1차 과제

AES 10라운드 CPA를 통한 키 추출 보고서

2020270103 임현성

2024년 11월 24일

1. 분석 환경 설정

1.1 통신 설정

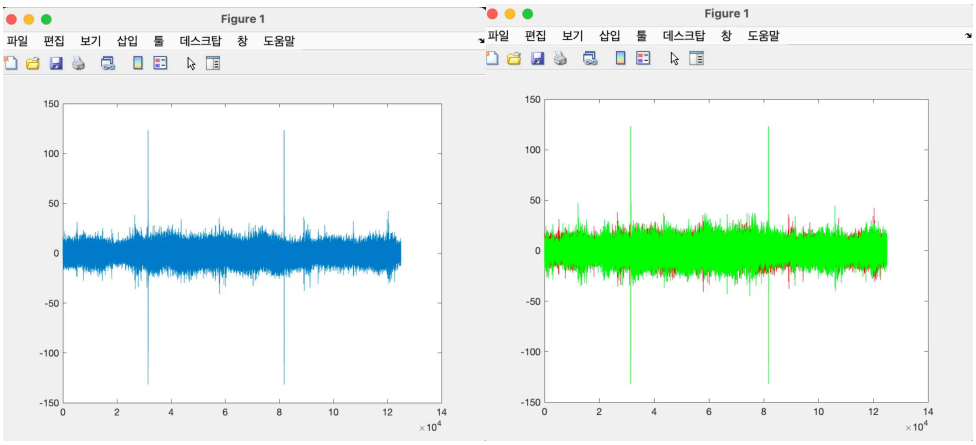
- 오실로스코프와 PC 연결
    - 방식: Ethernet (TCP/IP)
    - 오실로스코프 IP: 192.168.0.129
    - PC IP: 192.168.0.120
    - 포트: 1861
- PC와 아두이노 연결
    - 방식: USB
    - Baudrate: 115200

1.2 신호 설정

- 트리거: 9라운드 addroundkey 시작 시점과 10라운드 addroundkey 끝 시점
- 파형 길이: 파형의 길이는 125002포인트로 설정
- 파형 개수: 10,000개의 파형을 수집

1.3. 데이터 수집 과정

- AES-128 암호화 알고리즘을 사용하여 아두이노에서 암호문 생성
    - 평문: 16바이트 랜덤 데이터
    - 키: 고정된 16바이트 데이터
    - 수집된 파형: 9라운드 AddRoundKey부터 10라운드 AddRoundKey까지
- estraces 라이브러리를 사용하여 평문, 암호문, 키, 파형 데이터를 포함하는 ETS 파일 생성.



트리거를 기준으로 파형이 정렬되어 있는 상태임을 확인

2. 분석 방법론

AES-128 암호화 알고리즘에서 키를 추정하기 위해 CPA를 사용합니다. 이 방법은 S-Box의 출력값을 타겟으로 하여 예상되는 전력 소모량과 파형 사이의 상관관계를 분석함으로써 키를 추측합니다. 일반적으로는 해당 라운드의 평문을 알고 있어야만 S-Box 출력값을 이용해 키를 유추할 수 있습니다.

그러나 이번 실습에서 제공된 파형 데이터는 9라운드의 AddRoundKey부터 10라운드의 최종 AddRoundKey까지의 파형으로 한정되므로, AES 암호화 과정에서 S-Box 출력값을 사용하는 대신, 암호문과 InvS-Box 출력값을 활용하여 복호화 과정 중에 CPA를 수행하여 키를 추정했습니다.

2.1 데이터 전처리와 분석 방법

1. 데이터 전처리

- 제공된 파일은 Python의 estraces 라이브러리를 이용해 평문, 암호문, 파형의 데이터를 담은 ets파일입니다. 이를 C언어 CPA 코드에서 사용 가능한 형식(traces 파일)으로 변환하기 위해 데이터 전처리가 필요합니다.

- Python 코드를 활용하여 plaintext, ciphertext, 그리고 파형 데이터를 각각 다음과 같은 형식으로 변환했습니다:
  - plaintext: plaintext.txt (16진수 형식의 평문 데이터)
  - ciphertext: ciphertext.txt (16진수 형식의 암호문 데이터)
  - 파형 데이터: AES.traces (전력 소비 파형 데이터를 float 형식으로 저장)

```python
def convert_ets_to_traces(ets_file_path, output_traces_path):

    ths = estraces.read_ths_from_ets_file(ets_file_path)

    samples = ths.samples
    trace_count = len(samples)
    trace_length = len(samples[0])

    print(f"Trace Count: {trace_count}, Trace Length: {trace_length}")

    with open(output_traces_path, 'wb') as traces_file:
        traces_file.write(struct.pack('i', trace_length))
        traces_file.write(struct.pack('i', trace_count))

        for index, trace_data in enumerate(samples):
            traces_file.write(struct.pack(f'{trace_length}f', *trace_data))

            if (index + 1) % 100 == 0:
                print(f"Processed {index + 1}/{trace_count} traces")

    print(f"Converted {trace_count} traces to {output_traces_path}")

ets_file_path = "C:\\Users\\lhs\\Desktop\\subchannel\\trc.ets"
output_traces_path = "C:\\Users\\lhs\\Desktop\\subchannel\\AES.traces"

convert_ets_to_traces(ets_file_path, output_traces_path)
```

ets 파일을 traces파일로 변환하는 파이썬 코드 중 일부

2. 분석 방법

CPA로 마스터 키를 추정하려면 일반적으로 1라운드의 S-Box 출력값을 분석해야 합니다. 그러나 이번 실습에서는 제공된 파형 데이터가 9~10라운드에 한정되어 있으므로 복호화 과정에서 InvS-Box 출력값을 활용하여 키를 추정했습니다.

- 암호문을 이용하여 복호화 과정을 연산함으로써 10라운드의 InvS-Box 출력값을 계산하고, 이를 기반으로 10라운드 키를 추정하였습니다.

2.2 분석 과정

AES-128 복호화의 마지막 단계는 다음과 같은 연산으로 구성됩니다.

- 암호문 → AddRoundKey → InvShiftRows → InvSubBytes → 9라운드 데이터

- CPA의 타겟은 InvSubBytes의 출력값이며, 이를 통해 10라운드 키를 추정합니다.

iv (invS-Box 연산 결과)와 hw (해밍 웨이트)를 이용하여 상관계수를 계산합니다.

- S-Box 연산 또는 InvS-Box은 비선형 연산으로 하드웨어가 연산을 수행할 때 특정한 전력 소모 패턴을 보입니다. 이를 이용해 해밍 웨이트를 이용하여 각 키 후보마다 상관계수를 계산합니다.

- 0x00 ~ 0xff 범위의 예상 10라운드 키 (256가지 후보) 중 가장 높은 상관계수를 가지는 키를 CPA로 추정합니다.

```
for (j = 0; j < TraceNum; j++) {
    iv = InvSbox[ciphertext[j][i] ^ key];
    hw_iv = 0;
    for (k = 0; k < 8; k++) {
        hw_iv += ((iv >> k) & 1);
    }
    Sy += hw_iv;
    Syy += hw_iv * hw_iv;
    for (k = startpoint; k < endpoint; k++) {
        Sxy[k] += hw_iv * data[j][k];
    }
}
```

Iv와 hw를 이용하여 상관계수를 연산하는 코드

3. 결과

3.1. 10라운드 키 추출

- CPA를 통해 AES 10라운드의 키를 추정했습니다.

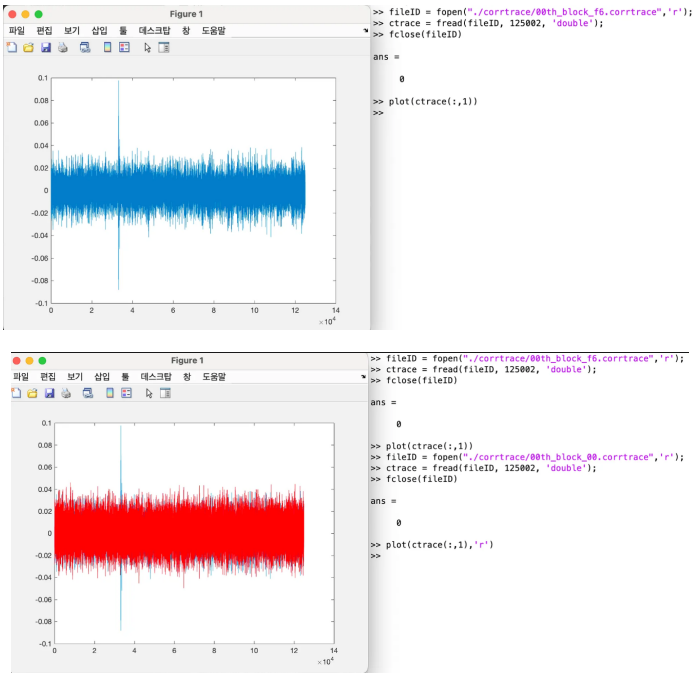
- 10라운드 키 추정 결과: F64E800BD1F9F0A523D54C24AD0297FD

```
1 warning generated.
Key Byte 0: F6, Max Correlation: 0.097620
Key Byte 1: 4E, Max Correlation: 0.101452
Key Byte 2: 80, Max Correlation: 0.107474
Key Byte 3: 0B, Max Correlation: 0.109767
Key Byte 4: D1, Max Correlation: 0.113692
Key Byte 5: F9, Max Correlation: 0.101817
Key Byte 6: F0, Max Correlation: 0.123528
Key Byte 7: A5, Max Correlation: 0.135323
Key Byte 8: 23, Max Correlation: 0.092794
Key Byte 9: D5, Max Correlation: 0.108017
Key Byte 10: 4C, Max Correlation: 0.106711
Key Byte 11: 24, Max Correlation: 0.110305
Key Byte 12: AD, Max Correlation: 0.108368
Key Byte 13: 02, Max Correlation: 0.091062
Key Byte 14: 97, Max Correlation: 0.123101
Key Byte 15: FD, Max Correlation: 0.132112
➜  subchannel
```

3.2 10라운드 키 검증

- 추출된 corrtrace 파형을 확인하여 특정 키를 추정할 때 상관계수가 높게 측정되는 것을 확인했습니다.

- Matlab을 이용하여 상관계수 파형을 확인하였습니다. 키 후보의 corrtrace파형이 다른 키에 비해 높은 정점을 보였습니다. 이를 통해 CPA 결과가 신뢰할 수 있음을 검증했습니다.



(0라운드에서 추정된 후보 키 0xf6의 파형(파랑)과 잘못된 키 0x00의 파형(빨강)

3.3. 마스터 키 복구

- AES 의 키 스케줄링 과정은 AES 알고리즘에서 암호화 및 복호화 시 사용되는 라운드 키들을 생성하는데 사용됩니다. 이 과정을 반대로 수행하면 상위 라운드 키에서 하위 라운드 키(마스터 키)를 계산할 수 있습니다.

- 10 라운드 키에서 키 스케줄링 과정을 거꾸로 적용하여 0 라운드 마스터 키를 복구합니다.

```python
def reverse_key_schedule(round_key: bytes, aes_round: int):
    assert len(round_key) * 8 == 128  # AES-128에 대한 키 길이 확인
    for i in range(aes_round - 1, -1, -1):  # 현재 라운드부터 0까지 반복
        # 현재 라운드 키의 16바이트를 4개 워드로 분리
        a2 = round_key[0:4]
        b2 = round_key[4:8]
        c2 = round_key[8:12]
        d2 = round_key[12:16]

        # 이전 라운드의 워드 d1, c1, b1, a1을 계산
        d1 = xor_bytes(d2, c2)  # d2 XOR c2
        c1 = xor_bytes(c2, b2)  # c2 XOR b2
        b1 = xor_bytes(b2, a2)  # b2 XOR a2
        a1 = xor_bytes(a2, rot_word(sub_word(d1)), rcon[i])
        # a2 XOR RotWord(SubWord(d1)) XOR rcon[i]

        # 이전 라운드의 키를 재구성
        round_key = a1 + b1 + c1 + d1

    return round_key
```

10 라운드 키를 통해 각 라운드 키를 구하는 코드 중 일부

```
 0: 0123456789abcdef123456789abcdef0
 1: 653ec9dfec950430fea15248641d8cb8
 2: c35aa59c2fcfa1acd16ef3e4b5737f5c
 3: 4888ef4967474ee5b629bd01035ac25d
 4: feada33299eaedd72fc350d62c99928b
 5: 00e29e4399087394b6cb23429a52b1c9
 6: 202a43fbb922306f0fe9132d95bba2e4
 7: 8a102ad133321abe3cdb0993a960ab77
 8: da72df02e940c5bcd59bcc2f7cfb6758
 9: cef7b51227b770aef22cbc818ed7dbd9
10: f64e800bd1f9f0a523d54c24ad0297fd
```

코드 실행 결과

마스터키: 0123456789abcdef123456789abcdef0

이번 실습에서 CPA를 통해 10라운드 키를 성공적으로 추출하고, 이를 역추적하여 마스터 키까지 복구할 수 있음을 확인했습니다.