

## 情報通信実験 2（セキュリティ） 課題 5

### 注意事項

- 注意事項は課題 1 ～4 と同じです。
- レポート課題の提出日 T2SCHOLA を参照してください。
- 難しさの目安
  - [難易度☆]：易しい（作業）
  - [難易度☆☆]：普通（できる）
  - [難易度☆☆☆]：少し難しい（できるはず）
  - [難易度☆☆☆☆]：かなり難しい（できる人はいる）
  - [難易度☆☆☆☆☆]：激むず（できたらすごい）

## A. 古典暗号技術に関する課題

### 【課題 5 - 1】 [難易度☆☆]

平文 plaintext.txt を換字暗号で暗号化し、その際、暗号に利用した換字テーブル (A-Z、a-z、スペース、カンマ、ピリオド)、平文、暗号文を示せ。また、生成された暗号文を換字暗号によって復号し、正しく復号されることを確認せよ。この復号に利用した換字テーブル (暗号の逆変換テーブル) および復号文を示せ。

#### ■■■■ ヒント ■■■■

substitution\_cipher.c を利用してかまいません。このプログラムの内容については、以下を参照のこと。

- substitution\_cipher.c は、乱数を発生させ、その乱数に従って暗号化/復号用の換字テーブルを生成し、換字暗号によって暗号化/復号を行うプログラムです。ただし、復号の際の逆変換テーブルは定義されていないので、そのままでは復号は正しく行えません。各自復号の部分のコーディングを行い、プログラムを完成させて利用してください。以下にプログラム (substitution\_cipher.c) の追記すべき箇所を記します。

(182 行目から)

```
/** ランダム逆換字テーブルの作成・保存 */

/* ランダムテーブル鍵および初期値の入力・設定 */
scanf("%d", &KEY);

printf("Decryption key = [%d]¥n¥n", KEY);

srand(KEY);

for(i = 32; i < 128; i++)
{
    table[i] = i;
}

/* ランダム換字テーブルの設定 */
for(i = 0; i < 55; i++)
{
```

```

        j = rand()%55;
        t = table[t_table[i]];
        table[t_table[i]] = table[t_table[j]];
        table[t_table[j]] = t;
    }

/***** ここに逆換字テーブルを定義するコードを記述してください。 *****/
/***** 順方向の換字テーブル (table[i]) はすでに定義されているので *****/
/***** table[i]を逆方向に置換するテーブルを inv_table[i]として定義 *****/
/***** します。for 文を使えば 1 行で書けます。 *****/

/* ランダム換字逆テーブルの保存 */
for(i = 32; i < 128; i++)
{
    fprintf(fo_it,"%c, %c¥n",i, inv_table[i]);
}

for(i = 32; i < 128; i++)
{
    fprintf(fo_it_a,"%d, %d¥n",i, inv_table[i]);
}

/* 復号 */
shuffle_ascii(cipher_text, decipher_text, inv_table);

```

- 入力ファイルは平文 plaintext.txt、出力ファイルは暗号用換字テーブル (subcipher\_table.txt、subcipher\_table\_a.txt)、復号用換字テーブル (subcipher\_inv\_table.txt、subcipher\_inv\_table\_a.txt)、暗号文 (cipher.txt)、復号文 (decipher.txt) です。また、暗号鍵となる換字テーブルは、乱数の種 (10 桁以内の整数値) を入力することで、自動的にランダムな変換ルールを生成し、換字テーブルを作成しています。
- この換字テーブルでは、平文に含まれる文字のみランダム置換しています。平文に含まれる文字の種類とそれに対応する ASCII コードは以下の通りです。

表 1. 平文 (plaintext.txt) に含まれる ASCII コード

文字	ASCII コード
(スペース)	32
, (カンマ)	44
. (ピリオド)	46
A-Z (大文字アルファベット)	65-90
a-z (小文字アルファベット)	97-122

- 出力される換字テーブルのフォーマットは以下の通りです。

➤ subcipher\_table.tx、subcipher\_inv\_table.txt

(変換前文字) , (変換後文字)

➤ subcipher\_table\_a.tx、subcipher\_inv\_table\_a.txt

(変換前アスキーコード) , (変換後アスキーコード)

※それぞれ、1 行に一つの変換ルールが記載されています。

- 実行例

```
-bash-4.1$ gcc substitution_cipher.c -o substitution_cipher
```

```
-bash-4.1$ ./substitution_cipher
```

※実行すると、以下の処理が行われます。

- ① 平文表示
- ② 暗号鍵 (乱数の種となる整数値) 入力・表示 (入力後 Enter キーを押す)
- ③ 暗号文表示
- ④ 復号鍵 (乱数の種となる整数値) 入力・表示 (入力後 Enter キーを押す)
- ⑤ 復号文表示
- ⑥ 終了

【課題 5 - 2】[難易度☆☆]

5 - 1 で生成した暗号文のヒストグラム分布 (A-Z、a-b、スペース、ピリオド、カンマの出現頻度分布) を作成せよ。ヒストグラム分布は、グラフ表示すること。また、このヒストグラム分布から、換字暗号の問題点を指摘せよ。

■■■■ ヒント ■■■■

get\_text\_histogram.c を利用してかまいません。このプログラムの内容については、以下を参照のこと。

- get\_text\_histogram.c は、暗号文 cipher.txt を読み込み、このヒストグラム分布を出力するプログラムです。ただし、ヒストグラム生成の部分は定義されていないので、各自コーディングを行い、プログラムを完成させて利用してください。以下にプログラム (get\_text\_histogram.c) の追記すべき箇所を記します。

```
void get_char_histogram(char *input, int *char_hist)
{
    /*
        ascii コードは 32~127(96 個)
    */

    int i, j, t;
    int len;

    /* 文字列の長さ取得 */
    len = strlen(input);

    /* ヒストグラムの初期化 */
    for(i = 32; i < 128; i++)
    {
        char_hist[i] = 0;
    }

    /***** ここにヒストグラム分布を定義するコードを記述してください。 *****/
    /***** 各文字の頻度は、各文字の ACSII コードを i とすると char_hist[i] *****/
    /***** で定義されます。for 文を使えば 1 行で書けます。 *****/

}
```

- 入力ファイルは暗号文 (cipher.txt)、出力ファイルはヒストグラム (char\_histogram.txt) です。出力されるヒストグラムのフォーマットは以下の通りです。

➤ 出力画面

[ (文字) , (頻度) ]

➤ 出力ファイル (char\_histogram.txt)

(文字の ASCII コード) , (頻度)

※それぞれ、1 行に一つの文字の頻度が記載されています。

- 暗号文は A-Z、a-b、スペース、ピリオド、カンマから構成されていますが、出力するヒストグラムは 32 から 127 の ASCII コードについて定義します。

- 実行例

```
-bash-4.1$ gcc get_text_histogram.c -o get_text_histogram
```

```
-bash-4.1$ ./get_text_histogram
```

※実行すると、以下の処理が行われます。

- ① 暗号文表示
- ② ヒストグラム表示
- ③ 終了

【課題 5 - 3】 [難易度☆☆☆]

暗号文 cipher\_p.txt は、ある平文を換字暗号したものである。この暗号文を解読し、解読した平文および暗号解読に用いた換字テーブルを示せ。なお、cipher\_p.txt および元の平文は A-Z、a-b、スペース、ピリオド、カンマから構成される。

■■■■ ヒント ■■■■

暗号解読には、英文中の一般的なヒストグラム分布 (histogram\_of\_alphabet.pdf) や平文 plaintext.txt のヒストグラム分布を参考にしてください。解読を行うプログラムについては、analyze\_subcipher.c を利用してかまいません。このプログラムの内容については、以下を参照のこと。

- `analyze_subcipher.c` は、暗号文 `cipher_p.txt` と逆換字テーブル `subcipher_inv_table_a_p.txt` を読み込み、`cipher_p.txt` を復号した復号文 `decipher_p.txt` を出力するプログラムです。
- 逆換字テーブル `subcipher_inv_table_a_p.txt` のフォーマットは、課題 5 - 1 で作成した換字テーブルのアスキーコード版と同じです（以下の通り）。

(変換前アスキーコード) , (変換後アスキーコード)

※それぞれ、1 行に一つの変換ルールが記載されています。

- 各自で逆換字テーブル `subcipher_inv_table_a_p.txt` を（手動編集などで）作成し、プログラムにかけてみてください。
- 実行例

```
-bash-4.1$ gcc analyze_subcipher.c -o analyze_subcipher
-bash-4.1$ ./analyze_subcipher
```

※実行すると、以下の処理が行われます。

- ① 暗号文表示
- ② 逆換字テーブル表示
- ③ 復号結果表示
- ④ 終了

## B. ブロック暗号に関する課題

### 【課題 5 - 4】[難易度☆☆]

標準的なブロック暗号（DES や AES など）を用いて plaintext.txt を暗号化し、暗号文をダンプ（バイナリデータの 16 進表記）せよ。また作成した暗号文のバイト値分布ヒストグラム（16 進数表記で 00~FF の頻度分布）を求めよ。このブロック暗号による暗号文のバイト値分布ヒストグラムと、換字暗号による暗号文のバイト値分布ヒストグラムを比較し、換字暗号とブロック暗号の安全性の違いについて考察せよ。

#### ■■■■ ヒント①■■■■

ブロック暗号は、OpenSSL のコマンドを利用可能です。AES128bit を利用する場合、以下の手順に従って暗号文を作成してください。

- plain.txt を AES 128bit で暗号化し、crypted に出力する。

```
-bash-4.1$ openssl enc -e -aes128 -in plain.txt -out crypted
```

```
enter aes-128-cbc encryption password: (パスワードを入力)
```

```
Verifying - enter aes-128-cbc encryption password: (再度パスワードを入力)
```

- crypted を plain.txt に復号する。

```
-bash-4.1$ openssl enc -d -aes128 -in crypted -out plain.txt
```

```
enter aes-128-cbc decryption password: (パスワードを入力)
```

#### ■■■■ ヒント②■■■■

ダンプ結果を表示する際には、write\_dump.c を利用しても構いません。このプログラムの内容については、以下を参照のこと。

- write\_dump.c は、任意のファイル（ここでは crypted を指定）を読み込み、ダンプ結果を dump.txt に出力するプログラムです。
- 入力ファイルの形式は任意です。ただしファイルサイズの制限があります。
- 出力ファイル（dump.txt）のフォーマットは、以下の通りです。

（この行に出力する先頭バイト番号）：（データ系列（一行に 16 個））：（テキスト形式表示）

- 実行例

```
-bash-4.1$ gcc write_dump.c -o write_dump
```

```
-bash-4.1$ ./write_dump
```



- 出力ファイル (dump.txt) の表示例

The screenshot shows a text editor window titled "dump.txt - メモ帳". The content is a hex dump. The left column displays hexadecimal values in groups of 16 per line, with line numbers on the far left. The right column displays the corresponding ASCII text. A red box highlights a portion of the output. Below the screenshot, three blue arrows point to specific parts of the output, each with a corresponding label in a box:

- この行に出力する  
先頭バイト番号
- データ系列  
(一行に 16 個)
- テキスト形式表示  
データ系列

■■■■ ヒント③ ■■■■

バイト値分布ヒストグラムを作成するには、`get_binary_histogram.c` を利用してかまいません。このプログラムの内容については、以下を参照のこと。

- `get_binary_histogram.c` は、暗号文 `crypted` を読み込み、このバイト値分布ヒストグラム分布を出力するプログラムです。ただし、ヒストグラム生成の部分は定義されてい

いので、各自コーディングを行い、プログラムを完成させて利用してください。以下にプログラム (get\_binary\_histogram.c) の追記すべき箇所を記します。

```
void get_binary_histogram(unsigned char *input, int *char_hist)
{

    int i, j, t;
    int len;

    /* バイト列の長さ取得 */
    len = strlen(input);

    /* ヒストグラムの初期化 */
    for(i = 0; i < 256; i++)
    {
        char_hist[i] = 0;
    }

    /***** ここにヒストグラム分布を定義するコードを記述してください。 *****/
    /***** 各バイト値の頻度は、各バイト値を i とすると char_hist[i] で定 *****/
    /***** 義されます。for 文を使えば 1 行で書けます。 *****/

}
```

- 入力ファイルは暗号文 (crypted)、出力ファイルはヒストグラム (binary\_histogram.txt) です。出力されるヒストグラムのフォーマットは以下の通りです。

➤ 出力画面

[ (バイト値) , (頻度) ]

➤ 出力ファイル (binary\_histogram.txt)

(バイト値) , (頻度)

※それぞれ、1 行に一つのバイト値の頻度が記載されています。

- 出力するヒストグラムは 0 (00) から 255 (FF) のバイト値について定義します。

- 実行例

```
-bash-4.1$ gcc get_binary_histogram.c -o get_binary_histogram
```

```
-bash-4.1$ ./get_binary_histogram
```

※実行すると、以下の処理が行われます。

- ① 暗号文表示
- ② ヒストグラム表示
- ③ 終了

#### 【課題 5 - 5】[難易度☆☆☆]

5 - 4 で暗号化を行った平文を一文字変更し、この平文に対し、5 - 4 で暗号化した際に用いた鍵と同じ鍵を用いてブロック暗号化せよ。この暗号文と 5 - 4 で生成した暗号文とを比較し、これらの暗号文にどの程度の誤差があるかを示せ。誤差の表記方法としては、二つの暗号文の排他的論理和 (XOR) 演算を行い、その結果をダンプ表示すること。比較として、換字暗号についても同様に、平文を一文字変更させたときの暗号文と、変更前の平文を暗号化した暗号文との誤差分布を求めよ。これら結果から、換字暗号とブロック暗号の安全性の違いについて考察せよ。

#### ■■■■ ヒント ■■■■

ここではサンプルプログラムは提供しません。これまで作成したプログラムを利用して、自身でプログラムを書いてみてください。なお、排他的論理和演算は、以下の演算子を利用します。

(変数 a と変数 b の排他的論理和演算を変数 c に格納する場合)

```
c = a ^ b;
```

#### 【課題 5 - 6】[難易度☆☆]

周辺の学生 3 人に対し、その学生毎に異なる暗号鍵を生成、安全に共有し、何かしらの文書 (テキストファイル) をブロック暗号で暗号化し、生成した暗号文をその学生へ送付しなさい。受け取った学生はその暗号文を共有した鍵で復号しなさい。暗号文を送った分については、平文、暗号鍵、暗号文を、暗号文を受け取った分については、暗号文、復号鍵、復号文

をそれぞれ学生氏名と共に示せ。暗号文についてはダンプ表示すること。

■■■ヒント■■■

ブロック暗号は、課題 5－4 と同様、OpenSSL のコマンドを利用可能です。