公立はこだて未来大学 2017 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2017 System Information Science Practice
Group Report

プロジェクト名
FUN-ECM プロジェクト
Project Name
FUN-ECM Project

グループ名
A グループ
Group Name
A Group

プロジェクト番号/Project No.
11-A

プロジェクトリーダ/Project Leader
1015014　中島俊平　Shunpei Nakajima

グループリーダ/Group Leader
1015014　中島俊平　Shunpei Nakajima

グループメンバ/Group Member
1015014　中島俊平　Shunpei Nakajima
1015082　福永慧　Kei Fukunaga
1015112　小澤貴也　Takaya Ozawa
1015182　落合航平　Kouhei Ochiai
1015202　金子真澄　Masumi Kaneko
1015237　水上敬介　Keisuke Mizukami
1015260　外山拓　Taku Toyama

指導教員
白勢政明　由良文孝
Advisor
Masaaki Shirase　Fumitaka Yura

提出日
2018 年 1 月 19 日
Date of Submission
January 19, 2018

概要

私達のプロジェクトの目的は，より大きな桁数の素因数を見つけることである．素因数分解が重要であることの背景として，RSA 暗号がある．RSA 暗号は，40 年前に考案された初めての公開鍵暗号で，現在でもデジタル署名などで利用されている．RSA 暗号は，「大きい桁数の 2 つの素数からなる合成数を素因数分解することが難しい」ということで安全が保証されているが，近年，より高い安全性を持つ楕円曲線暗号が利用されてきている．そこで FUN-ECM プロジェクトでは，楕円曲線法（ECM）を用いて，より大きい桁数の素因数分解を行うプログラムを作成し，RSA 暗号の安全性について検証を行う．更に私たちは，大きい数の素因数分解をランキングしたサイトである，ECMNET[2] や STUDIO KAMADA[3] へのランクインを目標として掲げ活動を行った．

まず私達は，理論班とプログラム班に分かれ活動を行っている．理論班の目的は，ECM の高速化のための論文を理解し，プログラム班に引き継ぐことである．プログラム班の目的は，理論班が準備したアルゴリズムを実装し，高速化できているか確かめることである．

理論班は，ECM の Stage2 をより高速に行えるアルゴリズムの発見・理解を目標とした．前期は昨年度のプログラムに使われていた理論の理解，Baby-step Giant-step の理解，PARI/GP での実装を行った．後期は Montgomery 曲線の理解，素数ペアリングの理解，Pollard's ρ 法の理解，FFT の実装の検討を行った．

プログラム班は，前年度に作成された素因数分解プログラムをさらに高速化することを目標とした．前期は昨年度のプログラムで余分に座標変換を行っていた部分のコードの修正，並列処理の強制終了の機能の追加，Baby-step Giant-step の実装，Stage2 の実行速度のテストを行った．後期は，Montgomery 曲線を使用した ECM プログラムの作成した．また，素数テーブルの実装や Xeon に置ける階層化，シェルスクリプトなどを行った．また，ソースファイルの説明書の作成を作成した．

以上の 2 つの班が互いに足りない部分を補完しあい活動を行った．その結果，約 80 倍の Stage2 の高速化に成功し，素因数分解を高速に行えるプログラムが完成した．

キーワード　素因数分解，楕円曲線法，ECMNET，STUDIO KAMADA，エドワーズ曲線，拡張射影座標，RSA 暗号，Baby-step Giant-step，Pollard's ρ 法，FFT

（※文責: 中島俊平）

Abstract

The goal of our project team is to find prime factor as large as possible. Factorizations in prime numbers have become more important because of RSA cryptosystem. RSA cryptosystem which was invented 40 years ago is still being used for digital signatures. RSA cryptosystem has been seemed to be secure because it is difficult to prime factorize a composite number composed of two prime numbers with large number of digits. However, elliptic curve cryptography has been used in recent years because it's security is higher than RSA. Therefore, we create a program that performs prime factorization with a larger number of digits by the elliptic curve method (ECM). Then, we verify security of the RSA cryptosystem. For that purpose, we have carried out activity with the goal of ranking in ECMNET[2] and STUDIO KAMADA[3]. ECMNET and STUDIO KAMADA is a website ranked by large number of prime factorization.

First, we divided into two groups, that Theory group and Program group. The purpose of Theory group was to understand manuscripts for speeding up ECM, and to hand over it to Program group. The purpose of Program group is to implement algorithm which prepared by Theory group and to validate whether or not we can do increase in speed of program.

Theory group aimed for algorithmic discovery and understanding to be able to perform Stage2 of ECM with high-speed.We understood a theory used for a program of last year of project, we understood Baby-step Giant-step, and we implemented them in PARI/GP in first semester. We understood a theory Montgomery curve and Pollard's ρ method and we considered whether to implement FFT in second semester.
Program group aimed to speed up the ECM program created in last year. We modified the code of the part where the coordinate conversion was wasting last year program, implemented Baby-step Giant-step method, added the function of forcibly terminating parallel processing, and tested speed of Stage 2.Program group made ECM program using Montgomery Curve. In addition, we implemented prime table, layered in Xeon, and made shell script etc. In addition, we created a description of the source file.

These two groups complemented each other missing parts and carried out activities. As a result, we succeeded in speeding up Stage 2, and completed a program capable of fast prime factorization.

Keyword Elliptic Curve Method, prime factorization, ECMNET, STUDIO KAMADA, Twisted Edwards Curve, Extended Twisted Edwards Coordinates, RSA cryptosystem, Baby-step Giant-step method, Pollard's ρ method, FFT

（※文責: 中島俊平）

目次

第 1 章　背景

RSA 暗号は，大きい桁数の素因数分解が難しいことを安全性の根拠としている．そこで，楕円曲線法 (ECM) を利用し高速に素因数分解を行い，RSA 暗号の安全性を検証することが私たちの目的である．また，そのプログラムを利用して大きい桁数の素因数分解を行い，ECMNET や STUDIO KAMADA へランクインすることも目標として掲げた．

（※文責: 中島俊平）

1.1　本プロジェクトの背景

暗号技術は，情報の保護やコンピュータセキュリティにおいて欠かせない技術である．ファイルの暗号化の他に，HTTPS や，無線 LAN における通信など多くの場面で暗号技術が利用されている．しかし，暗号技術は常に進化する攻撃方法により解読の脅威に晒されている．様々な攻撃方法から安全な暗号アルゴリズムを作成するためには，作成する側が暗号解読の方法を知る必要がある．暗号の安全性評価は暗号解読の技術が利用されていて，暗号の強度は暗号解読に必要な情報量と計算量によって評価される．今回のプロジェクトでは，その暗号解読アルゴリズムの 1 つである，楕円曲線法 (ECM) を学ぶ．

現在，有名な公開鍵暗号の 1 つとして RSA 暗号がある．RSA 暗号は，桁数が大きい合成数の素因数分解が困難であることを安全性の根拠としている暗号である．RSA 暗号を解読する時は，合成数の元となる 2 つの素因数を見つけ出す必要がある．楕円曲線法では，与えられた曲線の点のスカラー倍が無限遠点になることによって，因数が発見される．この性質を利用することで，RSA 暗号が解読できる．

楕円曲線法には Stage1 と Stage2 があり，Stage1 で素因数分解できなかった場合，Stage2 で素因数分解を試みる．前年度のプロジェクトでは Stage1 のプログラムはほぼ完成していたため，今年度のプロジェクトでは Stage2 を完成させることを課題とした．また，ECMNET や STUDIO KAMADA という分解した素因数の大きさを競うサイトがあり，Stage2 を実装することで ECMNET や STUDIO KAMADA でのランクインをすることを目標として掲げた．

（※文責: 福永慧）

1.2　ECMNET とは

ECMNET とは，楕円曲線法を用いてカニンガム数を素因数分解し，見つかった素因数の大きさをランキング形式で競う Web サイトである．

ECMNET は，特定の範囲のカニンガム数の素因数分解を行うことで，Cunningham project に貢献することを主な活動としている．

カニンガム数，Cunningham project の詳細を以下に記す．

カニンガム数

以下の条件を満たす数をカニンガム数という．なお $ECMNET$ では，変数 b, n の範囲を限定している．

$$\text{カニンガム数:}\quad b^n \pm 1 \qquad s.t. \quad b, n \in \mathbb{N}, \quad b \text{ は累乗数でない}$$

Cunningham project

Cunningham project とは，b, n を表 1.1 の範囲に限定したカニンガム数を素因数分解するプロジェクトである．

b	2	3	5	6	7	10	11	12
n の上限	1300	850	550	500	450	400	350	350

表 1.1　Cunningham project における b，n の範囲

2018 年 1 月 19 日現在，ECMNET にランクインするためには，2017 年のトップ 10 で 65 桁，全期間のトップ 50 で 68 桁以上の素因数を発見する必要がある．

（※文責: 中島俊平）

1.3　STUDIO KAMADA とは

STUDIO KAMADA とは，レプディジットやニアレプディジットなど合成数の素因数分解を行い，結果を報告する個人 Web サイトである．合成数を素因数分解するための方法も多様であり，その中の 1 つに楕円曲線法がある．ECMNET 同様に，解読された素因数の大きさをランキング形式で競うことが出来る．STUDIO KAMADA で対象とされている合成数を以下で説明する．

レピュニット

repeated unit の略で，1，1111，11111111 のような 1 だけからなる自然数のこと．

レプディジット

repeated digit の略で，レピュニットを含む，すべての桁の数字が同じ（ぞろ目）自然数のこと．

ニアレプディジット

$near-repeated\,unit$ の略で，レプディジットから 1 桁だけ他の数字に置き換えた自然数のこと．

プラトウアンドデプレッション

レプディジットから，両端の数字を共通の異なる数字に置き換えた自然数のこと．

クワージレプディジット

レプディジットから，2 つの桁の数字を共通の異なる数字に置き換えた自然数のこと．

ニアレプディジット回文数

ニアレプディジットのうち，桁数が奇数で中央の数字だけが異なる自然数のこと．

（※文責: 福永慧）

1.4　課題の概要

本プロジェクトでは，楕円曲線法の Stage1 の部分については去年の時点でほぼ完成していたため，Stage2 を中心に作成することにした．Stage2 を完成させることで，大きな合成数を素因数分解することを目指す．そして，ECMNET や STUDIO KAMADA へのランクインを目指す．

（※文責: 福永慧）

第 2 章　到達目標

2.1　本プロジェクトにおける目的

RSA 暗号は桁数の大きい合成数の因数分解が難しいことを安全性の根拠としている．しかし，その合成数の因数分解が容易にできてしまうと RSA 暗号は簡単に解読できてしまう．簡単に解読されてしまうと RSA 暗号の安全性が保証されない．当プロジェクトでは，RSA 暗号の安全性を保証するために素因数分解を行うプログラムを作成した．プログラムを作成するために課題を設定した．

2.1.1　プログラムの性能の向上化

プログラムの性能を確認するために，ECMNET と STUDIO KAMADA に記載されている合成数などを用いて計算を行った．ECMNET や STUDIO KAMADA に記載されている合成数の素因数分解が成功した場合，当プロジェクトの名前が ECMNET や STUDIO KAMADA に記載される．合成数の素因数分解を行うプログラムの性能を向上させるために，既存のプログラムを改善し，単位時間ごとの計算速度を向上させる必要がある．そのため，この目的を達成するために以下のことを実施した．

（※文責: 水上敬介）

- 既存のプログラムのバグの修正
- プログラムの無駄な処理の除去
- 論文の読解
- アルゴリズムの検討・理解
- Stage2 においての新たなアルゴリズムの実装
- プログラム内で扱う曲線の変更
- プログラムの性能検証のための合成数の決定

（※文責: 水上敬介）

2.2　課題達成の為の班分け

前年度のプロジェクトでは前期で ECM についての学習を行い，後期でアルゴリズムの提案・実装を行っていた．しかし，このような日程でプロジェクトを進行していくと以下のような問題が発生した．

- 実際にプログラムを実装する期間が少ない
- 完成したプログラムを試行する期間が少ない
- 巨大な合成数の分解を行いにくい

本プロジェクトでは，5 月上旬まで全員で楕円曲線法についての基礎学習を行った．5 月中旬以降，既存のプログラムを改善するためにグループ全員でアルゴリズムの理解と実装をするのは効率が悪いと判断し，プログラム班，理論班の 2 つのグループに班分けをした．以下にそれぞれの班の課題を述べる．

理論班

楕円曲線法について理解を深め，前期では，計算速度の向上のためのアルゴリズム，Stage2 での新たなアルゴリズム (Baby-step Giant-step) の検討，理解に取り組んだ．後期では，モンゴメリー曲線を用いた計算，ρ 法，高速フーリエ変換，素数ペアリングの検討，理解に取り組んだ．逐次，理解したアルゴリズムをプログラム班に説明した．

プログラム班

前期では，既存のプログラムの内容を理解し，プログラムの動作内容の理解に取り組んだ．理論班から提示されたアルゴリズム (Baby-step Giant-step) をもとに，既存のプログラムを基に実装を行った．後期では，モンゴメリー曲線，素数ペアリングの実装を行った．

（※文責: 水上敬介）

第 3 章　活動内容

プロジェクトが発足した当初，楕円曲線についての基礎知識がなかったため，去年のプロジェクトでも基礎知識を身につけるために使われた資料 [1] を用いて，理解した．理解できないところはプロジェクトリーダーが主体となって解説をしてもらい楕円曲線についての基礎知識を学んだ．その後，アルゴリズムの提案をする理論班，提案されたアルゴリズムをプログラムに実装するプログラム班に分けてプロジェクトを進行した．

（※文責: 水上敬介）

3.1　基礎学習

去年のプログラムを理解するために 5 月の中頃までメンバー全員が楕円曲線法のアルゴリズムや基礎知識についての学習を行った．具体的な内容は以下の通りである．

有限体

素数 p に対し，0 から $p-1$ までの整数の集合 $\mathbb{F}_p = \{0, 1, \cdots, p-1\}$ を有限体と言う．$\mathbb{F}_p$ では四則演算が可能であり，楕円曲線法ではこの範囲で考える．

Euclid の互除法

自然数 a, $b(a \geq b)$ に対して以下の操作を繰り返し行い，余りが 0 になるまで行うことによって a, b の最大公約数を求めるものである．

Algorithm 1 Euclidean Algorithm

Require: $a, b \in \mathbb{N}, \quad a, b \neq 0, \quad a \geq b$

Ensure: $\gcd(a, b)$

```
while b ≠ 0 do
    q ← a/b
    r ← a mod b
    a ← b
    b ← r
end while
```

以後，a, b の最大公約数を $\gcd(a, b)$ と表記することとする．

拡張 Euclid の互除法

与えられた整数 a, b, c に対し，未知数 x, y に関する一次方程式 $ax + by = c$ の整数解は，1 組存在すれば無数に存在する．この方程式を一次不定方程式という．一次不定方程式の解を求めるには，拡張 Euclid の互除法が有効である．拡張 $Euclid$ の互除法は，自然数 a，b に関する一次不定方程式 $ax + by = gcd(a, b)$ を満たす無数の整数 x, y を効率よく求めることができるというものである．例として $174x + 69y = 3$ の整数解を求める．まず Euclid

の互除法を用いて 174 と 69 の最大公約数を求める．

$$\begin{aligned} 174/69 &= 2 * 69 + 36 \\ 69/36 &= 1 * 36 + 33 \\ 36/33 &= 1 * 33 + 3 \\ 33/3 &= 11 \end{aligned}$$

となる．そしてこれらは

$$\begin{aligned} 3 &= 36 - 33 * 1 \\ 33 &= 69 - 36 * 1 \\ 36 &= 174 - 69 * 2 \end{aligned}$$

と表せるので

$$\begin{aligned} 3 &= 36 - 33 * 1 \\ &= 36 - (69 - 36 * 1) * 1 \\ &= 36 * 2 + 69 * (-1) \\ &= (174 - 69 * 2) * 2 + 69 * (-1) \\ &= 174 * 2 + 69 * (-5) \end{aligned}$$

以上より，$174x + 69y = 3$ の整数解は $(x, y) = (2, -5)$ と求めることができる．有限体 $\mathbb{F}_p$ において除算 $a \div b$ を計算する場合，p と b は互いに素なので，拡張 Euclid の互除法により不定方程式 $px + by = 1$ の解 (x_0, y_0) を求めることができる．このとき $px_0 + by_0 = 1$ となるので，有限体 $\mathbb{F}_p$ 上では $by = 1$，つまり $b^{-1} = y_0$ が成立する．よって $a \div b = a \times b^{-1} = a \times y_0$ と変形することで，除算を乗算に置き換えて計算できる．

楕円曲線の定義方程式

$a, b \in \mathbb{F}_p$に対して $y^2 = x^3 + ax + b$ で定義される曲線を有限体 $\mathbb{F}_p$上の楕円曲線という．

楕円曲線の加算と逆元

(加算) 楕円曲線上のある 2 点 P, Q を通る直線をとすると，楕円曲線と直線 ℓの 3 つ目の交点 R' $(= P \times Q)$ の x 軸に関する対称点を R とする．これで得られた点 R を R'の逆元と呼び，$R = -R'$が成り立つ．また，R を P と Q を加算した点と定義し，$R = P + Q$ が成り立つ．

無限遠点

楕円曲線上の点 P とその逆元 $-P$ をとり，$P + (-P)$ を考える．そうすると，2 点を通る直線と楕円曲線には P と $-P$ の他には交点が存在しない．このような状態のときに，存在しない点を仮想的に考え，それを無限遠点と呼び，$P + (-P) = O$ が成り立つ．

楕円曲線の２倍算

楕円曲線上の点 P の接線を ℓとし，楕円曲線と直線 ℓの P 以外の交点を R'とし，R'の逆元を R とする．この R は $R = P + P = 2P$ であり．楕円曲線の 2 倍算と定義する．

楕円曲線の scalar 倍算

点 P と自然数 d に対して, 点 P を d 倍 ($dP = P + P + P + P + \cdots + P$ (d 個の和)) することを，楕円曲線の *scalar* 倍算という．

以上のことを基礎学習として学んだ．以下の章では，２つの班のそれぞれの活動内容を記述する．

（※文責: 落合航平）

3.2 理論班

理論班では，楕円曲線法の，特に Stage2 を高速化するために論文を探し理論を理解した．そして，プログラム班にアルゴリズムの実装を引き継いだ．前期では，Baby-Step Giant-Step や Montgomery 曲線の理論をプログラム班に提案した．後期では，Montgomery 曲線の詳細な実装，Baby-step Giant-step の発展形である素数ペアリング，Brent のアルゴリズムをプログラム班に提案した．その他にも，Pollard's ρ 法や FFT についても検討を行った．

（※文責: 落合航平）

3.2.1 楕円曲線法とは

理論班では，楕円曲線法について，[5] を参考にし，理解をした．楕円曲線法は，1985 年 Lenstra によって提案された中規模合成数の因数分解にとってよいとされているアルゴリズムである．楕円曲線は曲線のパラメータを変えることで位数がランダムに変わるという性質を持っている． そのため群の位数と合成数の相性が悪ければ，パラメータを変更し曲線を変更すればよいため，効率的に因数を見つけることができる．楕円曲線法は大きく Stage1，Stage2 に分けられる．

楕円曲線法の Stage1

本節では，合成数 N の因数 p を見つけることを考える．合成数 N の桁数から，因数の大きさを予想して，パラメータ B_1，B_2 を定める．合成数 N と B_1，B_2 の対応表を表 3.1 に記す．

digits	20	25	30	35	40	45	50
optimal B_1	11,000	50,000	250,000	1,000,000	3,000,000	11,000,000	43,000,000
optimal B_2	1,100,000	5,000,000	25,000,000	100,000,000	300,000,000	1,100,000,000	4,300,000,000

表 3.1　素因数の桁数に対応する適切な B_1，B_2 の値

ある楕円曲線上の射影座標で与えられた点 $P = (X_P : Y_P : Z_P)$ を利用し，以下の手順で，因数を見つけることができる．

Step1 楕円曲線を決定し，点 P を選択する．
Step2 B_1 以下の自然数全てにおける最小公倍数 $l = lcm\,(2, 3, ..., B_1)$ を求める．
Step3 点 $Q = l \times P = (X_Q : Y_Q : Z_Q)$ を計算する．
Step4 Z_Q と N の最大公約数 $d = \gcd(Z_Q, N)$ を計算する．
(1) d が 1 よりも大きいならば，プログラムを終了する．
(2) d が 1 ならば，Stage2 に移行する．

楕円曲線法の Stage2

Stage1 のみを利用していると，素因数を見つけやすくするには B_1 を大きくし，非常に大きい数 l に対して，scalar 倍 $l \times P$ を計算しなければならない．scalar 倍の計算は，非常に計算量が大きくなってしまうため，Stage2 を用いる．

楕円曲線法の Stage1 で因数を見つけることができなかったとき，mod$p$ での位数は，少なくとも B_1 よりも大きい．そのため，楕円曲線法の Stage2 では，B_1 から B_2 までの素数 s の中に，位数があるかを調べる．

$B_1 < s < B_2$ を満たす全ての素数 $s_1, s_2, s_3, \ldots, s_k$ について Stage1 の結果得られた点 Q に対する scalar 倍をそれぞれ求める．

$$
\begin{aligned}
s_1 \times Q &= Q_1 = (x_1, y_1) \\
s_2 \times Q &= Q_2 = (x_2, y_2) \\
s_3 \times Q &= Q_3 = (x_3, y_3) \\
&\cdots \\
s_k \times Q &= Q_k = (x_k, y_k)
\end{aligned}
$$

$x_1, x_2, x_3, \ldots, x_k$ を求め，合成数 N との最大公約数を求める．$Q_1, Q_2, Q_3, \ldots, Q_k$ のうちある点 Q_i で $Q_i = O \pmod p$ を満たせば素因数が求まり，成功となる．

（※文責: 金子真澄）

3.2.2 Baby-step Giant-step

去年までのプログラムでは楕円曲線法の Stage2 の計算に，総当たり法を用いていた．この方法では，k 回の scalar 倍が必要となるため高速ではない．そこで今年は総当たり法に代わり Baby-step Giant-step を実装した．Baby-step Giant-step とは，$B_1 < s < B_2$ を満たす全ての素数 $s_1, s_2, s_3, \ldots, s_k$ に対して，s を変形して計算を行うことで高速化を図る手法である．

Stage2 は $s \times P \equiv O$ のとき成功する．この式を変形していく．まず s を

$$
s = av + u \quad \left(-\frac{a}{2} < u < \frac{a}{2}\right) \tag{3.1}
$$

と変形する．すると

$$
\begin{aligned}
s \times Q &\equiv O \\
(av + u) \times Q &\equiv O \\
(av) \times Q &= -u \times Q
\end{aligned} \tag{3.2}
$$

と言い換えられる．この式を満たすかを確認すればよい．($u \times Q$ は $-\frac{a}{2} < u < \frac{a}{2}$ であるから事前に計算しておける)

ここで逆元の関係にある点の x 座標は等しい．つまり $(av) \times Q$ の x 座標を G_x，$-u \times Q$ の x 座標を H_x とすると

$$
\begin{aligned}
G_x &\equiv H_x \pmod p \\
G_x - H_x &\equiv 0 \pmod p \leftarrow G_x - H_x \text{ が } p \text{ の倍数}
\end{aligned}
$$

が成り立つ．

これまでの説明をまとめると，$G_x - H_x \equiv 0 \pmod{p}$ となる s が存在するならば，$(av) \times P = -u \times P$ となる s が存在する．すなわち，$s \times P = O$ となる s が存在するため，Stage2 が成功，素因数分解が完了となる．

ここで，全ての s について $G_{x_k} - H_{x_k}$ を計算しすべてを掛け合わせた数を d をすると，$G_x - H_x \equiv 0 \pmod{p}$ となる s が存在すれば，$d \equiv 0 \pmod{p}$ となる．したがって d と合成数 N の最大公約数を求めることで素因数 p が判明する．Baby-step Giant-step の手順は，以下のようになる．

Step1 $B_1 < s < B_2$ を満たすすべての素数 s について計算する．
(1) 素数 s について v, u を求める
(2) (3.2) 式の左辺の x 座標 G_x，(3.2) 式の右辺の x 座標 H_x を求める
Step2 $d = (G_{x_1} - H_{x_1})(G_{x_2} - H_{x_2}) \ldots (G_{x_k} - H_{x_k})$ を計算する
Step3 d と合成数 N の最大公約数をとる．
Step4 最大公約数が 1，N 以外ならその数は求めたい素因数であり，成功する．

Algorithm 2 Baby-step Giant-step

Require: E: 楕円曲線，Q_0: 楕円曲線法の Stage1 で得られた点，B_1，B_2
Ensure: p: 因数

```
for each i = 1 to a − 1 do
  H[i] ← i × Q_0
end for
Q ← a × Q_0
d ← 1
for each prime s = B_1 to B_2 do
  u ← s/a
  v ← s%a
  G ← v × Q
  d ← d × (G_x − H[i]_x)
end for
q ← gcd(d, N)
if q > 1 then
  return q
else
  return FAIL
end if
```

（※文責: 金子真澄）

3.2.3 素数ペアリング

Baby-step Giant-step をより高速化する手法として，素数ペアリングがある．楕円曲線の定義から，点 H と逆元の関係にある点 $-H$ は x 座標が等しいため，$G_x - H_x$ を計算すると，$G_x + H_x$

も同時に計算していることになる．そのため，$vw \pm u$ の両方が素数になるとき，結果的に計算量が少なくなる．以下，詳しい理論である．

$$(av) \times P = -u \times P \tag{3.3}$$

先ほどの (3.3) 式の計算を掘り下げて考える．逆元の定義より，$-u * P$ と $u * P$ は，x 軸に対して対称であり，x 座標の値が等しい．

つまり，$(av) \times P = -u \times P$ と $(av) \times P = u \times P$ ではその後の $(G_x - H_x)$ の値は同じになる．

もし，$av + u$，$av - u$ がともに素数だったとき，従来の計算手法だと，同じ計算を 2 重に行うことになり効率が悪い．そこで，$av + u$ と $av - u$ がともに素数になるペアを計算し，s がどちらかの時のみ $G_x - H_x$ を求めることにより，Step1 の計算回数が減るため高速化できる．

積極的なペアリング

上記のペアリングの発展形として，より積極的なペアリングを考えることができる．

$$\begin{aligned} s &= av + u & (av) \times P &= -u \times P \\ &= a(v-1) + (a+u) & (a(v-1)) \times P &= -(u+a) \times P \\ &= a(v-2) + (2a+u) & (a(v-2)) \times P &= -(u+2a) \times P \\ &= a(v-3) + (3a+u) & (a(v-3)) \times P &= -(u+3a) \times P \\ &\dots \end{aligned} \tag{3.4}$$

式 (3.3) は式 (3.4) のように変形できる．もし，$s = av + u$ は素数だったが，$av - u$ が素数でないためにペアリングできなかったときに，$s = av + u = a(v-1) + (a+u) = a(v-2) + (2a+u) = a(v-3) + (3a+u) = \dots$ と考えることで，より深くペアを探すことができる．ペアの数が増え，手順の回数がより減るため，高速化が期待できる．一方で，(3.3) 式の右辺が増える分，多くの事前計算が必要となることを考慮するべきである．

（※文責: 落合航平）

3.2.4 Montgomery 曲線

私たちのプロジェクトでは，楕円曲線として Twisted Edwards 曲線を利用していたが，Montgomery 曲線 [6] を導入するため，論文を探した．

Montgomery 曲線の特徴として，y 座標を用いずに加算，2 倍算ができるということがある．そのため，y 座標を計算せず，計算時間を減らすことができる．Weierstrass 曲線，Twisted Edwards 曲線では $(x, y) = (X/Z, Y/Z)$ という射影座標系を用いていたが，Montgomery 曲線では，$x = (X/Z)$ 座標のみで計算することができる．以下は Montgomery 曲線について理解したことである．

Montgomery 曲線の定義

Montgomery 曲線は，パラメータ a, b を用いて，

$$E_p(a, b) : by^2 = x^3 + ax^2 + x \quad \text{但し } b(a^2 - 4) \neq 0 \tag{3.5}$$

で定められる．

Weierstrass 曲線と同様に，射影座標系を導入することを考える．このとき方程式 (3.5) に，$x = X/Z$, $y = Y/Z$ を代入すると

$$\text{方程式: } bY^2Z = X^3 + aX^2Z + XZ^2 \tag{3.6}$$

Algorithm 3 PAIR

Require: $B_1, B_2 \in \mathbb{N}$

Ensure: $\{(u, v) \in \mathbb{N}^2 \mid s = av \pm u$ are prime numbers between B_1 and $B_2\}$

```
a_min ← ⌊(B_1 + w)/2w⌋
set Q_q to empty state for each q satisfying |q| < w
while more primes between B_1 and B_2 do
  get next prime s, where B_1 < s < B_2
  a ← ⌊(s + w)/2w⌋
  while a > a_min + L do
    a_min ← a_min + L − ⌈u_max/w⌉
    for all q satisfying |q| < w do
      for all a'Q_q with a' < a_min do
        remove a' from Q_q
        output the pair (2a', |q|)
      end for
    end for
  end while
  q ← s − 2aw {so s = 2aw + q}
  {look for mate in Q_{−q} or save in Q_q}
  repeat
    if Q_{−q} is nonempty then
      remove the front element a' of Q_{−q}
      {we have found two primes 2aw + q and 2a'w − q}
      u ← w(a − a') + q
      if u > u_max then
        output the pair (2a', |q|)
      else
        output the pair (a + a', u)
      end if
    else
      insert a at the rear of Q_q
      u ← 0 {force exit from loop}
    end if
  until u < u_max
end while{end of loop over primes}
for all q satisfying |q| < w do
  while Q_q is nonempty do
    remove front element a' of Q_q
    output the pair (2a', |q|)
  end while
end for
```

が得られる．

Montgomery 曲線上の点の加算，二倍算

Montgomery 曲線上の点 $P = (X :: Z)$ に対して，自然数 m, n を用いて，$P_m = mP = (X_m :: Z_m)$, $P_n = nP = (X_n :: Z_n)$，$P_{m-n} = P_m - P_n = (X_{m-n} :: Z_{m-n})$ とする．このとき Montgomery 曲線上の点における加算，二倍算は次の計算でできる．

$$
\begin{aligned}
&\text{加算}: P_{m+n} = P_m + P_n = (X_{m+n} :: Z_{m+n}) \\
&X_{m+n} = Z_{m-n}[(X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n)]^2 \\
&Z_{m+n} = X_{m-n}[(X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n)]^2
\end{aligned}
$$

$$
\begin{aligned}
&\text{二倍算}: P_{2n} = P_n + P_n = (X_{2n} :: Z_{2n}) \\
&4X_nZ_n = (X_n + Z_n)^2 - (X_n - Z_n)^2 \\
&X_{2n} = (X_n + Z_n)^2(X_n - Z_n)^2 \\
&Z_{2n} = (4X_nZ_n)[(X_n - Z_n)^2 + (\frac{A+2}{4})(4X_nZ_n)]
\end{aligned}
$$

加算，二倍算の式を見ると，確かに y 座標を用いずに計算できることがわかる．P_m と P_n の加算をするとき，P_{m-n} を用いることは，不便である．

Montgomery ladder

P_m と P_n の加算をするとき，P_{m-n} を用いることは，不便に思うかもしれないが，楕円曲線法においては，重大な問題とならず，Montgomery ladder と呼ばれる方法を用いると，点 P_{m-n} を変化させずに計算することができる．

Algorithm 4 は，Montgomery 曲線上の点 P，ある 2 進数 d に対して，scalar 倍 $d \times P$ を計算する Montgomery ladder である．但し，add_point(R_0, R_1) は点の加算，double_point(R_0)，double_point(R_1) は点の二倍算を意味している．

Algorithm 4 Calculate dP with Montgomery ladder

Require: Point $P = (X :: Z)$ on a Montgomery curve, the binary representation of $d = \sum_{i=0}^{m} k_i 2^i$

Ensure: The point $k \times P = (X_k :: Z_k)$

```
R_0 ← (0 :: 0)
R_1 ← P
for (i = m; i ≥ 0; i − −) do
  if k_i = 0 then
    R_1 ← add_point(R_0, R_1)
    R_0 ← double_point(R_0)
  else
    R_0 ← add_point(R_0, R_1)
    R_1 ← double_point(R_1)
  end if
end for
return R_0
```

アルゴリズムにおける R_0, R_1 は，常に $R_0 = kP$, $R_1 = (k+1)P$, $k \in \mathbb{N}$ が成り立つため，R_0，R_1 の差は P となり，効率的に計算することができる．

（※文責: 金子真澄）

3.2.5 Pollard's ρ 法

楕円曲線法のほかにも合成数から因数を見つけるアルゴリズムが研究されてきた．理論班では，因数分解する方法として，ρ 法に関して学習した．

ランダムウォーク関数 f を用いて，$x_{k+1} = f(x_k)$, $y_k = x_{2k}$ と定める．例えばランダムウォーク関数として，$f(x) = x^2 + 1 \pmod N$ を用いる．

Step1 x_i, y_i を計算する．
Step2 $\alpha = \prod_{i=1}^{k} |x_i - y_i|$ を計算する．
Step3 $\gcd(\alpha, N)$ を計算する．

理論班では，ρ 法を実現できるか論文を探していたが，楕円曲線を用いた方法を見つけることができず，楕円曲線法を用いた方法と比べて効率的に計算することができないと考えた．ρ 法は運が悪ければ，因数を見つける時間がいくらでも長くなる可能性がある．これら 2 点の理由から，ρ 法の実装をしなかった．

（※文責: 落合航平）

3.2.6 Brent のアルゴリズム

Brent は，Birthday Paradox に基づく方法を提案している．楕円曲線法の Stage2 を解くアルゴリズムとして提案された．Brent のアルゴリズム [7] は，次の手順で行う．

Step0 ランダムウォーク関数 f を用意する．
楕円曲線法の Stage1 で得られた点 P に対して，$Q_1 = k \times P$, $Q_2 = l \times P$ を計算する．

$$f(P) = \begin{cases} P + Q_1 & (r \bmod k \equiv 0) \\ P + Q_2 & (r \bmod k \equiv 1) \\ \cdots & \\ P + Q_k & (r \bmod k \equiv k) \end{cases}$$

Step1 $\alpha = 1$, $P_1 = P = (X_1 : Y_1 : Z_1)$ を設定する．
Step2 $P_2 = f(P_1), P_3 = f(P_2), \ldots, P_k = f(P_{k-1})$ を計算する．
Step3 $\alpha = \prod_{i=1}^{k} \prod_{j=1}^{i} (X_i Z_j - Z_i X_j)$ を計算する．
Step4 $d = \gcd(\alpha, N)$ を計算する．

Stage1 で得られた点 Q，$B_1 < q < B_2$ となる因数 q があるとすると，$qQ \equiv O$ が成り立つ．

前述の Step3 において α を求める計算は，莫大な時間がかかる．この演算を行うのに，FFT(fast Fourier transform:高速フーリエ変換)[8] と，IFFT（inverse fast Fourier transform:逆高速フーリエ変換）を利用することで計算量を減らすことができると言われている．そのため，理論班では，FFT についての論文を読み，理解に努めた．しかし，FFT についての前提知識が十分でないため，論文を読むことが困難であったことに加えて，GMP を利用した FFT の実装は難易度が高く，最終発表に間に合わないと判断したため実装には至らなかった．

離散フーリエ変換（離散フーリエ変換の式）

$$F(t) = \sum_{x=0}^{N-1} f(x) e^{-i\frac{2\pi tx}{N}}$$

高速フーリエ変換（高速フーリエ変換の式）

$$F(t) = \sum_{x=0}^{N-1} f(x) e^{-i\frac{2\pi tx}{N}}$$

（※文責: 落合航平）

3.3 プログラム班

プログラム班では，昨年度の FUN-ECM プロジェクトで作成したプログラムを使用し，単位時間あたりの計算量を増幅させるために，プログラムの改善を行った．高速化するために，Stage2 での Baby-step Giant-step の実装，Montgomery 曲線の実装，Stage2 における素数テーブルや素数ペアリングでの計算，去年のソースコードの不具合の修正を行った．今後のプロジェクトのことを考え Xeon におけるソースコードの階層化，ソースコードのドキュメントの作成を行った．作成したプログラムをテストするために VPS をレンタルした．また，昨年度のプログラムと今年度のプログラムの速度を比較するために，計測方法を確立させた．具体的には以下の通りである．

（※文責: 水上敬介）

3.3.1 座標変換の際の冗長なコストの削減

前年度のプロジェクトで作成された楕円曲線法プログラムでは，scalar 倍をする際の座標を射影座標から拡張射影座標に変換していた．scalar 倍を行う処理は楕円曲線法プログラムを実行する際に何度も使われるので，処理速度に影響する．そのため，最初から拡張射影座標を用意することによって座標変換する分のコストを減らすことが出来た．射影座標から拡張射影座標に変換する際に使われたアルゴリズムを Algorithm 5 に示す．

Algorithm 5 Projective Coordinates to Extended Coordinates

Require: (PX,PY,PZ) is Projective, (EX,EY,ET,EZ) is Extended, $N \geq 2$

Ensure: (EX,EY,ET,EZ)

$EX \leftarrow PX \times PZ$
$EX \leftarrow EX \bmod N$
$EY \leftarrow PY \times PZ$
$EY \leftarrow EY \bmod N$
$ET \leftarrow PX \times PY$
$ET \leftarrow ET \bmod N$
$EZ \leftarrow PZ \times PZ$
$EZ \leftarrow EZ \bmod N$

Algorithm 5 では，乗算を 4 回と mod の計算を 4 回行っている．プログラム班では，scalar 倍を行う前に行われていた Algorithm 5 を省略することによって，計算コストを削減することが出来た．

（※文責: 福永慧）

3.3.2 ソースコードの改善

scalar 関数における引数の修正

前年度のプロジェクトでは scalar 倍を行うときに座標Ｐのみを用意して，座標Ｐを scalar 倍したものを座標Ｐに代入していた．そのため，scalar 倍の処理を複数回行った場合，scalar 倍された座標に scalar 倍を行っていたことが分かった．しかし，scalar 倍を複数回行った場合でも，1 回分の scalar 倍の座標が必要であった．この問題は scalar 倍が行われる座標Ｐの他に，その結果を代入するための座標を用意することで解決した．以上を簡単に計算式で表す．
前期の scalar 倍算

$$P = kP \quad (k = 1, 2, ..., m) \tag{3.7}$$

後期の scalar 倍算

$$Q = kP \quad (k = 1, 2, ..., m) \tag{3.8}$$

Extended dedicated add 関数の修正

前年度のプロジェクトで作成された楕円曲線法プログラムでは，拡張射影座標を用いて加算をしていた．ECM プログラムで加算を行う関数の名前を Extended dedicated add と名付けていた．しかし，関数内の記述に書き損じがあり，正しく加算が出来ていなかった．今年度のプロジェクトではその記述を発見し，修正した．

main 関数における並列処理部分の変更

楕円曲線法プログラムでは，OpenMP を導入して並列処理が行なえるようにしていた．並列処理は main 関数内にある for 文に用いられていた．

今年度のプロジェクトでは main 関数内にある for 文の冗長性を発見し，ソースコードの修正を行った．前年度のプロジェクトで記述されていた処理を Listing 3.1 に記す．

Listing 3.1 前年度の並列処理

```
1  int found=0;
2  for(i=0;i<number_of_elliptic_curves;i++) {
3          mpz_t factor;
4          mpz_init(factor);
5          if(found == 0){
6                  ecm(factor, N, X, Y, d, B1, B2, fp, window_size );
7                  if(mpz_cmp_ui(factor, 1) != 0 && mpz_cmp_ui(factor, N) != 0)
8                          found = 1;
9          }
10 }
```

Listing 3.1 の概要を以下に記す．

- 関数 ecm は与えられた合成数Nの素因数分解を行う関数である．関数 ecm では，素因数分解された素数を引数 factor へ返す．
- 関数 ecm で素因数分解された素数が 1 かつ合成数 N でない場合，found に 1 を代入する．
- found が 1 の時，for 文の処理が終わるまで ecm 関数を計算しない．

今年度のプロジェクトでは，このソースコードの改善点を 2 つ見つけた．1 つ目に引数 found が 1 でも，ループから抜け出せていない点があった．2 つ目にこの for 文は並列処理で行われているので，他のスレッドの処理が終わるまでプログラムの処理が終わらないという点があった．今年度のプロジェクトではこれらの 2 点を改善するために，Listing 3.1 の 9 行目と 10 行目の間に exit 関数を用いた．また，found が 1 の時のみ exit 関数を呼び出すように記述した．

（※文責: 福永慧）

3.3.3 Montgomery 曲線による楕円曲線法の実装

後期のプロジェクトでは Montgomery 曲線を用いて，stage1 と stage2 の実装を行った. 計算公式は，Montgomery Ladder Algorithm というアルゴリズムを用いて，scalar 倍算を行った. このアルゴリズムはどの曲線にも適用することが可能であり，アフィン座標や射影座標の両方で実行することが可能であるが，特に Montgomery 曲線を射影座標で表現している場合に効率的だと言われている. また，射影座標で表現されている場合アルゴリズムの中間計算は X 及び Z 座標のみを使用して実行することができ，Y 座標を使用することがない. ただし，計算された後の座標の Y 座標は必要な場合 X 座標と Z 座標から取得することが可能である. 今回は，stage1 と stage2 の両方で Montgomery 曲線を用いているため，Y 座標を取得する必要がなかったので，使うことがなかった.

また，Montgomery 曲線は $by^2z = x^3 + ax^2z + xz^2$（ただし，$b(a^2 - 4)$ が 0 以外）の式から $x = 1$，$z = 2$ とし，a，y はランダムで取り，b はそれぞれの取った値を代入した式から計算して求めた.

次に，作成された Montgomery 曲線を使用した楕円曲線法プログラムの stage1 のソースコードを Listing3.2 に記す.

Listing 3.2 Montgomery_ecm

```
1    while (p <= B1) {
2          int found =0;
3        e = (int) (log (B1) / log (p));
4        for (i = 1; i <= e; i++){
5            montgomery_scalar (P, P, p, montgomery_a, N);
6            mpz_gcd (f, P->Z, N);
7
8            if (mpz_cmp_ui (f, 1) != 0 && mpz_cmp (f, N) != 0) {
9                mpz_set_ui (f, 1);
10               goto FACTOR_FOUND;
11             }
12         }
13       mpz_nextprime (prime, prime);
14       p = mpz_get_ui (prime);
15     }
```

計算の概要を以下に記す.

- 関数 Montgomery_ecm は与えられた合成数Nの素因数分解を行う関数である．関数 ecm では，素因数分解された素数を引数 factor へ返す．
- 任意の Montgomery 曲線に対して，B1 以下の素数 p で関数 montgomery_scalar で計算を行う．
- 関数 Montgomery_ecm で素因数分解された素数が 1 かつ合成数 N でない場合，found に 1 を代入する．
- 素因数分解されなかった場合または，関数 Montgomery_ecm で素因数分解された素数が 1 かつ合成数 N だった場合，mpz_nextprime により p の次の素数を p に代入する．
- P は，任意に定められたモンゴメリ曲線上の一点の x，y，z の値を格納している．

また，新たに関数 montgomery_scalar を作成した．関数 montgomery_scalar のアルゴリズムは Montgomery ladder(詳しくは Algorithm2 を参照) を参考に作成された．関数 montgomery_scalar のプログラムは Listing3.3 に記す．

Listing 3.3　montgomery_scalar

```
1     set(R0, P);
2     set(TP, P);
3     montgomery_double (R1, R0, a, N);
4       while (k > 0)
5       {
6         bit[i] = k % 2;
7         k >>= 1;
8         i++;
9       }
10    for (i = m - 2; i >= 0; i--)
11      {
12        if (bit[i] == 1)
13          {
14            montgomery_add (R0, R1, R0, TP, N);
15            montgomery_double (R1, R1, a, N);
16          }
17        else
18          {
19            montgomery_add (R1, R0, R1, TP, N);
20            montgomery_double (R0, R0, a, N);
21          }
22      }
```

計算の概要を以下に記す．

- k は scalar 倍算の係数を引数に取っており，bit 配列に k の二進数を格納している．
- set は第 1 引数の座標に第 2 引数の座標を代入する関数である．
- montgomery_add 関数は加算の計算をしており，第 2 引数と第 3 引数の加算を第 1 引数に代入している．第 4 引数は加算される座標同士の距離であり，第 5 引数は合成数である．
- montgomery_double 関数は 2 倍算の計算をしており，第 2 引数を 2 倍算して第 1 引数に代入している．第 3 引数は Montgomery 曲線における係数 a であり，第 4 引数は合成数である．
- R0 の初期値は，任意に定められた Montgomery 曲線上の一点の x，y，z の値を格納している．

- R1 の初期値は，R0 の 2 倍算の座標である．
- a は，任意に定められた Montgomery 曲線の係数 a を表している．
- montgomery_scalar 上では，R0 と R1 の座標の距離が常に TP で成り立つため，montgomery_add 関数において R0 と R1 の減算を計算する必要がない．

scalar 倍算の流れとして，素数 k を 2 進数に変換する．次に 2 進数の最大桁数から順番に値を見て 1 だった場合，R0 ← R0 + R1，R1 ← 2 * R1 を行う．0 だった場合，R1 ← R0 + R1，R0 ← 2 * R0 の計算を行う．全ての計算が終了すると R1 を返す．

次に，Montgomery 曲線を使用した楕円曲線法プログラムの stage 2 を作成した．stage2 のアルゴリズムは，twisted edwards 曲線を使用したプログラム同様に Baby-step Giant-step を元に作成した．工夫した点として，本プログラムでは，Giant-step の区切りを 210 として作成した．そのため，Baby-step は 210 と互いに素な値が必要となる．よって，偶数は必要ないので，計算方法を工夫することで奇数倍の計算しか行わないようにした．加算を用いて奇数倍のみを作成する計算方法として，

$$k + (k + 1) = 2k + 1 \quad (k = 1, 2, ..., m) \tag{3.9}$$

という式を用いた．この計算式の結果は 2k がどの k の値にもかかわらず必ず偶数となるため，2k+1 は必ず奇数になる．詳細な計算は，Listing3.4 に記す．

Listing 3.4　Montgomery 曲線を使用した Baby-step の取り方

```
1     set(add_mP, P);
2     montgomery_double(double_mP, P, ma, N);
3     set(temp, P);
4     set(temp2, double_mP);
5
6     for (i = 0; i < 53; i++)
7       {
8         mpz_set_ui (ti, 2 * i + 1);
9         mpz_gcd (gcd, ti, tf);
10        if (mpz_cmp_ui (gcd, 1) == 0)
11          {
12            set (baby_step[i], add_mP);
13          }
14        if (i == 0 || i == 1)
15          {
16            montgomery_add (add_mP, add_mP, double_mP, P, N);
17            if (i == 0)
18              set (temp3, add_mP);
19          }
20        else
21          {
22            montgomery_add (temp, temp3, P, temp2, N);
23            montgomery_add (add_mP, temp, temp3, P, N);
24            set (temp2, temp3);
25            set (temp3, temp);
26          }
27      }
```

計算の概要を以下に示す．

- P の初期値は，任意に定められた Montgomery 曲線上の一点の x，y，z の値を格納している．

- temp，add_mp の初期値は P で，double_mp の初期値は，P を 2 倍算したものである．
- temp2 の初期値は，double_mp を代入したものである．
- temp3 の初期値は add_mp と double_mp を add 関数で加算した結果の 3P である．
- 23 行目の montgomery_add 関数は式 (3.6) を利用したものであり，add_mP に奇数値を代入している．

また，Baby-step 同様に Giant-step も montgomery_add 関数を利用するため，計算方法を工夫する必要があった．Giant-step におけるプログラムを Listing3.5 に記す．

Listing 3.5　Montgomery 曲線を使用した Giant-step の取り方

```
1  unsigned long int v2;
2  PRIME_TYPE *s=&(prime_table[nextprimeindex(prime_table,B1)]);
3  v2 = (*s - f / 2) / f;
4  montgomery_scalar (giant_step, P, f, ma, N);
5  montgomery_scalar (Giant, giant_step, v2, ma, N);
6  montgomery_scalar (temp, giant_step, (v2 - 1), ma, N);
7  projective_point_set (temp2, Giant);
8  while (*s <= B2){
9          v = (*s - f / 2) / f;
10         while (v != v2){
11                 v2++;
12                 montgomery_add (Giant, Giant, giant_step, temp, N);
13                 projective_point_set (temp3, Giant);
14                 projective_point_set (temp, temp2);
15                 projective_point_set (temp2, temp3);
16     }
17     u = abs ((*s - f / 2) % f - f / 2);
18     mpz_mul_mod (product, Giant->X, baby_step[(u - 1) / 2]->Z, N);
19     mpz_mul_mod (a1, Giant->Z, baby_step[(u - 1) / 2]->X, N);
20     mpz_sub (product, product, a1);
21     mpz_mul_mod (d, d, product, N);
22     s++;
23 }
24 mpz_gcd (p, d, N);
```

計算の概要を以下に示す．

- 間接参照演算子*s には素数テーブルで格納した素数の値が入っている．22 行目の s++ でポインタをインクリメントすることによって，素数テーブル内の次の素数を割り当てている．
- Giant の初期値は v2fP であり，素数テーブルへ参照している値の倍算である．
- giant_step の初期値は fP であり，1 つ分の大きさの Giant-step を示している．
- temp の初期値は (v2-1)fP であり，Giant と giant_step の距離を示している．
- v は素数と Giant-step の大きさ f の徐算であり，素数が Giant-step を越えるたびに v の値が 1 ずつ増えていく．
- v2 は過去の v を保存するための変数であり，10 行目の while 分で，v != 2 が成り立つ時に v2 から v までの回数分の Giant-step を加算を行う．
- 12 行目の montgomery_add は Giant と giant_step の距離を第 4 引数に代入する必要があるため，temp を用意し代入した．

（※文責: 小澤貴也）

3.3.4 素数テーブルの実装

プログラム中の実行時間について調べたところ，gmp_nextprime 関数が時間を多く使っていることがわかった． gmp_nextprime 関数は与えられた数の次の素数を返す関数である．この関数は内部で計算を行っていたため，より効率良く素数を求めることができる素数テーブルを作成することとした．

素数テーブルの概要を示す．素数テーブルとはあらかじめメモリ上に小さい順に素数を用意しておく手法である．メモリ上には unsigned long long int 型の配列として素数が 2,3,5,7,11,13,... と連番で配置されており，配列としてアクセスすることにより簡単に n 番目の素数を取得することができる．この手法により，毎回計算せず 1 回のメモリアクセスで次の素数を取得できるため，高速化につながる．

今回の素数テーブルで必要な素数の上限はプログラムのパラメータである B1,B2 に依存する．素因数の桁数を 50 桁と想定すると最適な B1 の値は 4300 万となる．B2 を B1 の 100 倍とすると B2 は 43 億となり，50 桁の素因数を想定した場合に必要な素数の上限は 43 億となる．また同様に，60 桁の素因数が想定される場合は 2900 億までの素数が必要となる．素数を格納するデータ型は unsigned long long int としているが，これは 2900 億を格納するためには unsigned long int ではオーバーフローするためである．

なお，この手法は多くのメモリを消費する．前期では 43 億までの素数テーブルを作成したが，後期では 2900 億までの素数テーブルを作成した．43 億までの素数テーブルはおよそ 1.6[GB] となり，2900 億の素数テーブルはおよそ 92[GB] となった．

具体的な実装としては素数テーブルの内容となるファイルを作成し，プログラムを実行した際にそのファイルをメモリ上に配置して利用する．素数テーブルを作成したことにより，最初に素数テーブルをメモリ上に読み込む時間は必要となるが，楕円曲線を用いた演算の部分は大幅に高速化した．

（※文責: 外山拓）

3.3.5 VPS の導入

より多くの素因数分解を試すため，VPS を導入することとした．前期まででは学内のサーバー台で素因数分解を行っていたが，一台のみでは素因数分解の試行回数が限られていた．また，素因数分解の実行中はプログラムの開発が行うことが難しいという事情があった．そのため，後期からは新たに 2 台の VPS を導入することとした．これにより学内サーバー，VPS1，VPS2 の 3 台により開発・実行を行えるようになった．学内サーバーは主に ECMNET の素因数分解に使用し，VPS1 はプログラムの開発，VPS2 は STUDIO KAMADA の素因数分解に使用した．その結果，開発と実行を円滑に進めることができ，STUDIO KAMADA の合成数 2 つを素因数分解するという結果に繋がった．

（※文責: 外山拓）

3.3.6　開発環境と実行環境

プログラムの実行環境について，前期は Xeon E5-2640 を用いてプログラムを実行していた．Xeon E5-2640 のスペックとしては，メモリ 64[GB] で CPU12 コアである．しかし，多くの素因数分解を行うためには計算機資源が足りないと感じたため，新たに VPS を 2 台追加した．VPS のスペックとしては，メモリ 16[GB] 仮想 8 コア SSD200[GB] とした．

（※文責: 外山拓）

3.3.7　プログラムの実行方法

作成したプログラムには方式，合成数 N，B1，B2，出力ファイル名を入力する必要がある．つまり，以下のように使用する．

./funecm [options] N B1 B2 filename

options には以下のものが使用できる．

-h: ヘルプを表示する．
-c number: 楕円曲線の試行回数を変更する．
-w bit: Window Method における window size を変更する．
-a: Atkin-Moraine ECPP を使用する．
-m: Montgomery を使用する．

合成数 N には素因数分解を試したい数とする．また，B1 は予想される素因数の桁数により決定する．また，B2 は基本的に B1 の 100 倍として設定する．出力ファイル名には，プログラムの実行結果を出力するファイル名を表示する．出力ファイルには，最初に合成数 N，B1，B2，楕円曲線の試行回数，Window Method における window size，thread 数が記載されている．その後試行した曲線ごとに，Atkin-Moraine ECPP のパラメータ d，Stage1 と Stage2 とにかかった時間，一試行にかかった時間 (つまり Stage1 と Stage2 をあわせた時間) が記載される．素因数が見つかった場合は 2 つの素因数とその桁数が表示され，最後にプログラム全体でかかった時間が表示される．

（※文責: 外山拓）

3.3.8　本プロジェクトで作成したソースファイルの説明書の作成

今年度では新たに，素因数分解の自動化を行うためのシェルスクリプトや合成数を作成するソースファイルなどを追加した．また，Montgomery 曲線のみでの素因数分解を行うために，前年度のソースファイルに新たなオプションを追加した．上記のように今年度から様々な機能が増えた．そのため，今までのプログラムを来年度のプロジェクトメンバに理解を促すための作業が必要だった．理解を促す作業として，今年度では学内の Xeon E5-250 に保存したソースファイルの使い方を記したファイルを追加した．また，そのソースファイルのソースコード中にある分かりにくい部分にコメントを追加した．

（※文責: 福永慧）

3.3.9　スクリプトの作成

今回の活動では 3 つのスクリプトを作成し，それぞれテストデータの作成，プログラムの実行，データの集計に使用した．

テストデータの生成は乱数をもとに素数を 2 つ生成し，その 2 つの素数をかけ合わせたものを素因数分解に使用する合成数とした．なお，テストデータの生成では合成数の桁数を指定する．合成数のもととなる 2 つの素数の桁数は，それぞれ合成数の桁数の半分に近い桁数となるようにした．

プログラムの実行に関してもスクリプトを作成した．FUN-ECM のプログラムは桁数が大きくなるにつれ実行時間が準指数関数的に増大していく．プログラムの実行を人間が手で行うことは非効率であるためである．スクリプトは入力として一行に以下の内容を含む CSV ファイルを受け取る．

桁数，素数 A，素数 B，合成数 (A × B)，B1，B2

スクリプトは各行に記載されている合成数に対して指定された B1，B2 を用いて，今年度と昨年度の FUN-ECM のプログラムを実行する．すべての行の処理が終了すると，実行結果の出力ファイルを解析し，各桁の平均を算出しファイルに保存する．なお，スクリプトの実行には nohup コマンドを用いることでログアウト後も処理を継続することができる．

データの集計に関しては，プログラムの実行を行うと，結果や実行時間についてのログファイルが保存される．このログファイルを加工し，実行時間を CSV 形式で保存するスクリプトを作成した．

（※文責: 外山拓）

3.3.10　Xeon におけるソースコードの階層化

昨年度ではソースコードのファルダ分けがされておらず，すべて同一のフォルダに実行ファイルやソースコードが含まれていた．そのため，フォルダ内にファイルが多く含まれ，データの管理やソースファイルの管理が難しくなっていた今年度では，ソースコードを管理しやすくするために，.c と.o やヘッダファイル，実行ファイルをフォルダごとに分類を行った. ソースコードの階層化につき Makefile を大幅に変更した Makefile を実行した際，生成されるファイルは自動的に各フォルダに振り分けられるようになっている．詳細なディレクトリは下記に示す．

（※文責: 水上敬介）

ディレクトリ構造

```
funecm　※各ディレクトリに説明のための Readme.txt が存在する
├── bin <-プログラムファイル
│   ├── funecm
├── data <-素数テーブルのデータ
│   ├── prime_data.bin
│   ├── prime_table_generator
│   └── Readme.txt
├── inc <-ヘッダファイル
│   ├── atkin.h
│   ├── bsgs.h
│   ├── double_add.h
│   ├── ecm.h
│   ├── gmp.h
│   ├── normal_add.h
│   ├── point.h
│   ├── prime.h
│   └── scalar.h
├── Makefile
├── result <-実行結果フォルダ
├── obj <-オブジェクトファイル
│   ├── atkin.o
│   ├── bsgs.o
│   ├── double_add.o
│   ├── ecm.o
│   ├── main.o
│   ├── normal_add.o
│   ├── point.o
│   ├── prime.o
│   └── scalar.o
└── src <-.c ファイル
    ├── atkin.c
    ├── bsgs.c
    ├── double_add.c
    ├── ecm.c
    ├── main.c
    ├── normal_add.c
    ├── point.c
    ├── prime.c
    └── scalar.c
```

（※文責: 水上敬介）

3.4 中間発表

中間発表は，7 月 14 日 (4 時限〜5 時限) に，1 階階スタジオで行った．発表時間は，全 6 回で前後半 3 回ずつ行った．1 回あたり 15 分の発表で，10 分間プレゼンして，残り 5 分間質問を受けた．人数の振り分けは前半 3 人，後半 4 人で行った．前後半合わせて教員，生徒合計 74 名聴衆した．

（※文責: 小澤貴也）

3.4.1 準備

ポスター

初めに，前年度のプロジェクトで作成されたポスターを参考に構成を決定した．次に，概要，基礎学習，理論班，プログラム班の 4 つの項目に分け，作成を分担した．ポスターには，イラストを使用するなどよりわかりやすく楕円曲線法について理解できるように工夫した．ポスターは，理論班・プログラム班合計 2 枚作成した．ポスターの作成には「Microsoft PowerPoint」というソフトウェアを使用した．ポスターが完成次第，理論班・プログラム班でレビューを行い，誤字脱字やフォントの違い等を修正した．しかし，中間発表当日に誤字が発見された．

（※文責: 小澤貴也）

プレゼンテーション資料

本プロジェクトの内容を説明するのにポスターだけでは不十分と判断しプレゼンテーション資料を作成することにした．理論班，プログラム班から一人ずつメインに作成する人を決め，他メンバーと話し合いながら，発表内容を考えた．プレゼンテーション資料の作成には「Microsoft PowerPoint」を使用し，共有に「SharePoint Online for Office 365」を使用した．また，先生から，内容やデザインの指摘を受け，改良することで，どんな人でも理解できるように努めた．

（※文責: 福永慧）

原稿

前述のプレゼンテーション資料の作成と並行して，発表用の原稿の作成を行い，グループ内で担当を決め各ページの原稿を作成した．楕円曲線方法を理解してもらうようになるため，基礎学習の部分を最低限の知識だけを伝えるように作成した．作成した原稿は，担当教員に確認していただき，伝わりにくい表現の修正を行った．

（※文責: 水上敬介）

3.4.2 アンケート

中間発表用アンケートを作成した．アンケートの質問項目は

- 発表技術について（10 段階評価，コメント）

- 発表内容について（10 段階評価，コメント）
- 発表内容は理解しやすいものでしたか（10 段階評価）
- その他の発表内容について（コメント）

の 4 点であった．

中間発表終了後のアンケートの回答を分析を行った．

アンケートは発表を聞いた教員生徒合計 74 人を対象に回答を行った．分析結果として「発表技術について」は，10 段階評価の内平均 6.6 であった．コメントとしては，「声が小さい」「わかりにくい」などがあった．また，「発表内容について」では，10 段階評価の内平均 5.4 であった．コメントとしては，「内容が難しかった」「理解できなかった」などがあった．「発表内容は理解しやすいものでしたか」は，10 段階評価の内平均 4.9 であった．

アンケートの分析結果として，発表技術では，声の小さいなど発表者の準備不足があったと考える．また，発表内容については，全体的に低い評価が多かった．理由として，楕円曲線法という馴染みがない知識について理解が難しかったなどが考えられる．しかし，「その他の発表内容について」の項目でポスターやスライドなどについては「イラストがあってわかりやすい」など肯定的なコメントがあった．

以上のことから反省点として，まず発表の準備不足であったことが考えられる．また，楕円曲線法についてわかりやすく説明を行う工夫をする必要があった．

（※文責: 小澤貴也）

3.4.3 発表

発表はスライドの説明をメインとし，ポスターにはスライドの内容をより詳しくしたものを用意した．中間発表会での発表を行い，その結果，スライドに数式が多く，発表時間の短さも相まって，数学・楕円曲線法をよく知らない人にとって分かりにくい発表になってしまっていた．発表の反省を行った結果，後期末の発表では数学に興味のない人でも概観が分かるようにすることに決めた．具体的には，スライドでは極力数式を出さずに説明し，興味を持った人に対して，ポスターを用いて専門的な説明をする，という形式をとることにした．

（※文責: 福永慧）

3.5 最終発表

成果発表会は，12 月 8 日 (4 時限〜5 時限) に 3 階モールで行った．発表時間は，全 6 回で前後半 3 回ずつ行った．一回あたり 15 分の発表で，10 分間プレゼンして，残り 5 分間質問を受け付けた．前半 4 人，後半 3 人で発表を行った．前後半合わせて教員，生徒合計 87 名聴衆した．

（※文責: 小澤貴也）

3.5.1 準備

ポスター

初めに，中間発表で作成されたポスターを参考に構成を決定した．次に，概要，活動内容，楕円曲線法について，計測結果の 4 つの項目に分け，作成を分担した．今年度活動では，STUDIO KAMADA に本プロジェクトの名前を載せることができたため，その結果を知ってもらうために QR コードを貼った．また，中間発表で作ったポスターの内容に加えて，1 年間の活動を分かりやすくまとめたり，後期で理解したこと，去年と今年の前期と後期のプログラムの計測結果のグラフなどを 3 枚のポスターでまとめた． ポスターの作成には「Microsoft PowerPoint」というソフトウェアを使用した．ポスターが完成次第，理論班・プログラム班でレビューを行い，誤字脱字やフォントの違い等を修正した．

（※文責: 小澤貴也）

プレゼンテーション資料

中間発表と同様，スライドを作成した．しかし，中間発表のスライドでは，理論的なことが多く書かれていたため，スライドを見ただけでは理解できないところがあった．そのため，成果発表でのスライドでは，数式をできる限り減らし，理論的な説明は発表後にポスターを見てもらうという形をとった．さらに，STUDIO KAMADA に載ったことを聞いてもらった人たちに知ってもらうために，ポスターだけでなく，スライドにも載せることにした．

（※文責: 落合航平）

原稿

前述のプレゼンテーション資料の作成と並行して，発表用の原稿の作成を行い，グループ内で担当を決め各ページの原稿を作成した．前期では，専門用語や数式を使用しすぎて内容が理解できないという意見があった．そのため，成果発表用の原稿では，可能な限り専門用語や数式を減らし当プロジェクトの概要を理解してもらえるように努めた．発表練習として，メンバーの前で発表をしあい冗長な表現をなくし，聴衆に理解がえられるように原稿を作成した．

（※文責: 水上敬介）

3.5.2 アンケート

最終成果発表用アンケートを作成した．アンケートの質問項目は

- 発表技術について（10 段階評価，コメント）
- 発表内容について（10 段階評価，コメント）
- 発表内容は理解しやすいものでしたか（10 段階評価）
- その他の発表内容について（コメント）

の 4 点であった．

最終成果発表終了後のアンケートの回答を分析を行った．

アンケートは発表を聞いた教員生徒合計 87 人を対象に回答を行った．分析結果として

「発表技術について」は，10 段階評価の内平均 8.8 であった．コメントとしては，「聞きやすかった」「スライドを上手く使っていた」などがあった．また，「発表内容について」では，10 段階評価の内平均 6.3 であった．コメントとしては，「活動内容がわかった」「楕円曲線法が少しわかった」などがあった．「発表内容は理解しやすいものでしたか」は，10 段階評価の内平均 6.7 であった．

アンケートの分析結果として，発表技術では，発表に対して準備は出来ていたと考えられる．また，発表内容については，中間発表時比べると評価が上がった．理由として，中間発表の反省から楕円曲線法など技術的な内容をポスターに書くことで発表には，あまり理解が難しい内容を簡略化したことが考えられる．しかし，そのことによって，技術的内容を発表から簡略化したことで「内容がなかった」など否定的な意見もあった．

以上のことから反省点として，まず中間発表と比較して全体的には良い評価だったと考える．しかしまだ，楕円曲線法についてわからなかったという意見もあったので，よりわかりやすくすることが挙げられる．また，内容を簡略化する部分と発表するべき部分をしっかりと見極める必要がある．

（※文責: 水上敬介）

3.5.3 発表

中間発表の反省を踏まえ，スライドでの説明はほぼ数式を使わず行った．また，今年は SUDIO KAMADA に名前を載せることができたので，そのことを最初の方で発表し，かつ時間を 8〜9 分に収めて発表を行った．その後の時間は質疑応答やポスターを見ていただいた．

成果発表を見ていただいた方からの評価アンケートでは，なぜ楕円曲線法を使うのか，どうして高速化できたのかなどという，理論的な説明が足りないという指摘をいただき，少し省きすぎてしまったと反省した．その一方で，分りやすかったという声が中間発表のときよりも増え，成果に対しての称賛なども得れたので，中間発表の反省を少しは踏まえられた発表になったと考えられる．

（※文責: 水上敬介）

第 4 章　プロジェクト内のインターワーキング

- 中島俊平（プロジェクトリーダー・理論班）
  (1) 楕円曲線法の基礎を学んだ.
  (2) 大まかな作業スケジュールを作成し，進捗管理を行った.
  (3) ”Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware” を落合，金子と協力して読解した.
  (4) PARI/GP で Baby-step Giant-step のプログラムを作成し，メンバーに共有することで，理解を深めた.
  (5) 中間発表会に向けて，進捗管理を行った.
  (6) 中間発表会に向けて，「理論班」の部分のポスターの原案を作成した.
  (7) 落合，金子と協力して，FFT，素数ペアリング，Pollard's ρ 法の理解をした.
  (8) 最終発表会に向けて，進捗管理を行った.
  (9) 最終発表会に向けて，「理論班」の部分のポスター・スライドの原案を作成した.
- 金子真澄（理論班）
  (1) 楕円曲線法の基礎を学んだ.
  (2) ”Implementing the Elliptic Curve Method o Factoring in Reconfigurable Hardware” を落合，中島と協力して読解した.
  (3) 落合，中島と協力して，PARI/GP で ECM を実装した.
  (4) 落合, 中島と協力して, Baby-step Giant-step のアルゴリズムをプログラム班に教えた.
  (5) 中間発表に向けて，「理論班」の部分のスライドを作成した.
  (6) 落合，中島と協力して，FFT の理解をした.
  (7) 落合，中島と協力して，素数ペアリングの理解をした.
  (8) 落合，中島と協力して，Pollard's ρ 法を理解した.
  (9) 落合，中島と協力して，Brent のアルゴリズムを理解した.
- 落合航平（理論班）
  (1) 楕円曲線法の基礎を学んだ.
  (2) ”Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware” を金子，中島と協力して読解した.
  (3) 金子, 中島と協力して, Baby-step Giant-step のアルゴリズムをプログラム班に教えた.
  (4) 金子，中島と協力して，PARI/GP で ECM を実装した.
  (5) 中間発表会に向けて，ポスターの「概要」の部分を作成した.
  (6) 金子，中島と協力して，FFT の理解をした.
  (7) 金子，中島と協力して，素数ペアリングの理解をした.
  (8) 小澤，水上と協力して，ポスターを作成した.
  (9) 水上，福永と協力して，成果発表のスライド，原稿を作成した.
- 水上敬介（プログラム班）
  (1) 楕円曲線法の基礎を学んだ.
  (2) プログラム班と協力して昨年度のプログラムの不具合の修正を行った.

(3) プログラム班でのスケジュール管理を行った.
(4) プログラム班での進行役を務め，課題に対してのメンバーの役割を決めた.
(5) 理論班から PARI/GP を用いて実装されたアルゴリズムをプログラムに実装した.
(6) GitHub のソースコードの管理を行った.
(7) 中間発表に向けてのスライド資料・プログラム班の原稿の原案を作成した.
(8) 論文の”Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware”を福永，小澤と協力して読解した.
(9) 福永，小澤と協力して Montgomery 曲線での楕円曲線法で用いる，加算，2 倍算，scalar 倍の関数を作成した.
(10) 福永，小澤と協力して Montgomery 曲線での Stage1 を作成した.
(11) 福永，小澤と協力して Montgomery 曲線での Stage2 を作成した.
(12) 最終発表会に向けて，福永と協力して各曲線での実行速度の測定を行った.
(13) 最終発表会に向けて，小澤と協力して後期のポスターを作成した.
(14) 後期の実行速度のデータ収集を行い，集計を行った.

- 福永慧（プログラム班）
  (1) 楕円曲線法の基礎を学んだ.
  (2) Baby-step Giant-step の文献を読み，原理を理解した.
  (3) 昨年度の FUN-ECM で作られたプログラムを理解した.
  (4) Baby-step Giant-step のアルゴリズムを実装した.
  (5) 中間発表に向けて，評価アンケートの作成をした.
  (6) 中間発表に向けて，ポスターの英訳を行った.
  (7) ”Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware”を水上，小澤と協力して読解した.
  (8) 水上，小澤と協力して Montgomery 曲線での楕円曲線法で用いる， 加算， 2 倍算，scalar 倍算の関数を作成した.
  (9) 水上と協力して PARI/GP を用いて作成した加算，2 倍算，scalar 倍算の関数のテストを行った.
  (10) 水上，小澤と協力して Montgomery 曲線での Stage1，Stage2 を作成した.
  (11) 水上と協力して作成段階のプログラムを Github で一時共有した.
  (12) Xeon に上がっている楕円曲線法のソースコードの使い方を説明するファイルを作成した.
  (13) 成果発表会に向けて，評価アンケートの編集を行った.
  (14) 成果発表会に向けて，水上と協力して各曲線での実行速度の測定を行った.
  (15) 成果発表会に向けて，落合と協力してスライドを作成した.
- 小澤貴也（プログラム班）
  (1) 楕円曲線法の基礎を学んだ.
  (2) 昨年度の ECM プログラムを理解した.
  (3) 昨年度のプログラムの修正をした.
  (4) Baby-step Giant-step のアルゴリズムを実装した.
  (5) 中間発表に向けて，ポスターを作成した.
  (6) ”Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware”を水上， 福永と協力して読解した.

(7) 水上，福永と協力して Montgomery 曲線での楕円曲線法で用いる， 加算， 2 倍算，scalar 倍の関数を作成した．
(8) 水上，福永と協力して Montgomery 曲線での Stage1 を作成した．
(9) 水上，福永と協力して Montgomery 曲線での stage2 を作成した．
(10) 成果発表会に向けて，水上と協力してポスター作成した．

- 外山拓（プログラム班）
  (1) 楕円曲線法の基礎を学んだ．
  (2) Baby-step Giant-step の原理を学んだ．
  (3) Baby-step Giant-step の実装を行った．
  (4) プログラムを実行するサーバの管理を行った．
  (5) 昨年度の FUN-ECM のプログラムを理解した．
  (6) 昨年度のプログラムの並列処理を高速化した．
  (7) テストデータを作成するためのスクリプトを作成した．
  (8) 自動実行を行うためのスクリプトを作成した．
  (9) VPS の導入を行い，各種設定を行った．
  (10) 素数テーブルの設計，実装を行った．
  (11) STUDIO KAMADA に記載されている合成数のうち，素因数分解を行う合成数を選んだ．
  (12) プログラムの可読性を向上させた．
  (13) プログラムのディレクトリを階層化し，可読性を向上させた．
  (14) プログラムに関する適切なパラメータについて考察した．

（※文責: 中島俊平）

第 5 章　結果

本プロジェクトでは，前期はメンバー全員で楕円曲線法の学習から初め，楕円曲線法の概要を理解した．その後は理論班とプログラム班に分かれた．

（※文責: 外山拓）

5.1　理論班

前期では，まず Baby-step Giant-step の理解がある．基礎学習を終えた後，理論班は Baby-step Giant-step について学習した．その後，理論班のメンバーそれぞれがそれに関する論文を探し，読むことで，Baby-step Giant-step をより深く理解することができた．そして，その知識を生かし，理論班で最適なアルゴリズムを考え，プログラム班に伝えることができた．また，去年のプログラムで使われていた twiceted Edwards 曲線を Montgomery 曲線に変換することで，プログラムを高速化することができることをプログラム班に伝えることができた．そして，昨年度のプログラムに誤りがあったため，改良すべき点としてプログラム班に提案をしたことが挙げられる．

後期では，プログラムをより高速化させるため，素数ペアリング，FFT の理解と ρ 法の実装検討が挙げられる．ρ 法の実装は様々な論文を探したが，楕円曲線を用いた方法を見つけられず，楕円曲線を用いた方法と比べて効率的に計算することができないと考えたため，最終的に実装を行わなかった．素数ペアリングについては，理解してプログラム班に伝えることができた．しかし，FFT は成果発表の準備，プログラムを動かす時間等により，しっかりと理解してプログラム班へ伝えるというところまでには至らなかった．

（※文責: 落合航平）

5.2　プログラム班

5.2.1　計測結果

プログラム班の前期の活動では，Stage2 において Baby-step Giant-step を実装した．確立した計測方法を実証したところ，昨年度のプログラムより Stage2 の精度が向上し，高速化していることが確認できた．実証した結果は図 5.1，図 5.2 のようになった

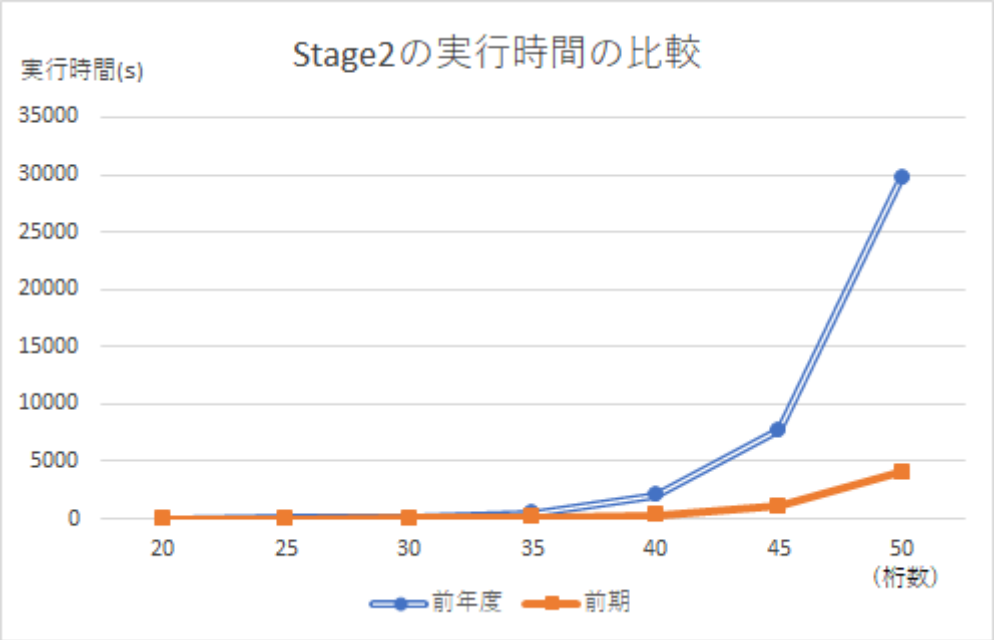

図 5.1　Stage2 の実行にかかった時間

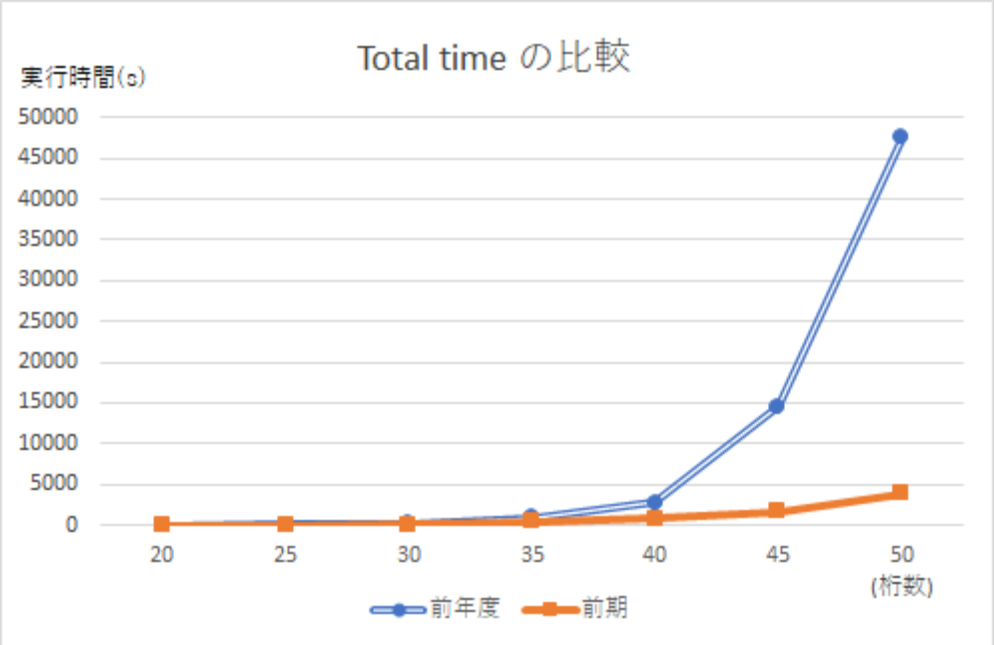

図 5.2　プログラム全体の実行にかかった時間

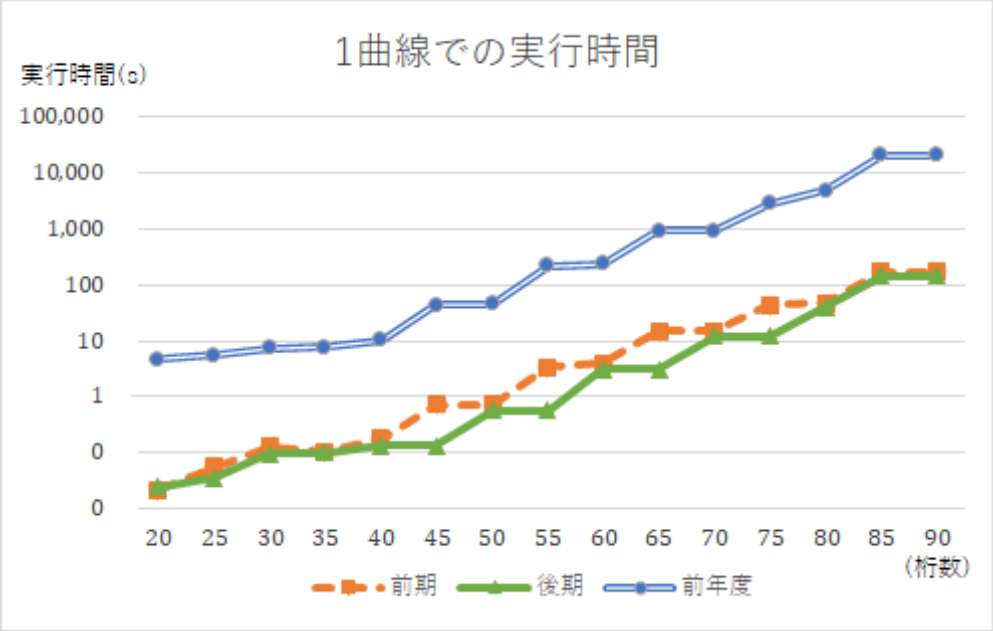

図 5.3　1 つの楕円曲線における実行時間

図 5.1，図 5.2 の結果から，Stage2 においての計算速度が 6〜7 倍向上したことが分かり，プログラム全体の速度の大幅な改善が確認された．後期では，Montgomery 曲線を用いた計算を実装した．前期と計算方法を変更したものを実証した．結果は図 5.3 のようになった．図 5.3 の結果から 1 つの楕円曲線における Stage1 と Stage2 の計算速度が昨年度より約 80 倍向上したことがわかり，プログラム全体の速度の大幅な改善が確認された．

（※文責: 水上敬介）

5.2.2　STUDIO KAMADA

STUDIO KAMADA に載っているまだ素因数が見つかっていなかった合成数 2 つの素因数分解に成功した．1 つめの合成数は $\frac{52\times10^{246}+11}{9}$ である．素因数分解を行う前は以下のようになっていた．なお，括弧内の値は桁数を示す．

$7 \times 103 \times 117762003928963_{<15>} \times 68048786062214478469218668994018055669526980242453547559171263711276158032268303309020968416265854862611637394409172500439894855913620105666975824826219068165435329048574429632381845191827800361219414933427569218654888877675993473_{<230>}$

230 桁の部分は合成数であるため，この部分の素因数分解を行った．プログラムを実行した結果，以下のように素因数分解を行うことができた．

$$28753460898983765014039295571154959466769317050690 07_{<52>} \times$$
$$23666294051099612175553719415465240274530286448304318716082751524302213245 42$$
$$11837339898502443688813509377105093362890791387756908406454911971355789544 00$$
$$53134451072345916622441463 9_{<179>}$$

プログラムの実行は VPS で行った．また，プログラムのパラメータ B1，B2 は表 3.1 を用いて決定した．曲線の試行回数は 10000 回を上限とし，上限に達した時点で素因数が見つからなかった場合は次のパラメータを使用して再度 10000 回施行するという方式を用いた．実行時間としては，40 桁の試行で 135804[s]，45 桁の試行で 439470[s]，50 桁の試行で 242554[s] となり，素因数を発見した．合計では 817828[s]=9 日 11 時間 10 分 28 秒となった．

2 つめの合成数は $\frac{76\times10^{247}-31}{9}$ である．素因数分解を行う前は以下のようになっていた．

$$3 \times 85711 \times 650112876289_{<12>} \times 1337489853071_{<13>} \times$$
$$37768870939999955935652840314549308400289920473643240278782149969812962642 85$$
$$77551917195361263313610647697437239342828593788456435090167692417024548383 84$$
$$53993029222427328138588151891118136810735008254956779607733862496 83_{<219>}$$

219 桁の部分は合成数である，この部分の素因数分解を行った．プログラムを実行した結果，以下のように素因数分解を行うことができた．

$$91585637851471680049375897339286952218012351_{<44>} \times$$
$$41238857779482126403208428029315652193041420561058173603962752654899856898 97$$
$$56211778640612035939924709473942448503628774034777252714666247529230530423 27$$
$$56815902519295676657133_{<175>}$$

実行は VPS で行い，プログラムのパラメータ B1，B2 は ECMNET の Optimal Parameter を参考に素因数の桁数が 50 桁に対応するものを使用した．実行時間としては，221701[s]=2 日 13 時間 35 分 01 秒となった．

（※文責: 外山拓）

第 6 章　まとめ

6.1　前期活動成果

前期はメンバー全員で楕円曲線法の学習から初め，楕円曲線法の概要を理解した．その後は理論班とプログラム班に分かれた．

理論班は Baby-step Giant-step について理解し，その内容をプログラム班に教えた．また，様々な論文 [4][5] を探して，効率の良い曲線や計算量を少なくするために計算方法の見直し，アルゴリズムについての考察を行った．

プログラム班は昨年度のプログラムの解読を行い，省メモリ化，高速化を行なった．また，理論班の説明を受け Baby-step Giant-step のアルゴリズムの実装を行った．

（※文責: 外山拓）

6.2　後期活動成果

後期では，理論班は Montgomery 曲線を利用した楕円曲線法について理解し，その内容をプログラム班に伝えることができた．また，Baby-step Giant-step をより高速化するため，素数ペアリングの理解をした．また，Pollard's ρ 法という，新たな Stage2 のアルゴリズムの実装検証や FFT の理解にも取り組んだ．

プログラム班はスクリプトや素数テーブルを作成するグループと Montgomery 曲線を使用した楕円曲線法の Stage1 と Stage2 を作成するグループに分かれ活動を行った．これらの作業をそれぞれ行い，完成させた．完成後は去年，前期，後期で作成したプログラムの速度比較を行った．その結果，合成数が 20 90 桁の場合，楕円曲線 1 つ分を回す実行速度が去年度のプログラムよりも今年度のプログラムの方が約 80 倍早くなることが分かった．また，前期から VPS で STUDIO KAMADA での合成数の素因数分解を行っていたプログラムの出力結果を確認した．結果として 2 つの合成数の分解に成功し，STUDIO KAMADA に名前を載せることが出来た．

（※文責: 福永慧）

6.3　今後の展望

最初に今後の課題として，理論班では FFT や符号付き 2 進数展開，Rucus chain などの学習・理解からプログラム班へ伝えることが残っている．プログラム班も，Stage1 で Montgomery 曲線を利用した後，Stage2 で twisted edwards 曲線を利用する曲線変換を導入するということが残っている．それらをすることで，プログラムの高速化がまだまだできることが課題の 1 つとして挙げられる．

また，最終的にできた今年のプログラムは，2 ヶ月ほどしか動かすことができなかった．そのため，大きな合成数を試すためには，プログラムを動かすために長い時間が必要だということが分かった．しかし，2 ヶ月という短い期間で STUDIO KAMADA に名前を載せるということが

できた．もし，プログラムを動かす時間の確保ができれば，本プロジェクトの目的の一つである，ECMNET に名前を載せるということができたかもしれないだろう．

以上より，理論班とプログラム班それぞれのやり残したことをやり遂げることでの更なるプログラムの高速化，プログラムを動かすための時間の確保，ECMNET に名前を載せるということが課題である．また，去年作られた本プロジェクトのウェブサイトがあるが，それを編集する権限を引き継いでいなかったので，去年の方から権限を引き継ぐ，または新しくウェブサイトを作ることで，本プロジェクトの活動をいろんな人たちに知ってもらえるようにしたいと思う．

最後に，本プロジェクトの背景で書いていた，RSA 暗号の安全性を確かめるという部分について，現在どれだけの桁数までが本当に安全であるかということを検証することを新しく行っていきたい．

（※文責: 落合航平）

参考文献

[1] 伊豆哲也. 楕円曲線暗号入門. https://researchmap.jp/mulzrkzae-42427/, (最終アクセス 2018 年 1 月 19 日)

[2] ECMNET. https://members.loria.fr/PZimmermann/ecmnet/, (最終アクセス 2018 年 1 月 19 日)

[3] STUDIO KAMADA. http://stdkmd.com/, （最終アクセス 2018 年 1 月 19 日）

[4] Kris Gaj, Soonhak Kwon, Patrick Baier, Paul Kohlbrenner, Hoang Le, Mohammed Khaleeluddin, Ramakrishna Bachimanchi. Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware. Cryptographic Hardware and Embedded Systems - CHES 2006, 2006

[5] 楕円曲線法（ECM）faireal.net. http://www.faireal.net/articles/6/07/,（最終アクセス 2018 年 1 月 19 日）.

[6] Peter Lawrence Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. Mathematics of Computation. 48 (177): 243264, 1987

[7] Richard P. Brent. Some Integer Factorization Algorithms using Elliptic Curves. Australian Computer Science Communications 8 (1986), 149-163

[8] Peter Lawrence Montgomery. An FFT extension of the elliptic curve method of factorization. University of California at Los Angeles Los Angeles, CA, USA, 1992