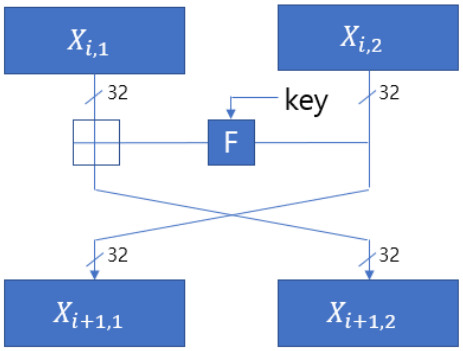
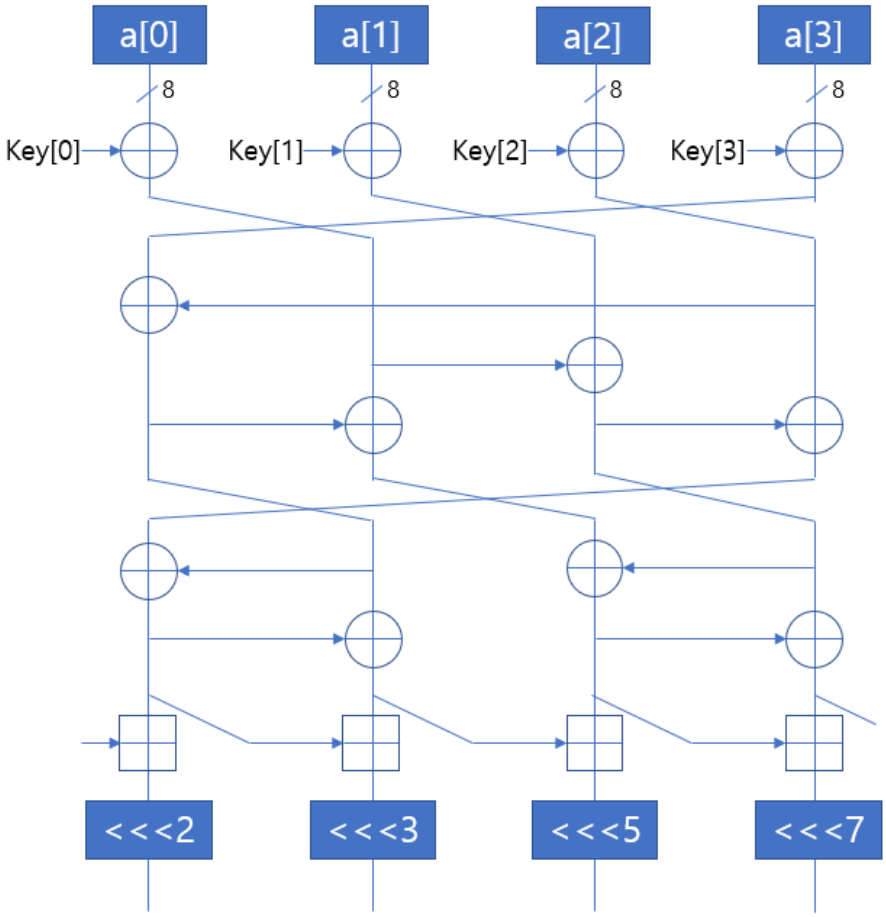
2019 암호경진대회 : 5번 문제 답안

답)

1. 암호 알고리즘 구조



한 라운드는 왼쪽의 그림처럼 생겼고 16라운드 반복된다.
F 함수는 아래 그림처럼 생겼다.



2. 비밀 키

개요 :

조건 1. 제공되는 정보는 8비트 기반 소프트웨어로 구현된 암호 알고리즘이 Atmel XMEGA128 8-bit Microcontroller에서 동작할 때 소비되는 전력을 수집한 결과이다. 1회 암호화 연산이 수행되는 부분을 대상으로 수집하였다.

조건 2. 암호는 64비트 데이터 입출력, 64비트 비밀 키를 사용하는 비공개 블록 암호이다. 라운드 키 생성함수는 따로 존재하지 않으며, 64비트 비밀 키를 32비트씩 끊어서 라운드 키로 반복 사용한다.

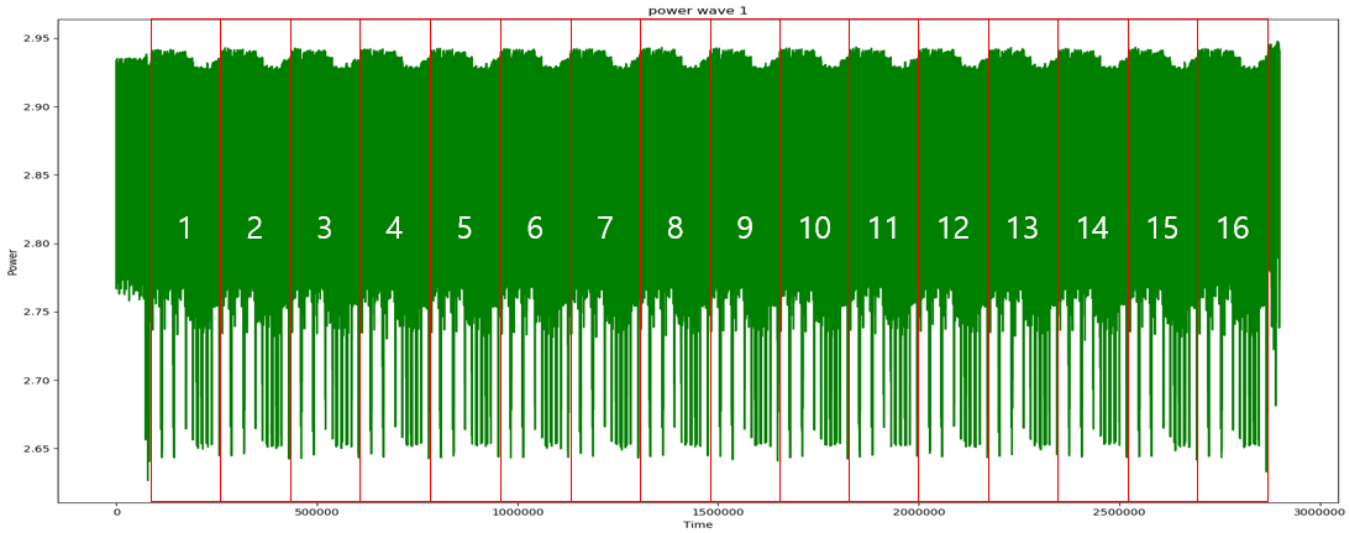
제공되는 동일한 비밀 키로 500개의 서로 다른 평문을 암호화하는 과정에 수집된 소비 전력 및 사용된 평문, 암호문 정보를 이용하여 암호 알고리즘 구조 및 암호 알고리즘 동작 시 사용된 비밀 키를 찾아야 한다.

풀이 :

0. 파형 수와 파형의 포인트 수

주어진 파일 'Context_SCA_problem.trace'의 HEADER 정보를 통해 파형의 수는 500개이며, 파형의 포인트 수는 2,900,000개임을 알 수 있었다.

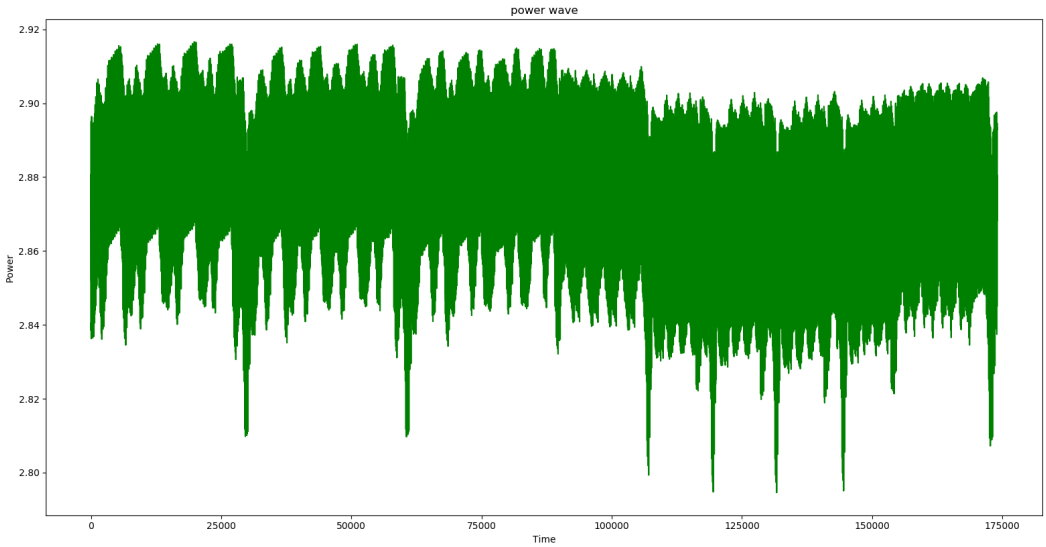
1. 라운드 수 찾기



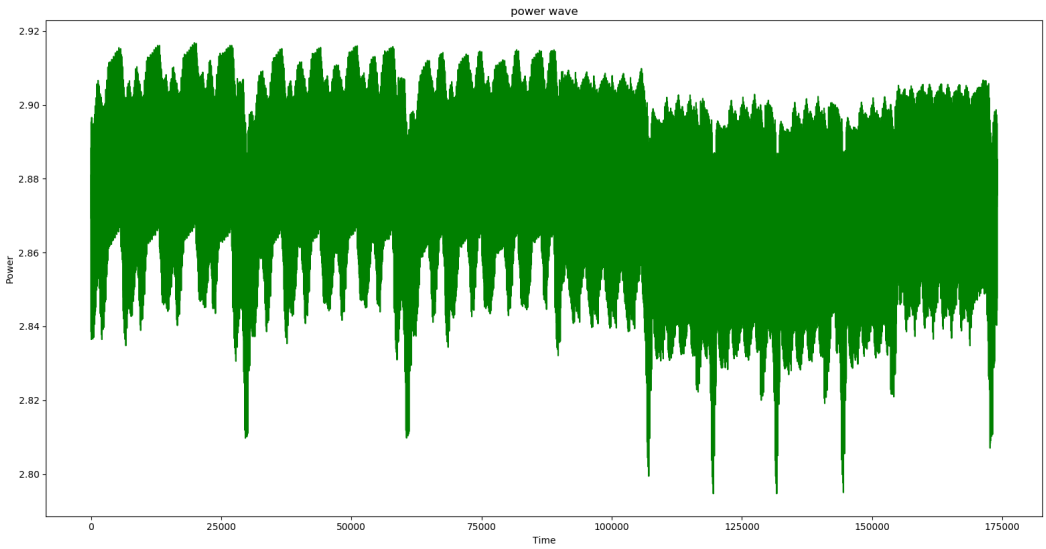
< 그림 1. 첫 번째 평문 암호문의 암호 알고리즘이 시행된 소비 전력 >

파형으로 미루어 보아 16라운드로 추측하였다.

2. 구조 예측하기

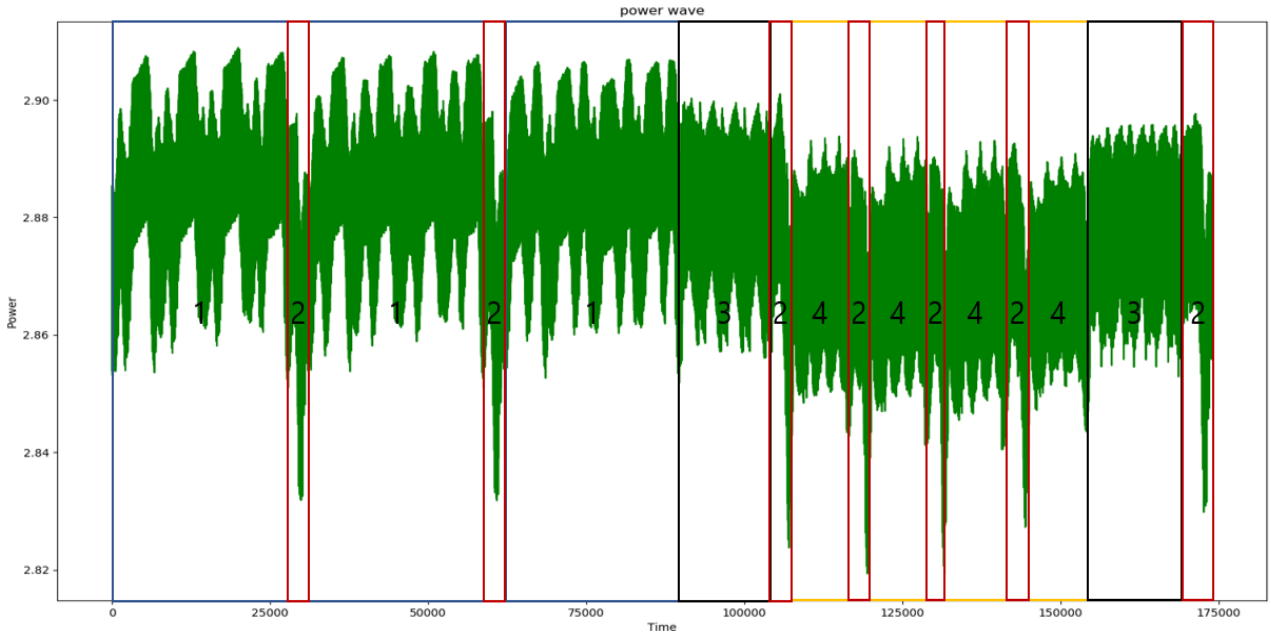


< 그림 2. 500개 홀수 라운드 전력 평균 >



< 그림 3. 500개 짝수 라운드 전력 평균 >

그림 2, 그림 3을 통해 홀수 라운드, 짝수 라운드에서의 과정이 다르지 않을 것이라고 추측할 수 있다.



< 그림 4. 500개 모든 라운드 전력 평균 >

그림 4는 파형을 보고 비슷한 연산으로 추정되는 구간끼리 1, 2, 3, 4번으로 나누어 본 것이다. 1, 3번은 한 구간 안에서 같은 동작이 네 번 발생하고, 3번은 네 번의 동작이 발생하나 같은 동작인지는 알 수 없다.

2-1. Feistel 구조

조건 2에서 암호는 입출력 데이터가 64비트이고 한 라운드에서 32비트 라운드 키를 사용하는 블록 암호이므로 Feistel 구조를 예측하였다.

2-1-1. 2번 예측

Feistel 구조라면 마지막 연산이 swap(대입 연산)일 것이므로 2번을 대입 연산으로 추측한다.

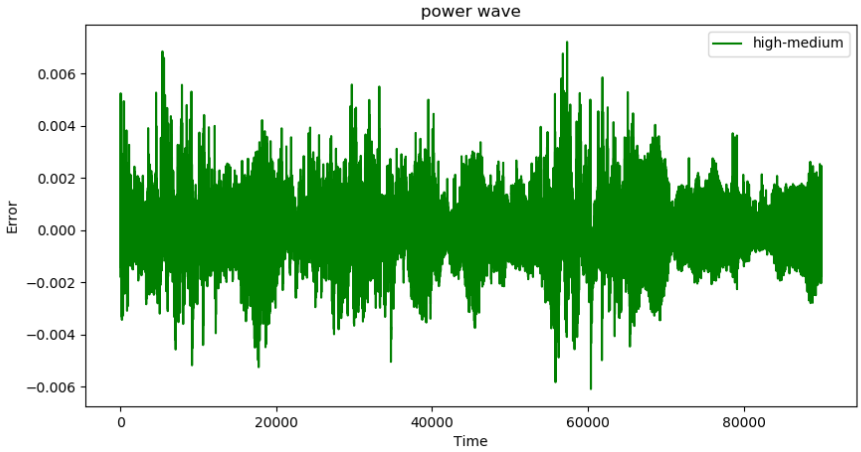
2-1-2. 3번 예측

swap 직전의 연산이 xor 혹은 addition일 것이다. 만약 3번이 xor이라고 가정하면 두 번째 3번은 상위 32비트와 하위 32비트의 xor일 것이고 첫 번째 3번은 key xor일 것이다. 하지만 이 경우 첫 라운드에서 key xor 이전의 연산들이 의미가 없어지므로 3번은 xor이 아닐 것이라 판단했다. 따라서 3번을 addition으로 추측하였다.

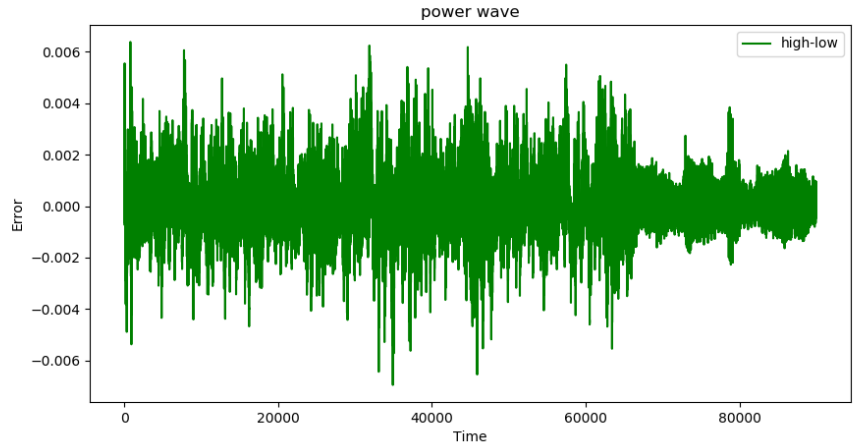
3번은 한 구간 안에서 같은 동작이 네 번 발생하므로 8비트끼리 addition이 4번 발생하는 것으로 생각할 수 있다.

2-1-3. Hamming Weight

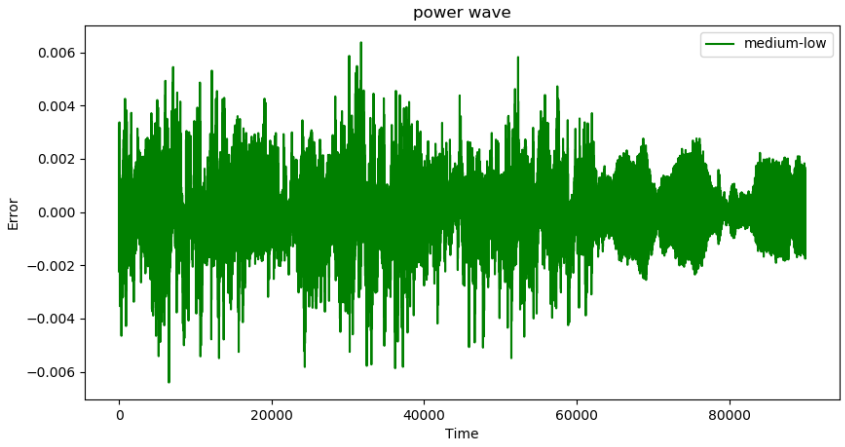
페이스텔 구조라면, round 시작 전 PT가 저장되어 F 함수로 들어가는 부분이 존재한다. 그렇다면 라운드가 시작되기 전에 HW에 따라서 전력소모의 차이가 나는 부분이 있을 것이라 생각했다. 그래서 주어진 500개 PT의 5번째 바이트의 HW를 기준으로 HW가 4보다 큰 PT(high), HW가 4인 PT(medium), HW가 4보다 작은 PT(low)으로 나누어 각각에 해당하는 파형들끼리 평균 내어 라운드가 시작되기 전 부분만 그래프로 그려보았다. 각각의 파형에서는 특별히 다른 점을 찾을 수 없어 high와 medium의 차이, high와 low의 차이, medium과 low의 차이를 각각 그래프로 그려보았다.



< 그림 5. high-medium 그래프 >



< 그림 6. high-low 그래프 >



< 그림 7. medium-low 그래프 >

앞과 중간 부분에서는 특별한 차이를 보이지 않지만, x좌표가 80000인 지점에서는 조금의 차이가 보였다. high-low가 high-medium 보다 더 튀는 그래프 형태를 가지고 있어 그 지점에서 함수호출이 발생하고, PT를 저장하지 않을까 추측해 볼 수 있었다.

2-2. 1번 예측

만약 첫 번째 1번이 key xor이 아니라면 첫 라운드에서 초반 부분은 의미가 없어 한 라운드를 버리는 꼴이 된다. 즉, 효율적이지 못하다. 따라서 첫 번째 1번은 key xor이어야한다.

1번도 한 구간 안에서 같은 동작이 네 번 발생하므로 8비트끼리 xor이 4번 발생하는 것으로 생각할 수 있다.

2-3. sbox의 유무

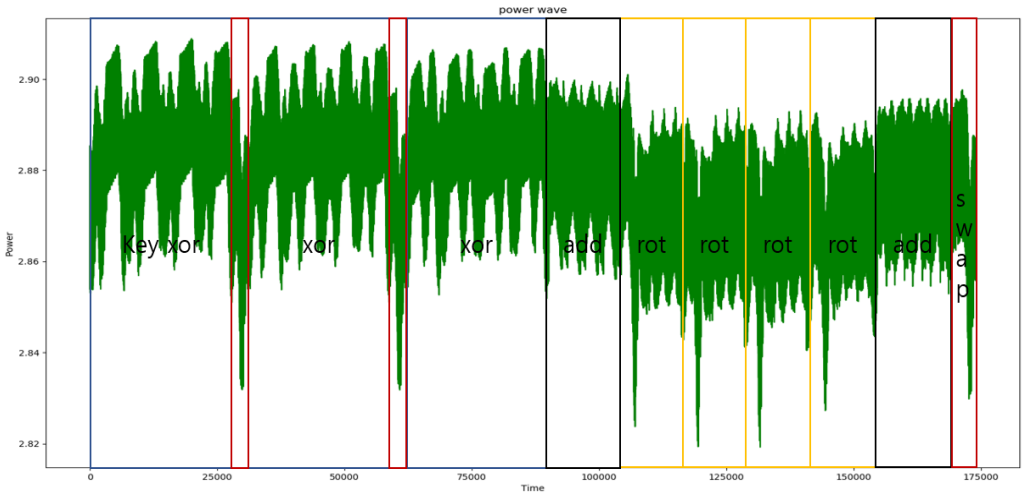
4번이 sbox라고 가정하자. 조건 1을 생각하여 같은 8비트 기반 소프트웨어로 구현된 Arduino UNO R3가 동작할 때에 각각의 연산들이 걸리는 시간을 측정해보았다(문제에서 주어진 Atmel XMEGA128 8-bit Microcontroller를 제조한 곳과 같은 회사이기 때문에 유사할 것이라고 판단하였다). 측정 기기가 다르기 때문에 측정한 시간을 그대로 이용하는 것이 아니라 각 연산들 중 어느 것이 더 빠른지를 알아보고 그것을 이용하였다. 이로 측정한 sbox의 연산 시간은 3번 addition보다 길기 때문에 4번은 sbox가 아니다.

따라서 이 암호에서 sbox는 존재하지 않으며, sbox가 없는 ARX 구조 즉, 경량암호라고 생각하였다. ARX 구조를 예측하고 LEA, HIGHT 등의 암호 알고리즘 구조들을 보며 아이디어들을 얻었다.

2-4. 4번 예측

2-3에 의해 ARX 구조를 예측하였다. 현재 rotation만 없을 뿐이다. Atmel XMEGA128 8-bit Microcontroller에는 따로 rotation 함수가 구현되어있지 않다. 따라서 4번을 shift와 or로 생각하면 2번과 4번을 합친 것이 rotation으로 만들 수 있다. shift와 or만으로 rotation을 하기 위해서 shift의 방향은 왼쪽임을 알 수 있다.

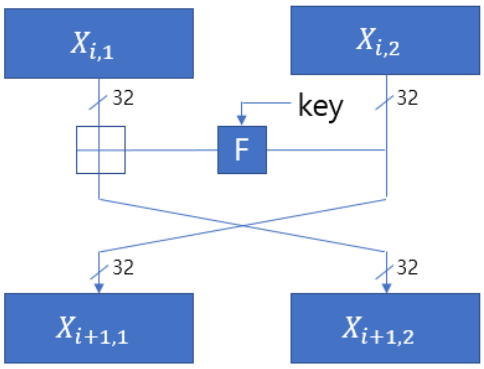
위 예측들을 토대로 1번은 xor, 2번은 대입연산, 3번은 addition, 4번은 shift와 or이라 판단하였다. Arduino UNO R3가 동작할 때에 각각의 연산들이 걸리는 시간을 측정하여 비교한 결과, 위의 추측을 부정할 수 없었다.



< 그림 8. 한 라운드 안에서의 연산 예측 >

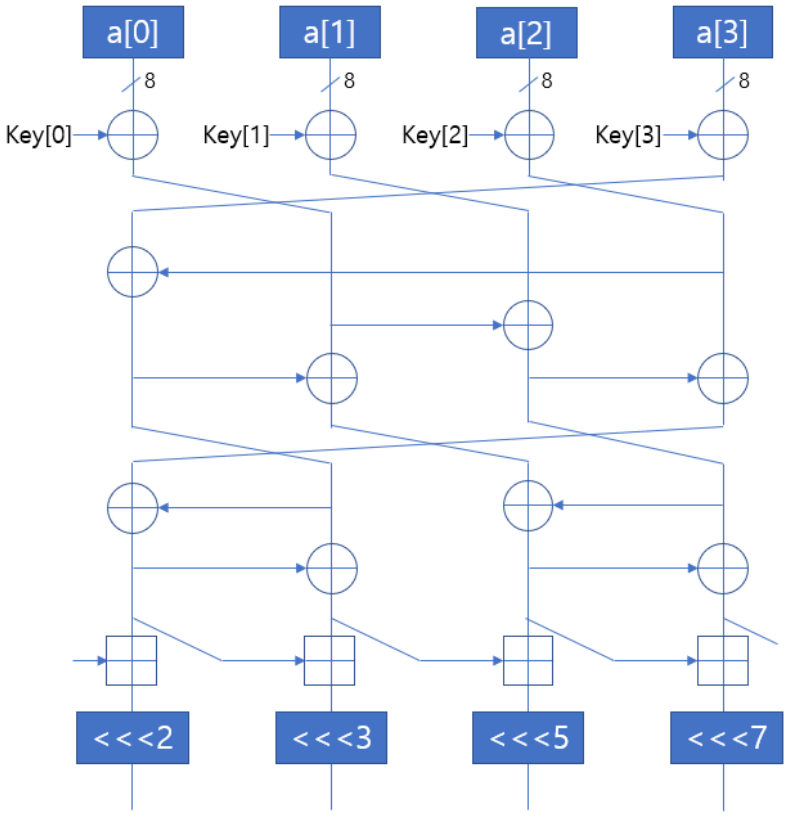
3. 연산을 바탕으로 암호 알고리즘 예측하기

3-1. Feistel 구조



< 그림 9. 한 라운드의 구조 >

이 한 라운드가 16번 반복된다.



< 그림 10. F 함수의 구조 >

첫 번째 key xor 부분은 간단하게 해결된다. 그 다음 8비트 xor 연산이 8번이 나와야 한다. 위의 그림과 같은 방법으로 xor 구조를 짜면 잘 섞어지고 차분이 들어와도 차분이 잘 퍼진다.

key xor 이후의 부분을 X_1, X_2, X_3, X_4라 하자.

첫 번째 swap 이후, X_4, X_1, X_2, X_3

위의 두 번의 xor 연산을 하면, $X_3 \oplus X_4, X_1, X_1 \oplus X_2, X_3$

다음 두 번의 xor 연산을 하면, $X_3 \oplus X_4, X_1 \oplus X_3 \oplus X_4, X_1 \oplus X_2, X_1 \oplus X_2 \oplus X_3$

두 번째 swap 이후, $X_1 \oplus X_2 \oplus X_3, X_3 \oplus X_4, X_1 \oplus X_3 \oplus X_4, X_1 \oplus X_2$

다음 두 번의 xor 연산을 하면, $X_1 \oplus X_2 \oplus X_3, X_3 \oplus X_4, X_1 \oplus X_3 \oplus X_4, X_1 \oplus X_2$

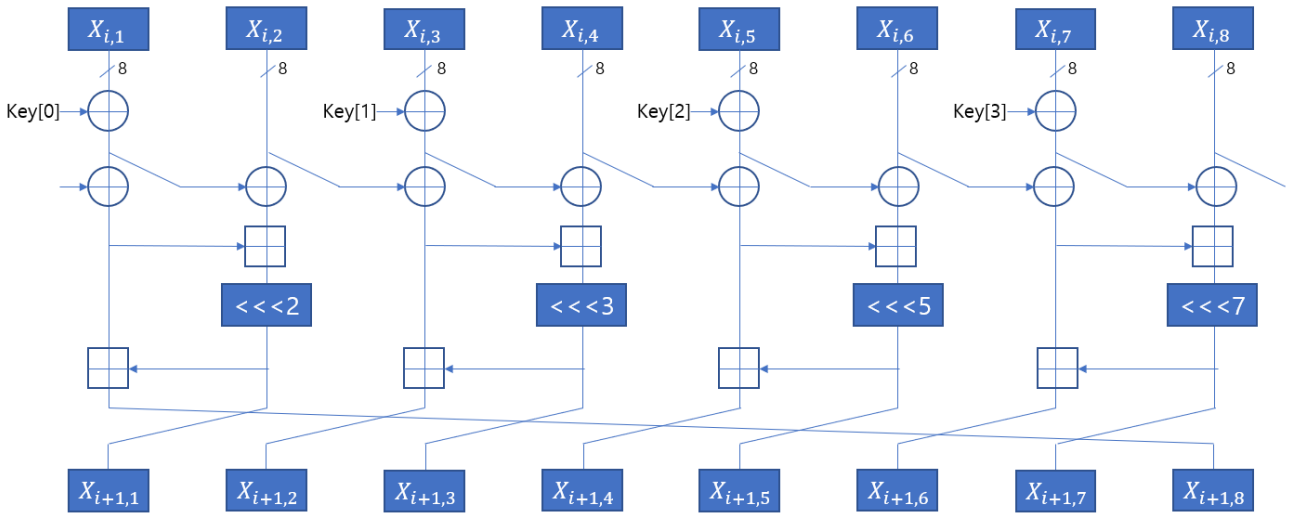
다음 두 번의 xor 연산을 하면, $X_1 \oplus X_2 \oplus X_4, X_1 \oplus X_2 \oplus X_3, X_2 \oplus X_3 \oplus X_4, X_1 \oplus X_3 \oplus X_4$

즉, 8비트 xor 8번을 통해 각 자리에 3개의 XOR 값들을 고르게 섞을 수 있다.

addition은 가볍게 그려볼 수 있다.

rotation은 모두 같은 부분만큼 rotation 하는 것은 바람직하지 못하므로 임의로 숫자를 각각 부여하였다.

3-2. GFN 구조



< 그림 11. GFN 구조로 예측했을 경우 >

key XOR은 간단하게 홀수 박스에만 넣었다. 위와 같이 XOR 하기 위해서는 초깃값을 저장해야 한다. 그 저장하는 부분이 전력 그래프에서 XOR들 사이에 있는 빨간 박스라고 생각하였다.

addition은 간단하게 추측하였다.

rotation 또한 3-1과 같은 논리로 추측하였다.