정보보호
Homework 07

201204025
김대래

과목 명 : 정보보호
담당 교수 : 류재철
분반 : 00 분반

Index

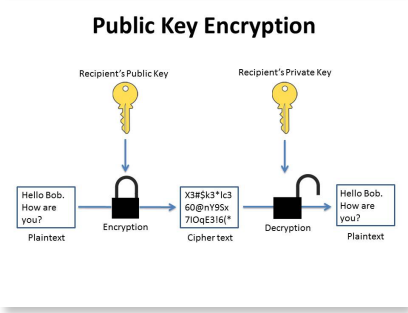
1. 실습 환경

OS : macOS(Mojave)
Language : C
Tool : Visual Studio Code, neovim, gcc

2. 주요 개념

I. 공개 키 암호방식

A. 개념

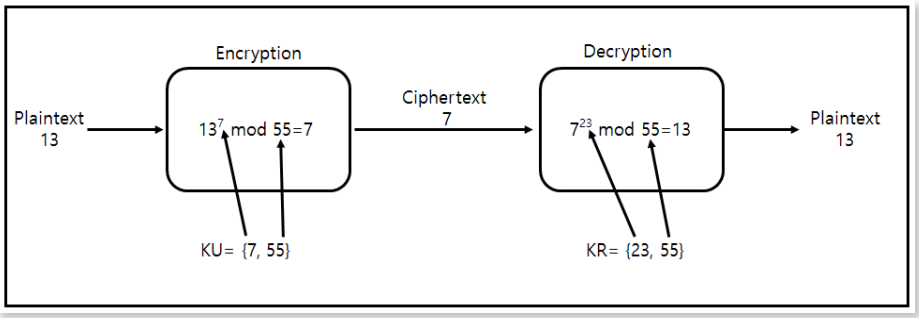


공개 키 암호 방식은 사전에 비밀 키를 나눠 갖지 않는 사용자들이 안전하게 통신할 수 있도록 하는 암호 방식의 한 종류이다.
공개 키 암호 방식에는 공개 키와 개인 키(비밀 키)가 존재하며, 공개 키는 누구나 알 수 있지만 그에 대응하는 개인 키는 소유자만 알 수 있어야 한다.(= 비대칭 암호)
즉, 암호화와 복호화에 사용하는 키가 서로 다른 암호화 방식을 의미한다.

B. 종류

- 디피-헬만(Diffie-Hellman) 키 교환
- RSA 암호
- DSA
- Rabin
- etc.

II. RSA 암호



A. 개념

공개 키 암호 방식의 하나로 암호화뿐만 아니라 전자서명이 가능한 알고리즘이다.
큰 숫자를 소인수 분해하는 것이 어렵다는 개념에 기반을 두고 수학적으로 암호화하는 방식을 갖고 있다.

B. 표현식

암호화의 경우 공개 키(N, e)를 통해 암호화하여 $c = M^e mod N$으로 표현한다.
복호화의 경우 개인 키(N, d)를 통해 복호화하여 $M = c^d mod N$으로 표현한다.

3. 과제 7 RSA를 이용한 파일 암•복호화

I. 과제 개요

- RSA를 이용하여 키 쌍을 생성하고 생성된 키를 이용하여 암•복호화를 진행한다.

프로그램을 실행(./file)하면 키 쌍을 생성(prkey.pem, pukey.pem, rsa.key)하고 평문(Plain.txt)을 공개 키로 암호화(plain.enc) 및 복호화(plain.enc.dec)하여 출력한다.

II. 문제 해결 과정

A. 주어진 RSA, BIO 함수 분석
B. make_keypair()함수로 별도 작성
C. RSA_public_encrypt, RSA_private_decrypt 함수 분석 - 인자(parameter)로 어떤 것이 들어가야 할 지 확인
D. plain.txt를 저장할 buff와 암호화 및 복호화 결과를 저장할 encrypted_buff, decrypted_buff 생성
E. 각 저장된 buff를 fwrite함수를 통해 파일 출력

III. 소스 코드

A. RSA* make_keypair()

```
RSA* make_keypair(){
    RSA *keypair = RSA_generate_key(KEY_LENGTH, PUB_EXP, NULL, NULL);

    //To get the C-string PEM form:
    BIO *pri = BIO_new(BIO_s_mem());
    BIO *pub = BIO_new(BIO_s_mem());

    PEM_write_bio_RSAPrivateKey(pri, keypair, NULL, NULL, 0, NULL, NULL);
    PEM_write_bio_RSAPublicKey(pub, keypair);
```

- RSA_generate_key 함수를 통해 키 쌍 생성
  - bit단위로 KEY_LENGTH 부분이 1024보다 작으면 안전하지 않다고 간주함
  - PUB_EXP는 공개 지수 e로 일반적으로 3, 17 또는 65537
  (과제에서는 1024, 3으로 정의)
- pri와 pub 메모리 바이오 생성
- PEM_write_bio_RSA(Private/Public)key()함수를 통해 BIO pri와 pub에 키쌍을 RSA 개인 키와 공개 키로 분리 저장

```
    int pri_len = BIO_pending(pri);
    int pub_len = BIO_pending(pub);

    char* pri_key = malloc(pri_len + 1);
    char* pub_key = malloc(pub_len + 1);

    BIO_read(pri, pri_key, pri_len);
    BIO_read(pub, pub_key, pub_len);

    pri_key[pri_len] = '\0';
    pub_key[pub_len] = '\0';
```

- BIO_pending함수로 pri와 pub의 데이터 양을 길이(len) 저장
- key의 내용을 저장할 char* 포인터 변수를 길이+1만큼 할당
- BIO_read함수를 통해 BIO pri와 pub를 포인터 변수 buf에 저장
- 마지막에 NUL 포인터 추가(문자열 특성을 갖기 위해)

```
    FILE *key_file;
    key_file = fopen("prkey.pem", "w");
    fprintf(key_file, "%s", pri_key);
    fclose(key_file);
    key_file = fopen("pukey.pem", "w");
    fprintf(key_file, "%s", pub_key);
    fclose(key_file);

    key_file = fopen("rsa.key", "w");
    RSA_print_fp(key_file, keypair, 0);
    fclose(key_file);

    return keypair;
}
```

- 개인 키 문자열과 공개 키 문자열이 저장된 pri_key와 pub_key
- 각각 prkey.pem 파일과 pukey.pem 파일로 저장
- 공개 키 쌍 keypair는 rsa.key 파일로 저장

B. main

```
int padding = RSA_PKCS1_PADDING;

int main(){
    RSA* keypair = make_keypair();

    char plainFileName[256];
    char encryptFileName[256];
    char decryptFileName[256];

    sprintf(plainFileName, "plain.txt");
    sprintf(encryptFileName, "plain.enc");
    sprintf(decryptFileName, "plain.enc.dec");

    FILE *input_FD = fopen(plainFileName, "rb");
    FILE *enc_output_FD = fopen(encryptFileName, "wb");
    FILE *dec_output_FD = fopen(decryptFileName, "wb");

    unsigned char *encrypted_buff = (unsigned char *)malloc(sizeof(unsigned char) * BLOCK_SIZE);
    unsigned char *decrypted_buff = (unsigned char *)malloc(sizeof(unsigned char) * BLOCK_SIZE);
    unsigned char *buff = (unsigned char *)malloc(sizeof(unsigned char) * BLOCK_SIZE);

    int t;
    while( 0 < (t = fread(buff, sizeof(unsigned char), BLOCK_SIZE, input_FD))){
        int encrypted_length, decrypted_length;
        printf("File Size : %d\n", t);
        encrypted_length = RSA_public_encrypt(t, buff, encrypted_buff, keypair, padding);
        fwrite(encrypted_buff, sizeof(unsigned char), encrypted_length, enc_output_FD);
        printf("File Size : %d\n", encrypted_length);

        decrypted_length = RSA_private_decrypt(encrypted_length, encrypted_buff, decrypted_buff, keypair, padding);
        fwrite(decrypted_buff, sizeof(unsigned char), decrypted_length, dec_output_FD);
    }

    fclose(input_FD);
    fclose(enc_output_FD);
    fclose(dec_output_FD);

    return 0;
}
```

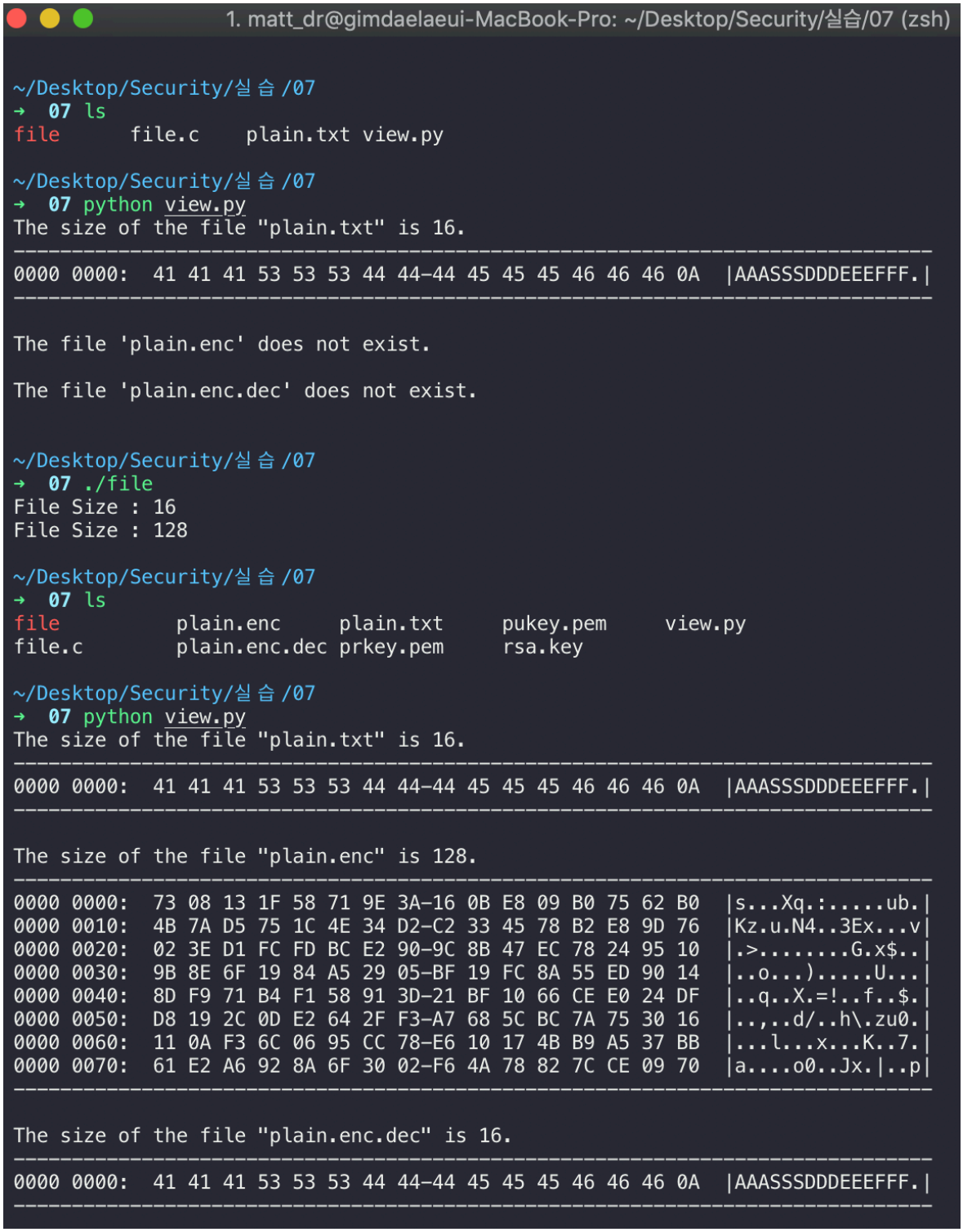
- 평문과 암호문, 복호문의 이름을 정의 및 fopen()함수로 파일 오픈
- 각 문자열을 저장할 buff, encrypted_buff, decrypted_buff를 BLOCK_SIZE만큼 할당(BLOCK_SIZE = 256)
- 평문을 읽어 buff에 저장
- buff를 암호화하여 encrypted_buff에 저장하고 파일쓰기
- 암호화된 encrypted_buff를 복호화하여 decrypted_buff에 저장하고 파일쓰기

함수 설명

- int RSA_public_encrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding) 함수
  - 공개키 rsa(= keypair)를 사용하여 from(=buff)에서 파일 크기 flen(=t) 바이트를 암호화하고 to(=encrypted_buff)에 암호화 텍스트를 저장하고 암호화 된 크기를 반환

- int RSA_private_decrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding) 함수
  - 개인키 rsa(= keypair)를 사용하여 해독하여 from(=encrypted_buff)에서 to(=decrypted_buff)로 저장하고 복호화 된 크기를 반환
- padding = RSA_PKCS1_PADDING으로 정의

4. 결과 화면


1. matt_dr@gimdaelaeui-MacBook-Pro: ~/Desktop/Security/실습/07 (zsh)
~/Desktop/Security/실습/07
→ 07 ls
file file.c plain.txt view.py
~/Desktop/Security/실습/07
→ 07 python view.py
The size of the file "plain.txt" is 16.
0000 0000: 41 41 41 53 53 53 44 44-44 45 45 45 46 46 46 0A |AAASSSDDDEEEFFF.|
The file 'plain.enc' does not exist.
The file 'plain.enc.dec' does not exist.
~/Desktop/Security/실습/07
→ 07 ./file
File Size : 16
File Size : 128
~/Desktop/Security/실습/07
→ 07 ls
file plain.enc plain.txt pukey.pem view.py
file.c plain.enc.dec prkey.pem rsa.key
~/Desktop/Security/실습/07
→ 07 python view.py
The size of the file "plain.txt" is 16.
0000 0000: 41 41 41 53 53 53 44 44-44 45 45 45 46 46 46 0A |AAASSSDDDEEEFFF.|
The size of the file "plain.enc" is 128.
0000 0000: 73 08 13 1F 58 71 9E 3A-16 0B E8 09 B0 75 62 B0 |s...Xq.:.....ub.|
0000 0010: 4B 7A D5 75 1C 4E 34 D2-C2 33 45 78 B2 E8 9D 76 |Kz.u.N4..3Ex...v|
0000 0020: 02 3E D1 FC FD BC E2 90-9C 8B 47 EC 78 24 95 10 |.>.....G.x$..|
0000 0030: 9B 8E 6F 19 84 A5 29 05-BF 19 FC 8A 55 ED 90 14 |..o...).....U...|
0000 0040: 8D F9 71 B4 F1 58 91 3D-21 BF 10 66 CE E0 24 DF |..q..X.=!..f..$.|
0000 0050: D8 19 2C 0D E2 64 2F F3-A7 68 5C BC 7A 75 30 16 |..,..d/..h\.zu0.|
0000 0060: 11 0A F3 6C 06 95 CC 78-E6 10 17 4B B9 A5 37 BB |...l...x...K..7.|
0000 0070: 61 E2 A6 92 8A 6F 30 02-F6 4A 78 82 7C CE 09 70 |a....o0..Jx.|..p|
The size of the file "plain.enc.dec" is 16.
0000 0000: 41 41 41 53 53 53 44 44-44 45 45 45 46 46 46 0A |AAASSSDDDEEEFFF.|