첨부파일 개요

- `recover_12bytes.py`: AES 전력소모 파형 데이터를 분석하여 비밀키 12바이트를 복구하는 코드
- `recover_4bytes.cpp`: 나머지 4바이트를 무차별 대입으로 복구하는 코드
- `result.py`: 정상 복구를 검증하는 코드
- `AES.cpp, AES.h`: AES의 C++ 구현체

1 목표

본 문제의 목표는 제시된 AES전력소모 파형을 분석하여 비밀키를 복구하는 것이다.

2 12바이트 복구 - Correlation Power Analysis

CPA는 이론 모델과 실제 데이터 사이의 상관계수를 계산하여, 키 후보군 중 가장 높은 상관계수를 가지는 키를 구하는 방법이다.

이를 계산하는 Python 코드는 다음과 같으며, 상관계수가 0.7이 넘는 키 12개를 복구할 수 있었다. 나머지 4개 키는 상관계수가 0.1에 가깝기 때문에 복구가 되지 않은 것으로 간주하였고, 이는 무차별 대입법으로 계산하였다.

```
from scipy.stats import pearsonr
from sbox import SBox
from util import load_wave_file
from joblib import Parallel, delayed

plaintexts = open("2024-contest-sca-pt.txt", "r").readlines()
plaintexts = list(map(lambda pt: bytes.fromhex(pt), plaintexts))

WAVE_NUM, LEN, waves = load_wave_file("2024-contest-sca-tr.bin")

HW = [bin(n).count('1') for n in range(256)]

# search for each byte
def process_data(i):
    # get all ith byte plaintext
    plaintexts_i = [x[i] for x in plaintexts]

    ans_point = 0
    ans_p = 0
    ans_k = 0

    for keyGuess in range(256):
        if keyGuess % 16 == 0:
            print(f'Thread {i}: {keyGuess}')
        hammingPower = [HW[SBox[pt ^ keyGuess]] for pt in plaintexts_i]
        for point in range(LEN):
            actualPower = [wave[point] for wave in waves]
            pGuess = abs(pearsonr(hammingPower, actualPower).statistic)
            if pGuess > ans_p:
                ans_point = point
                ans_k = keyGuess
                ans_p = pGuess

    print(i, ans_point, ans_k, ans_p)
    return ans_k
```

```
results = Parallel(n_jobs=-1, backend='loky')(delayed(process_data)(data) for data in range(16))

print(results)
```

12바이트 복구 결과는 아래와 같으며, 0, 5, 10, 15번째 바이트는 복구되지 못함을 확인할 수 있다.

```
인덱스, 파형에서 위치, 복구 값, 상관계수 순으로 데이터를 나열함
0 842 16 0.13077799285434402
    1 300 50 0.7285700265786639
    2 201 52 0.7530488270812588
    3 102 75 0.7432896726503209
    4 3 114 0.7542411724892242
5 202 165 0.14279682965204238
    6 332 114 0.737164444435327
    7 234 121 0.7385455267518652
    8 135 112 0.756461075376013
    9 36 116 0.7454948505434374
10 106 235 0.1410921570190006
    11 341 83 0.7208636963078783
    12 267 111 0.7501859888116553
    13 168 108 0.7489794586842916
    14 69 118 0.7760135826791072
15 362 181 0.14422015249316208
```

3 4바이트 복구 - Brute Force

복구되지 않은 나머지 키는 아래에 제시된 C++코드로 복구하였다.

```
#include <iostream>
#include "AES.h"

int main() {
    unsigned char plain[] =    {
        0x5F, 0x57, 0x3B, 0xD5,
        0x70, 0x84, 0x53, 0x94,
        0x9A, 0xD5, 0x07, 0x23,
        0xDA, 0x8D, 0x86, 0x3E };
    unsigned char cipher[] = {
        0xe1,0xd7,0xf,0xc4,
        0x52,0xec,0xc4,0x40,
        0x36,0x91,0x93,0xe8,
        0x34,0xfb,0xbb,0xd6
    };
    unsigned char key[] = { 0, 50, 52, 75,
    114, 0, 114, 121,
    112, 116, 0, 83,
    111, 108, 118, 0 }; //key example
    unsigned int plainLen = 16 * sizeof(unsigned char);  //bytes in plaintext
    unsigned char* c;
    bool same;

    AES aes(AESKeyLength::AES_128);  ////128 - key length, can be 128, 192 or 256

    int start;
    scanf("%d", &start);

    for (int i1 = start; i1 < 256; i1++)
    {
        printf("== %d\n", i1);
```

```
        key[0] = i1;
        for (int i2 = 0; i2 < 256; i2++)
        {
            key[5] = i2;
            for (int i3 = 0; i3 < 256; i3++)
            {
                key[10] = i3;
                for (int i4 = 0; i4 < 256; i4++)
                {
                    key[15] = i4;
                    c = aes.EncryptECB(plain, plainLen, key);
                    same = true;
                    for (int i = 0; i < 16; i++) {
                        if (c[i] != cipher[i]) {
                            same = false;
                            break;
                        }
                    }
                    if (same) {
                        printf("%d %d %d %d\n", i1, i2, i3, i4);
                        exit(0);
                    }
                }
            }
        }
    }
}
```

4 결과

```
    K = [
    39, 50, 52, 75,
    114, 67, 114, 121,
    112, 116, 111, 83,
    111, 108, 118, 101]
# Ascii "24KrCryptoSolve"
```

최종적으로 복구된 키는 위과 같으며, 제시된 1000개 평문-암호문 쌍에 대해 검증하여 올바르게 복구되었음을 확인하였다.

```
from Crypto.Cipher import AES
from joblib import Parallel, delayed

# 12바이트 복구
k = [-1, 50, 52, 75, 114, -1, 114, 121, 112, 116, -1, 83, 111, 108, 118, -1]
# 나머지 4바이트 복구
k[0], k[5], k[10], k[15] = 39, 67, 111, 101


cipher = AES.new(bytes(k), AES.MODE_ECB)

plaintexts = open("2024-contest-sca-pt.txt", "r").readlines()
plaintexts = list(map(lambda pt: bytes.fromhex(pt), plaintexts))

ciphertexts = open("2024-contest-sca-ct.txt", "r").readlines()
ciphertexts = list(map(lambda ct: bytes.fromhex(ct), ciphertexts))

for p, c in zip(plaintexts, ciphertexts):
    assert cipher.encrypt(p) == c
```

5 참고 문헌

[1] https://www.tandfonline.com/doi/epdf/10.1080/23742917.2016.1231523?needAccess=true
[2] https://yan1x0s.medium.com/side-channel-attacks-part-2-dpa-cpa-applied-on-aes-attack-66baa356f03f