

# 情報理工学演習III 演習課題（第4回）

テーマ : 暗号

実施日 : 2020年5月14日（木）

担当 : 有村 博紀, 堀山 貴史, 金森 憲太朗 (TF), 加井 丈志 (TA)

授業連絡先 : [enshuiii-csit@ist.hokudai.ac.jp](mailto:enshuiii-csit@ist.hokudai.ac.jp) (演習課題の提出・質問など)

担当連絡先 : [kanamori@ist.hokudai.ac.jp](mailto:kanamori@ist.hokudai.ac.jp) (金森), [kai@ist.hokudai.ac.jp](mailto:kai@ist.hokudai.ac.jp) (加井)

以下の各問に関して、それぞれプログラムを実装せよ。実装したコードは必ず、以下に示す提出様式を守って、**本授業のメールアドレス ([enshuiii-csit@ist.hokudai.ac.jp](mailto:enshuiii-csit@ist.hokudai.ac.jp))**宛にメールで提出すること。

- メールの件名 : 「[csit ex3]xxxxxxxx」, ここで xxxxxxxx は自分の学生番号。
- 添付ファイル名 : 第 n 回演習の問 m に対するコードのファイル名は、「n\_m.c」とする。

例 : 学生番号02180000の学生の場合

- メール宛先 : [enshuiii-csit@ist.hokudai.ac.jp](mailto:enshuiii-csit@ist.hokudai.ac.jp)
- メール件名 : [csit ex3]02180000
- 添付ファイル : 4\_1.c, 4\_2.c, 4\_3.c

メール本文には、学生番号、所属コース、名前、実施回（第4回演習）、解いた問題番号を記載すること。課題提出によって出席をとるため、解いている途中の状態でも構わないので、演習時間内に一度は提出すること。

実装にはC言語を用い、gcc (ver.4.8.5 以降) でコンパイルすること。

コンパイル・実行例

```
% gcc 4_1.c      // コンパイル  
% ./a.out        // 実行
```

演習時間内に解き終わらなかった場合は、次回の演習時間までに提出すること。期間中であれば、何度でも再提出してよい。提出状況等は本授業のホームページまたはELMSグループ（moodle）上に掲示予定なので、適宜確認すること。

問題番号に \* が付されたものは、発展課題である。解答は必須ではないが、解ければ成績評価に加点されるので、是非挑戦してほしい。

**暗号**とは、メッセージを読解不能な文字列や、図、数列等に変換し、特定の知識を持つものにしかそのメッセージを理解できないようにする方法のことである。一般に、誰にでも読解可能な元のメッセージのことを**平文**といい、これを読解不能な形式に変換することを**暗号化**、暗号化によって得られるものを**暗号文**という。逆に、暗号文から平文の内容（多くの場合、平文そのもの）を復元することを**復号**という（注：「復号化」という語は誤り）。以降、説明にはこれらの用語を用いる。

いにしえの時代から、人々は情報を効率よく通信する手段が必要であることを知っていた。同時に、正当な受信者以外から、その情報を秘匿することの重要性も認識していた。紀元前5世紀頃の古代ギリシア世界では既に、戦争における作戦情報等の軍事機密の秘匿のために暗号が用いられていたと言われている。現代にあっても、暗号やその理論は軍事機密の秘匿のみならず、ログインパスワード、クレジットカード決済、SSH、仮想通貨（暗号資産）等、私たちの身近な環境にも広く利用されており、その重要性は益々高まっている。

最も理解しやすく、また歴史上最も広く用いられてきた暗号のひとつに、単換字式暗号がある。これは、平文の各文字を一定のルールに基づいて別の文字に置換し、別の文字列で表される暗号文に変換する方法である。

たとえばいま、M = I\_LOVE\_YOU という平文があるとする。これを単換字式暗号で暗号化すると、暗号文はた

とえば  $C = \text{LCORYHCARX}$  のようになる。これは、平文の各文字をそれぞれ3文字ずつ後ろの文字に置換して得られる（ $_$  は  $Z$  の次の文字とし、アルファベットは …  $XYZ\_ABC$  … のように循環するものとする）。

さて、暗号文  $C = \text{LCORYHCARX}$  を受信した者が正当な方法でこれを復号するには、以下の2つの知識が必要である。

- 暗号文が単換字式暗号で暗号化されていること
- 文字の置換ルールが「後ろに3文字ずらす」であること

前者を **プロトコル** といい、後者を **鍵** という。この2つの知識を有する者は、暗号文の各文字を3文字ずつ前の文字に置換して、愛の告白を受け取ることができる。

一般に現代では、システムを構築する関係等から、プロトコルは誰でも知ることができる。すなわち、第三者が暗号文  $C$  を傍受したとき、それがどのような暗号方式で暗号化されたものであるのかを知ることは容易である。一方で、鍵を暗号文や周囲の環境から知ることは難しい。暗号文を復号することは、その鍵を知ることと本質的に同じである。つまり、第三者が鍵入手することが困難な暗号ほど、良い暗号であると言える。

しかしながら、鍵についてはひとつ重要な問題がある。それは、暗号文の正当な受信者に、事前に安全に鍵を配布しておかねばならないということである。どんなに堅牢で解析困難な鍵を設計しても、この段階で第三者に鍵を鍵を知られてしまっては意味がなく、実際、歴史上そうして破られた暗号は枚挙に暇がない。インターネットを介してはじめから遠隔の相手と通信することの多い現代では、この鍵配送の問題は非常に厄介である。

**公開鍵暗号方式** は、このような問題を解決した革新的な暗号方式である。この暗号方式では、事前に通信相手に接触し、お互いだけで秘密の鍵をこっそり共有しておくという必要がない。

公開鍵暗号方式では、**公開鍵** と **秘密鍵** という2つの鍵を用いる。名前の通り、公開鍵は公開し、秘密鍵は秘密にしておく。公開鍵と秘密鍵は互いに対になっていながら、公開鍵から秘密鍵を推測することは困難であるという特殊な性質を持つ。

いま、アリスがボブに秘密のメッセージ  $M$  を送りたいと仮定し、公開鍵暗号方式でメッセージを暗号化する場合を考える。このとき、アリスは公開されているボブの公開鍵  $P$  を用いて平文  $M$  を暗号化し、暗号文  $C$  を得る。この暗号文  $C$  は、ボブの公開鍵  $P$  で復号することはできない。 $C$  を受信したボブは、自身の秘密鍵  $S$  を用いて  $C$  を復号し、元のメッセージである平文  $M$  を得る。数学的に書くならば、 $P$  と  $S$  は以下の条件を満たす。

- $P(M) = C$ かつ  $P(C) \neq M$
- $S(C) = M$

公開鍵暗号方式の仕組みを理解するには、南京錠を考えるとわかりやすい。ボブの公開鍵  $P$  は、解錠されている状態の南京錠である。これは誰でも施錠することができるが、解錠するには、南京錠の鍵を持っていなければならない。この「南京錠の鍵」が、ボブの秘密鍵  $S$  である。

いま、アリスは秘密のメッセージ  $M$  をボブに送信したいとする。アリスはボブが配布している南京錠をひとつ貰ってきて、 $M$  を入れた宝箱に施錠する。この時点で、 $M$  は秘匿され、施錠済み宝箱  $C$  となる。 $C$  を受け取ったボブは、彼だけが持つ鍵で南京錠を解錠し、宝箱の中のメッセージ  $M$  を得る。

**RSA暗号** は、現在最もよく知られ、また最も広く利用されてきた公開鍵暗号方式の暗号のひとつである。

### RSA暗号

$p$  と  $q$  を互いに異なる大きな素数とし、 $n = pq$ 、 $\varphi(n) = (p-1)(q-1)$  とする。  
 $e$  を  $\varphi(n)$  と互いに素な任意の小さな奇数とし、 $d$  を  $\varphi(n)$  を法とする  $e$  の逆数とする。すなわち  $e$  と  $d$  について、 $ed \equiv 1 \pmod{\varphi(n)}$  が成り立つ。

$P = (e, n)$  を公開鍵、 $S = (d, n)$  を秘密鍵とする。このとき、平文  $M$  と暗号文  $C$  に関して、以下の関係が成り立つ。

- $P(M) = M^e \pmod{n} = C$  （暗号化）
- $S(C) = C^d \pmod{n} = M$  （復号）

本来、平文  $M$  及び暗号文  $C$  は文字列であり、このまま幂乗や剰余の演算を行うことはできない。ここでは簡単のため、演算の際にはそれぞれ、

- $M$  : 各文字の ASCII コードから成る整数列
- $C$  : 整数列

と見なし、演算は各整数に対して行うこととする。

たとえば、 $M = \text{abracadabra}$  のとき、この各文字の ASCII コードから成る整数列は、

97 98 114 97 99 97 100 97 98 114 97

である。 $C$  は、この各整数に対して暗号化演算を行い得られる整数列である。すなわち、たとえば  $e = 3$ ,  $n = 187$  とすると、 $C$  は、

113 21 130 113 143 113 111 113 21 130 113

と得られる。

---

**【問1】** 以下の Sample 1 は、標準入力から下記の 入力形式 1 で公開鍵  $P = (e, n)$  と100文字以内の平文  $M$  を受け取り、 $M$  を  $P$  で暗号化して得られる暗号文  $C$  を、整数列の形式で出力するRSA暗号のプログラムである。これを 4\_1.c として実装せよ。

Sample 1

```
1: #include <stdio.h>
2:
3: int main(int argc, char const* argv[])
4: {
5:     int e;
6:     unsigned long long int n;
7:     char mes[100];
8:     unsigned long long int crypt[100];
9:     scanf("%d %llu", &e, &n);
10:    scanf("%s", mes);
11:
12:    int i = 0;
13:    while(mes[i] != '\0') {
14:        int exp = e;
15:        unsigned long long int base = mes[i] % n;
16:        unsigned long long int enc = 1ULL;
17:        while(exp > 0) {
18:            if (exp & 1) {enc = (enc * base) % n;}
19:            base = (base * base) % n;
20:            exp >>= 1;
21:        }
22:        crypt[i] = enc;
23:        printf("%llu ", crypt[i]);
24:        i++;
25:    }
26:    printf("\n");
27:
28:    return 0;
29: }
```

(註) 17 - 21行目では、ビットシフトを用いて効率的に冪乗の計算を行っている。& はビットの AND, >>= は右シフト後に代入する演算子である。

$e$ $n$ $M$	入力形式 1
----------------	--------

実装したプログラム 4\_1.c に以下の Input 1 を入力して実行し、結果を確認せよ。

7 10509761 abracadabra	Input 1
---------------------------	---------

**【問2\*】** 秘密鍵  $S = (d, n)$ 、及びRSA暗号で暗号化された  $m$  文字の暗号文  $C$  を受け取り、元の平文  $M$  を復号して出力するプログラムを 4\_2.c として実装せよ。このとき、 $C$  は秘密鍵  $S = (d, n)$  に対応する公開鍵  $P = (e, n)$  で暗号化されているものとする。

入力の暗号文  $C$  は、各要素が半角スペースで区切られた整数列の形式で、文字数  $m$  とともに下記の 入力形式 2 で標準入力から与えられるものとする。

出力は ASCII コードを各文字に変換して得られる文字列とし、末尾には改行を入れること。

$d$ $n$ $m$ $C$	入力形式 2
-----------------------	--------

107 187 11 113 21 130 113 143 113 111 113 21 130 113	入力例 1
--	-------

abracadabra	出力例 1
-------------	-------

実装したプログラム 4\_2.c に以下の Input 2 を入力して実行し、結果を確認せよ。

7502323 10509761 18 973730 9122666 3287610 4050902 8784581 9122666 4141681 4284341 2019515 2436396 9122666 4141681 2019515 195441 4141681 4518274 747714 10314103	Input 2
--	---------

**【問3\*】** RSA暗号は、公開鍵から秘密鍵を計算することの困難性を安全性の根拠としている。これはすなわち、公開鍵  $P = (e, n)$  から、 $n$  の素因数である2つの素数  $p$  と  $q$  を計算する素因数分解の難しさを意味する。仮にこの2つの素数  $p$  と  $q$  を知られると、そこから  $\varphi(n) = (p-1)(q-1)$  を法とする  $e$  の逆数  $d$  を計算することで、秘密鍵  $S = (d, n)$  を入手されてしまう。

ここでいう「計算が困難」とは、現在知られているどんな効率的なアルゴリズムを用いても、計算結果を得るのに、何千年、何万年という途方もない時間がかかることを意味する。RSA暗号においては、公開鍵の  $n$  を大きく

設定することで（すなわち、大きな素数  $p$  と  $q$  を用いることで）、この素因数分解は非常に困難になり、十分な安全性が得られる。しかしながら、 $n$  が小さい場合には、現実的な時間で素因数分解を計算することができ、RSA暗号は破られる。

脆弱なRSA暗号の公開鍵  $P = (e, n)$  を受け取り、これに対応する秘密鍵  $S = (d, n)$  を算出して、 $d$  を出力するプログラムを 4\_3.c として実装せよ。

入力は以下の入力形式 3 で標準入力から与えられるものとする。出力の末尾には改行を入れること。

入力形式 3
$e \ n$

実装したプログラム 4\_3.c に  $(e, n) = (7, 10509761)$  を入力して実行し、結果を確認せよ。