软件安全实验报告

2022 年 12 月 28 日

课程名称：	软件安全
成绩评定：	
实验名称：	密码技术应用实验
实验编号：	实验 7
指导教师：	刁文瑞
姓　　名：	李晨漪
学　　号：	202000210103
学　　院：	山东大学网络空间安全学院
专　　业：	网络空间安全

目录

1 实验目的

本实验的学习目标是让学生熟悉密钥加密与单向散列函数相关的概念。完成实验后，学生应该能够获得有关加密算法、加密模式、单向散列函数的第一手经验。此外，学生将能够使用工具和编写程序来加密/解密消息，为给定的消息生成单向散列值。

开发人员在使用加密算法和模式时会犯许多常见的错误。这些错误会削弱加密的强度，并最终导致出现漏洞。本实验向学生展示一些错误，并要求学生发起攻击以利用这些漏洞。

1. 密钥加密。
2. 加密模式、初始向量（IV）和填充（Padding）。
3. 使用加密算法的常见错误。
4. 使用密码库进行编程。

2 实验步骤与结果

2.1 Task 1：使用不同的密码算法和加密模式加密

分别使用-aes-128-cbc，-aes-128-cfb，-bf-cbc 进行加密。

```
[12/21/22]seed@VM:~/.../cipher$  openssl enc -aes-128-cbc -e -in plain.txt -out
cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[12/21/22]seed@VM:~/.../cipher$  openssl enc -bf-cbc -e -in plain.txt -out ciphe
r1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[12/21/22]seed@VM:~/.../cipher$  openssl enc -aes-128-cfb -e -in plain.txt -out
cipher2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

图 1:

2.2 Task 2：加密模式：ECB vs. CBC

正确设置头部信息，然后将 header 和数据组合到一个新文件中。请你自己选择一张图片，重复上面的实验并报告你观察到的现象。

从原图前 54 字节获取 header，并与加密后的图片第 55 字节开始拼接成完整 bmp。

```
[12/21/22]seed@VM:~/.../cipher$  openssl enc -aes-128-ecb -in ./pic_original.bmp
 -out ./p_ecb.bmp -e -K 00112233445566778889aabbccddeeff
[12/21/22]seed@VM:~/.../cipher$ openssl enc -aes-128-cbc -in ./pic_original.bmp
-out ./p_cbc.bmp -e -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[12/21/22]seed@VM:~/.../cipher$ head -c 54 ./pic_original.bmp >header
[12/21/22]seed@VM:~/.../cipher$ tail -c +55 p_ecb.bmp > ecb_body
[12/21/22]seed@VM:~/.../cipher$ tail -c +55 p_cbc.bmp > cbc_body
[12/21/22]seed@VM:~/.../cipher$ cat header ecb_body > newECB.bmp
[12/21/22]seed@VM:~/.../cipher$ cat header cbc_body > newCBC.bmp
[12/21/22]seed@VM:~/.../cipher$
```

图 2:

ECB 加密效果：

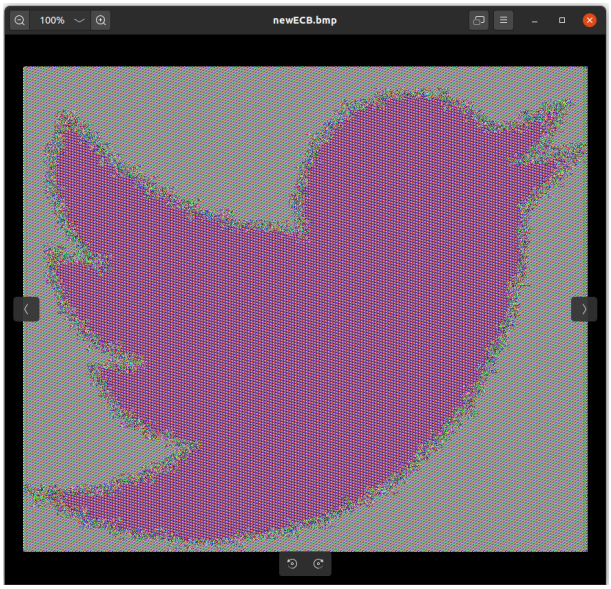

图 3:

CBC 加密效果：

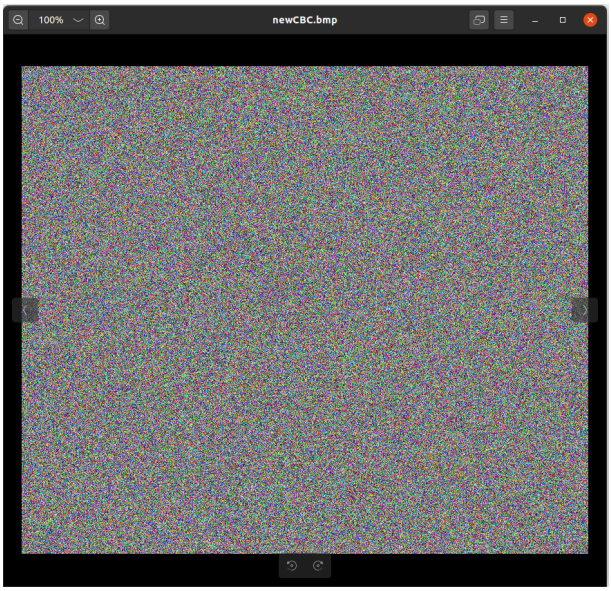

图 4:

重复实验：

```
[12/21/22]seed@VM:~/.../cipher$ openssl enc -aes-128-cbc -in ./sbn.bmp -out ./p_
cbc1.bmp -e -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[12/21/22]seed@VM:~/.../cipher$  openssl enc -aes-128-ecb -in ./sbn.bmp -out ./p
_ecb1.bmp -e -K 00112233445566778889aabbccddeeff
[12/21/22]seed@VM:~/.../cipher$ head -c 54 ./sbn.bmp >header1
[12/21/22]seed@VM:~/.../cipher$ tail -c +55 p_ecb1.bmp > ecb_body1
[12/21/22]seed@VM:~/.../cipher$ tail -c +55 p_cbc1.bmp > cbc_body1
[12/21/22]seed@VM:~/.../cipher$ cat header ecb_body1 > newECB1.bmp
[12/21/22]seed@VM:~/.../cipher$ cat header cbc_body1 > newCBC1.bmp
[12/21/22]seed@VM:~/.../cipher$
```

图 5:

原图：

图 6:

CBC 模式：

图 7:

ECB 模式：

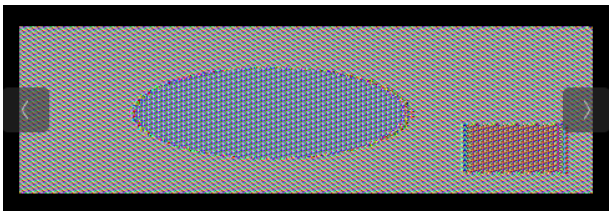

图 8:

可以发现，ECB 模式加密效果不如 CBC 模式好，ECB 模式能够得到图片的一些信息，CBC 模式则无法得到图片信息。

2.3 Task 3：错误传播-被破坏的密文

1. 创建一个至少 1000 字节长的文本文件。

2. 使用 AES-128 算法加密文件。

3. 损坏第 55 个字节的某一个 bit。

4. 使用正确的密钥和 IV 解密损坏的密文文件。

使用 ECB、CBC 分别加密。

```
[12/22/22]seed@VM:~/.../cipher$  openssl enc -aes-128-cbc -e -in f.txt -out f_cb
c.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[12/22/22]seed@VM:~/.../cipher$  openssl enc -aes-128-ecb -e -in f.txt -out f_ec
b.bin -K 00112233445566778889aabbccddeeff
[12/22/22]seed@VM:~/.../cipher$
```

图 9:

CBC 恢复的信息与明文有何不同用黑色标记：

```
[12/22/22]seed@VM:~/.../cipher$  openssl enc -aes-128-cbc -d -in f_cbc.bin -out
cbc1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[12/22/22]seed@VM:~/.../cipher$ xxd cbc1.bin
00000000: 496e 7465 726e 6174 696f 6e61 6c20 4665  International Fe
00000010: 6465 7261 7469 6f6e 206f 6620 4173 736f  deration of Asso
00000020: 6369 6174 696f 6e20 466f 6f74 6261 6c6c  ciation Football
00000030: 41eb 4fb4 bed1 dc7d dc19 c06c 6ce5 e498  A.O....}...ll...
00000040: 696f 6e20 7768 6962 6820 6465 7363 7269  ion whibh descri
00000050: 6265 7320 6974 7365 6c66 2061 7320 616e  bes itself as an
```

图 10:

ECB 恢复的信息与明文有何不同用黑色标记：

```
[12/22/22]seed@VM:~/.../cipher$  openssl enc -aes-128-ecb -d -in f_ecb.bin -out
ecb.bin -K 00112233445566778889aabbccddeeff
[12/22/22]seed@VM:~/.../cipher$ xxd ecb.bin
00000000: 496e 7465 726e 6174 696f 6e61 6c20 4665  International Fe
00000010: 6465 7261 7469 6f6e 206f 6620 4173 736f  deration of Asso
00000020: 6369 6174 696f 6e20 466f 6f74 6261 6c6c  ciation Football
00000030: 84bc 6496 b10f b2e4 825e c701 6a93 a486  ..d......^..j...
00000040: 696f 6e20 7768 6963 6820 6465 7363 7269  ion which descri
```

图 11:

问题：如果工作模式是 ECB 或 CBC，你分别能从解密后的损坏文件中恢复出多少信息？请在做这个任务之前回答这个问题，然后在完成此任务之后看看你的答案是否正确。给出你的理由。

如图：与原文比较，ECB 模式仅第 4 块（图中 00000030 行）信息损坏，损坏 128bit。与原文比较，CBC 模式仅第 4、5 块（图中 00000030、00000040 行）信息损坏，损坏 128+1bit。（注尽管 0040 行仅有 1bit 信息与原文不同，但第 5 行无法恢复出原文）

理由：AES-128 以 16 字节作为一块进行加密。ECB 模式 1bit 损坏**最多影响** 1 块即 128bit。CBC 模式 1bit 损坏**最多影响** 2 块即 256bit。

2.4 Task 4：寻找密钥

利用 openssl 中 aes-128-cbc 加密已知明文，枚举 words 中的密钥看是否与已知密文匹配，输出正确密钥。

```
#include <string.h>
#include<stdio.h>
#include<stdlib.h>
#include <openssl/err.h>
#include <openssl/evp.h>

const unsigned char res[] ={
0x78,0x27,0xba,0xf4,0x9c,0x24,0x64,0x15,
0x65,0x28,0xd9,0x9a,0x0f,0xcd,0xa1,0xc7,
0x14,0xd7,0xcb,0x59,0x1f,0xbe,0xd2,0x36,
0x4c,0x41,0xa9,0xfc,0x03,0x59,0x6c,0xc4};

void EN(const unsigned char* plain, char* key, const EVP_CIPHER*
    ciphertype, unsigned char
* buf, int* len){
    const unsigned char iv[17] ={
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x00, 0x0a, 0x0b, 0x0c, 0x0D, 0x0E, 0x0F};

    unsigned char K[17]; //char to unsigned char
    for (int i = 0; i < 17; i++){
        K[i] = key[i];}

    EVP_CIPHER_CTX* ctx;
    ctx = EVP_CIPHER_CTX_new();
    EVP_EncryptInit_ex(ctx, ciphertype, NULL, K, iv);
    EVP_EncryptUpdate(ctx, buf, len, plain, 21);
    EVP_EncryptFinal_ex(ctx, buf+*len, len);
    EVP_CIPHER_CTX_free(ctx);}

int equal(const unsigned char* a, const unsigned char* b){
    for (int i = 0; i < 32; i++){
        if (a[i] != b[i])
            return 0;}
    return 1;
}

int main(){
    unsigned char ciphertext[200];
```

```
39      int len;
40      unsigned char message[] ="This is a top secret.";
41
42      char getkey[20];
43      FILE* words= fopen( "words.txt","r");
44      while (fgets(getkey,20,words)){
45          int size = strlen(getkey)-1;
46          for (int i = size; i < 16; i++){
47              getkey[i] = '#';}
48          getkey[16] = '\0';
49
50          EN(message, getkey, EVP_aes_128_cbc(), ciphertext, &len);
51          if (equal(ciphertext, res)){
52              printf("the key is:\n");
53              for (int i= 0; i < size; i++){
54                  printf("%c", getkey[i]);}
55              printf("\n");
56              break;}}
57      fclose(words);
58      return 0;}
```

运行结果：key：wizard。

```
[12/28/22]seed@VM:~/.../cipher$ gcc findkey.c -o findkey -lcrypto -ldl -lstdc++
[12/28/22]seed@VM:~/.../cipher$ findkey
the key is:
wizard
```

图 12:

2.5 Task 5：生成消息摘要

分别使用-md5，-sha1，-sha256 生成消息摘要。

```
[12/22/22]seed@VM:~/.../cipher$ openssl dgst -md5 plain.txt
MD5(plain.txt)= 55aafe2183c86078ec9e7a7373cb8633
[12/22/22]seed@VM:~/.../cipher$ openssl dgst -sha1 plain.txt
SHA1(plain.txt)= 8d3b090efe42d1344c7d6d60a534a7f525e2b2e7
[12/22/22]seed@VM:~/.../cipher$ openssl dgst -sha256 plain.txt
SHA256(plain.txt)= 8bb6056870a453b486096ed317f6a7434aee28bf4090e6b0f6d8b4eccbe63
1a0
```

图 13:

2.6 Task 6：哈希函数的输出特性

1. 使用 SHA1 算法为 plain.txt 生成哈希值 H1。
2. 修改输入文件的一位。你可以使用 Bless 来完成此修改。

3. 为修改后的文件生成哈希值 H2。

```
[12/22/22]seed@VM:~/.../cipher$ openssl dgst -md5 plain1.txt
MD5(plain1.txt)= 2e29b733815a6269e107884fb99ef803
[12/22/22]seed@VM:~/.../cipher$ openssl dgst -md5 plain.txt
MD5(plain.txt)= 55aafe2183c86078ec9e7a7373cb8633
[12/22/22]seed@VM:~/.../cipher$
```

图 14:

其中，plain1.txt 是修改 1bit 的文件。发现输出的 H1 和 H2 完全不相似。看出哈希函数具有良好的单向性，不能从结果倒推出最初的哈希函数输入值。

2.7 Task 7：单向性与抗碰撞性

哈希得 SHA1 的前 24 位是：4dc981。

```
[12/28/22]seed@VM:~/.../cipher$ openssl dgst -sha1 original.txt
SHA1(original.txt)= 4dc981d2d8a8ea3901aedb7097d45dbd45109f61
```

图 15:

运行 5 次结果如下：
平均枚举次数：4299304。
原文本为（20 个字符）：
Lpob8KQxxrWdcHP2lMBf
6vO6x8SbpG1b2Tc41Yx0
5TcmvFTuVZm6mVl9nyAf
1cEECI04vsVgBYbm3017
v2QMa6nK8JOKZERegzqj

```
[12/28/22]seed@VM:~/.../cipher$ gcc brute.c -o brute -lcrypto -ldl -lstdc++
[12/28/22]seed@VM:~/.../cipher$ brute
Lpob8KQxxrWdcHP2lMBf
brute-counts:8681022
6vO6x8SbpG1b2Tc41Yx0
brute-counts:2448871
5TcmvFTuVZm6mVl9nyAf
brute-counts:4198499
1cEECI04vsVgBYbm3017
brute-counts:1261715
v2QMa6nK8JOKZERegzqj
brute-counts:4906412
```

图 16:

```c
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>
#include<time.h>
#include <stdlib.h>
```

```
6  const unsigned char Char[63] = "0123456789
       abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
7  int generateString(unsigned char* str, const unsigned int len){
8      unsigned int i, random;
9      for (i= 0; i < len;i++){
10         random = rand() % 62;
11         *str= Char[random];
12         str++;}
13     *str = '\0';
14     return len;}
15
16 int equal(const unsigned char* a, const unsigned char* b){
17 for (int i = 0; i < 3; i++){
18     if (a[i] != b[i])
19         return 0;}
20     return 1;
21 }
22
23 int main(){
24     for (int j = 0; j < 5; j++) {
25         int cout = 0;
26         while (1) {
27             const unsigned char res[] = { 0x4d,0xc9,0x81 };
28             const unsigned int strl = 20;
29             unsigned char str[21] = {};
30             int len = generateString(str, strl);
31             unsigned char out[20];
32
33             SHA1(str, strlen((const char*)str), out);
34             if (equal(res, out)){
35                 for (int i = 0; i < strlen((const char*)str); i++){
36                     printf("%c", str[i]);}
37                 printf("\n");
38                 break;}
39             cout++;}
40         printf("brute-counts:%d\n", cout);}
41     return 0;}
```

参考文献

[1] 杜文亮. 计算机安全导论：深度实践 [M]. 北京: 高等教育出版社,2020.4.