

実験報告書

実験演習記録			判定・指示		
	年月日時	共同作業者			
1	2023 年 04 月 17 日				
2					
3					
4					
レポート提出記録					
	提出年月日	期限年月日			
初	2023 年 04 月 30 日	2023 年 04 月 30 日			
再					
科目名		テーマ指導教員	学年	学期	単位
知能情報システム工学実験 2A		渡辺先生	3	前期	2
テーマ番号・テーマ名		学籍番号	名前		
RSA 暗号		21266002	赤坂 颯泰		

## 1 目的

代表的な公開鍵暗号である RSA 暗号について学び, 演習課題 18 を通して RSA 暗号及び素数判定の実装を行うことを目的とする.

## 2 実験環境

演習課題の実装にあたり, 言語は C++ を用いた. `#include <bits/stdc++.h>` を行頭に記述することで, 標準ライブラリを全て読み込んだ. また, 一部 C++17 移行でサポートされている機能を使用したため, `g++ <src-name>.cpp -std=gnu++17` を実行することでコンパイルした. なお, 本実験で作成したプログラムは全て, 演習課題毎に付録として末尾に掲載した.

## 3 演習課題 1 ユークリッドの互除法

ユークリッドの互除法とは, 2 つの自然数  $a, b$  の最大公約数  $\gcd(a, b)$  を時間計算量  $O(\log N)$  で解くことができるアルゴリズムである.

### 3.1 演習課題 1 の目的

ユークリッドの互除法をプログラムで実装することを目的とした.

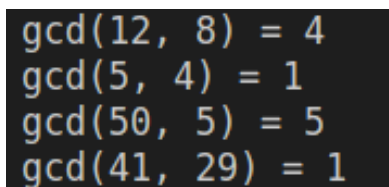
### 3.2 ユークリッドの互除法の設計

自然数  $a, b (a > b)$  が与えられた時,  $\gcd(a, b) = \gcd(b, a \% b)$  が成り立つことを利用して, 再帰関数を用いて実装した. 余りが 0 になった時点での余りを  $a_i$  とすると,  $\gcd(a, b) = a_i$  を返り値として終了とした. なお, 入力の  $a, b$  について, 素数と合成数の組み合わせを  $a > b$  という条件下で全通り試すために

- (合成数, 合成数) = (12, 8)
- (素数, 合成数) = (5, 4)
- (合成数, 素数) = (50, 5)
- (素数, 素数) = (41, 29)

以上の四通りを採用した.

### 3.3 ユークリッドの互除法の実行結果



```
gcd(12, 8) = 4
gcd(5, 4) = 1
gcd(50, 5) = 5
gcd(41, 29) = 1
```

図1: ユークリッドの互除法の出力結果

図 1 より, 正しくユークリッドの互除法により, 入力値に対する最大公約数を出力できていることが確かめられた.

## 4 演習課題 2 拡張ユークリッド

拡張ユークリッドとは,  $\gcd(a, b) = \alpha a + \beta b$  を満たす整数の組  $(\alpha, \beta)$  を求めるアルゴリズムである.

### 4.1 演習課題 2 の目的

拡張ユークリッドアルゴリズムを用い, 乗法逆元を求めるプログラムの作成を目的とする.

### 4.2 プログラムの設計

引数に 2 つの整数  $a, b (a > b)$  を与えた, 法を  $N$  とした時,  $ay \equiv 1 \pmod{N}$  となる乗法逆元  $y$  を求める関数 `modinv` を作成した. `modinv` 関数では, さらに内部で拡張ユークリッドアルゴリズムを計算する `calcExtendedEuclid` 関数を実装した. `calcExtendedEuclid` 関数は, 引数に整数  $a, m, x, y$  をとった. それぞれ  $ax+by=1$  を満たす整数であり,  $\gcd(a, b)$  を返り値としながら,  $ax+by=1$  を満たす  $x, y$  を計算した. これらの処理について, 演習課題 1 の `calcEuclid` 関数と同様に再帰的に計算を行った. `calcExtendedEuclid` 関数で計算された  $x$  に対して,  $m$  で割ったときの余りを正で返す `mod` 関数を実装した. この余りを `modinv` 関数の返り値とした. なお, 入力  $a, b$  について, 演習課題 1 と同様に

- (合成数, 合成数) = (12, 8)
- (素数, 合成数) = (5, 4)
- (合成数, 素数) = (50, 5)
- (素数, 素数) = (41, 29)

以上の四通りを採用し,  $a, b, x, y$  および逆元を出力した.

### 4.3 拡張ユークリッドの実行結果

```
(a, b, x, y) = (12, 8, 1, -1)      Inverse modulo: 1
(a, b, x, y) = (5, 4, 1, -1)      Inverse modulo: 1
(a, b, x, y) = (50, 5, 0, 1)      Inverse modulo: 0
(a, b, x, y) = (41, 29, -12, 17)  Inverse modulo: 17
```

図2: 拡張ユークリッドの出力結果

図 2 の結果について順に  $ax+by$  を計算すると,  $12-8=4=\gcd(12, 8)$ ,  $5-4=1=\gcd(5, 4)$ ,  $0+5=5=\gcd(50, 5)$ ,  $41 \times (-12) + 29 \times 17 = 1 = \gcd(41, 29)$  となった.

また,  $b$  を法とした時の  $a * x \equiv 1 \pmod{b}$  を満たす整数  $x$  は, 順に 1, 1, 0, 17 であった.

### 4.4 拡張ユークリッドの考察

$ax+by$  の計算結果が, 全て演習課 1 で求めた入力値に対する最大公約数に等しくなったことから, 正しくユークリッドの拡張による演算が行われていることがわかった. 一方で, 実行結果の逆元は 1, 1, 0, 17 であり, 理論的に  $a * y \equiv 1 \pmod{b}$  を満たす  $y$  を順に求めると, 存在しない, 1, 存在しない, 17 となる. このように結果が異なった理由は, 逆元が必ずしも存在する訳ではないためだと考える. 実際に  $a * y \equiv 1 \pmod{b}$  を満たす  $y$  が存在するための必要十分条件は  $\gcd(b, a) = 1$  であることが知られている. したがって, 理論的には逆元が存在し得ない組み合わせにおいて得られた実験結果は, 誤りだったといえる. よって, 乗法逆元を求めることができる入力値は, 互いに素な自然数の組み合わせに限ると考えられる.

## 5 演習課題 3 RSA 暗号による暗号化および復号化

### 5.1 演習課題 3 の目的

素数  $p, q$  と  $e$  ならびにメッセージ  $m \in 1, \dots, N$  が与えられた時, RSA 暗号による暗号化並びに復号化を行うプログラムの実装を目的とした.

### 5.2 プログラムの設計

まず, 整数  $p, q$  については標準入力で受け取った.  $n$  は  $p$  と  $q$  の積で求めた.  $e$  は, 第 2 節で実装した `calcEuclid` 関数の第一引数に  $(p-1)*(q-1)$ , 第二引数に  $e$  を渡し, 結果が 1 になるまで繰り返したときの試行回数を  $e$  とした.  $d$  は, 第 2 節で実装した `modinv` 関数の第一引数に  $e$ , 第二引数に  $(p-1)*(q-1)$  を渡した返り値とした. その後, RSA 暗号による暗号化を行う `encryptor` 関数を実装した. この関数は第一引数に  $e$ , 第二引数に  $n$  を渡した. この関数内では, 自然数  $m$  の初期値を 0 とし,  $m$  の  $e$  乗を  $n$  で割った余りを求める関数 `power` を計算した. この結果を  $c$  とすると,  $c$  が暗号化された値に相当する.  $m$  をインクリメントし, 繰り返し自乗法を使った法  $n$  のべき乗計算を行う `power` 関数によって  $n$  回暗号化を行った. 暗号化の結果を `angou` という配列に格納した. 次に, RSA 暗号の復号化を行う `decryptor` 関数を作成した. この関数は, 第一引数に  $d$ , 第二引数に  $n$  をとった. さらに, `angou` の各要素を範囲 `for` 文を用いて  $c$  という変数名で順に取得し, `power` 関数を用いて  $c$  の  $d$  乗を  $n$  で割った余りを計算することで復号化した. この復号化の結果を `hirabun` という配列に格納した. 元のメッセージ  $m \in 1, \dots, N$  と暗号化した配列 `angou`, および `angou` を復号化した `hirabun` を出力し, 元のメッセージが `hirabun` と一致しているかどうかを判定した.

### 5.3 プログラムの実行結果及び考察

RSA 暗号による暗号化と復号化の結果を以下に示す.

```
p : 13
q : 7
N : 91
e : 5
d : 29
angou : 013261233141638818272381314717475448076212943351522784228852243442
43461265666353618543773555685252679454849576789863697643940588762701511471
61720777853199108328506068305990

hirabun : 012345678910111213141516171819202122232425262728293031323334353637
38394041424344454647484950515253545556575859606162636465666768697071727374
75767778798081828384858687888990
```

図3: RSA 暗号による暗号化と復号化

`hirabun` は 0  $N$  までの値が順に並んだ数字列である. これを暗号化したところ図 3 の `angou` が得られた. さらに平文に復号化したところ `hirabun` が得られた.

## 5.4 RSA 暗号による暗号化と復号化の考察

正しく `angou` が `hirabun` に変換されていることを確かめるために, `for` 文で 0 から `N` までの値を順にファイル出力し, `hirabun` と一致しているかを判定した. 0 から `N` までの値を `hirabun_1.txt` に, `hirabun` の内容を `hirabun_2.txt` にファイル出力し, `diff -y` コマンドによって差分を出力した. その結果を以下に示す.

```
diff hirabun_1.txt hirabun_2.txt -y
0123456789101112131415161718192021222324252627282930313233343
0123456789101112131415161718192021222324252627282930313233343
```

図4: `diff -y` コマンドによる差分表示

図 4 より, 暗号化以前の `hirabun` に対して差分は見られなかった. よって, `hirabun` を正しく暗号化し, 復号化することができたと言える.

## 6 演習課題 4 総当たりによる素数判定法

### 6.1 演習課題 4 の目的

総当たりによる素数判定法をプログラムで実装し, ビット長 `n` を大きくしていったときの実行時間について調べることを目的とした.

### 6.2 プログラムの設計

総当たりによる素数判定のアルゴリズムについて述べる. 素数の候補を `p` とすると, `p` が  $k = 2, \dots, \lfloor \sqrt{p} \rfloor$  に対して順に割り切れるかどうかを調べた. すなわち,

$$\forall k \in [2, \lfloor \sqrt{p} \rfloor], \quad p \not\equiv 0 \pmod{k} \Rightarrow isPrime$$

上式を実装した. ここでは  $p = 2, \dots, 10^5$  とし, そのうち素数と判定するまでの繰り返し回数を記録し出力した.

### 6.3 プログラムの実行結果

総当たりによる素数判定の結果を以下に示す. 横軸を素数, 縦軸を反復回数の対数とした.

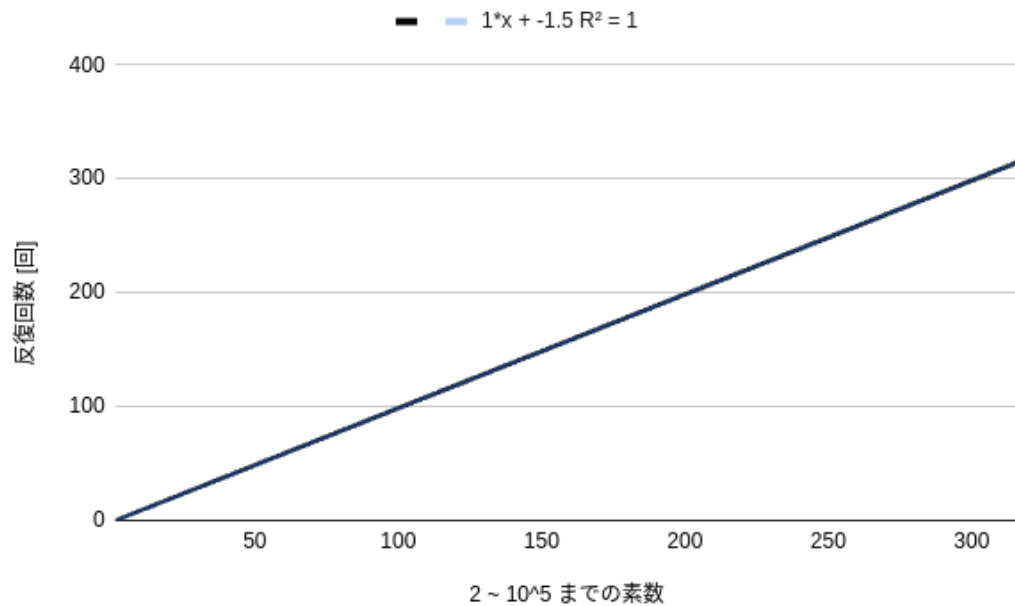


図5: 素数の検出にかかる反復回数の出力結果

上図の反復回数に対する片対数グラフの回帰直線は、傾きは1であり素数の増大に伴い反復関数も比例する関係性が得られた。また、 $R^2 = 1$ であったことから、この関係性には非常に強い正の相関があると言える。

#### 6.4 総当りにおける素数判定法の考察

素数を判定するには  $\sqrt{p}$  回だけ試行するため、その時間計算量は  $O(\sqrt{p})$  となると考えられる。そのために、片対数を取った場合に図5のように比例した結果が得られたといえる。このように、反復回数が素数の候補である数の長さに比例して指数的に増加してしまう。したがって、総当りにおける素数判定法は、効率的な素数判定法であるとはいえない。

### 7 演習課題 5 フェルマーテスト

フェルマーテストは、フェルマーの小定理を利用した乱択アルゴリズムに分類される素数判定法である。フェルマーの小定理とは、

フェルマーの小定理:

$p > 0$  の素数に対して  $1 \leq a \leq p-1$  とすると、

$$a^p \equiv a \pmod{p}$$

上のような関係式が成り立つ定理である。この式を変形すると

$$a^{p-1} \equiv 1 \pmod{p}$$

となる。さらに対偶をとれば、

$$a^{p-1} \not\equiv 1 \pmod{p}$$

が成り立つならば、 $p$  は素数ではないと言える。また、乱択アルゴリズム (Randomized Algorithm) とは、乱数によって動作が決定されるアルゴリズムである。したがって、必ずしも正しい判定が行われるとは限らない。一方で、決定的な手続きに即したアルゴリズムと比較して高速に処理することが可能である。また、カーマイケル数とは、合成数のうち  $a^{n-1} \equiv 1 \pmod{n}$  を満たす正の整数  $n$  のことである。

## 7.1 演習課題 5 の目的

フェルマーテストをプログラムで実装し、カーマイケル数以外で誤判定確率を評価することを目的とした。

## 7.2 フェルマーテストの設計

フェルマーテストのアルゴリズムについて述べる。

フェルマーテスト:

- 入力: 素数の候補  $p$ , 最大繰り返し回数  $S$
- 出力:  $p$  が素数である確率 [%]

とし,  $1 \leq i \leq S$  の間以下の処理 1~3 を行った。

1.  $a \in \{1, 2, \dots, p-1\}$  をランダムに選択した。
2.  $a^{p-1} \not\equiv 1 \pmod{p} \Rightarrow \text{isNotPrime}$  Otherwise  $\Rightarrow \text{isPrime}$  として各回数を記録
3.  $i$  をインクリメントし, ステップ 1 に戻る。

なお, 乱数の生成には C++ 標準ライブラリで提供される乱数生成器 `std::mt19937` を用いた。

## 8 フェルマーテストの実験結果

最も小さいカーマイケル数は 561 であるため, 561 を含まない 1 500 までの数に対して素数になる確率を下図に示した。

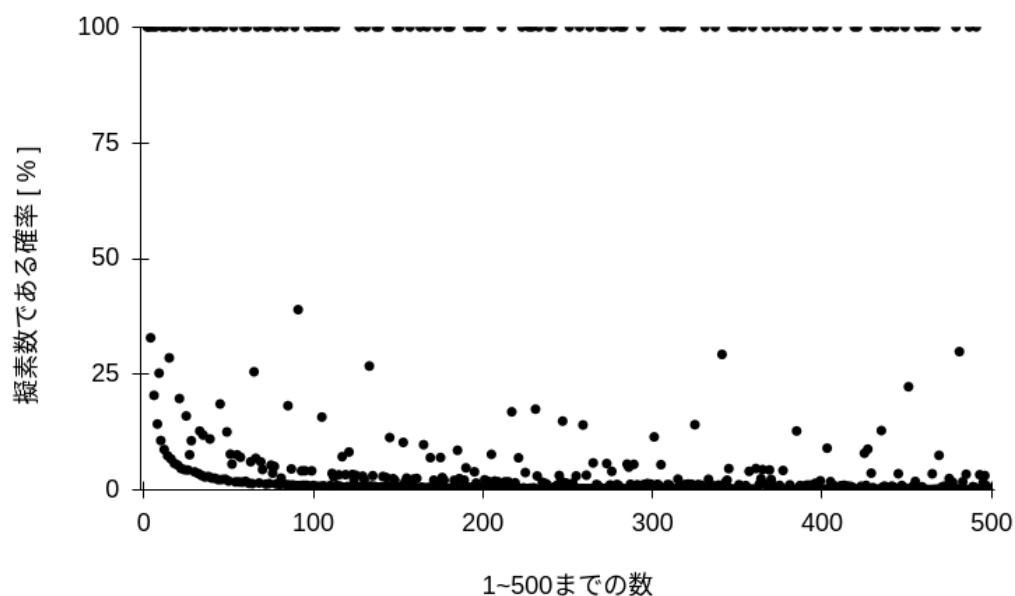


図6: 1 500 に対するフェルマーテストの結果

図 5 より, 素数は 100% の確率で素数だと判定され, 合成数の素数と判定される確率は全て 50% 以下であった。合成数の素数判定確率が最も高かったものは 91 の 38.98% であった。したがって, カーマイケル数以外の合成数に対する素数判定は, 50% を閾値とすると正常に機能しているといえる。

## 8.1 フェルマーテストの考察

カーマイケル数におけるフェルマーテストの性能について考察する。カーマイケル数は小さい順に 561, 1105, 1729, 2465, 2821, 6601, 8911 であり、これらに対してフェルマーテストによって素数と判定される確率を調べた。その結果を下図に示す。図 7 より、どのカーマイケル数に対しても 50% 以上の

```
Probability that 561 is a prime number: 57.14%
Probability that 1105 is a prime number: 70.08%
Probability that 1729 is a prime number: 74.54%
Probability that 2465 is a prime number: 72.84%
Probability that 2821 is a prime number: 76.44%
Probability that 6601 is a prime number: 79.98%
Probability that 8911 is a prime number: 80.67%
```

図7: カーマイケルに対するフェルマーテストの結果

確率で素数だと判定されていることがわかる。何故このような結果が得られたのかについて述べる。まず、フェルマーテストはフェルマーの小定理より、 $a$  と  $n$  を互いに素な自然数とすると、

$$a^{p-1} \equiv 1 \pmod{p}$$

を満たす場合、 $p$  を素数であると判定する。ここで、全てのカーマイケル数は合成数でありながら、上式を満たす互いに素な自然数  $a$  をもつ。したがって、フェルマーテストを通過すると考えられる。そのため、カーマイケル数に対して素数の誤判定を防ぐためには、フェルマーテストよりも条件が厳しい素数判定法を用いる必要があるといえる。

## 9 演習課題 6 Miller-Rabin テスト

演習課題 5 で実装したフェルマーテストではカーマイケル数に対しては正しく機能しない。この欠点を改善したアルゴリズムが Miller-Rabin テストである。このテストの根幹には次の性質がある。

補題:  $p$  が素数であるとき、自然数  $a$  が  $a^2 \equiv 1 \pmod{p}$  を満たすならば、 $a \equiv \pm 1 \pmod{p}$  が成り立つ。

補題の対偶:  $a^2 \equiv 1 \pmod{p}$   $a \not\equiv \pm 1 \pmod{p}$  を満たす  $a$  が存在するならば、 $p$  は素数ではない。

上記の補題の対偶を利用して Miller-Rabin を実装することが可能である。

### 9.1 演習課題 6 の目的

Miller-Rabin テストをプログラムで実装し、カーマイケル数に対して誤判定確率を評価することを目的とした。

### 9.2 Miller-Rabin テストの設計

Miller-Rabin テストのアルゴリズムについて述べる。



Miller-Rabin テスト:

- 入力: 素数の候補  $p$ , 最大繰り返し回数  $S$
- 出力:  $p$  が素数である確率 [%]

とし,  $1 \leq i \leq S$  の間以下の処理 1~3 を行った.

1.  $a \in \{1, 2, \dots, p-1\}$  をランダムに選択.
2. if  $a^{p-1} \not\equiv 1 \pmod{p} \Rightarrow \text{isNotPrime}$   
else if  $j = 1, \dots, u$  に対して,  $a^{2^j v} \equiv 1 \pmod{p} \wedge a^{2^{j-1} v} \not\equiv \pm 1 \pmod{p}$  となる  $j$  が存在.  $\Rightarrow \text{isNotPrime}$   
Otherwise  $\Rightarrow \text{isPrime}$  として各回数を記録
3.  $i=S$  ならば終了. そうでなければ,  $i$  をインクリメントし, ステップ 1 に戻る.

なお, 乱数の生成には C++ 標準ライブラリで提供される乱数生成器 `std::mt19937` を用いた. また本実験では  $s = 10^5$  とした.

### 9.3 Miller-Rabin テストの実験結果

フェルマーテストで扱ったカーマイケル数に対して Miller-Rabin テストを実行した場合の結果を以下に示す.

```
Probability that 561 is a prime number : 1.797%
Probability that 1105 is a prime number : 2.746%
Probability that 1729 is a prime number : 9.363%
Probability that 2465 is a prime number : 2.882%
Probability that 2821 is a prime number : 9.721%
Probability that 6601 is a prime number : 4.993%
Probability that 8911 is a prime number : 19.906%
```

図8: カーマイケル数に対する Miller-Rabin テストの結果

フェルマーテストでは素数だと誤判定されていたカーマイケル数は, 素数と判定される確率が全て 20% 以下となった. 50% を境界とすれば, 全てのカーマイケル数に対して正しく合成数と判定できているといえる.

### 9.4 Miller-Rabin テストの考察

Miller-Rabin テストではカーマイケル数に対して有効である理由について述べる. カーマイケル数は, 前述の通りある自然数  $n$  を素因数分解したときに,  $(n-1)$  を因数に持ち, かつ全ての素因数  $p$  について  $p-1$  が  $(n-1)$  の約数であるような合成数である. この性質から, カーマイケル数  $n$  を素数と誤判定するためには,  $n$  のある素因数  $p$  に対して, Miller-Rabin テストによって  $p$  を素数と判定する必要がある. ところが, カーマイケル数にはこのような素因数  $p$  が複数存在する. よって, 全ての素因数が Miller-Rabin テストを通過する必要がある. しかし, その確率は非常に低いため, カーマイケル数に対して Miller-Rabin は有効であると考えられる.

## 10 演習課題 7 素数生成

### 10.1 演習課題 7 の目的

素数生成アルゴリズムに演習課題 6 で作成した Miller-Rabin テストを組み合わせ, 素数をランダムに生成するプログラムを実装することを目的とした.

## 10.2 素数生成器の設計

本実験では素数をランダムに生成するアルゴリズムを採用した。以下にその具体的な処理について述べる。

素数生成器:

- 入力: 生成したい素数の長さ  $n$ (ビット), 最大繰り返し回数  $K$
- 出力: 素数か否かの bool 値初期値は *False*

とし,  $1 \leq i \leq K$  の間以下の処理 1~3 を行った。

1.  $(n - 1)$  ビットの整数  $p'$  を  $\{0, 1\}^{n-1}$  からランダムに選択。
2. 素数の候補を  $p = 1||p'$  ( $p'$  の先頭に 1 を付加した  $n$  ビットの整数) とした。
3. Miller-Rabin テストにより,  $p$  が素数であれば *True* を返し終了。そうでなければ  $i$  をインクリメントし, ステップ 1 に戻る。

本実験では, 入力したい素数の長さは 16 ビットとし, 最大繰り返し数を 100 回とした。

## 10.3 素数生成器の実験結果

実験結果を下図に示す。

```
3 is Prime Number
7 is Prime Number
11 is Prime Number
19 is Prime Number
47 is Prime Number
73 is Prime Number
179 is Prime Number
331 is Prime Number
571 is Prime Number
1181 is Prime Number
2459 is Prime Number
5783 is Prime Number
9871 is Prime Number
17581 is Prime Number
38651 is Prime Number
```

```
3 is Prime Number
5 is Prime Number
13 is Prime Number
17 is Prime Number
37 is Prime Number
97 is Prime Number
193 is Prime Number
353 is Prime Number
607 is Prime Number
1451 is Prime Number
2339 is Prime Number
5563 is Prime Number
9857 is Prime Number
23609 is Prime Number
45077 is Prime Number
```

```
3 is Prime Number
5 is Prime Number
13 is Prime Number
17 is Prime Number
37 is Prime Number
79 is Prime Number
149 is Prime Number
337 is Prime Number
719 is Prime Number
1279 is Prime Number
2729 is Prime Number
5981 is Prime Number
10909 is Prime Number
22039 is Prime Number
34781 is Prime Number
```

```
3 is Prime Number
5 is Prime Number
11 is Prime Number
17 is Prime Number
47 is Prime Number
83 is Prime Number
151 is Prime Number
313 is Prime Number
743 is Prime Number
1289 is Prime Number
2659 is Prime Number
4421 is Prime Number
9511 is Prime Number
21187 is Prime Number
44953 is Prime Number
```

図9: 素数生成器の実行結果

素数生成器プログラムを 4 回実行した結果,  $n=15$  個の素数がそれぞれ生成された.

## 11 素数生成器の考察

図 9 より, 素数のビット数が多くなるにつれて, 多様な素数を生成することが可能になると考えられる. 例えば  $n=2$  の場合は 2 ビットで表せる素数は 3 のみである. よって, 4 回の試行回数の中の結果でも 2 ビットの素数の出力結果は 3 で共通であった. ところが,  $n=16$  ビットで表せる素数は,  $2^{15}2^{16} - 1$  の数の範囲にある. 故に, 図 9 で生成された数値は順に, 38651, 45077, 34781, 44953 と全て異なる素数になったといえる. 一般化すれば,  $2^{n-1}2^n - 1$  までの範囲にある数値からランダムに素数を生成することができる. したがって, 素数のビット数の増加に伴い, 生成可能な素数も増加するといえる.

## 12 演習課題 8 RSA 暗号システム

### 12.1 演習課題 8 の目的

演習課題 7 の素数生成アルゴリズムと演習課題 3 の暗号化/復号化アルゴリズムを組み合わせ、RSA 暗号システムを実装することを目的とした。

### 12.2 RSA 暗号システムの設計

RSA 暗号システムを実現する関数 calcRSASystem の実装について述べる。calcRSASystem には 2 つの素数を入力値として与える必要があるため、事前に演習課題 7 で作成した generatePrime 関数を用いて素数  $p, q$  を求めた。ただし、 $p, q$  は互いに異なる素数である必要があるため、 $q$  が  $p$  と一致しないという条件を満たすまで、generatePrime 関数を実行した。その後、 $p, q$  を calcRSASystem 関数に与えた。具体的なアルゴリズムについて以下に示す。

calcRSASystem:

- 入力: 生成する素数のビット  $n$ , 最大繰り返し回数  $s$ , 素数の候補  $p$ , 素数の候補  $q$
- 出力: なし

とした。

1.  $p, q$  の積  $n$  を求めた。
2.  $e$  を演習課題 2 で作成した calcE 関数を用いて求めた。
3.  $e, (p-1) * (q-1)$  に対して、演習課題 2 で実装した modinv 関数を用いて乗法逆元  $d$  を求めた。
4.  $e, n$  を用いて演習課題 3 で実装した暗号化の関数 encryptor を用いて暗号を生成した。
5.  $d, n$  を用いて演習課題 3 で実装した復号化の関数 decryptor を用いて暗号を復号化した。
6. 暗号化したメッセージ angou と、angou を復号化したメッセージ hirabun を出力した。

## 13 RSA 暗号システムの実験結果

calcRSASystem 関数の実行結果を以下に示す。

```
p: 13, q: 11
N: 143
E: 7
d: 103
angou: 0, 1, 128, 42, 82, 47, 85, 6, 57, 48, 10, 132, 12, 117, 53, 115, 3, 30, 138, 46, 136,
109, 22, 23, 106, 64, 104, 14, 63, 94, 134, 125, 98, 110, 122, 139, 75, 93, 25, 52, 105, 24,
81, 43, 99, 111, 84, 31, 126, 36, 41, 116, 13, 92, 76, 55, 56, 73, 20, 71, 135, 74, 127, 2,
103, 65, 66, 89, 29, 108, 60, 124, 19, 83, 35, 114, 54, 77, 78, 40, 141, 16, 69, 8, 72, 123,
70, 87, 88, 67, 51, 130, 27, 102, 107, 17, 112, 59, 32, 44, 100, 62, 119, 38, 91, 118, 50, 68,
4, 21, 33, 45, 18, 9, 49, 80, 129, 39, 79, 37, 120, 121, 34, 7, 97, 5, 113, 140, 28, 90, 26,
131, 11, 133, 95, 86, 137, 58, 96, 61, 101, 15, 142,
hirabun: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95,
96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114,
115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
```

図10: RSA 暗号システムの実行結果

図 10 より, 出力された hirabun は暗号化前のメッセージである 0 から 142 までの整数と一致したことがわかった.

## 14 RSA 暗号システムの考察

calcRSA 暗号システムの実行により, 素数の生成及び, 素数を用いたメッセージの暗号化, 復号化ができたといえる. 本実験では入力する素数が 13, 11 と二桁の素数であった. だが, 巨大な素数を用いた場合, 公開鍵の長さが増えるので, 暗号文を解読するための時間計算量は増加するといえる. よって, 素数が非常に大きい場合に, RSA 暗号の暗号化や復号化の処理時間が長くなり実用性が低下するが, その分安全性は増加するというトレードオフの関係が内在すると考えられる.

## 15 おわりに

代表的な公開鍵暗号である RSA 暗号について学び, RSA 暗号及び素数判定の実装を行うことができた.

## 参考文献

公開鍵暗号入門, 渡辺峻, <https://classroom.google.com/c/NTE3NTYyOTMxNjc1/m/NTE4NDU1NTU0MzQ2/details,p.2-1> 2-13, (2023 年 4 月 28 日参照)

## 付録

Listing 1: 演習課題 1 のソースコード

```

1      #include <bits/stdc++.h>
2
3      using namespace std;
4      using ll = long long;
5      #define rep(i, a, b) for (int i = a; i < b; i++)
6
7
8
9      using namespace std;
10
11     int calcEuclid(int a, int b){
12         if(b==0){
13             return a;
14         }
15         return calcEuclid(b, a%b);
16     }
17
18     int main(){
19
20         vector<pair<int, int>> test =
21         {{12, 8}, {5,4}, {50, 5}, {41, 29}};
22         for(const auto [a, b] : test){
23             cout << "gcd(" << a << ", " << b << ") = "
24             << calcEuclid(a, b) << endl;
25         }
26
27     }

```

Listing 2: 演習課題 2 のソースコード

```

1      #include <bits/stdc++.h>
2
3      using namespace std;
4      using ll = long long;
5      #define rep(i, a, b) for (int i = a; i < b; i++)
6
7
8
9      using namespace std;
10
11
12     //拡張ユークリッド関数
13     int calcExtendedEuclid(int a, int b, int &x, int &y){
14         if(b==0){
15             x = 1;
16             y = 0;

```

```

17         return a;
18     }
19     int d = calcExtendedEuclid(b, a%b, y, x);
20     y -= a/b*x;
21     return d;
22 }
23
24 //aをmで割った正の余りを計算
25 int mod (int a, int m){
26     return (a%m + m) %m;
27 }
28
29 //逆元を計算
30 int modinv(int a, int m){
31     int x, y;
32     calcExtendedEuclid(a, m, x, y);
33     cout << "(a, b, x, y) = ("
34     << a << ", " << m << ", " << x << ", " << y;
35     if(y==17) cout << ")\t" ;
36     else cout << ")\t\t" ;
37     cout << "Inverse modulo: ";
38     return mod(x, m);
39 }
40
41 int main(){
42
43     //5y=1(mod. 13) -> 5*8-13*3=1
44     // -> 5y+13x=1を満たすyがmodにおける乗法の逆元(ここではy=8)
45     // int a = 5, b = 13;
46     // int x, y;
47
48     vector<pair<int, int>> test
49     = {{12, 8}, {5,4}, {50, 5}, {41, 29}};
50     for(const auto [a, b] : test){
51         cout << modinv(a, b) << endl;
52     }
53 }

```

Listing 3: 演習課題3のソースコード

```

1     #include <bits/stdc++.h>
2
3     using namespace std;
4     using ll = long long;
5     #define rep(i, a, b) for (int i = a; i < b; i++)
6

```

```

7 long long int calcExtendedEuclid(long long int a, long long int b,
8                                 long long int &x, long long int &y){
9     if(b==0){
10         x = 1;
11         y = 0;
12         return a;
13     }
14     long long int d = calcExtendedEuclid(b, a%b, y, x);
15     y -= a/b*x;
16     return d;
17 }
18
19 long long int mod (int a, int m){
20     return (a%m + m) % m;
21 }
22
23 long long int modinv(long long int a, long long int m){
24     long long int x, y;
25     calcExtendedEuclid(a, m, x, y);
26     return mod(x, m);
27 }
28
29
30 long long int calcEuclid(long long int a, long long int b){
31     if(b==0){
32         return a;
33     }
34     return calcEuclid(b, a%b);
35 }
36
37 long long int calcE(long long int p, long long int q){
38     long long int e = 2;
39     while(calcEuclid((p-1)*(q-1), e) !=1){
40         e++;
41     }
42     return e;
43 }
44
45 /*
46  * 繰り返し自乗法を使った法  $n$  のべき乗計算
47  *  (  $a$  の  $k$  乗を  $n$  で割った余りを求める )
48  *
49  * long long int a : 底
50  * long long int k : 指数
51  * long long int n : 法
52 */

```



```

53 long long int power(long long int a, long long int k,
54                     long long int n) {
55
56     a %= n;
57
58     if(a == 0 || n == 0){
59         return 0;
60     }
61     if(k == 0){
62         return 1 % n;
63     }
64
65     int i;
66     long long int value = 1;
67     for(i = 0; i < k; i++) {
68         value *= a;
69         if(value >= n) {
70             value %= n;
71         }
72     }
73     return value;
74 }
75
76
77 vector<int> hirabun, angou;
78
79 void encryptor(int e, int n){
80     for(int m=0; m<n; m++){
81         int c = power(m, e, n);
82         angou.emplace_back(c);
83     }
84 }
85
86 void decryptor(int d, int n){
87     for(const auto &c: angou){
88         int m = power(c, d, n);
89         hirabun.emplace_back(m);
90     }
91 }
92
93
94 int main(){
95     int p = 13, q = 7;
96     int n = p*q;
97     int e = calcE(p,q);
98

```

```

99     cout << "p: " << p << endl;
100    cout << "q: " << q << endl;
101    cout << "N: " << n << endl;
102    cout << "e: " << e << endl;
103
104    int d = modinv(e, (p-1)*(q-1));
105    cout << "d: " << d << endl;
106
107    encryptor(e, n);
108    decryptor(d, n);
109    cout << "angou:\t";
110    for(const auto &v : angou){
111        cout << v;
112    } cout << endl;
113
114    std::ofstream hirabun_file_1, hirabun_file_2;
115    std::string filename_1 = "hirabun_1.txt";
116    hirabun_file_1.open(filename_1, std::ios::out);
117
118    cout << "hirabun:\t";
119    for(const auto &v : hirabun){
120        cout << v;
121        hirabun_file_1 << v;
122    } cout << endl;
123
124    std::string filename_2 = "hirabun_2.txt";
125    hirabun_file_2.open(filename_2, std::ios::out);
126
127    for(int i=0; i<n; i++){
128        cout << i;
129        hirabun_file_2 << i;
130    } cout << endl;
131
132
133
134
135    int original = 2;
136    int a = power(original, e, n);
137    int b = power(a, d, n);
138    cout << original <<"->"<<a << "->"<<b<<endl;
139
140 }

```

Listing 4: 演習課題 4 のソースコード

```

1 #include <bits/stdc++.h>

```

```

2  #include <fstream>
3
4  using std::ofstream;
5  using namespace std;
6
7  ofstream ofs("prime.csv"); // ファイルパスを指定する
8
9  int main(){
10     for(int i=2; i<1e5; i++){
11         int cnt = 0;
12         bool isPrime = true;
13         for(int k=2; k<=sqrt(i); k++){
14             cnt++;
15             if(i%k==0) {
16                 cout << i << " is not prime number" << endl;
17                 isPrime = false;
18                 break;
19             }
20         }
21         if(isPrime) ofs << i << ", " << cnt << endl;
22     }
23
24 }

```

Listing 5: 演習課題 5 のソースコード

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  long long int mod (int a, int m){
6      return (a%m + m) % m;
7  }
8  /*
9   * 繰り返し自乗法を使った法  $n$  のべき乗計算
10   * ( $a$  の  $k$  乗を  $n$  で割った余りを求める)
11   *
12   * unsigned long int a : 底
13   * unsigned int k : 指数
14   * unsigned int n : 法
15   */
16  unsigned int power(unsigned int a, unsigned int k,
17                     unsigned int n) {
18
19      a %= n;
20

```

```

21     if(a == 0 || n == 0){
22         return 0;
23     }
24     if(k == 0){
25         return 1 % n;
26     }
27
28     int i;
29     unsigned int value = 1;
30     for(i = 0; i < k; i++) {
31         value *= a;
32         if(value >= n) {
33             value %= n;
34         }
35     }
36     return value;
37 }
38
39 std::random_device rd;
40 std::mt19937 gen(rd());
41
42 int random(int low, int high) {
43     std::uniform_int_distribution<> dist(low, high);
44     return dist(gen);
45 }
46
47 using std::ofstream;
48
49 ofstream ofs("felmerTest.csv"); // ファイルパスを指定する
50
51
52 int main(){
53     int s = 1e4;
54     int p = 76;
55     bool isPrime = true;
56     vector<int> carmichaelNumbers = {
57         561, 1105, 1729, 2465, 2821, 6601, 8911,
58         // 7, 13, 21, 51, 23, 57
59     };
60
61     for(const auto v : carmichaelNumbers){
62         // for(int p = 2; p < 500; p++){
63             p = v;
64             int cnt = 0;
65             for(int i=0; i<s; i++){
66                 int a = random(1, p-1); //乱数生成

```

```

67         int res = power(a, p-1, p); //  $a^{(p-1)}$  の mod.p
68         // cerr << "a: " << a << " ";
69         if(res == 1){
70             cnt++;
71         }
72     }
73     // cout << cnt << endl;
74     cout << "Probability that " << p << " is a prime number: "
75     << (double(cnt)/double(s))*100 << "%" << endl;
76     ofs << p << ", " << (double(cnt)/double(s))*100 << endl;
77 }
78
79 }

```

Listing 6: 演習課題 6 のソースコード

```

1     #include <bits/stdc++.h>
2
3     using namespace std;
4
5     long long int mod (int a, int m){
6         return (a%m + m) % m;
7     }
8     /*
9     * 繰り返し自乗法を使った法  $n$  のべき乗計算
10    (  $a$  の  $k$  乗を  $n$  で割った余りを求める )
11    *
12    * unsigned long int a : 底
13    * unsigned int k : 指数
14    * unsigned int n : 法
15    */
16    unsigned int power(unsigned int a, unsigned int k,
17                        unsigned int n) {
18
19        a %= n;
20
21        if(a == 0 || n == 0){
22            return 0;
23        }
24        if(k == 0){
25            return 1 % n;
26        }
27
28        int i;
29        unsigned int value = 1;
30        for(i = 0; i < k; i++) {

```

```

31         value *= a;
32         if(value >= n) {
33             value %= n;
34         }
35     }
36     return value;
37 }
38
39 std::random_device rd;
40 std::mt19937 gen(rd());
41
42 int random(int low, int high) {
43     std::uniform_int_distribution<> dist(low, high);
44     return dist(gen);
45 }
46
47 using std::ofstream;
48
49 ofstream ofs("miller-rabin.csv"); // ファイルパスを指定する
50
51
52 int p;
53 int s = 1e5;
54 vector<int> carmichaelNumbers = {
55     561, 1105, 1729, 2465, 2821, 6601, 8911,
56     // 7, 13, 21, 51, 23, 57
57 };
58
59 void calcMillerRabin(){
60     for(const auto &p : carmichaelNumbers){
61         int u = 0, v = p - 1;
62         while (v % 2 == 0) {
63             ++u;
64             v >>= 1;
65         }
66         // cout << p-1 << ", " << u << ", " << v << endl;
67
68         int notPrime = 0;
69         int prime = 0;
70
71         for(int i=0; i<s; i++){
72             int a = random(1, p-1); //乱数生成
73             int res = power(a, p-1, p); //a^(p-1)のmod.p
74             bool isPrime = true;
75             if(res != 1){
76                 notPrime++;

```

```

77         } else if(res == 1){
78             for(int j=1; j<=u; j++){
79                 int res1 = power(a, pow(2, j)*v, p);
80                 int res2 = power(a, pow(2, j-1)*v, p);
81                 if(res1 == 1 and res2 != 1 and res2 != p-1){
82                     //p-1 == -1(mod.p)
83                     notPrime++;
84                     isPrime = false;
85                     break;
86                 }
87             }
88             if(isPrime) {
89                 prime++;
90                 isPrime = true;
91             }
92         }
93     }
94     // cout << notPrime << " + " << prime << " == " << s << endl;
95
96     cout << "Probability that "<< p <<" is a prime number : "
97     << (double(prime)/double(s))*100 << "%" << endl;
98     ofs << p << ", " << (double(prime)/double(s))*100 << endl;
99 }
100 }
101
102 int main(){
103
104     ofs << "iteration, " << s << endl;
105     calcMillerRabin();
106
107 }

```

Listing 7: 演習課題7のソースコード

```

1     #include <bits/stdc++.h>
2     #include <fstream>
3
4     using std::ofstream;
5     using namespace std;
6
7     ofstream ofs("random.csv"); // ファイルパスを指定する
8
9     long long int mod (int a, int m){
10         return (a%m + m) % m;
11     }
12     /*

```

```

13  * 繰り返し自乗法を使った法  $n$  のべき乗計算
14  (  $a$  の  $k$  乗を  $n$  で割った余りを求める )
15  *
16  * unsigned long int a : 底
17  * unsigned int k : 指数
18  * unsigned int n : 法
19  */
20  unsigned int power(unsigned int a, unsigned int k,
21                      unsigned int n) {
22
23      a %= n;
24
25      if(a == 0 || n == 0){
26          return 0;
27      }
28      if(k == 0){
29          return 1 % n;
30      }
31
32      int i;
33      unsigned int value = 1;
34      for(i = 0; i < k; i++) {
35          value *= a;
36          if(value >= n) {
37              value %= n;
38          }
39      }
40      return value;
41  }
42
43  std::random_device rd;
44  std::mt19937 gen(rd());
45
46  int random(int low, int high) {
47      std::uniform_int_distribution<> rand(low, high);
48      return rand(gen);
49  }
50
51  /**
52   * @brief ミラーラビン法による素数判定
53   * @param p 素数の候補
54   * @param s 反復回数
55   * @return true
56   * @return false
57   */
58  bool calcMillerRabin(int p, int s){

```



```

59
60 //例外処理
61 if (p == 2) {
62     return true;
63 }
64 if (p < 2 || p % 2 == 0) {
65     return false;
66 }
67
68 int u = 0, v = p - 1;
69 while (v % 2 == 0) {
70     ++u;
71     v >>= 1; // v/=2;
72 }
73
74 for(int i=0; i<s; i++){
75     int a = random(1, p-1); //乱数生成
76     int res = power(a, p-1, p); //a^(p-1)のmod.p
77     if(res != 1){
78         return false;
79     } else if(res == 1){
80         for(int j=1; j<=u; j++){
81             int res1 = power(a, pow(2, j)*v, p);
82             int res2 = power(a, pow(2, j-1)*v, p);
83             if(res1 == 1 and res2 != 1 and res2 != p-1){
84                 //p-1 == -1(mod.p)
85                 return false;
86             }
87         }
88     }
89 }
90 return true;
91 }
92
93 // nビットの素数を生成する
94 int generatePrime(int n, int k) {
95     if(n==1) return -1;
96     //n-1ビットの整数pを{0, 1}^(n-1)からランダムに選び,
97     //素数の候補をpの先頭に1を付加したnビットの整数
98     //最後に+1で奇数にする
99     int p = (1 << (n - 1)) + random(0, 1 << (n - 2)) + 1;
100     while (!calcMillerRabin(p, k)) {
101         p = (1 << (n - 1)) + random(0, 1 << (n - 2)) + 1;
102     }
103     return p;
104 }

```

```

105
106
107
108 int main(){
109
110     int n=16;
111     int s=1e2;
112     for(int i=2; i<=n; i++){
113         cout << generatePrime(i, s) << "\tis Prime Number" << endl;
114     }
115
116
117 }

```

Listing 8: 演習課題 8 のソースコード

```

1    #include <bits/stdc++.h>
2    #include <fstream>
3
4    using std::ofstream;
5    using namespace std;
6
7    ofstream ofs("random.csv"); // ファイルパスを指定する
8
9    long long int calcExtendedEuclid(long long int a, long long int b,
10                                     long long int &x, long long int &y){
11        if(b==0){
12            x = 1;
13            y = 0;
14            return a;
15        }
16        long long int d = calcExtendedEuclid(b, a%b, y, x);
17        y -= a/b*x;
18        return d;
19    }
20
21    long long int mod (int a, int m){
22        return (a%m + m) % m;
23    }
24
25    long long int modinv(long long int a, long long int m){
26        long long int x, y;
27        calcExtendedEuclid(a, m, x, y);
28        return mod(x, m);
29    }
30

```

```

31  /*
32   * 繰り返し自乗法を使った法 n のべき乗計算
33   *  (a の k 乗を n で割った余りを求める )
34   *
35   * unsigned long int a : 底
36   * unsigned int k : 指数
37   * unsigned int n : 法
38  */
39  unsigned int power(unsigned int a, unsigned int k,
40                    unsigned int n) {
41
42      a %= n;
43
44      if(a == 0 || n == 0){
45          return 0;
46      }
47      if(k == 0){
48          return 1 % n;
49      }
50
51      int i;
52      unsigned int value = 1;
53      for(i = 0; i < k; i++) {
54          value *= a;
55          if(value >= n) {
56              value %= n;
57          }
58      }
59      return value;
60  }
61
62  std::random_device rd;
63  std::mt19937 gen(rd());
64
65  int random(int low, int high) {
66      std::uniform_int_distribution<> rand(low, high);
67      return rand(gen);
68  }
69
70  /**
71   * @brief ミラーラビン法による素数判定
72   * @param p 素数の候補
73   * @param s 反復回数
74   * @return true
75   * @return false
76   */

```

```

77 bool calcMillerRabin(int p, int s){
78
79     //例外処理
80     if (p == 2) {
81         return true;
82     }
83     if (p < 2 || p % 2 == 0) {
84         return false;
85     }
86
87     int u = 0, v = p - 1;
88     while (v % 2 == 0) {
89         ++u;
90         v >>= 1; // v/=2;
91     }
92
93     for(int i=0; i<s; i++){
94         int a = random(1, p-1); //乱数生成
95         int res = power(a, p-1, p); //a^(p-1)のmod.p
96         if(res != 1){
97             return false;
98         } else if(res == 1){
99             for(int j=1; j<=u; j++){
100                 int res1 = power(a, pow(2, j)*v, p);
101                 int res2 = power(a, pow(2, j-1)*v, p);
102                 if(res1 == 1 and res2 != 1 and res2 != p-1){
103                     //p-1 == -1(mod.p)
104                     return false;
105                 }
106             }
107         }
108     }
109     return true;
110 }
111
112 // nビットの素数を生成する
113 int generatePrime(int n, int k) {
114     if(n==1) return -1;
115     //n-1ビットの整数pを{0, 1}^(n-1)からランダムに選び,
116     //素数の候補をpの先頭に1を付加したnビットの整数
117
118     int p = (1 << (n - 1)) + random(0, 1 << (n - 2)) + 1;
119     while (!calcMillerRabin(p, k)) {
120         p = (1 << (n - 1)) + random(0, 1 << (n - 2)) + 1;
121     }
122     return p;

```

```

123 }
124
125
126 vector<int> hirabun, angou;
127
128 void encryptor(int e, int n){
129     for(int m=0; m<n; m++){
130         int c = power(m, e, n);
131         angou.emplace_back(c);
132     }
133 }
134
135 void decryptor(int d, int n){
136     for(const auto &c: angou){
137         int m = power(c, d, n);
138         hirabun.emplace_back(m);
139     }
140 }
141
142 long long int calcEuclid(long long int a, long long int b){
143     if(b==0){
144         return a;
145     }
146     return calcEuclid(b, a%b);
147 }
148
149
150 long long int calcE(long long int p, long long int q){
151     long long int e = 2;
152     while(calcEuclid((p-1)*(q-1), e) !=1){
153         e++;
154     }
155     return e;
156 }
157
158 void calcRSASystem(int bit, int s, int p, int q){
159     cout << "p: " << p << ", q: " << q << endl;
160     int n = p*q;
161     int e = calcE(p,q);
162     cout << "N: " << n << endl;
163     cout << "E: " << e << endl;
164
165     int d = modinv(e, (p-1)*(q-1));
166     cout << "d: " << d << endl;
167
168     encryptor(e, n);

```

```

169     decryptor(d, n);
170     cout << "angou:\t";
171     for(const auto &v : angou){
172         cout << v << ", ";
173     } cout << endl;
174
175     cout << "hirabun:\t";
176     for(const auto &v : hirabun){
177         cout << v << ", ";
178     } cout << endl;
179
180 }
181
182 int main(){
183     int bit=4;
184     int s=1e2;
185     int p = generatePrime(bit, s);
186     int q;
187     while((q = generatePrime(bit, s) )== p){};
188     calcRSASystem(bit, s, p, q);
189 }

```