8라운드 DES(Data Encryption Standard) 암호

차분 공격 구현 보고서

Sepaper

서론

DES(Data Encryption Standard)는 블록 암호 중 하나로, 56bit 키를 이용해 64bit 평문을 암호화하여 64bit 암호문을 만들어낸다. 짧은 키를 사용하기 때문에 키에 대해 brute-force 공격을 하여, 2^{56}의 복잡도로 손쉽게 키를 획득할 수 있는 문제점을 가지고 있다. 하지만, 16 Full round DES는 암호 자체의 취약점을 찾아 공격하는 Cryptanalysis 공격에 대해서 brute-force 공격과 맞먹는 수준의 복잡도를 보여 잘 설계되었다고 평가받는다. 따라서, DES에 대해 Cryptanalysis 공격 중 하나인 차분공격을 구현해보는 것이 공부에 도움이 될 것이라 생각하였다. 성능과 자원이 부족한 환경을 고려하여 16라운드 DES 대신 8라운드 DES를 선택하여 C++으로 차분공격을 구현하였다.

이론적 배경

DES는 Feistel 구조를 가지고 있어 암호화나 복호화가 크게 라운드 함수 F 연산, F의 결과가 left half와 XOR되는 연산으로 구성된다고 볼 수 있다. Feistel 구조를 이용한 암호는 암호화와 복호화의 알고리즘이 같아 구현이 쉽다는 장점이 있지만, 평문에서 암호문 방향으로, 암호문에서 평문 방향으로, 위 두 방향을 이용해 정보들을 손쉽게 얻을 수 있다.

차분공격은 Chosen Plaintext attack(CPA) 환경에서 이루어지는 것으로, n비트 입력 X와 또 다른 입력 X'이 있을 때, 이 둘의 XOR차이(차분)를 ΔX라 하고, 각 입력들에 대응되는 출력 Y와 Y'의 차분을 ΔY라 했을 때, 위키백과에 따르면, 어떤 ΔX에 대해 특정 ΔY가 나타날 확률은 이상적인 경우에 $1/2^n$이 된다. 그렇지 않고 특정 ΔY가 나타날 확률이 다른 ΔY보다 높게 나타난다면, 이 공격에 취약하다고 볼 수 있다. DES에 대한 차분공격은 라운드 함수 F 안에 존재하는 S-Box 치환(Substitution)과정이 위와 같은 취약점을 가지고 있는지 살피고, 그런 취약점을 이용해 공격을 전개한다.

Permutation

DES에서는 Diffusion을 위해 입력의 비트들의 순서를 바꾸거나 비트들을 중복해 사용하여 길이를 늘리는 Permutation 연산을 사용한다. S-Box 이전에 Expansion(E), S-Box 이후에 Permutation(P)와 같은 Permutation 과정이 존재하기 때문에, 입력의 비트가 위 연산들에 의해 어

디로 이동하여, 출력의 어느 비트에 영향을 주는지 잘 살펴야 한다.

참고한 논문 [1]에서는 DES에서 제일 처음과 끝에 있는 Initial Permutation(IP)와 Inverse IP가 공격 자체에 아무런 영향이 없기 때문에 위 두 연산을 고려하지 않았으나, 이 보고서에서는 순수 DES 자체를 공격하는 것을 목표로 하였기 때문에, IP와 Inverse IP를 고려하였다.

사용한 표기

이 보고서에서는 [1]과 [2]에서 사용한 표기들을 혼합하여 사용한다.

$X^{(1)}, X^{(2)}, X'$: 어떤 한 쌍의 비트들을 의미한다. X1과 X2를 XOR연산한 결과인 차분은 X′이다.

X_L, X_R : X의 left half와 right half

$IP(X)$: Initial Permutation (IP)

$IP^{-1}(X)$: Inverse initial Permutation (IP)

$F(X)$: 라운드함수 F

$P(X)$, $inv\_P(X)$: 라운드함수 F의 Permutation P와 inverse P

$E(X)$: 라운드함수 F의 Expansion E

$P^{(1)}, P^{(2)}, P'$: 평문 쌍과 차분

$C^{(1)}, C^{(2)}, C'$: 암호문 쌍과 차분

$a, b, \dots h$: 각 라운드에서 F의 입력

$A, B, \dots H$: 각 라운드에서 F의 출력

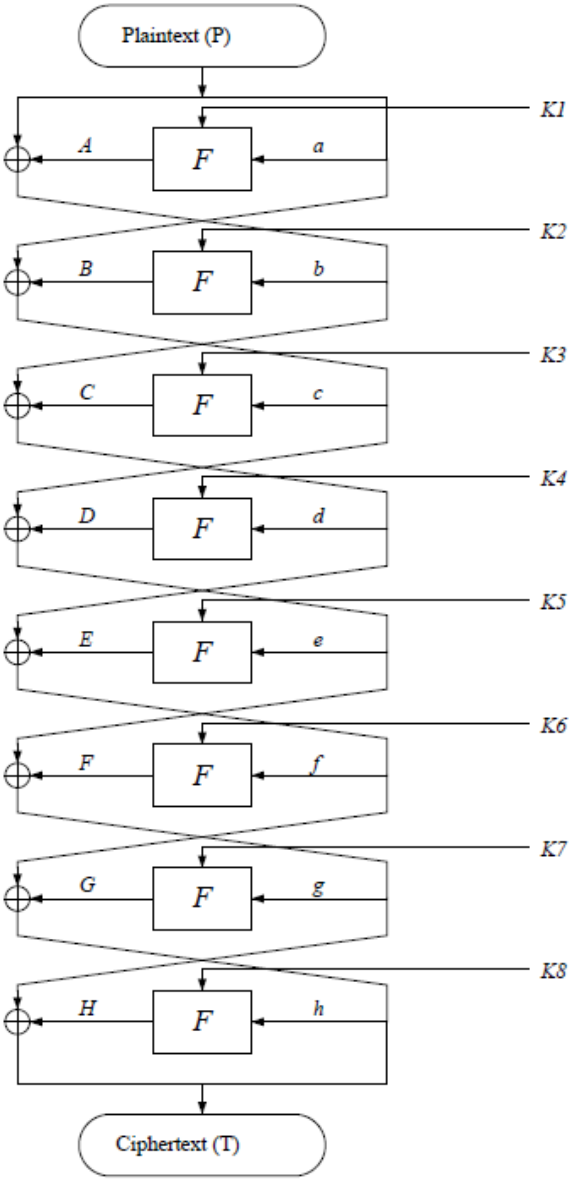
$S[j](X)$: X가 입력일 때, S-Box j에서의 출력

$S_{EX}[j], K_X[j], S_{IX}[j], S_{OX}[j]$: 각각 라운드 X에서의 j번째 Expansion 이후 결과, 서브키, S-Box 입력, S-Box 출력을 의미. $X = \{a, b, \dots . h\}, j = \{1,2, \dots 8\}$

$$S_{EX}[j] \oplus K_X[j] = S_{IX}[j]$$

$$S[j](S_{IX}[j]) = S_{OX}[j]$$

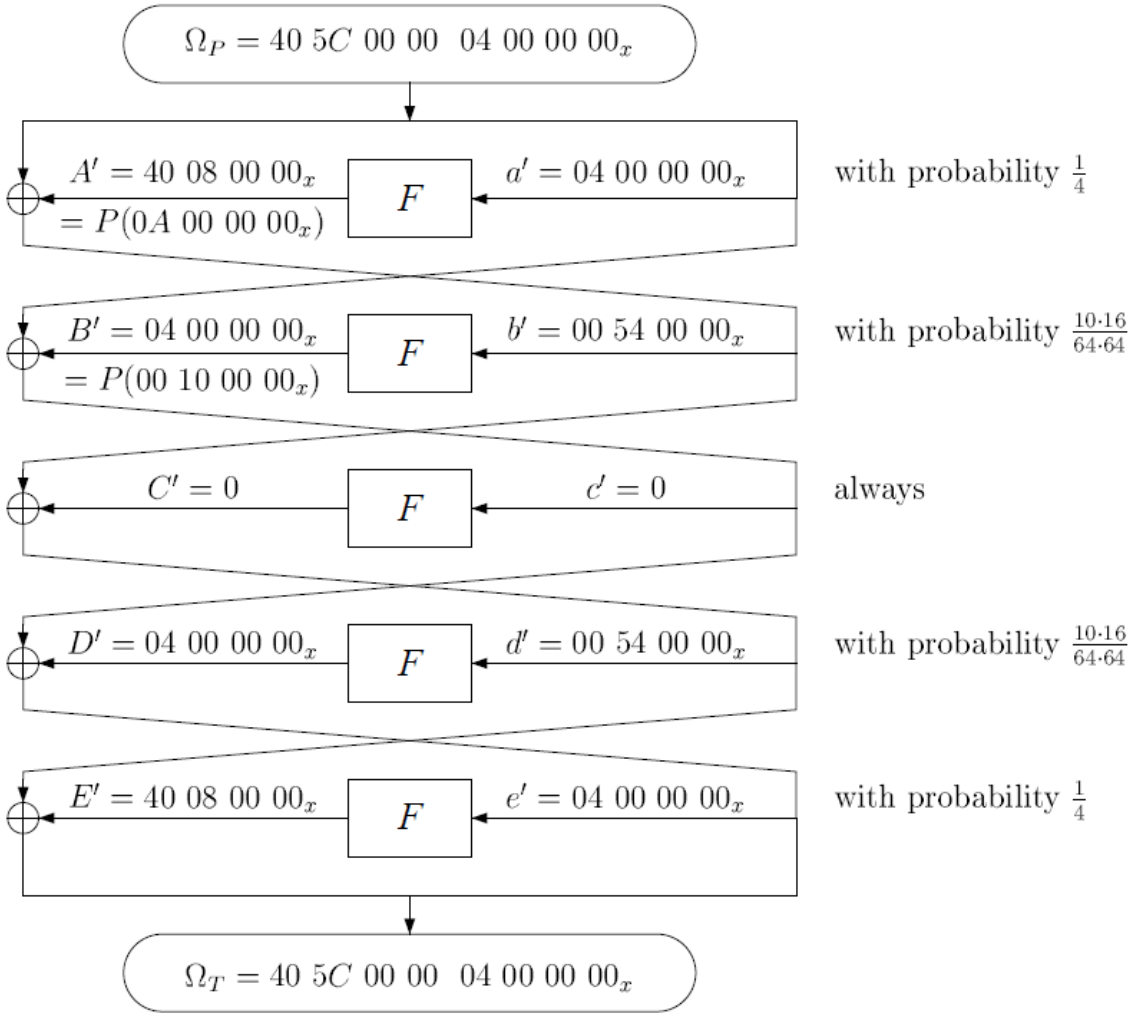
DES 8라운드 구조



공격 방법과 원리

차분에서는 다음과 같은 성질이 성립한다.

1. S-Box의 입력의 차분이 0이라면, 출력의 차분도 반드시 0이다.
2. $S_{EX}[j]' = S_{IX}[j]'$ S_{EX}에 키를 XOR한 결과가 S_{IX}이기 때문에, S_{EX}의 차분과 S_{IX}의 차분이 같다.



DES는 5라운드에 대해 $\frac{1}{10485.76}$의 확률로 위의 특성이 성립한다. 위 그림은 IP를 고려하지 않았기 때문에, IP를 고려해 $\mathrm{IP}(\mathrm{P}^{(1)}) \oplus \mathrm{IP}(\mathrm{P}^{(2)}) = 40\ 5\mathrm{C}\ 00\ 00\ 04\ 00\ 00\ 00$일 때, 5라운드 후 결과의 차분이 $40\ 5\mathrm{C}\ 00\ 00\ 04\ 00\ 00\ 00$으로 되는 것이 위 확률 정도로 성립한다는 것이다.

Step 1. 위 특성을 갖는 암호문 쌍 $\left(\mathrm{C}^{(1)}, \mathrm{C}^{(2)}, \mathrm{C}'\right)$ 20만 개 생성

첫번째로 구해야 하는 것은, $\mathrm{IP}(\mathrm{P}^{(1)}) \oplus \mathrm{IP}(\mathrm{P}^{(2)}) = 40\ 5\mathrm{C}\ 00\ 00\ 04\ 00\ 00\ 00$인 $(\mathrm{P}^{(1)}, \mathrm{P}^{(2)})$쌍이다. 임의의 $\mathrm{P}^{(1)}$을 선택한 뒤에 $\mathrm{P}^{(2)} = \mathrm{IP}^{-1}(\mathrm{IP}(\mathrm{P}^{(1)}) \oplus \{40\ 5\mathrm{C}\ 00\ 00\ 04\ 00\ 00\ 00\})$인 $\mathrm{P}^{(2)}$을 구한다.

각 $\mathrm{P}^{(1)}, \mathrm{P}^{(2)}$에 대응되는 $(\mathrm{C}^{(1)}, \mathrm{C}^{(2)})$쌍을 구한다. 그 후 두 암호문의 차분을 구해 최종적으로 20 만 개의 $(\mathrm{C}^{(1)}, \mathrm{C}^{(2)}, \mathrm{C}')$을 구하여 저장한다.

5라운드 결과의 차분이 $40\ 5\mathrm{C}\ 00\ 00\ 04\ 00\ 00\ 00$와 같다면, $\mathrm{e}' = 04\ 00\ 00\ 00, \mathrm{f}' = 40\ 5\mathrm{C}\ 00\ 00$이 된다.

6라운드에서 f'이 라운드함수 F에 들어가게 되면 다음과 같이 전개된다.

$\mathrm{f}' = 40\ 5\mathrm{C}\ 00\ 00$																																							
0	1	0	0		0	0	0	0		0	1	0	1		1	1	0	0		0	0	0	0		0	0	0	0		0	0	0	0		0	0	0	0

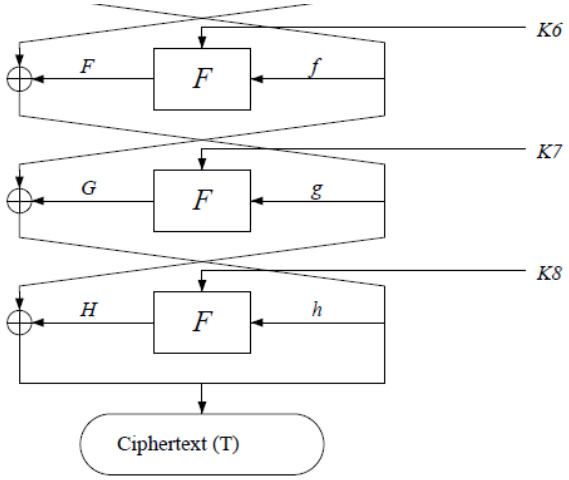
After Expansion							
1~6	7~12	13~18	19~24	25~30	31~36	37~42	43~48
Nonzero	Zero	Nonzero	Nonzero	Zero	Zero	Zero	Zero

Expansion의 과정을 통해 f'에 존재하는 1인 비트들이 자리가 변하거나, 확장되는 과정에서 여러 개가 생기는데, 1~6, 13~18, 19~24번째 비트들은 0이 아니기 때문에, S-Box의 입력의 차분이 0이 아니므로, 출력의 차분이 어떻게 될지는 알 수가 없다. 반면, 나머지 비트들인 7~12, 25~30, ..., 43~48에 대해서는 다 0이기 때문에 위에서 살펴본 차분의 성질 1에 의해 S-Box의 입력 차분이 0이므로, S-Box의 출력 차분이 0일 것이라는 것을 확신할 수 있다. 따라서, 6라운드 SBOX 2, 5, 6, 7, 8의 출력의 차분은 0이다. ($S_{Of}^{'}[j] = 0, j = \{2,5,6,7,8\}$) 그렇다면, SBOX의 출력 차분 중 20bit가 0이라는 정보를 알게 된 것이다.

이제 SBOX의 출력 차분은 라운드 함수 F 중 P(Permutation)을 거치게 되는데, 이 결과가 F'이 된다. (라운드 함수 F와 라운드 결과 F는 다름.)

$$F^{'} = P(\{1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\})$$

SBOX 1, 3, 4의 출력 차분은 어떻게 되는지 알 수 없으므로, 임의로 15(1111)로 둔다.



Feistel 구조에 따라 $H = IP(C)_L \oplus g = IP(C)_L \oplus e \oplus F$ 라는 것을 알 수 있고, 위에서 $e' = 04\ 00\ 00\ 00$, $C^{'}$, $F^{'}$을 구했으므로 $H^{'}$을 손쉽게 구할 수 있다. ($H^{'} = IP(C^{'})_L \oplus e^{'} \oplus F^{'}$) 다만, DES의 마지막에 inverse IP를 하는 것을 고려하여, 암호문 차분에 inverse IP의 역연산인 IP를 적용해 줘야 한다.

$H^{'}$이 암호화 방향과 반대 방향으로 라운드 함수 F에 들어가게 되면, P(Permutation)를 먼저 만나게 되기 때문에, $H^{'}$에 inverse P를 적용해주면, 8라운드에서의 SBOX 출력 차분을 구할 수 있다. Inverse P를 적용하게 되면, $F^{'}$을 구할 때 적용한 P를 결과적으로 없애 버리게 된다. 따라서 8라운

드에서도 SBOX 2,5,6,7,8의 출력 차분을 확실하게 알 수 있는 것이다.

$$\text{inv\_P}(H') = \text{inv\_P}(\text{IP}(C')_L) \oplus \text{inv\_P}(e') \oplus \text{inv\_P}(F')$$
$$= \text{inv\_P}(\text{IP}(C')_L) \oplus \text{inv\_P}(e') \oplus \{1,1,1,1,\ 0,0,0,0,\ 1,1,1,1,\ 1,1,1,1,\ 0,0,0,0,\ 0,0,0,0,\ 0,0,0,0,\ 0,0,0,0\}$$

Feistel 구조를 살피면, h의 차분 $h' = \text{IP}(C')_R$도 암호문의 차분을 통해 구할 수 있다는 것을 알 수 있다. 그 후, $S_{Ih}' = S_{Eh}' = E(h')$ h의 차분을 이용해 SBOX의 입력 차분 역시 구할 수 있다. 그렇게 되면, 8라운드에서 SBOX의 입력 차분과 출력 차분(출력 차분의 경우 20bit만)을 둘 다 알고 있는 셈이 된다.

Step 2. Right pair 선별하기

이제 해야 할 것은 위에서 살펴본 5라운드 특성(입력 차분이 40 5C 00 00 04 00 00 00일 때, 출력 차분이 40 5C 00 00 04 00 00 00)이 성립하는 쌍들만 선택하는 것이다. 위에서 $\text{IP}(P^{(1)}) \oplus \text{IP}(P^{(2)}) =$ 40 5C 00 00 04 00 00 00 을 만족하는 $(P^{(1)}, P^{(2)}, C^{(1)}, C^{(2)}, C')$ 을 구하는데, 평문 차분이 40 5C 00 00 04 00 00 00 이라 해서 평문 쌍이 5라운드까지 전개된 후 결과도 40 5C 00 00 04 00 00 00라 100%로 확신할 수 없다. 위 5라운드 특성은 $\frac{1}{10485.76}$의 확률로 성립하므로, 위에서 구한 20 만 개의 쌍들 중에서 $\frac{1}{10485.76}$ 확률로 위 특성을 가지는 쌍들을 찾아내어, 이들을 공격에 사용하는 것이다. 이를 [1]에서는 right pair라고 부른다.

해당 쌍이 right pair인지는 8라운드에서 SBOX의 입력 차분과 출력 차분을 이용하여 검증한다. 위 5라운드 특성이 성립할 경우, 반드시 8라운드 SBOX 2,5,6,7,8에서 암호문 차분의 right half인 h′를 Expansion후 구한 SBOX 입력 차분에 대해 위에서 구한 출력 차분을 가질 수 있어야 한다. http://www.cs.technion.ac.il/~cs236506/ddt/S1.html에서 제공되는 이미 구해진 SBOX의 Difference Distribution 표를 참고하여, 해당 SBOX에서 어떤 입력 차분에 대해 출력 차분이 나오는 횟수가 0 이상인지 확인하면 된다. 횟수가 0 이상이라는 것은 어떤 입력 차분에 대해 특정 출력 차분이 가능하다는 말과 같기 때문이다. (위 과정을 SBOX 2,5,6,7,8에 대해 확인한다.) 위 과정을 통해 right pair를 선별하게 되면, 기존의 20 만개 중 약 7000~8000개의 right pair만이 남게 된다.

Step 3. 8라운드 서브키 중 18bit 찾기

위 right pair들을 이용해서 키를 찾는 핵심 원리는 아래와 같다.

right pair 중 하나를 선택하여 $(C^{(1)}, C^{(2)}, C')$를 구하고, 위에서 암호문의 차분 C'을 통과시킨 것 대신 $\text{IP}(C^{(1)}), \text{IP}(C^{(2)})$ 각각을 h로 보고 Expansion에 통과시키게 되면 $S_{Eh}^{(1)}$, $S_{Eh}^{(2)}$을 구할 수 있다. 추측한 K_h을 이용해 $S_{Ih}^{(1)} = S_{Eh}^{(1)} \oplus K_h$, $S_{Ih}^{(2)} = S_{Eh}^{(2)} \oplus K_h$ 을 구할 수 있고, 이들 각각을 SBOX에 통과시켜 $S_{Oh}^{(1)}, S_{Oh}^{(2)}$을 구한 후 이 둘의 차분인 S_{Oh}'을 구할 수 있다.

$$H' = \text{IP}\left(C'\right)_L \oplus e' \oplus F'$$

$$\text{expected } S_{Oh}' = \text{inv\_P}(H')$$

C'을 이용하여 위에서 구한 것과 같이 예상되는(expected) 또 다른 S_{Oh}'을 구한다. 추측한 K_h을 이용해 얻은 SBOX의 출력 차분과 C'을 이용하여 구한 expected SBOX의 출력 차분이 같을 경우 count를 1 증가시킨다. 이를 right pair 모두에 대해 적용한다. 만약 키가 올바르다면, 확률적으로 다른 틀린 키들 보다 count가 클 것이다. 따라서 count가 가장 클 때의 키가 올바른(correct) 키다.

요약하자면, K_h 후보에 대해 $S_{Eh}^{(1)}, S_{Eh}^{(2)}$을 이용해 $S_{Ih}^{(1)}, S_{Ih}^{(2)}$을 구한 후 $S_{Oh}^{(1)}, S_{Oh}^{(2)}$을 이용해 구한 S_{Oh}'과 expected S_{Oh}'을 비교하는 과정 후 같다면 count를 증가시키는 과정을 right pair 전체에 대해 전개한다. 최종적으로 count가 가장 큰 K_h가 올바른 키가 된다.

8라운드의 서브키 중 $K_h[6], K_h[7], K_h[8]$을 구하기 위해 위 방법을 적용할 수 있다. 이때,K_h 후보는 2^{18}개가 된다. expected S_{Oh}'와 S_{Oh}'을 비교할 때, 모든 비트에 대해 비교를 하는 것이 아니라, 키를 구하는 부분에 해당하는 SBOX 6,7,8의 출력 차분에 대해서 비교를 한다. 두 차분의 SBOX 6,7,8부분이 같을 경우 count를 1 증가시킨다.

Step 4. Invalid pair 제거하기

$K_h[6], K_h[7], K_h[8]$키를 구하게 된다면 8라운드의 서브키 48bit 중 18bit를 찾은 상황이 된다. 이제 이 18bit를 이용해서 right pair 중 invalid pair들을 제거한다. Right pair의 기준은 SBOX 2,5,6,7,8에서 입력 차분에 대해 출력 차분이 가능한 지였다. 이제 올바른 $K_h[6], K_h[7], K_h[8]$을 알게 되었기 때문에 SBOX 6,7,8에 대해서 정확한 출력 차분을 구할 수 있게 되었다. right pair 중 SBOX 6,7,8의 출력 차분이 $S[j](K_h[j] \oplus S_{Eh}^{(1)}) \oplus S[j](K_h[j] \oplus S_{Eh}^{(2)})(j = \{6,7,8\})$을 만족하지 않는다면 invalid하다고 판단하여 제거한다. 그럼 상당 수의 right pair가 제거된다. 7000~8000개의 right pair 중 약 70~80개의 valid pair만 남게 된다.

Step 5. 8라운드 서브키 중 12bit 찾기

$K_h[2], K_h[5]$에 대해서 Step 4에서 구한 valid pair들에 대해 Step 3를 적용해 서브키 중 12bit를 찾는다.

Step 6. 나머지 키 $K_h[1], K_h[3], K_h[4]$ 찾기

나머지 부분인 SBOX 1,3,4에 대해서는 Step 1에서 사용한 5라운드 특성을 적용할 수 없으므로, 다른 방법을 사용해야한다. SBOX 2,5,6,7,8 부분의 키 비트는 정확하게 아는 상황에서 SBOX 1,3,4에 대해 2^{18}개의 K_h 후보가 생긴다. 만약 K_h가 정확하다면, $h^{(1)} = IP(C^{(1)})_R$을 이용해 정확한 $H^{(1)} = F(K_h, h^{(1)})$을 구할 수 있다. Feistel 구조에 따라 $g^{(1)} = H^{(1)} \oplus IP(C^{(1)})_L$ 을 구할 수 있다.

DES의 경우 key scheduling algorithm에 의해 메인 64bit 키의 많은 비트가 라운드 서브 키들끼리 공유된다. 8라운드 서브키와 7라운드 서브키도 많은 키 비트들을 공유하고 있다.

메인 키의 비트들을 1,2, …, 64라고 했을 때, 다음은 각각 8라운드 서브키와 7라운드 서브키다.

8라운드 서브키: (19 60 43 33 58 26) (42 1 11 18 57 51) (41 44 35 34 17 2) (3 10 9 36 27 50) (29 39

46 61 12 15) (54 37 47 28 30 4) (5 63 45 7 22 31) (20 21 55 6 62 38)

7라운드 서브키: (27 3 51 41 1 34) (50 9 19 26 36 59) (49 52 43 42 25 10) (11 18 17 44 35 58) (37 47 54 6 20 23) (62 45 55 5 38 12) (13 4 53 15 30 39) (28 29 63 14 7 46)

위에서 44처럼 8라운드 서브키와 7라운드의 서브키가 많은 수의 비트를 공유하고 있는 것을 확인할 수 있다. 8라운드 서브키 K_h의 모든 비트들이 다 구해진 상황이므로(30bit는 확실하게 구하였고, 18bit는 추측된 상황), 7라운드 서브키 K_g을 구성할 수 있다. 만약, K_h과 K_g가 공유하는 비트가 아니라면, independent한 비트라면 0으로 두고, K_g을 구성한다.

위에서 구한 $g^{(1)}$와 K_g을 이용해 $S_{Og}{}^{(1)}$을 구할 수 있다. 또 다른 암호문 $C^{(2)}$에 대해서도 $g^{(2)}$을 구하여 $S_{Og}{}^{(2)}$을 구한다. 그럼 $S_{Og}{}'$을 구할 수 있다. 5라운드 특성에 따라 $f' = 40\,5C\,00\,00$이고 $h' = IP\left(C'\right)_R$이므로 Feistel 구조에 의해 $G' = f' \oplus h'$을 구할 수 있다. G'에 inverse P를 적용하면 expected $S_{Og}{}'$을 구할 수 있고 이 값을 $S_{Og}{}'$와 비교하여 일치한다면, count를 1 증가시킨다. Step 5와 마찬가지로 valid pair들에 대해 counting scheme을 적용하여 확률적으로 count가 가장 큰 값이 올바른 18bit 키일 확률이 높을 것이다.

[1]에서 제시한 또 다른 Step 6 방법

위 Step 6에서 소개한 방법은 $K_h[1], K_h[3], K_h[4]$을 한번에 찾아내지만, 조금 더 빠르게 키 비트들을 찾기 위해 $K_h[1], K_h[4]$을 먼저 구하고, 그 후에 $K_h[3]$을 구하는 방법을 [1]에서는 제시하였다. 그렇게 되면 $2^{12} + 2^6$ 의 복잡도로 이전의 2^{18} 에 비해 더 빠르게 비트들을 찾아낼 수 있다. $K_h[1], K_h[4]$을 찾을 때, $K_h[3] = 0$이라 두고 $K_h[1], K_h[4]$에 대해 Step 6를 적용하고, 그 후 나머지 $K_h[3]$에 대해 Step 6를 적용하면 8라운드의 서브키 전체 비트를 찾아낼 수 있다.

발생한 문제와 문제 해결 방법

[1]에서는 제시한 $K_h[1], K_h[4]$을 먼저 구하고, 그 후에 $K_h[3]$을 구하는 방법을 Step 6에서 적용할 때, 임의로 선택한 $K_h[3]$의 값이 만약 틀린 값이라면, Step 6를 전개했을 때, $S_{Og}{}'$과 expected $S_{Og}{}'$의 모든 비트가 전부 같을 확률이 급격하게 감소하게 되어, 모든 2^{12}개의 후보들에 대해 count가 0이 나와 counting scheme으로 올바른 키를 찾아낼 수 없는 문제가 발생하였다.

$K_h[3]$의 값이 틀린 것이 확률 감소의 원인이기 때문에, $S_{Og}{}'$ 중 $K_h[3]$의 영향을 받는 비트들을 counting scheme을 적용할 때 최대한 고려하지 않는 방법을 생각해보았다.

Step 6에서 잘못된 $K_h[3]$의 영향을 받는 부분은 두 개다.

1. K_h을 바탕으로 K_g가 구성하는 부분

2. $H^{(1)} = F(K_h, h^{(1)})$

8라운드 서브키: (19 60 43 33 58 26) (42 1 11 18 57 51) (41 44 35 34 17 2) (3 10 9 36 27 50) (29 39 46 61 12 15) (54 37 47 28 30 4) (5 63 45 7 22 31) (20 21 55 6 62 38)

7라운드 서브키: (27 3 51 41 1 34) (50 9 19 26 36 59) (49 52 43 42 25 10) (11 18 17 44 35 58) (37 47 54 6 20 23) (62 45 55 5 38 12) (13 4 53 15 30 39) (28 29 63 14 7 46)

1번의 경우, 8라운드의 서브키 중 SBOX 3 부분의 비트와 7라운드 서브키가 비트를 공유하는 부분은 7라운드에서 SBOX 1, 4에 해당하는 부분이다. 따라서 $S_{Og}{}'$과 expected $S_{Og}{}'$을 비교할 때, 모든 비트를 비교하는 것이 아니라 SBOX 1,4를 제외한 SBOX 2,3,5,6,7,8의 출력 차분을 비교한다.

2번의 경우, $K_h[3]$가 잘못되었다면, 라운드 함수 F의 SBOX 3의 출력 부분이 잘못되었을 것이다. SBOX의 출력이 다음 단계인 P(Permutation)를 거치게 되어 구한 $H^{(1)}$, 그리고 $H^{(1)}$을 바탕으로 $g^{(1)}$가 구해지고, $g^{(1)}$이 7라운드에서 라운드 함수 F로 들어가 E(Expansion)을 거치게 된다.

8라운드 SBOX 32bit 출력 중 SBOX 3의 출력에 해당하는 9,10,11,12번째 비트가 P를 통과하면서 $g^{(1)}$의 6,16,24,40번째 비트가 되고, E를 거치게 되면 7라운드 SBOX 48bit 입력의 9,23,25,35,37,45번째 비트에 영향을 주게 된다.

7라운드 SBOX 입력: (1 2 3 4 5 6) (7 8 9 10 11 12) (13 14 15 16 17 18) (19 20 21 22 23 24) (25 26 27 28 29 30) (31 32 33 34 35 36) (37 38 39 40 41 42) (43 44 45 46 47 48)

그럼 최종적으로 $K_h[3]$에 영향을 받는 SBOX는 2,4,5,6,7,8이 된다. 1번의 경우와 2번의 경우를 둘 다 고려하면 SBOX 3만이 $K_h[3]$ 의 영향을 받지 않는다는 것을 알 수 있다. 따라서, $K_h[1], K_h[3], K_h[4]$을 한번에 구할 때와 다르게, $K_h[1], K_h[4]$와 $K_h[3]$로 나눠서 구할 때는 $S_{Og}{}'$과 expected $S_{Og}{}'$을 비교할 때 SBOX 3 부분의 출력 차분만 비교하면 가장 정확하다는 결론을 내릴 수 있었다.

실제로 이 방법으로 실제 올바른 키를 찾을 수 있었다. 만약 이 방법으로 부족하다면, SBOX 3와 더불어 1,2,4,5,6,7,8 중 하나의 SBOX 부분을 더 선택할 수 있다. SBOX 3로 후보들을 간추린 후, SBOX 3와 또 다른 SBOX를 선택하여 두 SBOX의 출력 차분을 비교하는 방법을 통해 키 후보를 계속 좁혀 나가 올바른 키를 찾을 수 있을 것이다. 하지만, 잘못된 $K_h[3]$의 영향을 아예 받지 않는 SBOX 3로 얻은 1차적인 키 후보가 가장 정확할 것이므로, 추가적인 SBOX들에 대한 검증은 1차적인 키 후보 중에서 올바르지 않은 것을 제외시키는 데 사용하는 것이 바람직할 것이다.

구현 결과

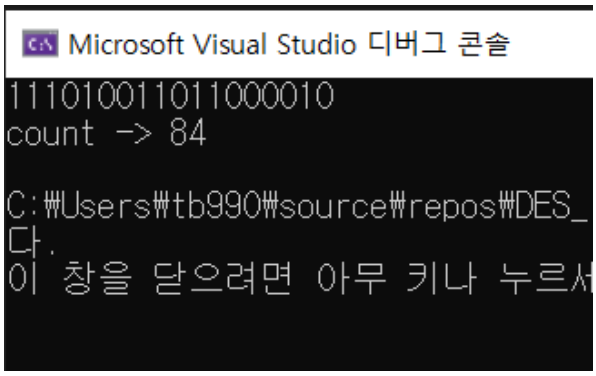
구현 시 사용한 64bit 메인 키와 8라운드 서브키는 다음과 같다.

메인 키: $\{0123456789ABCDEF\}_{hex}$

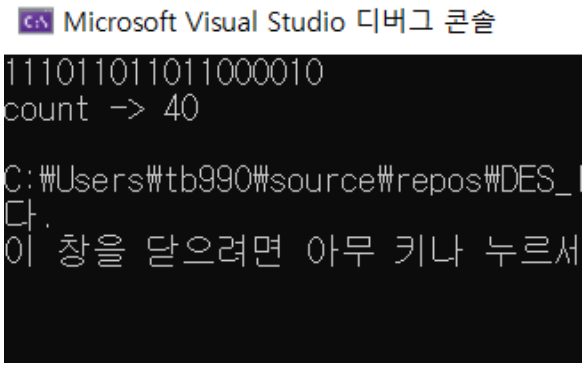
8라운드 서브키: {001111 001110 100000 000011 000101 111010 011011 000010}

본 보고서에서는 20 만개 쌍을 중점으로 공격을 진행하였고, 8라운드 서브키를 완벽하게 구할 수 있었다. 이후 확률 비교를 위해 15만 개, 25만개 쌍에 대해 추가적인 공격을 진행하였다.

Step 3 실행 결과

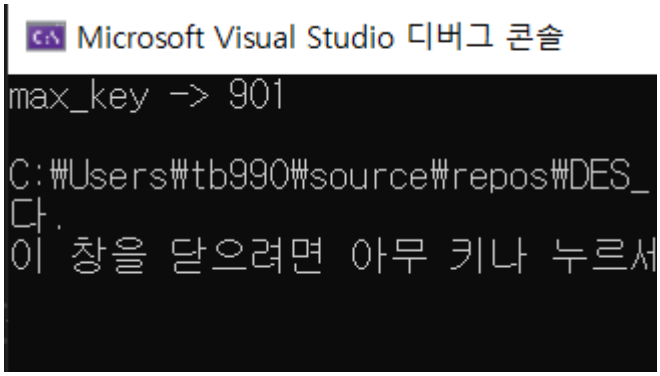


SBOX 6,7,8에 해당하는 {111010 011011 000010}을 찾아낸 것을 확인할 수 있었다.



잘못된 키일 때 count는 올바른 키일 때의 count의 1/2 수준으로, 올바른 키일 때 pair에 대해 검증을 성공할 확률이 틀린 키일 때보다 2배인 것을 알 수 있다.

Step 5 실행 결과



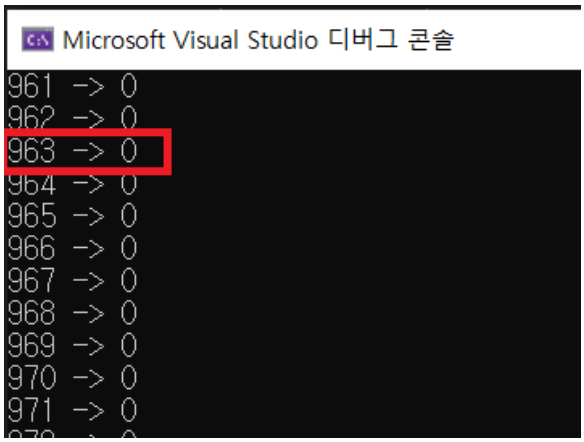
SBOX 2,5에 해당하는 $901_{10} = \{001110\ 000101\}$ 키 비트를 찾아낸 것을 확인할 수 있다.

Step 6 실행 결과

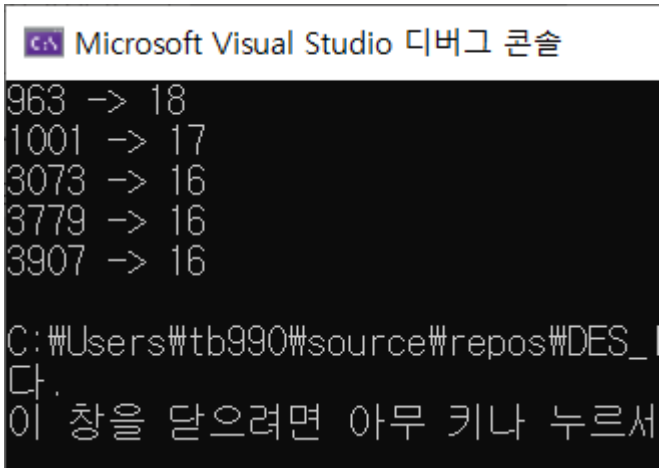
$K_h[1], K_h[3], K_h[4]$을 한번에 구할 때, 올바른 키의 count가 다른 후보들의 count보다 약 두 배였기 때문에 손쉽게 올바른 키를 찾을 수 있었다. 반면에 $K_h[1], K_h[4]$와 $K_h[3]$을 나눠서 구할 때는 위에서 소개한 문제가 발생하였다.

$K_h[1], K_h[4]$에 해당하는 올바른 키는 $963_{10} = \{001111\ 000011\}$이다.

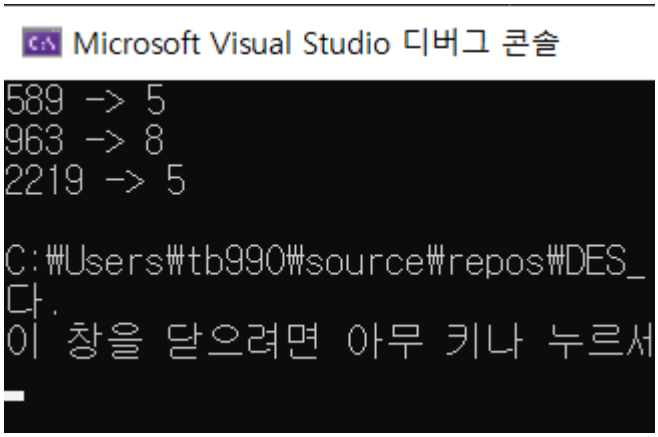
잘못된 $K_h[3]$가 선택이 되었을 때, valid pair가 검증을 통과할 확률이 낮아져 모든 키 후보에 대해 count가 0가 나오고, 올바른 키인 963일 때도 count가 0이 되는 결과를 얻었다.



이를 해결하기 위해 SBOX 3의 출력 차분만 비교하여 검증하는 방법을 도입하였고, 올바른 키인 963이 count=18을 보이며, 다른 키 후보들보다 검증 성공률이 높다는 것을 보여주었다.



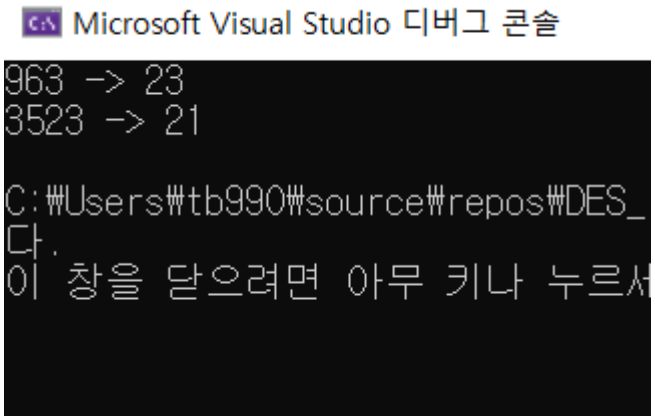
하지만, 다른 키 후보인 1001, 3073, 3779, 3907과 비교했을 때, count가 별 차이가 안 난다는 문제가 발생하였다. 이를 해결하기 위해 SBOX 3뿐 아니라, 추가적으로 SBOX 8의 출력 차분에 대해서도 검증을 하였다.



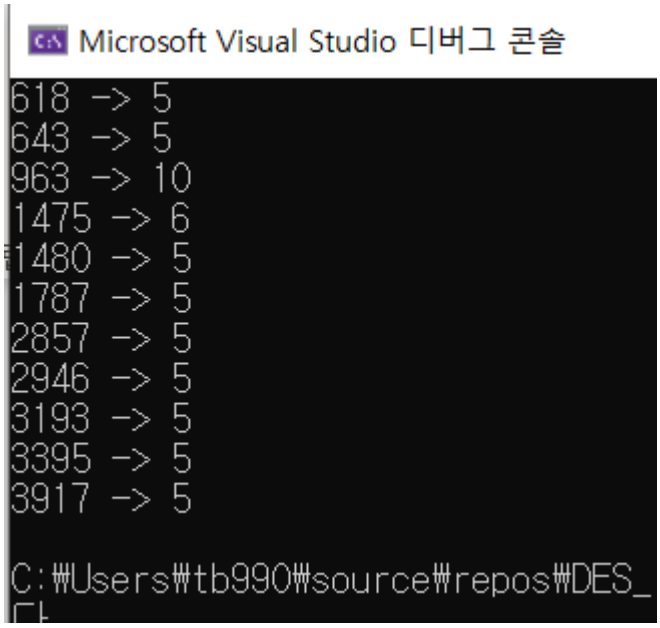
그 결과, 963일 때 가장 많은 count를 보였고, SBOX 2만 검증했을 때 나온 키 후보들인 1001, 3073, 3779, 3907이 이번 검증에서는 등장하지 않았다. 따라서 확률적으로 963이 올바른 키일 것이라는 결론을 내릴 수 있었다.

25 만개 쌍에 대한 Step 6 실행 결과

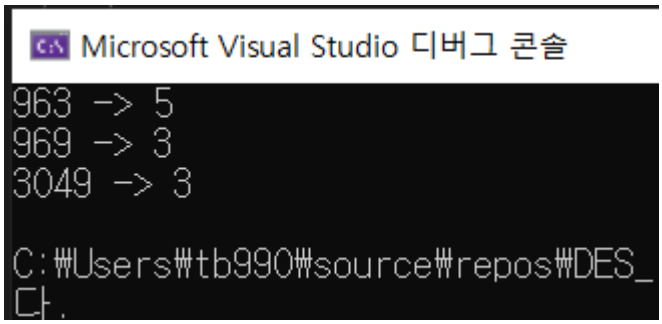
SBOX 3의 출력 차분에 대해서만 검증한 결과는 다음과 같다.



SBOX 3와 SBOX 6의 출력 차분에 대해 검증한 결과는 다음과 같다.



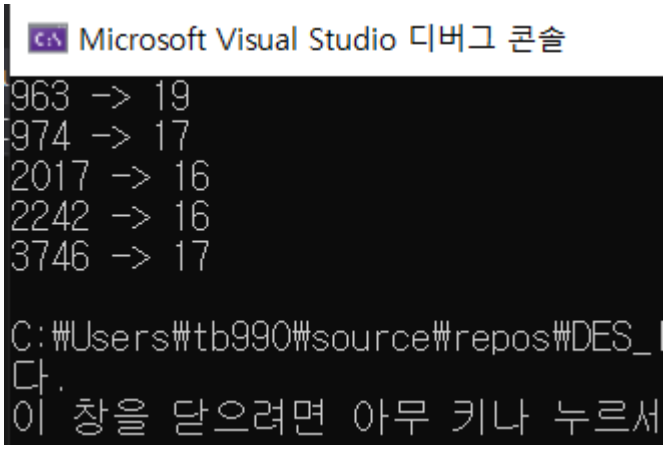
SBOX 3,5,6의 출력 차분에 대해 검증한 결과는 다음과 같다.



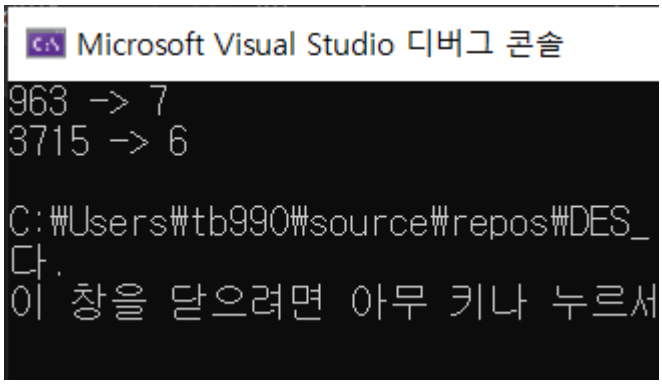
Pair 수를 25 만개로 늘리는 경우, 3개 이상의 SBOX에 대해 출력 차분을 검증할 수 있기 때문에, 더 세밀하게 올바른 키를 찾아 나갈 수 있다.

15 만개 쌍에 대한 Step 6 실행 결과

SBOX 3의 출력 차분에 대해 검증한 결과는 다음과 같다.



SBOX 3와 SBOX 8에 대해 검증한 결과는 다음과 같다.



15 만, 20 만, 25 만의 실행 결과 비교

15만, 20만, 25만 모두 SBOX 3의 출력 차분만에 대해 검증을 실시할 경우 올바른 키 963을 찾아내는 것을 확인할 수 있었다. 다른 SBOX의 출력 차분에 대해 추가적으로 검증을 실시하는 경우, 20만, 25만의 경우 올바른 키일 때 count가 틀린 키일 때 count의 두 배정도가 나와 확연하게 차이를 확인할 수 있었지만, 15만일 때 SBOX 3,8에 대해 검증할 경우 올바른 키와 틀린 키의 count 차이가 1정도로 별로 차이가 나지 않았다. 따라서 사용하는 pair 수가 늘어날수록, 올바른 키를 확실하게 찾아낼 수 있다는 결론을 얻을 수 있었다.

결론

이 프로젝트는 차분 공격에 대한 이해를 위해 공격의 구현, 특성과 구조 분석에 중점을 두었다. [1]과 [2]에서의 설명을 바탕으로 공격을 직접 구현해보고, 보고서에 내용을 정리해보면서 차분공격에 대해 자세히 이해할 수 있었다. 하지만, 8라운드의 경우 20만개 쌍에 대해 공격을 전개하더라도 공격이 완료되기까지 시간이 그다지 오래 소요되지 않았기 때문에 프로그램 코드의 효율성 및 공격 소요시간에 대한 부분은 고려되지 않았다는 점이 아쉬웠다. 이번 프로젝트를 바탕으로 추후에 12라운드 차분 공격 코드의 효율성을 중점으로 연구해보는 것도 의미가 있을 것이라 생각한다.

참고 문헌

[1] Eli Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystem", 1990

[2] "hxp | 0CTF Quals 2019: zer0des writeup", hxp, last modified March 27. 2019, accessed April 5. 2020, https://hxp.io/blog/56/0CTF-Quals-2019-zer0des-writeup/