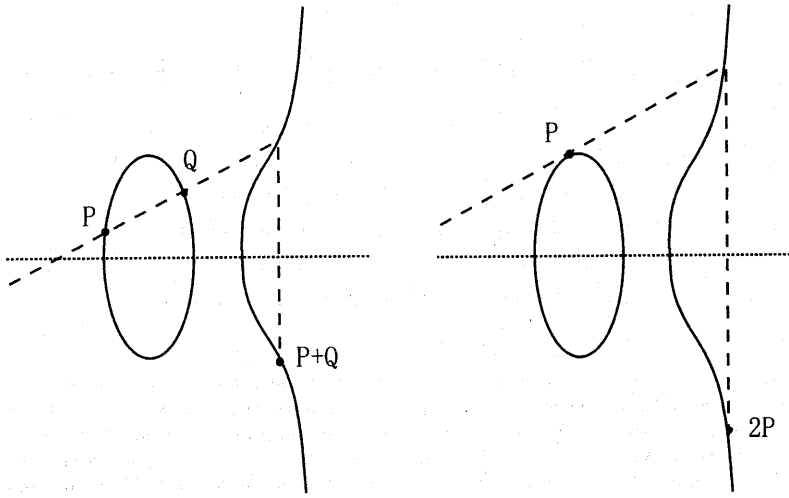

図 1: "点と点の加法" の幾何学的概念

2.1 スカラー倍点

M は楕円曲線 E 上の任意の 1 点, e は自然数とする. このような e, M に対して, 点 eM は以下の式

$$eM = \overbrace{M \oplus M \oplus \cdots \oplus M}^{e} \tag{1}$$

で表される E 上の点である. 即ち eM は, M を $\oplus$ に関して e 個加え合わせた点である. このような自然数倍の点を総称して M のスカラー倍点と呼ぶ. スカラー倍点計算は, 与えられた e, M に対しスカラー倍点 eM を求める計算である.

楕円曲線暗号では自然数 e が 150bit を超える数であるため, (1) 式のように M を逐次的に 1 個ずつ加える計算方法は現実的ではない. 既存のスカラー倍点計算法としては, 大別して加算連鎖 (addition-chain) 問題に基づく手法と巾乗計算に類似した手法の 2 つが存在する.

2.2 加算連鎖問題に基づく手法

数列 $a_0(=1), a_1, a_2, \ldots, a_r(=e)$ の各要素 a_i $(i = 1, 2, \ldots, r)$ が以下の関係式

$$a_i = a_j + a_k \quad k \leq j < i$$

を満たすような数列を, e に対する長さ $r+1$ の加算連鎖と呼ぶ [3]. この数列に従って, スカラー倍点 eM を r 回の $\oplus$ 演算で求めることができる. 当然, r が小さい程 eM を計算する効率が良い. しかし, 与えられた e に対して長さ最短の加算連鎖を求める問題は NP 困難である. そのため, [6] の Extended Window Method 等のような, 最短ではないが実時間で構成可能な加算連鎖を用いたアルゴリズムが数々提案されている.

2.3 巾乗計算アルゴリズムに類似した手法

楕円曲線暗号では有限体上で定義された楕円曲線が使用される [5]. この楕円曲線上の点集合は $\oplus$ を基本演算とする群を構成し, これは乗法群と同型である. 従って $\oplus$ を乗算と置き換えることで, 反復二乗法に類似したスカラー倍点計算アルゴリズムが得られる. これは Binary Method と呼ばれる (Algorithm 1). ここで, $\mathcal{O}$ は $\oplus$ の零元に相当する点, n は e の bit 数である. また, 自然数 e の 2 進表現の下位から i bit 目の値を $e[i]$ で表す.

Algorithm 1 : Binary Method
set $Y = \mathcal{O}$, $n = \lfloor \log_2 e \rfloor + 1$
For $i = n$ **Downto** 1 **by** -1
  $Y \leftarrow 2Y$ [**2 倍算**]
  If $e[i] = 1$ **Then**
    $Y \leftarrow Y \oplus M$
Return Y

Algorithm 1 より, Binary Method は $O(n) = O(\log_2 e)$ 時間のアルゴリズムである.

Binary Method を改良した計算法として Window Method がある. Window Method では, まず入力 e を 2 進数化し w bit 長のセグメントに区切る. 簡単のため n は w の倍数であるとする. セグメントの数を $n' = n/w$ とし, 下位から i 番目のセグメントの値を e_i で表す.

Algorithm 2 : Window Method
set $Y = \mathcal{O}$, $n' = n/w$
For $i = n'$ **Downto** 1 **by** -1
  $Y \leftarrow 2^w Y$
  If $e_i \neq 0$ **Then**
    If $e_i M$ is already computed **Then** Read $e_i M$.
    Else Compute $e_i M$.
  $Y \leftarrow Y \oplus e_i M$
Return Y

セグメント長 w bit で表現可能なセグメントは 2^w 種類であるが, e の 2 進表現をセグメントに

分割したときに生成されるセグメント数が 2^w を上回ると, 同一の値 e_i を持つセグメントが複数個生じる. 従って上のアルゴリズムの 6 行目で e_iM の計算結果を記録しておけば, 以降はその記録を参照するだけで良い. e の bit 長にも関連するが, 適切な w の値を設定するならば Window Method は Binary Method より効率が良くなる. 例えば, e が 150〜160bit の場合 w は 3 または 4 と設定するのが普通である. なお、実際の Window Method では、常にセグメントの末尾の bit が 1 となるような方法など、より巧妙なセグメント分割手法が用いられている.

3 中間結果の計算方法

通常 w が比較的小さいことから, Window Method における各中間結果 e_iM の計算には Binary Method を用いる. 例えば $e_i = 3$, $e_j = 7$ という 2 種類のセグメント $e_i, e_j (i < j)$ が設定されている場合, Binary Method を用いて

$$\begin{aligned} e_jM &= 7M = 2(\underline{2M \oplus M}) \oplus M \\ e_iM &= 3M = \underline{2M \oplus M} \end{aligned}$$

と $7M$, $3M$ の順に計算する. しかし, 上の 2 式では, 中間結果としての $3M$ は $7M$ の計算中にも出現している. 従って先に $3M$ を求めて記録しておき, $7M = 2 \cdot 3M \oplus M$ と計算すれば, $7M$ の計算に必要な演算回数が Binary Method よりも減少する. そこで, $e_{n'}M, \ldots, e_1M$ を順に求めるのではなく, うまい順番で計算することで上のように演算回数を削減することを考える.

3.1 計算のコスト

e_iM の値を用いて e_jM を計算するときのコストを次のように定義する.

楕円曲線上では点 X の逆元 $-X$ の計算が容易であることから, ここでは, $\oplus X$, $2 \cdot X$ に $\oplus(-X)$ を加えた 3 演算を使用する (X は楕円曲線上の任意の点). 計算は e_i, e_j の大小関係で大きく 2 つに場合分けされる. ここで y は $2^y e_i \le e_j < 2^{y+1} e_i$ を満たす非負整数とする.

$$\begin{aligned} (e_i > e_j \text{ のとき})\ e_jM &= e_iM \oplus \{-(e_i - e_j)M\} && (2) \\ (e_i < e_j \text{ のとき})\ e_jM &= 2^y e_iM \oplus (e_j - 2^y e_i)M \\ &= 2^y e_iM \oplus r_1M && (3) \\ e_jM &= 2^{y+1} e_iM \oplus (e_j - 2^{y+1} e_i)M \\ &= 2^{y+1} e_iM \oplus (-r_2M) && (4) \end{aligned}$$

上の計算式に従って e_iM から e_jM を計算する際の演算回数を表 1 に示す.

表 1: e_iM をもとに e_jM を求める計算コスト

条件	加算	2 倍算
$e_i > e_j$	$e_i - e_j$	0
$e_i < e_j$ ((3) 式)	$\nu(r_1)$	$\lambda(\lfloor r_1/2^y \rfloor) + y$
$e_i < e_j$ ((4) 式)	$\nu(r_2)$	$\lambda(\lfloor r_2/2^{y+1} \rfloor) + y + 1$
Binary Method	$\nu(e_j) - 1$	$\lambda(e_j)$

ここで, $\nu(x)$ は x の 2 進表現中に現れる 1 の数を表し, $\lambda(x) = \lfloor \log_2 x \rfloor$ とする (但し $\lambda(0) = 0$).$e_i < e_j$ の場合, (3) 式と (4) 式の 2 つの計算法が存在する. 個々の e_i, e_j の場合について, 加算および 2 倍算の回数比較を行なって効率の良い方の計算式を用いる. また, 楕円曲線上では, 通常、加算の方が 2 倍算よりも演算のコストが大きいため, このことを考慮に入れる必要がある. このようにして e_iM から e_jM を計算するときのコストを定める. このコストを $w(e_i, e_j)$ と表す. なお $e_i = 1$ とすると, $r_1 + r_2 = 2^y$ より $\lfloor r_1/2^y \rfloor \in \{0, 1\}$, $\lambda(1) = \lambda(0) = 0$ となり, Binary Method を用いた場合の演算回数と等しくなる.

4 スカラー倍点計算アルゴリズム

必要な中間結果 $\{e_{n'}M, e_2M, \ldots, e_1M\}$ が全て計算されているならば, w 回の 2 倍算と e_iM の加算を交互に実行することで eM が計算できる. そこで, まず $\{e_{n'}M, e_2M, \ldots, e_1M\}$ を計算し, これらの値を参照しながら eM を計算するアルゴリズムを提案する.

入力 e, M が与えられると, 提案法は以下の方法でスカラー倍点 eM を計算する.

1. e の 2 進表現を w bit 長のセグメントに分割する
2. セグメントの値で 0 以外のものを要素とする集合をあらためて $V = \{e_1, e_2, \ldots, e_N\}$ とする
3. $e_0 = 1$ とし, $V \cup \{e_0\}$ が節点集合で、各有向辺 (e_i, e_j) に重み $w(e_i, e_j)$ が付いた完全有向連結グラフ G を考え、G 上に存在する、節点 e_0 を根とする総重み最小の spanning arborescence を求める
4. 得られた有向木に従って、M から出発し全ての中間結果 e_iM を求める $(e_i \in V)$
5. 求めた e_iM を適時参照してスカラー倍点 eM を求める

Step 3 の $w(e_i, e_j)$ は 3 章の方法で求めた計算コストである.

Spanning arborescence 全ての中間結果 $\{e_1M, e_2M, \ldots, e_NM\}$ を 1 つも計算していない状態では, 入力として与えられた M だけが既知である.

ここで, aM をもとに bM を計算する作業は, 節点 a から有向辺 (a, b) を通って節点 b を訪れることに対応する. すると, M から出発して $e_1M, e_2M, \ldots, e_NM$ を求めることは, G において節点 e_0 から他の全節点 $e_i \in V$ を 1 度だけ訪れることに相当する. このような経路 T は (1) 節点 '1' を唯一の根とし, (2) 閉路を含まず, (3) 根からその他の全節点 $e_i \in V$ への有向経路がただ 1 本ずつ存在する. このような T を spanning arborescence と呼ぶ.

G の各有向辺 (e_i, e_j) の重みとして前節の $w(e_i, e_j)$ が与えられているので, spanning arborescence T の辺の重みの総和は $\{e_1M, e_2M, \ldots, e_NM\}$ の計算に必要な総演算回数に相当する. 当然, 辺の重みの総和が最小な spanning arborescence が, 最も効率良く $\{e_1M, e_2M, \ldots, e_NM\}$ を計算する方法を表している.

Fulkerson のアルゴリズム 辺の重みの総和が最小な spanning arborescence を求める Fulkerson のアルゴリズムを以下に示す [4].

G を重み付き有向連結グラフ, r を G の根節点とし, r を指す有向辺は存在しないものとする. G の各有向辺 $e_{ij} = (i, j)$ に対して, 重み w_{ij} が付与されているものとする.

1. 各節点 $t \in V - \{r\}$ について, t を指す有向辺の中から最小重みを持つ辺 e_{ut} を求める $(u \in V - \{t\})$. 次に t を指す有向辺全てに, w_{ut} だけ減らした重み $w'_{xt} = w_{xt} - w_{ut}$ を新たに割り当てる

2. 選択した辺による G の部分グラフに含まれる, 空でない連結成分の個数を p とする

3. $p > 1$ であれば, 根節点 r を含まない連結成分を縮約し新しい有向グラフ G' の節点とする (これにより, G' 上で, 節点 x から節点 y への有向辺が複数本存在する場合がある. この場合その中で最小の重みを持つ有向辺で全ての辺を置き換える). 新たなグラフ G', 重み w' に対して再び 1. から処理を開始するが, 1. の処理は G' で新たに出現した節点についてのみ行なう.

4. $p = 1$ であれば, 今までに選択した有向辺で G の部分有向グラフが構成されている. 根からの深さ優先探索を行ない、すでに訪れた節点を指す有向辺を削除する.

Theorem 1 Fulkerson のアルゴリズムは, 有向連結グラフ G 上に r を根とする spanning arborescence が存在するならば, 重みの総和が最小なものを出力する.

Proof: 文献 [2] 参照. □

5 数値実験

提案法と Binary Method, Window Method の各方式を計算機上に実装し, スカラー倍点計算処理に関する実験を行なった. 各方式の実装には, 射影座標系の 1 つである Jacobian 座標系を用いた. この座標系の利点として加算および 2 倍算の計算式で除算を必要とせず, 従って affine 座標系の場合と比べて計算量が小さくなることがある [1].

5.1 実行演算回数

ある e, M に対するスカラー倍点 eM の計算で実行される加算 (逆元の加算含む), 2 倍算の回数を測定した (表 2). ここで e は 155bit の自然数である. また, Window Method のセグメント長は 3, 提案法のセグメント長は 4 とする.

表 2: e, M に対する実行演算回数

計算方式	総回数	演算量	中間結果の計算	その他	n	N
Binary Method	[77, 154]	333.1	—	[77, 154]	—	—
Window Method	[43, 159]	257.9	[4, 5]	[39, 154]	40	4
提案法	[37, 161]	246.1	[7, 7]	[30, 154]	31	8

ここで, $[x, y]$ で x 回の加算と y 回の 2 倍算を表すものとする. また演算量は, 加算と 2 倍算の計算量の比 c を考慮に入れ, 総演算回数 $[X, Y]$ に対し $c \cdot X + Y$ として算出したものである. Jacobian 座標系の場合, 実験で使用した楕円曲線では $c = 16/7 \approx 2.3$ となる [1]. n は設定されたセグメント数, N は セグメントの種類の数を表す.

提案法では, Window Method と比較して総演算回数の内 2 倍算が少々増加する代わり, 加算の回数が減少している. 通常, 計算量に関して加算は 2 倍算よりも大きく, 特に Jacobian 座標系ではその比 c が大きくなっている [1]. 従って, 加算と 2 倍算だけに注目した場合, 提案法は Window Method よりも効率が良い.

5.2 実行時間

上記の各手法について, 異なる e, M の組 100 個に対するスカラー倍点 eM の総計算時間を計測した (表 3). ここで, e は 155bit のランダムな値, M は楕円曲線上の点をランダムに選択したものである.

表 3: スカラー倍点計算 1 回あたりの平均実行時間

計算方式	実行時間 (秒)	(%)	(%)
Binary Method	1.3191	100	123.6
Window Method	1.0664	80.8	100
提案法	1.0536	79.8	98.7

表 3 における Window Method, 提案法のセグメント長は 5.1 節と同条件とする. 提案法は Binary Method の 80 %, および Window Method の 99 %の時間で 100 個のスカラー倍点 eM を計算している.

5.3 提案法に関する考察

提案法は, 表 2 の演算量に関して約 4 %の改善がみられる. しかし表 3 の計算時間では目立った改善結果が得られていない. その理由として, 中間結果 $e_1M, e_2M, \ldots, e_NM$ の計算処理は全体の処理の一部であり, 中間結果の計算を効率化して得られる改善効果も限られたものになっているからである.

本実験で用いた実装では, Window Method の中間結果の計算順序を求めるために提案法を適用している. 今後, 実装面についてより詳細な検討を進めることで, 演算量の削減と同等の改善を得ることができると考えている.

6 まとめ

総重み最小な spanning arborescence に基づいたスカラー倍点計算法を提案した. 既存の各計算法との比較を行なった結果, Window Method に対しては演算回数の面では約 4 %の改善が見られ, 演算回数の削減に効果があることが確認された.

参考文献

[1] Atsuko Miyaji, Takatoshi Ono, Henri Cohen, "Efficient elliptic curve exponentiation." *Advances in Cryptology-Proceedings of ICICS '97.*

[2] Jack Edmonds, "Optimum Branchings." Journal of Research of the National Bureau of Standards Vol.71B, No.4, pp233–240, 1967.

[3] D.E.Knuth, "Seminumerical Algorithm." The Art of Computer Programming vol.2, Addison Wesley, 1981.(和訳:D.E.Knuth 著, 中川圭介訳, "準数値算法／算術演算", The art of Computer Programming 第 4 分冊, サイエンス社, 1986.)

[4] D.R.Fulkerson, "Packing rooted directed cuts in a weighted directed graph." Mathematical Programming 6 (1974).

[5] "暗号・ゼロ知識証明・数論." 情報処理学会 監修, 岡本龍明・太田和夫 共編, 共立出版, 1995.

[6] 国廣昇, 山本博資, "Restricted addition-substaction chain への情報理論的アプローチについて." 1996 年暗号と情報セキュリティシンポジウム SCIS96 講演論文集 SCIS96-3B, 1996.