情報通信実験 2（セキュリティ）課題 6

注意事項

- 注意事項は課題 1 〜5 と同じです。
- レポート課題の提出日は T2SCOLA 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. RSA に関する基本問題

【課題６－１】[難易度☆]

RSA の平文 M と暗号文 C の関係は、以下の式で表される。

$$C = M^e (mod\ n) \quad (1)$$

$$M = C^d (mod\ n) \quad (2)$$

公開鍵　$KU = \{e, n\}$、秘密鍵　$KR = \{d, n\}$とする。この時、n のビット長が 512bit とすると、暗号化可能な平文の bit 長は何 bit になるか。また、その理由を述べよ。

【課題６－２】[難易度☆☆]

RSA 暗号の鍵生成は、以下の手順によって行われる。

1. p, q を選ぶ（p と q は素数）
2. $n = p \times q$を計算する
3. $\phi(n) = (p-1)(q-1)$を計算する
4. 整数 e を指定する　$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
5. d を計算する　$d = e^{-1} mod \phi(n)$
6. 公開鍵　$KU = \{e, n\}$
7. 秘密鍵　$KR = \{d, n\}$

（１）　p=7, q=17, e=5 と設定したとき、秘密鍵　$KR = \{d, n\}$を求めよ。

（２）　（１）の設定において、公開鍵　$KU = \{e, n\}$および（１）式を用いて平文 M＝19 を暗号化せよ。その計算過程を示すこと。

（３）　（２）で生成した暗号文に対し、秘密鍵　$KR = \{d, n\}$および（２）式を用いて復号し、正しく復号できることを示せ。その計算過程を示すこと。

■■■ヒント■■■

鍵生成の手順の 5 番$d = e^{-1} mod \phi(n)$の計算は、

$$de = 1\ mod \phi(n)$$

となるような d を求めます。

【課題６－３】[難易度☆☆☆]

公開鍵 KU から秘密鍵 KR を求めるためには、n から $\varphi(n)$ を推定する（つまり n の素因数分解）ことが必要になる。ここでは、2 つの素数の素因数分解にどの程度時間を要するかを調査する。n を横軸、素因数分解に要した時間を縦軸とするグラフを作成せよ。横軸は対数表示すること。この結果から、素因数分解の計算に 1 年以上を要する n を求めよ。

■■■ヒント■■■

prime_factor_decomposition.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。

- prime_factor_decomposition.c は、ある整数を読み込み、その整数の素因数分解を行うプログラムです。
- このプログラムでは、素因数分解を行う整数値と、その素因数分解を繰り返し行う試行回数を入力します。
- このプログラムでは、19 桁の数値までしか素因数分解を行うことができません。
- 19 桁以内の素因巣分解は、ほぼ一瞬で終わります。よって、1 回の素因数分解の試行ではなく、同じ処理を複数回繰り返し試行して測定を行ってください（この繰り返し回数が、入力する「試行回数」です）。
- 試行回数の目安は、総処理時間が 1 秒以上になるように設定してください。3 桁の整数値の素因数分解を行う場合には、1 億回（10 の 8 乗）が目安です。
- 素数の一覧は prime number list.txt を参照してください。
- 素因数分解の計算時間の推定値は、厳密な値でなくて構いません。
- 実行例

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

※Warning が出ますが、無視してください。

```
Input integer number: 45    ←入力する整数値を入力

Input number of trials: 111    ←試行回数を入力

******Prime factor dcomposition is shown******
```

Prime factor:3　Multiplier:2　←最初の素因数と乗数（3 の 2 乗）
Prime factor:5　Multiplier:1　←次の素因数と乗数（5 の 1 乗）

****

Tatal processing time: 0.000331[sec]　←総処理時間（ここでは 111 回試行した時間）
Time per one process: 2.981982e-06[sec]　←1 回あたり処理時間（e-06 は 10^{-6}）

【課題 6 － 4 】[難易度☆]
　鍵長 512bit の RSA 公開鍵と秘密鍵を生成しなさい。また、この RSA 鍵ペアを用いて、RSA 暗号による暗号化、復号を行い、正しく暗号化・復号されることを確認せよ。このときの平文、暗号文、復号文を 16 進表記で示すこと。

■■■ヒント①■■■
　鍵生成には rsa_keygenerate.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。

- rsa_keygenerate.c は、RSA512 ビットの公開鍵および秘密鍵を自動的に生成するプログラムです。
- 関数は OpenSSL（https://www.openssl.org/）のライブラリを利用しています。コンパイル時に OpenSSL のライブラリを参照する必要があります（参照の仕方は下記実行例を参照のこと）。
- e の値は 65537 です。
- RSA 鍵は PEM 形式（鍵を ASN.1 でエンコーディングした後に Base64 によってテキスト化されたファイル）で保存されます。詳しく知りたい人はググってください。
- 出力ファイルは、公開鍵（public.pem）と秘密鍵（private.pem）の 2 種類です。
- 実行例

-bash-4.1$ gcc rsa_keygenerate.c -o rsa_keygenerate -lcrypto -lssl
-bash-4.1$ ./rsa_keygenerate

- 実行時の出力例

Private-Key: (512 bit)

modulus:　（n = p*q，法）

00:d5:a2:b4:92:92:bb:54:58:08:a2:ec:9b:f0:fb:
8e:e3:a2:32:ba:2c:ca:a7:94:18:07:f2:1a:29:00:
73:36:2e:29:c8:2c:19:40:ab:89:81:29:19:d1:ac:
f6:1a:8c:96:99:03:05:e7:cf:07:5a:55:2c:18:18:
90:40:d1:b5:0f

publicExponent: 65537 (0x10001)　（e = 65537，公開鍵）

privateExponent:（d = e^(-1) mod (p-1)*(q-1)，秘密鍵）

00:c4:c0:d2:d9:63:36:10:19:fb:ea:41:4b:e5:87:
69:34:10:bf:f3:63:29:49:69:45:30:9c:32:a7:ac:
78:6f:5e:28:42:94:3e:dc:36:a5:21:de:67:1a:0c:
8b:2c:89:65:24:9d:6e:39:0b:db:84:ca:43:96:ac:
99:22:18:2a:81

prime1:（p）

00:ff:58:15:1a:6e:97:16:11:95:61:f0:29:8b:80:
01:21:fe:eb:61:9c:09:a1:29:7d:59:39:63:28:7d:
01:d6:21

prime2:（q）

00:d6:2f:31:e3:53:80:ba:25:47:62:fb:e3:3b:66:
b8:ad:9f:6a:c4:d5:6b:7a:de:71:5e:29:24:c8:c5:
34:c5:2f

exponent1:（d mod (p-1)）

00:c2:d8:1a:da:5d:93:2a:c2:e6:23:a2:d8:80:db:
7f:81:ca:7d:20:b1:a9:e3:71:be:75:cc:45:af:0a:
9c:d1:21

exponent2:（d mod (q-1)）

43:ae:ea:4e:f2:06:4d:cc:96:00:7b:a4:d5:12:a2:
ed:8a:e1:0c:8e:7c:c6:79:20:ce:26:a6:4f:23:b2:
87:55

coefficient:（ q^(-1) mod p）

0a:ac:61:5a:fd:79:8b:90:e8:c0:6d:fd:48:32:b7:
62:c4:b6:9d:32:d3:ae:1a:a5:f3:e9:cd:b2:8b:95:
ed:c4

■■■ヒント②■■■

暗号化・復号には rsa_encdec.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。

- rsa_encdec.c は、RSA512 ビットの公開鍵および秘密鍵を読み込んで、暗号化、復号を行うプログラムです。
- これも OpenSSL のライブラリを参照する必要があります（参照の仕方は下記実行例を参照のこと）。
- e の値は 65537 です。
- RSA 鍵は PEM 形式の公開鍵（public.pem）と秘密鍵（private.pem）を実行ファイルと同じディレクトリに置いておきます（ファイル名固定）。
- 平文は、ソースコードに直接書き込みます。ただし 512 ビット未満（英文 64 文字以内）で書き込んでください。詳しくは以下を参照のこと。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/engine.h>

int main(int argc, char *argv[])
{
    RSA *public, *private;
    FILE *fp;
    int    i;
    long rsaSize;
    unsigned char *enc, *dec;
    unsigned char planeData[] = "This is plaintext.";
```

ここに暗号化したい平文を英文 64 文字以内で書き込みます。

- 出力は画面出力のみです。ファイルは出力されません。
- 実行例

-bash-4.1$ gcc rsa_encdec.c -o rsa_encdec -lcrypto -lssl
-bash-4.1$ ./rsa_encdec

平文: This is plaintext.

平文(HEX):
0x54 0x68 0x69 0x73 0x20 0x69 0x73 0x20 0x70 0x6C 0x61 0x69 0x6E 0x74 0x65 0x78 0x74 0x2E
暗号文(HEX):
0x96 0xB0 0xB3 0x2C 0x17 0xC7 0x05 0xF2 0x8E 0xC9 0x26 0x83 0x84 0x05 0x48 0x5A 0x4C 0x70 0x54 0x63 0x52 0x79 0x58 0x52 0x33 0x54 0xEA 0xE8 0xC7 0xE7 0x96 0xEF 0x87 0xD0 0xC4 0xE8 0x15 0x46 0x1A 0x26 0x25 0xD0 0x87 0x0A 0x4A 0x86 0xF6 0x62 0x73 0x6B 0x80 0xFA 0x47 0x85 0x4A 0x37 0x35 0x90 0x15 0x53 0x5F 0x54 0x3C 0x08
復号文: This is message.
復号文(HEX):
0x00 0x02 0x6A 0xCD 0x03 0x90 0x20 0x72 0x6D 0x46 0x4E 0xAC 0x80 0xBE 0x42 0x11 0xDA 0x32 0x32 0x77 0x83 0xBD 0xE3 0xB2 0x7C 0xB8 0x70 0xA5 0x3A 0x16 0xF3 0x82 0x6E 0x70 0xB6 0x8F 0xCD 0x64 0x30 0x1F 0x2F 0x9E 0x7F 0x0F 0x2C 0x00 0x54 0x68 0x69 0x73 0x20 0x69 0x73 0x20 0x70 0x6C 0x61 0x69 0x6E 0x74 0x65 0x78 0x74 0x2E

【課題６－５】[難易度☆☆☆☆]
課題６－４で行った暗号化・復号は、同じ平文および鍵を用いて暗号化、復号を行った場合でも、実行するたびに、生成される暗号文、復号文は変化する。これはなぜか。理由を述べよ。

B. RSA 署名に関する問題

【課題６－6】[難易度☆☆]
ある平文の RSA 署名を作成し、その署名の正当性を検証せよ。この時の平文、平文のハッシュ値、署名値、復号したハッシュ値を示すこと。

■■■ヒント■■■
RSA 署名・検証には rsa_signature.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。

- rsa_signature.c は、RSA512 ビットの公開鍵および秘密鍵を読み込んで、平文のハッシュ値生成、署名の生成、署名の検証を行うプログラムです。
- これも OpenSSL のライブラリを参照する必要があります（参照の仕方は下記実行例を参照のこと）。
- e の値は 65537 です。
- RSA 鍵は PEM 形式の公開鍵（public.pem）と秘密鍵（private.pem）を実行ファイルと

同じディレクトリに置いておきます（ファイル名固定）。

- 平文は、ソースコードに直接書き込みます。ただし 512 ビット未満（英文 64 文字以内）で書き込んでください。詳しくは以下を参照のこと。

```
#include <stdlib.h>
#include <string.h>
#include <openssl/sha.h>
#include <openssl/rsa.h>
#include <openssl/evp.h>
#include <openssl/pem.h>

int main ( int argc, char *argv[] )
{
        FILE *fp; // PEMファイル読み込みしに使用するファイルポインタ
        RSA *private; //  私密鍵
        RSA *public;   //  公開鍵
        int i;

        unsigned char message[] = "This is a message for the RSA digital signature.";
        unsigned char hash[SHA256_DIGEST_LENGTH]; //  署名対象. 平文のSHA-256
```

ここに電子署名を作成したいメッセージを英文 64 文字以内で書き込みま

- 出力は画面出力のみです。ファイルは出力されません。
- 実行例

-bash-4.1$ gcc rsa_signature.c -o rsa_signature -lcrypto -lssl
-bash-4.1$ ./rsa_signature
SHA256 digest:　←メッセージのハッシュ値
0x83 0x23 0xD8 0x53 0x7E 0x1A 0xCA 0x4F 0x8B 0x59 0x62 0x63 0xBA 0x39 0x14 0xB1 0x89 0x3D 0x1C 0xD9 0x8D 0x13 0xC4 0xFB 0xC9 0x64 0x25 0xE5 0x17 0x33 0x7C 0x2E
署名成功
署名文:　←作成した電子署名
0x3A 0x64 0xB6 0xEE 0xDC 0x01 0x0D 0x83 0x10 0xDB 0x8F 0xB8 0x95 0xB1 0x60 0x4D 0xDA 0x07 0x86 0x65 0x4F 0x3A 0xDF 0x2A 0x5E 0x31 0x55 0xE7 0x3F 0xBD 0x3D 0xFF 0x12 0x55 0x46 0xF6 0x8E 0x32 0x80 0xFA 0xAC 0x1C 0x55 0xD9 0xE7 0xA4 0xA7 0xF9 0x5C 0xDD 0x05 0xA4 0x43 0xA0 0x31 0xD5 0x15 0xA7 0x87 0x34 0x9B 0x12 0x09 0xDB

署名検証用データ(HEX):　←署名検証用データ（RSA 公開鍵で復号処理した値）
0x30 0x31 0x30 0x0D 0x06 0x09 0x60 0x86 0x48 0x01 0x65 0x03 0x04 0x02 0x01 0x05 0x04 0x20 0x83 0x23 0xD8 0x53 0x7E 0x1A 0xCA 0x4F 0x8B 0x59 0x62 0x63 0xBA 0x39 0x14 0xB1 0x89 0x3D 0x1C 0xD9 0x8D 0x13 0xC4 0xFB 0xC9 0x64 0x25 0xE5 0x17 0x33 0x7C 0x2E
署名検証成功　←検証結果

【課題６－７】[難易度☆☆☆☆]

RSA 署名の検証では、メッセージのハッシュ値と電子署名から生成した検証用データの一致を確認することで検証を行っている。しかし、課題６－６の結果を見ると、メッセージのハッシュ値と検証用データが一致していないにも関わらず、署名検証成功と表示されている。これはなぜか。その理由を述べよ。

【課題６－８】[難易度☆☆☆]

Problem1.txt, Problem2.txt, Problem3.txt, Problem4.txt には、それぞれ問題文が書かれている。これら４つのファイルから作成した電子署名ファイルが、それぞれ Problem1.txt.sig, Problem2.txt.sig, Problem3.txt.sig, Problem4.txt.sig,である。これらの電子署名の中に、公開鍵 public_for_problem.pem に対応した秘密鍵で署名したものが一つまたは二つ含まれている。それを特定し、特定したファイルの問題を回答せよ。

■■■ヒント■■■

- 必要なファイルは signature verification [6-8].zip から入手してください。
- 署名の検証方法については、rsa_signature.c を利用してかまいません。
- パディング方式は、PKCS #1 v1.5 を、ハッシュ値生成は、SHA256 を利用しています。
- ファイルは、バイナリデータとして取り扱ってください。

C. 共通鍵との比較に関する問題

【課題６－９】[難易度☆☆]

米国国立標準技術研究所（NIST）のセキュリティ関連文書である SP.800-57 には、共通鍵暗号と公開鍵暗号（RSA）の鍵長と暗号強度の関係が以下のとおり示されている。

セキュリティー強度（Bits）	共通鍵暗号	整数因数分解暗号（例:RSA）
≤80	2-Key Triple DES	k=1024
112	3-Key Triple DES	k=2048
128	AES-128	k=3072
192	AES-192	k=7680
256	AES-256	k=15360

出典：http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

これを参考に、セキュリティ強度が、80、112、128bit の共通鍵暗号と公開鍵暗号の演算速度を比較せよ。また、暗号化、復号にかかる時間に非対称性がある場合には、その理由を考察せよ。

■■■ヒント■■■

暗号速度の測定には all_speed_check.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。

- all_speed_check.c は、セキュリティ強度 80、112、128bit の共通鍵暗号および RSA 暗号の暗号化、復号処理を行い、その処理時間を計測するプログラムです。
- これも OpenSSL のライブラリを参照する必要があります（参照の仕方は下記実行例を参照のこと）。
- 1 回の試行では処理が瞬時に終わってしまい、計測が行えないので、それぞれ 10 万回の試行時間を計測しています。
- 入力は特にありません。
- 出力は、それぞれのセキュリティ強度に対し、共通鍵暗号のアルゴリズム、処理時間などが表示されます。
- 実行例

```
-bash-4.1$ gcc all_speed_check.c -o all_speed_check -lcrypto -lssl
-bash-4.1$ ./all_speed_check
```

※実行例は紹介しません。

【課題６－１０】[難易度☆☆]

周辺の学生３人に対し、何かしらの文書（テキスト）を RSA 暗号の公開鍵で暗号化し、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自身の秘密鍵で復号しなさい。暗号文を送った分については、平文、暗号文を、暗号文を受け取った分については、暗号文、復号文をそれぞれ学生指名と共に示せ。平文についてはテキスト表記、暗号文については 16 進表記（ダンプ）すること。なお、各自の RSA 公開鍵は、以下の手順に従って Web 上に公開すること。また、この実験と前回の実験課題【５－６】から、共通鍵暗号方式と公開鍵暗号方式の鍵配送における違いを考察せよ。

■公開鍵のアップ方法

①　自身の公開鍵ファイル（RSA512）に以下のファイル名を付けます。

public_(学籍番号).pem

例）学生番号が 12_34567 の場合

public_12_34567.pem

②　T2-Box のページにアップします。

（注意事項）

- T2-Box のアクセス情報については、T2SCHOLA にアップする「公開鍵アップロード用共有ボックス情報.txt」を参照してください。
- なお、T2-Box の有効期限は最長２週間のため、共有ボックスの運用開始時には、T2SCHOLA よりメールを送ります。
- このボックス内のデータは、読み書き削除は誰でも行えます。ファイル管理は自己責任でお願いします。

■■■ヒント■■■

暗号化・復号にはそれぞれ rsa_enc.c および rsa_dec.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。

- rsa_enc.c は、rsa_encdec.c の暗号化部分を利用し、暗号結果を出力するプログラムです。rsa_dec.c は、rsa_encdec.c の復号部分を利用し、復号結果を出力するプログラムで

す。

- これらも OpenSSL のライブラリを参照する必要があります（参照の仕方は下記実行例を参照のこと）。
- 暗号化時には PEM 形式の公開鍵、復号時には PEM 形式の秘密鍵を実行ファイルと同じディレクトリに置いておきます。
- 暗号時に入力する平文は、ソースコードに直接書き込みます。ただし 512 ビット未満（英文 64 文字以内）で書き込んでください。出力する暗号文は、バイナリ形式のファイルで出力されます。
- 復号プログラム rsa_dec.c では、暗号ファイル（バイナリ形式）をと復号鍵（PEM 形式）読み込み、復号文が出力されます。
- 詳しくは以下を参照のこと。

■暗号化プログラム（rsa_enc.c）の編集箇所

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/engine.h>

int main(int argc, char *argv[])
{
    RSA *public, *private;
    FILE *fp_r, *fp_w;
    int    i;
    long rsaSize;
    unsigned char *enc, *dec;
    unsigned char planeData[] = "This is test phrase for RSA 512.";
    unsigned char publickey[] = "public.pem";
    unsigned char encryptedData[] = "encrypted rsa.dat";
```

ここに暗号化を行う公開鍵のファイル名を書き込みます。

ここに RSA で暗号化したい平文を英文 64 文字以内で書き込みます。

ここに暗号化データのファイル名を書き込みます。

■復号プログラム（rsa_dec.c）の編集箇所

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/engine.h>

int main(int argc, char *argv[])
{
    RSA *public, *private;
    FILE *fp_rk, *fp_re, *fp_w;
    int    i;
    long rsaSize;
    unsigned char *enc, *dec;
    unsigned char privatekey[] = "private.pem";
    unsigned char encryptedData[] = "encrypted_rsa.dat";
```

ここに復号を行う秘密鍵のファイル名を書き込みます。

ここに暗号化データのファイル名を書き込みます。

- 実行例（暗号化）

-bash-4.1$ gcc rsa_enc.c -o rsa_enc -lcrypto -lssl
-bash-4.1$ ./rsa_enc

平文: This is test phrase for RSA 512.
平文(HEX):
0x54 0x68 0x69 0x73 0x20 0x69 0x73 0x20 0x74 0x65 0x73 0x74 0x20 0x70 0x68 0x72 0x61
0x73 0x65 0x20 0x66 0x6F 0x72 0x20 0x52 0x53 0x41 0x20 0x35 0x31 0x32 0x2E
暗号文(HEX):
0xA4 0x06 0xE1 0x9F 0x0A 0xC4 0x88 0x87 0x1D 0xF7 0x39 0x99 0x91 0x41 0x9D 0x81
0x58 0xFF 0x49 0x69 0x42 0xAA 0x9E 0x79 0xFB 0xCD 0x5F 0x62 0x31 0x8D 0xFE 0x0A
0xAA 0x80 0x38 0xE4 0xE1 0x34 0x19 0xB0 0xE1 0x52 0x24 0xCC 0xA8 0x97 0x0B 0xB3
0xAC 0x3D 0x65 0x15 0x99 0xEA 0xEE 0x2A 0x7E 0x33 0x58 0xC7 0x7A 0x58 0xFE 0x9F

- 実行例（復号）

-bash-4.1$ gcc rsa_dec.c -o rsa_dec -lcrypto -lssl

-bash-4.1$ ./rsa_dec

暗号文(HEX):
0xA4 0x06 0xE1 0x9F 0x0A 0xC4 0x88 0x87 0x1D 0xF7 0x39 0x99 0x91 0x41 0x9D 0x81
0x58 0xFF 0x49 0x69 0x42 0xAA 0x9E 0x79 0xFB 0xCD 0x5F 0x62 0x31 0x8D 0xFE 0x0A
0xAA 0x80 0x38 0xE4 0xE1 0x34 0x19 0xB0 0xE1 0x52 0x24 0xCC 0xA8 0x97 0x0B 0xB3
0xAC 0x3D 0x65 0x15 0x99 0xEA 0xEE 0x2A 0x7E 0x33 0x58 0xC7 0x7A 0x58 0xFE 0x9F
復号文: This is test phrase for RSA 512.
復号文(HEX):
0x00 0x02 0x17 0x1F 0x55 0xE6 0xB6 0xC6 0x07 0x4A 0xC9 0x6F 0x36 0x82 0xF9 0xF9
0x11 0x6E 0x5A 0xFF 0xA3 0xCD 0x9F 0xEB 0xB9 0x0A 0xCB 0xA1 0xB0 0xC3 0x64 0x00
0x54 0x68 0x69 0x73 0x20 0x69 0x73 0x20 0x74 0x65 0x73 0x74 0x20 0x70 0x68 0x72 0x61
0x73 0x65 0x20 0x66 0x6F 0x72 0x20 0x52 0x53 0x41 0x20 0x35 0x31 0x32 0x2E

【課題６－１１】[難易度☆☆☆]
自分の学生証に格納されている公開鍵証明書を読み出し、以下の情報をレポートに記載しなさい。
・発行者
・有効期限の終了日
・Common Name
また、証明書からわかる現在東工大ポータルで利用している証明書認証の問題点を考察しなさい。

（注意事項）
- IC カード R/W を所有していない人は、TA に申し出てください（マイナンバーカード用に IC カード R/W を購入した人は、それを利用することができます)。所有しない学生には、一時貸し出しを行いますが、数に限りがありますので、早めに申し出てください。(USB-C　2 台、USB-A 5 台）
- 証明書の取り出しには、証明書管理ツールが必要です。今まで利用したことがない人は、あらかじめ以下を参照して、自分の PC にインストールしておいてください。
  https://portal.titech.ac.jp/ezguide/cr-setup.html