LEA에 대한 CPA공격

IT융합공학부 김현준

https://youtu.be/nTMoks9eJOM

부채널 전력 분석 공격

- 단순 전력 분석(simple Power Analysis)
  단 몇 번의 암호 알고리즘이 수행되는 동안 소비되는 전력 패턴을 관찰하여 직접 분석

- 차분 전력 분석(Differential Power Analysis)
  다수의 전력 소비 패턴을 사용하여 통계적으로 비밀키 값 분석

- 템플릿 분석(Template Attack)
  해당 디바이스의 전력소모 패턴을 사전지식으로 활용하여 비밀키를 분석

CPA 공격

- 상관 전력 분석(Correlation Power Analysis, CPA)

  전력 소비 모델과 수집파형의 포인트별 상관계수를 계산하는 공격
  추측키가 비밀키와 동일 할 경우 가장 높은 상관계수를 보인다.

  단계 1. 알고리즘의 공격지점 설정

  단계 2. 전력파형 수집

  단계 3. 중간 값 추측

  단계 4. 전력소비모델로 변환

  단계 5. 피어슨 상관계수를 활용 중간 값과 소비전력 파형을 비교

CPA 공격 – AES

단계 1. 알고리즘의 공격지점 설정

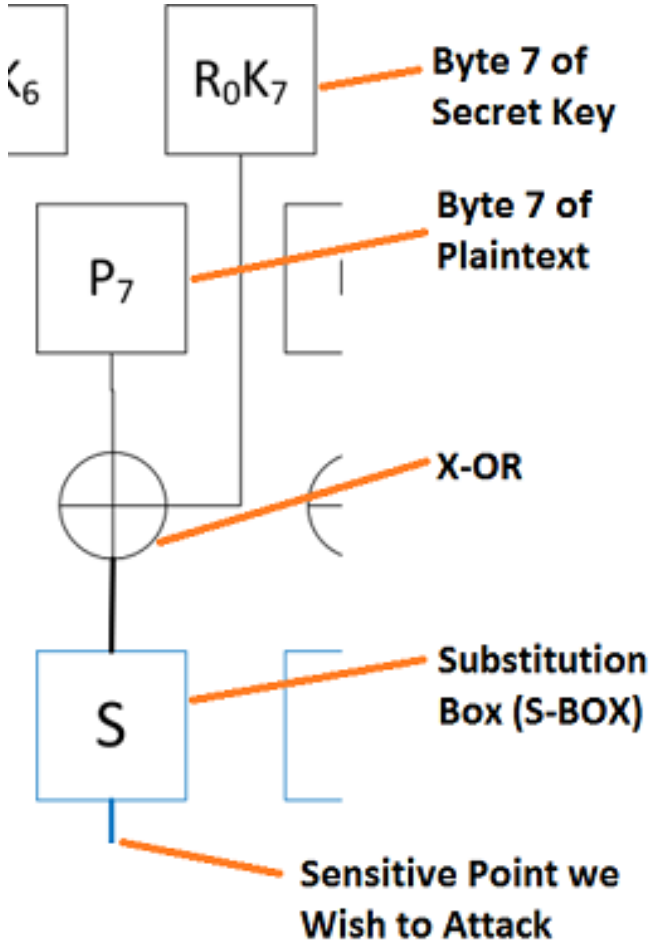
- 추측하고자 하는 비밀키와 공격자가 조작 가능한 정보로 이루어진 부분을 공격지점으로 설정

단계 2. 전력파형 수집

- 중간 값 연산이 수행되는 부분의 소비전력을 수집

단계 3. 중간 값 추측

- 발생 가능한 모든 추측키 집합
- AES는 8비트 부분키를 추측 하므로 0 ~ 255까지 256개



CPA 공격 – AES

단계 4. 전력소비모델로 변환

- 헤밍웨이트 모델로 변환
- HammingWeight(sbox[pt ^ keyguess])

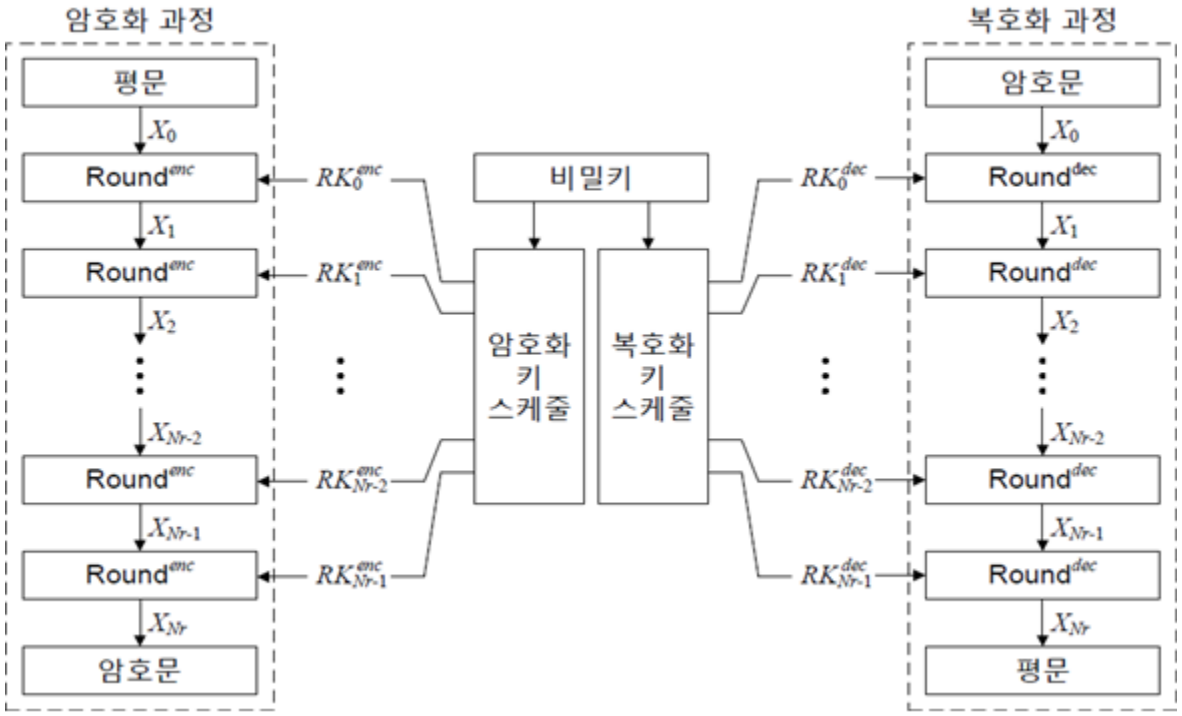
단계 5. 상관계수를 활용 중간 값과 소비전력 파형비교

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sqrt{E[(X-\mu_X)^2]E[(Y-\mu_Y)^2]}}$$

```
def intermediate(pt, keyguess):
    return sbox[pt ^ keyguess]
for bnum in range(0, 16):
    cpaoutput = [0]*256
    maxcpa = [0]*256
    for kguess in range(0, 256):
        print "Subkey %2d, hyp = %02x: "%(bnum, kguess),


        #Initialize arrays & variables to zero
        sumnum = np.zeros(numpoint)
        sumden1 = np.zeros(numpoint)
        sumden2 = np.zeros(numpoint)

        hyp = np.zeros(numtraces)
        for tnum in range(0, numtraces):
            hyp[tnum] = HW[intermediate(pt[tnum][bnum], kguess)]


        #Mean of hypothesis
        meanh = np.mean(hyp, dtype=np.float64)

        #Mean of all points in trace
        meant = np.mean(traces, axis=0, dtype=np.float64)

        #For each trace, do the following
        for tnum in range(0, numtraces):
            hdiff = (hyp[tnum] - meanh)
            tdiff = traces[tnum,:] - meant

            sumnum = sumnum + (hdiff*tdiff)
            sumden1 = sumden1 + hdiff*hdiff
            sumden2 = sumden2 + tdiff*tdiff

        cpaoutput[kguess] = sumnum / np.sqrt( sumden1 * sumden2 )
        maxcpa[kguess] = max(abs(cpaoutput[kguess]))

        print maxcpa[kguess]
```

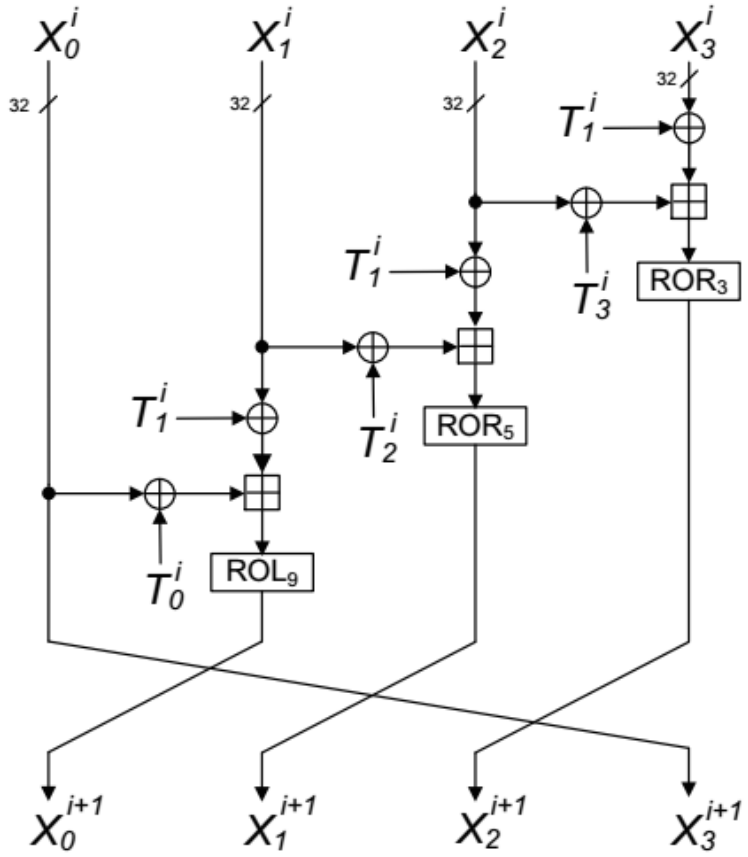
LEA 알고리즘

- 블록 암호 LEA는 128비트 데이터 블록을 암호화하는 알고리즘
- S-box의 사용을 배제하여 경량 구현
- 라운드 함수는 32비트 단위의 ARX(Addition, Rotation, XOR) 연산만으로 구성하여, 이 연산을 지원하는 32비트 플랫폼에서 고속으로 동작



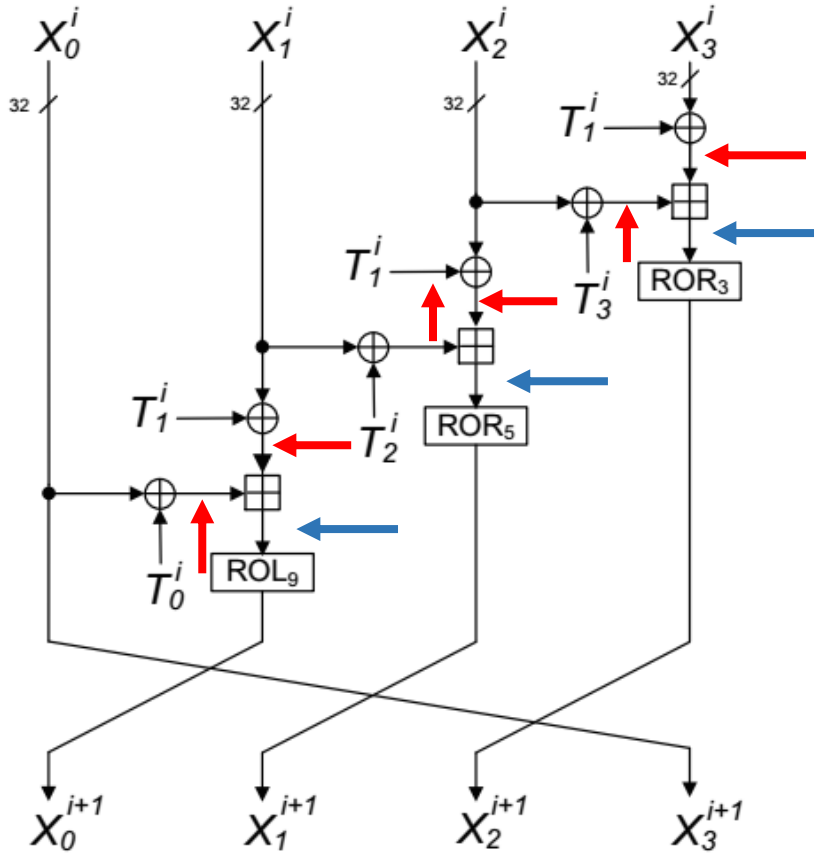
LEA 알고리즘 공격방법 1

- 한 라운드에서 사용되는 서브 키의 두 쌍을 찾아내는 방법
- X_0^0와 X_0^1 값을 전력 소비 모델로 설정하여 해밍 거리를 계산 (T_0^0 ,T_1^0)
- 32비트의 키를 8비트씩 나누어 추측 키로 사용



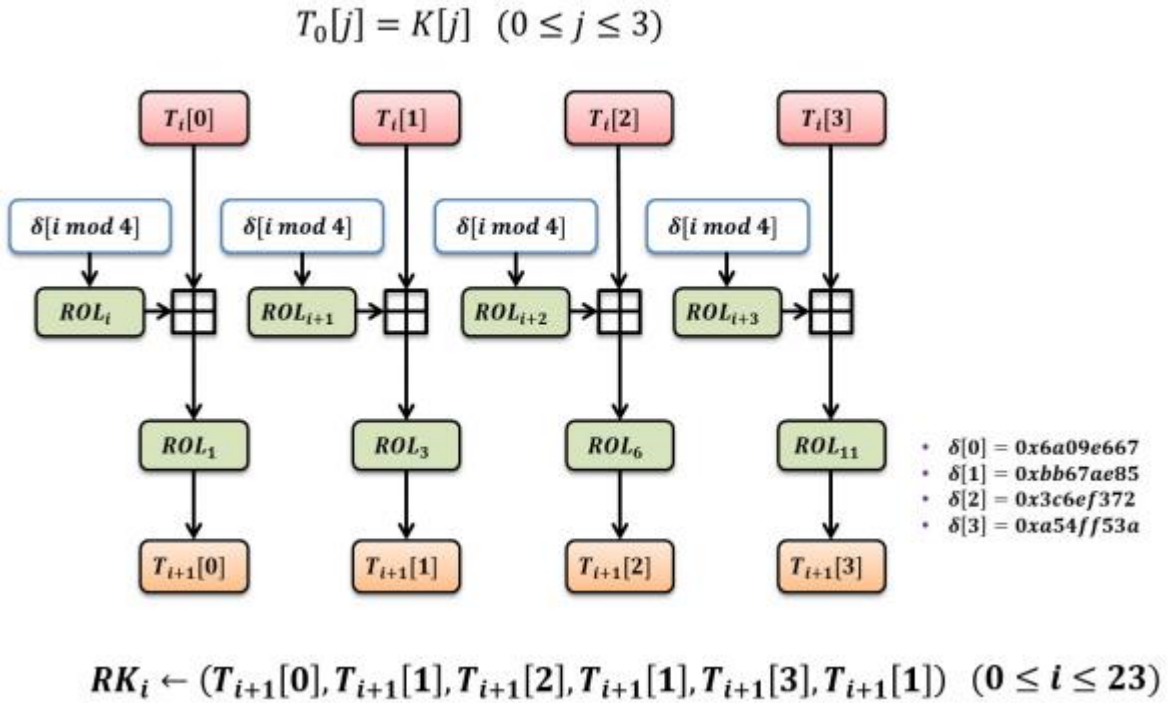
LEA 알고리즘 공격방법 2

- 평문을 이용하여 ⊞연산이 사용되는 부분을 8비트씩 추측
- ⊕연산이 일어난 지점을 중간 값으로 하여 분석
- ⊞연산이 일어난 지점을 중간 값으로 하여 분석



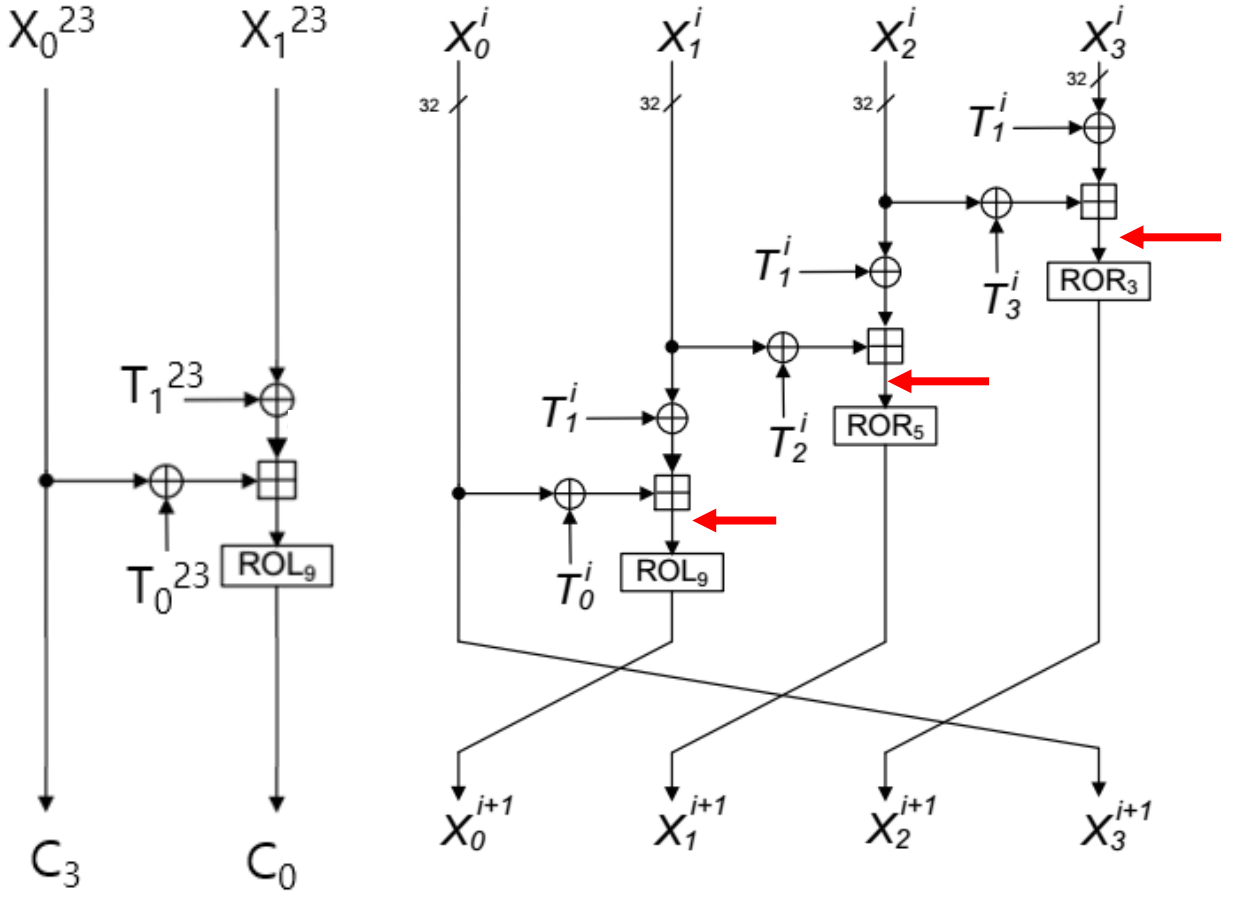
LEA 알고리즘 공격방법 3

- 8비트씩 추측 평문과 암호문 사용
- 마지막 라운드에서부터 공격 수행
- 출력된 암호문 C_0 과 C_3 을 사용
- $ROR_9(C_0) \boxminus (C_3 \oplus \hat{k})$ 연산 을 수행
  $=T_0^{23}$ 을 찾아냄
- 역으로 연산 하여 첫 번째 라운드 키 획득
- 획득 한 키를 활용하여 나머지 키 분석
  $(X_0^0 \oplus T_0^0) \boxplus (X_1^0 \oplus \hat{k})$



LEA 알고리즘 공격방법 3

- 8비트씩 추측 평문과 암호문 사용
- 마지막 라운드에서부터 공격 수행
- 출력된 암호문 C_0 과 C_3 을 사용
- $ROR_9(C_0) \boxminus (C_3 \oplus \hat{k}) = T_0^{23}$ 연산 을 수행 $\hat{k}$ 을 찾아냄
- 역으로 연산 하여 첫 번째 라운드 키 획득
- 획득 한 키를 활용하여 나머지 키 분석
  $(X_0^0 \oplus T_0^0) \boxplus (X_1^0 \oplus \hat{k})$



감사합니다