REPORT

Assignment#01

rand() 함수 무력화를 통한 비밀키 획득

과목명	시큐어코딩	담당교수	이승광 교수님
학 번	32204292	전 공	모바일시스템공학과
이 름	조민혁	제 출 일	2024/11/24

목 차

1. Introduction

Fig 1. Cryptography Logo

암호학은 고대부터 현대에 이르기까지 데이터 보호와 비밀 통신을 위한 핵심 기술로 발전해왔다. 고대 스파르타의 스키테일, 율리우스 카이사르의 시저 암호, 2 차 세계대전 중 독일군이 사용한 에니그마와 같은 초기 암호 기법은 특정 데이터를 비밀리에 전달하고자 하는 생각에서 시작되었다. 현대 암호학은 이러한 기초에서 발전하여 대칭키 암호화(AES, DES), 공개키 암호화(RSA), 암호학적 해시 함수(SHA-256), 그리고 최근 주목받는 양자 암호학 등으로 기술적 깊이와 범위가 넓어지고 있다.

암호학은 현대 정보화 사회에서 특히 중요한 의미를 가진다. 네트워크와 인터넷이 필수적인 도구가 된 현재, 금융 데이터, 개인 정보, 민감한 통신 등의 보안은 암호학의 역할에 크게 의존한다. 예를 들어, 2 차 세계대전 당시 에니그마 암호의 해독은 연합군의 전쟁 승리에 결정적인 역할을 했으며, 이는 암호학이 단순히 기술적 도구에 머무르지 않고 인류의 역사와 사회에 큰 영향을 미쳐왔음을 보여준다. 오늘날에도 암호학은 데이터 유출, 악의적인 해킹 시도, 보안 침해로부터 중요한 정보를 보호하기 위해 필수적인 기술로 자리잡고 있다.

본 레포트에서는 AES 암호화를 통한 결과에서 rand() 함수로 생성된 난수와 xor 연산을 통해 생성된 암호문에서 비밀키를 찾아내는 프로그램을 구현한다. 구체적으로는 AES 암호화와 XOR 연산을 결합하여 생성된 암호문을 분석하는 프로그램을 다루며, 여기에서 사용된 rand() 함수를 무력화하고, 역연산을 통해 AES 비밀키를 도출하는 프로그램을 구현한다.

본 레포트의 구성은 다음과 같다. 2 장에서는 프로그램 구현을 위해 필요한 요구사항을 살펴본다. 3 장에서는 요구사항을 기반으로 프로그램의 설계 및 구현 과정을 설명하고, 4 장에서는 구현 결과를 확인한다. 마지막으로 5 장에서는 결론을 서술하며 본 레포트를 마무리한다.

2. Requirements

Table 1. Requirements Table

Index	Requirements
1	rand() 함수를 항상 0을 리턴시키도록 바이너리를 수정
2	본인에게 할당된 16바이트 비밀키를 분석
3	popen으로 sec_hw1을 호출하여 출력값을 읽기
4	sec_hw1의 결과로 비밀키를 역산하는 함수를 구현하여 결과 출력

본 레포트에서 요구되는 요구사항들이 Table 1 에 제시 되어있다.

3. Implements

본 장에서는 프로그램을 구현하기에 앞서 주어진 프로그램을 분석한 후, 프로그램을 구현하도록 한다.

3-1. Process of Target Program

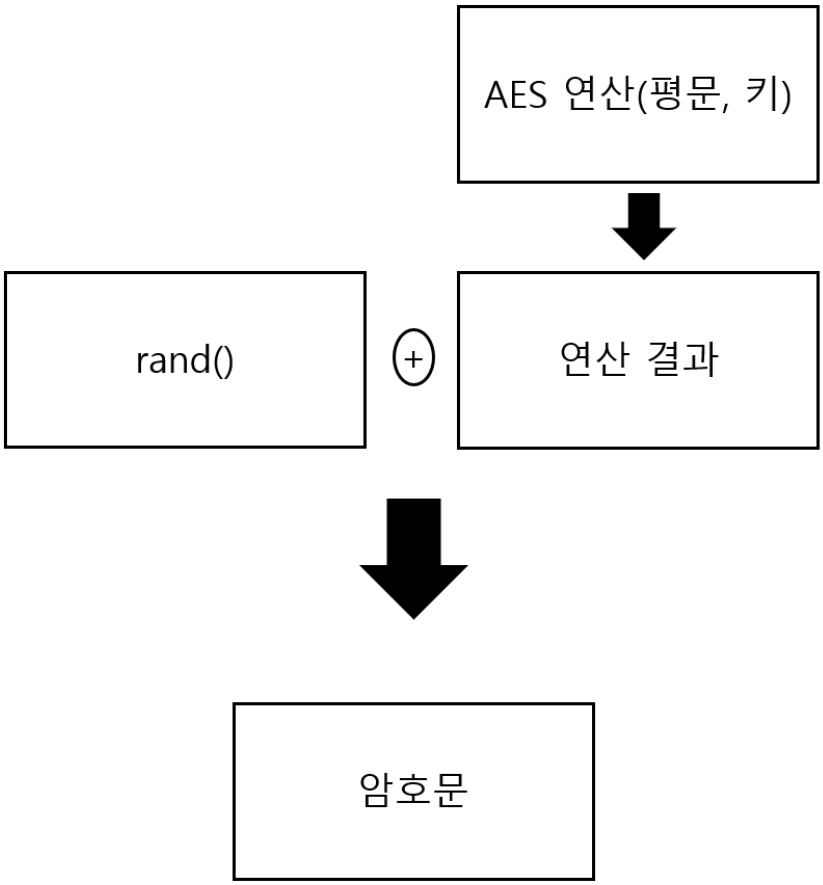

Fig 2. Process of Target Program

분석 대상이 되는 sec_hw1 프로그램의 순서도가 Fig 2 에 제시 되어있다. 먼저, 평문과 tab.bin 에 할당된 비밀키를 통해 AES 암호화 연산을 수행한다. 이후 rand() 함수로 생성된 난수와 xor 연산을 수행하여 암호문을 생성한다.

```
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver2
bc6b8c4792568c9f2fb78373edc64627
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver2
df56830b4c55469c308f891d85ddb7c2
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver2
997cbac8da570a9003e715d3abcb91ed
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver2
5e625e5a2192dd2eb95b07aa2daee987
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver2
3351ccf0d46baf81bc5cd9b046445736
```

Fig 3. Result of sec_hw1

Fig 3 의 sec_hw1 프로그램의 실행 결과가 제시 되어있다. 프로그램을 실행 시 매번 결과가 달라지는 것을 확인할 수 있다. 그 이유는 rand() 함수가 난수를 생성한 후 xor 연산을 수행하기에 매번 결과가 달라지는 것이다. 추가적으로 sec_hw1 프로그램은 평문이 고정되어 있다는 가정하에 AES 연산을 통해 동일 디렉터리 내 tab.bin 을 참조하여 AES Key 로 사용한다.

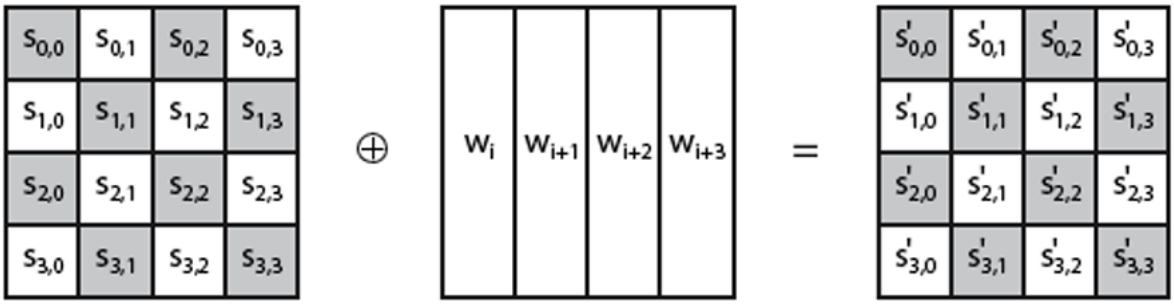

Fig 4. Addroundkey

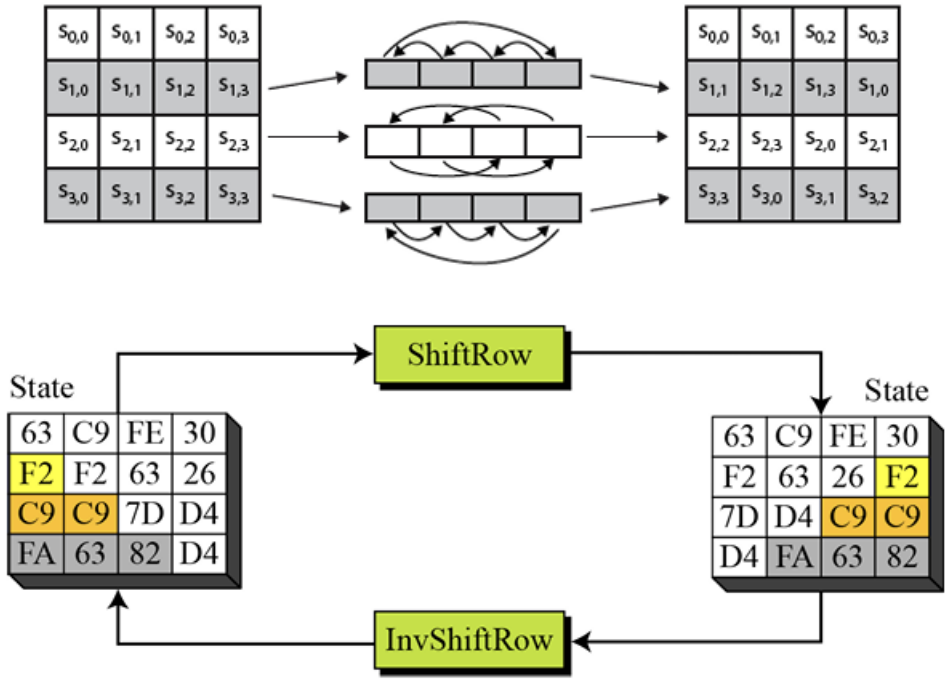

Fig 5. ShiftRows

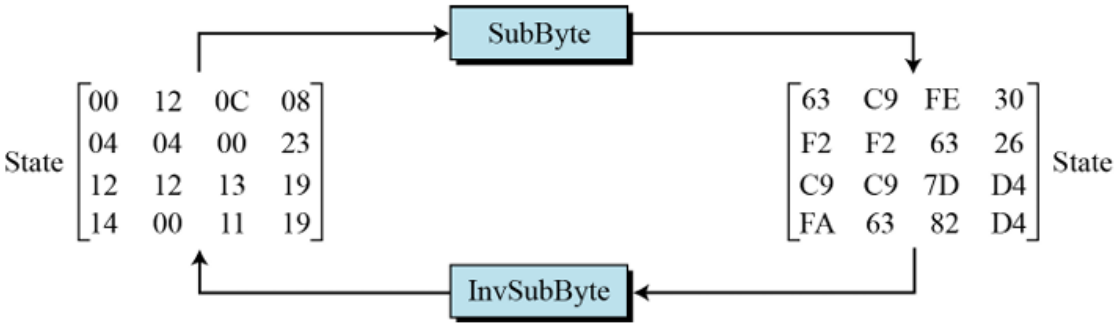

Fig 6. SubBytes

Fig 4, 5, 6 에 AES 암호의 1 Round 연산이 나타나 있다. 본 레포트에서는 분석의 용이함을 위해 1 Round 의 AES 암호화만 진행하였다. 먼저 Fig 4 에서 평문과 키의 xor 연산을 수행한다. 이후 Fig 5 에서 ShiftRows 를 통해 각 행마다 서로 다른 Shift 를 진행하고, Fig 6 에서 SubBytes 를 통해 미리 정의된 S-box 를 통해 치환을 수행한다.

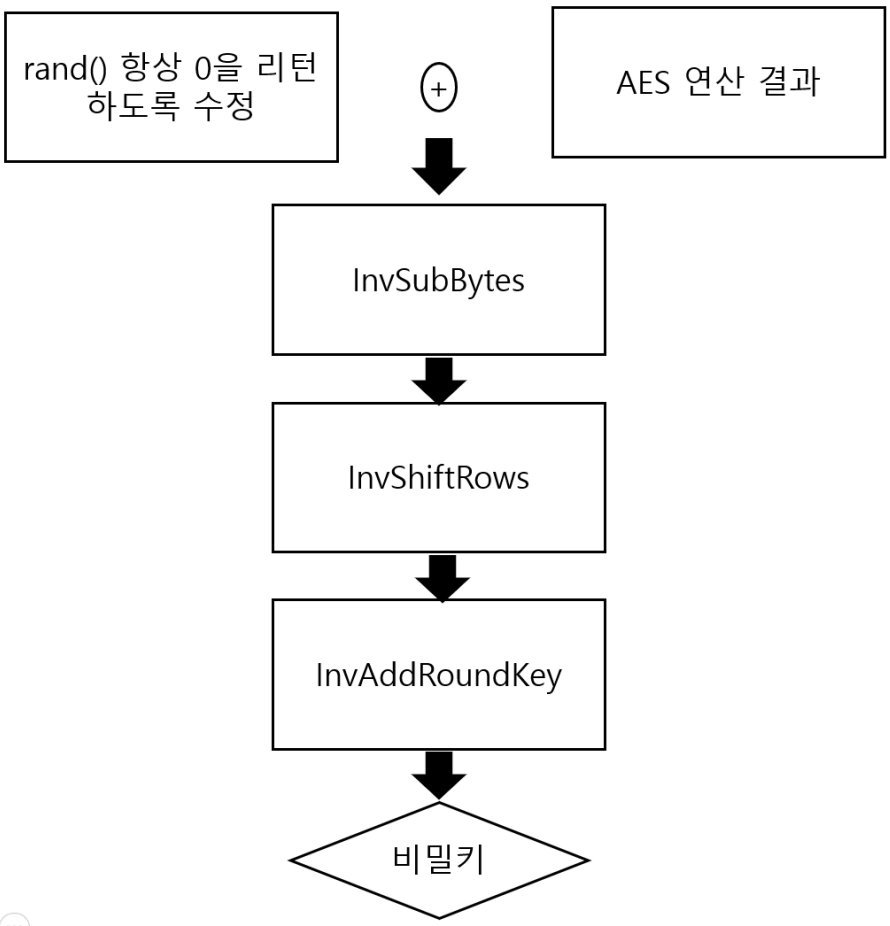

Fig 7. Process of Implemented Program

본 레포트에서 제시하는 프로그램의 순서도가 Fig 7 에 제시되어 있다. 구현하고자 하는 프로그램에서는 먼저 rand() 함수가 항상 0 을 리턴하도록 수정한다. 이를 위해 elfinject 를 통해 GOT 엔트리를 수정하여 rand() 함수가 호출될 때 마다 정의한 섹션이 호출되도록 코드의 흐름을 변경한다. 이후, AES 의 연산 결과와 xor 연산을 하게되면 결과적으로 AES 연산 결과가 그대로 출력된다. 이후, AES 암호화의 역연산인 InvSubBytes, InvShiftRows, InvAddRoundKey 연산을 통해 최종적으로 비밀키를 획득한다.

3-2. Paralyzation of Rand Function

앞서 언급했듯이 rand() 함수를 항상 0을 리턴하도록 하기 위해 elfinject 를 통한 무력화를 진행한다. 먼저, 섹션을 정의하기에 앞서 rand() 함수가 호출될 때 참조되는 GOT table의 주소를 확인하도록 한다. 이후, '.injected' 섹션을 정의한 후 elfinject를 통해 바이너리에 해당 섹션을 삽입한다. 이후, hexedit 도구를 사용하여 참조되는 GOT table의 주소를 .injected 섹션의 주소로 변경하도록한다.

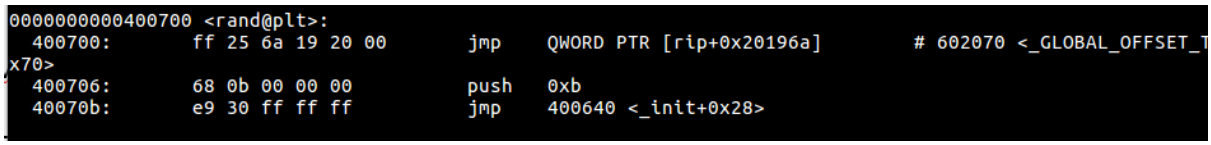

Fig 8. Address of rand()'s GOT

S

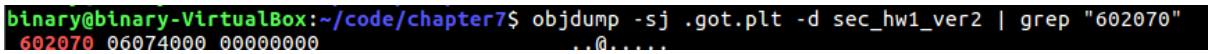

Fig 9. Value of rand()'s GOT

Fig 8에 rand 함수의 GOT table 주소가 나타나있다. 해당 주소를 추적하여 확인한 결과 Fig 9와 같이 해당 주소에서 호출하고 있는 값을 확인할 수 있다.

```
BITS 64
SECTION .text
global main

main:
  push rax                  ; save all clobbered registers
  push rcx                  ; (rcx and r11 destroyed by kernel)
  push rdx
  push rsi
  push rdi
  push r11

  pop r11
  pop rdi
  pop rsi
  pop rdx
  pop rcx
  pop rax

  mov rax, 0

  ret                       ; return
```

Fig 10. Assembly of .injected section

Fig 10에 정의한 섹션이 나타나 있다. 먼저 정의한 섹션으로 코드 흐름 변경 후 반환할 때 레지스터 값을 보존하고 복구하기 위해 push/pop 연산을 수행해준다. Rax는 어셈블리어에서 함수의 return 값을 담당하기 때문에 rax 값을 0으로 설정해줌으로써 rand() 함수 호출 시 항상 0만 return 하도록 해당 섹션을 정의한다.

S

```
binary@binary-VirtualBox:~/code/chapter7$ ./elfinject sec_hw1_ver1 rand-got.bin ".injected" 0x800000 -1
```

```
binary@binary-VirtualBox:~/code/chapter7$ readelf -S --wide sec_hw1_ver1
There are 31 section headers, starting at offset 0x2cb8:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        0000000000400238 000238 00001c 00   A  0   0  1
  [ 2] .init             PROGBITS        0000000000400618 000618 00001a 00  AX  0   0  4
  [ 3] .note.gnu.build-id NOTE            0000000000400274 000274 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        0000000000400298 000298 00001c 00   A  5   0  8
  [ 5] .dynsym           DYNSYM          00000000004002b8 0002b8 000150 18   A  6   1  8
  [ 6] .dynstr           STRTAB          0000000000400408 000408 00008c 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          0000000000400494 000494 00001c 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         00000000004004b0 0004b0 000030 00   A  6   1  8
  [ 9] .rela.dyn         RELA            00000000004004e0 0004e0 000018 18   A  5   0  8
  [10] .rela.plt         RELA            00000000004004f8 0004f8 000120 18  AI  5  24  8
  [11] .plt              PROGBITS        0000000000400640 000640 0000d0 10  AX  0   0 16
  [12] .plt.got          PROGBITS        0000000000400710 000710 000008 00  AX  0   0  8
  [13] .text             PROGBITS        0000000000400720 000720 0005e2 00  AX  0   0 16
  [14] .fini             PROGBITS        0000000000400d04 000d04 000009 00  AX  0   0  4
  [15] .rodata           PROGBITS        0000000000400d10 000d10 000029 00   A  0   0  4
  [16] .eh_frame_hdr     PROGBITS        0000000000400d3c 000d3c 000064 00   A  0   0  4
  [17] .eh_frame         PROGBITS        0000000000400da0 000da0 0001bc 00   A  0   0  8
  [18] .jcr              PROGBITS        0000000000601e20 001e20 000008 00  WA  0   0  8
  [19] .init_array       INIT_ARRAY      0000000000601e10 001e10 000008 00  WA  0   0  8
  [20] .fini_array       FINI_ARRAY      0000000000601e18 001e18 000008 00  WA  0   0  8
  [21] .got              PROGBITS        0000000000601ff8 001ff8 000008 08  WA  0   0  8
  [22] .dynamic          DYNAMIC         0000000000601e28 001e28 0001d0 10  WA  6   0  8
  [23] .got.plt          PROGBITS        0000000000602000 002000 000078 08  WA  0   0  8
  [24] .data             PROGBITS        0000000000602078 002078 000010 00  WA  0   0  8
  [25] .comment          PROGBITS        0000000000000000 002088 000034 01  MS  0   0  1
  [26] .bss              NOBITS          0000000000602088 002088 000008 00  WA  0   0  1
  [27] .injected         PROGBITS        0000000000800478 003478 000025 00  AX  0   0 16
  [28] .shstrtab         STRTAB          0000000000000000 002bab 00010c 00      0   0  1
  [29] .symtab           SYMTAB          0000000000000000 0020c0 0007c8 18     30  47  8
  [30] .strtab           STRTAB          0000000000000000 002888 00031e 00      0   0  1
```

Fig 11. elfinject of .injected section

Fig 11 은 elfinject 연산을 수행한 후 섹션 리스트를 보여준다. 27 번 인덱스에 .injected 섹션이 추가된 것을 확인할 수 있고, 해당 주소가 0x800478 로 설정된 것을 확인할 수 있다. 삽입한 섹션의 주소를 확인하였으므로 hexedit 도구를 이용하여 코드 패치를 수행한다.

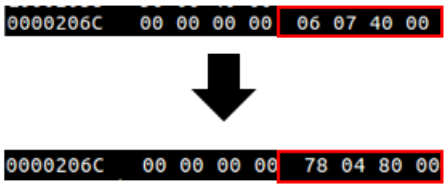

Fig 12. Code Patch with hexedit

Fig 12에서 코드 패치를 수행한 것을 확인할 수 있다.

```
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver1
d3cae82181fdb26baa54a7acb426cd36
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver1
d3cae82181fdb26baa54a7acb426cd36
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver1
d3cae82181fdb26baa54a7acb426cd36
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver1
d3cae82181fdb26baa54a7acb426cd36
binary@binary-VirtualBox:~/code/chapter7$ ./sec_hw1_ver1
d3cae82181fdb26baa54a7acb426cd36
```

Fig 13. Execution of sec_hw1 after elfinject

Fig 13에서 elfinject를 수행한 후 sec_hw1 프로그램의 결과를 확인할 수 있다. 5번의 실행을 수행하는 동안 모두 같은 결과를 출력하고 있다. 이를 통해 rand() 함수가 0만 return한 것을 알 수 있고, 성공적으로 elfinject가 적용된 것을 확인할 수 있다. Cyphertext를 획득했으므로 AES 역연산을 수행하는 프로그램을 구현함으로써 비밀키를 획득할 수 있다.

3-3. Inverse AES Operation Program

```
unsigned int key[256] = {0, };

static uint8_t plaintext[16] = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf};

static const uint8_t invSbox[16][16] = {
 {0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb},
 {0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb},
 {0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e},
 {0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25},
 {0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92},
 {0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84},
 {0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06},
 {0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b},
 {0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73},
 {0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e},
 {0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b},
 {0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4},
 {0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f},
 {0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef},
 {0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61},
 {0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d}
};
```

Fig 14. Global Variables of Program

Fig 14에 프로그램에서 사용하는 전역 변수가 제시 되어있다. 먼저 사전에 정의된 평문을 그대로 가져왔으며, Inverse S-box Table을 참고하여 전역 변수로 선언했다. 또한 키를 획득한 후 출력하기에 이를 전역 변수로 선언하였다.

```
void invSubBytes(unsigned int *inv_buf, int count){
        int i = 0;
        while(i < count){
                for(int j = 0; j < 16; j++){
                    if((inv_buf[i] >> 4) == j){
                       for(int k = 0; k < 16; k++){
                           if((inv_buf[i] & 0x0f) == k){
                               inv_buf[i] = invSbox[j][k];
                               break;
                           }
                       }
                       break;
                    }
            }
            i++;
           }
}
```

Fig 15. Inverse Sub Bytes Code

Fig 15에 Inverse Sub Bytes를 수행하는 코드가 제시 되어있다. 먼저, 암호문과 길이를 매개 변수로 받는다. 각 암호문은 1바이트로 count 만큼 구성 되어있으므로 inv_buf의 10의 자리를 bit-shift하여 0x0~0xf까지 일치하는 지 확인한다. 이후 1의 자리와 비교하여 사전에 정의한 Inverse S-box table을 통해 치환하도록 한다.

```
void invShiftRows(unsigned int *inv_buf){
        int count = 0;
        while(count < 4){
          if(count == 0){
                for(int i = 0; i < 4; i++){
                        key[i] = inv_buf[i];
                }
          }
          else if(count == 1){
                for(int i = 4; i < 8; i++){
                        if(i == 7) key[i-3] = inv_buf[i];
                        else key[i+1] = inv_buf[i];
                }
          }
          else if(count == 2){
                for(int i = 8; i <12; i++){
                        if(i == 11 || i == 10) key[i-2] = inv_buf[i];
                        else key[i+2] = inv_buf[i];
                }
          }
          else{
                for(int i = 12; i < 16; i++){
                        if(i == 12) key[i+3] = inv_buf[i];
                        else key[i-1] = inv_buf[i];
                }
         }
         count++;
        }

}
```

Fig 16. Inverse Shift Rows Code

Fig 16에 Inverse Shift Rows에 대한 코드가 제시 되어있다. Shift Rows는 첫 번째 행은 Shift 하지 않고, 두 번째 행부터 왼쪽으로 한 칸 Shift한다. 이후, 세 번째 행은 왼쪽으로 두 칸 Shift 하고, 네 번째 행은 왼쪽으로 세 칸 Shift한다. 이에 따라 Inver Shift Rows는 반대로 오른쪽으로 한 칸, 두 칸, 세 칸 Shift Rows를 수행하면 된다. 이에 따라 해당 함수는 Inverse Sub Bytes된 버퍼를 매개변수로 받아 각 행에 따라 해당 로직에 맞는 Shift를 수행한다.

```
void invXor(unsigned int *inv_buf){
        for(int i = 0; i < 16; i++){
                inv_buf[i] = inv_buf[i] ^ plaintext[i];
        }
}
```

Fig 17. Inverse AddRoundKey Code

마지막으로 Fig 17에 Inverse AddRoundKey에 대한 코드가 제시 되어있다. 기존의 AddRoundKey는 평문과 키를 xor하여 결과를 얻었다. 이를 Inverse 관점에서 생각하여 평문과 AddRoundKey 결과를 xor하면 키를 얻기에 해당 로직으로 코드를 구성하였다.

4. Results

본 장에서는 구현된 코드를 바탕으로 결과를 확인한다.

```
printf("\nSecret Key: ");

int counter = 0;
while(counter < 4){
        if(counter == 0){
                for(int i = 0; i < 16; i += 4){
                        printf("%02x ", key[i]);
                }
        }
        else if(counter == 1){
                for(int i = 1; i < 16; i += 4){
                        printf("%02x ", key[i]);
                }
        }
        else if(counter == 2){
                for(int i = 2; i < 16; i += 4){
                        printf("%02x ", key[i]);
                }
        }
        else if(counter == 3){
                for(int i = 3; i < 16; i += 4){
                        printf("%02x ", key[i]);
                }
        }
        counter++;
}
```

Fig 18. Print Logic Code

해당 프로그램은 Column-Oriented 방식을 통해 출력한다. 왜냐하면 AES의 Key는 Column-Oriented 방식을 사용하기에 실제 출력이 이와 같은 방법으로 출력되도록 Column 방향으로 출력을 하였다.

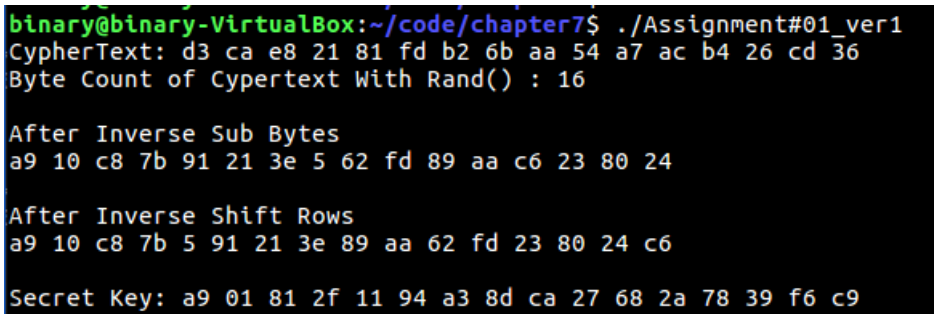

Fig 19. Result of Program

Fig 19에 구현한 프로그램의 결과가 나타나 있다. 먼저 rand() 함수를 무력화 하였기에 암호문이 출력되고 해당 암호문의 크기를 확인할 수 있다. 이후, AES 역연산의 수행 결과를 출력한다. Inverse Sub Bytes를 통한 결과를 출력하고, Inverse Shift Rows에 대한 결과를 출력한다. 그리고 Inverse AddRoundKey 연산을 수행하면 결과적으로 Secret Key가 생성되기에 해당 생성 결과를 출력하여 Secret Key의 결과를 확인할 수 있다.

5. Conclusion

본 레포트에서는 rand() 함수의 동작을 무력화하고, 이를 기반으로 AES 역연산을 수행하여 비밀키를 복구하는 과정을 다루었다. 구체적으로, rand() 함수가 항상 0을 반환하도록 섹션을 구성하였으며, 이를 구현하기 위해 elfinject 기법을 사용하여 코드에 섹션을 삽입했다. 이후 hexedit를 통해 GOT를 확인하고, rand() 함수 호출 흐름을 삽입한 섹션으로 변경하여 동작을 조작하였다. 이러한 과정을 통해 rand() 함수의 무작위성이 제거된 환경에서 암호화된 데이터를 역연산하여 비밀키를 성공적으로 복구할 수 있었다.

AES 역연산 과정에서는 Inverse Sub Bytes, Inverse ShiftRows, Inverse AddRoundKey와 같은 알고리즘의 핵심 원리를 활용하였다. 이러한 역연산은 AES 암호화 과정의 역방향으로 진행되며, 이를 통해 암호화된 데이터를 성공적으로 복호화하고 비밀키를 추출할 수 있었다. 이 과정에서 AES의 동작 원리를 깊이 이해하고, 암호학 알고리즘의 구조적 특성을 파악하는 데 유익한 경험이 되었다.

해당 프로그램을 구현하기 위해, 2장에서 요구사항을 정리하고, 분석에 필요한 기술과 도구를 파악하였다. 3장에서는 제공된 프로그램을 분석 및 구현하였고, 이를 바탕으로 rand() 함수의 동작을 제어하여 AES 암호화 및 복호화 과정을 수행하였다. 4장에서는 구현된 결과를 확인하여, 비밀키 복구와 복호화가 성공적으로 이루어졌음을 검증하였다. 이러한 일련의 과정은 암호학의 실무적 응용과 분석 기술을 익히는 중요한 계기가 되었다.

이번 과제를 통해, 암호 시스템에서 난수 생성기의 중요성과 그 취약점이 시스템 보안에 미치는 영향을 직접적으로 체감할 수 있었다. rand() 함수와 같은 난수 생성기의 취약성은 암호화된 데이터의 보안을 무력화할 수 있음을 확인하였고, 이와 같은 취약점을 악용하는 공격의 원리를 파악할 수 있었다. 특히, elf 구조와 코드 패치 기법, GOT 테이블 분석, AES 암호화 및 역연산의 기술적 이해를 심화할 수 있었으며, 이러한 경험은 향후 심화된 프로그램의 분석 및 역공학 기술

을 다루는 데 중요한 밑거름이 될 것이다.

이번 과제를 수행하면서 기술적 난관을 해결하고 암호 시스템을 분석해 나가는 과정은 도전적이었지만, 이를 통해 문제 해결 능력과 분석 역량을 크게 향상시킬 수 있었다. 특히, AES 역연산을 구현하며 암호학의 구조적 복잡성을 이해하고, 이론을 실제 코드에 적용하는 과정을 경험한 것은 의미 있는 배움이었다.

결론적으로, 과제를 수행하며 레포트 까지의 작성은 rand() 함수의 조작과 AES 복호화 역공학을 통해 암호 시스템의 동작을 이해하고, 취약점이 보안에 미치는 영향을 실질적으로 분석한 데 의미가 있었다.