耐量子計算機署名 ModFalcon の
高速な多項式乗算導入による速度評価

東京都立大学大学院理学研究科数理科学専攻
21843421 福原大毅

2023 年 1 月 10 日

目次

第 1 章

はじめに

近年社会の IT 化に伴い，情報通信はますます活性化しており情報セキュリティは以前にも増してより重要な役割を担っている．情報通信において，その信頼性を担保するために，主に電子署名などの認証技術が使われている．現在，電子署名には素因数分解問題を安全性の根拠とした RSA 暗号や，楕円曲線上の離散対数問題を安全性の根拠とした楕円曲線暗号を利用した署名が広く用いられている．しかし，量子計算機に関する研究の進展により，RSA 暗号や楕円曲線暗号が近い将来破られる危険性が指摘されている．その影響により，従来の電子署名において，データの改竄やなりすましなど，署名の完全性が損なわれる可能性がある．量子計算機に対するセキュリティは未完成であり，耐量子計算機暗号方式の実装が急がれている．

耐量子暗号はいくつか考案されており，「格子暗号」と呼ばれるものが最も安全性が高いとされている．格子暗号は格子点の探索問題に安全性の根拠を置いた暗号方式である．格子点探索問題としては，例えば以下の 2 つが知られている．

- SVP 問題 (Shortest Vector Problem)
  n 次元格子 L の基底 $\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}$ とベクトル $\boldsymbol{t} \in \mathbb{R}$ が与えられた時，格子上のユークリッドノルムが最も短い非零ベクトルを求めよ．
- CVP 問題 (Closest Vector Problem)
  n 次元格子 L の基底 $\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}$ とベクトル $\boldsymbol{t} \in \mathbb{R}$ が与えられた時， t に最も近いベクトルを求めよ．

これらの格子点探索問題は量子計算機を用いても，求解が困難である．そのため，現在これらの問題に安全性の根拠を置いた暗号の研究が進められている．

また，この暗号理論を応用し，量子計算機に対してもセキュアな通信を実現する技術の開発も進められている．その 1 つとして，耐量子計算機署名 **ModFalcon**[1] が Chuengsatiansup らによって提案されている．ModFalcon は，同じく耐量子計算機署名である **Falcon**[2] をベースとしており，ランクの大きな module 格子を組み込むことで署名サイズを小さくすることを可能にした．しかし，ModFalcon はスクリプト実装での評価のみで実環境での評価がなされていない．また，スクリプト実装での評価では，多項式乗算を Karatsuba 法や FFT による基本的な実装により行っており，高速化を検討していない．そこで本稿では，高速な多項式乗算を組み込み，ModFalcon の実環境での評価を行った．本稿では，まず第 2 章でデジタル署名の仕組みについて記述する．その後第 3 章で ModFalcon の基礎とそれに関する性質について述べた後，第 4〜7 章でスクリプト実装に用いられたアルゴリズムや Toom-Cook 法および Radix4 FFT による高速な多項式乗算の導入について記述する．最後に，第 8 章でその実装結果について報告する．評価には，1.2GHz デュアルコア Intel Core m3，メモリ 16GB，MacOS 11.4 Big Sur の PC を用いた．

第 2 章

デジタル署名

この章では，デジタル署名の概要について説明する．**デジタル署名 (digital signature)** とは，データが送られてきた際に，データの改竄がなされていないかや，間違いなく本人から送信されてきたものであるかを確認する技術である．公開鍵暗号の技術を応用して作られており，書面上の署名と同等のセキュリティを担保するために用いられる．以下，デジタル署名の仕組みを図 2.1 とともに記述する．

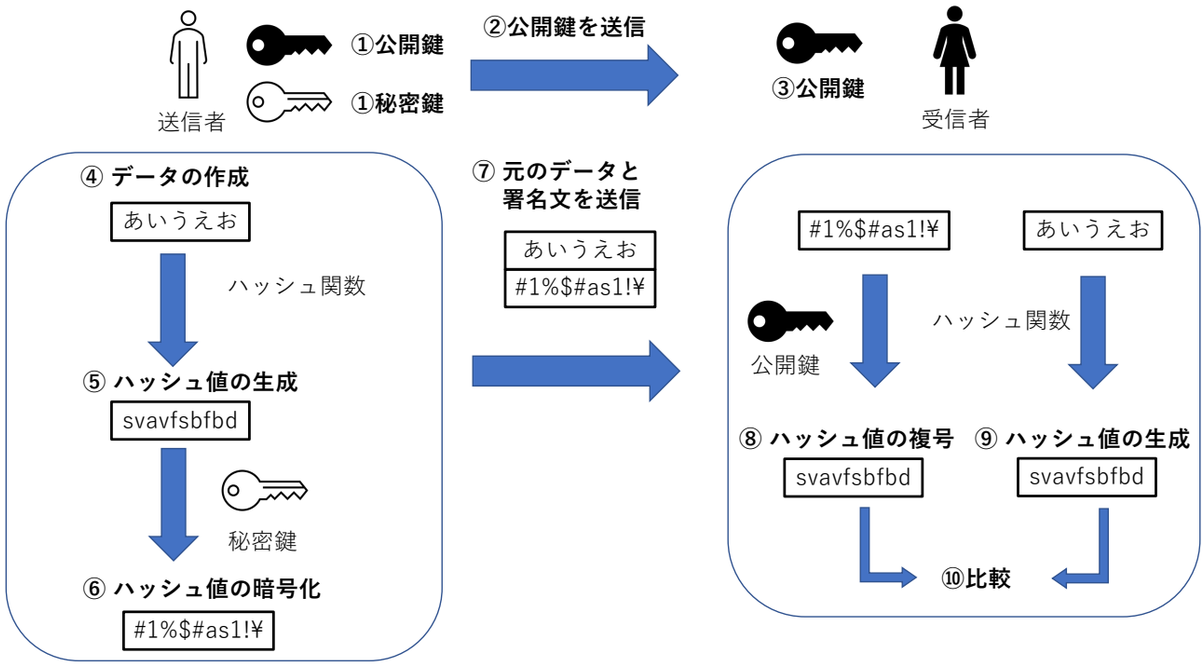

図 2.1　デジタル署名の仕組み

1. 送信者は秘密鍵と公開鍵を生成する．
2. 送信者は受信者に公開鍵を送信する．
3. 受信者は公開鍵を入手する．
4. 送信者が送信したいデータを作成する．

5. 作成したデータからハッシュ値を生成する．
6. ハッシュ値を秘密鍵により暗号化する．
7. データと暗号化されたハッシュ値を署名文として送信する．
8. 受信者は暗号化されたハッシュ値を公開鍵により復号化する．
9. 受信者は受信データをもとに送信者と同じハッシュ関数によりハッシュ値を生成する．
10. 8 の値と 9 の値を比較し，一致すれば正しいデータと確認できる．

暗号化されたデータを複号化できるのは，公開鍵を持っているものだけである．そのため，第三者に暗号化されたデータを盗まれたとしても，復号化することはできないので，改竄やなりすましを防ぐことができる．以上がデジタル署名の仕組みである．現在では，楕円曲線暗号や RSA 暗号などを利用した署名技術が普及している．しかし，近年量子計算機の研究の進展により，それらの完全性が損なわれる危険性が指摘されている．そのため，量子計算機にも耐性のある署名技術の研究が急がれている．

第 3 章

ModFalcon

この章では，耐量子計算機署名 ModFalcon の概要とそれに関するいくつかの性質，またその前身である Falcon の概要について紹介する．

3.1 Falcon

耐量子計算機署名として，Falcon が提案されている．Falcon は，NTRU 格子と Fast-Fourier-Sampling を組み込んだ GPV 署名であり，安全性と効率的な運用の実現という観点において高いレベルにある．

3.1.1 module 格子と NTRU 格子

定義 3.1. 線形独立なベクトル $\boldsymbol{b_1}, \cdots, \boldsymbol{b_n} \in \mathbb{R}^m$ の整数係数結合全体の集合 $L(\boldsymbol{b_1}, \cdots, \boldsymbol{b_n}) := \sum_{i=0}^{n} v_i \boldsymbol{b_i}$ $(v_i \in \mathbb{Z})$ を**格子 (lattice)** と呼ぶ.

定義 3.2. 多項式環 $R := \mathbb{Z}[x]/(x^d + 1)$ に対し，R^k の部分集合 M が以下を満たすとき，M は **R-module** であるという．ここで，1_R は R の乗法に関する単位元を表す.

- $\forall m \in M$, $1_R \cdot m = m$
- $\forall r, s \in R$, $\forall m \in M$, $r(sm) = (rs)m$
- $\forall r, s \in R$, $\forall m \in M$, $(r + s)m = rm + sm$
- $\forall r \in R$, $\forall m, n \in M$, $r(m + n) = rm + rn$

R に対して多項式の係数を $\mathbb{Z}^d$ に写す写像 $\phi : R \to \mathbb{Z}^d$ を定義すると，この ϕ を用いて，$(a_1, a_2, \ldots, a_k) \in R^k$ を $(\phi(a_1), \phi(a_2), \ldots, \phi(a_k)) \in \mathbb{Z}^{d \cdot k}$ に写す写像 $\psi : R^k \to \mathbb{Z}^{d \cdot k}$ を定義できる．このとき $\psi(M) \subset \mathbb{Z}^{d \cdot k}$ は格子となり，このような格子を **module 格子 (module lattice)** と呼ぶ．また，k を**ランク**と呼ぶ．本稿ではランク 2 の module 格子を **NTRU 格子 (NTRU lattice)** と呼ぶ.

Falcon は NTRU 格子上の **SIS 問題 (Small Integer Solution Problem)** を安全性の根拠としている．SIS 問題とは，行列 $A \in \mathbb{Z}^{n \times m}$ が与えられた時， 次の条件を満たす小さいベクトル $\boldsymbol{e} \in \mathbb{Z}_q^m$ を求めよという問題である.

- $A\boldsymbol{e} = \boldsymbol{0} \pmod q$
- $\boldsymbol{e} \neq \boldsymbol{0}$

この問題は，$\Lambda_q^{\perp}(A^T) := \{\boldsymbol{x} \in \mathbb{Z}^m | A\boldsymbol{x} = 0 \pmod q\}$ 上の SVP 問題と見ることができる．つまり，SVP 問題が解けなければ SIS 問題を解くことはできない.

3.1.2 GPV 署名

2008 年に Gentry，Peikert，Vaikuntanathangs は安全な格子ベースの署名を得るための枠組みである **GPV フレームワーク** [2] を確立した．その内容は以下の通りである．

- 公開鍵は q-ary 格子 Λ を生成するフルランク行列 $A \in \mathbb{Z}_{n\times m}$ $(m > n)$ を含む．ここで，q-ary 格子 Λ とは，$\mathbb{Z}^n$ に埋め込まれた格子 Λ がある整数 q に対し $q\mathbb{Z} \subset \Lambda$ を満たすことである．
- 秘密鍵は $\Lambda_q^{\perp}$ を生成する行列 B を含む．ここで，$\Lambda_q^{\perp}$ は，mod q で Λ と直交することを表す．すなわち，任意の $\boldsymbol{x} \in \Lambda, \boldsymbol{y} \in \Lambda_q^{\perp}$ に対して，$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0 \pmod q$ が成り立つことである．
- 平文 m に対し，m の署名は $\boldsymbol{s}A^t = H(m)$ となるような短いベクトル $\boldsymbol{s} \in \mathbb{Z}_q^m$ である．ここで，$H : \{0,1\}^* \to \mathbb{Z}_q^n$ はハッシュ関数である．A が与えられれば，$\boldsymbol{s}$ が有効な署名であることを検証するには，$\boldsymbol{s}$ が短く，$\boldsymbol{s}A^t = H(m)$ であることを確認すれば良い．

GPV フレームワークは，SIS 問題のもと，ランダムオラクルモデルにおいて安全であることが証明されている．ランダムオラクルとは，入力されたある値に対して一様に分布するランダムな値を返すが，同じ入力値に対して同じ値を出力するものである．また，安全性を証明する際にランダムオラクルとしてハッシュ関数を用いている場合，ランダムオラクルモデルにおいて安全という．

GPV フレームワークに基づいた署名を，**GPV 署名**と呼ぶ．GPV フレームワークを用いるには，

1. 安全性の根拠となる SIS 問題を特徴できる格子，
2. トラップドアサンプリングアルゴリズム

が必要である．Falcon では，1 として NTRU 格子を，2 として Fast-Fourier-Sampling を採用した．

3.1.3 Fast-Fourier-Sampling

トラップドアサンプリングにはいくつか知られているものがあるが，Falcon には，

1. 処理速度が速い，
2. 出力するベクトルが短い，
3. NTRU 格子との相性が良い

という 3 つの条件を満たすトラップドアサンプリングを組み込む必要があった．Fast-Fourier-Sampling はそれらの条件を全て満たしているため，このアルゴリズムが採用された．Fast-Fourier-Sampling の詳しいアルゴリズムについては，[2] を参照したい．

3.1.4 Falcon の特徴

Falcon には，以下のような特徴がある．

- 公開鍵と署名文のデータサイズの合計値が小さい．
- NTRU 格子を導入することで，鍵生成を効率よく行うことができる．
- サイドチャネル攻撃にも耐性がある．

実際に他の耐量子計算機署名の公開鍵と署名文のデータサイズの比較をした表を示す．

方式	公開鍵 (byte)	署名文 (byte)	合計 (byte)
Falcon-1024	1793	1274	3067
Rainbow	148992	64	149056
Dilithium3	1472	2701	4173

表 3.1　各署名方式のデータサイズ

表 3.1 のように，他の署名方式に比べて，Falcon は公開鍵と署名文のデータサイズの合計値が小さいことがわかる．

ただし，Falcon には欠点がある．Falcon は多項式環 $R = \mathbb{Z}[x]/(x^d + 1)$ (d は 2 の冪乗) を用いている．その影響で，ビットセキュリティは d に依存しており，安全性を高めようとした場合に d の値が過剰になり，効率が著しく低下する．例えば，$d = 512$ から，$d = 1024$ とすると，保証できる量子セキュリティビットは 103 ビットから 230 ビットになるが，同時に署名文のデータサイズも 617 バイトから 1233 バイトまで跳ね上がる．セキュリティとデータサイズの間での妥協点を探すためには，適切な環を選ぶ必要があるが，R のように便利な環はほとんど存在しない．そのため，R はそのままにして，上記の欠点を解決する ModFalcon の研究が進められている．

3.2　ModFalcon

ModFalcon は Chuengsatiansup [1] らによって提案された耐量子計算機署名である．ModFalcon は Falcon をベースとし，NTRU 格子よりもランクの高い module 格子上の SIS 問題に安全性の根拠をおいている．ランクの大きな module 格子を組み込むことで，パラメータ選択の自由度が高まる．そのため，異なるセキュリティビットに対して Falcon よりも署名文のデータサイズがより小さくて済む．このように，Falcon の欠点を解決した．

3.2.1　ModFalcon のアルゴリズム

ModFalcon は，鍵生成，署名，検証の 3 つのアルゴリズムからなる．

Algorithm 1 鍵生成

1: $R = \mathbb{Z}[x]/(x^d + 1)$
2: repeat
3: 　sample $R^{n\times n} \ni \boldsymbol{F} \leftarrow D_f, R^n \ni \boldsymbol{g} \leftarrow D_g$
4: until $\boldsymbol{F}$ invertible mod q
　and $\|M(\boldsymbol{B_{F,g}})\|_{GS} \leq GS\_SLACK \cdot q^{\frac{1}{n+1}}$
5: $R^n \ni f_0 \leftarrow D_{f_0}, R \ni g_0 \leftarrow D_{g_0}\ s.t.\ \det\boldsymbol{F} - \boldsymbol{f_0} \cdot \tilde{\boldsymbol{F}} \cdot \boldsymbol{g}^t$
6: $\boldsymbol{B_{F,g}} \leftarrow \begin{pmatrix} \boldsymbol{g}^t & -\boldsymbol{F} \\ g_0 & \boldsymbol{f_0} \end{pmatrix}$
7: compute $\boldsymbol{h}^t = \boldsymbol{F}^{-1} \cdot \boldsymbol{g}^t \bmod q \in R^n$
8: return　$vk = \begin{pmatrix} 1 \\ \boldsymbol{h}^t \end{pmatrix},\ sk = \boldsymbol{B_{F,g}}$

Algorithm 1 の vk が公開鍵，sk が秘密鍵にあたる．また，$\|\boldsymbol{B_{F,g}}\|_{GS}$ の下限は $q^{\frac{1}{n+1}}$ であり，セキュリティの観点からできるだけ小さいことが好ましい．GS_SLACK を式 (3.1) のように設定し，$\|\boldsymbol{B_{F,g}}\|_{GS}$ の上限を $GS\_SLACK \cdot q^{\frac{1}{n+1}}$ とすることが最適であると実験的にわかっている．

$$\begin{aligned} n = 1, 2 &: GS\_SLACK = 1.17 \\ n = 3 &: GS\_SLACK = 1.24 \end{aligned} \tag{3.1}$$

Algorithm 2 署名

$\boldsymbol{r} \leftarrow \{0,1\}^{\epsilon}$
$\mu \leftarrow H(\boldsymbol{r}||\boldsymbol{msg})$
$\boldsymbol{c} \leftarrow (\mu, 0, \cdots, 0) \in R^{n+1}$
$\boldsymbol{t} \leftarrow \boldsymbol{c} \cdot \boldsymbol{B}_{\boldsymbol{F},\boldsymbol{g}}^{-1}$
compute $z \in R^{n+1}$ $s.t.$ $\boldsymbol{s} = (\boldsymbol{t} - \boldsymbol{z}) \cdot \boldsymbol{B}_{\boldsymbol{F},\boldsymbol{g}}^{-1} \leftarrow \boldsymbol{D}_{L_{NTRU}}$
$S \leftarrow Compress(\boldsymbol{s})$
$sig \leftarrow (\boldsymbol{r}, S)$
return sig

Algorithm 3 検証

$\boldsymbol{s} \leftarrow Decompress(S)$
If $|\boldsymbol{s}| > \rho \vee \boldsymbol{s} \cdot \boldsymbol{vk} \neq H(\boldsymbol{r}||\boldsymbol{msg})$
return $False$
else
return $True$

H はハッシュ関数である．また，$Compress$，$Decompress$ は符号化関数とよばれ，$Compress : R^{n+1} \rightarrow \{0,1\}^*$，$Decompress : \{0,1\}^* \rightarrow R^{n+1}$ であり，$Decompress \circ Compress$ は恒等写像となる．

実装においては，最適化されていれば Falcon と同等の時間で実行が可能である．しかし，ModFalcon はスクリプト実装のみであり，実環境での評価がなされておらず，Falcon に比べて高速化検討が十分になされていない．高速化検討前の ModFalcon と Falcon のタイミングデータを以下に示す (表 3.2).

アルゴリズム	鍵生成 (s)	署名 (s)	検証 (s)
Falcon-1024	0.024	0.637×10^{-3}	0.136×10^{-3}
ModFalcon	1.335	3.256×10^{-3}	3.892×10^{-3}

表 3.2　計算時間の比較

表 3.2 から，Falcon に比べて 3 つすべてのアルゴリズムにおいて，低速であることがわかる．Algorithm 1,2,3 における R は多項式環であるため，各アルゴリズムにおいて多項式の演算の占める割合が高い．そのため，高速化のために多項式乗算に着目して取り組んだ．

第 4 章

Karatsuba 法

4.1 はじめに

Karatsuba 法とは，多倍長整数や多項式の乗算に用いられるアルゴリズムであり，計算量を通常の乗算の約 4 分の 3 倍に抑えられる．ModFalcon のスクリプト実装における多項式乗算には Karatsuba 法や Radix2 FFT が用いられている．この章では，Karatsuba 法のアルゴリズムについて紹介する．FFT については 6 章で記述する．

4.2 Karatsuba 法のアルゴリズム

n 次多項式 $f(x), g(x)$ に対し，$z(x) = f(x) \cdot g(x)$ を計算する．
$f(x), g(x)$ を $\frac{n}{2}$ 次以下の多項式 $a(x), b(x), c(x), d(x)$ を用いて以下のように表す．

$$f(x) = a(x) \cdot x^{\frac{n}{2}} + b(x) \tag{4.1}$$

$$g(x) = c(x) \cdot x^{\frac{n}{2}} + d(x) \tag{4.2}$$

2 つの積をとると，

$$f(x) \cdot g(x) = a(x) \cdot c(x) \cdot x^n + (a(x) \cdot d(x) + b(x) \cdot c(x)) \cdot x^{\frac{n}{2}} + b(x) \cdot d(x) \tag{4.3}$$

となり，$\frac{n}{2}$ 次以下の多項式の乗算に帰着できる．同様の操作を $a(x), b(x), c(x), d(x)$ に再帰的に施していけば，$f(x)$ と $g(x)$ の乗算を求めることができる．この方法は，古典的な多項式乗算を再帰的に表したものであり，1 度の操作に 4 回の乗算，3 回の加算を必要とする．Karatsuba 法では，式 (4.3) を下の様に変形する．

$$f(x) \cdot g(x) = a(x) \cdot c(x) \cdot x^n + \{a(x) \cdot c(x) + b(x) \cdot d(x) + (a(x) - b(x)) \cdot (d(x) - c(x))\} \cdot x^{\frac{n}{2}} + b(x) \cdot d(x) \tag{4.4}$$

このように変形することで，$a(x) \cdot c(x)$ および $b(x) \cdot d(x)$ の値が再使用できるため，1 度の操作に 3 回の乗算，6 回の加算で計算を行える．通常，加算にかかる時間的なコストは乗算よりも軽いため，高速に計算できる．計算量は古典的な方法が $O(n^2)$ なのに対して，Karatsuba 法は $O(n^{\log_2 3})$ と，その約 4 分の 3 倍に抑えられる．

4.3 疑似コード

Karatsuba 法のアルゴリズムの疑似コードを 4 に示す．

Algorithm 4 Karatsuba 法 (a, b, n)

Require: 多項式 a, b, 項数 n

Ensure: 多項式 a, b の積 ab

if $n = 1$ **then**
  $ab = [a[0] \cdot b[0], 0]$
else
  for $0 \leq i < \frac{n}{2}$ **do**
    $a0[i] \leftarrow a[i]$
    $a1[i] \leftarrow a[i + \frac{n}{2}]$
    $b0[i] \leftarrow b[i]$
    $b1[i] \leftarrow b[i + \frac{n}{2}]$
    $ax[i] \leftarrow a0[i] + a1[i]$
    $bx[i] \leftarrow b0[i] + b1[i]$
    $a0b0 \leftarrow karatsuba(a0, b0, \frac{n}{2})$
    $a1b1 \leftarrow karatsuba(a1, b1, \frac{n}{2})$
    $axbx \leftarrow karatsuba(ax, bx, \frac{n}{2})$
  end for
  for $0 \leq i < n$ **do**
    $axbx[i] \leftarrow axbx[i] - (a0b0[i] + a1b1[i])$
  end for
  for $0 \leq i < 2n$ **do**
    $ab[i] \leftarrow 0$
  end for
  for $0 \leq i < n$ **do**
    $ab[i] \leftarrow ab[i] + a0b0[i]$
    $ab[i + n] \leftarrow ab[i + n] + a1b1[i]$
    $ab[i + \frac{n}{2}] \leftarrow ab[i + \frac{n}{2}] + axbx[i]$
  end for
end if
return ab

第 5 章

Toom-Cook 法を利用した多項式乗算の高速化

5.1 はじめに

Toom-Cook 法とは多倍長の整数の乗算において乗数，被乗数をそれぞれ l 次，m 次の多項式と見做して乗算を行う方式であり，1963 年に A. L. Toom によって基本的な考え方が発表され，1966 年に S. A. Cook により計算機で作譜する方法が発表された．

Toom-Cook 法は乗数 A，被乗数 B を基数 x $(= 2^k)$ を用いてそれぞれ l 次，m 次の多項式 $a(x), b(x)$，その積を $f(x) = a(x) \cdot b(x)$ とし，$l + m + 1$ 個の値 $f(x_i) = a(x_i) \cdot b(x_i)$ $(0 \leq i \leq l + m)$ から $f(x)$ の係数を求め，x を代入することで $A \cdot B$ を求めるという手法である．分割数の取り方は一意ではなくこのような方式をまとめて Toom-Cook 法と呼ぶ．

5.2 Vandermonde 行列

定義 5.1. A を $n \times n$ 行列とする．$1 \leq \forall i \leq n$ $(n \in \mathbb{N})$ に対し，i 行目が $1, x_i, x_i^2, \cdots, x_i^{n-1}$ となるとき，A を **Vandermonde 行列**という．

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix}$$

定理 5.2 (Vandermonde 行列式). 定義 5.1 の Vandermonde 行列 A に対し，A の行列式 $\det A$ は，以下の式で与えられる．

$$\det A = \prod_{1 \leq i < j \leq n} (x_j - x_i)$$

証明. 帰納法を用いて証明する．

(i) $n = 2$ の時，$\det A = x_2 - x_1$ より，成立する．

(ii) $n = k - 1$ の時に，定理 5.2 が成り立つと仮定する．

$n = k$ の時，$\det A = \begin{vmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^{k-1} \end{vmatrix}$ である．

以下の手順で $\det A$ を変形させ，1 行目の $(1, 1)$ 成分以外を 0 にする．

1) 第 $k - 1$ 列の $-x_1$ 倍を第 k 列に加える．

2) 第 $k - 2$ 列の $-x_1$ 倍を第 $k - 1$ 列に加える．

$\vdots$

$k-2$) 第 2 列の $-x_1$ 倍を第 3 列に加える．

$k-1$) 第 1 列の $-x_1$ 倍を第 2 列に加える．

上記の操作を行うことで，

$$\det A = \begin{vmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & x_2 - x_1 & x_2(x_2 - x_1) & \cdots & x_2^{k-2}(x_2 - x_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k - x_1 & x_k(x_k - x_1) & \cdots & x_k^{k-2}(x_k - x_1) \end{vmatrix}$$ となる．

1 行についての余因子展開，各行の共通因子の括り出しを行うと，

$$\det A = (x_2 - x_1) \times (x_3 - x_1) \times \cdots \times (x_k - x_1) \begin{vmatrix} 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^{k-1} \end{vmatrix}$$ となる．

よって，$\det A = (x_2 - x_1) \times \cdots \times (x_k - x_1) \prod_{2 \le i < j \le k} (x_j - x_i) = \prod_{1 \le i < j \le k} (x_j - x_i)$ となり，$n = k$ で成立する．

以上より定理 5.2 が成立する． □

系 5.3. Vandermonde 行列 A $(n \times n)$ において，$1 \le \forall i, j \le n$ $(i \ne j)$ に対し，$x_i \ne x_j \Leftrightarrow \det A \ne 0$.

証明. $\Rightarrow$) $x_j - x_i \ne 0$ $(1 \le i < j \le n)$ となるから $\prod_{1 \le i < j \le k} (x_j - x_i) \ne 0$．つまり $\det A \ne 0$.

$\Leftarrow$) 対偶「$1 \le \exists i, j \le n$ $(i \ne j), x_i = x_j \Rightarrow \det A = 0$」が真であることを示す．

$x_i = x_j$ $(i < j)$ を満たす x_i, x_j が存在すると仮定すると，$\exists i, j$ $(i < j), x_j - x_i = 0$ となる．よって $\prod_{1 \le i < j \le k} (x_j - x_i) = 0$．つまり $\det A = 0$．したがって対偶が真であるから元の命題も真である．

□

5.3 Toom-$(l+1)$ 法の概略

Toom-$(l+1)$ 法を用いた多倍長整数の乗算方法について紹介する．2 つの整数 a, b が基数 $P = 2^k$ を用いて (5.1)，(5.2) のように P 進表現されるものとする．

$$a = a_l P^l + a_{l-1} P^l + \cdots + a_1 P + a_0 \ (0 \le a_i < P) \tag{5.1}$$

$$b = b_l P^l + b_{l-1} P^l + \cdots + b_1 P + b_0 \ (0 \le b_i < P) \tag{5.2}$$

ここで l 次多項式 $A(x)$, $B(x)$

$$A(x) = a_l x^l + a_{l-1} x^{l-1} + \cdots + a_1 x + a_0$$

$$B(x) = b_l x^l + b_{l-1} x^{l-1} + \cdots + b_1 x + b_0$$

を考え，その積を $2l$ 次多項式 $C(x)$,

$$C(x) = c_{2l} x^{2l} + c_{2l-1} x^{2l-1} + \cdots + c_1 x + c_0$$

とする．ここで適当な $2l+1$ 個の異なる有理数 $\left(\frac{n_0}{d_0}, \frac{n_1}{d_1}, \cdots, \frac{n_{2l}}{d_{2l}} \right)$ を $C(x)$ に代入すると，

$$C\left(\frac{n_0}{d_0}\right) = c_{2l} \left(\frac{n_0}{d_0}\right)^{2l} + c_{2l-1} \left(\frac{n_0}{d_0}\right)^{2l-1} + \cdots + c_0$$

$$\vdots$$

$$C\left(\frac{n_{2l-1}}{d_{2l-1}}\right) = c_{2l} \left(\frac{n_{2l-1}}{d_{2l-1}}\right)^{2l} + c_{2l-1} \left(\frac{n_{2l-1}}{d_{2l-1}}\right)^{2l-1} + \cdots + c_0$$

$$C\left(\frac{n_{2l}}{d_{2l}}\right) = c_{2l}\left(\frac{n_{2l}}{d_{2l}}\right)^{2l} + c_{2l-1}\left(\frac{n_{2l}}{d_{2l}}\right)^{2l-1} + \cdots + c_0$$

となり，$C(x)$ の各係数を変数とする $2l+1$ 変数多項式が $2l+1$ 個できる．これを連立して解けば $C(x)$ の各係数を求めることができる．この連立方程式を解く方法はいくつかあるが，本稿では行列を用いた方法を用いる．上記の連立方程式を行列を用いて表すと，

$$\begin{pmatrix} C\left(\frac{n_0}{d_0}\right) \\ \vdots \\ C\left(\frac{n_{2l-1}}{d_{2l-1}}\right) \\ C\left(\frac{n_{2l}}{d_{2l}}\right) \end{pmatrix} = \begin{pmatrix} 1 & \frac{n_0}{d_0} & \cdots & \left(\frac{n_0}{d_0}\right)^{2l-1} & \left(\frac{n_0}{d_0}\right)^{2l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \frac{n_{2l-1}}{d_{2l-1}} & \cdots & \left(\frac{n_{2l-1}}{d_{2l-1}}\right)^{2l-1} & \left(\frac{n_{2l-1}}{d_{2l-1}}\right)^{2l} \\ 1 & \frac{n_{2l}}{d_{2l}} & \cdots & \left(\frac{n_{2l}}{d_{2l}}\right)^{2l-1} & \left(\frac{n_{2l}}{d_{2l}}\right)^{2l} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{2l-1} \\ c_{2l} \end{pmatrix} \tag{5.3}$$

となる．ここで (5.3) の右辺の左側に現れる行列は，5.2 節で紹介した Vandermonde 行列である．$\frac{n_i}{d_i} \neq \frac{n_j}{d_j}$ $(\forall i, j)$ であるから $a \neq 0$ であり，逆行列が存在する．その逆行列を両辺にかけると，

$$\begin{pmatrix} c_0 \\ \vdots \\ c_{2l-1} \\ c_{2l} \end{pmatrix} = \begin{pmatrix} 1 & \frac{n_0}{d_0} & \cdots & \left(\frac{n_0}{d_0}\right)^{2l-1} & \left(\frac{n_0}{d_0}\right)^{2l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \frac{n_{2l-1}}{d_{2l-1}} & \cdots & \left(\frac{n_{2l-1}}{d_{2l-1}}\right)^{2l-1} & \left(\frac{n_{2l-1}}{d_{2l-1}}\right)^{2l} \\ 1 & \frac{n_{2l}}{d_{2l}} & \cdots & \left(\frac{n_{2l}}{d_{2l}}\right)^{2l-1} & \left(\frac{n_{2l}}{d_{2l}}\right)^{2l} \end{pmatrix}^{-1} \begin{pmatrix} C\left(\frac{n_0}{d_0}\right) \\ \vdots \\ C\left(\frac{n_{2l-1}}{d_{2l-1}}\right) \\ C\left(\frac{n_{2l}}{d_{2l}}\right) \end{pmatrix} \tag{5.4}$$

となる．$C(x) = A(x) \cdot B(x)$ であるから右辺のベクトルの各成分 $C\left(\frac{n_i}{d_i}\right)$ $(0 \leq i \leq 2l)$ は $C\left(\frac{n_i}{d_i}\right) = A\left(\frac{n_i}{d_i}\right) \cdot B\left(\frac{n_i}{d_i}\right)$ $(0 \leq i \leq 2l)$ で求めることができる．また右辺の逆行列はあらかじめ計算可能である．よって (5.4) を計算し，$C(x)$ の各項の係数を求め，$C(x)$ に $x = P$ を代入することで積 $a \cdot b$ を求めることができる．Toom-$(l+1)$ 法を用いて多項式乗算を行う際の係数同士の乗算回数は $C\left(\frac{n_i}{d_i}\right) = A\left(\frac{n_i}{d_i}\right) \cdot B\left(\frac{n_i}{d_i}\right)$ を求める $2l+1$ 回である．

n 次多項式 $A(x)$, $B(x)$ の乗算 $A(x) \cdot B(x) = C(x)$ を Toom-$(l+1)$ 法で行うアルゴリズムを Algorithm 5 に示す．以下, 疑似コード内では, $A(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, $B(x) = b_n x^n + b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$ とするとき, 長さが $n+1$ の配列 A, B を用いて $A(x)$, $B(x)$ を $A = [a_n, a_{n-1}, \ldots, a_1, a_0]$, $B = [b_n, b_{n-1}, \ldots, b_1, b_0]$ とし，$A \cdot B = C = [c_{2n}, c_{2n-1}, \ldots, c_1, c_0]$ を考える．

Algorithm 5 Toom-$(l+1)(A, B)$

Require: A, B は長さ $n+1$ の配列

Ensure: 積 $A \cdot B = C$ は長さ $2n+2$ の配列

if $n = 1$ **then**
  return $A[0] \cdot B[0]$
end if
$o = \lfloor \frac{n}{l+1} \rfloor$
$A = [A_l, A_{l-1}, \ldots, A_1, A_0]$
  (A_iは長さ$\frac{n}{l+1}$の配列)
$B = [B_l, B_{l-1}, \ldots, B_1, B_0]$
  (B_iは長さ$\frac{n}{l+1}$の配列)
for $0 \leq i < o$ **do**
  $A'_{2l}[i] = A_l[i] \cdot a_{2l}^l + A_{l-1}[i] \cdot a_{2l}^{l-1} + \cdots + A_0[i]$
  $A'_{2l-1}[i] = A_l[i] \cdot a_{2l-1}^l + A_{l-1}[i] \cdot a_{2l-1}^{l-1} + \cdots + A_0[i]$
    $\vdots$
  $A'_1[i] = A_l[i] \cdot a_1^l + A_{l-1}[i] \cdot a_1^{l-1} + \cdots + A_0[i]$
  $A'_0[i] = A_l[i] \cdot a_0^l + A_{l-1}[i] \cdot a_0^{l-1} + \cdots + A_0[i]$
  ($a_i = \frac{e_i}{d_i}$ $(0 \leq i \leq 2l, e_i, d_i \in \mathbb{Z})$)
end for
B も同様
for $0 \leq i < 2l+1$ **do**
  Toom-$(l+1)(A'_i, B'_i)$
end for
$v = (C'_{2l}, C'_{2l-1}, \ldots, C'_0)^t$
$(c_{2l}, c_{2l-1}, \ldots, c_0)^t = M^{-1} * v$
  (行列 $M(i,j) = a_{i-1}^{j-1}$ $(1 \leq i, j \leq 2l+1)$
for $0 \leq i < 2o$ **do**
  $C[i] = C'_0[i]$
  $C[i+o] = C'_1[i]$
    $\vdots$
  $C[i + (2l-1) \cdot o] = C'_{2l-1}[i]$
  $C[i + 2l \cdot o] = C'_{2l}[i]$
end for
return C

本稿では Toom-4 法と Toom-8 法を多項式乗算に組み込んで高速化を検討した.

5.4 計算量の比較

ModFalcon では主に 511 次多項式の乗算が用いられている．そのため本稿では，乗算する多項式の次数を 511 に固定し，高速化の検討を行った.

ここでは Karatsuba 法，Toom-4 法，Toom-8 法の計算量の比較を行うため，各方式を用いた 511 次多項式の乗算における演算回数の見積もりを立てた．今回計測を行った PC において int 型，double 型の演算 1 回のコストは乗算，加算の間に差がなかったため全て同じコストとしている．(表 5.1)

アルゴリズム	乗算	加算	合計	再帰処理
Karatsuba	19683	230025	249735	29523
Toom-4	60640	79984	140624	399
Toom-8	62792	79168	141960	240

表 5.1　511 次式の乗算における演算回数

既存のスクリプトに用いられている Karatsuba 法に比べ Toom-4 法，Toom-8 法ともに演算を約 11 万回，再帰呼び出しの回数においては約 2 万 9000 回削減できた．Toom-4 法と Toom-8 法の計算量を比較すると Toom-4 法の方が 1336 回少なかった．しかし，Toom-8 法の方が再帰呼び出しの回数が 155 回少なかったため，Toom-4 法より高速化する可能性があると判断した．

第 6 章

FFT を利用した多項式乗算の高速化

6.1 はじめに

この章では，既存のスクリプトにもある FFT の分割数に改良を加えた Radix4 FFT の概要，及び計算量の評価について示す．

高速フーリエ変換 FFT(Fast Fourier Transform) は 1965 年に J. W. Cooley と J. W. Tukey が発見したとされているアルゴリズムである．このアルゴリズムは，分割統治法を使うことで離散フーリエ変換 DFT (Discrete Fourier Transform) を効率的に計算できることで知られている．

定義 6.1. 複素関数 $f = \sum_{k=0}^{N-1} f_k x^k$ ($f_k \in \mathbb{C}$) の DFT $F(f) := F_k$ ($F_k \in \mathbb{C}$) を以下のように定義する．

$$F_k = \sum_{j=0}^{N-1} f_j W_N^{jk} \qquad (W_N = e^{-\frac{2\pi i}{N}})$$

定義 6.2. DFT の逆変換 (Inverse DFT) を以下のように定義する．(以下 IDFT と表記する．)

$$f_k = \frac{1}{N} \sum_{j=0}^{N-1} F_j W_N^{-jk} \qquad (W_N = e^{-\frac{2\pi i}{N}})$$

FFT は主に信号処理などで離散化されたデジタル信号の周波数解析などに使われる．また，畳み込み積分や多項式乗算にも利用されている．本章では，多項式乗算の高速化を目標に，FFT の改良に取り組む．

6.2 DFT を用いた多項式乗算

DFT を利用して，多項式乗算が計算できることが広く知られている．乗算したい $N-1$ 次多項式 $f(x)$, $g(x)$ を，それぞれ $f(x) = \sum_{k=0}^{N-1} f_k x^k$, $g(x) = \sum_{k=0}^{N-1} g_k x^k$ ($f_k, g_k \in \mathbb{R}$) と定義する．$f(x)$, $g(x)$ を定義 6.1 に従ってフーリエ変換したものを $F(f) := F_k$, $F(g) := G_k$ とすると，

$$F_k = \sum_{j=0}^{N-1} f_j W_N^{jk},\ G_k = \sum_{j=0}^{N-1} g_j W_N^{jk}$$

$F(f)$ と $F(g)$ の畳み込み積をとると，

$$F(f) * F(g) = \sum_{j=0}^{N-1} f_j W_N^{jk} \sum_{l=0}^{N-1} g_{l-j} W_N^{(l-j)k} \ (l-j<0 \text{ となる部分は } g_{l-j}=0 \text{ とする}) \tag{6.1}$$

$$= \sum_{l=0}^{N-1} \sum_{j=0}^{N-1} f_j W_N^{jk} g_{j-l} W_N^{(l-j)k} \tag{6.2}$$

$$= \sum_{l=0}^{N-1} \sum_{j=0}^{N-1} f_j g_{l-j} W_N^{lk} \tag{6.3}$$

$$= F(f * g) \tag{6.4}$$

両辺に逆変換 IDFT を施すと，

$$F^{-1}(F(f) * F(g)) = F^{-1}(f * g) = f * g$$

となる．このことから，DFT を利用しての多項式乗算は，

1. 乗算したい多項式を DFT する．
2. DFT したもの同士の畳み込み積を求める．
3. 2 で算出したものに IDFT する．

以上の手順を踏むことで求められることがわかる．この一連の計算量は $O(N^2)$ と，古典的な方法と同等であるが，DFT を高速に計算するアルゴリズムである FFT を用いることで，高速な多項式乗算が実現できる．

6.3 Radix2 FFT

上記の DFT の定義式に対して分割統治法を使うことで，計算量を減らすことができる．ここでは基数 2 の FFT を Radix2 FFT と呼ぶ．DFT の定義式を以下のように 2 分割する．

$$F_k = \sum_{j=0}^{\frac{N}{2}-1} f_{2j} W_{\frac{N}{2}}^{jk} + W_N^k \sum_{j=0}^{\frac{N}{2}-1} f_{2j+1} W_{\frac{N}{2}}^{jk} \tag{6.5}$$

$$F_{k+\frac{N}{2}} = \sum_{j=0}^{\frac{N}{2}-1} f_{2j} W_{\frac{N}{2}}^{jk} - W_N^k \sum_{j=0}^{\frac{N}{2}-1} f_{2j+1} W_{\frac{N}{2}}^{jk} \tag{6.6}$$

DFT における計算量は，$O(N^2)$ であるが, 上記のように分割することで，式 (6.5) で用いた W を式 (6.6) でも再使用することができ，計算量を減らせる．それにより，Radix2 FFT の計算量は $O(\frac{N}{2}\log N)$ に抑えられる．IDFT にも同様な式変形をすることで，計算量を減らすことができる．(以下 Radix2 IFFT と呼ぶ．)

6.4 Radix4 FFT

本稿では，基数 4 の FFT を用いて多項式乗算の高速化を試みた．DFT の定義式を以下のように 4 分割する．

$$F_{4k} = \sum_{j=0}^{\frac{N}{2}-1} f_{4j} W_{\frac{N}{4}}^{jk} + W_{\frac{N}{2}}^k \sum_{j=0}^{\frac{N}{2}-1} f_{4j+2} W_{\frac{N}{4}}^{jk} + W_N^k \sum_{j=0}^{\frac{N}{4}-1} f_{4j+1} W_{\frac{N}{4}}^{jk} + W_N^k W_{\frac{N}{2}}^k \sum_{j=0}^{\frac{N}{4}-1} f_{4j+3} W_{\frac{N}{4}}^{jk}$$

$$F_{4k+1} = \sum_{j=0}^{\frac{N}{2}-1} f_{4j} W_{\frac{N}{4}}^{jk} + W_{\frac{N}{2}}^k \sum_{j=0}^{\frac{N}{2}-1} f_{4j+2} W_{\frac{N}{4}}^{jk} - W_N^k \sum_{j=0}^{\frac{N}{4}-1} f_{4j+1} W_{\frac{N}{4}}^{jk} + W_N^k W_{\frac{N}{2}}^k \sum_{j=0}^{\frac{N}{4}-1} f_{4j+3} W_{\frac{N}{4}}^{jk}$$

$$F_{4k+2} = \sum_{j=0}^{\frac{N}{2}-1} f_{4j} W_{\frac{N}{4}}^{jk} - W_{\frac{N}{2}}^{k} \sum_{j=0}^{\frac{N}{2}-1} f_{4j+2} W_{\frac{N}{4}}^{jk} + W_{N}^{k} \sum_{j=0}^{\frac{N}{4}-1} f_{4j+1} W_{\frac{N}{4}}^{jk} - W_{N}^{k} W_{\frac{N}{2}}^{k} \sum_{j=0}^{\frac{N}{4}-1} f_{4j+3} W_{\frac{N}{4}}^{jk}$$

$$F_{4k+3} = \sum_{j=0}^{\frac{N}{2}-1} f_{4j} W_{\frac{N}{4}}^{jk} - W_{\frac{N}{2}}^{k} \sum_{j=0}^{\frac{N}{2}-1} f_{4j+2} W_{\frac{N}{4}}^{jk} - W_{N}^{k} \sum_{j=0}^{\frac{N}{4}-1} f_{4j+1} W_{\frac{N}{4}}^{jk} - W_{N}^{k} W_{\frac{N}{2}}^{k} \sum_{j=0}^{\frac{N}{4}-1} f_{4j+3} W_{\frac{N}{4}}^{jk}$$

Algorithm 6 Radix4 FFT(f, n)

Require: 複素数係数の多項式 f, f の項数 n

Ensure: f の FFT

if $n = 2$ **then**
  $F[0] \leftarrow f[0] + f[1] \cdot j$
  $F[1] \leftarrow f[0] - f[1] \cdot j$
else if $n = 4^m \cdot 2$ **then**
  for $0 \leq i < \dfrac{n}{2}$ **do**
    $F_{20}[i] \leftarrow f[2 \cdot i]$
    $F_{21}[i] \leftarrow f[2 \cdot i + 1]$
  end for
  Radix4 FFT$(F_{20}, \dfrac{n}{2})$
  Radix4 FFT$(F_{21}, \dfrac{n}{2})$
  for $0 \leq i < \dfrac{n}{2}$ **do**
    $F[2 \cdot i] \leftarrow F_{20}[i] + F_{21}[i] \cdot W$
    $F[2 \cdot i + 1] \leftarrow F_{20}[i] - F_{21}[i] \cdot W$
  end for
else if $n > 4$ **then**
  for $0 \leq i < \dfrac{n}{4}$ **do**
    $F_0[i] \leftarrow f[4 \cdot i]$
    $F_1[i] \leftarrow f[4 \cdot i + 2]$
    $F_2[i] \leftarrow f[4 \cdot i + 1]$
    $F_3[i] \leftarrow f[4 \cdot i + 3]$
  end for
  Radix4 FFT$(F_0, \dfrac{n}{4})$
  Radix4 FFT$(F_1, \dfrac{n}{4})$
  Radix4 FFT$(F_2, \dfrac{n}{4})$
  Radix4 FFT$(F_3, \dfrac{n}{4})$
  for $0 \leq i < \dfrac{n}{4}$ **do**
    $F[4 \cdot i] = F_0[i] + WF_1[i] + W(F_2[i] + WF_3[i])$
    $F[4 \cdot i + 1] = F_0[i] + WF_1[i] - W(F_2[i] + WF_3[i])$
    $F[4 \cdot i + 2] = F_0[i] - WF_1[i] + W(F_2[i] - WF_3[i])$
    $F[4 \cdot i + 3] = F_0[i] - WF_1[i] - W(F_2[i] - WF_3[i])$
  end for
end if
return F

6.5　計算量の比較

4 分割することにより，$O(\frac{3}{8}N\log N)$ まで計算量を減らすことができる．

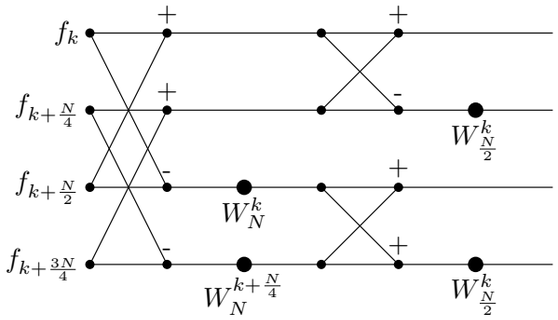

図 6.1　Radix2 FFT

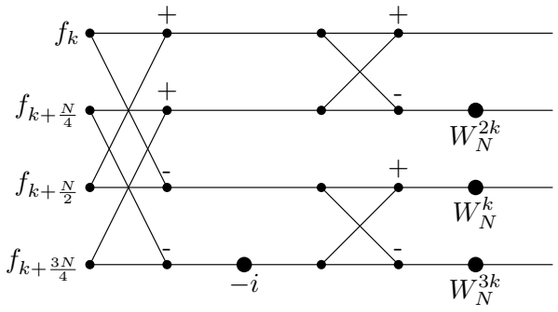

図 6.2　Radix4 FFT

上記はバタフライ演算図と呼ばれる図で，FFT のアルゴリズムを模式的に表したものである．図 6.1 は，Radix2 FFT のバタフライ演算を 2 回行った図，図 6.2 は Radix4 FFT のバタフライ演算を 1 回行った図である．Radix4 にすることにより，2 段目の乗算が 1 回増える代わりに，1 段目の乗算を削除することができる．ここで，1 段目に $-i$ がかけられているが，これは複素数の実部と虚部を入れ替えるだけなので，乗算を必要としない．そのため，図 6.1 では 4 回乗算が行われているが，図 6.2 では 3 回の乗算にとどまる．このような構造が全体に張り巡らされているため Radix4 FFT の計算量は Radix2 FFT の $\frac{3}{4}$ 倍に当たる $O(\frac{3}{8}N\log N)$ となる．Radix2 FFT と Radix4 FFT，それらの逆変換 Radix2 IFFT，Radix4 IFFT の比較をした結果を表 6.1 に示す．

アルゴリズム	乗算	加算	合計	再帰	Time (μs)
Radix2 FFT	2304	4096	6400	488	1.77
Radix4 FFT	1792	4096	5888	170	1.29
Radix2 IFFT	4096	4096	8192	488	1.87
Radix4 IFFT	3072	3840	6912	170	1.37

表 6.1　各アルゴリズムの比較

乗算回数と加算回数について，FFT，IFFT ともに Radix4 の方が少なく，計算時間も同様に Radix4 の方がより高速であることがわかる．演算量が減ったことにより，FFT，IFFT ともに 1.4 倍高速化することができた．ここで Radix2 FFT より Radix4 IFFT の方が演算回数が多いが，コード化した際，Radix4 IFFT は Radix2 FFT に比べて再帰呼び出しの回数が少ないため，表 6.1 のような結果になったと考えられる．

第 7 章

有限体上の多項式乗算

7.1 NTT

スクリプト実装では，有限体上の多項式乗算に**数論変換 NTT(Number Theoretic Transform)** を用いていた．NTT とは，FFT の理論を有限体上に限定したアルゴリズムである．

定義 7.1. 定義 6.1 と同様に，$f = \sum_{k=0}^{N-1} f_k x^k$ ($f_k \in \mathbb{Z}_q$, q は素数) の NTT $F(f) := F_k$ ($F_k \in \mathbb{Z}_q$) を以下のように定義する．

$$F_k = \sum_{j=0}^{N-1} f_j \alpha^{jk} \pmod{q} \qquad (\alpha^N \equiv 1 \pmod{q})$$

定義 7.2. NTT の逆変換 (Inverse NTT) を以下のように定義する．(以下 INTT と表記する．)

$$f_k = \frac{1}{N} \sum_{j=0}^{N-1} F_j \alpha_N^{-jk} \pmod{q} \qquad (\alpha^N \equiv 1 \pmod{q})$$

DFT と同様に，NTT を用いて有限体上の多項式乗算を計算することができる．

7.2 Radix2 NTT と Radix4 NTT

定義 7.1 の式に対して，式 (6.5), (6.6) と同様な変形を加えることで，計算量を減らすことができる．ここでは，基数 2 の NTT を Radix2 NTT と呼ぶ．スクリプト実装では，Radix2 NTT が用いられていた．本稿では，基数を 2 から 4 に増やし，有限体上の多項式乗算の高速化に取り組んだ．基数 4 の NTT を Radix4 NTT と呼ぶ．

7.2.1 計算量の評価

Radix2 NTT と Radix4 NTT の計算量及び実行時間を表 7.1 に示す．FFT と同様，乗算回数と加算回数は NTT, INTT ともに Radix4 NTT の方が少なく，計算時間も Radix4 の方がより高速であることがわかる．NTT は 1.3 倍，INTT は 2.3 倍高速化することができた．(表 7.1)

アルゴリズム	除算	乗算	加算	合計	再帰	Time (μs)
Radix2 NTT	4608	4608	4608	13824	488	2.36
Radix4 NTT	3328	3840	4608	11776	170	1.77
Radix2 INTT	4608	6656	4608	15872	488	3.94
Radix4 INTT	4480	5248	4608	14336	170	1.71

表 7.1　NTT の比較

第 8 章

実装結果

8.1 多項式乗算の比較

Karatsuba 法，Toom-4 法，Toom-8 法，Radix2 FFT を組み込んだ多項式乗算，Radix4 FFT を組み込んだ多項式乗算の計算量と時間を比較した．(表 8.1)

アルゴリズム	乗算	加算	合計	Time (μs)
Karatsuba	19683	230052	249735	504.213
Toom-4	60640	79984	140624	49.254
Toom-8	62792	79168	141960	**41.740**
Radix2 FFT	9216	12288	21504	53.565
Radix4 FFT	7168	12032	19200	**37.108**

表 8.1　多項式乗算の比較

スクリプト実装に使われていた多項式乗算は Karatsuba 法や Radix2 FFT によるアルゴリズムだった．本稿で取り組んだ実装 Toom-4 法は Karatsuba 法の 10 倍，Toom-8 法，Radix4 FFT は，いずれも Karatsuba 法の 12 倍以上高速化することができた．特に，Toom-8 法と Radix4 FFT はスクリプト実装の多項式乗算の中で最も高速だった Radix2 FFT を用いた多項式乗算よりも高速化することができた．その中でも，Radix4 FFT が 37μs と最も高速であった．

次に，有限体上の多項式乗算についての比較をする．7.1 節で記述した NTT に加えて，表 8.1 で高速な実装が実現した Toom-8 法，Radix4 FFT を組み込んだ多項式乗算の解を最後に mod q するよう実装した多項式乗算との比較を行う．(表 8.2)

アルゴリズム	除算	乗算	加算	合計	Time (μs)
Radix2 NTT	14336	16384	13824	44544	82.463
Radix4 NTT	11648	13440	13824	38912	47.821
Toom-8	3456	59848	79168	142472	47.682
Radix4 FFT	512	7168	12032	19712	**42.995**

表 8.2　有限体上の多項式乗算の比較

Radix4 NTT により，スクリプト実装に用いられてた Radix2 NTT を利用した多項式乗算よりも高速な計算ができたが，それ以上に，Radix4 FFT を利用した多項式乗算が高速であった．NTT を利用した多項式乗算の場合，int 型での実装になり，計算過程で int 型の有効桁を超えない様に mod q によって値を制限する必要があるため，計算コストがかかったと考えられる．

8.2 鍵生成，署名，検証の比較

多項式乗算におけるそれぞれの方式を ModFalcon に組み込み，鍵生成，署名の比較を行った結果を表 8.3 に示す．検証では，有限体上の多項式乗算がなされているため，表 8.4 で比較する．

アルゴリズム	鍵生成 (s)	署名 (s)
Karatsuba	1.335	3.256×10^{-3}
Toom-4	1.006	3.256×10^{-3}
Toom-8	1.002	3.256×10^{-3}
Radix2 FFT	1.044	3.256×10^{-3}
Radix4 FFT	**0.964**	$\mathbf{3.075 \times 10^{-3}}$

表 8.3　鍵生成，署名の比較

鍵生成について，Radix4 FFT を用いた多項式乗算を組み込んだ時が最も高速となり，Karatsuba 法を組み込んだ時より 1.4 倍高速化することができた．署名では，スクリプト実装において多項式乗算の処理はなかったが，その他の処理でフーリエ変換を利用していたため，Radix4 FFT を組み込んだ時に最も高速となった．

アルゴリズム	検証 (s)
Radix2 NTT	3.892×10^{-3}
Radix4 NTT	3.731×10^{-3}
Toom-8	$\mathbf{3.676 \times 10^{-3}}$
Radix4 FFT	$\mathbf{3.642 \times 10^{-3}}$

表 8.4　検証の比較

検証については，Toom-8 法，Radix4 FFT を用いた多項式乗算を組み込んだ時が最も高速となった．両方式を組み込んだとき，実行結果に大きな差は生まれなかったが，検証では有限体の多項式乗算の試行回数が少なかったためと考えられる．

第 9 章

まとめ

本稿では，鍵生成，署名，検証を行う上で一番コストの発生している多項式乗算に着目し，高速化検討を行った．高速化にあたって，Toom-4 法，Toom-8 法，Radix4 FFT を組み込んだ多項式乗算を実装した．Karatsuba 法に比べて，Toom-4 法，Toom-8 法では約 11 万回，Radix4 FFT の多項式乗算では約 23 万回演算回数を削減できた．その結果，511 次多項式の乗算においては Toom-8 法は 12 倍，Radix4 FFT の多項式乗算は 13 倍高速化することができ，鍵生成や署名，検証においても，既存のスクリプト実装よりも高速化することができた．特に，今回検討した方式の中では，Radix4 FFT を組み込んだ時が最も高速だった．

今後の展望として，Toom-Cook 法と FFT，NTT における最適な分割数の検討，本稿では扱っていない署名アルゴリズムにおける符号化関数などの高速化に着手し，ModFalcon の更なる高速化を目指す．

謝辞

本論文は筆者が東京都立大学大学院理学研究科数理科学専攻博士前期課程に在学中の研究成果をまとめたものである．学士 4 年より 3 年間，本研究を終始多大なご指導をしていただいた横山俊一准教授に，深く感謝の意を申し上げます．そしてご多忙の中，本論文の副査を快諾していただきました内山成憲教授と内田幸寛准教授に心より感謝いたします．また本研究を進めるにあたり，ご指導ならびに多数の資料提供をいただいた NTT 社会情報研究所齋藤恆和氏，山村和輝氏に感謝いたします．最後に，本研究をはじめとした様々な場面で協力していただいた髙橋雄人氏，その他友人や，家族，関わってくださった全ての方々にお礼申し上げます．

参考文献

[1] Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet,and Keita Xagawa, “ModFalcon: compact signature based on NTRU lattices”, ASIA CCS’20: The 15th ACM Asia Conference on Computer and Communications Security, pages 853–866, October 2020.

[2] Pierre Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest,Thomas Ricosset, Gregor Seiler, William Whyte and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU,pages 9-20,39,October 2020.

[3] Thomas Prest. ModFalcon. GitHub. March 2020. `https://github.com/tprest/ModFalcon`.

[4] Thomas Prest, Pierre Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`, (参照 2023/1/5).

[5] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan.“Trapdoors for hard lattices and new cryptographic constructions.” In Richard E. Ladner and Cynthia Dwork, editors, 40th ACM STOC, pages 197–206. ACM Press, May 2008.

[6] Thomas Prest, Thomas Ricosset, and Melissa Rossi. Simple, fast and constant-time gaussian sampling over the integers for Falcon. Second PQC Standardization Conference, 2019.

[7] Andrei Leonovich Toom, “The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers”, Sovidet Math–Translations of Dokl. Akad. Nauk SSSR 4,3 , Page 714-716, June 1963.

[8] Stephen Arthur Cook, “On the Minimum Computation Time of Functions”, PhD thesis, Harvard University Department of Mathematics, 1966.

[9] Stephen Arthur Cook, Stål Aanderaa, “On the Minimum Computation Time of Functions”, Transactions of the American Mathematical Society vol.142, Page 291-314, August 1969.

[10] Shui-Hung Hou and Wan-Kai Pang,“Inversion of confluent Vandermonde matrices”, Computers and Mathematics with Applications (3), Page 1539-1547, June 2002.

[11] James William Cooley and John Wilder Tukey,“An Algorithm for the Machine Calculation of Complex Fourier Series, Math. Comput.”, Vol.19, Page 297-301, 1965.

[12] Douglas Jones,“Digital Signal Processing: A User’s Guide”, Page 149-153, 2006.

[13] 紺谷拓弥，野呂正行，1999 年，多項式乗算の様々なアルゴリズムの比較，数理解析研究所講究録，1085 巻，140-150