FUN-ECM プロジェクト
FUN-ECM project

A グループ (A)　Search Group (A)

1015014　中島俊平　Shunpei Nakajima

1　背景

暗号化技術は，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化技術が利用されている．しかし，暗号化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．暗号の安全性評価に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，楕円曲線法を学ぶ．

現在，有名な公開鍵暗号の 1 つに 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た暗号である．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．この性質を利用して RSA 暗号を解読する．

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．前年度までのプロジェクト活動によって Stage1 のプログラムは完成していた．一方 Stage2 については高速化されていないという問題点があった．そこで今年度では，Stage2 をより高速化させることを課題とした．また，ECMNET や STUDIO KAMADA という暗号解読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．

2　課題の設定と到達目標

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．

- 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

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．

3 課題解決のプロセスとその結果

3.1 基礎学習

まず，ECM の理解のために基礎学習を行い，以下のことを理解した．

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．

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

Algorithm 1 Euclidean Algorithm

Require: $a, b \in \mathbb{N}, \quad a, b \neq 0, \quad a \geq b$
Ensure: $\gcd(a, b)$
  while $b \neq 0$ **do**
    $q \leftarrow a/b$
    $r \leftarrow a \mod b$
    $a \leftarrow b$
    $b \leftarrow r$
  end while

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．

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

楕円曲線の加算・2 倍算を利用して，点 P と自然数 d に対して，点 P を d 回足す演算を定義できる．この演算を楕円曲線の *scalar* 倍算という．

以上のことを基礎学習として学んだ．その後，理論班とプログラム班に分かれて活動を行った．以下にそれぞれの班の活動内容を示す．

3.2 理論班

理論班では主に以下の理論を検討し，プログラム班が実装できるよう準備を整えた．

- Baby-step Giant-step
- 素数ペアリング
- Montgomery 曲線を用いた ECM
- Pollard's ρ 法
- Brent のアルゴリズム

このうち，特に速度向上に貢献した Baby-step Giant-step について説明する．

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

$$s = av + u \quad \left(-\frac{a}{2} < u < \frac{a}{2}\right) \tag{1}$$

と変形する．すると

$$\begin{aligned} s \times Q &\equiv O \\ (av + u) \times Q &\equiv O \\ (av) \times Q &= -u \times Q \end{aligned} \tag{2}$$

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

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

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) (2) 式の左辺の x 座標 G_x, (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．

3.3 プログラム班

プログラム班では，昨年度の ECM プログラムに理論班から引き継いだ新たなアルゴリズムを実装した．また，プログラムの改善を行うなどプログラム全体の効率化を行った．主な活動内容を以下に示す．

- Baby-step Giant-step の実装
- 素数ペアリングの実装
- Montgomery 曲線を用いた ECM の実装
- 素数テーブルの実装
- VPS の導入
- ソースファイルのドキュメント作成
- テスト用のスクリプトの作成

- ソースコードの階層化

このうち，素数テーブルの実装について説明する．

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．この関数は内部で毎回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．メモリ上には配列として素数が 2,3,5,7,11,13,... と配置されており，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．

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．

3.4 活動結果

プログラムの計測結果

プログラム班がプログラムのテストを行った結果，図 1,2 の結果が得られた．

図 1,2 の結果から，昨年度のプログラムに比べて，前期終了時のプログラムの Stage2 の計算速度は 6～7 倍に向上した．また，後期終了時のプログラムの Stage1 と Stage2 を合わせた計算速度は 80 倍向上した．

STUDIO KAMADA へのランクイン

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 [1]．2 つめの合成数は $\frac{76\times10^{247}-31}{9}$ である [2]．それぞれ約 9 日，2 日プログラムを動かして素因数を発見した．

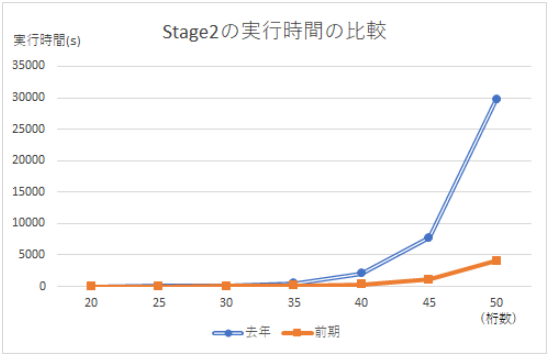

図 1　Stage2 の実行にかかった時間

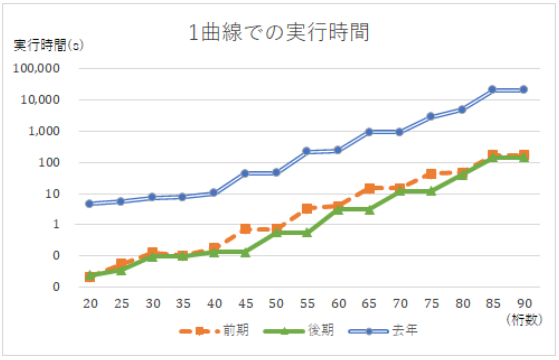

図 2　1 つの楕円曲線における実行時間

4 今後の課題

今後の課題として，以下が挙げられる．

- Pollard's ρ 法の実装
- FFT の実装
- 長期間（数か月）プログラムを動作させる
- 安定した実行環境の構築

参考文献

[1] STUDIO KAMADA. http://stdkmd.com/nrr/c.cgi?q=57779_246,（最終アクセス 2018 年 1 月 19 日）

[2] STUDIO KAMADA. http://stdkmd.com/nrr/c.cgi?q=84441_247,（最終アクセス 2018 年 1 月 19 日）