Information Security

OpenSSL

security programming with openssl

32171550 박다은

2020-12-21

Contents

1. Overview

(1) Introduction

정보 통신 기술의 발전으로 인해 시공간의 제약 없이 네트워크를 통해 인터넷을 이용할 수 있게 되었다. 오프라인에서 이루어지던 정보 전달 방식들을 다양한 통신기기들을 사용하여 온라인 상에서 빠르고 쉽게 수행할 수 있게 된 것이다. 네트워크를 통하는 정보의 양은 점점 증가하고 있고, 네트워크의 전송속도는 더욱 더 빨라지고 있다. 하지만 서버에 저장된 정보나 전송중인 정보에 대한 다양한 위협들이 존재하고 있다. 따라서 암호화 알고리즘은 안전한 정보 전달을 위해 매우 중요하다. 이러한 암호화 알고리즘을 인터넷 상에서 구현하기 위해, 잘 알려진 라이브러리들이 존재하고 있다.

(2) Goals.

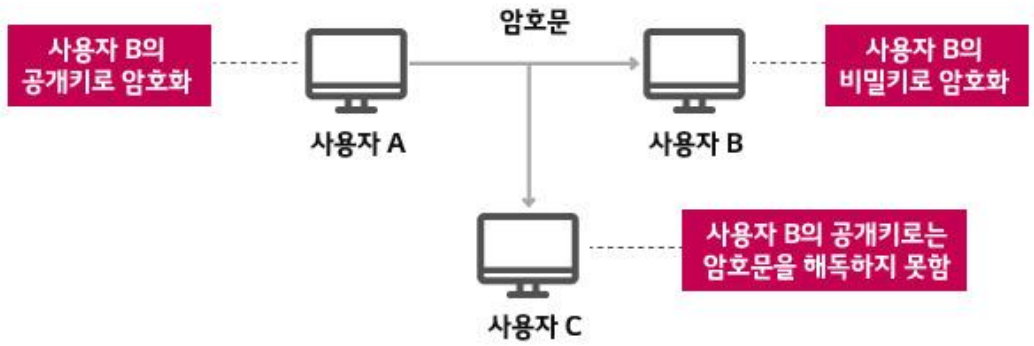
Openssl 을 통해 암호화/복호화 프로그램을 만든다. 시나리오는 다음과 같다.

나는 현재 머신러닝 프로젝트의 매우 중요한 데이터셋을 가지고 있다. 이 데이터셋을 다음 요구사항들을 충족시키며 머신러닝 프로젝트 팀에게 안전히 전달해야 한다.

1. 데이터셋을 암호화해야 한다.
2. 데이터셋의 해시를 구해야 한다.
3. 머신러닝 프로젝트 팀은 데이터셋의 무결성을 확인할 수 있어야 한다.
4. 머신러닝 프로젝트 팀은 데이터셋의 송신자를 입증할 수 있어야 한다.

(3) Basic Understandings

① 공개 키 암호화(RSA)



공개 키 암호 방식에서는 공개 키와 개인 키 두 개의 키를 사용한다. 공개 키를 이용해서 암호화하고 개인 키를 이용해서 복호화 하게 된다. 공개 키와 개인 키는 둘이 한 쌍이고 이를 키 쌍이라 부른다. 키 쌍을 이루는 2개의 키(공개 키와 개인 키)는 서로 밀접한 관계를 가지고 있고, 공개 키와 개인 키 쌍을 별개로 만들 수는 없다. 공개 키를 가지고 있는 사람이라면 누구든지 암호화할 수 있지만, 공개 키를 가져도 복호화 할 수는 없다. 복호화를 행할 수 있는 것은 복호화 키를 갖고 있는 사람뿐이다. 공개 키 암호를 이용해 데이터의 기밀성을 확인할 수 있다.

RSA 는 공개 키 암호 시스템의 하나로, 암호화뿐만 아니라 전자서명이 가능한 최초의 알고리즘으로 알려져 있다. 이번 프로그램에서는 암호화와 복호화하는 수단으로 RSA 를 사용할 것이다.

② 일방향 해시 함수

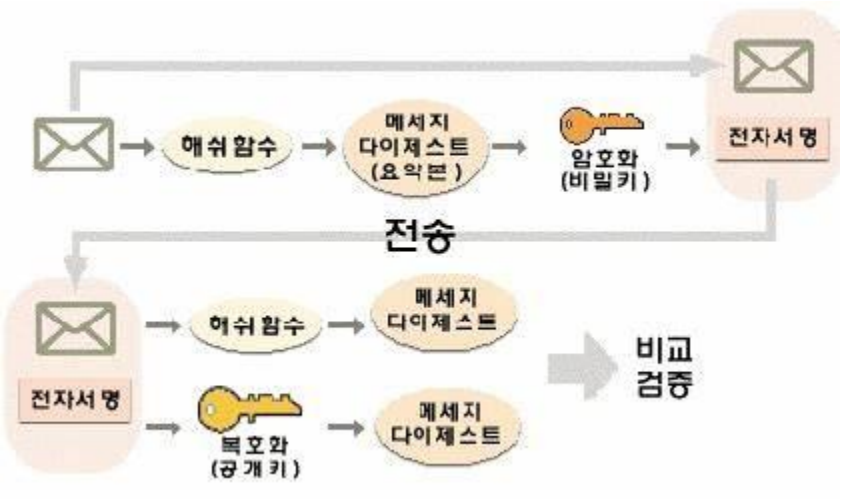
해시 함수란 임의의 길이의 데이터를 고정된 크기의 해시 값으로 변환해주는 함수이다. 일방향 해시 함수를 통해서는 데이터의 무결성을 확인할 수 있다. 데이터의 무결성을 검증하기 위해 데이터 그 자체를 비교하는 것이 아닌 일방향 해시 함수로 계산한 해시 값을 비교하여 무결성을 확인하게 되는 것이다. 메시지를 해시 함수에 넣으면 메시지의 고유한 해시 값이 생성된다. 메시지의 크기에 상관없이 고정된 해시 값을 가지므로 속도 면에서도 좋은 효율성을 보인다.

일방향 해시 함수의 예로는 MD4 와 MD5, SHA-1, SHA-256, SHA-512 등이 있다. SHA(Secure Hash Algorithm)은 NIST(National Institute of Standards and Technology)에서 만든 160 비트의 해시 값을 갖는 일방향 해시 함수이다. SHA-256, SHA-512 도 NIST 에서 만든 일방향 해시 함수로 각각 256, 512 비트 해시 값을 출력한다. 이번 프로그램에서는 SHA-256 을 이용하도록 하겠다.

③ 디지털 서명

공개 키 암호에서는 암호 키와 복호 키가 서로 나누어져 있어 암호키로 복호화를 행할 수가 없다. 또한 복호 키는 복호화를 수행하는 사람만이 갖고 있지만, 암호키는 암호화를 수행하는 사람이라면 누구나 가질 수 있다. 디지털 서명은 이 공개 키 암호를 역으로 사용함으로써 실현한다. 공개 키 암호에서 수신자가 복호화에 사용하는 개인 키가 디지털 서명에서는 서명자가 서명 작성에 사용하는 것에 해당된다. 또한 송신자가 암호화에 사용하는 공개 키는 디지털 서명에서 검증자가 서명 검증에 사용하는 것에 해당된다. 즉 송신자가 메시지를 '개인 키'로 암호화하는 것이 '서명 작성'에 해당하고, 수신자가 암호문을 '공개 키'로 복호화 하는 것이 '서명 검증'에 해당한다.

디지털 서명으로 서명을 작성하고 그 서명을 검증하는 방법은 '메시지에 직접 서명하는 방법'과 '메시지의 해시 값에 서명하는 방법' 두개가 있다.



'메시지의 해시 값에 서명하는 방법'은 다음과 같다. 먼저 송신자 앨리스는 메시지의 해시 값을 일방향 해시 함수로 계산하고 자신의 개인키로 해시 값을 암호화 해 메시지와 서명을 밥에게 송신한다. 이후 밥은 수신한 서명을 앨리스의 공개 키로 복호화하고 앨리스로부터 직접 수신한 메시지의 해시 값을 이 서명으로부터 얻어진 해시 값과 비교한다. '메시지에 직접 서명하는 방법'보다 복잡하지만 훨씬 적은 시간이 들고 효율성이 좋다.

'메시지의 해시 값에 서명하는 방법'이 더 안전하고 메시지를 전달할 때 사용하는 키쌍과 디지털 서명에 사용되는 키쌍이 다를 때 더욱 안전하므로 이 두 방법을 프로그램에서 사용하였다.

④ OpenSSL

SSL이란 (Secure Socket Layer)로 월드 와이드 웹 브라우저와 웹 서버 간에 데이터를 안전하게 주고받기 위한 업계 표준 프로토콜을 의미한다.

OpenSSL 은 네트워크를 통한 데이터 통신에 쓰이는 프로토콜인 TLS 와 SSL 의 오픈 소스 구현판이다. C 언어로 작성되어 있는 중심 라이브러리 안에는, 기본적인 암호화 기능 및 여러 유틸리티 함수들이 구현되어 있다.

(4) Program Structure

① 암호화(나)

```
+-----------------------------
|       Get message
|       Get Keys
|       Finding the hash value
```

| Encryption

| Writing signature

+----------

② 복호화(ML 프로젝트 팀)

+----------

| Get cipher message

| Get Keys

| Message decryption

| Finding the hash value

| Signature decryption

| Verifying signature

+----------

(5) Build Environment

Compilation: C (Visual Studio 2019)

2.Code & Consideration

(1) 키 쌍 만들기

키 쌍은 윈도우용 openssl 을 다운받은 후 명령 프롬포트(cmd)로 생성하였다. 암호화와 복호화하기 위한 수단으로 RSA 를 사용하였으므로 공개 키와 개인 키 두개가 필요하다. 또한 디지털 서명을 할 때도 키가 필요하므로 총 네 개의 키를 생성하였다.

```
C:₩Users₩ekdms₩openssl-0.9.8k_X64₩bin>openssl genrsa -out private.key 1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.............................++++++
.....................++++++
e is 65537 (0x10001)

C:₩Users₩ekdms₩openssl-0.9.8k_X64₩bin>type private.key
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDdncBU/s+WSbfIBGr9jAkIxXEs6Z4RIGt6pwqJHRfWISr6P+4k
+8dfx0L9wb4IAGZabySGj1iuJrXie6X7bUTUq0kYg3S5yoe/xyR8kJLCkX2KgJgc
xj0rinIY0Ie9daRgIv0YqvsQn8/juMIcEvHECB6tkmITFQMHpaLNumRrMwIDAQAB
AoGBANMYos3TcjT7v1k83hTpw0ID+Uwls/KW59ILLzPwLHZhXajGILfFx0HqjzWb
yMGib9yQpcAu5w/r+7DiivZ130rY7cmjVLjEr29gCX0IwxUjAQ+Jz3fU/roKKRvM
IF3wVZNa8nNBadQIc9CXCbSI0CKRQtPhxDZS2u1Fk36tAhrhAkEA8nW0kIc6qpFr
AErim3LxQvciI614XXKBQF92SKeIPU7/vp8IDq9i80frToVnFYfksWI0e3VS5KRH
+p72FRcxSwJBAOn+M6upXVP6BYzmwKb0C6UorK13aN7/Dyu899q8kAt04Ch/nmIm
SV79I4IER1Mile+VApyItKFicnixwjRR5LkCQQCg180ftVYB2T42j3cM0I+kcPYN
ORJMS63a2nIJcY14wnN+LWOT9N2aGwBKsszTx7a6ys5vKEG5VBI7/EAbdatJAkEA
meQjW3zR8o7244pHngXX0tIbbyKfiDm7Yoqvf96P5cAEUN5N8wfgrWZS3s5XEmgm
sCESTJzkH1e97X8LOJm7kQJAeeioqmTPTB13GzmZt9NwWtTMha6VUtN6+0+Fmkkj
mTUc+7sYdJGsI+309Mfef+nbEurvYyfBfN/Zt69S+ZPPDA==
-----END RSA PRIVATE KEY-----

C:₩Users₩ekdms₩openssl-0.9.8k_X64₩bin>openssl rsa -in private.key -out public.key -pubout
writing RSA key
```

먼저 openssl genrsa -out private.key 1024 라는 명령어를 이용하여 개인키를 생성하였고, 다시 openssl rsa -in private.key -out public.key -pubout 이라는 명령어를 사용하여 RSA 에서 사용되는 공개키를 생성하였다. 개인키로 공개키를 만든 것이다.

```
C:₩Users₩ekdms₩openssl-0.9.8k_X64₩bin>dir | findstr "pem key"
2020-12-21  오전 04:03               891 private.key
2020-12-21  오전 04:04               272 public.key
```

두 개의 키가 잘 생성된 것을 확인할 수 있다.

디지털 서명에서 사용될 키도 같은 방식으로 만들어 주었다. 생성된 개인키는 다음과 같다.

```
C:₩Users₩ekdms₩openssl-0.9.8k_X64₩bin>type private.key
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQC3EzsQbs0/ec55bU2Isx5ESqVcCuRrv5tc8++27cLRI6HsjIWI
Nx8j3ygu/Ic6j17JGm/a8ZIpzUY1mDs9I0rpcvRkSpQF3K+JPwd1cB+1cKUPCDX5
y8hreStqoulhn9FLjDL2ZnE5n0ugLoX10Bzl8jPIh6B4fhbl6aZ7oH0XnQIDAQAB
AoGAStNfVq4075m3XEXicJEvE7Z2s5H4HInoBCvnlBrFf6i8E9VxB9/pqF2zLNEl
7DdhBEz73r7UdzvhMFKSE7/6gtHKZGPq0JbFkKYUNIiBh3rGcd9Hu562H8LQh9Il
ymuJIXiw2wtvfW/K0gMNff0NiwDxsixaaPA6dWYtf504TcECQQDyv5Zr2MmcF6HM
XZgQvrq9RooN8Kqooj/dyU6lKhJXJkbAjM2rag2320f0CAt4l4diWMCbNfWc4gcf
O4fjKweFAkEAwRG1UvlaPzJQ9pBOrQVr0TBBmg1kBnEOaQ0LKZ5faJfuOMmoPYIV
JpKEQGJg8A1DIMZWoupWIGxiNcE8KD+v0QJAY+DKbmNLSKs8ik60l6emQ/yDDwtH
8lythH6wmip+U1dPYyhL80x2ioBxwebzeSH9MUzdKr9/6GXjkofBclmi1QJARCVX
9v2EWSINabn7NMbhE2kQ/qsTTyLAj1hs2jflPUbDSHalJ3M91uuI0qF+zbwgoYTg
N4i29aLL12VpMluolQJAU91EqStxROcaTYy8smU+N633MjhTcV/ppxQeQv3fRE3e
PyZPHxo0wxNV5rZqDzzlSRtWR+jH3skRYy41g9t7UQ==
-----END RSA PRIVATE KEY-----
```

(2) 주요 코드

① encryption.c (나)

1. 암호화할 메시지 가져오기

```
//암호화할 메시지 가져오기
FILE* r_file = fopen("message.txt", "r");
if (!r_file)return -1;
fread(plain_text, sizeof(plain_text), 1, r_file);
fclose(r_file);
//printf("암호화할 메시지는 다음과 같습니다.₩n%s₩n", plain_text);
```

- 먼저 암호화할 메시지파일을 가져오도록 하였다. 메시지가 길 수도 있으므로 fgets 함수 대신 fread 함수를 사용하였다.
- fread 함수를 통해 변수 plain_text 에 메시지를 저장하도록 하였다.

2. ML 팀의 공개 키 가져오기

```
//ML Team의 공개 키 가져오기
FILE* ML_key_f = fopen("MLTeamPublic.key", "r");
if (ML_key_f == NULL) {
    printf("'MLTeamPublic.key'를 찾을 수 없습니다.₩n");
    return 0;
}
RSA* ML_public_key = PEM_read_RSA_PUBKEY(ML_key_f, NULL, NULL, NULL);
if (ML_public_key == NULL) {
    printf("Read Public Key for RSA Error₩n");
    return 0;
}
fclose(ML_key_f);
```

- 머신러닝 팀(수신자)의 공개 키로 메시지를 암호화할 것이기 때문에 미리 만들어둔 머신러닝 팀의 공개 키를 불러오도록 하였다.

- PEM_read_RSA_PUBKEY 라는 함수를 사용해 pem 파일을 RSA 에서 사용할 수 있도록 하였다.

3. 나의 개인 키 가져오기

```
//나의 개인 키 가져오기
FILE* my_key_f = fopen("myPrivate.key", "r");
if (my_key_f == NULL) {
    printf("'myPrivate.key'를 찾을 수 없습니다.₩n");
    return 0;
}
RSA* my_private_key = PEM_read_RSAPrivateKey(my_key_f, NULL, NULL, NULL);
if (my_private_key == NULL) {
    printf("Read Private Key for RSA Error₩n");
    return 0;
}
fclose(my_key_f);
```

- 디지털 서명을 작성할 때 나(송신자)의 개인 키가 필요하므로 미리 만들어 둔 나의 개인 키를 불러오도록 하였다.
- PEM_read_RSAPrivateKey 함수를 사용하여 pem 파일을 RSA 에서 사용할 수 있도록 하였다.

4. 메시지의 해시 값 구하기

```
//메시지의 해시 값 구하기
result = calc_sha256(path, calc_hash);
//printf("메시지의 해시 값은 다음과 같습니다.₩n%s₩n", calc_hash);
```

- 일방향 해시 함수로 SHA-256 을 사용하였다.
- calc_sha256 함수는 해시를 구하기 위해 필요한 함수들이 들어있는 헤더파일인 hash.h 에 선언해두었다.
- hash.h 는 오픈소스를 사용하였으므로 따로 보고서에 작성하지는 않겠다.

5. 메시지 암호화하기

```
//메시지 암호화하기
memset(cipher_text, 0x00, sizeof(cipher_text));
num = RSA_public_encrypt(strlen(plain_text), plain_text, cipher_text, ML_public_key, RSA_PKCS1_PADDING);
//printf("암호문은 다음과 같습니다.₩n%s₩n", cipher_text);
```

```
FILE* w_file = fopen("cipher_text.txt", "w");
fwrite(cipher_text, num, 1, w_file);
printf("암호문을 저장했습니다.₩n");
fclose(w_file);
```

- 먼저 암호문을 저장할 cipher_text 를 memset 함수를 이용해 초기화 시켜주었다. 초기화하지 않으면 배열의 크기 때문에 쓰레기 값도 같이 출력되게 된다.
- RSA_public_encrypt 함수를 사용하여 메시지를 ML 팀의 공개키로 암호화시켰다.
- 그 후 ML 팀에게 전송할 목적으로 암호문을 저장하였다.

6. 해시 값 서명 작성하기

```
//해시 값 서명 작성하기
memset(sign, 0x00, sizeof(sign));
sign_len = RSA_private_encrypt(sizeof(calc_hash), calc_hash, sign, my_private_key, RSA_PKCS1_PADDING);
if (sign_len < 1) {
    printf("rsa private encrypt error₩n");
    return;
}
//printf("서명은 다음과 같습니다.₩n%s₩n", sign);

FILE* s_file = fopen("signature.txt", "w");
fwrite(sign, sign_len, 1, s_file);
printf("서명을 저장했습니다.₩n");
fclose(s_file);
```

- '메시지에 직접 서명하는 방법'이 아닌 '해시 값에 서명하는 방법'을 사용하였다.
- 위와 같은 이유로 memset 을 사용하여 서명이 저장될 sign 을 초기화 시켰다.
- RSA_private_encrypt 함수를 사용하여 해시 값을 나의 개인키로 암호화하였다.
- 역시 ML 팀에게 전송할 목적으로 서명을 저장하였다.

이와 같은 암호화 프로그램이 끝나고 나는 ML 프로젝트 팀에게 암호문과 서명을 전송하게 된다. 전송 방법은 이메일 등이 있다. 다음으로는 암호문과 서명을 수신 받은 ML 팀의 복호화 프로그램을 설명하도록 하겠다.

② decryption.c (ML 프로젝트 팀)

1. 복호화 할 암호문 가져오기

```
//복호화할 메시지 가져오기
FILE* file = fopen("cipher_text.txt", "r");
if (!file)return -1;
fread(cipher_text, sizeof(cipher_text), 1, file);
fclose(file);
//printf("암호문은 다음과 같습니다₩n%s₩n", cipher_text);
```

- 먼저, 수신 받은 암호문을 가져오도록 하였다. 암호화 때와 같은 이유로 fread 함수를 사용하였다.
- 암호문은 cipher_text 에 저장하도록 하였다.

2. ML 팀의 개인 키 가져오기

```
//ML Team의 개인 키 가져오기
FILE* ML_key_f = fopen("MLTeamPrivate.key", "r");
if (ML_key_f == NULL) {
    printf("'MLTeamPrivate.key'를 찾을 수 없습니다.₩n");
    return 0;
}
RSA* ML_private_key = PEM_read_RSAPrivateKey(ML_key_f, NULL, NULL, NULL);
if (ML_private_key == NULL) {
    printf("Read Public Key for RSA Error₩n");
    return 0;
}
fclose(ML_key_f);
```

- 암호문은 ML 팀의 개인 키로 복호화 되므로 미리 만들어 둔 ML 팀의 개인 키를 가져오도록 하였다.
- PEM_read_RSAPrivateKey 함수를 사용하여 pem 파일인 키를 RSA 에서 사용할 수 있도록 하였다.

3. 나의 공개 키 가져오기

```
//나의 공개 키 가져오기
FILE* my_key_f = fopen("myPublic.key", "r");
if (my_key_f == NULL) {
    printf("'myPublic.key'를 찾을 수 없습니다.₩n");
    return 0;
}
RSA* my_public_key = PEM_read_RSA_PUBKEY(my_key_f, NULL, NULL, NULL);
if (my_public_key == NULL) {
    printf("Read Private Key for RSA Error₩n");
    return 0;
}
fclose(my_key_f);
```

- 서명을 복호화 할 때 나의 공개 키가 필요하므로 미리 만들어 둔 나의 공개 키를 가져오도록 하였다.
- PEM_read_RSA_PUBKEY 함수를 통하여 pem 파일인 키를 RSA 에서 사용할 수 있도록 하였다.

4. 암호문 복호화하기

```
//메시지 복호화하기
memset(plain_text_receiver, 0x00, sizeof(plain_text_receiver));
num = RSA_private_decrypt(strlen(cipher_text), cipher_text, plain_text_receiver, ML_private_key, RSA_PKCS1_PADDING);
if (num < 1) {
    printf("rsa private encrypt error %d₩n",num);
    return;
}
//printf("복호화된 메시지는 다음과 같습니다.₩n%s₩n", plain_text_receiver);

FILE* d_file = fopen("decrypted_text.txt", "w");
fwrite(plain_text_receiver, num, 1, d_file);
fclose(d_file);
```

- memset 함수를 사용하여 복호화 된 메시지가 저장될 plain_text_receiver 을 초기화시켰다.
- RSA_private_decrypt 함수를 사용하여 암호문을 ML 팀 비밀 키로 복호화 시켰다.

5. 복호화 한 메시지의 해시 값 구하기

```
//메시지의 해시 값 구하기
result = calc_sha256(path, calc_hash);
//printf("복호화된 메시지의 해시 값은 다음과 같습니다.₩n%s₩n", calc_hash);
```

- 마찬가지로 hash.h 에 있는 함수인 calc_sha256 함수를 사용하여 복호화 한 메시지의 해시 값을 구했다.

6. 수신된 서명 복호화 하기

```
//수신된 서명 복호화하기
FILE* s_file = fopen("signature.txt", "r");
if (!s_file)return -1;
fread(my_sign, sizeof(my_sign), 1, s_file);
fclose(s_file);
//printf("수신된 서명은 다음과 같습니다.₩n%s₩n", my_sign);
```

```
memset(my_hash, 0x00, sizeof(my_hash));
my_hash_len = RSA_public_decrypt(strlen(my_sign), my_sign, my_hash, my_public_key, RSA_PKCS1_PADDING);
if (my_hash_len < 1) {
    printf("rsa private encrypt error %d₩n", my_hash_len);
    return;
}
//printf("복호화된 서명은 다음과 같습니다.₩n%s₩n", my_hash);
```

- 먼저 수신된 서명 파일을 불러왔다. 역시 fread 함수를 사용하였다.
- memset 을 사용하여 서명의 해시 값이 저장될 my_hash 를 초기화하였다.
- RSA_public_decrypt 를 사용하여 수신된 서명을 나의 공개 키로 복호화 하였다.

7. 서명 검증하기

```
//해시값 서명 검증하기
int Result;
Result = memcmp(my_hash, calc_hash, my_hash_len); //복호화한 값이 메시지 해시값과 같은지 확인한다.
if (!Result) printf("디지털 서명 검증이 완료되었습니다.₩n");
else printf("디지털 서명 검증에 실패하엿습니다.₩n");
```

- 마지막으로 복호화 한 메시지의 해시 값과 서명을 복호화 함으로써 얻은 해시 값을 비교하여 메시지의 무결성을 확인하고 거짓 행세를 검출하도록 하였다.

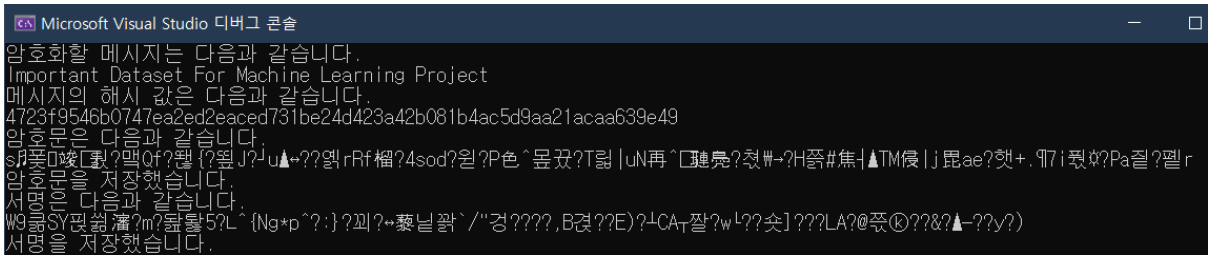
(3) Screenshots

① 출력화면

먼저 주석처리한 암호 알고리즘의 출력은 다음과 같다.

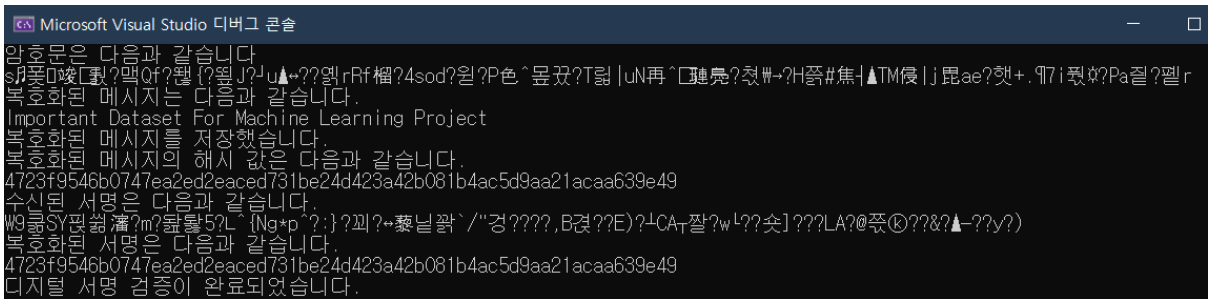
주석처리한 복호 알고리즘의 출력은 다음과 같다.

실제 프로그램이라면 이러한 출력도 문제없겠지만, 공부하고 있는 과정이므로 주요 결과들을 주석처리 하지 않은 출력도 살펴보도록 하겠다.



암호화 알고리즘의 출력은 다음과 같다.

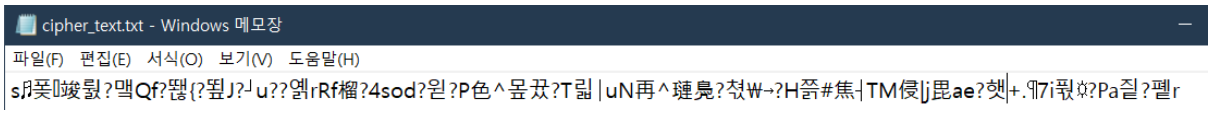
다음과 같은 메시지, 해시 값, 암호문, 서명을 갖고 있는 것을 확인할 수 있다.



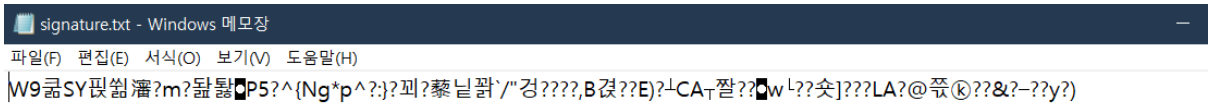
복호화 알고리즘의 출력은 다음과 같다.

암호문과 서명이 잘 전달된 것을 확인할 수 있고, 복호화도 잘 이루어진 것을 확인할 수 있다. 또한 복호화된 메시지의 해시 값과 복호화된 서명이 같아 서명 검증도 잘 이루어진 것을 확인할 수 있다.

② cipher_text.txt



③ signature.txt



④ decrypted_text.txt

(4) Problems & Solutions

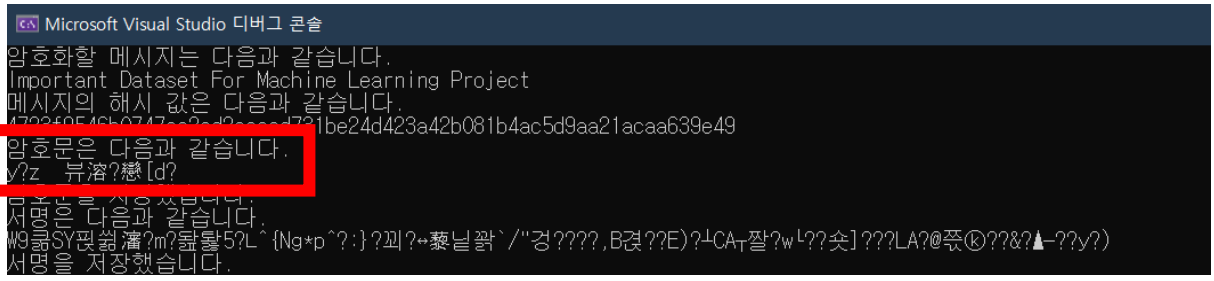
① 파일 입력

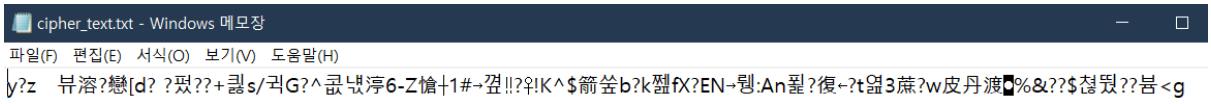
처음 파일 입력을 받아 배열에 저장할 때 fgets 함수를 사용하였었다. 인풋 메시지가 텍스트파일에서 한 줄이라 fgets 함수를 사용해도 별 문제가 되지 않았으나 메시지를 암호화한 암호문이 한 줄이 넘는 경우가 있어 오류가 발생하였었다. fgets 는 줄바꿈문자가 있다면 그 곳까지만 읽기 때문이다. 따라서 fread 함수를 사용하여 파일을 읽도록 하였다. fread 함수는 fgets 와는 달리 무조건 지정된 크기만큼 읽기 때문이다. 하지만 파일을 읽어 저장될 배열이 초기화 되어 있지 않는다면 실제 파일의 크기보다 배열이 더 클 때는 작성되지 않은 부분들이 쓰레기 값으로 채워지게 된다. 따라서 처음에 선언할 때 {0, }을 사용하여 미리 0 으로 전부 초기화 시켜주었다.

② RSA 함수

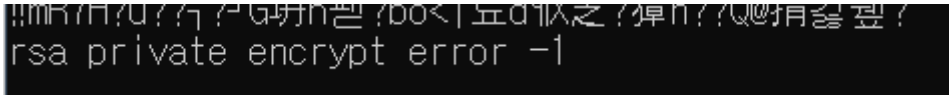
미리 만들어져 잘 쓰이고 있는 openssl 라이브러리를 활용하기 위해 인터넷에 검색을 많이 해보았었다. PEM_read_RSAPrivateKey 와 PEM_read_RSA_PUBKEY 함수도 그 예이다. 처음 인터넷을 통해 찾았을 때는 PEM_read_RSA_PUBKEY 함수가 아니라 PEM_read_RSAPublicKey 로 설명이 되어있어 이 함수를 사용했었다. 하지만 어떤 이유에서 인지 오류가 발생하였고 원래 있는 함수의 오류는 아닐 것이라 생각되어 다른 부분들만 고쳤었다. 하지만 결국에는 다른 부분들의 문제는 아니었고 함수를 바꾸어 주니 해결이 되었다. 익숙하고 잘 사용하던 라이브러리와 함수가 아니다 보니 어느 것이 문제인 지 모르고 주먹구구식으로 하나씩 고쳐보다가 해결됐다.

③ 전송 오류





비주얼 스튜디오 콘솔 창은 암호화 프로그램 후 실행되는 창이고, 메모장은 암호화된 메시지이다. 콘솔 창에서의 암호문과 파일 출력된 메모장을 비교해보면 서로 다른 것을 확인할 수 있다. 이유는 알 수 없지만 이런 식으로 콘솔창과 메모장의 암호문이 차이가 발생할 경우 복호화가 제대로 이루어지지 않았다.



이런 식으로 RSA_private_decrypt 함수에서 오류가 발생하게 되었다. 뒤에 -1 은 함수 실행 후 출력된 값이다. 함수를 실행시키면 복호화된 메시지의 길이가 반환되므로 양수의 값이 출력되어야 하는데 -1 이 출력됐던 것이다.

위에서 말했 듯이 익숙치 않은 라이브러리라 무엇이 문제인지 알아차리기 쉽지 않았고 역시 이유를 찾는 데 오랜 시간이 들었다. 인터넷에 쳐봐도 쉽게 나오지 않았기 때문이다. 암호화할 때마다 다른 암호문이 생성되므로 여러 번 돌려보다가 결국 이유를 찾을 수 있었다. 콘솔창과 메모장이 같은 암호문을 가지면 오류가 발생하지 않지만 조금이라도 다르다면 오류가 발생하는 것이었다.

만약 콘솔창과 메모장의 암호문이 다르더라도 암호화하는 프로그램에서 바로 복호화를 하면 문제가 발생하지 않았다. 하지만 암호문을 전송해 복호화하는 프로그램에서 복호화하면 위와 같이 오류가 발생하였다. 따라서 오류를 잡는 데 더 오랜 시간이 든 것 같다.

(5) Personal Feeling

이번 과제는 한 학기를 마무리하며 배웠던 여러 암호 기법들을 종합적으로 사용해보는 과제였다. 여태까지의 과제들과는 달리 미리 있던 라이브러리를 사용하였기 때문에 처음 시작할 때는 수월하게 할 수 있을 줄 알았다. 하지만 방대한 양으로 인해 라이브러리를 완전히 습득하는 것은 불가능에 가까웠고 코드를 작성할 때마다 인터넷의 힘을 빌리게 되었다.

직접 시나리오 속 주인공이 되어 코드를 작성하고 암호화와 복호화를 진행해보니 더욱 이해가 잘 되었다. 실무를 체험해보는 느낌도 들었다. 하지만 기말고사와 다른 과제들과 겹쳐 많은 시간을 쏟을 수는 없었어서 아쉬움이 남는다. 시간에 쫓겨 코드를 다듬지 못한 채로 제출하는 것이 매우 아쉽다. 이번 과제만 제출하면 한 학기가 마무리되므로 더욱 완벽하게 해 유종의 미를 거두고 싶었기 때문이다.

코드를 작성해보니 역시 관련 개념들은 완전히 이해를 할 수 있게 되는 것 같다. 강의를 듣고 책을 보며 공부할 때보다 직접 코드를 작성하니 암호기법의 단점들도 보이고 그냥 지식을 습득하는 이상으로 생각하며 공부하게 되는 것 같다. 코드를 작성하며 인터넷도 계속 찾아보니 추가적인 지식들도 꽤 얻게 되었다.

한 학기동안 여러 과제를 수행하고 공부하며 정보보호의 개념들을 많이 익힌 것 같아 뿌듯하다. 과제를 할 때는 원하는 대로 풀리지 않을 때도 있어 힘들었지만 한 학기를 돌아보며 생각하면 확실히 얻어가는 부분이 훨씬 큰 것 같다. 정보보호에 흥미도 많이 생겼고 과제를 하며 여러 언어들을 사용해볼 수도 있어 좋았다. 스스로도 한학기동안 고생했다고 칭찬해주고 싶다.

한 학기동안 감사했습니다 교수님.