$GF(2^m)$ 上の並列モンゴメリ乗算を用いた楕円曲線演算法

小池正修[†,††] 松本勉[†]

本論文では，多項式剰余演算系（PRNS）表現を利用した $GF(2^m)$ 上の並列モンゴメリ乗算アルゴリズムを提案する．PRNS 表現は特にハードウェアでの並列処理による高速演算に適した多項式の表現方法であるが，効率的な除算のアルゴリズムが知られていないため，剰余乗算に適用した例はほとんどなかった．本論文では剰余演算系（RNS）表現でのモンゴメリ乗算アルゴリズムを応用して，$GF(2^m)$ 上の PRNS モンゴメリ乗算アルゴリズムを構成した．モンゴメリ乗算は実効的な除算を含まないため，PRNS 表現での剰余乗算の高速並列処理が可能となる．さらに提案アルゴリズムを $GF(2^m)$ 上の楕円曲線演算に適用した場合の点のスカラー倍算アルゴリズムの検討を行い，処理性能の評価を行った．

Arithmetic of Elliptic Curve over $GF(2^m)$ Using Parallel Montgomery Multiplication

Masanobu Koike[†,††] and Tsutomu Matsumoto[†]

This paper presents a new method of parallel modular multiplication in $GF(2^m)$ using polynomial residue number systems (PRNS). Though PRNS is suitable for a fast and parallel computation, there are few algorithms to compute modular multiplication in PRNS representation since division is not efficiently implemented. We derive the proposed algorithm by analogy with Montgomery multiplication based on residue number systems. We apply this method to arithmetic of elliptic curve over $GF(2^m)$ and provide point scalar multiplication method in PRNS representation.

1. はじめに

公開鍵暗号の実装においては，多倍長整数演算および $GF(2)$ 上の多項式演算の処理性能が全体の処理時間を左右する．そのためこれらの処理の高速化に関する研究が多くなされている．多倍長整数ではこの 1 つに剰余演算系（Residue Number Systems，以下 RNS と略す）を利用する方法がある．RNS 表現では，基底と呼ばれる互いに素な複数の数を用意し，それらを法とした剰余演算を基本演算として処理を行う．RNS 表現は，加減乗算で各基底要素の演算を独立に実行可能なため，特にハードウェアでの並列実装に適した手法である．しかし，RNS 表現では除算の効率的手法が知られていないため，RNS 表現を利用するアルゴリズムは除算を含まないことが望ましい．Posch らは文献 9) でモンゴメリ乗算[8]を応用した RNS 表現での剰余乗算手法を提案している．モンゴメリ乗算は実効的な除算を含まないため，RNS 表現により高速並列処理が可能である．

一方 $GF(2)$ 上の多項式演算の高速化手法では，整数における RNS 表現の類似概念として，多項式剰余演算系（Polynomial Residue Number Systems，以下 PRNS と略す）を利用する方法がある．PRNS 表現も RNS 表現と同様に，互いに素な複数の多項式を基底とした法演算を基本演算として処理を行う．PRNS 表現も加減乗算の並列処理が可能だが，除算の効率的手法は知られていない．

Halbutoğullari らは文献 3) で PRNS 表現を利用した並列剰余乗算アルゴリズムを提案している．このアルゴリズムでは法多項式の倍数テーブルを利用することで PRNS 表現での実効的な除算を回避している．しかし倍数テーブルを用意したことにより，扱える法多項式が限定されてしまうという問題がある．さらにこの倍数テーブルが巨大であるということ，また倍数テーブルを参照するために PRNS 表現での演算中に基数表現を用いる場面があり，PRNS 表現の長所を生

† 横浜国立大学大学院環境情報学府/研究院
Graduate School of Environment and Information Sciences, Yokohama National University
†† 株式会社東芝 SI 技術開発センター
Systems Integration Technology Center, Toshiba Corporation

かしきれていないことも問題にあげられる．

本論文では，PRNS 表現での並列剰余乗算手法として，モンゴメリ乗算を応用したアルゴリズムを提案する☆．これは RNS モンゴメリ乗算アルゴリズムを PRNS 表現に応用したものであり，特にハードウェアでの並列処理に向いたアルゴリズムである．提案手法では，巨大な倍数テーブルや基数表現の利用を必要とせず，法多項式も任意に選べるという利点があり，Halbutoğullari らの手法の問題点を解決している．

また，提案アルゴリズムの $GF(2^m)$ 上の楕円曲線暗号への応用として，PRNS 表現での点のスカラー倍算アルゴリズムを検討し，処理性能の評価を行った．

2. 準　　備

2.1 RNS 表現

RNS 表現は整数 a を表現する一手法として知られている．この表現は互いに素な複数の整数からなる基底

$$f = \{f_1, f_2, \cdots, f_n\},$$
$$\gcd(f_i, f_j) = 1 \quad (\forall i \neq j)$$

により，a の各基底要素による剰余の組

$$< a >_f = (a_1, a_2, \cdots, a_n)$$

で a を表す方法である．ここで n は基底 f の要素数，$a_i = a \bmod f_i$ である．剰余の組 $< a >_f$ を基底 f の下での a の RNS 表現と呼ぶ．

RNS 表現された整数は，加減乗算を基底要素ごとに独立に計算できるという特長を持つ．したがって基底要素ごとの演算を並列化することで処理の高速化が期待できる．しかし RNS 表現においては，除算および 2 数の大小比較の効率的アルゴリズムが知られていない．そのため，RNS 表現を用いて高速化を図るアルゴリズムには，除算と大小比較が含まれていないことが望ましい．

2.2 PRNS 表現

PRNS 表現は多項式 $a(x)$ を表現する一手法で，整数での RNS 表現の類似概念である．PRNS 表現は互いに素な複数の多項式からなる基底

$$f = \{f_1(x), f_2(x), \cdots, f_n(x)\},$$
$$\gcd(f_i(x), f_j(x)) = 1 \quad (\forall i \neq j)$$

により，$a(x)$ の各基底要素による剰余の組

$$< a(x) >_f = (a_1(x), a_2(x), \cdots, a_n(x))$$

で $a(x)$ を表す方法である．ここで n は基底 f の要素数，$a_i(x) = a(x) \bmod f_i(x)$ である．剰余の組 $< a(x) >_f$ を基底 f の下での $a(x)$ の PRNS 表現と呼ぶ．

PRNS 表現では，基底要素のすべての積 $F(x) = \prod f_i(x)$ の次数 $\deg(F(x))$ より次数の小さな多項式を一意的に表現することができる．特に $GF(2)$ 上の m 次拡大体 $GF(2^m)$ は，$GF(2)$ 上の m 次既約多項式 $N(x)$ を法とした多項式の集合と見なせるため，$\deg(N(x)) < \deg(F(x))$ であれば基底 f による PRNS 表現で $GF(2^m)$ の元を表現することができる．したがって $GF(2^m)$ の演算も PRNS 表現を利用して並列化することで，処理の高速化が期待できる．ただし RNS 表現同様，除算および 2 多項式の次数比較の効率的アルゴリズムが知られていないため，PRNS 表現でのアルゴリズムには除算と次数比較が含まれていないことが望ましい．

2.3 記　　号

本節では，以下で利用する記号をまとめる．

まず $GF(2^m)$ を $GF(2)$ の m 次拡大体，$N(x)$ を $GF(2^m)$ を定義する m 次既約多項式とする．

次に，PRNS 表現では各基底要素を L 次多項式とし，$n \geq 2$ を基底の要素数とする．PRNS 表現された多項式の各成分の値は $(L-1)$ 次多項式（L ビット）となる．ここで n と L は，$GF(2^m)$ の元を表現できるように $nL > m$ と選ぶ．また，PRNS 基底 f のすべての基底要素の積を大文字の $F(x) = \prod f_i(x)$ で表し，$F_i(x) = F(x)/f_i(x)$ とする．

一方，通常の多項式での表現（基数表現と呼ぶことにする）では 1 ワードを $(L-1)$ 次多項式で表現することにする．すなわち基数として x^L を用いる．また n は $N(x)$ のワード数を指すことにする．したがって n は $nL \geq m$ を満たす最小の整数となる．このとき $GF(2^m)$ の元 $a(x)$ の基数表現は次のように書ける．

$$a(x) = \sum_{j=0}^{n-1} a_{(j)}(x) x^{jL} \tag{1}$$

また基数表現のベクトル表記を，係数を降順に並べた

$$a(x) = (a_{(n-1)}, a_{(n-2)}, \cdots, a_{(0)})_{(L)} \tag{2}$$

で表すことにする．ここで式 (1) の $a_{(j)}(x)$ は x^{jL} の係数である $(L-1)$ 次多項式を表し，式 (2) の $a_{(j)}$ は $a_{(j)}(x)$ の係数を抜き出して並べ L ビットの数値と見なしたものである．

2.4 処理性能評価の前提

本論文では PRNS 表現でのアルゴリズムを，L ビットの演算器を n 個並べた並列演算器に実装することを想定している．ハードウェア実装での処理性能評価には，処理速度と回路規模の観点から評価する必要が

☆ 本論文の内容については，部分的に 2002 年コンピュータセキュリティシンポジウム（CSS2002）において発表している[6]．

Input: $< a(x) >_{f\cup g}, < b(x) >_{f\cup g}$
Output: $< c(x) >_{f\cup g}$ $(c(x) = a(x)b(x) \bmod N(x))$

1: $< c(x) >_{f\cup g} \leftarrow < a(x) >_{f\cup g} < b(x) >_{f\cup g}$
2: $c(x) \leftarrow < c(x) >_{f\cup g}$
3: for $h = 2n-1$ downto n
4: for $i = L/r$ downto 1
5: $< c(x) >_{f\cup g} \leftarrow < c(x) >_{f\cup g} + < T'[c_{(h-1),i}] >_{f\cup g} < x^{(h-n-1)L+(i-1)r} >_{f\cup g}$
6: $c(x) \leftarrow c(x) + (c_{(h-1),i}(x)x^{nL} + T[c_{(h-1),i}])x^{(h-n-1)L+(i-1)r}$
7: return $< c(x) >_{f\cup g}$

図 1 PRNS 剰余乗算アルゴリズム
Fig. 1 PRNS modular multiplication algorithm.

あるが，これは利用する半導体製造技術やライブラリに依存する．そのため，ここではアルゴリズム自体の処理性能を評価するという見地から，PRNS 表現でのアルゴリズムの処理性能を，想定する並列演算器での処理ステップ数と事前計算値を格納するのに必要なメモリ量（以下，ROM 容量と呼ぶ）の 2 つの観点で行うことにする．

処理ステップ数の評価のため，“サイクル” という単位を導入する．並列演算器での並列処理 1 回を 1 サイクルとし，アルゴリズムの処理ステップ数をサイクル数で評価する．

3. PRNS 剰余乗算

Halbutoğullari らは PRNS 表現を利用した並列演算法として，文献 3) で**図 1** に示すような PRNS 剰余乗算アルゴリズムを提案した．ここで f, g はともに要素数 n の PRNS 基底であり，$a(x)b(x)$ を表現するために 2 つの基底を用いている．

Halbutoğullari らの PRNS 表現における剰余算実現のアイディアは $N(x)$ の倍数テーブルを利用することである．以下このアイディアを，倍数テーブルの作成，および実際の剰余算の 2 段階に分けて説明する．

まず，$N(x)$ の倍数テーブル T および T' を構成するために，すべての $(r-1)$ 次以下の多項式 $U(x)$ に対し，$V(x) = U(x)N(x)$ を計算する（ここで $0 < r \leq L$. 本論文では簡単のため r は L の約数とする☆）．式 (2) のように $V(x)$ をベクトル表記すると

$$V(x) = (V_{(n)}, V_{(n-1)}, \cdots, V_{(0)})_{(L)}$$

$V_{(n)}(x)$ は $(r-1)$ 次以下の多項式

となる．このとき基数表現での倍数テーブル T の $V_{(n)}$ 番目の要素 $T[V_{(n)}]$ を，$V(x)$ の下位 n ワードとして

$$T[V_{(n)}] = (V_{(n-1)}, V_{(n-2)}, \cdots, V_{(0)})_{(L)}$$

とする．図 1 のステップ 5 では PRNS 表現での倍数テーブル T' も必要である．こちらは $V_{(n)}$ も含めて

$$< T'[V_{(n)}] >_{f\cup g} = < V(x) >_{f\cup g}$$

とする．倍数テーブルは事前計算，あるいは演算時の前処理として計算される．これらは演算処理量とテーブルサイズのトレードオフで決まるが，本論文では事前計算により演算器内の ROM に格納されているものとする．

次に，h ワード（$h \geq n$）の多項式 $c(x)$ が与えられたとき法 $N(x)$ での剰余をテーブル参照により r ビット単位で求める方法を**図 2** に示す．図 2 の①は $c(x)$ の初期状態

$$c(x) = (c_{(h-1)}, c_{(h-2)}, \cdots, c_{(0)})_{(L)}$$

を示している．まず最上位ワード $c_{(h-1)}(x)$ を r ビットごとの L/r 個のブロックに分割する（図 2②）：

$$c_{(h-1)}(x) = \sum_{j=1}^{L/r} c_{(h-1),j}(x)x^{(j-1)r}.$$

次に $c_{(h-1)}(x)$ の高次側からブロックを走査して 0 でない最初のブロックを $c_{(h-1),i}$ とする．このブロックの値から倍数テーブル $T[c_{(h-1),i}]$ を参照し，

$$(c_{(h-1),i}(x)x^{nL} + T[c_{(h-1),i}])x^{(h-n-1)L+(i-1)r}$$

を $c(x)$ に加えたものを改めて $c(x)$ とおくと，倍数テーブルの構成方法から，$\bmod N(x)$ で $c(x)$ と合同で，元の $c(x)$ より次数を r 次小さくすることができる（図 2③）．これを次数が m より小さくなるまでくり返すことで $N(x)$ での除算を加算と乗算のみで実現できる．

図 1 のアルゴリズムの処理性能は，以下のように見積もることができる．まずサイクル数を考えるが，想定する並列演算器に関わるステップ 1，2，5 のみを見積りの対象とする．ステップ 6 の演算量をカウントしていないため，実際の演算量はさらに多くなることに

☆ Halbutoğullari らは $r = L$ の場合のみ扱っている．

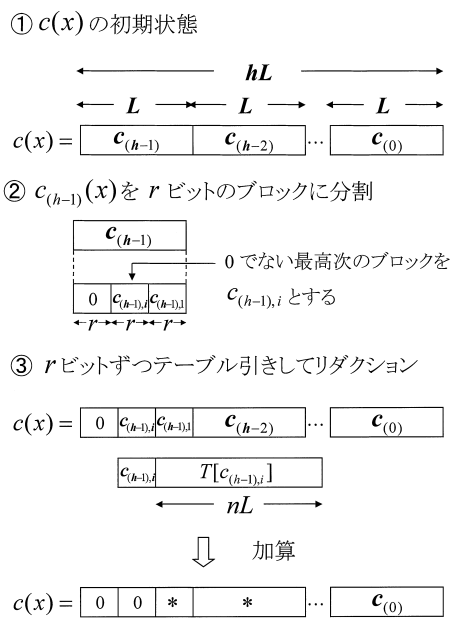

図 2 テーブル参照による剰余算
Fig. 2 Table lookup reduction.

表 1 PRNS 剰余乗算に必要なサイクル数
Table 1 Number of cycles for a PRNS modular multiplication.

ステップ	サイクル数 剰余乗算	加算
1	2	0
2	$2n+1$	$2n-1$
5	$2nL/r$	$2nL/r$
計	$2(1+L/r)n+3$	$2(1+L/r)n-1$

注意しておく．

表 1 は必要なサイクル数をまとめたものである．表 1 のステップ番号は図 1 のステップ番号に対応している．サイクル数の欄は，演算器での剰余乗算，加算の回数をカウントした値である．ステップ 2 のサイクル数については，5.2 節を参照されたい．ここで 5.2 節では 1 つの基底 g からの基数表現への変換を示しているが，図 1 のステップ 2 では 2 つの基底 f，g からの変換であることに注意されたい．

次に ROM 容量を見積もる．図 1 のステップ 2 では 5.2 節より $(2nL+4n^2L)/8$ バイトが必要である．また，ステップ 5 では倍数テーブル T' に $2^{r+1}nL/8$ バイト，$< x^{(h-n-1)L+(i-1)r} >_{f\cup g}$ $(n \le h \le 2n-1, 1 \le i \le L/r)$ に $2n^2L^2/8r$ バイト必要である．ステップ 6 での倍数テーブル T にも $2^r nL/8$ バイト必要なので，これらを総合すると，$nL(2nL/r+4n+3\cdot 2^r+2)/8$ バイトとなる．

Halbutoğullari らの手法では，以下のような問題点があげられる．まず，倍数テーブルを用意したことにより，法多項式の変更に対して柔軟性に欠けるということである．法多項式を変更した場合，並列演算器内の ROM に格納されている倍数テーブルを作り直す必要がある．

次に，事前計算値のメモリ量は，想定されるパラメータとして $m=160$，$L=32$，$n=6$ とすると，$r=8$ のとき約 20 K バイト，$r=16$ のとき約 4.5 M バイトと巨大になることである．PRNS 剰余乗算を実現する回路規模に大きな影響を与えるため，実用上は $r=8$ 程度が限界と考えられる．

さらに，図 1 のアルゴリズムでは，倍数テーブル参照のために $c(x)$ の基数表現が必要である．そのためステップ 2 で $c(x)$ の基数表現を求め，ステップ 6 で PRNS 表現と同じ処理を基数表現でも行っており，PRNS 表現を利用する意味が薄れている．

4. PRNS モンゴメリ乗算

本章では，PRNS 表現で剰余乗算を行う新しい手法として，モンゴメリ乗算を応用したアルゴリズムを提案する．まずその原型となる RNS 表現でのモンゴメリ乗算アルゴリズムについて簡単に説明する．

4.1 整数上でのモンゴメリ乗算

モンゴメリ乗算は入力 $a, b < N$ に対して

$$w = \frac{ab + (ab(-N)^{-1} \bmod R)N}{R} \qquad (3)$$

を計算するアルゴリズムである[8]．ここで R はモンゴメリ定数と呼ばれ，通常 $R > N$ なる 2 のべきを用いる．この場合，出力は $w < 2N$ までしか保証されないため，$w > N$ の場合は N を減じて N 未満にする処理（final subtraction と呼ぶ）が必要である．

4.2 RNS モンゴメリ乗算

RNS 表現でのモンゴメリ乗算アルゴリズムは文献 9) で提案されており，**図 3** のステップとなる．

RNS 表現でモンゴメリ乗算を行うために，2 つの RNS 基底 $f=\{f_1, f_2, \cdots, f_n\}$, $g=\{g_1, g_2, \cdots, g_n\}$ を用意する．基底 g の要素の積 G はモンゴメリ乗算における定数 R に相当する．図 3 は，基底 f，g で表現された a，b を入力とし，同じく 2 つの基底で表現された w を出力するアルゴリズムで，

$$w = abG^{-1} \bmod N$$

を計算する．

図 3 のステップ 3，7 は基底拡張と呼ばれる操作で，基底 f，g 相互での基底変換である．基底拡張は原理的には中国剰余定理を利用して 2 進数表現を経由して行う．たとえばステップ 3 では

Input: $< a >_{f\cup g}, < b >_{f\cup g}$
Output: $< w >_{f\cup g}$ $(w \equiv abG^{-1} \bmod N, w < 2N)$

	基底 f での演算	基底 g での演算
1:	$< s >_f \leftarrow < ab >_f$	$< s >_g \leftarrow < ab >_g$
2:	—	$< t >_g \leftarrow < s(-N^{-1}) >_g$
3:	$< t >_f \Longleftarrow < t >_g$	
4:	$< u >_f \leftarrow < tN >_f$	—
5:	$< v >_f \leftarrow < s + u >_f$	—
6:	$< w >_f \leftarrow < vG^{-1} >_f$	—
7:	$< w >_f \Longrightarrow < w >_g$	

図 3 RNS モンゴメリ乗算アルゴリズム
Fig. 3 RNS Montgomery multiplication algorithm.

Input: $< a(x) >_{f\cup g}, < b(x) >_{f\cup g}$
Output: $< w(x) >_{f\cup g}$ $(w(x) \equiv a(x)b(x)G(x)^{-1} \bmod N(x), \deg(w(x)) < \deg(N(x)))$

	基底 f での演算	基底 g での演算
1:	$< s(x) >_f \leftarrow < a(x)b(x) >_f$	$< s(x) >_g \leftarrow < a(x)b(x) >_g$
2:	—	$< t(x) >_g \leftarrow < s(x)N(x)^{-1} >_g$
3:	$< t(x) >_f \Longleftarrow < t(x) >_g$	
4:	$< u(x) >_f \leftarrow < t(x)N(x) >_f$	—
5:	$< v(x) >_f \leftarrow < s(x) + u(x) >_f$	—
6:	$< w(x) >_f \leftarrow < v(x)G(x)^{-1} >_f$	—
7:	$< w(x) >_f \Longrightarrow < w(x) >_g$	

図 4 PRNS モンゴメリ乗算アルゴリズム（MM）
Fig. 4 PRNS Montgomery multiplication algorithm (MM).

$$t = \sum_{i=1}^{n} (t_i G_i^{-1} \bmod g_i) G_i - kG$$

より求めた t を $\bmod f_i$ することで行う．ここで $G_i = G/g_i$ であり，k は $\bmod G$ のための補正項である．この演算は基底要素サイズの演算のみで，明示的に 2 進数表現を求めずに実行できる．基底拡張を効率的に行うアルゴリズムの詳細は文献 4)，7) を参照されたい．

4.3 多項式上でのモンゴメリ乗算

Koç らは整数上のモンゴメリ乗算を $GF(2)$ 上の多項式へ拡張した[5]．計算式は整数の場合の式 (3) と同様に

$$w(x) = \frac{a(x)b(x) + t(x)N(x)}{R(x)}$$

である．ここで

$$t(x) = a(x)b(x)N(x)^{-1} \bmod R(x)$$

である．多項式上でのモンゴメリ定数は $R(x)$ であり，$\deg(R(x)) > \deg(N(x))$ と選ぶと $\deg(w(x)) < \deg(N(x))$ となるため，整数版での final subtraction に対応する演算は不要である．

4.4 PRNS モンゴメリ乗算

以下，本論文で提案する PRNS 表現でのモンゴメリ乗算手法を述べる．

まず 2 つの PRNS 基底 $f = \{f_1(x), \cdots, f_n(x)\}$，$g = \{g_1(x), \cdots, g_n(x)\}$ を用意する．基底 g の要素の積 $G(x)$ はモンゴメリ定数 $R(x)$ に相当する．

図 4 は基底 f，g で表現された $a(x)$，$b(x)$ を入力とし，同じく 2 つの基底で表現された

$$w(x) = a(x)b(x)G(x)^{-1} \bmod N(x)$$

を出力するアルゴリズムである．

各ステップは図 3 のアルゴリズムと対応しているが，基底拡張に違いがある．たとえばステップ 3 では

$$t(x) = \sum_{i=1}^{n} (t_i(x) G_i(x)^{-1} \bmod g_i(x)) G_i(x)$$

により求めた $t(x)$ を $\bmod f_i(x)$ することで行うが，右辺の次数は $\deg(G(x))$ 以下なので，$\bmod G(x)$ の処理が不要となる．具体的には次の 2 ステップで行われる．

(1) $< \xi(x) >_g = < t(x) >_g < G_i(x)^{-1} >_g$

表 2 PRNS 剰余乗算と PRNS モンゴメリ乗算のサイクル数比較
Table 2 Comparison between number of cycles for a PRNS modular multiplication and a PRNS Montgomery multiplication.

アルゴリズム	サイクル数 剰余乗算	加算
PRNS 剰余乗算	$2(1+L/r)n+3$	$2(1+L/r)n-1$
PRNS モンゴメリ乗算	$2n+7$	$2n-1$

(2) for $i=1$ to n

$< t(x) >_f = < t(x) >_f + \xi_i(x) < G_i(x) >_f$

ここで $< G_i(x)^{-1} >_g$ は基底 g の下で表現された $(G_1(x)^{-1}, G_2(x)^{-1}, \cdots, G_n(x)^{-1})$ を表す．

基底拡張に必要な演算量は，剰余乗算が $(n+1)$ サイクル，加算が $(n-1)$ サイクルである．必要な事前計算値は，$< G_i(x)^{-1} >_g$ および $< G_i(x) >_f$ $(1 \le i \le n)$ の計 $(nL+n^2L)/8$ バイトである．

図 4 のアルゴリズムの処理性能は以下のように見積もることができる．まず PRNS モンゴメリ乗算 1 回に必要なサイクル数をカウントする．図 4 より，剰余乗算，加算が，基底拡張以外のステップでそれぞれ 5 サイクル，1 サイクル，2 回の基底拡張でそれぞれ $2(n+1)$ サイクル，$2(n-1)$ サイクル必要である．したがって全体で剰余乗算 $(2n+7)$ サイクル，加算 $(2n-1)$ サイクルが必要となる．

次に ROM 容量を見積もる．2 回の基底拡張で $2(nL+n^2L)/8$ バイト，ステップ 6 の $< G(x)^{-1} >_f$ で $nL/8$ バイトの計 $nL(2n+3)/8$ バイトである．

本提案アルゴリズムを Halbutoğullari らの手法と比較する．まず，法多項式の倍数テーブルが不要である．そのため一度作成した並列演算器に変更を加えることなしに任意の m 次既約多項式を法多項式として利用できるという利点がある．

次に演算量で比較する．**表 2** に Halbutoğullari らの手法と提案アルゴリズムに必要なサイクル数をまとめる．表 2 より加算のサイクル数は PRNS モンゴメリ乗算のほうが小さい．剰余乗算については PRNS 剰余乗算でのサイクル数が，PRNS モンゴメリ乗算でのサイクル数より小さくなる条件は

$$nL < 2r \tag{4}$$

である．3 章で述べたように，$r \le L$ であるので $nL < 2r \le 2L$，すなわち $n < 2$ となる．しかし PRNS 表現では $n \ge 2$ であるため，式 (4) は成立しない．したがって PRNS モンゴメリ乗算でのサイクル数のほうが PRNS 剰余乗算でのサイクル数より小さい，すなわち，より高速であると結論づけてよい．実際，$m=160$，$L=32$，$n=6$，$r=8$ とすると，PRNS 剰余乗算のサイクル数は剰余乗算 63，加算 59，PRNS モンゴメリ乗算のサイクル数は剰余乗算 19，加算 11 であるため，PRNS モンゴメリ乗算のほうが 3 倍以上高速である．

最後に ROM 容量で比較を行う．PRNS 剰余乗算の ROM 容量に対する PRNS モンゴメリ乗算の ROM 容量の比は

$$\frac{2n+3}{2nL/r+4n+3\cdot 2^r+2}$$

である．たとえば $m=160$，$L=32$，$n=6$，$r=8$ とすると，この値は約 1/56 となる．したがって，ROM 容量においても，提案手法のほうが Halbutoğullari らの手法より優れているといえる．

5. 楕円曲線演算への応用

本章では，4.4 節で提案した PRNS モンゴメリ乗算の $GF(2^m)$ 上の楕円曲線暗号への応用を検討する．楕円曲線暗号の多くの方式では，点のスカラー倍算を利用しているが，点のスカラー倍算は暗号処理時間の大部分を占めるため，高速化が求められている．以下，PRNS モンゴメリ乗算を点のスカラー倍算に適用した場合を考察する．なお，スカラー倍算のアルゴリズムはバイナリ法を想定する．

5.1 PRNS 表現による点のスカラー倍算

ここでは従来の点のスカラー倍算のインタフェースとの互換性から，入出力データは基数表現かつアフィン座標表示とする．この上で PRNS モンゴメリ乗算を利用するためには，各データの PRNS 表現への変換，モンゴメリ演算系への変換，およびこれら 2 つの逆変換が必要である．

またアフィン座標表示での点の加算，2 倍算には，$GF(2^m)$ での逆元計算が必要である．PRNS 表現では効率的に逆元計算が行えないため，点の加算，2 倍算において逆元計算の不要な mixed 座標表示を採用した[1)]．

以上をまとめると PRNS 表現でのモンゴメリ乗算を利用した楕円曲線上の点 $P=(P_X(x), P_Y(x))$ のスカラー倍算アルゴリズムは**図 5** のようになる．ステップ 1 は，各データを PRNS 表現に変換するための定数の計算である．PRNS 表現では効率的な除算アルゴ

Input: $P = (P_X(x), P_Y(x)), N(x), d$
Output: $Q = (Q_X(x), Q_Y(x)) = dP$

1: $G_N(x)^2 \leftarrow G(x)^2 \bmod N(x)$
2: $G_N(x)^2, N(x), P_X(x), P_Y(x)$ を PRNS 表現に変換
3: $< N(x)^{-1} >_g$ を計算
4: $< P >_{f \cup g}$ を射影座標表示 $< P_P >_{f \cup g}$ に変換
5: $< P_{PM} >_{f \cup g} \leftarrow MM(< P_P >_{f \cup g}, < G_N(x)^2 >_{f \cup g})$
6: $< Q_{PM} >_{f \cup g} \leftarrow d < P_{PM} >_{f \cup g}$
7: $< Q_P >_{f \cup g} \leftarrow MM(< Q_{PM} >_{f \cup g}, < 1 >_{f \cup g})$
8: $< Q_P >_g$ を基数表現 Q_P に変換
9: Q_P をアフィン座標表示 Q に変換

図 5 PRNS 表現を利用した楕円曲線スカラー倍算
Fig. 5 Scalar multiplication algorithm in PRNS representation.

リズムがないため，この計算には基数表現を用いる．ステップ 2 は，各データの PRNS 表現への変換である．ステップ 3 では PRNS モンゴメリ乗算で必要な $N(x)$ の逆元を計算する．この計算は，既約剰余類群 $(GF(2)[x]/(g_i(x)))^\times$ の exponent を e_i としたとき，各基底 $g_i(x)$ ごとに $e_i - 1$ 乗することで実現する．ここで exponent とは，$(GF(2)[x]/(g_i(x)))^\times$ の任意の元の位数の最小公倍数を指し，事前計算されているものとする．ステップ 4 は，点 P の射影座標 P_P への変換である．この変換は Z 座標に 1 を付けるのみである．記号 $< P >_{f \cup g}$ 等は点の各座標を基底 $f \cup g$ の下で PRNS 表現した $< P_X(x) >_{f \cup g}$，$< P_Y(x) >_{f \cup g}$，$< P_Z(x) >_{f \cup g}$ の 3 つ組を表している．ステップ 5 は，図 4 の PRNS モンゴメリ乗算アルゴリズム MM を用いて，P_P の X 座標，Y 座標，Z 座標をそれぞれ $< G_N(x)^2 >_{f \cup g}$ と PRNS モンゴメリ乗算を行い，モンゴメリ演算系に変換された点 P_{MP} を求めるステップである．ステップ 6 はスカラー倍算で，PRNS モンゴメリ乗算を $GF(2^m)$ 上の剰余乗算として用いる．ステップ 7 は，スカラー倍算で得られた点 Q_{PM} をモンゴメリ演算系から通常の演算系の点 Q_P に戻す操作である．射影座標表示された Q_P をアフィン座標表示に戻すためには $GF(2^m)$ での除算が必要なため，ステップ 8 で基数表現に変換した上でアフィン座標表示への変換（ステップ 9）を行うことで，最終的な出力 Q を得る．

5.2 PRNS 表現と基数表現との変換

本節では，図 5 のステップ 2 と 8 で必要となる，PRNS 表現と基数表現との変換手法を述べる．

基数表現された $a(x) = \sum_{j=0}^{n-1} a_{(j)}(x) x^{jL}$ を PRNS 表現に変換する式は

$$< a(x) >_f = \sum_{j=0}^{n-1} < a_{(j)}(x) >_f < x^{jL} >_f$$

である．この式の演算量は剰余乗算 n サイクル，加算 $(n-1)$ サイクルであり，必要な事前計算値は $< x^{jL} >_f$ $(0 \le j \le n-1)$ の $n^2 L/8$ バイトである．

逆に基底 g の下で PRNS 表現された $< a(x) >_g = (a_1(x), a_2(x), \cdots, a_n(x))$ を基数表現に変換する式は

$$a(x) = \sum_{i=1}^{n} (a_i(x) G_i(x)^{-1} \bmod g_i(x)) G_i(x)$$

である．上の式で $\xi_i(x) = a_i(x) G_i(x)^{-1} \bmod g_i(x)$，$G_i(x) = \sum_{j=0}^{n-1} G_{i(j)}(x) x^{jL}$ とすると

$$a(x) = (1, x^L, x^{2L}, \cdots, x^{(n-1)L}) \times \sum_{i=1}^{n} \xi_i(x) \begin{pmatrix} G_{i(0)}(x) \\ G_{i(1)}(x) \\ \vdots \\ G_{i(n-1)}(x) \end{pmatrix}$$

なので剰余乗算 $(n+1)$ サイクル，加算 $(n-1)$ サイクルで計算できる．必要な事前計算値は，$< G_i(x)^{-1} >_g$ および $< G_{i(j)}(x) >_f$ $(0 \le j \le n-1)$ の計 $(nL + n^2 L)/8$ バイトである．

5.3 点のスカラー倍算の性能見積り

Mixed 座標表示での点の加算，2 倍算公式[1]では，$GF(2^m)$ での演算が，点の加算において乗算が 11 回，2 乗算が 4 回，点の 2 倍算において乗算が 5 回，2 乗算が 5 回必要である．これらの $GF(2^m)$ での乗算を図 4 で示すモンゴメリ乗算 MM で行うと，スカラー d が m ビットのときスカラー倍算全体で，PRNS モンゴメ

表 3 PRNS 表現を利用したスカラー倍算に必要なサイクル数
Table 3 Number of cycles for a scalar multiplication in PRNS representation.

	サイクル数	
ステップ	剰余乗算	加算
2	$7n$	$7(n-1)$
3	$3L/2$	0
4	0	0
5	$3(2n+7)$	$3(2n-1)$
6	$35m(2n+7)/2$	$35m(2n-1)/2$
7	$3(2n+7)$	$3(2n-1)$
8	$3(n+1)$	$3(n-1)$

リ乗算が $(m/2)(11+4)+m(5+5)=35m/2$ 回必要なので，並列演算器での剰余乗算が $35m(2n+7)/2$ サイクル，加算が $35m(2n-1)/2$ サイクル必要となる．

その他のステップでのサイクル数もまとめると，**表 3** となる．ここでは基数表現での演算が必要なステップ 1 と 9 は対象外とし，ステップ 2 から 8 までをあげる．ステップ 2 では $N(x)$ は基底 f での表現に，その他の 3 個の値は基底 $f \cup g$ での表現に変換すること，またステップ 5，7，8 は各座標について行うのでそれぞれ前節まで述べたサイクル数の 3 倍必要であることに注意しておく．表 3 よりスカラー倍算で必要な総サイクル数は，剰余乗算が $(35m+22)n+3L/2+245m/2+45$，加算が $(35m+22)n-35m/2-13$ となる．

表 4 に，いくつかの m の値に対してスカラー倍算 1 回の処理時間見積りをあげる．これらの処理時間は剰余乗算 1 サイクルの処理時間によって変わってくるため，ここではその時間を 10〜100 ns と仮定した．また加算は排他的論理和 1 回で実現できるため，より処理量の多い剰余乗算だけで見積もることにする．なお，SECG（Standards for Efficient Cryptography Group）では楕円曲線暗号の安全性の理由から $m = 163, 193, \cdots$ の利用を定めているが[2]，これらの処理時間はそれぞれ $m = 160, 192, \cdots$ の処理時間に，スカラー値のビット数の増分だけ追加された PRNS モンゴメリ乗算の回数分増加する．したがってサイクル数の増加分は，スカラー値のビット数の増加分を Δm とすると，剰余乗算 $35\Delta mn + 245\Delta m/2$，加算 $35\Delta mn - 35\Delta m/2$ である．たとえば $m = 163$ では $m = 160$ の場合のサイクル数に剰余乗算が 997.5 サイクル，加算が 577.5 サイクル追加される．

次に必要な ROM 容量を考察する．基数表現から PRNS 表現への変換では 2 つの基底 f，g への変換が必要なので，5.2 節の 2 倍の $2n^2L/8$ バイト，PRNS モンゴメリ乗算では $nL(2n+3)/8$ バイト，PRNS 表現から基数表現への変換では，$< G_i(x)^{-1} >_g$ が PRNS モンゴメリ乗算と共有できるため $n^2L/8$ バイトである．したがって計 $nL(5n+3)/8$ バイトとなる．たとえば，$m = 160$，$L = 32$，$n = 6$ のとき 792 バイト，$m = 256$，$L = 32$，$n = 9$ のとき 1,728 バイトである．

表 4 $L = 32$ のときの処理時間見積り
Table 4 Computation time estimation when $L = 32$.

			剰余乗算 1 サイクルの時間		
m	n	cycles	10 ns	50 ns	100 ns
160	6	53,425	0.53 ms	2.67 ms	5.34 ms
192	7	70,807	0.71 ms	3.54 ms	7.08 ms
224	8	90,429	0.90 ms	4.52 ms	9.04 ms
256	9	112,291	1.12 ms	5.61 ms	11.2 ms

表 4 および ROM 容量の数値を評価するために，現在処理速度が最高レベルと思われる文献 10) の結果を参照する．文献 10) は，基数表現におけるモンゴメリ乗算の実装最適化，およびスカラーを冗長 2 進数表現することで高速処理を実現する方式である．文献 10) では，0.11 μm CMOS スタンダードセル・ライブラリによる ASIC 実装の性能評価を行っており，スカラー倍算 1 回の処理時間は，演算器を 32 ビットとしたとき，$m = 160$ の場合は 0.33 ms，$m = 256$ の場合は 1.17 ms である．なおこの実装では，32 ビット演算器での乗算が 10 ns 以上の速度で処理されていると考えられる．また，メモリ容量は $m = 160$ のとき 320 バイト，$m = 256$ のとき 512 バイトである．

よって提案した PRNS モンゴメリ乗算を用いたスカラー倍算の処理速度は，演算器を 32 ビット，剰余乗算 1 サイクルが 10 ns と仮定すると，文献 10) の手法と比べ同程度であるといえる．また，ROM 容量も十分実用的な範囲であるといえる．なお，提案手法は並列処理を行っているため，m が大きくなるほど高速化の効果が大きくなることが見込まれる．そのため将来，楕円曲線暗号の鍵長 m が大きくなった場合，提案手法は有望な高速計算手法となる．

ここで提案手法の性能評価は，処理速度は図 5 のステップ 1 と 9 を除いた剰余乗算のサイクル数のみで，またメモリ量は事前計算値の容量だけで行った荒い見積りであるのに対し，文献 10) では実装結果の評価であり，評価尺度が異なっていることに注意しておく．また，提案手法は複数の演算器を用いているので，より正確な評価のためには，提案手法の具体的な回路構成を行い，速度と回路規模両方の観点から考慮する必要があるが，これは今後の課題である．ただし提案手法が高速計算手法として有効であることは確認できたと考える．

6. むすび

本論文では PRNS 表現におけるモンゴメリ乗算アルゴリズムを提案した．これは RNS モンゴメリ乗算アルゴリズムを $GF(2)$ 上の多項式に応用したものである．従来の PRNS 剰余乗算アルゴリズム[3]の問題点であった，巨大な倍数テーブルや基数表現の利用を必要とせず，法多項式も任意に選べるという利点がある．さらに処理速度，メモリ量ともに PRNS 剰余乗算アルゴリズムより優れている．

また，PRNS モンゴメリ乗算を用いた楕円曲線上の点のスカラー倍算のアルゴリズムの検討も行い，処理性能の見積りを行った．その結果，文献 10) の方式と同程度の性能が得られる見込みを得た．また，提案手法は並列処理を行っているため，法多項式の次数 m が大きくなるほど高速化の効果が大きくなることが見込まれる．

今後の課題としては，具体的な回路構成を行い，提案手法の有効性をより詳細に評価することがあげられる．

参考文献

1) Blake, I., Seroussi, G. and Smart, N.: *Elliptic Curves in Cryptography*, LMS, Vol.265, Cambridge University Press (1999).
2) Certicom Research: *Standards for Efficient Cryptography Group* (*SECG*) (2000).
3) Halbutoğullari, A. and Koç, Ç.K.: Parallel Multiplication in $GF(2^k)$ Using Polynomial Residue Arithmetic, *Designs, Codes and Cryptography*, Vol.20, No.2, pp.155–173 (2000).
4) Kawamura, S., Koike, M., Sano, F. and Shimbo, A.: Cox-Rower Architecture for Fast Parallel Montgomery Multiplication, *Advances in Cryptology — EUROCRYPT 2000*, Preneel, B.(Ed.), IACR, pp.523–538, Springer-Verlag, (2000).
5) Koç, Ç.K. and Acar, T.: Montgomery Multiplication in $GF(2^k)$, *Designs, Codes and Cryptography*, Vol.14, No.1, pp.57–69 (1998).
6) 小池正修，松本　勉：並列モンゴメリ乗算による $GF(2^m)$ 上の楕円曲線演算法，コンピュータセキュリティシンポジウム（CSS），pp.413–418, 情報処理学会 (2002).
7) 小池正修，佐野文彦，川村信一：Cox-Rower アーキテクチャによる高速な並列モンゴメリ乗算法，暗号と情報セキュリティシンポジウム（SCIS），電子情報通信学会 (2000).
8) Montgomery, P.L.: Modular Multiplication without Trial Division, *Mathematics of Computation*, Vol.44, No.170, pp.519–521 (1985).
9) Posch, K.C. and Posch, R.: Modulo Reduction in Residue Number Systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.6, No.5, pp.449–454 (1995).
10) 佐藤　証，高野光司：$GF(2^n)$ 上のモンゴメリ乗算を用いた楕円演算コプロセッサ，暗号と情報セキュリティシンポジウム（SCIS），pp.1113–1118, 電子情報通信学会 (2002).

(平成 14 年 11 月 29 日受付)
(平成 15 年 6 月 3 日採録)

小池　正修（正会員）

1974 年生．1996 年東京大学理学部数学科卒業．1998 年東京大学大学院数理科学研究科修士課程修了．同年株式会社東芝入社．以来，暗号と情報セキュリティの研究開発に従事．2002 年より横浜国立大学大学院環境情報学府博士課程後期に在籍．

松本　勉（正会員）

1986 年東京大学大学院博士課程修了，工学博士．同年横浜国立大学工学部専任講師．同助教授，教授を経て，2001 年より同大学大学院環境情報研究院教授．1981 年より暗号や情報セキュリティの研究に従事．「明るい暗号研究会」創設メンバー．現在，暗号アルゴリズム，情報利用管理，ディジタル証拠性，情報ハイディング，バイオメトリクス，人工物メトリクス，耐タンパーソフトウェア等に広く関心を持つ．国際暗号学会 IACR 理事．暗号技術検討会構成員．ASIACRYPT '96 プログラム委員長．ASIACRYPT 2000 実行委員長．電子情報通信学会より「情報セキュリティの基礎理論」への貢献に関して業績賞を受賞．