NAIST-IS-MT1151009

修士論文

η_T ペアリングの並列実装と基底変換による拡大体上の乗算コストの削減

石井 将大

2013年2月7日

奈良先端科学技術大学院大学
情報科学研究科 情報科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

石井 将大

審査委員：

藤川 和利 教授	(主指導教員)
関 浩之 教授	(副指導教員)
山口 英 教授	(副指導教員)
猪俣 敦夫 准教授	(副指導教員)

η_T ペアリングの並列実装と基底変換による拡大体上の乗算コストの削減*

石井 将大

内容梗概

本研究では，ペアリングの並列実装に関して，実装実験を通してペアリングパラメタである超楕円曲線や，その曲線の定義体，或いはペアリングの計算に必要な拡大体の構成について，高速に並列実装出来るパラメタの決定を目的とし，実際にある η_T ペアリングの並列実装に適したパラメタを示した．

ペアリングは，超楕円曲線の Jacobian 上の群構造を保つ双線形写像であり，その数学的性質を用いて，ID ベース暗号やブロードキャスト暗号，ひいては関数型暗号等に応用されている．一般的に，ペアリングに係る演算は種数の高い代数曲線を選択すると計算コストはより高くなる．様々なペアリングを用いたセキュリティレベルの高い暗号の構成のために，ペアリングの高速化の研究は必要である．本研究において，ペアリングの高速化として GPU を用いた並列計算に取り組み，実際に η_T ペアリングの並列実装を行った．更に，もう一つのテーマとして，有限体の基底変換について研究を行った．具体的には拡大体上の乗算の並列実装と，効率的な乗算が行える拡大体の構成への基底変換を適用することにより，η_T ペアリングのより高速な並列実装を行い検討した．特に先行研究の拡大体の構成に対して効率的な並列化手法を示した．

キーワード

超楕円曲線暗号，η_T ペアリング，GPGPU, 基底変換

*奈良先端科学技術大学院大学 情報科学研究科 情報科学専攻 修士論文, NAIST-IS-MT1151009, 2013 年 2 月 7 日.

A parallel implementation of η_T pairing and reduction of a cost of multiplications on extention fields by changes of basis*

Masahiro ISHII

Abstract

In this study, I focused on efficient parameters of pairings and the extension field that can be implemented fast in parallel, and I tried to look for the parameters by the timing simulation and a parallel implementation of pairings. Also I showed the parameter of an η_T pairing suitable for parallel implementation.

Bilinear pairings on Jacobian of hyperelliptic curves have been applied into many cryptographic schemes; ID-based cryptography, functional encryption and so on. Generally, the pairing computation is complex and its cost is much larger when using curves of higher genus such as hyperelliptic curves. I implemented the pairing in parallel using GPU for speeding up computation of the η_T pairing and furthermore, I studied change of basis of finite fields. In this study, I applied techniques of parallelizing multiplication of the extension field and basis conversion of the extension field to effective computation of the η_T pairing.

From simulation result for parallel computation of η_T pairing, I showed an efficient parameter of the parallelized pairing in particular that I showed how to construct the extension field for the efficient parallelized η_T pairing.

Keywords:

hyperelliptic cryptosystems, η_T pairing, GPGPU, change of basis

*Master's Thesis, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1151009, February 7, 2013.

目 次

図 目 次

表 目 次

1. 序論

ペアリングは，楕円曲線上の点の集合，或いは，超楕円曲線の Jacobian 上の群構造を保つ双線形写像であり，その数学的性質を用いて，ID ベース暗号やブロードキャスト暗号，ひいては関数型暗号等に応用されている．ペアリングはこの様な柔軟で機能が豊富である暗号プロトコルを簡単に実現できる反面，一般的にペアリングに係る演算は，楕円曲線上の演算等に比べ複雑で，種数の高い代数曲線を選択すると計算コストはより高くなる．従ってセキュリティレベルを保ちながら，ペアリングによる暗号化，復号，又，超楕円曲線の Jacobian の因子計算，楕円曲線上の点のスカラー倍算などの，ペアリング計算の一部を成す処理の効率化の研究は必須である．

本研究では，ペアリングの並列実装に関して，実装実験を通してペアリングパラメタである超楕円曲線や，その曲線の定義体，或いはペアリングの計算に必要な拡大体の構成について，高速に並列実装出来るパラメタの決定を目的としている．そこで，本研究ではペアリングの高速化として GPU を用いた並列実装に取り組む．

今日 GPGPU 技術を用いた研究が盛んに行われ，暗号の分野でも並列処理により，暗号化，復号，又は，暗号解読における処理時間を劇的に短縮することが可能となってきている．ペアリング暗号は比較的に計算処理速度が大きいが，ペアリング写像の効率的なアルゴリズムの研究は盛んで，特別な定義体や拡大体上の算術の高速実装技術をうまく適用して，ある条件では高速なペアリングのアルゴリズムが提案されており，大変高速に実装できるペアリングも存在する．GPGPU によるペアリング実装に関しては，有限体上の演算等の基礎的な算術を GPU 上で高速に実装することによるペアリングの高速化等は行われているが，全体として並列化を適用させる実装はほとんど見かけず，又，並列化の適用が難しい部分もある．

本研究では，特に超楕円曲線上の η_T ペアリングについて，効率的な GPU 実装方針を考察し，実装実験による実行時間のシミュレーションを行い，いかに並列化による大きな高速化の効果が得られるかを調査，研究する．ペアリングに用いる曲線の選択，構成法は楕円曲線についてはよく知られており，沢山の選択肢が

ある．超楕円曲線について，さらに一般的なアーベル多様体に対しても，ペアリングに適した特別な曲線，多様体を選ぶことができる．本論文ではそこに立ち入らず，選ばれた曲線に対し GPU 上の実装法を考察する．特に，ペアリングの計算で必要となる，ペアリングに用いる超楕円曲線の定義体である基礎体の拡大体における乗算について，大きな並列化の効果が得られる様な実装を試みる．具体的には基礎体の基本算術である加減乗算の並列化と，拡大体の乗算を，基礎体の元を用いた Karatsuba method で計算することにより乗算のコストを減らし，更にその Karatsuba method にも並列化を適用し，基礎体の並列化と合わせて二重の並列実装を行うことによりペアリング全体の高速化を図る．又，拡大体の算術は，その拡大体の構成にも依り，適当な拡大体の構成を考察することには意義がある．

本研究では，η_T ペアリングの並列実装と，もう一つのテーマとして有限体の基底変換について取り組む．有限体，或いはその拡大体は基礎体上のベクトル空間と見ることができ，基底を持つ．拡大体の元を実際に計算するときは，それぞれの元を基底表示し，その基底が持つ性質，或いは，拡大体の構造に従って行うが，基底によって計算量は異なり，特別な基底においては，ある特別な演算を大変小さなコストで処理することが出来る．即ち，拡大体の算術において，その算術に適した基底に変換することは，全体の計算のコスト削減に繋がる．本研究は拡大体上の乗算の並列実装と，効率的な乗算が行える拡大体の構成への基底変換を適用することにより，η_T ペアリングの高速実装に取り組むものである．さらに，有限体の基底変換について，その変換行列の求解と変換コストについて，シミュレーションを行い考察を与える．

本提案手法に関して，拡大体上の Karatsuba method による乗算の並列化とその並列化に適した拡大体の構成については，GPGPU によるペアリングの並列化としてはこれまでに例を見ない有効な手法であり，基底変換の求解においても，任意の拡大体の間の高速な変換行列探索手法として，従来手法に対する優位性が示せた．

最後に，本論文の構成を述べる．第 2 章は本論文で必要となる数学的な専門用語の定義，それらの基礎的性質等の説明のために当てている．第 2.1 節では，超楕

円曲線について，曲線そのものの定義，曲線の Jacobian とその上の因子計算, 又，超楕円曲線を用いた暗号技術について述べる．第 2.2 節においては，本研究で扱う η_T ペアリングだけではなく，超楕円曲線上のペアリングの背景，現在の技術について，又，本論文で必要となる事柄についてまとめて記した．第 2.3 節では，ペアリングの計算として基礎函数である，基礎体上の算術，又，特に pairing-friendly field 等特別に効率的な実装が行える体上の算術や，Karatsuba method の拡大体の乗算への適用方法について見る．

第 3 章では，本研究で対象とする Barreto ら [5] による種数 2 の超楕円曲線上の η_T ペアリングについて，そのペアリングパラメタやアルゴリズムについて詳細に述べる．特に本研究で重要となる，η_T ペアリングの計算における拡大体上のある乗算について，その乗算に掛かるコストを基礎体の算術に対するコストを用いて評価した．

第 4 章では拡大体の基底変換について，基底変換の求解と変換コストについて，シミュレーションを通して考察を与えた．又，η_T ペアリングにおいて適用させるコスト最小の基底変換を計算しペアリングの高速化に活用した．

本研究では，Barreto ら [5] による η_T ペアリングに関して，彼らの拡大体の構成とは異なる構成におけるペアリングとしてアルゴリズムを書き直し，より高速な η_T ペアリングの並列実装を行った．第 5 章において，その拡大体の構成の変更によるアルゴリズムや，計算コストの差異について詳述した．更に，η_T ペアリングの並列実装に関して，基礎体，拡大体，η_T ペアリングの計算で行われる特別な乗算函数について，具体的に並列実装方針を示し，実行時間のシミュレーション結果を示した．又，ペアリングの計算以外の並列実装については，基礎体，拡大体のパラメタに対する実行時間についての考察を与えた．

第 6 章では，η_T ペアリングの並列実装に関して，その実行時間のシミュレーション結果から評価を行い，並列実装に効果的な，即ち，実行時間そのものが小さい，或いは，並列化による高速化率が高い場合の拡大体の構成や，ペアリングパラメタについて考察を与え，基礎体のパラメタに対して，効率的な並列化手法を示した．又，今回の GPGPU 技術を用いた並列実装に関し課題となる点を幾つか挙げた．更に，有限体の基底変換に関して，特に normal basis への変換行列求

解の提案手法について考察を与え，幾つか今後の課題を議論した．又，変換コストについては，normal basis への変換コストに関して，特に sparse な既約多項式の根が normal basis を成す条件について考察を与えた．

最後に第 7 章において，本論文の結論を述べる．

2. 数学的準備と研究背景

2.1 超楕円曲線暗号

本章では，本論文において必要となる，超楕円曲線を用いた暗号技術に関する，数学的な基本性質，専門用語などの解説，準備を行う．主に，[3, 16] を参考にして解説し，特に注意しなければ，記号はこれらの文献に従う．第 2.1.1 節において，楕円曲線，又，超楕円曲線の定義とその諸性質について，特に暗号の観点から，曲線の Jacobian について述べる．第 2.1.2 節では，Jacobian 上における算術，因子計算について説明し，最後に，第 2.1.3 節において超楕円曲線を用いた暗号技術の概要を述べる．

2.1.1 超楕円曲線とその Jacobian

本節では，超楕円曲線とその Jacobian について，それらの定義，基本性質について数学的な解説を与える．又，本論文では一般的な定義を与えず，超楕円曲線暗号，ペアリングの説明に十分である様に用語の定義，説明を行う．

定義 1. K を体とする．非特異曲線 C/K が

$$C\colon \{(x,y) \in K^2 \mid y^2 + h(x)y = f(x)\} \tag{†}$$

と定義されているとき**種数 $g \geq 1$ の超楕円曲線**と呼ばれる．但し，$h \in K[x]$ は高々 g 次の，$f \in K[x]$ は高々 $2g+1$ 次の monic 多項式である．

超楕円曲線は楕円曲線の一般化と考えられ，楕円曲線と同様，その Jacobian に群構造を持つ．以下では，曲線上の暗号を実現する数学的構造，即ち，超楕円曲線上の Jacobian とその上の因子計算について述べる．因子計算の詳細については第 2.1.2 節で説明する．

楕円曲線暗号では，特別に曲線上の点の演算により楕円曲線上の離散対数問題が考えられるが，一般的に超楕円曲線では代数多様体の因子の計算が対応する．

定義 2. $\overline{K}$ を体 K の代数閉包とする．曲線 C/K について，形式和

$$D = \sum_{P \in C(\overline{K})} n_P P, \quad (n_P \in \mathbb{Z})$$

（但し n_P は有限個を除き 0）を C 上の**因子**という．この因子 D に対し，

$$\deg\, D = \sum_{P \in C(\overline{K})} n_P$$

を D の**次数**と呼ぶ．n_P を D の P における**位数**と呼び，$\mathrm{ord}_P(D)$ と表す．

C 上の有理函数 f に対し，

$$\mathrm{div}(f) = \sum_{P \in C(\overline{K})} \mathrm{ord}_P(f) P$$

を**有理函数 f の因子**と呼ぶ．$\mathrm{ord}_P(f)$ は f の零点，極の位数であり，$\mathrm{div}(f)$ は次数 0 の因子である．

C 上の因子からなる自由 Abel 群を Div_C と表し，**因子群**と呼ぶ．$\mathrm{Div}_C^0 \subset \mathrm{Div}_C$ を C 上の次数 0 の因子からなる部分群とする．C 上の因子 D について，ある有理函数 f が存在して $D = \mathrm{div}(f)$ とかけるとき，D を**主因子**と呼ぶ．

C 上の主因子からなる集合は Div_C の部分群を成し，その部分群で因子群を割った剰余群を**因子類群**と呼び Pic_C と表す．又，因子類群のうち，次数 0 の因子で代表される同値類からなる部分群を Pic_C^0 と表す．

定義 3. 超楕円曲線 C に対し，Pic_C^0 を C の **Jacobian** と呼び，$\mathrm{Jac}_C(\simeq \mathrm{Pic}_C^0)$ とかく．なお，定義体 K 上において考慮している場合には，$\mathrm{Jac}_C(K)$ と強調して表すことがある．

2.1.2 Jacobian 上の因子計算

超楕円曲線暗号では，曲線の Jacobian 上の群演算を行う．即ち抽象的な定義ではなく，因子の表現を具体的に書き下し，実際に計算が可能でなければならない．

定義 4. 種数 g の超楕円曲線 C/K 上の因子 D が次の様に表せるとき**半被約因子 (semi-reduced divisor)** と呼ばれる.

$$D = \sum n_i P_i - \left(\sum n_i\right) \infty,$$

但し，$n_i \geq 0$ で，$n_i \neq 0$ のとき $P_i \neq \widetilde{P_i}$（$\tilde{P}$ は点 P の y 座標の符号を反転させた点）ならば $n_{\widetilde{P_i}} = 0$ とする．更に，$\sum n_i \leq g$ のとき D を**被約因子 (reduced divisor)** と呼ぶ.

C が定義 1 の定義多項式 (†) により定義されているとする．C 上の被約因子 $D = \sum n_i P_i - (\sum n_i)\infty$ に対し，$P_i = (x_i, y_i)$, $u(x) = \prod(x - x_i) \in K[x]$ として，次の性質を満たす多項式 $v \in K[x]$ が一意に存在する.

1. u は monic,
2. $\deg v < \deg u \leq g$,
3. $u | v^2 + vh - f$,

であり，$y_i = v(x_i)$ を満たす．このとき，$D = \text{g.c.d.}(\text{div}(u(x)), \text{div}(v(x) - y))$（文献 [16, 定義 A.4.2] を参照されたい）となる．この様な多項式 u, v により D を表す方法を **Mumford representation (Mumford 表現)** と呼び，$D = (u, v)$ と表す.

因子類群においては，次数 0 の因子 D に代表される因子類に対し，代表元として被約因子を一意にとることができる．従って，$\text{Jac}_C (\simeq \text{Pic}^0_C)$ の元を一意に被約因子で表現できる.

被約因子と Mumford 表現を用いることにより，因子計算を実際に計算する手続きである Cantor's algorithm [4, §5.1] を以下の Algorithm 1 に示す.

記号の詳細については述べないが，各ステップでは，多項式の基礎演算（四則演算），最大公約元 GCD を求める計算からなり，因子計算は多項式の演算を繰り返して求められる.

Algorithm 1 Cantor's algorithm [4, §5.1, Algorithm 2]

INPUT: $D_1 = (u_1, v_1),\ D_2 = (u_2, v_2),\ D' = (u, v)$

OUTPUT: $\rho(D_1 + D_2),\ f(x, u(x)) \pmod{u},\ g(x, v(x)) \pmod{u}$,
$\mathrm{lc}_\infty(h_{D_1,D_2});\ h_{D_1,D_2} = f/g$

1: Compute (d_1, e_1, e_2) such that $d_1 = e_1u_1 + e_2u_2 = \gcd(u_1, u_2)$
2: Compute (d, c_1, c_2) such that $d = c_1d_1 + c_2(v_1 + v_2 + H) = \gcd(d_1, v_1 + v_2 + H)$
3: $s_1 \leftarrow c_1e_1,\ s_2 \leftarrow c_1e_2,\ s_3 \leftarrow c_2$
4: $U \leftarrow (u_1u_2)/d^2,\ V \leftarrow (s_1u_1v_2 + s_2u_2v_1 + s_3(v_1v_2 + F))/d \pmod{U}$
5: $f \leftarrow d \pmod{u},\ g \leftarrow 1,\ h \leftarrow 1$
6: **while** $\deg(U) > g$ **do**
7: $\quad U' \leftarrow (F - VH - V^2)/U,\ V' \leftarrow (-H - V) \pmod{U}$
8: $\quad f \leftarrow f(u - V) \pmod{u}$
9: $\quad g \leftarrow gU' \pmod{u}$
10: $\quad$ **if** $\deg(V) > g$ **then**
11: $\quad\quad h \leftarrow -\mathrm{lc}(V)h$
12: $\quad$ **end if**
13: $\quad U \leftarrow U',\ V \leftarrow V'$
14: **end while**
15: **return** $(U, V), f, g, h$

2.1.3 超楕円曲線を用いた暗号

本節では，超楕円曲線を用いた暗号について，原理と基礎的な事柄を説明し，特に暗号のセキュリティレベルについて具体的に述べる．

楕円曲線暗号，或いは超楕円曲線を用いた暗号は，先の第 2.1.2 節で述べた曲線の Jacobian における離散対数問題を用いて構成される．一般的に群 G に対して，**G における底 $g \in G$ に対する離散対数問題**とは，$y \in G$ が与えられ，

$$g^x = y$$

なる整数 x が在れば，その x を探すことである．超楕円曲線 C と，その Jacobian

Jac_C において，因子 D, E

$$[x]D = E$$

なる x を見つけられるか，という問題を**超楕円曲線上の離散対数問題**等と呼ぶことがある．特に C が楕円曲線の時は，$[x]P = Q$ として楕円曲線上のスカラー倍算により計算が行われる．超楕円曲線上の離散対数問題を用いて，Diffie-Hellman 鍵交換，DSA 署名，ElGamal 暗号等のプロトコルを構成することが出来，これらの暗号技術を総称して，**超楕円曲線暗号**等と呼ぶ．

次にセキュリティレベルについて簡単に述べる．NIST [2] が発行する SP 800-57 [21] により，実質の世界標準としてのセキュリティレベルが定められている．以下の表 1 に SP 800-57 が定める，それぞれのセキュリティビットに対するブロック暗号，RSA, デジタル署名アルゴリズム，楕円曲線暗号に関するセキュリティレベルについて明記する．

Security bits	Symmetric key algorithm	FFC（DSA 等）	IFC（RSA 等）	ECC（ECDSA 等）
80	2TDEA	1024	1024	160 - 223
112	3TDEA	2048	2048	224 - 255
128	AES-128	3072	3072	256 - 383
192	AES-192	7680	7680	384 - 511
256	AES-256	15360	15360	512 -

表 1 NIST SP 800-57 が定めるセキュリテイレベル

ここで，それぞれ 2TDEA, 3TDEA はそれぞれ鍵を 2 つ，或いは 3 つ用いる triple DEA を表す．FFC (Finite Field Cryptography) は有限体を用いた署名，暗号方式を指し DSA 等が含まれる．表の数値は公開鍵のサイズを示した．IFC (Integer Factorization Cryptography) は素因数分解の困難性による暗号方式等を指し，RSA が主な例である．表に示した値は，この方式で一般的に考えられる鍵長である．ECC (Elliptic Curve Cryptography) は楕円曲線暗号を表し，表の値は楕円曲線上の離散対数問題に関する base point（底）の位数を示す．

楕円曲線暗号については，公開鍵暗号として主要に使われている暗号方式 RSA と比べ，同等のセキュリティレベルで鍵長が小さいという利点を持ち，様々な高速実装に関する研究により，楕円曲線暗号も RSA と同等，或いはより高速な実装が可能になっている．

最後に，より一般に超楕円曲線暗号について述べる．ここでは特に，種数 2 の超楕円曲線を用いた暗号について，主に [4, §3] を参考にして述べる．種数 g が高い超楕円曲線は，楕円曲線とは状況が異なり，同等の群の位数を持つものでもセキュリティレベルが異なる場合がある．それは種数によって，離散対数問題に対する攻撃方法である rho 法，更に強力な index calculus 等の適用範囲が異なるためであり，楕円曲線暗号については準指数時間による攻撃方法が見つかっていないが，$g = 3, 4$ のとき，index calculus が適用できる．又，超楕円曲線上のペアリングのセキュリティを考慮する際，曲線の埋め込み次数（詳細は後の第 2.2.2 節で述べる）が重要なパラメタである．

今，$C/\mathbb{F}_q$ を体 $\mathbb{F}_q$ 上で定義された超楕円曲線として，$r|(q^k - 1)$ なる整数 r と拡大体 $\mathbb{F}_{q^k}/\mathbb{F}_q$ を考える．この様な条件を満たす k の内最小のものを，C の埋め込み次数と呼ぶが，この時，例えばペアリングにより C 上の離散対数問題を，有限体 $\mathbb{F}_{q^k}$ 上の離散対数問題に帰着出来る．従って，ペアリングにおいては，曲線上，有限体上のセキュレティレベルについて，同時に考えなければならない．又，$\mathrm{Jac}_C(\mathbb{F}_q)$ の位数については，C の種数により

$$\sharp\mathrm{Jac}_C(\mathbb{F}_q) = q^g + O(q^{g-1/2})$$

として評価することが出来（[4, §3.1] を参照されたい），種数が大きい時，セキュリティレベルを保ち定義体のサイズを落とすことが出来る．

表 2 に種数 2 の超楕円曲線について，各セキュリティレベルに対応する Jacobian, 拡大体の位数，ρ 値と埋め込み次数の値を示す．ここで，ρ 値は $\rho = g \log q / \log r$ として定義され，ペアリング暗号においては ρ 値が小さい程高いセキュリティレベルを保ち効率的なペアリング計算が行える点で，ρ 値の小さな超楕円曲線（pairing-friendly 曲線）の探索は意義のある研究分野の一つとなっている．

Security bits	Subgroup size of $\mathrm{Jac}_C(\mathbb{F}_q)$ (r)	Extension field size (q^k)	Embedding degree (k)					
			$\rho \approx 1$	$\rho \approx 2$	$\rho \approx 3$	$\rho \approx 4$	$\rho \approx 6$	$\rho \approx 8$
80	160	1024	$6g$	$3g$	$2g$	$1.5g$	g	$0.8g$
112	224	2048	$10g$	$5g$	$3.3g$	$2.5g$	$1.6g$	$1.3g$
128	256	3072	$12g$	$6g$	$4g$	$3g$	$2g$	$1.5g$
192	384	7680	$20g$	$10g$	$6.6g$	$5g$	$3.3g$	$2.5g$
256	512	15360	$30g$	$15g$	$10g$	$7.5g$	$5g$	$3.8g$

表 2 種数 2 の超楕円曲線の埋め込み次数に関するセキュリティレベル ([4, §3.2, Table 3.1])

表 2 の種数 g は $g = 2$ を意味するが，他の種数に対してもここで示した値が基本的には当てはまる．但し，先に述べた様に，$g = 3, 4$ の時は index calculus の適用が可能であるため $\mathrm{Jac}_C(\mathbb{F}_q)$ の位数として少し大きな値を取らなければならない．詳細は [4, §3.2] を参照されたい．

2.2 ペアリング

本節ではペアリングについて，その基本的な定義，性質と，現在のペアリングの姿に関して網羅的に述べる．特に，本研究で対象としている η_T ペアリングについて詳細に説明する．第 2.2.1 節においてペアリングのルーツ，定義について述べ，第 2.2.2 節において，現在様々な研究が行われている超楕円曲線上のペアリングについて，その構成方法等を説明する．更に，実際にペアリングを計算する主要なアルゴリズムである Miller's algorithm について述べる．

2.2.1 ペアリングの背景

ここではペアリングのルーツについて簡単に述べる．詳しい説明は [3, Chapter 6] 等の文献を参照されたい．

まず初めに，ペアリングについては，数学，特に代数幾何，整数論の研究において研究道具として Weil pairing が用いられ，最近になって暗号に応用されてきたという経緯がある．楕円曲線上の Weil pairing については [23, Chapter III] に詳しく説明されているが，ここで少しその性質を述べる．

体 k ($\mathrm{char}(k) = p > 0$) と楕円曲線 E/k に対し，曲線上の m-torsion point を集めた部分群 $E[m]$（m 位の点がなす部分群）は

$$E[m] \simeq \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$$

として，階数 2 の $\mathbb{Z}/m\mathbb{Z}$ 上の自由加群の構造を持つ．[23, Chapter III.§ 8] に詳細な記述があるが，この構造と determinant を用いたペアリングについては，Galois 群の作用により不変でないため，以下に示す Weil pairing が提案された．

$$\begin{aligned} e_m \colon E[m] \times E[m] &\to \mu_m \\ (S, T) &\mapsto \frac{g(X+S)}{g(X)} \end{aligned}$$

として，e_m は双線型写像となる．但し，O を加法に関する単位元とし，g は

$$f \circ m = g^m \quad (\mathrm{div}(f) = m(T) - m(O),\ T \in E[m])$$

を満たす様な有理関数である．この様な函数を用いることにより原始 m 乗根 $g(X+S)/g(X)$ を得ている．このとき，e_m は以下の多くの良い性質を持つ．

- 双線型性
- 非退化
- Galois 群 $\mathrm{Gal}(\overline{k}/k)$ の元の作用により不変である．即ち，

$$e_m(S, T)^\sigma = e_m(S^\sigma, T^\sigma), \quad \sigma \in \mathrm{Gal}(\overline{k}/k).$$

- compatible である．即ち，

$$e_{mm'}(S, T) = e_m([m']S, T), \quad (S \in E[mm'], T \in E[m]).$$

これらの性質より，ペアリング暗号でも Weil pairing を使用することが出来る．即ち，同様にして超楕円曲線の Jacobian 等のアーベル多様体に対し，双線型性を用いた離散対数問題を暗号に応用できる．

更に，暗号としてより効率的に使用できるペアリングとしてTate pairingがある．Tate pairingの詳しい説明は[3, §6.2, §6.3]を参照されたい．本論文では，これ以上ペアリングの数学的な背景について述べないが，Tate pairing, 更に暗号として効果的なTate-Lichtenbaum pairingについては次の第2.2.2節で詳しく述べる．

2.2.2 超楕円曲線上のペアリング

本節では，超楕円曲線上のペアリングと，そのアルゴリズムの詳細について述べる．ここでも説明は主に文献[3, 4]の記法に従う．又，特に本研究で対象としているη_Tペアリングについて，[5]を参考にして子細に述べる．

Tate-Lichtenbaum pairingとその変形

具体的に記述され，計算が行えるペアリングとしてTate-Lichtenbaum pairingがある．以下，$C/\mathbb{F}_q$を体$\mathbb{F}_q$上で定義された超楕円曲線とする．$r|(q^k-1)$なる整数rと拡大体$\mathbb{F}_{q^k}/\mathbb{F}_q$を考える．

定義 5. $D \in \mathrm{Jac}_C(\mathbb{F}_{q^k})[r]$について，$f_D$を主因子$rD$の$C$上の有理函数とする．又，$E = \sum n_P P$とし，$D$と共通の素因子を持たないとする．この時，次のnon-degenerateな双線型写像

$$T_r \colon \mathrm{Jac}_C(\mathbb{F}_{q^k})[r] \times \mathrm{Jac}_C(\mathbb{F}_{q^k})/r\mathrm{Jac}_C(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^r$$
$$(D, E) \mapsto f_D(E) = \prod_P f_D(P)^{n_P}$$

を**Tate-Lichtenbaum pairing**と呼ぶ．

又，Jac_Cが位数r^2の元を持たないとき$\mathrm{Jac}_C(\mathbb{F}_{q^k})[r] \simeq \mathrm{Jac}_C/r\mathrm{Jac}_C(\mathbb{F}_{q^k})$より，

$$T \colon \mathrm{Jac}_C(\mathbb{F}_{q^k})[r] \times \mathrm{Jac}_C(\mathbb{F}_{q^k})[r] \to \mu_r \subset \mathbb{F}_{q^k}$$
$$(D, E) \mapsto T_r(D, E)^{(q^k-1)/r}$$

としてTate-Lichtenbaum pairingを修正したものが得られる．$T_r(D, E)$を$(q^k - 1)/r$乗することによりμ_rの値としてペアリングの値が一意に決まるが，この操作を**最終冪 (final exponentiation)**という．

楕円曲線，超楕円曲線，更に一般に種数の高い代数曲線に対し定義される，効率的で non-degenerate な pairing として Ate pairing [9] が提案された．

ペアリングについて，理論的な面を説明する際のごく一般的な状況で考える（Galois コホモロジーなど数学的な道具を用いる．詳細は文献 [3, Chapter 6] を参照されたい）．$\mathbb{G}_i$ $(i = 1, 2)$ を Abel 群とし，$\mathbb{G}_3$ を巡回群として，双線型写像

$$\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$$

を考える．超楕円曲線上のペアリングでは G_i $(i = 1, 2)$ が曲線の Jacobian の部分群等に当たる．

以降，次の様な状況で超楕円曲線，又，その Jacobian について考える．$C/\mathbb{F}_q$ を超楕円曲線とし，r を $\sharp\mathrm{Jac}_C(\mathbb{F}_q)$ を割る素数，k を $\mathrm{Jac}_C(\mathbb{F}_q)$ の r に関する**埋め込み次数（$r|(q^k-1)$ を満たす整数 k のうち最小のもの）**とする．π を q 乗 Frobenius map とし，

$$\mathbb{G}_1 := \mathrm{Jac}_C(\mathbb{F}_q^k)[r] \cap \ker\,(\pi - [1])$$
$$\mathbb{G}_2 := \mathrm{Jac}_C(\mathbb{F}_q^k)[r] \cap \ker\,(\pi - [q])$$

とする．

定義 6. 上の状況において，次の写像

$$a\colon \mathbb{G}_2 \times \mathbb{G}_1 \to \mu_r$$
$$(E, D) \mapsto f_E(D)$$

を **Hyperelliptic Ate Pairing** と呼ぶ．

Ate ペアリングは一般的なペアリングを考えているが，その中でも計算効率が良いもの（後述する Miller's algorithm のループ削減など）が提案されてきた．文献 [4, §4.5] では Hess [10], Vercauteren [25] によるタイプの超楕円曲線上のペアリングを **HV Pairing** として簡潔に述べてある．HV ペアリングには R-ate pairing などが含まれ，現在の最新のペアリングとして研究が進められている．本稿では，これらのペアリングには触れない．

他に，特に効率的に実装できるペアリングとして Twisted Ate pairing がある．これは特別に曲線 C に対し，“twist”（他の曲線への同型写像）と呼ばれるものが 在るとき，Ate ペアリングに対し群 $\mathbb{G}_1, \mathbb{G}_2$ の役割を入れ替えることから，計算コストの省略を行うものである．定義の詳細は文献 [4, §4.7] を参照されたい．

定義 7. 次数 δ の twist を持つ超楕円曲線 $C/\mathbb{F}_q$ に対し，$m = \gcd(k, \delta),\ e = k/m$ として，写像

$$a^{twist} \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mu_r$$
$$(D, E) \mapsto f_D(E)$$

を **Hyperelliptic Twisted Ate Pairing** と呼ぶ．ここで $\mathbb{G}_2$ は twist により $\mathbb{F}_{q^e}$ 上の Jacobian の部分群として定義できる．

ペアリングの計算アルゴリズム，即ち $f_D(E)$ について，有理函数と函数値の評価を行う Miller's algorithm について述べる．ここでは超楕円曲線上の Tate ペアリングについて，基本的な形の Miller's algorithm について見る．簡単のため，文献 [4, §5.1] の状況を下にして説明する．即ち，Tate ペアリング

$$T_r \colon \mathrm{Jac}_C(\mathbb{F}_{q^k})[r] \times \mathrm{Jac}_C(\mathbb{F}_{q^k})/r\mathrm{Jac}_C(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^r$$
$$(D, E) \mapsto T_r(D, E) = f_D(E) =: f_{r,D}(E)$$

を考える．以下の様に因子の Mumford 表現を用いることにより，$f_{r,D}$ とその函数値を計算することができる．因子 D に対し $\rho(D)$ を D と線型同値な被約因子とする．よって $\mathrm{div}(f_{i,D}) = iD - \rho(iD)$ である．関係式

$$f_{i+j,D} = f_{i,D} f_{j,D} h_{iD,jD}, \quad \mathrm{div}(h_{iD,jD}) = \rho(iD) + \rho(jD) - \rho(iD + jD)$$

より再帰的に $f_{r,D}$ が $\log_2 r$ ステップで計算できることが分かる．

Miller アルゴリズムの流れは，因子計算として加算，二倍算を繰り返す，binary 法からなる．又，被約因子の Mumford 表現により，多項式の計算に帰着されている．具体的な因子計算は後の Cantor's algorithm で与えられる．最後の step の函数 Res() は終結式の計算であり，有理函数値の評価として，効率的な計算のため用いられる．

Algorithm 2 Miller's algorithm [4, §5.1, Algorithm 1]

INPUT: $D_1 = (u_1, v_1),\ D_2 = (u_2, v_2),\ d,\ k = \sum_{i=0}^{N} k_i 2^i$

OUTPUT: $f_{k,D_1}^{\mathrm{norm}}(\epsilon(D_2))^d$

1: $D \leftarrow D_1$
2: $f_1 \leftarrow 1,\ f_2 \leftarrow 1,\ f_3 \leftarrow 1$
3: **for** $i = N - 1$ downto 0 **do**
4: $\quad f_1 \leftarrow f_1^2 \pmod{u_2},\ f_2 \leftarrow f_2^2 \pmod{u_2},\ f_3 \leftarrow f_3^2$
5: $\quad (D, h_1, h_2, h_3) \leftarrow \mathrm{Cantor}(D, D, D_2)$
6: $\quad f_1 \leftarrow f_1 h_1 \pmod{u_2},\ f_2 \leftarrow f_2 h_2 \pmod{u_2},\ f_3 \leftarrow f_3 h_3$
7: $\quad$ **if** $k_i = 1$ **then**
8: $\quad\quad (D, h_1, h_2, h_3) \leftarrow \mathrm{Cantor}(D, D_1, D_2)$
9: $\quad\quad f_1 \leftarrow f_1 h_1 \pmod{u_2},\ f_2 \leftarrow f_2 h_2 \pmod{u_2},\ f_3 \leftarrow f_3 h_3$
10: $\quad$ **end if**
11: **end for**
12: **return** $\left(\mathrm{Res}(f_1, u_2)/(f_3^{\deg(u_2)} \cdot \mathrm{Res}(f_2, u_2))\right)^d$

η_T ペアリング

η_T ペアリングは，上で述べた hyperelliptic twisted Ate pairing の一種であり，Barreto らにより [5] においてその構成が，具体的な例を用いて提案された．楕円曲線，或いは低い種数を持つ超楕円曲線上のペアリングとして，最も高速な実装が行われているものの内の一つである．本研究ではその一例を取り上げ，本提案手法において効率的に並列実装を行った．Barreto らは Duursma, Lee [7] による，ある supersingular な超楕円曲線に対する，効率的な Tate ペアリングを大幅に改良したペアリングを η_T ペアリングとして構成した．以下にその構成方法を述べる．ここで楕円曲線 $E/\mathbb{F}_q$ が **supersingular** であるとは，$E/\overline{\mathbb{F}}_q$ において位数 p $(q = p^m)$ の点が存在しない（同値であることの $\sharp E/\mathbb{F}_q \equiv 1 \pmod{p}$）ことであり，超楕円曲線が supersingular であるとは，その Jacobian が supersingular な楕円曲線の直積と同種となることである．

Supersingular な超楕円曲線 $C/\mathbb{F}_q,\ (q = p^m)$ について，埋め込み次数が $k > 1$

で，次の distortion map (twist に相当する)

$$\psi \in \mathrm{End}(C(\mathbb{F}_{q^k})), \quad p \in C(\mathbb{F}_q) \Rightarrow \psi(p) \in C(\mathbb{F}_{q^k})$$

として，$\psi(p)$ の x 座標が中間体 $\mathbb{F}_{q^{k/2}}$ に属するものを持つものを考える．この distorition map の条件はペアリングの分母消去のテクニックを用いるためのものである．詳しくは [7] を参照されたい．D を $C(\mathbb{F}_q)$ 上の因子として，自然数 n に対し因子 nD と同値である被約因子を D_n とかき，ある m について $f_{n,D}$ を $nD - D_n - m(\infty)$ に対応する有理函数とする．$n < 0$ の時は，$nD = (-n)(-D)$ で，$D_n \sim (-n)(-D)$ の時，$f_{n,D}$ は $(-n)(-D) - (D_n) - m(\infty)$ となるので，Tate ペアリング

$$\langle D, D' \rangle_N = f_{N,D}(D')$$

が定義できる．Barreto らは，この時ある整数 T を選び Miller loop の長さを大幅に削減する Tate ペアリングが構成できることを示し，その重要なパラメタ T を用いて η_T ペアリング

$$\eta_T(D, D') = f_{N,D}(\psi(D'))$$

を定義した．以下にパラメタ T と他の η_T ペアリングに必要なパラメタについて述べる．

今，D を $C(\mathbb{F}_q)$ 上の因子とし，ある整数 N について $\sharp C(\mathbb{F}_q) | N$ とする．更に，$M = (q^k - 1)/N$ として，以下を満たす T が存在するとする．

- ある $\gamma \in \mathrm{Aut}_C(\mathbb{F}_q)$ が存在し，因子類群に置いて $TD \equiv \gamma(D)$ が成り立つ．
- 任意の $Q \in C(\mathbb{F}_q)$ に対し $\gamma\psi^q(Q) = \psi(Q)$ が成り立つ．
- ある $a \in \mathbb{Z}_{\geq 0}$ と $L \in \mathbb{Z}$ により $T^a + 1 = LN$ が成り立つ．
- ある $c \in \mathbb{Z}$ により $T = q + cN$ と書ける．

以上を満たす T に対し，

$$(\langle D, D' \rangle_N^M)^L = (\eta_T(D, D')^M)^{aT^{a-1}} \tag{1}$$

が成り立ち ([5, §4, Theorem 1]), 特に N のパラメタの取り方により，Miller loop 長の削減を達成できる Tate ペアリングとして η_T ペアリングを構成できる．η_T ペアリングは上記のように，全ての条件を満たす様にうまくパラメタを取った時の式 (1) の形のものを指す．実際に Miller loop 長の削減方法については本研究の先行研究として，具体的に後の第 3.1 節で述べる，種数 2 の超楕円曲線上の η_T ペアリングを参照されたい．

2.3 有限体上の算術と有限体の基底

本節では，有限体の基本性質について，特に有限体，又，その拡大体の算術の実装の面と，有限体の基底変換に関する事柄に注意して説明する，本研究における並列実装の子細な説明は後の第 5.2.4 節で行われるが，ここでも本実装で採用しているアルゴリズムについては簡単に注意しておく．

最初に，第 2.3.1 節で有限体の算術について，拡大体の構成，その上の算術と実装方法について，基礎的な事柄の説明と，本研究の実装面から注意を与える．次に第 2.3.2 節では，本実装には直接的に関わりはないが，pairing-friendly field について，その基本性質とその上の算術について簡単に述べる．更に，第 2.3.3 節で，本研究でも並列実装に取り入れている，Karatsuba method について説明する．最後に第 2.3.4 節において，本研究の有限体の基底変換に関する準備として，有限体の自己同型などの基本性質，基底に関する事柄について説明する．

2.3.1 有限体上の算術

本節では，有限体の構成等，基本的性質を説明する．又，本研究の実装に関する説明の準備として，拡大体の算術について幾つか述べる．本実装では η_T ペアリングの並列実装を行ったが，まず，ペアリングが持つ曲線の定義体，更に定義体の上の拡大体における算術を実装する必要がある．本実装では，基本的に有限体の算術は，拡大体を多項式基底により基礎体上の有限次ベクトル空間と見て表現し，計算を行った．即ち，拡大体 $\mathbb{F}_{q^k} \simeq \mathbb{F}_q[x]/(f(x))$ を，$\mathbb{F}_q$ 上の k 次の既約多

項式 $f(x)$ を用いて構成した時，$f(x)$ の根の一つを θ とすると

$$\{1, \theta, \theta^2, ..., \theta^{k-1}\}$$

は $\mathbb{F}_{q^k}$ において基底を成し，これを**多項式基底**と呼ぶ．実装においては多項式環 $\mathbb{F}_q[x]/(f(x))$ において，$\{1, x, x^2, ..., x^{k-1}\}$ を多項式基底として取り，算術は根には依らない．算術に必要なのは，定義多項式 $f(x)$ のみであり，加減乗算は $\mathbb{F}_q[x]$ の多項式として計算した後に mod $f(x)$ の計算を行い，除算については Euclidean algorithm による．

本実装では，高速な mod 乗算手法である Montgomery 乗算等は考慮せず，今回の対象である η_T ペアリングは binary field において定義されているが，Window method 等の高速化手法も考察の対象外としており，ここでもそれらの手法について説明しない．本研究は実装実験を行うため，特定の体に関する高速化手法は考察の対象外としている．

上記の理由で，特に binary field においては二乗算，除算について高速化手法があるが本稿では特に説明しない．詳細は [3, §11.2] を参照されたい．

2.3.2 Pairing-friendly Fields

Koblitz と Menezes[17] は，ペアリングの計算が効率的に行うことが出来る特別な拡大体として pairing-friendly field を提案した．この節では pairing-friendly field の定義と基本性質を説明する．

拡大体 $\mathbb{F}_{q^k}$ に対し，$p \equiv 1 \pmod{12}$ で，拡大次数 k が $2^i 3^j$ の形で表される時，$\mathbb{F}_{q^k}$ を **pairing-friendly fields** と呼ぶ．ここで，$j = 0$ ならば $p \equiv 1 \pmod{4}$ で良い．今，$\mathbb{F}_{q^k}$ が pairing-friendly field であるとすると，$\mathbb{F}_{q^k}$ は $\mathbb{F}_q$ 上で二次，或いは三次拡大を繰り返して構成する事ができるが，条件 $p \equiv 1 \pmod{12}$ から，ある既約二項式 $x^2 - \alpha$, 或いは既約三項式 $x^3 - \beta$ に対し，α の平方根，或いは β の三乗根を添加して拡大体の塔を構成できる．従って，それぞれの中間の体拡大が sparse な定義多項式により構成されており，その上での算術を効率的に行える利点が挙げられる．

更に，二次拡大，或いは三次拡大を繰り返して構成されるため，それぞれの体拡大において，二次拡大ならば Karatsuba method, 三次拡大ならば Toom-Cook method を適用することが出来，拡大体のサイズが大きい場合は効率的に乗算が行える利点を持つ．これらの性質から，ペアリングは曲線の定義体上の拡大体における計算が必須であるため，pairing-friendly field が選択できることは利点となる．

2.3.3 拡大体上の Karatsuba method による乗算

Karatsuba method は次数 1 の多項式に限らず，任意の次数に対して同様のアルゴリズムが適用できる ([26, §3.2, Algorithm 2]). 次数 d の多項式

$$A(x) = \sum_{i=0}^{d} a_i x^i, \quad B(x) = \sum_{i=0}^{d} b_i x^i$$

に対して，これらの乗算について考える．各 $i = 0, ..., d,$ に対し，

$$V_i := a_i b_i, \tag{2}$$

を計算し，$0 \leq s < t \leq d,$ に対し，

$$V_{s,t} = (a_s + a_t)(b_s + b_t), \tag{3}$$

を計算する．この時，乗算 $a(x)b(x) = \sum_{i=0}^{2d} c_i x^i$ について，

$$\begin{aligned} c_0 &= V_0 \\ c_{2d} &= V_d \\ c_i &= \begin{cases} \sum_{s+t=i} V_{s,t} - \sum_{s+t=i} (V_s + V_t) & i\text{: odd}, \\ \sum_{s+t=i} V_{s,t} - \sum_{s+t=i} (V_s + V_t) + V_{i/2} & i\text{: even}, \\ \qquad \text{for } 0 \leq s < t \leq d,\ 0 < i < 2d. \end{cases} \end{aligned} \tag{4}$$

として計算できる．

この任意の次数の多項式に対するKaratsuba methodにより，拡大体$\mathbb{F}_{q^k}/\mathbb{F}_q$ $(q = p^m))$ の乗算が，特に基礎体$\mathbb{F}_q$のサイズが大きい時効率的に行える．即ち$\mathbb{F}_{q^k}/\mathbb{F}_q$の元を多項式基底で表現し，その元として高々$k-1$次の多項式$a(x), b(x)$を取った時，多項式としての乗算$a(x)b(x)$をKaratsuba methodにより計算する．本研究においては，この拡大体上の乗算を**k次のKaratsuba methodによる乗算**等と呼ぶが一般的ではない．次数kのKaratsuba methodによる乗算において，基礎体の乗算は$k(k+1)/2$回行われ，通常の乗算 (schoolbook method) に比べ乗算回数を抑えることが出来，基礎体のサイズが比較的大きい時は効率的に乗算が行える．本実装では拡大体上の乗算として，それぞれk次のKaratsuba methodを並列化し，それらを基本演算として計算を行った．

2.3.4 有限体上の自己同型と有限体の基底

この節では有限体上自己同型群の元について，乗法を保つ自己線型写像という観点から基礎的な有限体の性質を述べる．本研究において，η_T ペアリングの計算における拡大体上の乗算について，本提案手法では基底変換を用いることによりそのコストの削減を図っている．その際，より効率的な変換を求める必要がある．本節ではまず，乗算を保つ変換を通して拡大体の元の表現を変えるという動機に従い，乗算を保つ自己線型同型写像について考える．

拡大体$\mathbb{F}_{q^k}/\mathbb{F}_q$は$\mathbb{F}_q$線型空間であり，その自己線型同型写像全体$GL_{\mathbb{F}_q}(\mathbb{F}_{q^k})$の元の内，乗法を保つものについて考える．$f \in GL_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ に対し，fが乗法を保つ，即ち，$f(ab) = f(a)f(b)$ $(a, b \in \mathbb{F}_{q^k})$ を満たすとき，

$$f(1) = f(1 \cdot 1) = f(1)f(1)$$

より，体は整域であるから，$f(1) = 0$, 即ち, $f = 0$ 或いは，$f(1) = 1$ となる．従って，線形性より $\lambda \in \mathbb{F}_q$ に対し，

$$f(\lambda) = f(\lambda \cdot 1) = \lambda f(1) = \lambda$$

となるから，$f|_{\mathbb{F}_{q^k}} = \mathrm{id}_{\mathbb{F}_{q^k}}$ である．よって，$f \in GL_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ が乗法を満たすとき，$f \in \mathrm{Aut}_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ となるから，逆は常に成り立つため，乗算を保つ線型同型から成

る集合は $\mathrm{Aut}_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ 全体となる．従ってガロア群 $\mathrm{Gal}(\mathbb{F}_{q^k}/\mathbb{F}_q) \simeq \mathrm{Aut}_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ の作用が重要であり，この後でも Frobenius map の作用として頻繁に計算が行われる．

ここで拡大体 $\mathbb{F}_{q^k}$ の構成について考える．上の考察は，拡大体の同一の構成における同一の乗算を保つ写像について考察したが，基底変換を用いると，拡大体のそれぞれ別の構成における乗算を保つことができ，より自由度が高くなる．即ち拡大体が二つの基底 $\mathcal{B}, \mathcal{B}'$ により

$$\mathbb{F}_{q^k\mathcal{B}} \xrightarrow{P} \mathbb{F}_{q^k\mathcal{B}'}$$

と表されている時，基底変換 P により

$$P(a_{\mathcal{B}} \cdot_{\mathcal{B}} b_{\mathcal{B}}) = (Pa_{\mathcal{B}}) \cdot_{\mathcal{B}'} (Pb_{\mathcal{B}})$$

が成り立つが，当然 P は体の自己同型とは限らない．先述の通り，本研究では，この基底変換 P の中でも変換コストがより小さいものを選ぶ必要があり，変換コストとして，P を基底により行列表示した際の，0 でない成分の個数として捉えることが出来る．

上の基底変換 P に対し，次の変形が考えられる．Galois 群の元 $\sigma \in \mathrm{Aut}_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ に対し，

$$\mathbb{F}_{q^k\mathcal{B}} \xrightarrow{P^\sigma} \mathbb{F}_{q^k\mathcal{B}'}$$

が考えられる．即ち同型 σ を ${\mathbb{F}_{q^k}}^{\sigma}_{\mathcal{B}}$ として拡大体全体に作用させ，変換 P を施すものであり，逆変換は当然 σ^{-1} と P^{-1} を作用させる変換である．これについて，特に $\mathcal{B}, \mathcal{B}'$ が多項式基底であるとき，σ による作用は定義多項式の根の置換を引き起こすため，多項式基底を成す根の置換により拡大体の構成に変更はないが，基底変換の形は変わり得る．従って，Galois 群の元による作用の分だけ基底変換の形に自由度があり，本研究ではその中でコストが最小のものを求め，本実装に取り入れている．

最後に有限体の基底について述べる．拡大体 $\mathbb{F}_{q^k} \simeq \mathbb{F}_q[x]/(f(x))$ について，$f(x)$ の根の一つを θ により

$$\{1, \theta, \theta^2, ..., \theta^{k-1}\}$$

として表されるものを多項式基底と呼ぶことは先に述べた．ここで

$$\{\theta, \theta^2, ..., \theta^k\}$$

も拡大体において一次独立であり，基底を成す．これを**擬多項式基底**と呼び，状況に応じてこの基底を用いて効率的に計算できる．次に特別な基底として normal basis を挙げる．拡大体の元 $a \in \mathbb{F}_{q^k}$ について，a に Galois 群の元を作用させたもの，即ち共役元を並べたもの

$$\{a, a^{\sigma}, ..., a^{\sigma^{k-1}}\}$$

が基底を成すとき，これを **normal basis** と呼ぶ．形から normal basis で表した元に対し，Galois 群の元，即ち Frobenius map による作用がシフト演算で行える．後に詳しく述べるが，本研究の基底変換の求解において重要な役割を果たす．その他 dual basis や normal basis の中でも特別な，Gauss period normal basis 等特別な性質を持つ基底が幾つかあるが，本論文の範囲外であるためここでは述べない．

3. 先行研究における η_T ペアリング

Barreto らは [5] において，η_T ペアリングを提案し，一例として binary field 上定義された種数 2 の超楕円曲線上の η_T ペアリングを挙げている．本章では，本研究において，先行研究として取り扱う Barreto ら [5] による種数 2 の超楕円曲線上の η_T ペアリングについて，Jacobian 上の計算，ペアリングアルゴリズム等を子細に述べる．

まず，第 3.1 節において，本研究で取扱う η_T ペアリングを定める種数 2 の超楕円曲線，又，その曲線が持つ distortion map 等の η_T ペアリングのパラメタについて具体的に述べる．次に第 3.2 節において，先行研究における η_T ペアリングのアルゴリズム，計算コストを詳細に述べる．

3.1 曲線とペアリングパラメタ

本節では本研究で対象とする，Barreto ら [5] による定義体 $\mathbb{F}_{2^m}$ 上の種数 2 の超楕円曲線を用いた η_T ペアリングに関し，そのペアリングパラメタについて詳細に述べる．記号，用語は [5, §7] に従って記述する．

$(m, 6) = 1$ として，定義体 $\mathbb{F}_{2^m}$ 上の supersingular 超楕円曲線

$$C\colon y^2 + y = x^5 + x^3$$

について考える．この曲線 C は，埋め込み次数が 12 で下記の distortion map を持つ．[8, §8] の状況において，ω を $x^6 + x^5 + x^3 + x^2 + 1$ の根として取り，

$$s_2 = \omega^8 + \omega^4 + \omega, \quad s_1 = \omega^4 + \omega^2$$

として，s_0 を $y^2 + y = \omega^5 + \omega^3$ の根の一つとすると，

$$\sigma_\omega;\ (x, y) \mapsto (x + \omega, y + s_2 x^2 + s_1 x + s_0)$$

は曲線 C の自己同型である．ここで，基礎体 $\mathbb{F}_{2^m}$ 上の拡大体 $\mathbb{F}_{2^{12m}}$ を，

$$\mathbb{F}_{2^{6m}} \simeq \mathbb{F}_{2^m}[x]/(x^6 + x^5 + x^3 + x^2 + 1)$$

として 6 次拡大体を構成し，その上に

$$\mathbb{F}_{2^{12m}} \simeq \mathbb{F}_{2^{6m}}[y]/(y^2+y+w^5+w^3), \quad w^6+w^5+w^3+w^2+1=0$$

として 12 次拡大体を構成し，s_0 を $y^2+y+w^5+w^3$ の根として，多項式基底

$$\{1, w, w^2, w^3, w^4, w^5, s_0, s_0w, s_0w^2, s_0w^3, s_0w^4, s_0w^5\}$$

を用いて表す．この拡大体の構成に対し，先の自己同型 σ_ω について，$\omega = w$ として拡大体 $\mathbb{F}_{2^{12m}}$ 上で

$$\psi(x,y) = (x+w, y+s_2x^2+s_1x+s_0), \quad s_1 = w^4+w^2,\ s_2 = w^8+w^4+w = w^4+1$$

定義し，$C(\mathbb{F}_{2^{12m}})$ 上の自己同型 ψ を distortion map として用いる．distortion map により，C の定義体 $\mathbb{F}_{2^m}$ の元を 12 次拡大体 $\mathbb{F}_{2^{12m}}$ へ写し，拡大体上でペアリングの値を計算する．

曲線 C 上の η_T ペアリングでは，因子計算において 8 倍算を用いると効率が良い．更に，ヤコビ多様体 Jac_C の位数等から η_T ペアリングのパラメタを次の様に定めると効率的な η_T ペアリングが構成できる．η_T ペアリングのパラメタについては第 2.2.2 節に詳細を述べた．

C 上の自己同型

$$\varphi(x,y) = (x+1, y+x^2+1) \tag{5}$$

に対し，$q = 2^{3m}$ とすると因子 D に対し，

$$[q]D = [2^{3m}]D = \varphi^m(D)$$

が成り立つ．そこで Barreto らは $q = 2^{3m}$, $\gamma = \varphi^m$ として η_T ペアリングのパラメタを設定している（$\mathbb{F}_{2^m}$ ではなく $\mathbb{F}_{2^{3m}}$ 上で考え，8 倍算を基本演算とする）．具体的には，与えられた distortion map ψ に対し，η_T ペアリングの構成に必要な条件として第 2.2.2 節において述べた [5, §4, Theorem 1] の条件 3,

$$\gamma\psi^q(Q) = \psi(Q), \quad Q \in C(\mathbb{F}_q) \tag{6}$$

を満たす必要があるが，実際に自己同型 γ はこの条件を満たす．又，今 $m \equiv 7 \pmod{24}$ を仮定すると（本実装では $m = 103, 511$ を取り，何れもこの仮定を満た

す)，Jac_C の位数が $N = 2^{2m} + 2^{(3m+1)/2} + 2^m + 2^{(m+1)/2} + 1$ で，$T = -2^{(3m+1)/2} - 1$ とすると

$$[T]D = [q]D = \gamma(D) \tag{7}$$

を満たし，$L = 2^{m+1} - 2^{(m+3)/2} + 2$ として

$$(\eta_T(D, D')^M)^{2T} = (\langle D, \psi(D')\rangle_N^M)^L$$

が成り立ち，上記のパラメタの Tate ペアリングとして η_T ペアリングが定義できる．詳細は，[5, §7.2] を参照されたい．特に，このとき $T = -2^{(3m+1)/2} - 1$ より，ペアリング計算の Miller loop において，8 倍算を各ステップで行うためそのループ長は $m/2$ 程度であり，ループ長の大幅な削減に成功している．

3.2 η_T ペアリングのアルゴリズム

本節では，[5, §7, Appendix A,B] に従って，本研究で取り扱う η_T ペアリングのアルゴリズムを具体的に記述する．特に，[5] においては基礎体の拡大次数 m に関し，一般の場合の記述はない（大きな変更はない）ため，本論文では一般的な形でアルゴリズムを書いておく．まず，第 3.2.1 節において，η_T ペアリングのアルゴリズムを概観する．アルゴリズムのそれぞれの箇所の詳細な計算方法は，続く第 3.2.2, 3.2.3 節で行い，ここではペアリング計算の流れを大きく追って説明する．最後に，第 3.2.4 節においてアルゴリムの流れ，詳細な形を擬似コードによって示す．

3.2.1 η_T ペアリングのアルゴリズムの概要

この節では，上で述べてきた，η_T ペアリングのアルゴリズムを概観する．

まず第 3.1 節で述べた T に関する式について，因子 $-D$ に対し，$-T$ が対応するので $T > 0$ としてよい．即ち，$T = 2^{(3m+1)/2} + 1$ として考える．初めに，アルゴリズムの中核を成す 8 倍算を行う式，octupling formula について詳しく説明する．

Barreto らは [5, §7.1] において，因子 $D = (P) - (\infty)$ についての octupling formula を

$$8D = \varphi\pi^6(P) - (\infty), \quad \varphi(x, y) = (x + 1, y + x^2 + 1),\ \varphi \in \mathrm{Aut}(C)$$

とし，$[8]P := \varphi\pi^6(P)$，π: 2-Frobenius map という記号を用いて表している．ここで，実際に因子 $D(P) - (\infty)$ の 8 倍を計算するために，$8D$ に対応する有理関数 $f_{8,P}(x, y)$ を求める必要があるが，$f_{8,P} = 8(P) - ([8]P) - 7(\infty)$ より

$$f_{8,P}(x, y) = \frac{(y + b_4(x))^2(y + b_8''(x))}{a_4'(x)^2 a_8'(x)} = \left(\frac{y + b_4(x)}{a_4'(x)}\right)^2 \frac{y + b_8''(x)}{a_8'(x)}$$

と書ける．ここで，

$$b_4(x) = x^3 + (x_P^8 + x_P^4)x^2 + x_P^4 x + y_P^4$$
$$b_8''(x) = (x_P^{32} + 1)x + {}^2 + (x_P^{32} + x_P^{16})x + (y_P^{16} + x_P^{16} + x_P^{48} + 1)$$

であり，$P = (x_P, y_P)$ である．又，分母消去の理論より，a_4', a_8' の計算は省略できる．詳細は [5, Appendix A] を参照されたい．ペアリング計算では，この octupling formula に distortion map ψ を合成した函数を Miller loop の各 step で計算し，最終的に冪乗 $f_{T,P} \circ \psi$ を求める．各ステップでは

$$\alpha = (y + b_4(x))^2 \circ \psi, \quad \beta = y + b_8''(x) \circ \psi$$

として，$f_{8,P} \circ \psi = \alpha\beta$ を計算する．更に，$T = 2^{(3m+1)/2} + 1 = 2^{3(m-1)/2+2} + 1$ より，$m - 1$ 回の 8 倍算と 2 回の 2 倍算，最後に一回の加算により $f_{T,P} \circ \psi$ が計算される．即ち，

$$f_{T,P}(\psi(Q)) = \prod_{i=0}^{(m-3)/2} f_{8,[2^{3i}]P}(\psi(Q))^{2^{3(m-3-2i)/2}} l_1(\psi(Q))^2 l_2(\psi(Q)) l_3(\psi(Q))$$

を計算すれば良い（l_1, l_2は2倍算, l_3は加算から来る函数）．又，後ろの2乗算と加算を final doublings / addition と表す．更にペアリング計算の最後のフェーズとして最終冪を行う必要があるが，今回のパラメタでは $M = (q^k - 1)/N = (2^{12m} - 1)/N$ が

$$M = (2^{6m} - 1)(2^{3m} - 2^{(3m+1)/2} + 1)(-2^{(3m+1)/2} - 1) = (2^{6m} - 1)(2^{3m} - 2^{4m}2^{(m+1)/2} - 1)$$

の形で表され，final doublings / addition による結果の f に対し，

$$f^{(2^{6m}-1)(2^{3m}-2^{(3m+1)/2}+1)}$$

を計算することになる．

η_T ペアリングのアルゴリズムは以下の流れになる．

1. C 上の点 P, Q に対し Miller loop の各ステップで 8 倍算 $f_{8,P}(\psi(Q)) = \alpha\beta$ を計算していく．これは拡大体 $\mathbb{F}_{2^{12m}}$ 上で行われる．
2. Miller loop により計算された $\prod_{i=0}^{(m-3)/2} f_{8,[2^{3i}]P}(\psi(Q))^{2^{3(m-3-2i)/2}}$ に対し final doublings / addition を適用し，f を得る．
3. 最終冪として，$f^{(2^{6m}-1)(2^{3m}-2^{(3m+1)/2}+1)}$ を計算する．

3.2.2 Miller loop

本節では，ペアリングアルゴリズムの中核を成す Miller loop の計算について詳細を述べる．本研究で取り扱っている η_T ペアリングについては，事前計算を行うことにより，ループ内で効率的に因子の 8 倍算が行える．ここではその事前計算と，その結果を用いた第 3.2.1 節で述べた α, β の具体的な計算方法について子細に述べる．更に，[5] では，簡単のため基礎体の拡大次数 m について $m = 3 \pmod 4$ の場合の記述のみに留まっているので，本論文では任意の m におけるアルゴリズムを場合分けを用いて完全に記す．

事前計算

α, β を計算する際，$P = (x_P, y_P)$ に 2-power Frobenius map π を複数回作用させたものを事前に計算し，table を作っておくと効率的である．即ち，

$$x_P^{(i)} := \pi^i(x_P) = x_P^{2^i}, \quad y_P^{(i)} := \pi^i(y_P) = y_P^{2^i} \ (i = 0, 1, ..., m-1)$$

を計算しておく．因子 $D = (P) - (\infty)$ の octupling formula

$$8D = \varphi\pi^6(P) - (\infty), \quad [8]P := \varphi\pi^6(P)$$

について，

$$\gamma_1(i) = \begin{cases} 1 & \text{if } i\text{: odd,} \\ 0 & \text{if } i\text{: even,} \end{cases} \qquad \gamma_3(i) = \begin{cases} 1 & \text{if } i \equiv 1, 2 \pmod 4, \\ 0 & \text{otherwise} \end{cases}$$

とすると，$P = (x_P, y_P)$ に対し，

$$\varphi^i \pi^{6^i}(P) = (x_P^{(6i)} + \gamma_1(i), y_P^{(6i)} + \gamma_1(i) x_P^{(6i+1)} + \gamma_3(i))$$

を得る．ここで，$6i \pmod m$ として計算する．

α の計算

$\alpha = (y + b_4(x))^2 \circ \psi$ の具体的な形は以下の様になる ([5, Appendix B.2]).

$$(y+b_4(x))\circ\psi = y+s_2x^2+s_1x+s_0+(x+w)^3+(x_P^8+x_P^4)(x+w)^2+x_P^4(x+w)+y_P^4$$

より，

$$\begin{aligned} \alpha &= (y + s_2x^2 + s_1x + s_0 + (x+w)^3 + (x_P^8 + x_P^4)(x+w)^2 + x_P^4(x+w) + y_P^4)^2 \\ &= y^2 + wx^4 + (w^4 + w + 1)x^2 + s_0 + w^5 + w^3 + x^6 + w^2x^4 + w^4x^2 \\ &\quad + w^5 + w^3 + w^2 + 1 + (x_P^{16} + x_P^8)(x^4 + w^4) + x_P^8(x^2 + w^2) + y_P^8 \end{aligned}$$

となる．整理して多項式基底

$$\{1, w, w^2, w^3, w^4, w^5, s_0, s_0w, s_0w^2, s_0w^3, s_0w^4, s_0w^5\}$$

を用いて表現すると

$$\begin{aligned} &(y^2 + x^6 + (x_P^{16} + x_P^8)x^4 + (x_P^8 + 1)x^2 + y_P^8 + 1, \\ &\qquad x^4 + x^2, x^4 + x_P^8 + 1, 0, x_P^{16} + x_P^8, 0, 1, 0, 0, 0, 0, 0) \end{aligned}$$

となる．

β の計算

α と同様にして，$\beta = (y + b_8'') \circ \psi$ の具体的な式は以下の様にかける．

$$\begin{aligned}
\beta &= (y + b_8'') \circ \psi \\
&= y + s_2x^2 + s_1x + s_0 + (x_P^{32} + 1)(x + w)^2 + (x_P^{32} + x_P^{16})(x + w) + y_P^{16} + x_P^{16} + x_P^{48} + 1 \\
&= y + (w^4 + 1)x^2 + (w^4 + w^2)x + s_0 + (x_P^{32} + 1)(x^2 + w^2) \\
&\quad + (x_P^{32} + x_P^{16})(x + w) + y_P^{16} + x_P^{16} + x_P^{48} + 1
\end{aligned}$$

より，整理して

$$\begin{aligned}
(y + x_P^{32}x^2 + (x_P^{32} + x_P^{16})x + y_P^{16} + x_P^{16}(x_P^{32} + 1), \\
x_P^{32} + x_P^{16}, x + x_P^{32} + 1, 0, x^2 + x, 0, 1, 0, 0, 0, 0, 0, 0)
\end{aligned}$$

を得る．

以上より $Q = (x_Q, y_Q) \in \mathrm{Jac}_C(\mathbb{F}_{2^m})$ に対し，$f_{8,[8^i]P}(\psi(Q))$ は α, β を x_Q, y_Q の函数と見て次の様に計算できる．即ち $\alpha_{[8^i]P}(x_Q, y_Q)$ のベクトル表示は最初の成分が

$$\begin{aligned}
y_Q^2 + x_Q^6 + (x_P^{(6i+4)} + x_P^{(6i+3)})x_Q^4 + (x_P^{(6i+3)} + \gamma_1(i) + 1)x_Q^2 \\
+ y_P^{(6i+3)} + \gamma_1(i)x_P^{(6i+4)}\gamma_3(i) + 1
\end{aligned}$$

で，残りの成分が

$$(x_Q^4 + x_Q^2, x_Q^4 + x_P^{(6i+3)} + \gamma_1(i) + 1, 0, x_P^{(6i+4)} + x_P^{(6i+3)}, 0, 1, 0, 0, 0, 0, 0)$$

となる．同様にして，$\beta_{[8^i]P}(x_Q, y_Q)$ のベクトル表示の最初の成分は

$$\begin{aligned}
y_Q + (x_P^{(6i+5)} + \gamma_1(i))x_Q^2 + (x_P^{(6i+5)} + x_P^{(6i+4)} + 1)x_Q \\
+ y_P^{(6i+4)} + \gamma_1(i)x_P^{(6i+5)} + \gamma_3(i) + (x_P^{(6i+4)} + \gamma_1(i))(x_P^{(6i+5)} + x_P^{(6i+4)} + 1)
\end{aligned}$$

$$\begin{aligned}
= y_Q + (x_P^{(6i+5)} + \gamma_1(i))x_Q^2 + (x_P^{(6i+5)} + x_P^{(6i+4)})x_Q \\
+ y_P^{(6i+4)} + x_P^{(6i+4)}(x_P^{(6i+5)} + \gamma_1(i) + 1) + \gamma_3(i) + 1
\end{aligned}$$

で，残りが

$$(x_P^{(6i+5)} + x_P^{(6i+4)}, x_Q + x_P^{(6i+5)} + \gamma_1(i) + 1, 0, x_Q^2 + x_Q, 0, 1, 0, 0, 0, 0, 0)$$

とかける．

α, β の基本的な形は上の式で表され，Miller loop の各ステップではこの式を用いてそれぞれ α, β の冪を計算する．即ち，[5, Appendix B.4] にある様に，Miller's algorithm のメインの計算箇所である

$$\prod_{i=0}^{(m-3)/2} f_{8,[2^{3i}]P}(\psi(Q))^{2^{3(m-3-2i)/2}}$$

について，先の α, β の冪乗を計算し，

$$\prod_{i=0}^{(m-3)/2} f_i$$

と表す．即ち，

$$f_i = \alpha^{2^{3(m-3-2i)/2}} \cdot \beta^{2^{3(m-3-2i)/2}}$$

を計算する．

まず基底の元の冪乗について，基底に入っている関係式により具体的な式を書き下す．

基底の元の冪乗

今，$\mathbb{F}_{2^{12m}}$ を構成している多項式基底

$$\{1, w, w^2, w^3, w^4, w^5, s_0, s_0 w, s_0 w^2, s_0 w^3, s_0 w^4, s_0 w^5\}$$

について，その基底を成す元の冪乗の式を [5, Appendix B.4] の解説にある様に具体的に書き下す．ここでは $m \equiv 3 \pmod 4$ の場合だけでなく，$m \equiv 1 \pmod 4$ の場合についても，冪乗の形を求めることにより，正確にペアリングアルゴリズムの詳細を述べておく．

今，$w^8 = w + 1$ より $w^{8^i} = w + \gamma_1(i)$ である．$m \equiv 3 \pmod 4$ のとき，[5, Appendix B.4] にある様に，$(m, 12) = 1$ より m は奇数で，

$$(m - 3 - 2i)/2 \equiv i \pmod 2$$

でありそれぞれ $w^{2^{3(m-3-2i)/2}}$ 等の式は [5, Appendix B.4] に述べられている通りである．更に $m \equiv 1 \pmod 4$ の場合も考えると，

$$w^{2^{3(m-3-2i)/2}} = \begin{cases} w + \gamma_1(i+1) & \text{if } m \equiv 1 \pmod 4, \\ w + \gamma_1(i) & \text{if } m \equiv 3 \pmod 4 \end{cases}$$

であり，従って，

$$(w^2)^{2^{3(m-3-2i)/2}} = \begin{cases} w^2 + \gamma_1(i+1) & \text{if } m \equiv 1 \pmod 4, \\ w^2 + \gamma_1(i) & \text{if } m \equiv 3 \pmod 4, \end{cases}$$

$$(w^4)^{2^{3(m-3-2i)/2}} = \begin{cases} w^4 + \gamma_1(i+1) & \text{if } m \equiv 1 \pmod 4, \\ w^4 + \gamma_1(i) & \text{if } m \equiv 3 \pmod 4 \end{cases}$$

となる．又，このとき

$$\begin{aligned} s_0^8 &= s_0 + w^2 \\ s_0^{8^2} &= s_0 + 1 \\ s_0^{8^3} &= s_0 + w^2 + 1 \end{aligned}$$

より

$$(m-3-2i)/2 \equiv \begin{cases} \begin{cases} 3 & \text{if } i \equiv 0 \pmod 4, \\ 2 & \text{if } i \equiv 1 \pmod 4, \\ 1 & \text{if } i \equiv 2 \pmod 4, \\ 0 & \text{if } i \equiv 3 \pmod 4, \end{cases} & \text{if } m \equiv 1 \pmod 8, \\ \begin{cases} 0 & \text{if } i \equiv 0 \pmod 4, \\ 3 & \text{if } i \equiv 1 \pmod 4, \\ 2 & \text{if } i \equiv 2 \pmod 4, \\ 1 & \text{if } i \equiv 3 \pmod 4, \end{cases} & \text{if } m \equiv 3 \pmod 8, \\ \begin{cases} 1 & \text{if } i \equiv 0 \pmod 4, \\ 0 & \text{if } i \equiv 1 \pmod 4, \\ 3 & \text{if } i \equiv 2 \pmod 4, \\ 2 & \text{if } i \equiv 3 \pmod 4, \end{cases} & \text{if } m \equiv 5 \pmod 8, \\ \begin{cases} 2 & \text{if } i \equiv 0 \pmod 4, \\ 1 & \text{if } i \equiv 1 \pmod 4, \\ 0 & \text{if } i \equiv 2 \pmod 4, \\ 3 & \text{if } i \equiv 3 \pmod 4, \end{cases} & \text{if } m \equiv 7 \pmod 8 \end{cases} \pmod 4$$

だから,

$$s_0^{2^{3(m-3-2i)/2}} = \begin{cases} s_0 + \gamma_1(i+1)w^2 + \gamma_3(i+1) & \text{if } m \equiv 1 \pmod 8, \\ s_0 + \gamma_1(i)w^2 + \gamma_3(i) & \text{if } m \equiv 3 \pmod 8, \\ s_0 + \gamma_1(i+1)w^2 + \gamma_3(i-1) & \text{if } m \equiv 5 \pmod 8, \\ s_0 + \gamma_1(i)w^2 + \gamma_3(i) + 1 & \text{if } m \equiv 7 \pmod 8 \end{cases}$$

と表せる.

α の冪乗

上記の基底の元の冪乗における関係式を用いて $\alpha^{2^{3(m-3-2i)/2}}$ の具体的な表示を以下の様に得る．まず記号として

$$\alpha_{0,i} := y_Q^2 + x_Q^6 + (x_P^{(6i+4)} + x_P^{(6i+3)})x_Q^4 + (x_P^{(6i+3)} + 1 + \gamma_1(i))x_Q^2 + y_P^{(6i+3)} + \gamma_1(i)x_P^{(6i+4)} + \gamma_3(i) + 1,$$

$$\alpha_i := \alpha_{[8^iP]} = (\alpha_{0,i}, x_Q^4 + x_Q^2, x_Q^4 + x_P^{(6i+3)} + \gamma_1(i) + 1, 0, x_P^{(6i+4)} + x_P^{(6i+3)}, 0, 1, 0, 0, 0, 0, 0),$$

を定める．下付き文字の i は Miller loop の各ステップにおける α 等の元を表す意味を持っている．又，x に i 回 Frobenius map を適用したものを $x^{(i)} = x^{2^i}$ という記号で表す．

$\alpha_i^{2^{3(m-3-2i)/2}}$ のベクトル表示について考える．$\alpha_{0,i}$ の初めの成分を $2^{3(m-3-2i)/2}$ 乗すると

$$y_Q^{((3m-7-6i)/2)} + (x_Q^{((3m-7-6i)/2)})^3 + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)})x_Q^{((3m-5-6i)/2)} + (x_P^{((3m-3+6i)/2)} + 1 + \gamma_1(i))x_Q^{((3m-7-6i)/2)} + y_P^{((3m-3+6i)/2)} + \gamma_1(i)x_P^{((3m-1+6i)/2)} + \gamma_3(i) + 1$$

が得られる．又，w, w^2, w^4, s_0 の $2^{3(m-3-2i)/2}$ 乗による 1 の成分として

$$\begin{cases} \gamma_1(i+1)(x_Q^{((3m-7-6i)/2)} + \gamma_1(i) + 1 + x_P^{((3m-1+6i)/2)}) + \gamma_3(i+1) & \text{if } m \equiv 1 \pmod 8, \\ \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + \gamma_1(i) + 1 + x_P^{((3m-1+6i)/2)}) + \gamma_3(i) & \text{if } m \equiv 3 \pmod 8, \\ \gamma_1(i+1)(x_Q^{((3m-7-6i)/2)} + \gamma_1(i) + 1 + x_P^{((3m-1+6i)/2)}) + \gamma_3(i-1) & \text{if } m \equiv 5 \pmod 8, \\ \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + \gamma_1(i) + 1 + x_P^{((3m-1+6i)/2)}) + \gamma_3(i) + 1 & \text{if } m \equiv 7 \pmod 8 \end{cases}$$

を得るので，この式と上の式を足し合わせたものが $\alpha^{2^{3(m-3-2i)/2}}$ の 1 の成分とな

る．更に，$\alpha^{2^{3(m-3-2i)/2}}$ の残りの項は，

$$\begin{cases}(x_Q^{((3m-5-6i)/2)}+x_Q^{((3m-7-6i)/2)})w\\ +(x_Q^{((3m-5-6i)/2)}+x_P^{((3m-3+6i)/2)}+\gamma_1(i)+\gamma_1(i+1)+1)w^2 & \text{if } m\equiv 1 \pmod 4\\ +(x_P^{((3m-1+6i)/2)}+x_P^{((3m-3+6i)/2)})w^4+s_0,\\ (x_Q^{((3m-5-6i)/2)}+x_Q^{((3m-7-6i)/2)})w\\ +(x_Q^{((3m-5-6i)/2)}+x_P^{((3m-3+6i)/2)}+\gamma_1(i)+\gamma_1(i)+1)w^2 & \text{if } m\equiv 3 \pmod 4\\ +(x_P^{((3m-1+6i)/2)}+x_P^{((3m-3+6i)/2)})w^4+s_0\end{cases}$$

$$=\begin{cases}(x_Q^{((3m-5-6i)/2)}+x_Q^{((3m-7-6i)/2)})w\\ +(x_Q^{((3m-5-6i)/2)}+x_P^{((3m-3+6i)/2)})w^2 & \text{if } m\equiv 1 \pmod 4\\ +(x_P^{((3m-1+6i)/2)}+x_P^{((3m-3+6i)/2)})w^4+s_0,\\ (x_Q^{((3m-5-6i)/2)}+x_Q^{((3m-7-6i)/2)})w\\ +(x_Q^{((3m-5-6i)/2)}+x_P^{((3m-3+6i)/2)}+1)w^2 & \text{if } m\equiv 3 \pmod 4\\ +(x_P^{((3m-1+6i)/2)}+x_P^{((3m-3+6i)/2)})w^4+s_0\end{cases}$$

となる．

β の冪乗

α のときと同様にして，$\beta_i^{2^{3(m-3-2i)/2}}$ のベクトル表示を計算する．

$$\beta_{0,i}:=y_Q+(x_P^{(6i+5)}+\gamma_1(i))x_Q^2+(x_P^{(6i+5)}+x_P^{(6i+4)})x_Q+y_P^{(6i+4)}$$
$$+x_P^{(6i+4)}(x_P^{(6i+5)}+\gamma_1(i)+1)+\gamma_3(i)+1,$$

$$\beta_i:=\beta_{[8^iP]}=(\beta_{0,i},x_P^{(6i+5)}+x_P^{(6i+4)},x_Q+x_P^{(6i+5)}+\gamma_1(i)+1,0,x_Q^2+x_Q,0,1,0,0,0,0,0,0)$$

$\beta_{0,i}$ を $2^{3(m-3-2i)/2}$ 乗すると

$$y_Q^{((3m-9-6i)/2)}+(x_P^{((3m+1+6i)/2)}+\gamma_1(i))x_Q^{((3m-7-6i)/2)}$$
$$+(x_P^{((3m+1+6i)/2)}+x_P^{((3m-1+6i)/2)})x_Q^{((3m-9-6i)/2)}+y_P^{((3m-1+6i)/2)}$$
$$+x_P^{((3m-1+6i)/2)}(x_P^{((3m+1+6i)/2)}+\gamma_1(i)+1)+\gamma_3(i)+1$$

が得られ，これに

$$\begin{cases} \gamma_1(i+1)(x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + x_Q^{((3m-7-6i)/2)}) + \gamma_3(i+1) & \text{if } m \equiv 1 \pmod 8, \\ \gamma_1(i)(x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + x_Q^{((3m-7-6i)/2)}) + \gamma_3(i) & \text{if } m \equiv 3 \pmod 8, \\ \gamma_1(i+1)(x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + x_Q^{((3m-7-6i)/2)}) + \gamma_3(i-1) & \text{if } m \equiv 5 \pmod 8, \\ \gamma_1(i)(x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + x_Q^{((3m-7-6i)/2)}) + \gamma_3(i) + 1 & \text{if } m \equiv 7 \pmod 8 \end{cases}$$

を足し合わせたものが $\beta^{2^{3(m-3-2i)/2}}$ の 1 の成分となる．又，$\beta^{2^{3(m-3-2i)/2}}$ の残りの項は，

$$\begin{cases} (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})w \\ +(x_Q^{((3m-9-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1 + \gamma_1(i+1))w^2 & \text{if } m \equiv 1 \pmod 4 \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)})w^4 + s_0, \\ (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})w \\ +(x_Q^{((3m-9-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1 + \gamma_1(i))w^2 & \text{if } m \equiv 3 \pmod 4 \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)})w^4 + s_0 \end{cases}$$

$$= \begin{cases} (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})w \\ +(x_Q^{((3m-9-6i)/2)} + x_P^{((3m+1+6i)/2)})w^2 & \text{if } m \equiv 1 \pmod 4 \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)})w^4 + s_0, \\ (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})w \\ +(x_Q^{((3m-9-6i)/2)} + x_P^{((3m+1+6i)/2)} + 1)w^2 & \text{if } m \equiv 3 \pmod 4 \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)})w^4 + s_0 \end{cases}$$

となる．

以上により，Miller loop で計算される

$$\prod_{i=0}^{(m-3)/2} f_{8,[2^{3i}]P}(\psi(Q))^{2^{3(m-3-2i)/2}}$$

について，具体的に

$$\prod_{i=0}^{(m-3)/2} f_i = \prod_{i=0}^{(m-3)/2} \alpha_i^{2^{3(m-3-2i)/2}} \cdot \beta_i^{2^{3(m-3-2i)/2}}$$

として計算が行える，

3.2.3 final doublings / addition と最終冪

ここでは，final doublings / addition と最終冪で行われる拡大体上の演算について，特にその演算回数を述べておく．詳細は [5, § 7.5, Appendix B.7] を参照されたい．

final doublings / addition

η_T ペアリングのアルゴリズムについて，$(3m+1)/2 = 3(m-1)/2+2$ より $(m-1)/2$ 回の 8 乗算後，残りの二回の 2 乗算と一回の足し算を行う必要がある．[5, Appendix B.7] にある様に，今回の case では degenerate divisor として因子計算を行えるので最後の足し算を省略できる．よって，二回の 2 乗算，即ち一度 4 乗算を行えば良い．因子の 4 乗算は以下の式

$$\begin{aligned}(y+b_4(x))\circ\psi &= y+t_2x^2+t_1x+t+(x+s)^3+(x_P^8+x_P^4)(x+s)^2+x_P^4(x+s)+y_P^4\\ &= (y+(x_P^8+x_P^4+x+1)x^2+x_P^4x+y_P^4+x_P^8+x_P^4+1,\\ &\qquad x^2+x+x_P^8+x_P^4+1, x^2+x, x_P^8+x, 1, 0, 1, 0, 0, 0, 0, 0)\end{aligned}$$

で表される．

最終冪

詳細は [5, § 7.5] に述べられているが，今，拡大体 $\mathbb{F}_{2^{12m}}$ 上で $z^{2^{6m}+1}=1$ より，$z^{2^{6m}}=z^{-1}$ が成り立つため一度の逆元算により 2^{6m} 乗の計算が行える．よって最終冪

$$f^{(2^{6m}-1)(2^{3m}-2^{4m}2^{(m+1)/2}-1)}$$

は，$(m+1)/2$ 回の 2 乗算，4 回の Frobenius map の適用，2 回の乗算，1 回の逆元算により計算が完了する．

3.2.4 アルゴリズムの詳細

以上より，事前計算から最終冪まで，具体的に計算式が得られているので，η_T ペアリングのアルゴリズムの詳細を $m \equiv 3 \pmod 4$ の場合において，Algorithm

3 ([5, §7.3, Algorithm4]) に示す．Step 1 が η_T ペアリングの事前計算の箇所に当り，ここで各 Frobenius map による作用に対する計算結果の表を作成し，Miller loop の各ステップでそれらの値を用いる．Step 3 から step 21 は Miller loop であり，各ステップで α (step 9-12), β (step 15-18) の計算を行い，step 20 において乗算 $\alpha\beta$ を計算し，それ以前の計算結果に $\alpha\beta$ を掛けていく．最後に step 28 において final doublings / addition に掛かる二度の二乗算，step 31 において最終冪を求めることになる．

Algorithm 3 $\eta_T$pairing ($m \equiv 3 \pmod 4$) [5, §7.3, Algorithm4]

Input: $P = (x_P, y_P),\ Q = (x_Q, y_Q) \in \mathrm{Jac}_{C_2}(\mathbb{F}_{2^{103}})$

Output: $f \in \mathbb{F}_{2^{12m}}$

$x_1[i] \leftarrow x_P^{2^i},\ y_1[i] \leftarrow y_P^{2^i},\ x_2[i] \leftarrow x_Q^{2^i},\ y_2[i] \leftarrow y_Q^{2^i}\ (0 \le i \le m-1)$
$f \leftarrow 1$
for $i = 0$ TO $(m-3)/2$ **do**
  以下，k_j は $\pmod m$ で考える．
  $k_1 \leftarrow (3m-9-6i)/2,\ k_2 \leftarrow k_1+1,\ k_3 \leftarrow k_2+1$
  $k_4 \leftarrow (3m-3-6i)/2,\ k_5 \leftarrow k_4+1,\ k_6 \leftarrow k_5+1$

  Compute $\alpha \leftarrow a + bw + cw^2 + dw^4 + s_0$
  $d \leftarrow x_1[k_4] + x_1[k_5]$
  $a \leftarrow y_2[k_2] + (x_1[k_4] + 1 + x_2[k_3]) \cdot x_2[k_2] + dx_2[k_3] + y_1[k_4]$
  $b \leftarrow x_2[k_3] + x_2[k_2]$
  $c \leftarrow x_2[k_3] + x_1[k_4] + 1$

  Compute $\beta \leftarrow e + f_2 w + gw^2 + hw^4 + s_0$
  $f_2 \leftarrow x_1[k_5] + x_1[k_6]$
  $e \leftarrow y_2[k_1] + f_2 x_2[k_1] + y_1[k_5] + x_1[k_6] \cdot (x_1[k_5] + x_2[k_2]) + x_1[k_5]$
  $g \leftarrow x_2[k_1] + x_1[k_6] + 1$
  $h \leftarrow x_2[k_2] + x_2[k_1] + 1$

  $f \leftarrow f \cdot (\alpha \cdot \beta)$
end for

$x_P \leftarrow x_1[100] + 1$
$y_P \leftarrow y_1[100] + x_1[101]$

Perform the final doublings / addition
$t \leftarrow y_2[0] + x_2[1] \cdot (1 + x_2[0] + x_P^8 + x_P^4) + x_P^4 x_2[0] + y_P^4$
$f \leftarrow f^4 \cdot (t, x_2[1] + x_P^4, x_P^8 + x_P^4, 1, x_2[1] + x_2[0], 0, 1, 0, 0, 0, 0, 0)$

Perform the final exponentiation
$f \leftarrow f^{(2^{6m}-1)(2^{3m}-2^{4m}2^{(m+1)/2}-1)}$
return f

3.3 η_T ペアリングにおける拡大体上の乗算の計算コスト

この節では，第 3.2 節で述べた η_T ペアリングの具体的なアルゴリズムにおいて，特に Miller loop の各ステップで行われる拡大体上の乗算 $\alpha\beta$ の計算コストについて，基礎体の算術，又は拡大体上での Karatsuba method を用いた乗算等のコストを用いて評価する．

本研究では，拡大体 $\mathbb{F}_{2^{12m}}$ の構成を後の第 5.1.4 節で述べる様に，第 3.1 節で述べた 6 次拡大から始める構成とは別の構成を考え，その際，乗算 $\alpha\beta$ がより効率的に行えることを示す．従って，ここではその計算コストの詳細を述べ，第 5.1.4 節において，計算コストの比較を行う．具体的にコストを評価する指標として以下のものを用いる．

- $\boldsymbol{A}$: 基礎体 $\mathbb{F}_{2^m}$ における加算一回当りのコスト
- $\boldsymbol{M}$: 基礎体 $\mathbb{F}_{2^m}$ における乗算一回当りのコスト
- $\boldsymbol{A}_{\mathbf{ext}}$: 拡大体 $\mathbb{F}_{2^{km}}$ における加算一回当りのコスト．k は計算により 3, 或いは 6 を取る．
- $\boldsymbol{K3}$: 3 次の Karatsuba method による乗算一回あたりのコスト．即ち，3 次拡大体の元を Karatsuba method を用いて掛け合わせ，mod 演算を省略した処理に掛かるコストである．

本研究の並列実装により，拡大体の加算コスト A_{ext} は，単に基礎体の加算に掛かる実行時間に比例するわけではなく，$3, 6, 12$ 次程度の拡大次数では，基礎体と同等の時間で処理できる．従って，拡大体の加算コストも用いて $\alpha\beta$ の計算コストを評価する．

第 3.1 節で述べた様に，[5, §7] の通りに拡大体 $\mathbb{F}_{2^{12m}}$ を構成する．即ち，$\mathbb{F}_{2^{6m}} \simeq \mathbb{F}_{2^m}[x]/(x^6+x^5+x^3+x^2+1)$ として，$\mathbb{F}_{2^{12m}} \simeq \mathbb{F}_{2^{6m}}[y]/(y^2+y+w^5+w^3),\ w^6+w^5+w^3+w^2+1=0$ として構成され，s_0 を $y^2+y+w^5+w^3$ の根とすると，基底

$$\{1, w, w^2, w^3, w^4, w^5, s_0, s_0w, s_0w^2, s_0w^3, s_0w^4, s_0w^5\}$$

を用いて表される．この基底を $\boldsymbol{s_0w}$**-basis** と呼び，拡大体の元 $a \in \mathbb{F}_{2^{12m}}$ について s_0w-basis を用いて表現したものを a_{s_0w} とかく．

α, β の具体的な式は第 3.2.2 節で述べたが，ここに再掲する．

$$\alpha_{0,i} := y_Q^2 + x_Q^6 + (x_P^{(6i+4)} + x_P^{(6i+3)})x_Q^4 + (x_P^{(6i+3)} + 1 + \gamma_1(i))x_Q^2 + y_P^{(6i+3)} + \gamma_1(i)x_P^{(6i+4)} + \gamma_3(i) + 1,$$

$$\alpha_i := \alpha_{[8^iP]} = (\alpha_{0,i}, x_Q^4 + x_Q^2, x_Q^4 + x_P^{(6i+3)} + \gamma_1(i) + 1, 0, x_P^{(6i+4)} + x_P^{(6i+3)}, 0, 1, 0, 0, 0, 0, 0, 0),$$

$$\beta_{0,i} := y_Q + (x_P^{(6i+5)} + \gamma_1(i))x_Q^2 + (x_P^{(6i+5)} + x_P^{(6i+4)})x_Q + y_P^{(6i+4)} + x_P^{(6i+4)}(x_P^{(6i+5)} + \gamma_1(i) + 1) + \gamma_3(i) + 1,$$

$$\beta_i := \beta_{[8^iP]} = (\beta_{0,i}, x_P^{(6i+5)} + x_P^{(6i+4)}, x_Q + x_P^{(6i+5)} + \gamma_1(i) + 1, 0, x_Q^2 + x_Q, 0, 1, 0, 0, 0, 0, 0, 0)$$

について，$\alpha_i^{2^{3(m-3-2i)/2}}, \beta_i^{2^{3(m-3-2i)/2}}$ は $m = 103$ のとき，以下の様にかける．即ち，$\alpha_i^{2^{3(m-3-2i)/2}}$ の定数項は

$$\begin{aligned} &y_Q^{((3m-7-6i)/2)} + (x_Q^{((3m-7-6i)/2)})^3 + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)})x_Q^{((3m-5-6i)/2)} \\ &+ (x_P^{((3m-3+6i)/2)} + 1 + \gamma_1(i))x_Q^{((3m-7-6i)/2)} + y_P^{((3m-3+6i)/2)} + \gamma_1(i)x_P^{((3m-1+6i)/2)} \\ &\quad + \gamma_3(i) + 1 + \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + \gamma_1(i) + 1 + x_P^{((3m-1+6i)/2)}) + \gamma_3(i) + 1 \end{aligned}$$

$$\begin{aligned} &= y_Q^{((3m-7-6i)/2)} + (x_Q^{((3m-7-6i)/2)})^3 + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)})x_Q^{((3m-5-6i)/2)} \\ &\quad + (x_P^{((3m-3+6i)/2)} + 1)x_Q^{((3m-7-6i)/2)} + y_P^{((3m-3+6i)/2)} \end{aligned}$$

で，残りの項は

$$(x_Q^{((3m-5-6i)/2)} + x_Q^{((3m-7-6i)/2)})w + (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-3+6i)/2)} + 1)w^2 + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)})w^4 + s_0$$

となる．又，$\beta_i^{2^{3(m-3-2i)/2}}$ の定数項は

$$
\begin{aligned}
&y_Q^{((3m-9-6i)/2)} + (x_P^{((3m+1+6i)/2)} + \gamma_1(i))x_Q^{((3m-7-6i)/2)} \\
&\qquad + (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})x_Q^{((3m-9-6i)/2)} \\
&\quad + y_P^{((3m-1+6i)/2)} + x_P^{((3m-1+6i)/2)}(x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i) + 1 \\
&\qquad + \gamma_1(i)(x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + x_Q^{((3m-7-6i)/2)}) + \gamma_3(i) + 1
\end{aligned}
$$

$$
\begin{aligned}
&= y_Q^{((3m-9-6i)/2)} + x_P^{((3m+1+6i)/2)}x_Q^{((3m-7-6i)/2)} \\
&\qquad + (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})x_Q^{((3m-9-6i)/2)} \\
&\qquad\qquad + y_P^{((3m-1+6i)/2)} + x_P^{((3m-1+6i)/2)}(x_P^{((3m+1+6i)/2)} + 1)
\end{aligned}
$$

で，残りの項は

$$
\begin{aligned}
&(x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)})w + (x_Q^{((3m-9-6i)/2)} + x_P^{((3m+1+6i)/2)} + 1)w^2 \\
&\qquad\qquad + (x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)})w^4 + s_0
\end{aligned}
$$

と表される．

以下，簡単のため $\alpha_i^{2^{3(m-3-2i)/2}}, \beta_i^{2^{3(m-3-2i)/2}}$ を α, β とかく．更に，α, β のそれぞれの各係数を

$$
\begin{aligned}
\alpha &= (a_0, a_1, a_2, 0, a_4, 0, 1, 0, 0, 0, 0, 0) = a_0 + a_1 w + a_2 w^2 + a_4 w^4 + s_0 \\
\beta &= (b_0, b_1, b_2, 0, b_4, 0, 1, 0, 0, 0, 0, 0) = b_0 + b_1 w + b_2 w^2 + b_4 w^4 + s_0
\end{aligned}
$$

(但し，$a_i, b_i \in \mathbb{F}_{2^m}$，$(i = 0, 1, 2, 4)$) と簡略化してかく．このとき，

$$
f = (a_0, a_1, a_2, 0, a_4, 0),\ g = (b_0, b_1, b_2, 0, b_4, 0)
$$

とすると，

$$
\alpha\beta = (f, 1)(g, 1) = (fg, f + g, 1)
$$

より前節と同様で，乗算については $\mathbb{F}_{2^{6m}}$ 上の積 fg に対して計算コストを考えれば良い．より効率的に計算を行うため，乗算 fg は 6 次の Karatsuba method では

なく 3 次の Karatsuba method を用いて処理する．実際，基底の元の内の w^3, w^5 の成分は 0 であるから，3 次の Karatsuba method のみを用いて以下のコストで計算できる．

$$f_0 = (a_0, a_1, a_2),\ f_1 = (0, a_4, 0)$$
$$g_0 = (b_0, b_1, b_2),\ g_1 = (0, b_4, 0)$$

と置き，

$$fg = f_1g_1 + (f_0g_1 + f_1g_0) + f_0g_0 \pmod{x^6 + x^5 + x^3 + x^2 + 1}$$

として考える．

- f_0g_0 の計算．3 次の Karatsuba method を用いて乗算を行うため，$K3$ のコスト（乗算に関しては $3M$ のコスト）が掛かる．又，f_0g_0 において w の最高次数は 4 なので mod 演算する必要はない ($f_0g_0 \in \mathbb{F}_{2^{6m}}$).

- $f_0g_1 + f_1g_0$ の計算．この状況では，$\mathbb{F}_{2^{6m}}$ 上で Karatsuba method を用いて乗算を行うより，直接 $f_0g_1 + f_1g_0$ として計算したほうがコストが小さい．$f_0g_1 = (a_0, a_1, a_2)(0, b_4, 0)$ に掛かる乗算コストは $3M$ で，$f_1g_0 = (0, a_4, 0)(b_0, b_1, b_2)$ も同様に $3M$ である．又，

$$\begin{aligned} f_0g_1 &= (a_0, a_1, a_2)(0, b_4, 0) = a_2b_4w^6 + a_1b_4w^5 + a_0b_4w^4 \\ f_1g_0 &= (0, a_4, 0)(b_0, b_1, b_2) = a_4b_2w^6 + a_4b_1w^5 + a_4b_0w^4 \end{aligned}$$

  だが mod 演算に掛かるコストは後にまとめて算出する．

- f_1g_1 の計算．$(0, a_4, 0)(0, b_4, 0) = a_4b_4w^8 = a_4b_4(w + 1)$ よりコスト M で計算できる．

今，$w^6 = w^5 + w^3 + w^2 + 1$ より，

$$\begin{aligned} f_0g_1 + f_1g_0 + f_1g_1 &= \{(a_1b_4 + a_4b_1)w^5 + (a_0b_4 + a_4b_0)w^4 + a_4b_4(w + 1)\} \\ &+ \{(a_2b_4 + a_4b_2)(w^5 + w^3 + w^2 + 1)\} \end{aligned} \tag{8}$$

であるから，コスト $3A+A_{\mathrm{ext}}$，或いは基礎体の加算回数を見て，コスト $3A+2A=5A$ で計算できる．更に式 (8) にコスト A_{ext}，或いはコスト $5A$ で f_0g_0 を加え，fg を得る．従って，fg はコスト

$$K3+3M+3M+M+5A+5A=K3+7M+10A$$

又は，A_{ext} を用いてコスト

$$K3+3M+3M+M+3A+A_{\mathrm{ext}}+A_{\mathrm{ext}}=K3+7M+2A_{\mathrm{ext}}+3A$$

で計算できる．

更に，

$$\alpha\beta=(f,1)(g,1)=(fg,f+g,1)$$

において，

$$f+g=(a_0+b_0,a_1+b_1,a_2+b_2,0,a_4+b_4,0)$$

はコスト $4A$ 又は，A_{ext} で計算でき，多項式 $\bmod y^2+y+w^5+w^3$ による剰余に掛かる計算コストは実質無視出来る．

従って，

$$\mathbb{F}_{2^{12m}}=\langle 1,w,w^2,w^3,w^4,w^5,s_0,s_0w,s_0w^2,s_0w^3,s_0w^4,s_0w^5\rangle$$

のとき $\alpha\beta$ は $K3+7M+14A$ 又は，$K3+7M+3A_{\mathrm{ext}}+3A$ で計算される．

4. 拡大体の基底変換に関する提案

本章では，有限体の基底変換について，基底変換の求解アルゴリズムに関して，normal basis への変換を用いることによる，従来の手法に比べて変換行列の探索時間を大幅に短縮できる効率的な手法を提案し，提案手法に関するシミュレーション結果を示す．又，基底変換の変換コストについて，シミュレーション結果を示し，考察を与える．ここで変換コストは，その変換行列の 0 でない成分の個数とし，変換に必要な基礎体の加算，又は乗算のコストを表すものとして定義した．更に，本研究における η_T ペアリングの並列実装において，提案手法である基底変換による拡大体上の乗算の効率化のために，実際に本研究で取り扱う，η_T ペアリングのパラメタである拡大体 $\mathbb{F}_{2^{12m}}$ における基底変換について述べる．特に，本実装において適用出来る基底変換の中でコスト最小の変換行列を明示する．

本研究では，拡大体上の乗算を基底変換を適用して高速に並列化することにより η_T ペアリングを効率的に並列実装した．本実装においては，η_T ペアリングが持つ超楕円曲線の定義体，又，その拡大体の元を多項式基底を用いて表現し，並列計算を行っている．従って，本章で述べる基底変換は，主に多項式基底間の変換について研究を行ったものである．又，η_T ペアリングのアルゴリズムにおいて，本実装では normal basis へ変換し Frobenius map の作用の計算を行うので，normal basis への変換についても本章において説明する．

まず，第 4.1 節において，基底変換に関する本研究におけるターゲットを明確に説明する．次に，第 4.2 節において，基底変換の求解に関する提案手法について具体的に述べ，その手法によるシミュレーション結果を示す．これは筆者らが [12] で行った研究内容である．最後に，第 4.3 節において，本研究で扱う η_T ペアリングに関して，提案手法である並列実装で必要となる拡大体 $\mathbb{F}_{2^{12m}}$ のコスト最小の基底変換を明示する．

4.1 本研究において対象とする基底変換の基礎

本節では，本研究の有限体の基底変換に関する研究対象について具体的に述べる．本研究では有限体の基底変換として，拡大体 $\mathbb{F}_{p^k}$ の多項式基底の間の基底変

換を対象とする．ここで，$\mathbb{F}_p$ は素体である．一般の拡大体でなく，素体上の単拡大体について考えているのは，$\mathbb{F}_{q^k},\ q = p^m$（$p$ は素数）が

$$\mathbb{F}_{q^k} \simeq \mathbb{F}_q[x]/(f(x))$$

として構成されている時，$f(x)$ が同時に $\mathbb{F}_p$ 上の既約多項式である場合（本研究の η_T ペアリングもこの場合）が少なくないからである．この時，$\mathbb{F}_{p^k} \simeq \mathbb{F}_p[x]/(f(x))$ において基底変換を考えれば変換行列の成分は素体の元が並び，変換の際の $\mathbb{F}_{q^k}$ 上の乗算コストを 0 にできる．

多項式基底間の基底変換

本研究では多項式基底の間の基底変換について考察する．その動機として，本研究で取り組む η_T ペアリングの並列実装において，曲線の定義体である基礎体の算術，又，ペアリングの計算で必要な拡大体上の算術の並列処理を多項式基底を用いて達成している点がある．まず，基底変換を求めるために次のことに注意する．$\mathbb{F}_{p^k}$ の二つの基底 $\mathcal{B}, \mathcal{B}'$ の間の基底変換について考える．$\mathbb{F}_{p^k}{}_{\mathcal{B}}$ は，自身とその中間体を構成する既約多項式の根を，基礎体に添加して得られ，その根が基底 $\mathcal{B}$ をなしている．この時，そのそれぞれの既約多項式の根を $\mathbb{F}_{p^k}{}_{\mathcal{B}'}$ 内で求めることにより，基底 $\mathcal{B}$ の元の $\mathcal{B}'$ の表現を得る．即ち，

$$\mathcal{B} = \{v_1, v_2, ..., v_k\}, \quad \mathcal{B}' = \{w_1, w_2, ..., w_k\}$$

として，例えば v_{i_j} が既約多項式の元のとき，

$$v_{i_j} = \sum_{l=1}^{k} \{p_{i_j}\}_l w_l$$

とすると $\mathcal{B}$ の他の元は，v_{i_j} を掛けあわせて得られるものだから，同様にそれらを $\mathcal{B}'$ で表して

$$v_{i'_j} = \sum_{l=1}^{k} \{p_{i'_j}\}_l w_l$$

となる．このとき $p_{i_j}, p_{i'_j}$ をそれぞれ i_j, i'_j 列に並べた行列を P とすると

$$\begin{pmatrix} v_1 & v_2 & \cdots & v_k \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & \cdots & w_k \end{pmatrix} P$$

となり，基底変換 P が定まる．更に P の作り方から，

$$a_{\mathcal{B}} = \begin{pmatrix} v_1 & v_2 & \cdots & v_k \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & \cdots & w_k \end{pmatrix} P \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} = \{Pa_{\mathcal{B}}\}_{\mathcal{B}'}$$

による変換は $\mathcal{B}$ の元が $\mathcal{B}'$ の元の間の関係式を $\mathbb{F}_{p^k\mathcal{B}}$ において満たすため，乗算を保存する．又，逆に乗算を保存する基底変換 P が在るとき，当然上のことは成り立つ．従って，求める基底変換は，一方の基底表示を用いて，他方の基底を成す既約多項式の根を求めることで定まる．

Equal Degree Factorization による基底変換

多項式基底の間の基底変換について EDF (Equal Degree Factorization) を用いたもの [24] がある．EDF は多項式の既約分解を求めるアルゴリズムであり，EDF を用いて，拡大体を構成する定義多項式の一次の既約式，即ち定義多項式の根を直接求めることにより基底変換を行う．EDF により定義多項式の根を求め，既定変換を得る方法を Algorithm 4 に示す．ここで基底変換は

$$\mathbb{F}_{p^k\mathcal{B}} \simeq \mathbb{F}_p[x]/(f(x)) \leftrightarrow \mathbb{F}_{p^k\mathcal{B}'} \simeq \mathbb{F}_p[x]/(g(x))$$

の間のものを考える．

Algorithm 4 EDF による多項式基底間の基底変換

Input: $\mathbb{F}_{p^k{}_{\mathcal{B}}} \simeq \mathbb{F}_p[x]/(f(x)),\ \mathbb{F}_{p^k{}_{\mathcal{B}'}} \simeq \mathbb{F}_p[x]/(g(x))$

Output: 基底変換 $T : \mathcal{B} \to \mathcal{B}'$

1: **EDF による一次既約因子の探索**
2: $F(x) = f(x) \in \mathbb{F}_{p^k{}_{\mathcal{B}'}}[x]$
3: **while** $\deg(F(x)) \neq 1$ **do**
4: ランダムに $A(x) \in \mathbb{F}_{p^k{}_{\mathcal{B}'}}[x]$ を取る
5:
6: **trace の計算**
7: $T(x) = A(x) + A(x)^{\sigma} + \cdots + A(x)^{\sigma^{k-1}}$
8:
9: $F(x) = (F(x), T(x))$
10: **if** $F(x) = 1$ **then**
11: $F(x) = f(x)$
12: **end if**
13: **end while**
14: $f(x)$ の根 $\theta_{\mathcal{B}'}$ を得る
15: $(1\ \theta_{\mathcal{B}'}\ \cdots\ {\theta_{\mathcal{B}'}}^{k-1}) = \mathcal{B}'T$ なる変換行列 T を計算する

提案手法では，normal basis を用いて，この EDF による基底変換アルゴリズムの改良を行う．

Normal basis を用いた基底変換

野上らは [19] において，GNB (Gauss period normal basis) という normal basis の中でも特別なものを用いて，この基底を経由して与えられた基底の間の変換を求める手法を提案した．後の第 4.2 節で詳細に述べるが，与えられた基底に対し，その基底とある normal basis との基底変換を求めることが可能である．従って，二つの基底間の基底変換をそれぞれの基底と normal basis との変換を求め normal basis を経由して，つまり，それらの変換を合成して求める方法が考えらる．しか

し，normal basis を生成する拡大体の元は多く存在し，それぞれ基底に対して求めた normal basis が基底として一致していなければ，基底変換を合成できない．野上らはより特別な normal basis を用いることにより，この問題を解決している．GNB については，一度その生成元を見つけると，他の生成元を見つけたとしてもそれぞれの生成元が互いに共役（Frobenius map で写り合う）である，即ち同じ normal basis を生成する．従って，GNB の生成元を一つ見つければ，GNB を経由して基底変換を求めることが可能である．加えて，GNB の生成元は拡大体上の元として，その位数によって特徴付けられ，探索をより効率的に行うことが出来る．

しかし，GNB を持つ拡大体には，標数，拡大次数に制限があるため，本研究では GNB の様な特別な基底を用いることは考えず，先に述べた EDF を用いたアルゴリズムについて考察する．

4.2 基底変換の求解についての提案

本節では，先の第 4.1 節で述べた多項式基底間の基底変換の求解について提案手法を述べる．特に注意しなければ，記号は第 4.1 節のものに従う．本節において，第 4.2.1 節では基底変換の求解のための提案手法を述べる．初めに，与えられた多項式に対して，その基底とある normal basis との変換を求めるアルゴリズムについて提案する．そして，normal basis への変換を利用した EDF による基底変換求解のアルゴリズムのコスト削減を図る手法について子細に説明する．第 4.2.2 節では提案手法の評価として，基底変換のシミュレーション結果を示す．更に，変換自体のコストについてシミュレーション結果と考察を与える．

4.2.1 提案手法

本節では，初めに，与えられた多項式基底に対し，その基底表示を用いた，ある normal basis を成す元の探索法について述べる．次に，任意の多項式基底間の基底変換を，先の第 4.1 節で示した Algorithm 4 を改良することにより，より短時間で求めることが可能であることを示す．

Normal basis を成す元の探索

まず，有限体論より，拡大体の元の組が基底を成すときの必要十分条件について復習する．q は標数 p の冪とする．

補題 8 ([18, Corollary 2.38])．$\alpha_1, ..., \alpha_k \in \mathbb{F}_{q^k}$ に対し，$\{\alpha_1, ..., \alpha_k\}$ が $\mathbb{F}_{q^k}$ の $\mathbb{F}_q$ 上の基底であるためには，

$$\begin{vmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_k \\ \alpha_1^q & \alpha_2^q & \cdots & \alpha_k^q \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{q^{k-1}} & \alpha_2^{q^{k-1}} & \cdots & \alpha_k^{q^{k-1}} \end{vmatrix} \neq 0 \tag{9}$$

を満たすことが必要十分である．

次に，normar basis について，定義を再掲する．拡大体 $\mathbb{F}_{q^k}/\mathbb{F}_q$ の基底 $\mathcal{B}$ が，ある元 $a \in \mathbb{F}_{q^k}$ と Frobenius map $\sigma \in \mathrm{Aut}_{\mathbb{F}_q}(\mathbb{F}_{q^k})$ により

$$\mathcal{B} = \{a, \sigma(a), ..., \sigma^{k-1}(a)\}$$

と書けるとき，$\mathcal{B}$ を normal basis と呼ぶ．任意の有限体の有限次拡大体に対し normal basis は存在する．

式 (9) より，

$$\begin{vmatrix} a_{\mathcal{B}} & \sigma(a_{\mathcal{B}}) & \cdots & \sigma^{k-1}(a_{\mathcal{B}}) \\ \sigma(a_{\mathcal{B}}) & \sigma^2(a_{\mathcal{B}}) & \cdots & a_{\mathcal{B}} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{k-1}(a_{\mathcal{B}}) & a_{\mathcal{B}} & \cdots & \sigma^{k-2}(a_{\mathcal{B}}) \end{vmatrix} \neq 0 \tag{10}$$

を満たす元 $a_{\mathcal{B}}$ を見つければ

$$\mathcal{N} = \{a_{\mathcal{B}}, \sigma(a_{\mathcal{B}}), ..., \sigma^{k-1}(a_{\mathcal{B}})\}$$

は $\mathbb{F}_{q^k{}_{\mathcal{B}}}$ において normal basis を成す．式 (10) の行列式の値は，$\mathbb{F}_{q^k{}_{\mathcal{B}}}$ 上の多項式

$$f(x) = x^k - 1, \quad g(x) = a_{\mathcal{B}} x^{k-1} + a_{\mathcal{B}}^q x^{k-2} + \cdots + a_{\mathcal{B}}^{q^{k-2}} x + a_{\mathcal{B}}^{q^{k-1}} \tag{11}$$

における終結式$\mathrm{Res}(f(x), g(x))$と一致するから ([18, Theorem 2.39]), $\mathcal{N}$がnormalであることと，$f(x)$と$g(x)$が互いに素である（共通根を持たない）ことは同値である．

以下，$q = p$として考える．上の議論から，$\mathbb{F}_{q^k}{}_{\mathcal{B}}$に対し，式 (11) の$f(x), g(x)$に対し$(f(x), g(x)) = 1$なる$a_{\mathcal{B}}$はnormal basis $\mathcal{N}$を成し，

$$\mathcal{N} = (a_{\mathcal{B}}, \sigma(a_{\mathcal{B}}) \cdots \sigma^{k-1}(a_{\mathcal{B}})) = \mathcal{B}P$$

として基底変換$P\colon \mathcal{N} \to \mathcal{B}$が求まる．ここで，提案手法としてGCD計算の代わりに，$f(x), g(x)$が共通根を持つかどうか直接確かめる方法について考察する．

まず，$f(x) = x^k - 1$の根の探索について考察を与える．位数$(k, p^k - 1)$の元が成す巡回群を直接求める方法により，$f(x)$の根を求めることが出来る．

- $k \nmid p^{k-1}$のとき．
  - $(k, p^k - 1) = 1$のとき，位数がkの約数である元は1のみ．よって，$x^k - 1$の根は1のみ．
  - $(k, p^k - 1) = d > 1$のとき，拡大体の元$a \in \mathbb{F}_{p^k}$で$(a^{(p^k-1)/d})^m \neq 1,\ (1 \leq m \leq d-1)$なるものを確率的に探す．この様な$a$は位数$d$の巡回群をなし，$x^k - 1$の$\mathbb{F}_{p^k}$における根をすべて含む．
- $k \mid p^{k-1}$のとき，$c = (p^k - 1)/k$として，$(a^c)^m \neq 1\ (1 \leq m \leq k-1)$なる元$a \in \mathbb{F}_{p^k}$は位数$k$の巡回群を生成する．このとき，その巡回群は，$x^k - 1$の$\mathbb{F}_{p^k}$における根をすべて含む．

これにより，$f(x)$の根の候補を調べあげ，$g(x)$と共通根を持たないような$a_{\mathcal{B}}$の探索を試みる．しかし，ここで注意が必要である．**$f(x)$の根は拡大体$\mathbb{F}_{p^k}$において，完全に分離するわけではない．**$k \mid p^{k-1}$のとき，$f(x)$は$\mathbb{F}_{p^k}$に根をk個持つので，$f(x)$と$g(x)$が共通根を持たないことと，$(f(x), g(x)) = 1$を満たすことは同値である．しかし，$k \nmid p^{k-1}$のとき，$f(x)$は全ての根を$\mathbb{F}_{p^k}$に持つわけではないため，得られた$f(x)$の根が$g(x)$の根と共通でなくとも，$g(x)$と共通因子を持つ可能性がある．従ってこの手法で$a_{\mathcal{B}}$を求めた時，$k \nmid p^{k-1}$の場合では変換行列が生成できない可能性がある．本研究では，変換行列として正則行列が得ら

れない場合を考慮し，どの程度の失敗が含まれるかシミュレーションにより確かめた．

提案手法についてまとめる．シミュレーションでは比較のため次の 3 つの方法による，多項式基底と normal basis との基底変換について実装を行った．

- **GCD:** $(f(x), g(x)) = 1$ を満たす $a_{\mathcal{B}}$ を確率的に探索する方法．
- **no common root:** $\mathbb{F}_{p^k}$ における $f(x)$ の根を求めて，$g(x)$ と共通根を持たないような $a_{\mathcal{B}}$ を確率的に探索する方法．**変換行列の生成に失敗する可能性がある．**
- **GCD + no common root:** 上の no common root の方法において，$k \nmid p^{k-1}$ の時は GCD 計算によるもので normal basis の生成元を求めるもの．これにより変換行列の生成に失敗することはない．

多項式基底間の基底変換

最後に，多項式基底間の基底変換について，提案手法を述べる．本研究では，第 4.1 節で述べた様に，任意の拡大体のパラメタを許容出来るように，特別な GNB などの基底を用いず，EDF を用いた素朴なアルゴリズムを使用する．Algorithm 4 において，EDF による基底変換は，step 7 の trace の計算のコストが拡大次数のサイズに対し急激に増加する．そこで，本提案手法では，上で述べた normal basis への基底変換用いて trace の計算を効率的に行い，変換行列の探索の高速化を実現した．具体的な手法を Algorithm 5 に示す．

Algorithm 5 Normal basis を用いた EDF による多項式基底間の基底変換

Input: $\mathbb{F}_{p^k\mathcal{B}} \simeq \mathbb{F}_p[x]/(f(x)),\ \mathbb{F}_{p^k\mathcal{B}'} \simeq \mathbb{F}_p[x]/(g(x))$

Output: 基底変換 $T : \mathcal{B} \to \mathcal{B}'$

1: **EDF による一次既約因子の探索**
2:
3: **基底変換 Q: $\mathcal{B}' \to \mathcal{N}$ を求める**
4: $F(x) = f(x) \in \mathbb{F}_{p^k\mathcal{B}'}[x]$
5: **while** $\deg(F(x)) \neq 1$ **do**
6: ランダムに $A(x) \in \mathbb{F}_{p^k\mathcal{B}'}[x]$ を取る
7:
8: trace の計算
9: **$A(x)_{\mathcal{N}}$ を計算する**
10: $T(x) = A(x)_{\mathcal{N}} + A(x)^{\sigma}_{\mathcal{N}} + \cdots + A(x)^{\sigma^{k-1}}_{\mathcal{N}}$
11:
12: $F(x) = (F(x), T(x))$
13: **if** $F(x) = 1$ **then**
14: $F(x) = f(x)$
15: **end if**
16: **end while**
17: **$A(x)_{\mathcal{B}'}$ を計算する**
18: $f(x)$ の根 $\theta_{\mathcal{B}'}$ を得る
19: $(1, \theta_{\mathcal{B}'} \cdots {\theta_{\mathcal{B}'}}^{k-1}) = \mathcal{B}'T$ なる変換行列 T を計算する

trace 計算を normal basis による表現へ変換して行うことで，Frobenius map による作用はシフト演算で行えるため非常に高速である．全体として基底変換に掛かる時間よりも，normal basis において trace 計算を行う方が高速である．

4.2.2 シミュレーション結果

本節では，第 4.2 節で述べた基底変換に関する提案手法に対して，シミュレーション結果を示し幾つか考察を与える．加えて基底変換のコストについてもシミュレーション結果を述べる．シミュレーションは NTL を用いた C++によって実装した．以下の表 3 に，基底変換に関するシミュレーション環境を示す．

OS	Fedora 17
CPU	Intel Core i7-3960X, 3.30GHz, 6 Cores
Memory	DDR3-1333, 64GB
Compiler	GCC-4.5.3
Language	C++ with NTL-5.5.2

表 3 実装環境（基底変換求解）

Normal basis を成す元の探索

図 1 において，標数が 3 の時の，多項式基底と normal basis との基底変換求解に掛かる実行時間に関するシミュレーション結果を示す．小標数においては，提案手法である $f(x)$ と $g(x)$ の共通根をチェックする手法が GCD に比べ高速である．ここで，計算時間が 0 を取っている拡大次数が 90 と 105 付近に二つあるが，これは変換行列として計算した行列が正則ではなかった場合である．しかし，小標数の時でさえも，変換行列生成の失敗が起こる場合は少なく，ほとんどの拡大次数において，$k \nmid p^{k-1}$ の時であっても変換行列を求めることが出来ている．

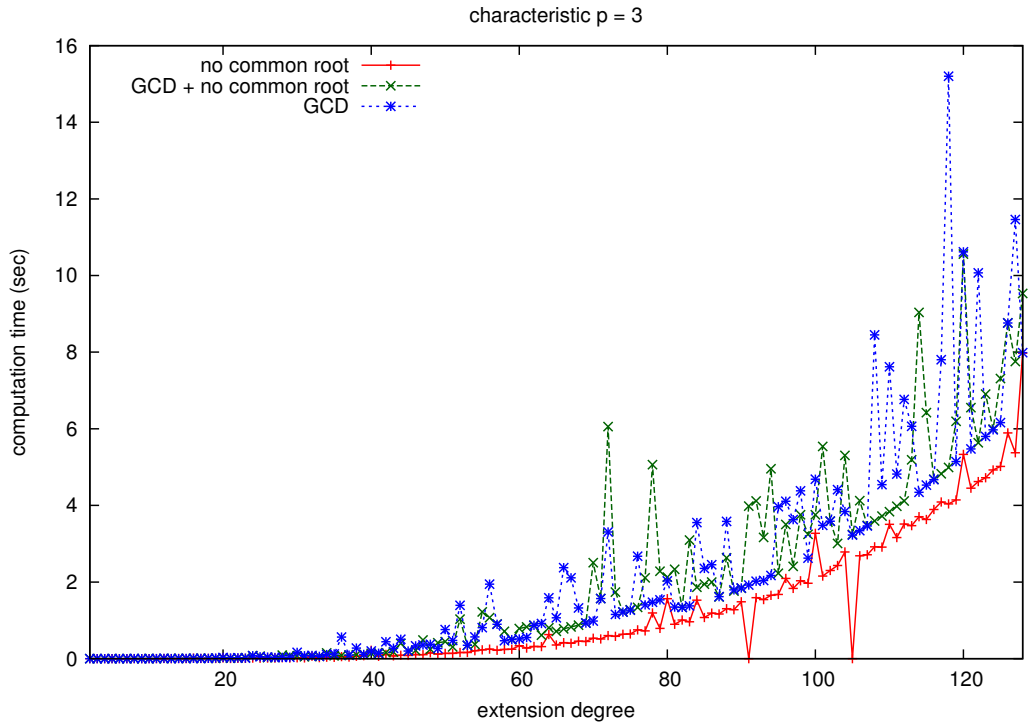

図 1 $\mathbb{F}_{p^k}$ $(p = 3)$ の多項式基底と normal basis の間の基底変換求解に掛かる時間

次に，図 2 において，標数が 128-bits prime の時の，多項式基底と normal basis との基底変換求解に掛かる実行時間に関するシミュレーション結果を示す．この時は，安定して GCD による求解が高速であり，提案手法はそれぞれの拡大次数において約 10 倍程度 GCD に比べ解の探索に時間が掛かっている．又，標数が少しでも大きいと，変換行列生成が失敗することはまず起こらなくなる．

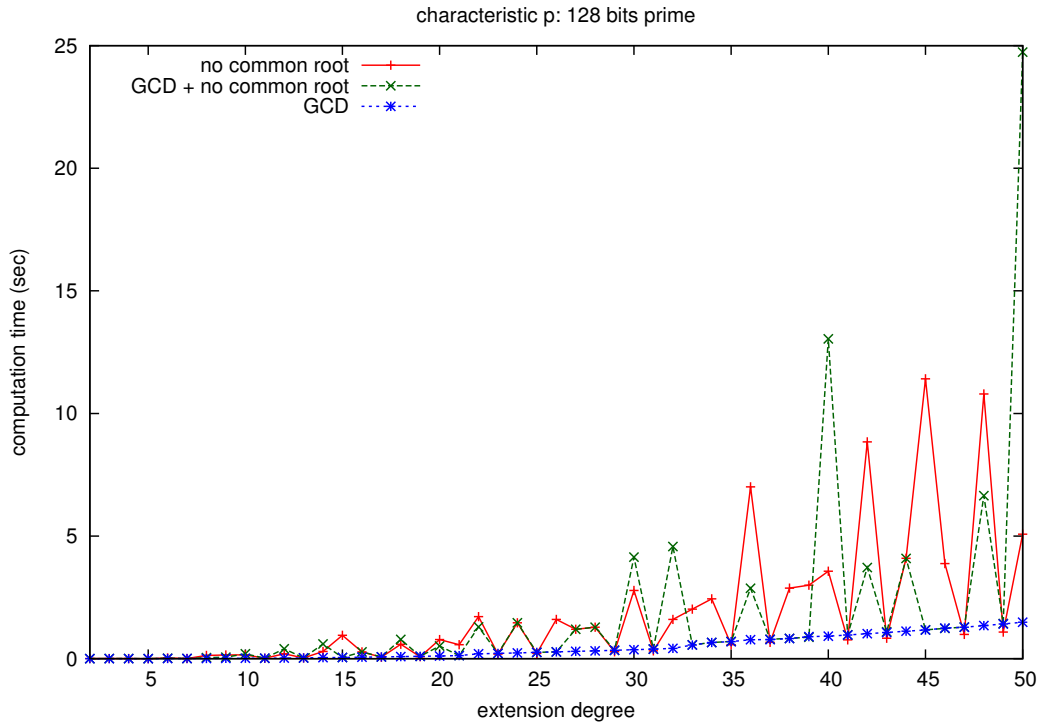

図 2 $\mathbb{F}_{p^k}$ p: 128-bits prime の多項式基底と normal basis の間の基底変換求解に掛かる時間

多項式基底間の基底変換

次に，多項式基底間の基底変換の求解についてシミュレーション結果を見る．図 3 において，小標数 $p = 3$ の時のシミュレーション結果を示した．小標数の場合では，提案手法と既存手法に差異はあまり見られない．

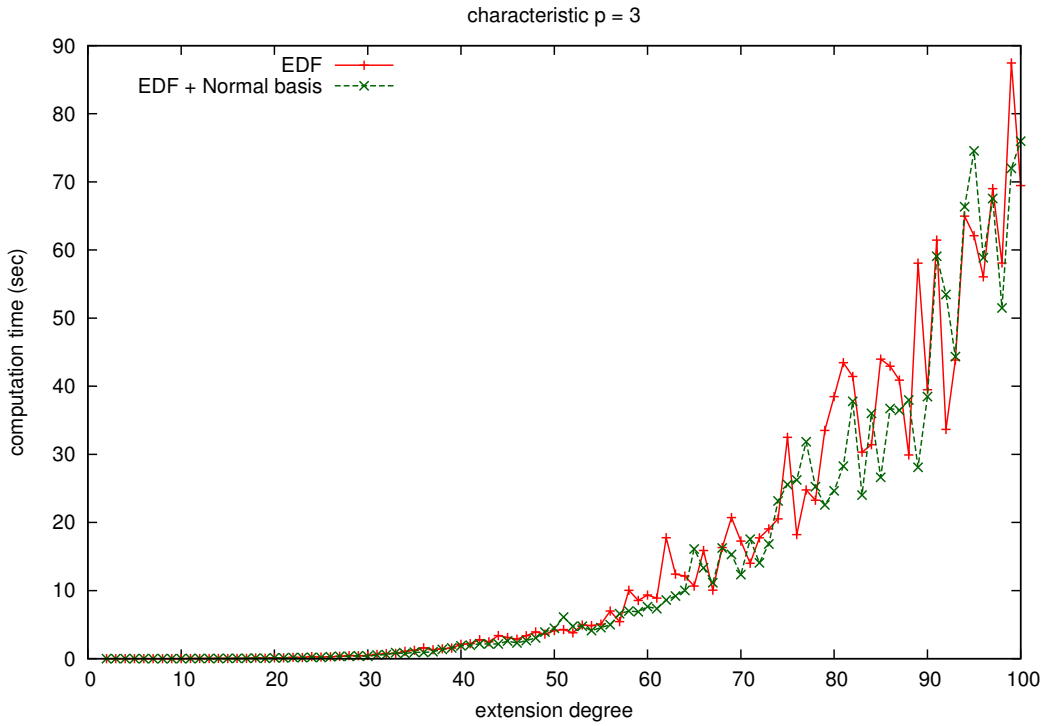

図 3 $\mathbb{F}_{p^k}$ $(p = 3)$ の多項式基底間の基底変換求解に掛かる時間

最後に，図 4 において，標数が 16-bits prime の時の多項式基底間の基底変換求解に関するシミュレーション結果を示す．標数が大きい程，既存手法では拡大次数の増加により急激に求解に掛かる計算時間が増加するが，提案手法においては，増加度合いも既存手法に比べ緩やかで，大きなサイズの $\mathbb{F}_{p^k}$ においても現実的な計算コストで基底変換を求めることが出来る．

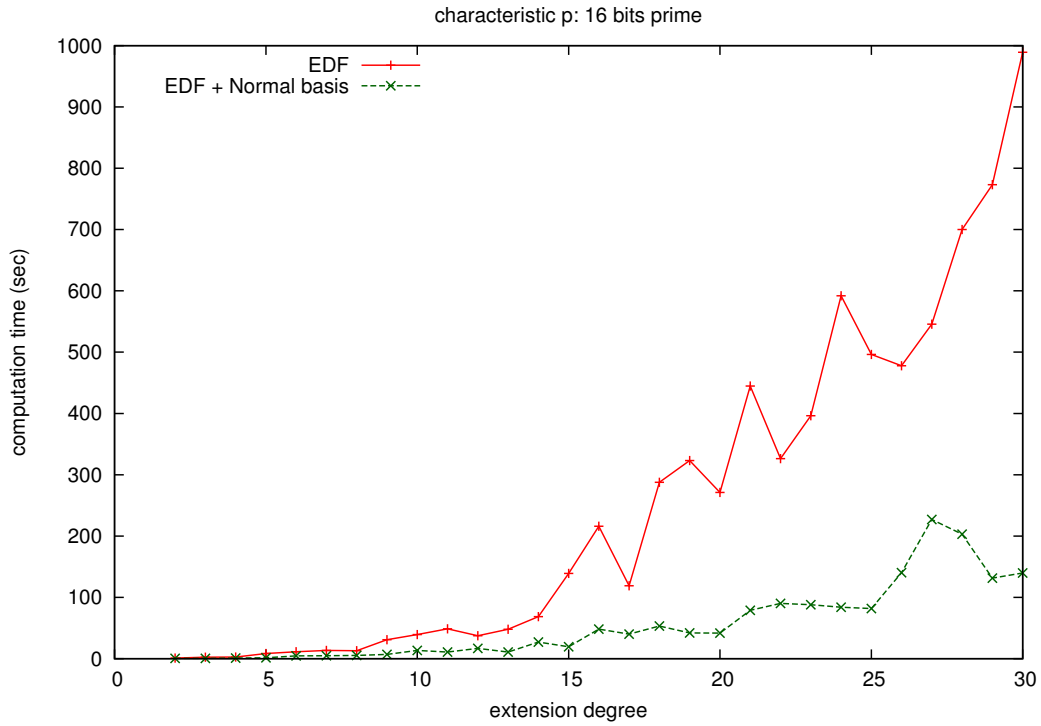

図 4 $\mathbb{F}_{p^k}$ p: 16-bits prime の多項式基底間の基底変換求解に掛かる時間

基底変換のコスト

ここでは有限体の基底変換について，その変換行列のコストに着目する．基底変換は変換行列を掛けることにより行われるが，変換行列が成分に 0 を多く持つような sparse なものであると変換に掛かるコストが小さく効率的である．特に，現在考察している素体上の単拡大ではなく，サイズの大きな拡大上の体拡大や，大きなサイズの標数を持つ時は基底変換に掛かるコストを無視することは出来ない．本研究では，**変換コストを変換行列の 0 でない成分の個数**とし，シミュレーションを行う．

図 5 において，多項式基底を成す定義多項式をランダムに取り，それらの多項式基底間の変換行列のコストに対するシミュレーション結果を示す．多項式基底

の間の基底変換は，多項式基底を成す定義多項式の根の取り方に依り変化するため，根を動かした時の変換コストの最小，平均，最大値をエラーバー付プロットで描く．$p = 2, 3, 5$ 等の小標数の場合は，コストの振れ幅も大きいが，$p = 67$ の様に少しでも標数が大きくなると，最小，最大値に差はなく，多項式基底間の変換コストの最大値である $k^2 - k + 1$ に近づき，ほとんど上限値を取ってしまう．直接的に考えられることだが，ランダムに基底を取るとコストは平均的に $(p-1)/p$ 程度となる．従って，標数が少しでも大きい時は，定義多項式間の根の関係が簡単であるような組み合わせの基底を用いるべきである．極端な例としては，定義多項式の根が normal basis の生成元を持つとき，即ち，根全体が normal basis を成すとき変換行列は単位行列，或いはその列を置換したものが考えられ，実装面から見るとこの様な定義多項式を採用すると計算効率が良い．

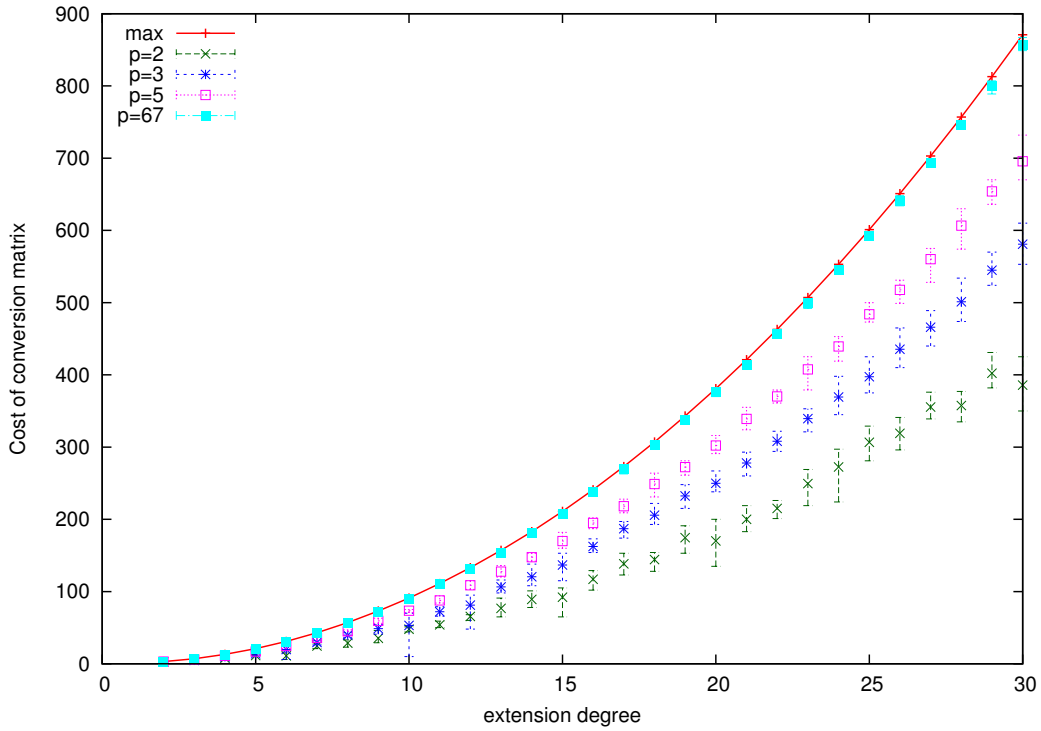

図 5　ランダムに取った多項式基底間の基底変換コストの比較

4.3 12 次拡大体上の基底変換

本節では，本研究で対象としている η_T ペアリングのパラメタである拡大体 $\mathbb{F}_{2^{12m}}$ の基底変換について述べる．η_T ペアリングの本並列実装においては，$\mathbb{F}_{2^{12m}}$ の二種類の基底とその構成法に従って実装を行っており，それらの基底間の基底変換をアルゴリズムの中で適用している．$\mathbb{F}_{2^{12m}}$ の一つの構成は，第 3.1 節で述べた先行研究における方法で，6 次拡大を構成し，その上に 2 次拡大体を構成したものである．以下に，具体的にその形を再掲する．

まず，

$$\mathbb{F}_{2^{6m}} \simeq \mathbb{F}_{2^m}[x]/(x^6 + x^5 + x^3 + x^2 + 1)$$

として 6 次拡大体を構成し，$\mathbb{F}_{2^{12m}}$ を

$$\mathbb{F}_{2^{12m}} \simeq \mathbb{F}_{2^{6m}}[y]/(y^2 + y + w^5 + w^3),\ w^6 + w^5 + w^3 + w^2 + 1 = 0$$

として構成し，s_0 を $y^2 + y + w^5 + w^3$ の根として，基底

$$\{1, w, w^2, w^3, w^4, w^5, s_0, s_0w, s_0w^2, s_0w^3, s_0w^4, s_0w^5\}$$

を用いて表す．この基底を s_0w-basis と呼んだ（第 3.3 節）．

更に，本研究の η_T ペアリングの並列化において，提案手法となる $\mathbb{F}_{2^{12m}}$ の 3 次拡大から始めた構成について，詳細は，後の第 5.1.1, 5.1.4 節において記述するが，ここで基底変換求解の説明のため，先にその構成法を紹介しておく．

$\mathbb{F}_{2^{3m}}$ を $\mathbb{F}_{2^m}$ 上の既約多項式 $x^3 + x + 1$ を用いて構成し，

$$\mathbb{F}_{2^{6m}} \simeq \mathbb{F}_{2^{3m}}[y]/(y^2 + y + w + 1),\ (w^3 + w + 1 = 0)$$

とし，最終的に 12 次拡大体

$$\mathbb{F}_{2^{12m}} \simeq \mathbb{F}_{2^{6m}}[z]/(z^2 + z + s + sw^2),\ (s^2 + s + w + 1 = 0)$$

を構成し，多項式基底

$$\{1, w, w^2, s, sw, sw^2, t, tw, tw^2, st, stw, stw^2\}$$

を用いて元を表す．この基底を stw-basis と呼ぶ．

ここでは，s_0w-basis と stw-basis との間の基底変換の内，最小のコストを持つものを求める．先の第 4.2.2 節における基底変換のコストについて言及した箇所で述べたが，多項式基底間の基底変換は，多項式基底を成すその定義多項式の根の選び方に依り，その変換行列の形が変わる．従って，変換コストが最も小さくなるような定義多項式の根を選ぶ必要がある．根の変換は Frobenius map $\sigma \in \mathrm{Aut}_{\mathbb{F}_2}(\mathbb{F}_{2^{12}})$ の元の作用で行われ，その作用としてそれぞれの基底で σ を行列表示したものを掛けることにする．ここで，基底変換は $\mathbb{F}_{2^{12m}}$ 上ではなく $\mathbb{F}_{2^{12}}$ において，求解を行うことに注意されたい．今，それぞれの基底において始めの拡大に用いられる定義多項式

$$x^6 + x^5 + x^3 + x^2 + 1, \quad x^3 + x + 1$$

が $\mathbb{F}_2$ 上でも定義出来るため，変換行列の成分は全て素体の元とすることが出来る．

s_0w-basis から stw-basis への基底変換行列 P' を求めるために，まず stw-basis による表現を用いて，定義多項式 $x^6 + x^5 + x^3 + x^2 + 1$ の根 w_{stw} を求める．ここで，a_{stw} は拡大体の元 a を stw-basis で表すことを意味する．次に，この w_{stw} を用いて，定義多項式 $y^2 + y + w_{stw}^5 + w_{stw}^3$ の根 s_{0stw} を求める．この時，

$$(1\ w_{stw}\ w_{stw}^2 \cdots w_{stw}^5\ s_{0stw}w_{stw}\ s_{0stw}w_{stw}^2 \cdots s_{0stw}w_{stw}^5) = (\ stw\text{-basis}\ )P'$$

として一つの変換行列 P' が求まる．ここで，Frobenius map $\sigma(x) = x^2$ を s_0w-basis で表現した行列を F と書くと，$P'F^i$ $(1 \leq i \leq k)$ により，定義多項式の根の置換による全ての基底変換を得ることが出来る．今回の計算においては，P' の具体的な形はここでは述べないが，任意の Frobenius map を取り，全ての変換行列 $P'F^i$

を求め，その中のコスト最小のものとして，

$$P = P'F^3 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{12}$$

を得ることが出来，そのコストは 32 であった．

上記と同様のことを s_0w-basis と stw-basis の役割を入れ替えて考えることが出来る．今回の計算では P' の逆行列として，基底変換

$$P'^{-1} = Q' \colon stw\text{-basis} \to s_0w\text{-basis}$$

を求めて，stw-basis を用いて σ を表現したものを F' とすると，$Q'F'^i$ の中でコス

ト最小のものとして，

$$Q = Q'F'^9 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{13}$$

を得ることが出来，そのコストは 28 であった．

この時，$\sigma^{12} = 1$ より $F^3F'^9 = F'^9F^3 = I$ であり，$P'^{-1} = Q'$ であったから今回の場合 P, Q は互いに逆変換であり，$Q = P^{-1}$ が成り立つ．一般の場合でコスト最小のものが互いに逆変換となるとは限らず，その時は，基底変換の適用が終了した際に帳尻合わせとして余った分の Frobenius map を作用させる必要があるが，今回の $\mathbb{F}_{2^{12m}}$ における s_0w-basis と stw-basis との間の基底変換においては，それぞれのコストが最小となる様な逆変換を互いに持ち，変換コストを最小と出来る．η_T ペアリングの本並列実装では，実際に基底変換の際上記の変換行列 P, Q を用いる．

5. 拡大体の基底変換を用いたη_Tペアリングの並列実装に関する提案

本章では，本研究のη_Tペアリングの効率的な並列実装に関する提案を述べる．本提案は，第3章で具体的に述べたBarretoら[5]によるη_Tペアリングに関して，拡大体$\mathbb{F}_{2^{12m}}$の構成を，既存研究のものとは異なる方法で拡大体を構成し，それにより拡大上の乗算，特に$\alpha\beta$を小さなコストで計算し，効率的な並列実装を行うものである．本章では拡大体の構成を変更した際のη_Tペアリングの構成と，拡大体上の乗算の計算式を第3章と同様に子細に述べる．更に本章において，提案手法を用いたη_Tペアリングの並列実装について，並列化手法，実装方法，又，基礎体，拡大体上の基本演算に対する並列実装結果を示す．

まず，第5.1節において，先行研究のη_Tペアリングの計算における拡大体$\mathbb{F}_{2^{12m}}$の構成を変更した際のペアリングのアルゴリズムについて子細に説明する．先行研究では，第3.1節で述べた様に，6次拡大から始め，その後2次拡大を構成し，12次拡大体を構成している．これに対し，本研究では$\mathbb{F}_{2^{12m}}$を3次拡大から始めて，その後2次拡大を二度繰り返した構成を取り，この構成によるdistortion map ψの表現，拡大体上の乗算$\alpha\beta$の具体的な計算式を与えた．特に，乗算$\alpha\beta$の計算コストについて，第3.3節と同様にして，拡大体の構成を変更した際の計算コストを具体的に求め，先行研究の拡大体の構成に比べ，$\alpha\beta$の計算コストを削減できることを示す．

次に，第5.2節において，本研究で取り組む，η_Tペアリングの並列実装に関して，実装方針，並列化の概要，基礎体，又は拡大体の基本的な算術の並列実装，更に，η_Tペアリングの並列化方針と具体的な実装内容について述べる．本節では，実装方針，実装内容のみならず，基礎体，拡大体の算術に関しては，実装結果として加算，乗算等の実行時間に関するシミュレーション結果を示し，ペアリングについては，第6章においてより具体的に議論，考察を与える．

5.1 拡大体の構成を変更した η_T ペアリング

本節では，η_T ペアリングの計算に係る拡大体 $\mathbb{F}_{2^{12m}}$ について，第 3.1 節の 6 次拡大，2 次拡大による構成ではなく，3 次拡大から始め，逐次拡大により拡大体を構成した場合の η_T ペアリングの具体的なアルゴリズムを記す．更に，η_T ペアリングの Miller loop における拡大体上の乗算 $\alpha\beta$ の計算コストについて，第 3.3 節と同様にして，基礎体，拡大体上の算術，或いは，Karatsuba method による乗算コスト等を用いて評価し，6 次拡大から始める既存研究の拡大体の構成に比べ，より小さなコストで $\alpha\beta$ の計算が行えることを明らかにする．

まず，第 5.1.1 節において，既存研究とは異なる拡大体の構成方法と，拡大体の構成の変更により，η_T ペアリングのアルゴリズムの形に変化がある箇所を具体的に挙げる．次に，第 5.1.2 節において，拡大体の構成を変更した実際の η_T ペアリングのアルゴリムを書き下すために，第 3.2.2, 3.2.3 節で述べた様に，基底の元，或いは，それらの冪乗の間の関係式を具体的に述べる．第 5.1.3 節では，第 3.2.4 節に記した Algorithm 3 の様に，拡大体の構成を変更した際の η_T ペアリングのアルゴリズムの詳細を述べる．最後に，第 5.1.4 節では，第 3.3 節との対比で，η_T ペアリングの計算において必要となる拡大体上の乗算 $\alpha\beta$ の計算コストについて，拡大体の構成の変更による計算式の形，コスト削減について述べる．ここでは計算コストの理論的な推定に留まり，実装実験結果については，拡大体の構成を変更する前の場合とまとめて，後の第 5.2 節で述べる．

5.1.1 拡大体の構成の変更に伴うペアリングパラメタとデータ表現

第 3.1 節より，既存研究 [5, §7] では拡大体 $\mathbb{F}_{2^{12m}}$ を，

$$\mathbb{F}_{2^{12m}} \simeq \mathbb{F}_{2^{6m}}[y]/(y^2 + y + w^5 + w^3), \quad w^6 + w^5 + w^3 + w^2 + 1 = 0$$

として構成し，s_0 を $y^2 + y + w^5 + w^3$ の根として，多項式基底

$$\{1, w, w^2, w^3, w^4, w^5, s_0, s_0w, s_0w^2, s_0w^3, s_0w^4, s_0w^5\}$$

を用いて拡大体の元を表現した．本節では，まず 3 次拡大から始め，12 次拡大体を逐次的に構成する方法と具体的な定義多項式を示す．当初，拡大体の構成を変

更する動機としては，distortion map

$$\psi(x,y)=(x+w,y+s_2x^2+s_1x+s_0),\quad s_1=w^4+w^2,\ s_2=w^8+w^4+w=w^4+1$$

の構成にあった．この distiortion map は，6 次拡大を構成するために基礎体に添加する 6 次既約多項式 $x^6+x^5+x^3+x^2+1$ の根 w を用いて定義されており，本研究ではこの w を 3 次既約多項式（或いはこれより低次数の）から取り ([8, §8]), distortion map の表現を変え，α,β のより sparse な表現を得，拡大体上の乗算 $\alpha\beta$ のコストを削減しようと考えた．しかし，その様に既約 3 次式の根として w を選択すると，distortion map ψ は曲線 C 上の次数 6 の自己同型，即ち，$C(\mathbb{F}_{2^{6m}})$ 上の自己同型になり，同等のセキュリティレベルを保つために η_T ペアリングの構成を大幅に変更しなければならず，効率化も見込めない．よって，distortion map の構成は本質的に変更しないが，拡大体の構成を変更し拡大体上の乗算コストを削減させることに取り組んだ．又，この後の並列実装に関する第 5.2 節で述べる様に，3 次拡大体における Karatsuba 乗算が効率的に並列化できることから，本研究では次の様に 3 次拡大から拡大体を構成することにした．

まず，

$$\mathbb{F}_{2^{3m}}\simeq\mathbb{F}_{2^m}[x]/(x^3+x+1)$$

として 3 次拡大体を構成する．distortion map の構成にも既約 3 次式の根を使用することを考慮すると，[8, §8] より 3 次式は x^3+x+1, 或いは x^3+x^2+1 のどちらかを用いる必要がある．次に 6 次拡大を

$$\mathbb{F}_{2^{6m}}\simeq\mathbb{F}_{2^{3m}}[y]/(y^2+y+w+1),\quad w^3+w+1=0$$

として構成し，最終的に 12 次拡大体を

$$\mathbb{F}_{2^{12m}}\simeq\mathbb{F}_{2^{6m}}[z]/(z^2+z+s+sw^2),\quad s^2+s+w+1=0$$

として構成し，t を $z^2+z+s+sw^2$ の根として選択する．このとき，$\mathbb{F}_{2^{12m}}$ の元は多項式基底

$$\{1,w,w^2,s,sw,sw^2,t,tw,tw^2,st,stw,stw^2\}\tag{14}$$

を用いて表される．又，上の構成について，既約 2 次式 $y^2 + y + w + 1$ の根である s は，6 次拡大から始める拡大体の構成の既約 6 次式 $x^6 + x^5 + x^3 + x^2 + 1$ の根 w と対応しており，この s を用いて distortion map ψ を構成する．即ち，

$$\psi(x, y) = (x + s, y + t_2 x^2 + t_1 x + t), \tag{15}$$
$$t_1 = s^4 + s^2 = w^2 + 1, \quad t_2 = s^8 + s^4 + s = s + w^2 + w + 1$$

として構成すると，これは 6 次拡大から構成した拡大体の定義多項式の根を用いて構成したものと同一のものになる．このとき，この ψ と自己同型 (5) を用いて第 3.1 節で述べたパラメタ T を変更せず，全く同等の η_T ペアリングを構成できる．即ち，条件 (6) など η_T ペアリングの構成が可能であるための条件を全て満たす．つまり，η_T ペアリングを構成する φ や distortion map ψ を変更せず，拡大体の構成だけを変更したという事になる．

上記の多項式基底 (14), distortion map (15) を用いて η_T ペアリングを構成する際，パラメタ，又は計算式の表現の変更を含むものを第 3.2 節の記述と対応を付けて以下に挙げる．

- **α, β の基本形** Mumford 表現と Cantor のアルゴリズムによる 2 倍算，4 倍算，8 倍算の計算式には当然変更はないが，distortion map の変更により α, β の表現式が変わる．
- **基底の元の冪乗** 多項式基底の元の間に入っている関係式が異なるため，それぞれの基底の元の冪乗の形を計算し直す必要がある．
- **α, β の冪乗** 上記にある様に α, β と基底の元の冪乗の形に変更があるため，α, β の冪乗の形も変わる．
- **final doublings / addition** 今回の η_T ペアリングのパラメタにおいて，Miller loop での計算が終了した後に，4 倍算を一度行う必要がある．ここでは 4 乗算の計算式に distortion map を合成するため式の形が変わる．

アルゴリズムの流れとしては，拡大体の構成を変更する前の Algorithm 3 と変更はなく，詳細は後の第 5.1.3 節で述べる．又，α, β の冪乗の表現が変わるため，

乗算 $\alpha\beta$ の形にも変更があり，本提案手法の主要結果の一つである，より小さな計算コストにおける乗算 $\alpha\beta$ を達成することができる．その具体的な乗算コストについては後の第 5.1.4 節で詳しく見る．

5.1.2 拡大体の構成を変更した η_T ペアリングのアルゴリズム

本節では，先の第 5.1.1 節で述べた 3 次拡大から始める拡大体 $\mathbb{F}_{2^{12m}}$ の構成による，η_T ペアリングの計算において変更があるデータ，式について具体的に述べる．第 3.2.2, 3.2.3 節と対応付けた表記を用い，特に注意しなければ記号は [5, §7, Appendix A,B] に従う．

α の計算

拡大体の構成の変更に伴い distortion map

$$\psi(x,y) = (x+s, y+t_2x^2+t_1x+t), \quad t_1 = w^2+1, \quad t_2 = s+w^2+w+1$$

の表現も変わっている．これにより α の表現は以下の様にかける．$\alpha = (y+b_4(x))^2 \circ \psi$ だから，

$$(y+b_4(x))\circ\psi = y+t_2x^2+t_1x+t+(x+s)^3+(x_P^8+x_P^4)(x+s)^2+x_P^4(x+s)+y_P^4$$

より，

$$\begin{aligned}\alpha &= (y+t_2x^2+t_1x+t+(x+s)^3+(x_P^8+x_P^4)(x+s)^2+x_P^4(x+s)+y_P^4)^2\\ &= y^2+sx^4+(w^2+w+1)x^2+t+s+sw^2+x^6+(s+w+1)x^4+(w^2+w+s)x^2\\ &\quad +sw^2+w+(x_P^{16}+x_P^8)(x^4+w^2+w+s)+x_P^8(x^2+s+w+1)+y_P^8\end{aligned}$$

となる．整理して多項式基底 (14) を用いて表現すると

$$\begin{aligned}&(y^2+x^6+(x_P^{16}+x_P^8+1)x^4+(x_P^8+1)x^2+x_P^8+y_P^8,\\ &\qquad x^4+x_P^{16}+1, x_P^{16}+x_P^8, x^2+x_P^{16}+1, 0, 0, 1, 0, 0, 0, 0, 0, 0)\end{aligned}$$

とかける．

β の計算

$$\begin{aligned}\beta &= (y+b_8'')\circ\psi \\ &= y+t_2x^2+t_1x+t+(x_P^{32}+1)(x+s)^2+(x_P^{32}+x_P^{16})(x+s)+y_P^{16}+x_P^{16}+x_P^{48}+1 \\ &= y+(w^2+w+s+1)x^2+(w^2+1)x+t+(x_P^{32}+1)(x^2+s+w+1) \\ &\quad +(x_P^{32}+x_P^{16})(x+s)+y_P^{16}+x_P^{16}+x_P^{48}+1\end{aligned}$$

より，整理して

$$(y+x_P^{32}x^2+(x_P^{32}+x_P^{16}+1)x+y_P^{16}+x_P^{16}(x_P^{32}+x_P^{16}+1),$$
$$x^2+x_P^{32}+1, x^2+x, x^2+x_P^{16}+1, 0, 0, 1, 0, 0, 0, 0, 0)$$

を得る．

3 次拡大から始めた拡大体の基底の元の冪乗

多項式基底 (14),

$$\{1, w, w^2, s, sw, sw^2, t, tw, tw^2, st, stw, stw^2\}$$

に対し，6 次拡大から構成した拡大体 $\mathbb{F}_{2^{12m}}$ の多項式基底を成す元 w, s_0 に対する冪乗 $w^{2^{3(m-3-2i)/2}}, s_0^{2^{3(m-3-2i)/2}}$ の計算の通りに，s, t の $2^{3(m-3-2i)/2}$ 乗を考える．$s^8 = s+1$ より $s^{8^i} = s+\gamma_1(i)$ だから

$$s^{2^{3(m-3-2i)/2}} = \begin{cases} s+\gamma_1(i+1) & \text{if } m \equiv 1 \pmod 4, \\ s+\gamma_1(i) & \text{if } m \equiv 3 \pmod 4 \end{cases}$$

である．又，$w^8 = w$ より，$w^{8^i} = w$. 同様に $(w^2)^{8^i} = w^2$ が成り立つから，

$$\begin{aligned} w^{2^{3(m-3-2i)/2}} &= w \\ (w^2)^{2^{3(m-3-2i)/2}} &= w^2. \end{aligned}$$

更に，t については

$$\begin{aligned} t^8 &= t+w^2+s \\ t^{8^2} &= t+1 \\ t^{8^3} &= t+w^2+s+1 \end{aligned}$$

より，

$$t^{2^{3(m-3-2i)/2}} = \begin{cases} t+\gamma_1(i+1)(w^2+s)+\gamma_3(i+1) & \text{if } m \equiv 1 \pmod 8, \\ t+\gamma_1(i)(w^2+s)+\gamma_3(i) & \text{if } m \equiv 3 \pmod 8, \\ t+\gamma_1(i+1)(w^2+s)+\gamma_3(i-1) & \text{if } m \equiv 5 \pmod 8, \\ t+\gamma_1(i)(w^2+s)+\gamma_3(i)+1 & \text{if } m \equiv 7 \pmod 8 \end{cases}$$

となる．

α の冪乗

[5, Appendix B.2] と同様にして，α を (x_Q, y_Q) の函数と見て，α_i 最初の成分は

$$y_Q^2 + x_Q^6 + (x_P^{(6i+4)} + x_P^{(6i+3)} + 1)x_Q^4 + (x_P^{(6i+3)} + \gamma_1(i) + 1)x_Q^2 + y_P^{(6i+3)} + \gamma_1(i)x_P^{(6i+4)} + x_P^{(6i+3)} + \gamma_1(i) + \gamma_3(i)$$

で，残りが

$$(x_Q^4 + x_P^{(6i+4)} + \gamma_1(i) + 1, x_P^{(6i+4)} + x_P^{(6i+3)}, x_Q^2 + x_P^{(6i+4)} + \gamma_1(i) + 1, 0, 0, 1, 0, 0, 0, 0, 0)$$

である．α_i は第 3.2.2 節の α, β の冪乗計算の箇所で述べた様に，Miller loop の各ステップにおける α の値である．α_i の最初の成分を $2^{3(m-3-2i)/2}$ 乗すると，

$$y_Q^{((3m-7-6i)/2)} + (x_Q^{((3m-7-6i)/2)})^3 + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + 1)x_Q^{((3m-5-6i)/2)} + (x_P^{((3m-3+6i)/2)} + \gamma_1(i) + 1)x_Q^{((3m-7-6i)/2)} + y_P^{((3m-3+6i)/2)} + \gamma_1(i)x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i) + \gamma_3(i)$$

を得，これに

$$\begin{cases} \gamma_1(i+1)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i+1) & \text{if } m \equiv 1 \pmod 8, \\ \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i) & \text{if } m \equiv 3 \pmod 8, \\ \gamma_1(i+1)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i-1) & \text{if } m \equiv 5 \pmod 8, \\ \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i) + 1 & \text{if } m \equiv 7 \pmod 8 \end{cases}$$

を足し合わせたものが $\alpha_i^{2^{3(m-3-2i)/2}}$ の最初の成分になる．更に，$\alpha_i^{2^{3(m-3-2i)/2}}$ の残りの項は，

$$\begin{cases} (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 1 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1+\gamma_1(i+1))s + t, \\ \\ (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 3 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1+\gamma_1(i))s + t, \\ \\ (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 5 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1+\gamma_1(i+1))s + t, \\ \\ (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 7 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1+\gamma_1(i))s + t \end{cases}$$

$$= \begin{cases} (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 1 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)})s + t, \\ \\ (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 3 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t, \\ \\ (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 5 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)})s + t, \\ \\ (x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 7 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t \end{cases}$$

となる.

β の冪乗

α_i と同様に考える．$\beta_i(x_Q, y_Q)$ のベクトル表示について，最初の成分は

$$\begin{aligned} & y_Q + (x_P^{(6i+5)} + \gamma_1(i))x_Q^2 + (x_P^{(6i+5)} + x_P^{(6i+4)} + 1)x_Q \\ & \qquad + y_P^{(6i+4)} + \gamma_1(i)x_P^{(6i+5)} + \gamma_3(i) + (x_P^{(6i+4)} + \gamma_1(i))(x_P^{(6i+5)} + x_P^{(6i+4)} + 1) \end{aligned}$$

$$\begin{aligned} & = y_Q + (x_P^{(6i+5)} + \gamma_1(i))x_Q^2 + (x_P^{(6i+5)} + x_P^{(6i+4)} + 1)x_Q \\ & \qquad + y_P^{(6i+4)} + x_P^{(6i+4)}(x_P^{(6i+5)} + x_P^{(6i+4)} + \gamma_1(i) + 1) + \gamma_1(i) + \gamma_3(i) \end{aligned}$$

で，残りが

$$(x_Q^2 + x_P^{(6i+5)} + \gamma_1(i) + 1, x_Q^2 + x_Q, x_Q^2 + x_P^{(6i+4)} + \gamma_1(i) + 1, 0, 0, 1, 0, 0, 0, 0, 0)$$

である．β_i の最初の成分を $2^{3(m-3-2i)/2}$ 乗すると，

$$y_Q^{((3m-9-6i)/2)} + (x_P^{((3m+1+6i)/2)} + \gamma_1(i))x_Q^{((3m-7-6i)/2)} + (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)x_Q^{((3m-9-6i)/2)} + y_P^{((3m-1+6i)/2)} + x_P^{((3m-1+6i)/2)}(x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_1(i) + \gamma_3(i)$$

が得られ，これに

$$\begin{cases} \gamma_1(i+1)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i+1) & \text{if } m \equiv 1 \pmod 8, \\ \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i) & \text{if } m \equiv 3 \pmod 8, \\ \gamma_1(i+1)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i-1) & \text{if } m \equiv 5 \pmod 8, \\ \gamma_1(i)(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1) + \gamma_3(i) + 1 & \text{if } m \equiv 7 \pmod 8 \end{cases}$$

を足し合わせたものが $\beta_i^{2^{3(m-3-2i)/2}}$ の 1 の成分となる．更に，$\beta_i^{2^{3(m-3-2i)/2}}$ の残りの項は，

$$\begin{cases} (x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 1 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + \gamma_1(i+1))s + t, \\ \\ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 3 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + \gamma_1(i))s + t, \\ \\ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 5 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + \gamma_1(i+1))s + t, \\ \\ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i) + 1)w \\ +(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 7 \pmod 8 \\ +(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i) + 1 + \gamma_1(i))s + t \end{cases}$$

$$
= \begin{cases}
(x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i)+1)w & \\
+(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 1 \pmod 8 \\
+(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)})s + t, & \\
\\
(x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i)+1)w & \\
+(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 3 \pmod 8 \\
+(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t, & \\
\\
(x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i)+1)w & \\
+(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i+1))w^2 & \text{if } m \equiv 5 \pmod 8 \\
+(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)})s + t, & \\
\\
(x_Q^{((3m-7-6i)/2)} + x_P^{((3m+1+6i)/2)} + \gamma_1(i)+1)w & \\
+(x_Q^{((3m-7-6i)/2)} + x_Q^{((3m-9-6i)/2)} + \gamma_1(i))w^2 & \text{if } m \equiv 7 \pmod 8 \\
+(x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t &
\end{cases}
$$

となる.

5.1.3 アルゴリズムの詳細

本節では，第 3.2.4 節と同様の形で，拡大体の構成を変更した場合の η_T ペアリングのアルゴリズムを擬似コードの形で示す．本研究では $m = 103,\ 511$ として基礎体のパラメタとして取り並列実装の実験を行っている，このとき，$m \equiv 3 \pmod 4$, $m \equiv 7 \pmod 8$ より，$\alpha_i^{2^{3(m-3-2i)/2}}, \beta_i^{2^{3(m-3-2i)/2}}$ は以下の形になる．

まず，$\alpha_i^{2^{3(m-3-2i)/2}}$ の最初の成分は

$$
y_Q^{((3m-7-6i)/2)} + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + 1)x_Q^{((3m-5-6i)/2)} \\
+(x_P^{((3m-3+6i)/2)}+1+x_Q^{((3m-5-6i)/2)})x_Q^{((3m-7-6i)/2)}+y_P^{((3m-3+6i)/2)}+x_P^{((3m-3+6i)/2)}+\gamma_1(i)+1
$$

となり，残りの部分は

$$
(x_Q^{((3m-5-6i)/2)} + x_P^{((3m-1+6i)/2)} + \gamma_1(i)+1)w + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)})w^2 \\
+ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + sw^2 + t
$$

となる．

又，$\beta_i^{2^{3(m-3-2i)/2}}$ の最初の成分は，

$$y_Q^{((3m-9-6i)/2)} + (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)x_Q^{((3m-9-6i)/2)}$$
$$+y_P^{((3m-1+6i)/2)}+x_P^{((3m+1+6i)/2)}(x_Q^{((3m-7-6i)/2)}+x_P^{((3m-1+6i)/2)}+1)+x_P^{((3m-1+6i)/2)}+\gamma_1(i)+1$$

となり，残りの部分は

$$(x_Q^{((3m-7-6i)/2)}+x_P^{((3m+1+6i)/2)}+\gamma_1(i)+1)w+(x_Q^{((3m-7-6i)/2)}+x_Q^{((3m-9-6i)/2)}+\gamma_1(i))w^2$$
$$+ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t$$

となる．これより，η_T pairing のアルゴリズムは，[5, §7.3, Algorithm4] と同様にして Algorithm 6 として記述出来る．

Algorithm 6 拡大体の構成を 3 次拡大から始めた η_T pairing ($m \equiv 7 \pmod 8$)

INPUT: $P = (x_P, y_P),\ Q = (x_Q, y_Q) \in \mathrm{Jac}_C(\mathbb{F}_{2^m})$

OUTPUT: $f \in \mathbb{F}_{2^{12m}}$

$x_1[i] \leftarrow x_P^{2^i},\ y_1[i] \leftarrow y_P^{2^i},\ x_2[i] \leftarrow x_Q^{2^i},\ y_2[i] \leftarrow y_Q^{2^i}\ (0 \le i \le m-1)$
$f \leftarrow 1$
for $i = 0$ TO $(m-3)/2$ **do**
    以下，k_j は $\pmod m$ で考える．
    $k_1 \leftarrow (3m-9-6i)/2,\ k_2 \leftarrow k_1+1,\ k_3 \leftarrow k_2+1$
    $k_4 \leftarrow (3m-3-6i)/2,\ k_5 \leftarrow k_4+1,\ k_6 \leftarrow k_5+1$

    Compute $\alpha \leftarrow a + bw + cw^2 + ds + t$
    $c \leftarrow x_1[k_4] + x_1[k_5]$
    $a \leftarrow y_2[k_2] + (c+1)x_2[k_3] + (x_1[k_4] + 1 + x_2[k_3]) \cdot x_2[k_2] + y_1[k_4] + x_1[k_4] + \gamma_1(i)$
    $b \leftarrow x_2[k_3] + x_1[k_5] + \gamma_1(i) + 1$
    $d \leftarrow x_2[k_2] + x_1[k_5] + 1$

    Compute $\beta \leftarrow e + f_2 w + g w^2 + ds + t$
    $f_2 \leftarrow x_2[k_2] + x_1[k_6] + 1 + \gamma_1(i)$
    $e \leftarrow y_2[k_1] + (x_1[k_6] + x_1[k_5] + 1)x_2[k_1] + y_1[k_5] + d x_1[k_6] + x_1[k_5] + \gamma_1(i)$
    $g \leftarrow x_2[k_2] + x_2[k_1] + \gamma_1(i)$

    $f \leftarrow f \cdot (\alpha \cdot \beta)$
end for

$x_P \leftarrow x_1[100] + 1$
$y_P \leftarrow y_1[100] + x_1[101]$

Perform the final doublings / addition
$u \leftarrow y_2[0] + x_2[1] \cdot (1 + x_2[0] + x_P^8 + x_P^4) + x_P^4 x_2[0] + y_P^4 + x_P^8 + x_P^4 + 1$
$f \leftarrow f^4 \cdot (u, x_2[1] + x_2[0] + x_P^8 + x_P^4 + 1, x_2[1] + x_2[0], x_P^8 + x_2[0], 1, 0, 1, 0, 0, 0, 0, 0)$

Perform the final exponentiation
$f \leftarrow f^{(2^{6m}-1)(2^{3m} - 2^{4m} 2^{(m+1)/2} - 1)}$
return f

5.1.4 拡大体上の乗算の計算コスト

先行研究の拡大体の構成において，第 3.3 節で乗算 $\alpha\beta$ の計算コストを基礎体の加算，乗算，拡大体の加算，或いは 3 次拡大体における Karatsuba 乗算のコストを指標として具体的に述べた．本節では，先の第 5.1.2 節で詳しく述べた α, β の形から，第 3.3 節と同様にして $\alpha\beta$ の計算コストを見積もる．更に，本提案の拡大体の構成によって，$\alpha\beta$ をより小さなコストで計算できることを明示する．

第 5.1.1 節で一度具体的に述べたが，ここで 3 次拡大から始める拡大体 $\mathbb{F}_{2^{12m}}$ の構成について再掲する．$\mathbb{F}_{2^{3m}}$ を $\mathbb{F}_{2^m}$ 上の既約多項式 $x^3 + x + 1$ を用いて構成し，$\mathbb{F}_{2^{6m}} \simeq \mathbb{F}_{2^{3m}}[y]/(y^2 + y + w + 1),\ (w^3 + w + 1 = 0)$ として，12 次拡大体 $\mathbb{F}_{2^{12m}} \simeq \mathbb{F}_{2^{6m}}[z]/(z^2 + z + s + sw^2),\ (s^2 + s + w + 1 = 0)$ を構成し，多項式基底

$$\{1, w, w^2, s, sw, sw^2, t, tw, tw^2, st, stw, stw^2\}$$

を用いて元を表す．この基底を ***stw*-basis** と呼び，拡大体の元 $a \in \mathbb{F}_{2^{12m}}$ について stw-basis を用いて表したものを a_{stw} とかく．

本研究では基礎体の拡大次数 m として $m = 103, 511$ の場合の実装実験を行う．従って，ここでは特に $m \equiv 7 \pmod 8$ の場合について考える（どの場合においてもコスト評価に影響はない）．第 5.1.3 節より，Miller loop の各ステップにおける $\alpha_i^{2^{3(m-3-2i)/2}}$ の最初の成分は

$$y_Q^{((3m-7-6i)/2)} + (x_P^{((3m-1+6i)/2)} + x_P^{((3m-3+6i)/2)} + 1)x_Q^{((3m-5-6i)/2)}$$
$$+(x_P^{((3m-3+6i)/2)}+1+x_Q^{((3m-5-6i)/2)})x_Q^{((3m-7-6i)/2)}+y_P^{((3m-3+6i)/2)}+x_P^{((3m-3+6i)/2)}+\gamma_1(i)+1$$

で，残りは

$$(x_Q^{((3m-5-6i)/2)}+x_P^{((3m-1+6i)/2)}+\gamma_1(i)+1)w+(x_P^{((3m-1+6i)/2)}+x_P^{((3m-3+6i)/2)}+\gamma_1(i))w^2$$
$$+ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t$$

であり，$\beta_i^{2^{3(m-3-2i)/2}}$ の最初の成分は，

$$y_Q^{((3m-9-6i)/2)} + (x_P^{((3m+1+6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)x_Q^{((3m-9-6i)/2)}$$
$$+y_P^{((3m-1+6i)/2)}+x_P^{((3m+1+6i)/2)}(x_Q^{((3m-7-6i)/2)}+x_P^{((3m-1+6i)/2)}+1)+x_P^{((3m-1+6i)/2)}+\gamma_1(i)+1$$

で，残りの部分は

$$(x_Q^{((3m-7-6i)/2)}+x_P^{((3m+1+6i)/2)}+\gamma_1(i)+1)w+(x_Q^{((3m-7-6i)/2)}+x_Q^{((3m-9-6i)/2)}+\gamma_1(i))w^2$$
$$+ (x_Q^{((3m-7-6i)/2)} + x_P^{((3m-1+6i)/2)} + 1)s + t$$

である．

以下では，簡単に $\alpha_i^{2^{3(m-3-2i)/2}}, \beta_i^{2^{3(m-3-2i)/2}}$ を α, β とかく．更に，α, β のそれぞれの各係数を

$$\begin{aligned}\alpha &= (a_0, a_1, a_2, d, 0, 0, 1, 0, 0, 0, 0, 0) = a_0 + a_1 w + a_2 w^2 + cs + t \\ \beta &= (b_0, b_1, b_2, d, 0, 0, 1, 0, 0, 0, 0, 0) = b_0 + b_1 w + b_2 w^2 + cs + t\end{aligned}$$

(但し，$a_i, b_i \in \mathbb{F}_{2^m}$, $(i = 0, 1, 2)$) と簡略化してかく．このとき，

$$f = (a_0, a_1, a_2, d, 0, 0),\ g = (b_0, b_1, b_2, d, 0, 0)$$

とすると，

$$\alpha\beta = (f, 1)(g, 1) = (fg, f + g, 1)$$

より，乗算については $\mathbb{F}_{2^{6m}}$ 上の積 fg に対して計算コストを考える．基底の元の内の s, sw, sw^2 の成分には 0 が多く現れていることから，実際には以下のコストで計算できる．

$$\begin{aligned}f_0 = (a_0, a_1, a_2),\ f_1 = (d, 0, 0) \\ g_0 = (b_0, b_1, b_2),\ g_1 = (d, 0, 0)\end{aligned}$$

とすると，

$$\begin{aligned}fg &= (f_0 g_0, f_0 g_1 + f_1 g_0, f_1 g_1) \\ &= f_1 g_1 y^2 + (f_0 g_1 + f_1 g_0) y + f_0 g_0 \pmod{y^2 + y + w + 1}\end{aligned}$$

である．

以下で用いる計算コストを評価する指標 $A,\ M,\ A_{\text{ext}},\ K3$ は，第 3.3 節の始めに定義したものである．

- $f_0 g_0$ の計算．3 次の Karatsuba method を用いて乗算を行うため，$K3$ のコスト（乗算に関しては $3M$ のコスト）が掛かる．$\pmod{w^3 + w + 1}$ により $4A$, 又は，$2A_{\text{ext}}$ 掛かる．

- $f_0g_1 + f_1g_0$ の計算．今の場合，α, β の項 s の係数は一致している $(f_1 = g_1 = (c, 0, 0))$ ので，$(f_0 + g_0)(c, 0, 0)$ として計算すると，コストは $3A + 3M$, 又は，$A_{\text{ext}} + 3M$ である．

- f_1g_1 の計算．$(d, 0, 0)(d, 0, 0) = (d^2, 0, 0)$ については，

$$d = x_Q^{((3m-7+6i)/2)} + x_P^{((3m-1+6i)/2)} + 1$$

より

$$d^2 = x_Q^{((3m-5+6i)/2)} + x_P^{((3m+1+6i)/2)} + 1$$

だから，η_T ペアリングアルゴリズムの事前計算の結果を用いて，コスト A で計算できる．

最後に，$f_1g_1y^2+(f_0g_1+f_1g_0)y+f_0g_0 \pmod{y^2+y+w+1}$ の計算の際，$f_1g_1(w+1) = d^2w + d^2$ のコストは無視でき，コスト $3A$, 又は，A_{ext} で mod 演算が完了する．

全体として，fg はコスト

$$K3 + 4A + 3A + 3M + A + 3A = K3 + 3M + 11A$$

又は A_{ext} を用いて，

$$K3 + 2A_{\text{ext}} + A_{\text{ext}} + 3M + A + A_{\text{ext}} = K3 + 3M + 4A_{\text{ext}} + A$$

で計算できる．

更に，

$$\alpha\beta = (f, 1)(g, 1) = (fg, f + g, 1)$$

において，

$$f + g = (a_0 + b_0, a_1 + b_1, a_2 + b_2, 2c, 0, 0) = (f_0 + g_0, 0, 0, 0)$$

は先の $f_0g_1 + f_1g_0$ の計算の際求めており，コスト計算の必要はない．多項式 $z^2 + z + s + sw^2$ による剰余に掛かる計算コストは実質無視出来る．

従って，

$$\mathbb{F}_{2^{12m}} = \langle 1, w, w^2, s, sw, sw^2, t, tw, tw^2, st, stw, stw^2 \rangle$$

のとき $\alpha\beta$ は $K3 + 3M + 11A$, 又は，$K3 + 3M + 4A_{\text{ext}} + A$ で計算される．

先行研究における拡大体の構成においては $\alpha\beta$ の計算コストは $K3 + 7M + 14A$ 又は，$K3 + 7M + 3A_{\text{ext}} + 3A$ であった．従って，拡大体を 3 次拡大から始めて逐次拡大により構成することにより，$\alpha\beta$ を $4M + 3A$，或いは，$4M + 2A - A_{\text{ext}}$ のコストを削減して計算できることが分かった．

5.2 並列実装

本節では，GPU を用いた η_T ペアリングの並列実装について述べる．最初に 5.2.1 節において，有限体，その拡大体上の算術，ペアリングに関する並列実装について実装方針と並列化の概要を述べる．又，GPGPU によるペアリングに関する暗号実装について簡単に述べる．更に，本研究で並列実装を行った実装環境を記しておく．5.2.2 節では，有限体とその拡大体上の算術，特に加算，乗算の並列化について詳細に述べ，それぞれの算術の実行時間，並列化による実行時間と逐次計算による実行時間との比率を示す．更に，5.2.3 節において，本論文の η_T ペアリング計算において必要となる 12 次拡大体上の乗算に関して，実際に η_T ペアリングのアルゴリズムの中で計算される形の元に対する乗算の並列化について，並列化の詳細と実行結果を述べる．最後に 5.2.4 節において，η_T ペアリングの並列化について，基礎体のサイズによる並列化方針や有限体の基底変換の適用方法などを述べる．

5.2.1 並列実装方針の概要

この節では，本研究における η_T ペアリング，又，ペアリングの計算に関する基礎体，拡大体の算術の並列実装について，実装方針と概要を述べる．

本研究において，並列実装は GPU を汎用演算に用いる GPGPU によるものとし，GPU 上でのソフトウェア実装として計算シミュレーションを行った．今日 GPGPU 技術を用いた研究が盛んに行われ，暗号の分野でも並列処理により，暗

号化，復号における処理時間を劇的に短縮することが可能となってきている．ペアリングについては，楕円曲線上の高速なペアリング暗号の GPU 実装として加藤らの研究 [13] がある．彼らの研究では基礎体の元について，データ構造を並列化し，基礎体の基本演算の高速化と最適な bit 演算により，ペアリング全体の劇的な高速演算を実現している．詳細は後の第 5.2.2 節で述べるが，本研究では，基礎体，或いは拡大体のパラメタを可能な限り任意に取り，その上の算術を並列化出来るようなアルゴリズムを対象にする．従って，それぞれの体の標数，拡大次数に特化したアルゴリズムについては考察の対象としない．

ペアリングの並列化について，例えば Miller loop そのものの様に，ペアリング特有のアルゴリズムの並列化が考えられるが，基本的に Miller's algorithm は逐次的に計算を進めなければならない．本研究で対象としている η_T ペアリングについては，事前計算のおかげで特別に Miller loop の各ステップにおいて独立に計算を進めることができるが，本実装で取り組む様に小標数の基礎体に対し，基礎体，或いは拡大体の算術を並列化して処理したほうが，並列化による高速化が望める．同様に，因子計算においても並列化が適用出来る場面が多いとは言えない．よって，本研究において，η_T ペアリングの並列化とは，曲線の定義体である基礎体，又，ペアリングの計算で必要になる基礎体上の拡大体の加減算，乗算を並列化し，ペアリング全体としての並列実装を行うものである．

本節の冒頭で述べた通り，本実装は実装実験の意味をもつため，基礎体，拡大体の様々なパラメタによって計算を進められるように GPGPU によるソフトウェア実装として並列実装を行っている．現在，GPGPU によるプログラミングは CUDA [1] によるものが主流である．以下に，簡単に CUDA について述べる．

CUDA

CUDA (Compute Unified Device Architecture) は 2007 年に NVIDIA が導入した，アプリケーションが CPU, GPU を統合的に実行できるよう設計されたプログラミングモデルであり，現在，GPU を汎用演算に用いる技術である GPGPU (General Purpose Graphics Processing Unit) を用いたプログラミングを達成させる開発環境として一般的なものとなっている．CUDA を用いて，C/C++による

馴染みのあるプログラミングで，GPU の性能を十分に引き出した効率的な GPU 上の実装を行うことが出来る．

実装環境

本並列実装における実装環境について述べる．本実装は前述の CUDA と C++ を用いて行っている．実装環境として用いたマシンのスペック，グラフィックボード，コンパイラ等の情報を表 4 に示す．

OS	Fedora 17
CPU	Intel Core i7-3960X, 3.30GHz, 6 Cores
Memory	DDR3-1333, 64GB
GPU	NVIDIA GeForce GTX 590, 512×2 CUDA Cores
Compiler	GCC-4.5.3, NVCC-4.2

表 4 実装環境（並列実装）

5.2.2 基礎体，拡大体上の演算の並列化

本節では，η_T ペアリングの定義体である，基礎体の，又，その拡大体の加算，乗算の並列化について具体的に述べる．

基礎体の算術の並列化

ここでは，η_T ペアリングの定義体である基礎体 $\mathbb{F}_{2^m}$ について，その有限体における加算（減算），乗算の並列実装について述べる．第 2.3.1 節で述べた様に，一般に $\mathbb{F}_q$ 上の m 次拡大体 $\mathbb{F}_{q^m}$ が，$\mathbb{F}_q$ 上の m 次既約多項式 $f(x)$ によって $\mathbb{F}_q[x]/(f(x))$ と表されている時，$\mathbb{F}_{q^k}$ の元は高々 $m-1$ 次の多項式で表され，加算，乗算はそれぞれ多項式の加算，乗算を用いて計算できる．

加藤ら [13] は多項式の加算，乗算を並列に処理することにより拡大体の高速実装を実現している．特に乗算に関して，$\mathbb{F}_3$ の元の無駄のない bit 表示 [13, §3.2,

Implementation III] と Comb method の並列化 [13, §3.2, Implementation II] により，GPU 上の並列実装として最適といえるパフォーマンスを得ている．

本実装では，基礎体，或いは，拡大体のパラメタを変更し，拡大体上の乗算や，η_T ペアリングの計算シミュレーションを目的としているため，あるパラメタに対する最適な実装を行うことは考えていない．例えば，本研究で扱う η_T ペアリングの定義体（基礎体）は binary field $\mathbb{F}_{2^m}$ であるが，本実装においてこの基礎体上の基本算術は，最適な bit 演算等によるものではない．よって本研究では基礎体の加算，減算，乗算の並列実装を以下のシンプルな方法で実装している．基礎体 $\mathbb{F}_q[x]/(f(x))$ の元

$$a(x) = \sum_{i=0} a_i x^i, \quad b(x) = \sum_{i=0} b_i x^i$$

に対し，加減算を多項式としての演算 $a(x) \pm b(x)$ により行い，それぞれの多項式 $a(x), b(x)$ の係数を並列に演算することにより並列化する．即ち，$a_i \pm b_i$ として，独立に各係数を計算し求める多項式を得る．よって，直接的に考えて並列化により $1/\deg f$ の実行時間で計算できる．

乗算に関しては，上に注意した様に直接的に並列化を行う．即ち，

$$c(x) = \sum_{k=0} c_k x^k = a(x)b(x), \quad c_k = \sum_{i+j=k} a_i b_j$$

のとき，独立に係数 c_k を並列に計算することにより多項式の乗算の並列実装を行う．拡大体の元として，mod 演算を行い $c(x) \pmod{f(x)}$ を求める必要があるが，**本実装では多項式の剰余計算は並列実装を行わず，全て CPU 上で逐次的に処理する．**これは先に注意した様に，有限体の様々なパラメタに対応するために基本的なアルゴリズムで実装を行う必要があるためで，効率的に並列化が適用出来る場合でも mod 演算に関しては逐次的に処理を行う．

拡大体上の算術の並列化

本節では，基礎体の拡大体上の加減算，乗算の並列実装について詳細に述べる．本実装では拡大体上の乗算を，基礎体の元を用いた Karatsuba method により実装し，それを並列化する．又，本節における拡大体上の乗算は，拡大体を構成す

る既約多項式による mod 演算の処理は含まず，Karatsuba method による乗算の箇所だけ説明する．拡大体上の乗算の全体の説明に関しては，本研究のペアリングパラメタである 12 次拡大体 $\mathbb{F}_{2^{12m}}$ に対象を絞って，次の 5.2.3 節で詳細に述べる．以下，基礎体を $\mathbb{F}_{2^m}$ の形のものとし，拡大体 $\mathbb{F}_{2^{km}}$ について考える．

まず，拡大体上の加算，減算の並列化について述べる．$\mathbb{F}_{2^{km}}$ の元 a は

$$a = (a_0, a_1, ..., a_{k-1}), \quad a_i \in \mathbb{F}_{2^m}$$

の様に，$\mathbb{F}_{2^{km}}$ を $\mathbb{F}_{2^m}$ 上のベクトル空間と見て，k 個の基礎体の元を用いて表されている．よって，拡大体上の元の加減算は，基礎体の元からなるベクトルの加減算を行うことになり，二種類の並列化が考えられる．即ち，ベクトルの各成分を計算する際，5.2.2 節で述べた基礎体の元の並列演算を行う方法と，その並列化に加え，ベクトル表示した拡大体の元の各成分の加減算を独立に並列に行う方法である．前者の基礎体の算術の並列化のみ適用させる方法を**基礎体並列化**，後者の拡大体のベクトル演算も並列に行う方法を**二重並列化**と呼ぶ．

次に Karatsuba method による拡大体上の乗算に関して述べる．第 2.3.3 節において Karatsuba method の詳細な説明を行った．繰り返しになるが，ここに Karatsuba method による計算式を表す．拡大体 $\mathbb{F}_{2^{km}}$ の元 $A(x), B(x)$ を基礎体上の $k-1$ 次の多項式と見た時，即ち，

$$A(x) = \sum_{i=0}^{k-1} a_i x^i, \quad B(x) = \sum_{i=0}^{d} b_i x^i.$$

に対し，各 $i = 0, ..., k-1$ に対し

$$V_i := a_i b_i, \tag{16}$$

を計算し，$0 \leq s < t \leq k-1$ に対し

$$V_{s,t} = (a_s + a_t)(b_s + b_t), \tag{17}$$

を求める．このとき，

$$
\begin{aligned}
c_0 &= V_0 \\
c_{2(k-1)} &= V_{k-1} \\
c_i &= \begin{cases}
\displaystyle\sum_{s+t=i} V_{s,t} - \sum_{s+t=i} (V_s + V_t) & i\text{: odd}, \\
\displaystyle\sum_{s+t=i} V_{s,t} - \sum_{s+t=i} (V_s + V_t) + V_{i/2} & i\text{: even}, \\
\qquad\qquad \text{for } 0 \le s < t \le k-1,\ 0 < i < 2(k-1)
\end{cases}
\end{aligned}
\tag{18}
$$

として $\displaystyle\sum_{i=0}^{2(k-1)} c_i x^i = a(x)b(x)$ が得られる．ここで，この乗算の並列化を以下の様に行う．Karatsuba method に対し並列化を行う箇所は，式 (16), (17) で表される事前計算に対して適用する．即ち，

1. 各 i に対し $V_i = a_i b_i$ を独立に並列に計算する (16).
2. 各 $0 \le s < t \le k-1$ に対し，独立に並列に $V_{s,t} = (a_s + a_t)(b_s + b_t)$ を計算する (17).
3. 求める乗算結果の多項式の係数を，それぞれ逐次的に計算する (18).

先の拡大体上の加算，減算の並列化と同様にして，この Karatsuba method の並列化と，基礎体の算術の並列化を同時に適用することが出来る，従って．乗算に関しても，基礎体の算術のみに並列化を適用させる方法を基礎体並列化，加えて Karatsuba method の事前計算の箇所に並列化を適用させたものを二重並列化と呼ぶ.

基礎体，拡大体上の演算の並列化に関するシミュレーション結果

本節の終わりに，基礎体，拡大体の基本演算の並列化について，計算シミュレーションによる実行時間結果を示す．上で説明したそれぞれの実装方式を以下の様に記す.

実装 I. GPU 上で逐次的に計算する方法.

実装 II. 基礎体 $\mathbb{F}_{2^m}$ 上の算術にのみ並列化を適用する方法.

実装 III. 二重並列化を適用する方法．即ち，拡大体上の加減算に関しては実装 II と並列ベクトル演算，乗算に関しては実装 II と Karatsuba method の並列化を適用した方法である.

又，実行時間計測について，次の用語を定義する.

- **Host 時間** それぞれの算術計算に掛かる全体の実行時間．GPU 上での処理を含む場合，GPU 側へのデータ転送など，CUDA の API による処理に掛かる時間も含めて測定する．本研究において，実行時間の計測は，ペアリングの暗号化函数としての実行時間，即ち函数値の計算時間，メモリの読み書き，転送に掛かる時間，入出力処理に掛かる全ての時間を合わせて計測した.

- **GPU 時間** 並列実装により，GPU 上で計算が行われる際に掛かる実行時間であり，GPU が持つメモリへの転送時間等を含まず，純粋に並列計算を行う際の実行時間を指す．GPU 時間の測定は，CUDA Runtime API の一つである `cudaEvent` を用いた.

CUDA による GPU 実装では，GPU 上で汎用計算を行う際，CUDA の用語で，CPU, システムメモリ等を指す host 側にあるデータを，GPU を指す device 側に転送し，GPU 上で並列処理を行い，再び演算結果を host 側へ転送するという処理が繰り返される．本実験では GPU 時間として，これらのデータ転送に掛かる時間を含めておらず，純粋に並列処理に掛かる時間を計測した.

まず，基礎体の基本演算について，計算シミュレーション結果を示す．本論文ではペアリングパラメタである基礎体として，拡大次数 $m = 103, 511$ における $\mathbb{F}_{2^{103}}, \mathbb{F}_{2^{511}}$ についてシミュレーションを行った．表 5,6 において，基礎体上の多項式をランダムに取り，それらの加算，乗算一回に掛かる実行時間の平均値を示した.

ここで，基礎体の基本演算に関するシミュレーション結果について，実装 I のシリアル実装と，実装 II の並列実装の実行時間の比較と，基礎体の拡大次数 m

拡大次数	実装 I	実装 II	高速化率（実装 I / 実装 II）
$m = 103$	95.84 (40.46)	58.04 (6.55)	1.651 (6.177)
$m = 511$	228.29 (167.592)	61.61 (6.47)	3.705 (25.903)

表 5　基礎体 $\mathbb{F}_{2^m}$ 上の多項式の加算の Host 時間（括弧内は GPU 時間）100 回平均 (μs)

拡大次数	実装 I	実装 II	高速化率（実装 I / 実装 II）
$m = 103$	4119.65 (4065.86)	79.73 (34.11)	51.670 (119.198)
$m = 511$	99141.2 (99080.2)	274.82 (225.52)	360.750 (439.341)

表 6　基礎体 $\mathbb{F}_{2^m}$ 上の多項式の乗算の Host 時間（括弧内は GPU 時間）100 回平均 (μs)

による実行結果の比較を行う．図 6 において，基礎体の加算に関する実行時間について表 5 を基にシリアル，並列実装の比較を行った．加算に関して，並列化による高速化度合いは単純に $1/m$ に近い値を取らないが，今回並列実装に用いた GPU のコア数は比較的多いため，$m = 103, 511$ のどちらの場合も，同程度の時間で並列処理が行われている．

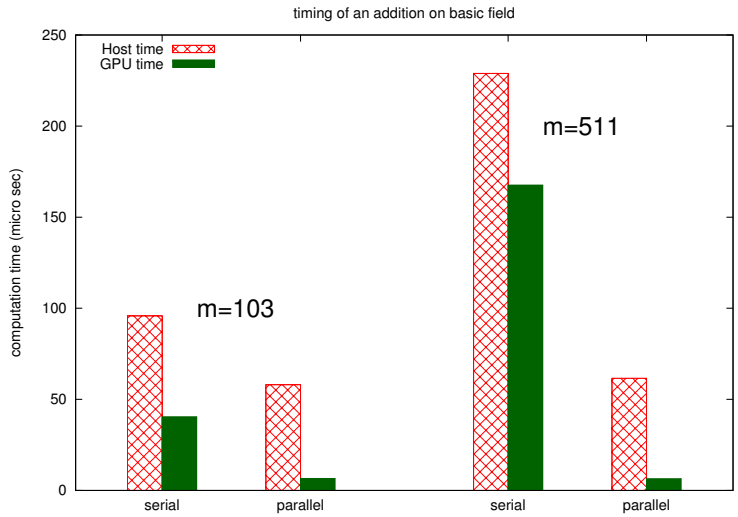

図 6　基礎体 $\mathbb{F}_{2^m}$ 上の加算における並列化による高速化の比較

加算の場合と同様にして，基礎体の乗算について，表 6 を用いて比較のグラフを図 7 に示した．乗算は加算に比べ，m が大きい程，m のサイズの二乗のオーダで実行時間が増加するが，並列化による効果も大きく，小標数による定義体を利用することで，定義体の算術の並列化により，ペアリング等の定義体の算術を繰り返す函数に対する並列化の効果が，大きく見込めることが分かる．

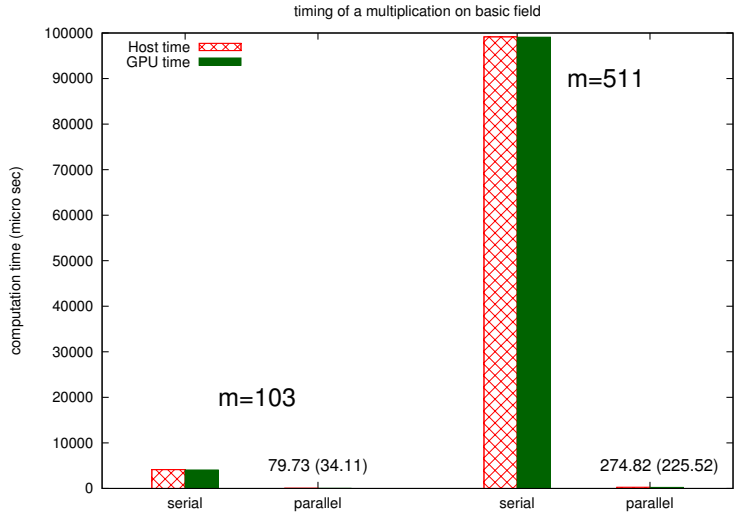

図 7　基礎体 $\mathbb{F}_{2^m}$ 上の乗算における並列化による高速化の比較

表 5,6 より，本実装における，基礎体の加算，乗算の GPU 上並列実装では，host から device 側へのデータ転送に掛かる時間は約 50 μs 程度であることが分かる．

ここで，基礎体の乗算については，多項式の乗算の後に基礎体を構成する既約多項式による mod 演算が必要だが，CPU の処理による mod 演算に掛かる実行時間を表 7 に示した．基礎体の元を掛けた後，高々$2(m-1)$ 次の多項式が得られるが，mod 演算に関しては $2(m-1)$ 次の多項式を m 次の既約多項式で割ることを想定し，実行時間の平均値を測定した．

拡大次数	$2(m-1)$ 次の多項式の m 次の多項式による mod 演算実行時間
$m = 103$	72.23
$m = 511$	3494.26

表 7　基礎体 $\mathbb{F}_{2^m}$ の定義多項式による剰余に掛かる実行時間 100 回平均 (μs)

次に，拡大体上の加算（減算）の並列実装の実行時間シミュレーション結果を表 8 に示す．ここでは，本研究で扱う η_T ペアリングアルゴリズムで計算される 3

次拡大体と 6 次拡大体の加減算について見る．基礎体のサイズに依らず，3 次拡大，6 次拡大のどちらの場合においてもほぼ同程度の時間で計算が行われており，又，6 次拡大体において 3 次拡大体と比べ実行時間はさほど大きくないため，二重並列化による加算は有効的であることが分かる．

基礎体の拡大次数	3 次拡大体 ($k = 3$)	6 次拡大体 ($k = 6$)
$m = 103$	180.15 (8.58)	239.67 (8.44)
$m = 511$	190.97 (8.81)	253.79 (8.47)

表 8　拡大体 $\mathbb{F}_{2^{km}}$ 上の加算（減算）の並列実装 Host 時間（括弧内は GPU 時間）100 回平均 (μs)

最後に Karatsuba method を用いた拡大体上の乗算について，並列実装による実行時間結果を表 9,10 に示す．

拡大次数	実装 I	実装 II	実装 III	高速化率（実装 I / 実装 III)
$k = 2$	10.601 (10.404)	1.641 (0.148)	0.524 (0.095)	20.231 (109.516)
$k = 3$	21.213 (20.943)	4.329 (0.328)	0.800 (0.114)	26.516 (183.711)
$k = 6$	76.059 (75.512)	20.414 (1.328)	3.367 (0.375)	22.590 (201.365)
$k = 12$	267.434 (266.812)	89.085 (5.446)	42.555 (1.731)	6.284 (154.137)

表 9　基礎体 $\mathbb{F}_{2^{103}}$ の拡大体上の Karatsuba method による乗算の Host 時間（括弧内は GPU 時間）100 回平均 (ms)

拡大次数	実装 I	実装 II	実装 III	高速化率（実装 I / 実装 III)
$k = 2$	274.139 (273.8)	2.271 (0.746)	0.957 (0.495)	286.457 (553.131)
$k = 3$	576.202 (575.1)	5.612 (1.158)	1.255 (0.513)	459.125 (1121.053)
$k = 6$	2163.33 (2161.47)	25.051 (5.512)	4.254 (0.974)	508.540 (2219.168)
$k = 12$	5961.31 (5960.57)	106.898 (20.905)	45.789 (3.499)	130.190 (1703.507)

表 10　基礎体 $\mathbb{F}_{2^{511}}$ の拡大体上の Karatsuba method による乗算の Host 時間（括弧内は GPU 時間）100 回平均 (ms)

今回，考察している η_T ペアリングにおいて基礎体の 12 次拡大体 $\mathbb{F}_{2^{12m}}$ について考える．$\mathbb{F}_{2^{12m}}$ の構成について，2 次拡大，3 次拡大，6 次拡大から始め逐次拡大を行う，或いは，直接 12 次拡大を行うという構成が考えられるが，例えば 3 次拡大から始めると，残り 2 次拡大を二度繰り返せば良いため，12 次拡大体上の多項式の乗算として，3 次拡大体上の Karatsuba method を 9 度行えば良い．表 9 より 3 次拡大から始めると，2 次，6 次から始める，或いは，12 次拡大体を直接構成するより，拡大体上の乗算が二重並列化を行う実装 III において速く行えることが分かる．例えば，6 次拡大から始めると，残り 2 次拡大を行うため，12 次拡大体上の多項式の乗算としては 6 次拡大体上の Karatsuba method を 3 回行う必要があるが，この実行時間は 3 次の Karatsuba method の実行時間を 9 倍したものより大きい．他の場合も同様の理由で実行時間はより大きい．従って，$m = 103$ のとき，$\mathbb{F}_{2^{12m}}$ の構成として，3 次拡大から始めて，2 次拡大を繰り返す構成が乗算の効率が良い．

同様の理由で $m = 511$ の時，表 10 より，Host 時間については 3 次拡大から始める構成が乗算の実行時間が一番小さいが，GPU 時間については 6 次拡大から始めた方が実行時間が小さい．これは基礎体の拡大次数が大きいとき，並列化による高速化の度合いがより高くなるためである．

$m = 103$ の時の Karatsuba 乗算に対する並列実装に関して，表 9 を基にして，図 8 において比較のためのグラフを示した．先に注意した様に，並列実装において，3 次の Karatsuba 乗算，或いは 6 次の Karatsuba 乗算は，12 次拡大体上の元の乗算を考えたとき，12 次の Karatsuba 乗算に比べ，比較的大きな優位性を持つことが見て取れる．

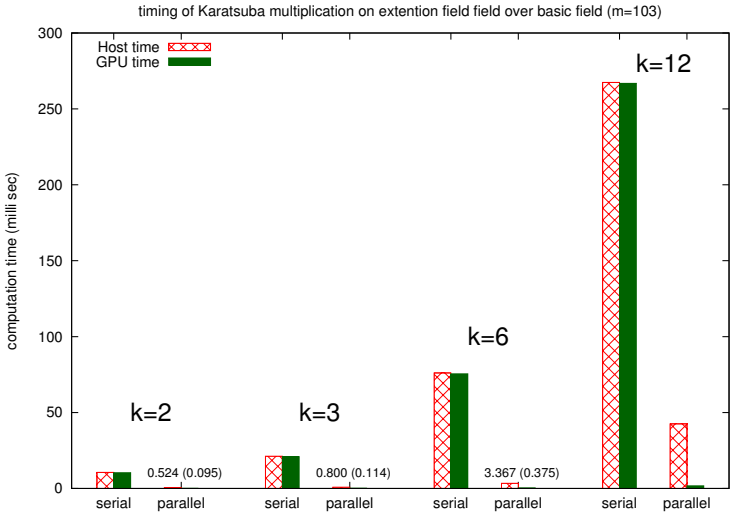

図 8　基礎体 $\mathbb{F}_{2^{103}}$ の拡大体上の，Karatsuba method を用いた乗算の並列化による高速化の比較

更に，$m = 511$ の時も同様にして，比較のグラフを図 9 において表 10 を元に描いた．12 次の Karatsuba method については，並列化による高速化度合いが他の拡大次数に比べて低い値を取っており，m が大きいほどそれが顕著となる．これは，Karatsuba method の並列化を事前計算の部分を対象箇所としており，次数が増えるほど事前計算後の，乗算後の多項式の係数を求める処理が大幅に大きくなってくるためである．よって 12 次，或いはより高次の Karatsuba 乗算は，並列化による高速度合いも低下し，又，計算コストも再帰的に Karatsuba 乗算を適用させる場合と比べ，より大きくなる．シミュレーション結果より，3 次，或いは 6 次の Karatsuba 乗算がバランスが取れており，並列化に適していると言える．

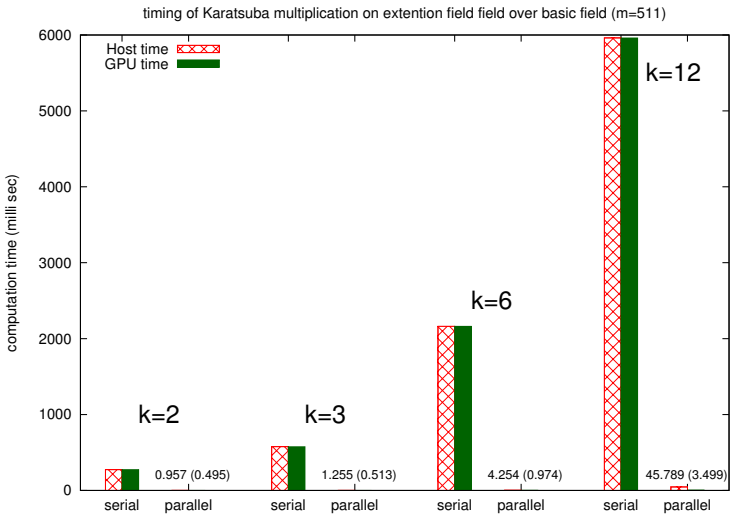

図 9　基礎体 $\mathbb{F}_{2^{511}}$ の拡大体上の，Karatsuba method を用いた乗算の並列化による高速化の比較

5.2.3　12 次拡大体上の乗算の並列実装

本節では，η_T ペアリングの計算において必要となる 12 次拡大体の乗算について，先の第 5.2.2 節で述べた，Karatsuba method による乗算を用いた並列実装と拡大体の構成による実行時間の比較を行う．まず，基礎体 $\mathbb{F}_{2^m}$ の 12 次拡大体 $\mathbb{F}_{2^{12m}}$ の乗算の並列実装について，実行時間のシミュレーションを行い，拡大体の構成に関して実行時間の比較を行う．又，η_T ペアリングのアルゴリズムでは，最終冪等で少なからず拡大体の乗算が二乗算として計算される．従って，二乗算の実行時間のシミュレーションも行う．更に，η_T ペアリングの Miller loop で拡大体の乗算として行われる乗算 $\alpha\beta$ の並列実装について，実行時間の比較，シミュレーション結果を示す．

一般の元による乗算の並列実装

第 5.2.2 節で述べた様に，12 次拡大体 $\mathbb{F}_{2^{12m}}$ の Karatsuba method を用いた乗

算の並列実装において，$\mathbb{F}_{2^{12m}}$ の構成方法により，実行時間に差があった．又，第 5.2.2 節では乗算結果として mod 演算を行なっておらず，拡大体の元として表されていない状態の実行時間を述べた．ここでは，$\mathbb{F}_{2^{12m}}$ の元としての乗算結果を得るために掛かる実行時間を示す．$\mathbb{F}_{2^{12m}}$ について，6 次，或いは 3 次拡大から始める構成において，第 5.2.2 節にある様に，それぞれの拡大次数の Karatsuba method を基本演算として用いて乗算を行うため，表 9,10 から乗算に掛かる実行時間は見積りやすいが，それぞれの拡大体における乗算後の mod 演算については計算方法が異なるため，実行時間のシミュレーション結果により拡大体の構成について比較，検討する．ここでは，$\mathbb{F}_{2^{12m}}$ の構成は第 3.1, 5.1 節で述べた様に元の構成である 6 次拡大から始め 2 次拡大するものと，3 次拡大から始め 2 次拡大を二度行うものについてシミュレーションを行った．

初めに，Karatsuba method を用いた 12 次拡大体上の乗算について，その乗算アルゴリズムにおいて，並列処理を行っている箇所と計算コストを明らかにし，擬似コードを用いて具体的に説明する．各算術において，式の後ろに “[並列]” と記したところは GPU 上で並列処理を行う箇所である．まず，Algorithm 7 により，6 次拡大から構成した拡大体と，s_0w-basis を用いて拡大体の元を表現した時の拡大体上の乗算の流れを示す．ここで，コストを表す指標として，$K6$ は 6 次の Karatsuba 乗算に掛かるコスト，Sub_{ext} は拡大体，ここでは 6 次拡大体上の減算に掛かるコストを表す．又，最後のステップ 12 における mod 演算では，$E_1F_1(-w^5 - w^3)$ の計算が必要だが，実際には 6 次拡大体上の乗算でなく基礎体の加算を数回行うことで計算できる．

Algorithm 7 6 次拡大から始めた構成の拡大体 $\mathbb{F}_{2^{12m}}$ 上の乗算

INPUT: $A = (E_0, E_1),\ B = (F_0, F_1) \in \mathbb{F}_{2^{12m}}$

OUTPUT: $C = AB \in \mathbb{F}_{2^{12m}}$

1: 6 次の Karatsuba 乗算により E_0F_0 を計算する [並列] cost: $K6$

2: $E_0F_0 \pmod{x^6 + x^5 + x^3 + x^2 + 1}$

3:

4: 6 次の Karatsuba 乗算により E_1F_1 を計算する [並列] cost: $K6$

5: $E_1F_1 \pmod{x^6 + x^5 + x^3 + x^2 + 1}$

6:

7: $E_0 + E_1,\ F_0 + F_1$ [並列] cost: $2A_{\text{ext}}$

8: 6 次の Karatsuba 乗算により $(E_0 + E_1)(F_0 + F_1)$ を計算する [並列] cost: $K6$

9: $(E_0 + E_1)(F_0 + F_1) \pmod{x^6 + x^5 + x^3 + x^2 + 1}$

10:

11: $(E_0 + E_1)(F_0 + F_1) - E_0F_0 - E_1F_1$ [並列] cost: $2Sub_{\text{ext}}$

12: $E_1F_1y^2 + (E_0F_1 + E_1F_0)y + E_0F_0 \pmod{y^2 + y - w^5 - w^3}$ [並列]

13: cost: $K6$ (sparse) $+ 2Sub_{\text{ext}}$

次に，Algorithm 8 により，3 次拡大から構成した拡大体と，stw-basis を用いて拡大体の元を表現した時の拡大体上の乗算の流れを示す．記号は Algorithm 7 と同様の使い方による．stw-basis による拡大の構成では，3 次の Karatsuba 乗算を行う度に，定義多項式 $x^3 + x + 1$ による mod 演算を行うと，mod 演算のコストが大きくなり効率的でない．従って，まとめて処理出来るところまで $x^3 + x + 1$ による mod 演算処理を先延ばしにすることで，効率的に乗算を行える様実装した．一方で，s_0w-basis においては，定義多項式である既約 6 次式での mod 処理を後に先延ばしにしても効果は特にない．

Algorithm 8 3 次拡大から始めた構成の拡大体 $\mathbb{F}_{2^{12m}}$ 上の乗算

INPUT: $A = (E_0, E_1),\ B = (F_0, F_1) \in \mathbb{F}_{2^{12m}}$

$E_0 = (P_0, P_1),\ E_1 = (P_2, P_3),\ F_0 = (Q_0, Q_1),\ F_1 = (Q_2, Q_3)$

OUTPUT: $C = AB \in \mathbb{F}_{2^{12m}}$

1: E_0F_0 の計算

2: 3 次の Karatsuba 乗算により P_0Q_0, P_1Q_1 を計算する [並列] cost: $2K3$

3: **$(\mathbf{mod}\ x^3 + x + 1)$ の処理は後にまとめて行う**

4:

5: $P_0 + P_1,\ Q_0 + Q_1$ [並列] cost: $2A_{\text{ext}}$

6: 3 次の Karatsuba 乗算により $(P_0 + P_1)(Q_0 + Q_1)$ を計算する [並列] cost: $K3$

7:

8: $(P_0 + P_1)(Q_0 + Q_1) - P_0Q_0 - P_1Q_1$ [並列] cost: $2Sub_{\text{ext}}$

9: $P_1Q_1y^2 + (P_0 + Q_1)(P_1 + Q_0)y + P_0Q_0 \pmod{y^2 + y + w + 1}$

10: E_0F_0 の計算終了

11:

12: E_1F_1 についても E_0F_0 と同様の計算を行う cost: $3K3 + 2A_{\text{ext}} + 2Sub_{\text{ext}}$

13: $E_0 + E_1 = (P_0 + P_2)(P_1 + P_3)$ [並列] cost: A_{ext}

14: $F_0 + F_1 = (Q_0 + Q_2)(Q_1 + Q_3)$ [並列] cost: A_{ext}

15:

16: E_0F_0 と同様に $(E_0 + E_1)(F_0 + F_1)$ を計算する cost: $3K3 + 2A_{\text{ext}} + 2Sub_{\text{ext}}$

17: $(E_0 + E_1)(F_0 + F_1) - E_0F_0 - E_1F_1$ [並列] cost: $2Sub_{\text{ext}}$

18:

19: **$\boldsymbol{E_0F_0}$, $\boldsymbol{(E_0 + F_1)(E_1 + F_0)}$, $\boldsymbol{E_1F_1}$ を構成する元に対し $(\mathbf{mod}\ x^3 + x + 1)$ を行う**

20: $E_1F_1z^2 + (E_0 + F_1)(E_1 + F_0)z + E_0F_0 \pmod{z^2 + z + s + sw^2}$ [並列]

21: cost: $2K3 + 2A_{\text{ext}}$

上で述べてきた拡大体上の乗算の実際のアルゴリズムに対し，並列実装結果を示す．まずは，基礎体の拡大次数が $m = 103$ の場合における $\mathbb{F}_{2^{12m}}$ の乗算の実行時間のシミュレーション結果を表 11 に示す．

拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$	実装 I	実装 III	高速化率（実装 I / 実装 III)
6 次拡大からの構成	317.926 (302.098)	27.696 (1.474)	11.479 (204.951)
3 次拡大からの構成	205.778 (188.515)	22.361 (1.003)	9.202 (187.951)

表 11 拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$ 上の乗算並列実装 Host 時間（括弧内は GPU 時間）100 回平均 (ms)

次に，基礎体の拡大次数が $m = 511$ の場合における $\mathbb{F}_{2^{12m}}$ の乗算の実行時間のシミュレーション結果を表 12 に示す．

拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$	実装 I	実装 III	高速化率（実装 I / 実装 III)
6 次拡大からの構成	9002.04 (8710.24)	306.52 (3.891)	29.369 (2238.56)
3 次拡大からの構成	5532.36 (5220.4)	313.364 (4.668)	17.639 (1118.338)

表 12 拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ 上の乗算並列実装 Host 時間（括弧内は GPU 時間）100 回平均 (ms)

更に，拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ の二乗算についても同様にシミュレーション結果を示す．本実装では二乗算について特別なアルゴリズムを使用しておらず，シミュレーションは一般の元の乗算を行う函数に，同一のオペランドを入力として与えて行ったものである．表 13,14 において，それぞれの基礎体の拡大次数における，拡大体 $\mathbb{F}_{2^{12m}}$ の二乗算の並列実装シミュレーション結果を示す．又，比較のため，表 15,16 において標数 3 の基礎体に対し，二乗算のシミュレーション結果を示す．

拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$ の構成方法	実装 I	実装 III
6 次拡大から始める	316.818 (301.942)	26.668 (1.497)
3 次拡大から始める	202.407 (188.437)	19.067 (1.035)

表 13 拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$ 上の二乗算並列実装 Host 時間（括弧内は GPU 時間） 100 回平均 (ms)

拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ の構成方法	実装 I	実装 III
6 次拡大から始める	8987.06 (8718.28)	285.122 (3.911)
3 次拡大から始める	5436.34 (5223.89)	241.833 (4.660)

表 14 拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ 上の二乗算並列実装 Host 時間（括弧内は GPU 時間） 100 回平均 (ms)

拡大体 $\mathbb{F}_{3^{12m}}$ $(m = 103)$（**標数 3**）	実装 I	実装 III
6 次拡大から始める	319.290 (302.187)	28.824 (1.524)
3 次拡大から始める	207.216 (188.456)	23.902 (1.066)

表 15 拡大体 $\mathbb{F}_{3^{12m}}$ $(m = 103)$（**標数 3**）上の二乗算並列実装 Host 時間（括弧内は GPU 時間） 100 回平均 (ms)

拡大体 $\mathbb{F}_{3^{12m}}$ $(m = 511)$（**標数 3**）	実装 I	実装 III
6 次拡大から始める	9034.055 (8710.42)	339.939 (3.932)
3 次拡大から始める	5565.13 (5220.19)	354.847 (4.696)

表 16 拡大体 $\mathbb{F}_{3^{12m}}$ $(m = 511)$（**標数 3**）上の二乗算並列実装 Host 時間（括弧内は GPU 時間） 100 回平均 (ms)

$\mathbb{F}_{2^{12m}}$ について，特に 3 次拡大から始める構成においては，一般の元の乗算に比べ実行時間が小さくなっている．今，基礎体の標数が 2 なので，例えば積 $(x+y)(x+y)$ については，項 $xy + yx = 2xy = 0$ の様に消える項があり，特に逐次拡大による Karatsuba method を再帰的に用いて乗算を行う際，0 が多く出現し，3 次拡大から始めた方の拡大体の構成では実行時間がより小さくなると考えられる．従って，標数 2 の時，拡大体の構成により，拡大体の塔において中間体の数が多い程，小

さな実行時間で二乗算が処理されると考えられる．実際，表 15,16 より，標数が 2 でない時は二乗算の実行時間と任意の元同士の乗算の実行時間に明確な差はない．

η_T ペアリングアルゴリズムにおける乗算 $\alpha\beta$ の並列化

次に，拡大体 $\mathbb{F}_{2^{12m}}$ の乗算の中でも，特に η_T ペアリングの計算における乗算 $\alpha\beta$ の並列実装のシミュレーション結果を述べる．乗算 $\alpha\beta$ については，3 次，或いは 6 次拡大から始めた拡大体の構成において計算し，第 3.3, 5.1.4 節で具体的にコストを述べた様に，拡大体の sparse な元同士による乗算になるので，一般の元の乗算に比べ高速に計算できる．表 17,18 に基礎体のそれぞれの拡大次数 $m = 103, 511$ における乗算 $\alpha\beta$ の実行時間のシミュレーション結果を示す．

基礎体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$	実装 I	実装 III	高速化率（実装 I / 実装 III)
6 次拡大からの構成	56.700 (50.156)	5.304 (0.421)	10.690 (119.135)
3 次拡大からの構成	37.334 (33.558)	3.322 (0.289)	11.238 (116.118)

表 17　拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$ 上の乗算 $\alpha\beta$ の Host 時間（括弧内は GPU 時間）100 回平均 (ms)

基礎体 $\mathbb{F}_{2^{12m}}(m = 511)$	実装 I	実装 III	高速化率（実装 I / 実装 III)
6 次拡大からの構成	1302.69 (1276.86)	26.207 (2.164)	49.708 (590.046)
3 次拡大からの構成	895.814 (879.191)	17.254 (1.391)	51.919 (632.057)

表 18　拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ 上の乗算 $\alpha\beta$ の Host 時間（括弧内は GPU 時間）100 回平均 (ms)

基礎体の拡大次数が $m = 103, 511$ どちらであっても 3 次拡大から始めた構成の方が実行時間が幾分小さく，第 3.3, 5.1.4 節で算出した，理論上のコスト計算の結果の正当性が確かめられている．

5.2.4 η_T ペアリングの並列実装

本節では，η_T ペアリングの並列実装について，これまで述べてきた拡大体上の乗算函数や，基底変換を適用させ拡大体の構成を変更する具体的な実装方法，又，実行時間のシミュレーション結果について述べる．

基礎体の構成と η_T ペアリングのアルゴリズムにおける事前計算

まず，基礎体 $\mathbb{F}_{2^m}$ の構成と，η_T ペアリングのアルゴリズムにおける，事前計算の箇所，即ち，Algorithm 3,6 における step 1 の具体的な計算方法について述べる．本実装実験では基礎体として $m = 103, 511$ を取りペアリングを計算しているため，この二つの基礎体の構成について述べる．どちらの拡大次数 m に対しても，$\mathbb{F}_2$ 上既約三項式が存在し，本実装においては $m = 103$ のとき

$$f_{103}(x) = x^{103} + x^9 + 1$$

を，$m = 511$ のとき

$$f_{511}(x) = x^{511} + x^{10} + 1$$

を用いて基礎体を構成する．

事前計算において，多項式基底等による表示を用いた Frobenius map を作用させる計算はコストが大きいため，本実装では基礎体の多項式基底と normal basis との変換を求め，normal basis による表示で計算を行う．各多項式基底から normal basis への基底変換は，なるべく変換コストの小さいものを探したが，コストが最小である変換は見つかっていない．実際 $\mathbb{F}_2[x]/(f_{103}(x))$ から，ある normal basis への変換としてコスト 5047 のものを採用し，逆変換のコストは 5369 である．又，$\mathbb{F}_2[x]/(f_{511}(x))$ に対しては，normal basis への変換行列として，変換コスト 130004 で，逆変換のコストが 130605 となるものを用いた．どちらの変換も normal basis への変換コストは $k^2/2$ 未満で平均より小さいが，基底変換は一部の範囲しか探索できなかったためこれらの変換がコスト最小であるとは限らない．

normal basis による表示に変換した後，$\sigma \in \mathrm{Gal}(\mathbb{F}_{2^m}/\mathbb{F}_2)$ による作用 x_P^σ はシフト演算で処理できるため高速に事前計算を終えることが出来る．以下に，事前計算に掛かる平均実行時間を表 19 に示す．事前計算は CPU を用いて計算し，本提

案手法においては影響を与えないため，事前計算に掛かる実行時間は η_T ペアリングに含めない．

基礎体の拡大次数	normal basis への変換	事前計算全体に掛かる実行時間
$m = 103$	270.22	877.26
$m = 511$	6250.45	21781.7

表 19　事前計算における基礎体の normal basis への変換と，事前計算全体に掛かる実行時間 100 回平均 (μs)

ここで，表 19 の normal basis への変換の実行時間は，事前計算，Algorithm 3,6 における step 1 において基礎体の 4 つの元 x_P, y_P, x_Q, y_Q の変換に掛かる時間である．

η_T ペアリングの並列計算の流れ

ここでは先に示した具体的な η_T ペアリングのアルゴリズム (Algorithm 6) について，第 5.2.3 節で述べた拡大体上の乗算を行う函数等を用い，粗くアルゴリズムを記述する．特に拡大体の構成による η_T ペアリングの実行時間の比較のため，アルゴリズムの実行時間として計測する計算箇所を明らかにして，アルゴリズムを再掲する．Algorithm 9 は，先行研究の，拡大体を 6 次拡大から構成し，s_0w-basis を用いて拡大体の元を表した際の，η_T ペアリングの並列実装に関する概要である．基本的に事前計算，α, β の表示を得る計算以外は二重並列化を行なっていない．又，[5, §7.5] にある様に，最終冪は $(m+1)/2$ の二乗算，4 回の Frobenius map の作用，2 回の乗算，1 回の逆元計算により求めることが出来るがここでは最終冪のコストの大部分を占める，二乗算の箇所をアルゴリズムに示した．

Algorithm 9 η_T pairing ($m \equiv 7 \pmod 8$) の並列実装

Input: $P = (x_P, y_P),\ Q = (x_Q, y_Q) \in \mathrm{Jac}_C(\mathbb{F}_{2^m})$

Output: $f \in \mathbb{F}_{2^{12m}}$

1: **事前計算：基礎体の基底を normal basis に基底変換し，Frobenius map による計算をシフト演算により効率的に行う**

2: $x_1[i] \leftarrow x_P^{2^i},\ y_1[i] \leftarrow y_P^{2^i},\ x_2[i] \leftarrow x_Q^{2^i},\ y_2[i] \leftarrow y_Q^{2^i}\ (0 \le i \le m-1)$

3:

4: **s_0w-basis, 或いは stw-basis を固定し拡大体を構成する**

5: $f \leftarrow 1$

6: **for** $i = 0$ TO $(m-3)/2$ **do**

7: $\quad \alpha, \beta$ を計算する [並列（基礎体のみ）]

8:

9: $\quad$ **乗算 $\alpha\beta$ の計算**

10: $\quad \alpha\beta$ [並列]

11:

12: $\quad$ **一般の元の拡大体上の乗算**

13: $\quad f \leftarrow f \cdot (\alpha \cdot \beta)$ [並列]

14: **end for**

15:

16: **Perform the final doublings / addition**

17: **2 回の二乗算と乗算**

18: $f \leftarrow f^4 \cdot c$ [並列]

19:

20: **Perform the final exponentiation**

21: **特に $(m+1)/2$ 回の二乗算**

22: $f \leftarrow f^{2^{(m+1)/2}}$ [並列]

23: **return** f

基底変換により拡大体の構成を変更した η_T ペアリング

ここでは，基礎体の拡大次数に対し，η_T ペアリングが効率的に，即ち，より小さい実行時間，或いは，並列化による高速化度合いのより大きな実装結果が得られるような拡大体の構成について見るために，η_T ペアリングの基底変換を適用した並列実装の実行時間のシミュレーション結果を示す．

基礎体 $\mathbb{F}_{2^m}$ 上の 12 次拡大体の基底変換については 4.3 節で詳しく述べた．具体的には s_0w-basis から stw-basis への基底変換行列としてコスト 32 の行列 P (12) を用い，P の逆変換であり最小コスト 28 の行列 Q (13) を用いて基底変換を行っている．表 20,21 にそれぞれの基礎体の拡大次数における，拡大体 $\mathbb{F}_{2^{12m}}$ の 6 次拡大から構成する s_0w-basis と 3 次拡大から構成する stw-basis との間の基底変換の平均実行時間を示す．

変換前後の基底	基底変換に掛かる実行時間
s_0w-basis $\rightarrow$ stw-basis	51.05
stw-basis $\rightarrow$ s_0w-basis	46.11

表 20　拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$ における基底変換に掛かる実行時間 100 回平均 (μs)

変換前後の基底	基底変換に掛かる実行時間
s_0w-basis $\rightarrow$ stw-basis	209.36
stw-basis $\rightarrow$ s_0w-basis	187.54

表 21　拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ における基底変換に掛かる実行時間 100 回平均 (μs)

本実装においては，拡大体の一回の乗算等と比べ基底変換に掛かる実行時間はあまり影響が無い程小さいものとなっている．第 5.2.3 節より，12 次拡大体上の乗算については，基礎体の拡大次数が $m = 511$ と大きいときは，若干ではある

が，拡大を 6 次拡大から始める構成の方が 3 次拡大から始めるものより Host 時間，GPU 時間共に小さい．又，二乗算に関しては，今，拡大体の標数が 2 であるため 3 次拡大から始める構成の方が Host 時間は小さい（GPU 時間は一般の乗算と大きく変わらない）．更に，第 5.2.3 節より，乗算 $\alpha\beta$ に関しては 3 次拡大から始める構成の方が実行時間は小さい．従って，η_T ペアリングのアルゴリズムにおいて，乗算 $\alpha\beta$ の計算は 3 次拡大からの構成で行うべきであり，ペアリングの事前計算，α, β の計算は初めから 3 次拡大からの構成，即ち，stw-basis 基底の表示の方が都合が良い．

次に Miller loop の各ステップにおいて，それ以前の冪乗 f と $\alpha\beta$ との計算箇所は一般の拡大体上の乗算なので 6 次拡大からの構成，即ち，s_0w-basis へ基底変換して計算する．Miller loop を抜け，final doublings / addition のセクションと最終冪に関しては，二乗算による拡大体上の乗算が大部分を占めるので，ここで stw-basis へ変換し計算を進める．或いは，GPU 時間としては s_0w-basis を用いたほうが幾分高速で，標数が 2 でないときは一般の元の乗算と状況は同じであるため，stw-basis へ基底変換せずに，最終冪を求め，ペアリング値を得た後に元の stw-basis の表現に戻す実装も行う．実際に以下の様に η_T ペアリングのアルゴリズムに基底変換を適用する．Algorithm 10 において η_T ペアリングの拡大体における乗算に関して，二乗算以外の，一般の元同士の乗算を s_0w-basis を用いて計算し，二乗算は stw-basis を用いて計算する流れを示した．二乗算以外の乗算については Miller loop 内で行われる，各ステップの $\alpha\beta$ の冪乗部分のみを対象として実装した．

更に，Algorithm 11 において，乗算 $\alpha\beta$ 以外の拡大体上の乗算を，stw-basis を用いて計算する η_T ペアリングのアルゴリズムの流れを示した．

実際の η_T ペアリングの並列実装実行時間のシミュレーション結果，その評価については後の第 6.1, 6.2 節において詳述する．

Algorithm 10 η_T pairing ($m \equiv 7 \pmod 8$) の二乗算以外の一般の乗算について，基底変換を作用させた並列実装

Input: $P = (x_P, y_P),\ Q = (x_Q, y_Q) \in \mathrm{Jac}_C(\mathbb{F}_{2^m})$

Output: $f \in \mathbb{F}_{2^{12m}}$

1: 事前計算：基礎体の基底を normal basis に基底変換し，Frobenius map による計算をシフト演算により効率的に行う
2: $x_1[i] \leftarrow x_P^{2^i},\ y_1[i] \leftarrow y_P^{2^i},\ x_2[i] \leftarrow x_Q^{2^i},\ y_2[i] \leftarrow y_Q^{2^i}\ (0 \le i \le m-1)$
3:
4: ***stw*-basis により拡大体を構成する**
5: $f \leftarrow 1$
6: **for** $i = 0$ TO $(m-3)/2$ **do**
7: $\quad \alpha, \beta$ を計算する [並列（基礎体のみ）]
8:
9: $\quad$ ***stw*-basis による乗算 $\alpha\beta$ の計算**
10: $\quad \alpha\beta$ [並列]
11:
12: $\quad$ **基底変換：*stw*-basis → *s*$_0$*w*-basis**
13: $\quad$ 一般の元の拡大体上の乗算
14: $\quad f \leftarrow f \cdot (\alpha \cdot \beta)$ [並列]
15: **end for**
16:
17: Perform the final doublings / addition
18: **基底変換：*s*$_0$*w*-basis → *stw*-basis**
19: 2 回の二乗算と乗算
20: $f \leftarrow f^4 \cdot c$ [並列]
21:
22: Perform the final exponentiation
23: 特に $(m+1)/2$ 回の二乗算
24: $f \leftarrow f^{2^{(m+1)/2}}$ [並列]
25: **return** f

Algorithm 11 η_T pairing ($m \equiv 7 \pmod 8$) の拡大体の乗算の際，基底変換を作用させた並列実装

Input: $P = (x_P, y_P),\ Q = (x_Q, y_Q) \in \mathrm{Jac}_C(\mathbb{F}_{2^m})$

Output: $f \in \mathbb{F}_{2^{12m}}$

1: 事前計算：基礎体の基底を normal basis に基底変換し，Frobenius map による計算をシフト演算により効率的に行う
2: $x_1[i] \leftarrow x_P^{2^i},\ y_1[i] \leftarrow y_P^{2^i},\ x_2[i] \leftarrow x_Q^{2^i},\ y_2[i] \leftarrow y_Q^{2^i}\ (0 \le i \le m-1)$
3:
4: ***stw*-basis により拡大体を構成する**
5: $f \leftarrow 1$
6: **for** $i = 0$ TO $(m-3)/2$ **do**
7: α, β を計算する [並列（基礎体のみ）]
8:
9: ***stw*-basis による乗算 $\alpha\beta$ の計算**
10: $\alpha\beta$ [並列]
11:
12: **基底変換：*stw*-basis → s_0w-basis**
13: 一般の元の拡大体上の乗算
14: $f \leftarrow f \cdot (\alpha \cdot \beta)$ [並列]
15: **end for**
16:
17: Perform the final doublings / addition
18: 2 回の二乗算と乗算
19: $f \leftarrow f^4 \cdot c$ [並列]
20:
21: Perform the final exponentiation
22: 特に $(m+1)/2$ 回の二乗算
23: $f \leftarrow f^{2^{(m+1)/2}}$ [並列]
24:
25: **基底変換：s_0w-basis → *stw*-basis**
26: **return** f

6. 評価と考察

本章では，これまで述べてきた η_T ペアリングの並列実装について，ペアリングの計算に係る拡大体の構成を変更する提案手法に対して，実装実験結果の評価を行う．又，有限体の基底変換に対しても，本研究における提案手法に対し，評価と考察を与える．第 6.1 節において，η_T ペアリングの並列実装のシミュレーション結果を示し，基礎体の拡大次数に対して，ペアリングの実行時間と，本研究において提案する拡大体の構成の評価を行う．次に，第 6.2 節において，η_T ペアリング，或いは基礎体，拡大体の基本算術の並列実装について幾つか考察を与える．最後に第 6.3 節において，第 4 章で述べた有限体の基底変換に関する考察と，基底変換求解と変換コストに関して今後の課題について述べる．

6.1 評価

ここでは，η_T ペアリングの並列実装の実験結果を示し，実装方法について評価を与える．表 22,23 において $m = 103, 511$ に対する η_T ペアリングの並列実装実行時間のシミュレーション結果を示す．それぞれの場合で，拡大体の構成方法として，先行研究の 6 次拡大の後に 2 次拡大で構成する s_0w-basis を用いたもの，提案手法の 3 次拡大から始め 2 次拡大を二度繰り返し構成する stw-basis を用いたものについて実装を行っている．更に，拡大体上での乗算 $\alpha\beta$ は stw-basis を用いて計算したほうがコストが小さいため，α, β の計算と乗算 $\alpha\beta$ の計算を stw-basis を用いて処理した．又，m が大きい時は拡大体の他の乗算を s_0w-basis を用いて計算したほうが効率的であるため基底変換をアルゴリズムの途中で適用させた実装も行った．加えて，今の定義体である基礎体は標数 2 であるため，本実装では拡大体上の二乗算が，一般の元同士による乗算に比べ高速に処理できる．従って，s_0w-basis への変換を，二乗算以外の拡大上の乗算に適用させる方法と，全ての拡大体上の乗算に対し基底変換を適用させる方法で実装を行った．実装方法について，それぞれの手法の詳細とその利点を以下にまとめる．

- **6 次拡大から始める (s_0w-basis)** 拡大体 $\mathbb{F}_{2^{12m}}$ を 6 次拡大から始めて構成し，多項式基底 s_0w-basis を用いて $\mathbb{F}_{2^{12m}}$ 上の算術を行う．基礎体の拡大次

数 m が大きい時は Karatsuba 乗算の並列化による効果等から，3 次拡大から始めて構成する場合に比べ，$\mathbb{F}_{2^{12m}}$ 上の乗算をより高速に行うことが出来る．

- **3 次拡大から始める (*stw*-basis)** 拡大体 $\mathbb{F}_{2^{12m}}$ を 3 次拡大から始めて構成し，多項式基底 stw-basis を用いて $\mathbb{F}_{2^{12m}}$ 上の算術を行う．この構成において，η_T ペアリングのアルゴルズムにおける $\mathbb{F}_{2^{12m}}$ 上の乗算 $\alpha\beta$ は，他の $\mathbb{F}_{2^{12m}}$ の構成に比べてより計算コストが小さく，高速に計算出来る．又，基礎体の拡大次数 m がさほど大きくないときは，$\mathbb{F}_{2^{12m}}$ 上の乗算を 6 次拡大から始めて構成した場合に比べより高速に処理出来る．更に，今回のペアリングパラメタである，曲線の定義体（基礎体）の標数が 2 であるため，本実装において，特別に二乗算を一般の元同士の乗算に比べてより高速に処理できる．標数が 2 でないときは，この通りでない．

- **$\mathbb{F}_{2^{12m}}$ 上の二乗算以外の乗算を s_0w-basis に基底変換し計算する** $\alpha\beta$ の計算は stw-basis を用いた計算が効率的であるため，α, β の計算と，乗算 $\alpha\beta$ は 3 次拡大から始めた構成で計算し，他の乗算について s_0w-basis への変換を考える．本並列実装において，基礎体の拡大次数 m が大きい時（$m = 511$ 或いはより高次），$\mathbb{F}_{2^{12m}}$ 上の乗算は s_0w-basis を用いて計算したほうが高速である．更に，今回の場合の様に基礎体の標数が 2 の時は，$\mathbb{F}_{2^{12m}}$ 上の二乗算は stw-basis による計算が高速である．加えて，η_T ペアリングの計算では $\mathbb{F}_{2^{12m}}$ の乗算として二乗算が過半数を占める．従って，本実装方法では，二乗算については stw-basis を用いて計算し，他の乗算を s_0w-basis へ基底変換して計算を行う．

- **$\mathbb{F}_{2^{12m}}$ 上の $\alpha\beta$ 以外の乗算を s_0w-basis に基底変換し計算する** 上の基底変換による手法と同様，α, β の計算と，乗算 $\alpha\beta$ は 3 次拡大から始めた構成で計算し，他の $\mathbb{F}_{2^{12m}}$ 上の乗算について s_0w-basis への変換を考える．ここでは，標数が 2 でない場合を考え，$\alpha\beta$ 以外の $\mathbb{F}_{2^{12m}}$ 上の全ての乗算を s_0w-basis を変換する．先に述べた様に，基礎体の拡大次数 m が大きい時は，s_0w-basis による計算の方が高速であるため，基底変換により高速化が見込める．

表 22,23 について，各実装方法に対する η_T ペアリングの並列計算に関するシミュレーション結果を示す．この後の第 6.1.1, 6.1.2 節において，それぞれの実装方法に関するペアリング計算の実行時間について，基礎体の拡大次数 m に対し，$m = 103, 511$ の二つの場合において評価する．

拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 103)$ の構成方法	実装 I	実装 III	高速化率（実装 I / 実装 III)
6 次拡大から始める	37.917 (35.680)	3.5638 (0.19724)	10.64 (180.9)
3 次拡大から始める	24.793 (22.701)	2.6701 (0.14033)	9.285 (161.8)
拡大体の二乗算は stw-basis による	30.437 (28.379)	3.0119 (0.16408)	10.11 (173.0)
拡大体の全ての乗算は s_0w-basis による	36.956 (34.852)	3.4925 (0.191454)	10.58 (182.0)

表 22 基礎体 $\mathbb{F}_{2^m}$ $(m = 103)$ 上の η_T ペアリング計算の Host 時間（括弧内は GPU 時間） 10 回平均 (sec)

拡大体 $\mathbb{F}_{2^{12m}}$ $(m = 511)$ の構成方法	実装 I	実装 III	高速化率（実装 I/実装 III)
6 次拡大から始める	5065.7 (4912.6)	163.209 (2.84339)	31.04 (1727.7)
3 次拡大から始める	3154.9 (3013.9)	146.853 (3.0268)	21.48 (995.7)
拡大体の二乗算は stw-basis による	4042.4 (3900.68)	149.824 (2.8421)	26.98 (1372.5)
拡大体の全ての乗算は s_0w-basis による	4962.5 (4811.6)	161.38 (2.62562)	30.75 (1832.6)

表 23 基礎体 $\mathbb{F}_{2^m}$ $(m = 511)$ 上の η_T ペアリング計算の Host 時間（括弧内は GPU 時間） 10 回平均 (sec)

6.1.1 基礎体 $\mathbb{F}_{2^{103}}$ 上の η_T ペアリング

ここでは，η_T ペアリングの並列実装について，$m = 103$ の場合における実装実験の結果を元に，実装方法による結果を比較して，拡大体上の構成方法，又は，η_T ペアリングの計算方法について評価を与える．図 10 において，表 22 で述べた値を用いて，それぞれの実装方法に対する実装実験の結果を比較するグラフを描いた．

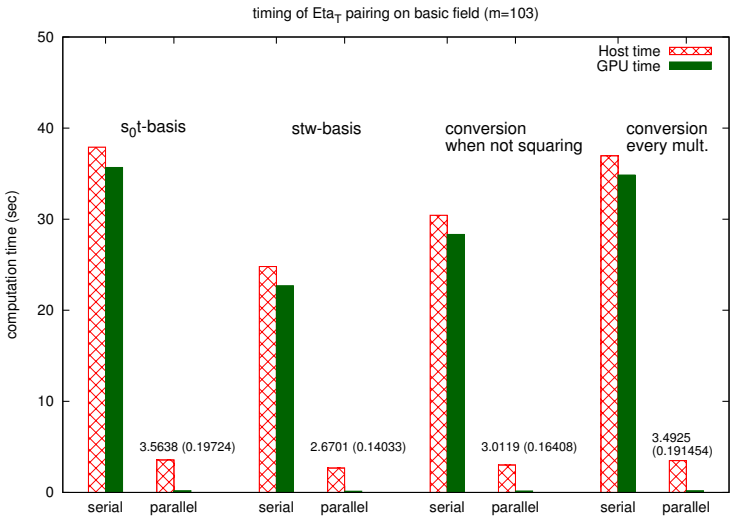

図 10　基礎体 $\mathbb{F}_{2^{103}}$ 上の η_T ペアリングの並列実装による高速化の比較

第 6.1 節で述べた様に，$m = 103$ の時は，$\alpha\beta$, 拡大体上の乗算，或いは二乗算において，stw-basis を用いた手法が，他の手法に比べ高速に計算が行える．よって，シリアル，並列実装において Host 時間，GPU 時間共に 3 次拡大から始め，stw-basis を用いて η_T ペアリングを実装したものが最も高速である．

又，表 15 で示した結果によると，標数が 2 以外の場合でも，$m = 103$ の時は同様に二乗算の計算は stw-basis による構成において高速である．従って，基礎体の拡大次数が $m = 103$ 程度であるとき，本提案手法である拡大体を 3 次拡大から始め，多項式基底 stw-basis によって拡大体の元を表現した際，より高速に並列計算できる．

6.1.2　基礎体 $\mathbb{F}_{2^{511}}$ 上の η_T ペアリング

先の第 6.1.1 節と同様にして，ここでは $m = 511$ の場合の η_T ペアリングの並列実装に対する実験結果について，実装方法による比較を行う．表 23 の値から，各実装おける並列計算結果を比較するグラフを図 11 に示す．

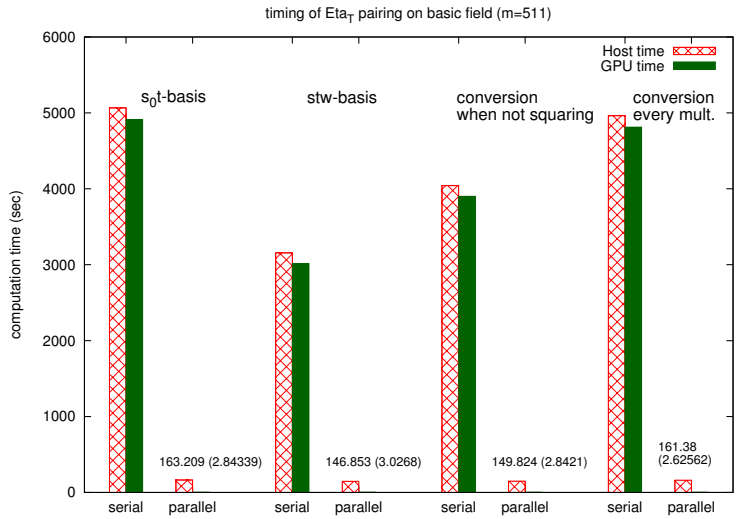

図 11　基礎体 $\mathbb{F}_{2^{511}}$ 上の η_T ペアリングの並列実装による高速化の比較

まず，シリアル実装の結果は，それぞれの実装方法において $m = 103$ のときと同様の関係になっており，並列化による高速化度合いは当然 $m = 511$ の場合のほうが大きい．シリアル実装において，stw-basis を用いた計算が，比較的高速であるのは標数 2 の拡大体上における二乗算の計算が高速に行えるためであり，$m = 511$ と基礎体の拡大次数が大きい時，その影響が顕著に現れている．

次に，特に並列実装の結果について評価を行う．まず，図 12 において，η_T ペアリングの並列処理に掛かる実行時間の比較のためのグラフを示す．これは，図 11 の並列実装による実行時間の値を並べたものである．図 11 のシリアル実装の結果に比べ，stw-basis による構成と，拡大体上の全ての乗算の際に基底変換を適用し，s_0w-basis によって構成する実装による実行時間の差がより小さくなっている．更に，表 16 の結果より，$m = 511$ のとき s_0w-basis による構成の方が，標数が 2 以外のときの二乗算をより高速に行える．即ち，一般に標数が 2 以外の場合，η_T ペアリングの並列化については，$\mathbb{F}_{2^{12m}}$ 上の乗算を s_0w-basis による構成で計算した方が効率的と言える．又，今回の場合，$m = 511$ の時は拡大体の一般の乗算は s_0w-basis による構成における計算が高速で，二乗算は stw-basis による構成における計算が高速であるため，二乗算以外の乗算に関し基底変換を施す三番目の

手法が，最も高速に計算を行えると推測される．しかし，本研究における η_T ペアリングの計算では，二乗算以外の乗算は $\alpha\beta$ とその Miller loop のステップ以前に計算された冪との乗算であり，$\alpha\beta$ は幾分疎な形をしているので，この時の乗算に掛かる実行時間は二つの構成において同等，或いは stw-basis による構成の方が高速となる．従って，今回の場合，特に標数 2 の時は，全ての乗算を stw-basis の構成の下で計算する方法が最も高速となった．

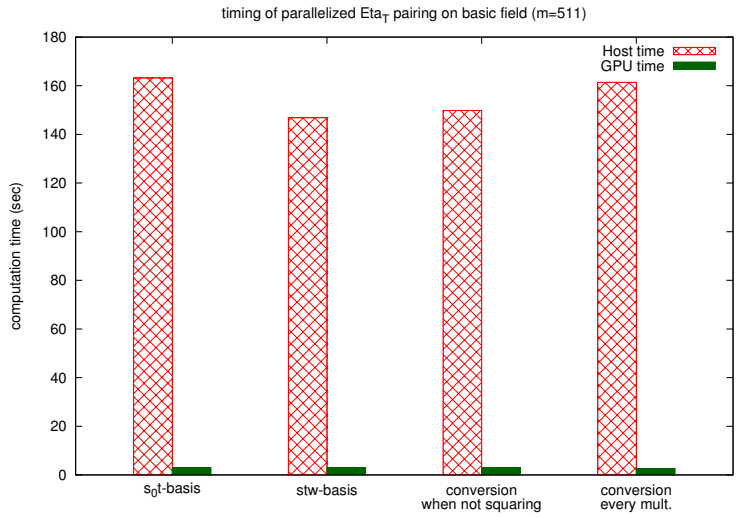

図 12　基礎体 $\mathbb{F}_{2^{511}}$ 上の η_T ペアリングの並列処理による実行時間の比較

最後に，図 13 において η_T ペアリングの並列処理の GPU 時間に注目し，比較のグラフを示す．これは図 12 の GPU 時間のみを比較のため並べたものである．GPU 時間は，GPU 上での並列処理に掛かる時間のみを計測した値であり，図 13 から分かるように，$\mathbb{F}_{2^{12m}}$ 上の乗算の際，基底変換を適用して 6 次拡大から始めた構成で計算する実装方法において，最も高速に η_T ペアリングの計算が達成されている．

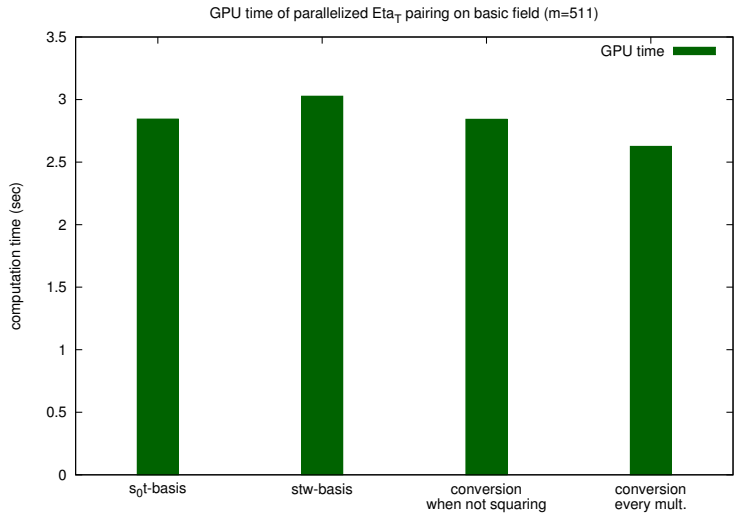

図 13　基礎体 $\mathbb{F}_{2^{511}}$ 上の η_T ペアリングの並列処理による GPU 実行時間の比較

以上より，基礎体の拡大次数 m の値が $m = 511$ 等比較的大きい時，並列化において並列度が大きくなるため，乗算 $\alpha\beta$ と，その他の拡大体上の乗算を基底変換を適用して，拡大体の構成を変更しながら計算を行うとより高速に η_T ペアリングの計算が可能であることが分かった．特に今回のペアリングパラメタにおいて，定義体の標数が 2 であることから，拡大体を 3 次拡大から始めて構成すると最も η_T ペアリングの高速計算が達成されるが，標数が 2 以外の場合は，$\mathbb{F}_{2^{12m}}$ 上の乗算 $\alpha\beta$ は 3 次拡大からの構成，その他の乗算は 6 次拡大からの構成による実装による計算が最も高速であることが明らかである．

6.2　並列実装に対する考察

本節において，これまで述べてきた η_T ペアリングの並列化について，そのペアリングの定義体である基礎体，その上の拡大体における算術の並列実装，又はペアリングの並列実装に関する考察を与え，今後の課題について幾つか言及する．又，第 4 章で述べた有限体の基底変換に関する提案についても，本節において考察し，今後の課題について検討する．第 6.2.1 節においては，基礎体，拡大体の

並列実装について，本実装において考察外とした手法や，課題点について説明する．第 6.2.2 節では，特に η_T ペアリングの並列化について考察を与える．

6.2.1 基礎体，拡大体における並列演算

本節では，第 5.2.2 節で具体的に述べた，基礎体，拡大体の並列実装について考察を与える．又，本実装では取り組まなかった拡大体の算術に関する実装手法について幾つか議論を行う．

基礎体における並列演算

基礎体の算術の並列化については，拡大体上の算術の並列化，ひいてはペアリングの並列化に多大な影響を与えるが，本並列実装ではシンプルな方法で実装を行った．本研究において，ペアリングの並列化に関して，GPU を用いた並列ソフトウェア実装による実験を通して研究を行うこととしたため，特別な有限体に対する高速化手法等を実装することはできなかった．しかし，これまでにシミュレーション結果で確かめたように，本実装の基礎体の算術に対するシンプルな並列実装において，基礎体を小標数として，その拡大次数が比較的大きい場合では，特に乗算において十分に並列化による高速化が達成できたと言える．又，その様なパラメタをもつ任意の基礎体に対し本実装が適用出来る．

又，乗算に関して，多項式として乗算を行った後に，定義多項式によって mod 演算を行う必要があるが，これについても実装実験を行うことと同等の理由により，並列化について考慮しなかった．定義多項式の形によっては，高速に mod 演算を行う，或いはそれを並列に処理できるが，任意の多項式をパラメタとして取る必要があるので，本実装では直接的にシリアルに多項式の剰余計算を行った．mod 演算については拡大体の並列演算についての考察として述べるが，並列化が適用出来る様に特別な基底への基底変換などが考えられ，これらは今後の課題とする．又，本研究では小標数の基礎体を対象としたが，サイズの大きい素体等に関しても並列化を考慮する必要がある．即ち多倍長演算の並列，且つ高速な実装についても検討する必要がある．

拡大体における並列演算

本実装において，拡大体の加算については，直接的に基礎体の加算の並列化と同じ要領で二重並列化を行ったが，実装結果としては十分に並列化の効果が得られている（表 8）．乗算については，今回，拡大体はペアリングで用いる曲線の定義体である基礎体を拡大したものとして考えているため，そのサイズは比較的大きく，乗算アルゴリズムとして単純に schoolbook method ではなく効率的な手法を取ることにした．本実装では，拡大体の乗算として Karatsuba method による手法を取った．大きなサイズの元の乗算をより効率的に行える手法として，他に Toom-Cook method 等が挙げられるが，本実装では効率的な並列化を考慮する必要があり，例えば Toom-Cook method においても並列に独立に計算を行える箇所は少なくはないが，Karatsuba method による並列化は CUDA を用いてプログラミングをする際，効率的に実装が行えるため本手法を採った．具体的に言うと，本実装では Karatsuba method の事前計算を並列に処理しているが，そのそれぞれ独立な計算式が互いに同じ形をしているため，GPU 上で効率的に処理出来る．逆に，全く別の内容の処理を GPU 上で並列に処理する際，少なからずオーバヘッドが生じる．このことから，本実装では Toom-Cook 等のより良い乗算手法を採らず，Karatsuba method の並列化により，拡大体の乗算を実装した．

Karatsuba method は二次拡大だけでなく，任意の次数において同等のアルゴリズムを適用できる．本実装ではそれにより，拡大体 $\mathbb{F}_{2^{12m}}$ について，基礎体のサイズが比較的小さい時は（$m = 103$ 等）3 次拡大から，大きい時は（$m = 511$ 等）6 次拡大から始めて構成すると効率的であることを示したが，Karatsuba 乗算は一般的には再帰的に適用した方が効率的に計算が行える．しかし，本実装において，Karatsuba 乗算を拡大体の乗算に適用しているため，拡大体の構成により Karatsuba 乗算の組み合わせ方に制限がある．よって，拡大体の構成と基底変換の考察も重要且つ必要な事柄となる．

拡大体上の乗算の並列化で，考察すべきこととして mod 演算に掛かるコストが挙げられる．表 11,12, 或いは Algorithm 7,8 より，$\mathbb{F}_{2^{12m}}$ の乗算に掛かる GPU 時間に比べ，Host 時間が大幅に大きいことと，GPU 時間は Karatsuba 乗算などの GPU 上での並列処理に掛かる時間が大半を占めることから，全体の時間の内 mod

演算に掛かるコストが大半であることが分かる．mod 演算のコスト削減の一つの対応策として，野上らによる CVMA (Cyclic Vector Multiplication Algorithm) [20] の並列化が考えられる．CVMA は特別な基底，即ち拡大体を既約多項式 all-one polynomial で構成し，その擬多項式基底，或いはこのとき，この基底と等価である normal basis を用いて乗算行うものである．CVMA は [20, §3] より，独立に並列に計算出来る箇所が比較的多く，更に拡大体の元としての乗算が行われるため，多項式基底による乗算の様に mod 演算を必要としない．但し，拡大体の一般の元同士の CVMA による乗算は，k 次拡大体において基礎体の乗算が $k(2k+1)$ 回必要となり，Karatsuba method の並列化と比べ，どの程度の実行時間が得られるのかは実際に実装実験を行わなければ不明である．更に，CVMA を用いて計算するために，拡大体を all-one polynomial による構成に変換しなければならない．これらの理由により，本実装では CVMA を採用せず Karatsuba method の並列化を行った．CVMA の並列化等，拡大体上の乗算の効率的な並列実装に関して今後検討を続ける必要がある．

6.2.2 η_T ペアリングアルゴリズムの並列化

本研究で対象とした η_T ペアリングは，特別に Miller loop の各ステップを独立に計算することができる．即ち，各ステップの積 $\alpha\beta$ を独立に計算し，後で掛け合わせることができる．しかし本実装では，小標数の定義体に注目し，基礎体，拡大体の算術を並列化することにより，ペアリング全体の並列実装を行った．理由として，一般にペアリングの計算は Miller's algorithm によるものが主で，基本的に同様の計算が行われるが，今回の η_T ペアリングの様に事前計算を通して Miller loop の各ステップを独立に計算できることは稀であることと，小標数の場合並列化による影響が何より大きいことが挙げられる．よって，本実装では，ペアリングの並列化については，ペアリングの計算に必須である拡大体の算術の効率的な並列実装が，主要な研究対象であると言える．

本研究で扱う η_T ペアリングは，超楕円曲線が持つ distortion map を用いて拡大体の元を表現し，それらの計算を行うものである．提案手法ではその distortion map により表される α, β に関して，それらの乗算を効率的に計算した．ここで，

Miller loop の各ステップの $\alpha\beta$ を掛け合わせる際，今回の場合，$\alpha\beta$ は常に幾分 sparse な表示を持つので，その表現に対して拡大体の乗算として特別に実装を行うと効率的である．又，第 5.1.1 節において考察を与えた distortion map の構成について，α, β を始め拡大体の元は distortion map により表現が決まるため，効率的に計算が行える様に distortion map を選択する必要がある．第 5.1.1 節で述べた様に，既約三次式の根を用いて distortion map を構成することは難しいが，他の構成については詳しく調べる必要がある．distortion map の選択により η_T ペアリングのパラメタも変更が必要になるため，それも含め今後調査の必要がある．

6.3 有限体の基底変換の求解と変換コスト

本節では，第 4 章で述べた有限体の基底変換の求解と変換コストに関する研究結果について考察と課題について議論する．基底変換求解，変換コストに関するシミュレーション結果とその評価は，第 4 章において論じている．

基底変換の求解

本研究の想定では，任意の拡大体のパラメタに適用可能なアルゴリズムについて考察した．古典的な EDF を用いた変換求解アルゴリズムに対して，本提案手法によって大幅な改善が達成された．しかし，第 4.1 節で述べた様に，拡大体のパラメタに制限が必要だが，GNB 等の特別な基底を用いた手法は，提案手法に比べ格段に高速で効率的に変換行列を求めることが出来る．拡大体のパラメタに制限が無く，EDF とは本質的に異なる基底変換求解の手法については今後検討していく必要がある．

又，与えられた多項式基底に対して normal basis の生成元を求める本提案手法については，実装の改良によりまだ効率化が図れる余地があると思われる．GCD については NTL [22] の函数により直接計算し，計算時間の増加は安定していた．提案手法に関する本実装においては式 (11) の $f(x)$ の根が成す巡回群について，巡回群の生成元の探索は比較的高速に終了するが，巡回群を $g(x)$ に代入して函数値を計算する箇所のコストが高い．今後実装面の改良も行っていきたい．

更に，本研究では考慮しなかったが，GCD を直接求める方法ではなく，提案手法の，式 (11) に対する共通根の有無の判定によるものは，$f(x)$ が位数 k の巡回群を成さないとき，変換行列が正確に得られるとは限らなかったが，その変換が失敗する確率については今後調べる必要がある．本シミュレーションでは，小標数の時でさえ滅多に失敗は起きなかったが，具体的に有限体の位数等を用いて考察する必要がある．

変換コスト

基底変換行列の変換コストについては，第 4.2.2 節で述べた様に，定義多項式の間の根の関係式が重要であり，コストの小さい変換を求めるためには，許される限り定義多項式の互いの根の表示が sparse である多項式基底を採用するべきである．ここでは特に，pairing-friendly field 等を構成する既約二項式，或いは既約三項式について，それぞれの多項式が normal basis を成す根を持つ条件を考察する．即ち，normal basis への変換コストが最小である様なより sparse な定義多項式を見つけたい．

まず，既約二項式について考察する．$\mathbb{F}_q$ 上の既約二項式 $f(x) = x^k - c$ は，常に normal basis の生成元を根に持たない，即ち同値であることの $f(x)$ の根を並べたものは常に normal basis ではない．なぜなら，Frobenius map $\sigma \in \mathrm{Gal}(\mathbb{F}_{q^k}/\mathbb{F}_q)$ を取り

$$f(x) = \prod_{i=0}^{k-1}(x - a^{\sigma^i})$$

とした時，右辺の展開式を

$$\sum_{i=0}^{k-1} a_i x^i = \prod_{i=0}^{k-1}(x - a^{\sigma^i})$$

と書くと，

$$\mathrm{trace}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(a) = \sum_{i=0}^{k-1} a^{\sigma^i} = -a_{k-1} = 0$$

より $a, a^{\sigma}, ..., a^{\sigma^{k-1}}$ は $\mathbb{F}_q$ 上一次従属となり normal basis を成さない．

従って，既約 k 次多項式 $f(x)$ に対して，その根が normal basis を成すために，$f(x)$ の $k-1$ 次の項の係数は 0 でない必要がある．よって，今後既約三項式 $f(x) = x^k + cx^{k-1} + d$ について，normal basis を成す根を持つものについてシミュレーションを通して調査したい．更に，より一般的に既約多項式の根が normal basis を成すための条件について理論的に研究を行っていく必要がある．

7. 結論

本研究において，binary field 上で定義された種数 2 の超楕円曲線上の η_T ペアリングに対して，基底変換により拡大体の構成を変更した効率的な並列化手法を提案した．更に，有限体の基底変換について，変換行列の求解に関する EDF を用いた既存のアルゴリズムに対して，normal basis を用いることでその計算コストを削減した効率的な求解アルゴリズムを提案した．

ペアリングは，今や様々な暗号プロトコルに利用されている．ペアリング暗号は，その数学的性質から，超楕円曲線暗号などに比べ暗号化，復号等の演算がより複雑であり，ペアリングの高速化に関する研究は，理論，実装の面から盛んに行われている．本研究では GPU を用いた並列実装によりペアリングの高速化について研究を行い，研究手法として並列化による実装実験を通して，ペアリングの持つパラメタをいかに選択すれば高速な並列化が達成されるかについて考察を与えた．ここで，ペアリングのパラメタとは，曲線の定義体である基礎体，又，ペアリングの計算で必須の基礎体上の拡大体，更に，今回研究対象とした η_T ペアリングについては distortion map 等である．

η_T ペアリングの並列化については，ペアリングが持つ超楕円曲線が定義されている体である基礎体と，その上の拡大体上の算術を GPU 上で並列に処理することにより実装を行った．本研究では並列化に対して実装実験を行うため，ペアリングのパラメタ，即ち超楕円曲線の定義体である基礎体，又，その上の拡大体について，標数，拡大次数等のパラメタに柔軟に対応しなければならない．例えば，特定の体における乗算に対する高速実装や，定義多項式による効率的な剰余計算等については考慮外としている．従って，基礎体の算術については，シンプルに多項式の算術を並列化することにより並列実装した．更に，拡大体については GPU 上での並列実装が効率的に行えることと，仮定として拡大体のサイズが大きいという理由から Karatsuba method による乗算を採用し，これを並列実装した．これにより，基礎体の拡大次数と，拡大体の構成に依って乗算の実行時間に変化があることを実装実験で確かめた．具体的には，基礎体の拡大次数 m が比較的大きい時は，拡大体 $\mathbb{F}_{2^{12m}}$ を 6 次拡大から始め 2 次拡大を行って構成したところで乗算を行った方が高速であるといった具合である．この様にして，基礎

体の大きさ対し，拡大体上の乗算をより高速に計算する拡大体の構成方法を発見した．

又，先行研究における η_T ペアリングの計算は拡大体 $\mathbb{F}_{2^{12m}}$ の構成が 6 次拡大から始める構成であったが，本提案手法では拡大体を 3 次拡大から構成することにより，ペアリングアルゴリズムにおける拡大体上の乗算 $\alpha\beta$ に掛かるコストの削減を実現した．この手法は，η_T ペアリングのパラメタである，超楕円曲線が持つ distortion map と拡大体の構成との関係における考察から導き出した．これにより，$\alpha\beta$ の計算は 3 次拡大から始める構成，その他の乗算は 6 次拡大から始める構成の下で計算するという，基底変換を用いた効率的な並列実装を実現した．今回の標数が 2 の場合では，拡大体上の二乗算が，本実装においては他の乗算に比べ特別に高速であるため，全ての拡大体上の乗算を 3 次拡大から始める構成の下で行う手法が最速となった．一方で標数が 2 以外の時は，実際 $m = 511$ の時，上記の $\alpha\beta$ は 6 次拡大から，その他の乗算は 3 次拡大からの構成の下での η_T ペアリングの計算が高速に実装出来ることを保証するデータを示し，この構成が η_T ペアリングだけでなく様々なペアリングの計算においても適用可能であることを示唆した．

η_T ペアリングの並列実装で用いた有限体の基底変換について，本研究では基底変換求解の効率的なアルゴリズムを示した．本研究の基底変換における対象として，任意の拡大体の多項式基底間の変換を求めることとした．この様な場合，実際に基底変換を求めるために，古典的な多項式の既約分解のアルゴリズムである EDF を用いる手法が挙げられるが，本提案手法では，この EDF を用いた手法を normal basis への基底変換を適用することにより，大幅な計算コスト削減を達成した．具体的には，EDF のアルゴリズムにおける trace の計算を，逐一 normal basis へ変換して行うことで，全体としての効率化に成功した．更に，多項式基底と normal basis との基底変換の求解についても，独自の計算方法を提案し， 小標数の素体上の拡大体において，単に GCD 計算を行う古典的な方法に比べ高速に変換行列を求めることに成功した．

本提案手法に関して，拡大体上の Karatsuba method による乗算の並列化とその並列化に適した拡大体の構成については，GPGPU によるペアリングの並列化

としてはこれまでに例を見ない有効的な手法であり，基底変換の求解においても，任意の拡大体の間の高速な変換行列探索手法として，従来手法に対する優位性が示せた．

謝辞

本研究の遂行にあたり，熱心な御指導，研究方針に関し適切な御助言を戴きました，主指導教員である本学総合基盤センターの藤川和利教授に深く感謝の意を表します.

研究内容に関して終始御指導を頂き，又，本研究の専門分野に近い研究者の方を御紹介して下さるなど，多大なる御援助を賜りました，副指導教員である本学総合基盤センターの猪俣敦夫准教授に心より感謝致します.

副指導教員である，本学情報科学研究科計算メカニズム学研究室の関浩之教授，本学情報科学研究科インターネット工学研究室の山口英教授の両先生には，中間報告において研究に関して鋭い御指摘，研究室内での議論とは別の角度からの御助言を賜りました．ここに感謝の意を表します.

本研究に関し，現実的な研究という観点からの御助言，御指摘，更に，研究とは何かについて熱心に御指導頂きました，慶應義塾大学大学院メディアデザイン研究科の砂原秀樹教授に感謝致します.

研究活動，或いは学生生活に於いて研究に対する姿勢等，あらゆる領域で熱い御指導を賜りました，本学情報科学研究科情報基盤システム学研究室の松浦知史特任准教授に心より感謝致します.

本研究における実装面等において，御指導，御鞭撻を戴きました本学情報科学研究科情報基盤システム学研究室の垣内正年助教に感謝致します.

本論文の執筆にあたり，基本的な論文の構成や，研究の進め方など基礎的で，且つ重要な事柄について惜しみない御援助を戴きました，本学情報科学研究科情報基盤システム学研究室の大平健司特任助教に感謝の意を表します.

本研究，或いは自身の研究活動に於いて，度々有用な御助言，御指摘を戴きました，本学情報科学研究科情報基盤システム学研究室の油谷曉助教，大分大学工学部知能情報システム工学科計算機システム第 2 研究室の池部実助教に感謝致します.

本研究を通して，情報基盤システム学研究室の皆様には多大なる御援助，励ましを戴き，心より感謝致します．岡本慶大氏，野口悟氏に於かれましては日々のミーティング，セミナーに於いて鋭く有用な御指摘をして戴き，本研究への大き

な助けとなりました．両氏に感謝の意を表します．同期の皆様に於かれましては，研究生活，日常生活における会話を通じて，互いの研究を発展させる事ができました．有難う御座います．後輩の皆様には，自身の拙い研究発表を聞いて頂き，様々な角度から御指摘を戴きました．ここに感謝の意を表します．

二年間，研究生活，学生生活に於いて多大なる御援助を賜り，快適に研究に励ませて戴きました，本学川本理恵，本学田村多佳子両女史に深く感謝致します．

学生生活を通して，学業，本研究に励むにあたり，精神的な面からも大きな支えとなって戴きました，慶應義塾大学大学院メディアデザイン研究科の德山眞実様に心より感謝致します．

最後になりましたが，今日まで長い間研究生活を支えて下さり，研究に打ち込める環境を作って頂いた家族に深謝申し上げます．家族の手助けなしに本論文の完成は有り得ません．本当に有難う御座います．

参考文献

[1] CUDA Zone, http://developer.nvidia.com/category/zone/cuda-zone.

[2] National Institute of Standards and Technology, http://www.nist.gov/index.html.

[3] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren: *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Discrete Mathematics and its Applications. Chapman & Hall/CRC, CRC Press, 2006.

[4] J. Balakrishnan, J. Belding, S. Chisholm, K. Eisenträger, K. E. Stange, and E. Teske: Pairings on Hyperelliptic Curves, CoRR, abs/0908.3731, Available: http://arxiv.org/abs/0908.3731v2, 2009.

[5] P. S. L. M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott: Efficient Pairing Computation on Supersingular Abelian Varieties, *Designs, Codes and Cryptography*, **42**, pp. 239–271, 2007.

[6] N. Benger and M. Scott: Constructing Tower Extensions of Finite Fields for Implementation of Pairing-based Cryptography, In *Proceedings of the Third International Conference on Arithmetic of Finite Fields, WAIFI 2010*, pp. 180–195, 2010.

[7] I. Duursma and H.-S. Lee: Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$, *Advances in Cryptography - Asiacrypt 2003, Lecture Notes in Computer Science*, **2894**, pp. 111–123, Springer-Verlag, 2003.

[8] S. D. Galbraith, J. Pujolas, C. Ritzenthaler, and B. A. Smith: Distortion Maps for Genus Two Curves, Available: http://uk.arxiv.org/abs/math.NT/0611471, 2006.

[9] R. Granger, F. Hess, R. Oyono, N. Thériault, and F. Vercauteren: Ate Pairing on Hyperelliptic Curves, *Advances in Cryptology - Eurocrypt 2007, Lecture Notes in Computer Science*, **4515**, pp. 419–436, Springer-Verlag, 2007.

[10] F. Hess: Pairing Lattices, *Pairing 2008, Lecture Notes in Computer Science*, **5209**, pp. 18–38, Springer-Verlag, 2008.

[11] M. Ishii, A. Inomata, and K. Fujikawa: A Study of Parallel Computation of Pairings on Hyperelliptic Curves with Pairing-friendly Fields, In *Proceedings of Computer Security Symposium **2012***, No. 3, pp. 843–848, 2012.

[12] M. Ishii, M. Shirase, A. Inomata, and K. Fujikawa: A Change of Basis of a Pairing-friendly Field, In *Proceedings of the 2013 Symposium on Cryptography and Information*, 2013.

[13] Y. Katoh, C. Cheng Y. Huang, and T. Takagi: Efficient Implementation of the EtaT Pairing on GPU, In *9th International Conference on Applied Cryptography and Network Security, ACNS 2011, Industrial Track*, pp. 119–133, 2011.

[14] N. Koblitz: Elliptic Curve Cryptosystems, *Math. Comp.*, **48**, pp. 203–209, 1987.

[15] N. Koblitz: Hyperelliptic Cryptosystems, *Journal of Cryptography*, **1**, pp. 139–150, 1989.

[16] N. Koblitz: *Algebraic Aspects of Cryptography*, Algorithms and Computation in Mathematics, **3**, Springer-Verlag, Berlin, 1998.

[17] N. Koblitz and A. Menezes: Pairing-based Cryptography at High Security Levels, *Cryptography and Coding 2005, Lecture Notes in Computer Science*, **3796**, pp. 13–36, 2005.

[18] R. Lidl and H. Niederreiter: *Finite Fields*, Encyclopedia of Mathematics and its Applications **20**, Cambridge University Press, Cambridge, UK, 1997.

[19] Y. Nogami, R. Namba, and Y. Morikawa: Finding a Basis Conversion Matrix via Prime Gauss Period Normal Basis, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **92**(6), pp. 1500–1507, 2009.

[20] Y. Nogami, A. Saito, and Y. Morikawa: Finite Extension Field with Modulus of All-one Polynomial and Representation of its Elements for Fast Arithmetic Operations, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E86-A**(9), pp. 2376–2387, 2003.

[21] National Institute of Standards and Computer Security Division Technology: Special Publications 800-57, Available: `http://csrc.nist.gov/publications/PubsSPs.html`.

[22] V. Shoup: A Library for doing Number Theory, Available: `http://www.shoup.net/ntl/`.

[23] J. Silverman: *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics, **106**, Springer-Verlag, Berlin, 1986.

[24] B. Sunar: An Efficient Basis Conversion Algorithm for Composite Fields with Given Representations, *IEEE Transactions on Computers*, **54**, pp. 992–997, IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[25] F. Vercauteren: Optimal Pairings, *IEEE Transactions on Information Theory*, **56**(1), pp. 455–461, 2010.

[26] A. Weimerskirch and C. Paar: Generalization of the Karatsuba Algorithm for Efficient Implementations, Cryptology ePrint Archive, Report 2006/224, Available: `http://uk.arxiv.org/abs/math.NT/0611471`, 2006.