2026/11/25, 2026 보안 기초 세미나

암호학과 네트워크 보안

-6장 DES, 7장 AES-

서 진 우(jinwoo@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

목 차

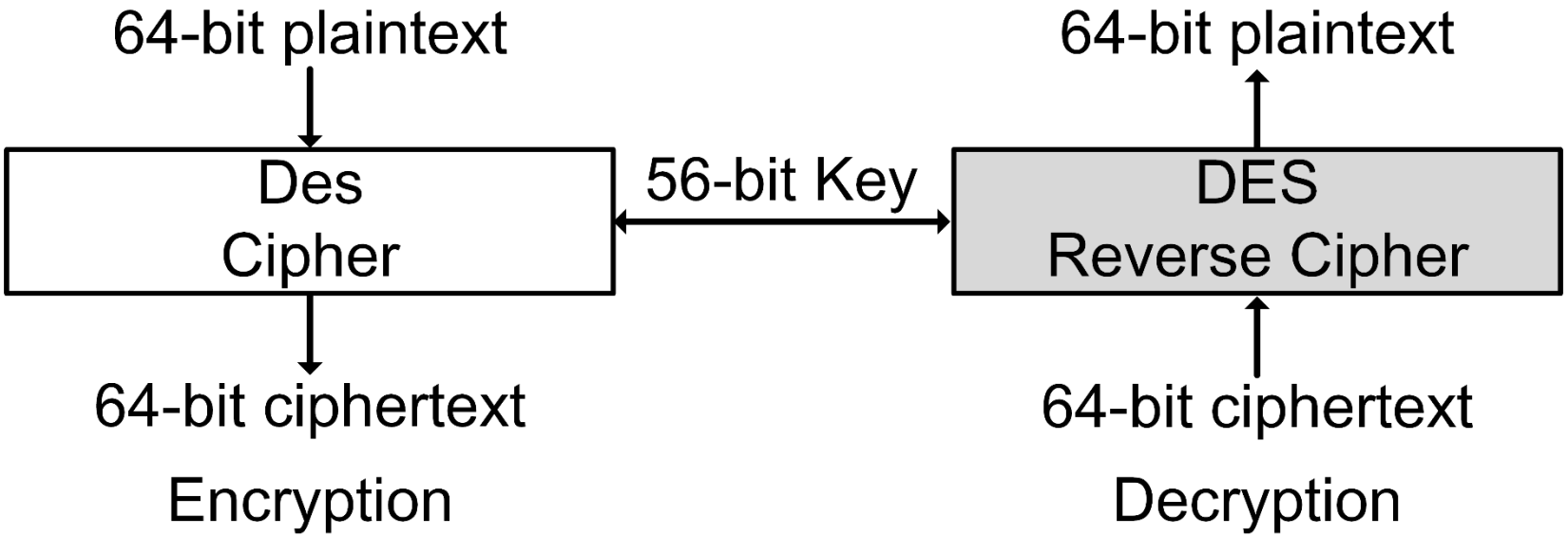
- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

개요

- DES (Data Encryption Standard) (1/2)
  - 정의
    - 미국 국립기술표준원(NIST, National Institute of Standards and Technology)이 공표한 대칭-키 블록 암호
  - 역사
    - 1973년 IBM (International Business Machines)의 Lucifer 수정안을 채택하여 1977년 미국 연방 표준으로 발표
    - 56비트 키가 짧아 모든 키를 대입하는 전수조사 공격(Brute-force Attack)에 약하다는 비판 발생
    - 이후 삼중 DES를 거쳐 AES (Advanced Encryption Standard)로 대체

개요

- DES (Data Encryption Standard) (2/2)
  - 특징
    - 64비트 평문 블록을 56비트 암호 키(Cipher Key)로 암호화하여 64비트 암호문 블록 생성
    - 복호화는 같은 56비트 암호 키로 64비트 암호문 블록을 64비트 평문 블록으로 복원



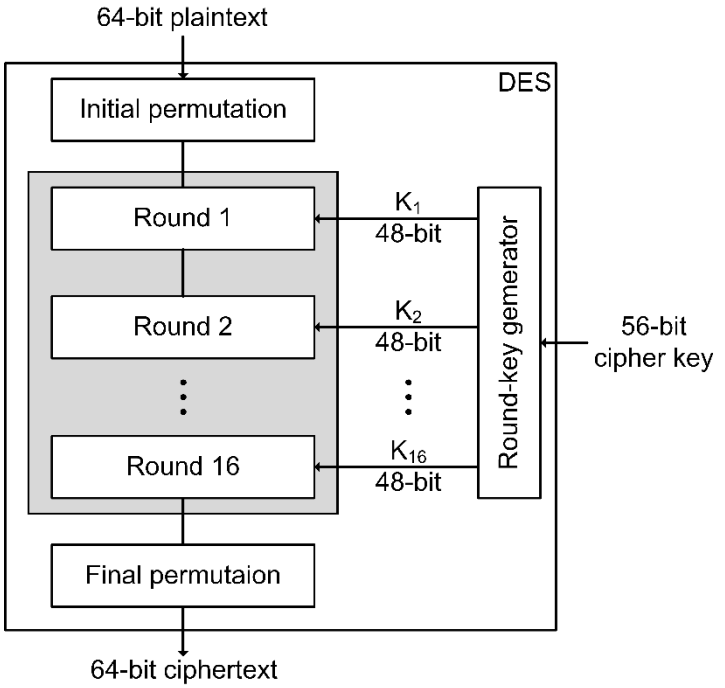
목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

DES 구조

- 전체 구조
  - 구성 요소
    - 초기 치환(Initial Permutation)
    - 최종 치환(Final Permutation)
    - 라운드(Round)
    - 라운드 키 생성기(Round-key generator)

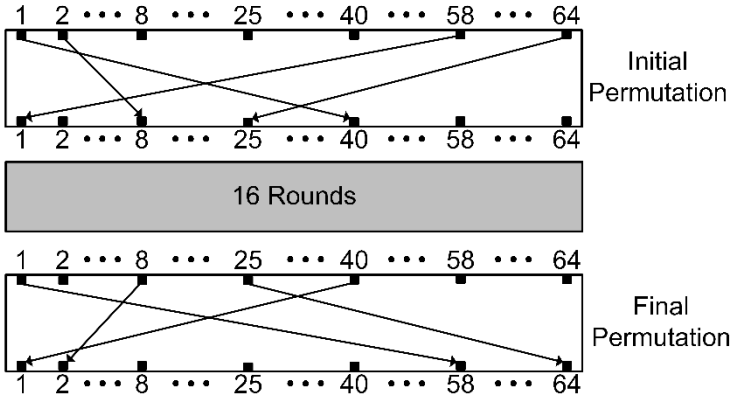
*Feistel 라운드(Feistel Round)
블록을 왼쪽과 오른쪽 절반으로 나누어, 오른쪽 절반과 라운드 키로 만든 값을 왼쪽 절반에 XOR한 후 두 절반을 교환하는 라운드



  - 개요
    - 64비트 평문 블록을 초기 치환16개의 *Feistel 라운드, 최종 치환순서로 처리하여 64비트 암호문 블록 생성
    - 라운드 키 생성기를 통해 56비트 암호키로 라운드에서 사용할 48비트 라운드 키 16개 생성
    - I번째 라운드는 라운드 키 K_I 사용 (단, $1 \leq I \leq 16$)

DES 구조

- 초기 치환과 최종 치환 (1/4)
  - 개요 (1/2)
    - 64비트 입력을 키 없이 정해진 표에 따라 64비트로 재배열하는 P-박스
      - 초기 치환은 첫 번째 라운드 앞, 최종 치환은 16번째 라운드 뒤에 위치
    - 키와 무관한 고정적인 비트 재배열이므로 암호학적 안전성에 직접 기여하지 않는 단계
      - 초기 치환 이후 64비트를 좌우 32비트 블록으로 분할하여 라운드 연산 수행
      - 최종 치환은 16라운드 연산 결과를 초기 치환의 역순으로 재배열하여 출력



DES 구조

- 초기 치환과 최종 치환 (2/4)
  - 개요 (2/2)
    - 각 치환 표의 칸 값은 해당 칸 위치의 출력 비트로 가는 입력 비트 번호
      - e.g., 초기 치환 표의 1~8행 3열 값이 42이므로 입력 42번째 비트가 출력 3번째 비트로 이동
      - e.g., 최종 치환 표의 1~8행 1열 값이 40이므로 16라운드 종료 후 출력된 40번째 비트가 최종 암호문의 1번째 비트로 이동

초기 치환	1	2	3	4	5	6	7	8
1~8	58	50	42	34	26	18	10	02
9~16	60	52	44	36	28	20	12	04
17~24	62	54	46	38	30	22	14	06
25~32	64	56	48	40	32	24	16	08
33~40	57	49	41	33	25	17	09	01
41~48	59	51	43	35	27	19	11	03
49~56	61	53	45	37	29	21	13	05
57~64	63	55	47	39	31	23	15	07

최종 치환	1	2	3	4	5	6	7	8
1~8	40	08	48	16	56	24	64	32
9~16	39	07	47	15	55	23	63	31
17~24	38	06	46	14	54	22	62	30
25~32	37	05	45	13	53	21	61	29
33~40	36	04	44	12	52	20	60	28
41~48	35	03	43	11	51	19	59	27
49~56	34	02	42	10	50	18	58	26
57~64	33	01	41	09	49	17	57	25

DES 구조

- 초기 치환과 최종 치환 (3/4)
  - 예제 6.1

16진수 입력 값 0x0002 0000 0000 0001의 초기 치환 출력 값을 구하시오.

- 풀이
  1. 입력 값을 2진수로 바꾸면 최상위 비트부터 15번째와 64번째 비트만 1
  2. 초기 치환 표에서 값 15는 출력 63번째 칸, 값 64는 출력 25번째 칸에 위치
     - 입력 15번째 비트는 출력 63번째 비트, 입력 64번째 비트는 출력 25번째 비트로 이동
  3. 출력 값은 25번째와 63번째 비트만 1이므로 16진수로 0x0000 0080 0000 0002

∴ 0x0000 0080 0000 0002

초기 치환	1	2	3	4	5	6	7	8
1~8	58	50	42	34	26	18	10	02
9~16	60	52	44	36	28	20	12	04
17~24	62	54	46	38	30	22	14	06
25~32	64	56	48	40	32	24	16	08
33~40	57	49	41	33	25	17	09	01
41~48	59	51	43	35	27	19	11	03
49~56	61	53	45	37	29	21	13	05
57~64	63	55	47	39	31	23	15	07

DES 구조

- 초기 치환과 최종 치환 (4/4)
  - 예제 6.2

최종 치환의 입력 값이 0x0000 0080 0000 0002일 때 출력 값을 구하고, 0x0002 0000 0000 0001의 초기 치환 출력이 0x0000 0080 0000 0002임을 이용하여 두 치환이 서로 역관계임을 보이시오.

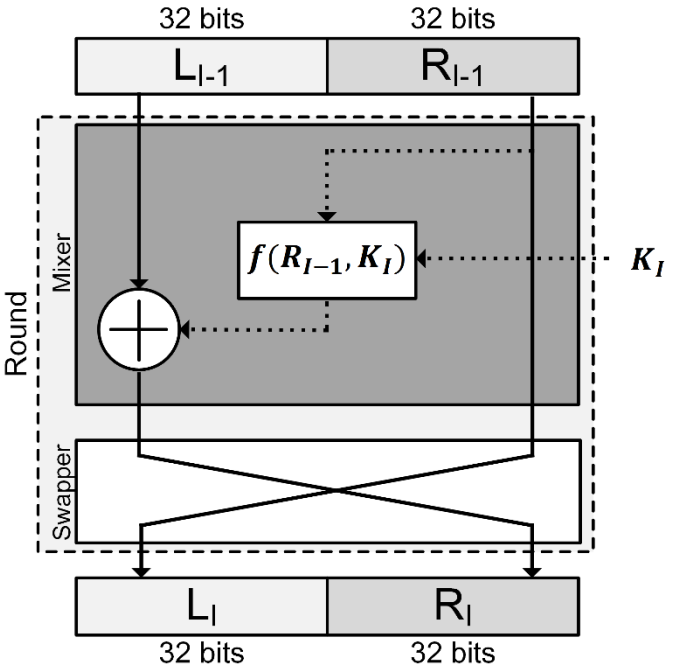
- 풀이
  1. 입력 값은 최상위 비트부터 25번째와 63번째 비트만 1
  2. 최종 치환 표에서 값 25는 출력 64번째 칸, 값 63은 출력 15번째 칸에 위치
  3. 출력 값은 15번째와 64번째 비트만 1이므로 16진수로 0x0002 0000 0000 0001
  4. 초기 치환 입력이 최종 치환 출력으로 복원되므로 두 치환은 역관계

  ∴ 출력 값 0x0002 0000 0000 0001, 두 치환은 서로 역관계

최종 치환	1	2	3	4	5	6	7	8
1~8	40	08	48	16	56	24	64	32
9~16	39	07	47	15	55	23	63	31
17~24	38	06	46	14	54	22	62	30
25~32	37	05	45	13	53	21	61	29
33~40	36	04	44	12	52	20	60	28
41~48	35	03	43	11	51	19	59	27
49~56	34	02	42	10	50	18	58	26
57~64	33	01	41	09	49	17	57	25

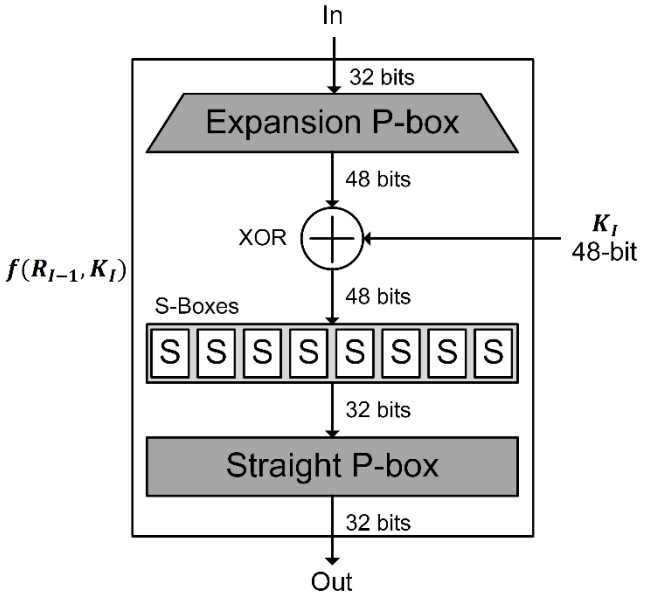
DES 구조

- 라운드 (1/7)
  - 개요
    - 이전 단계의 왼쪽 32비트 L_{I-1}, 오른쪽 32비트 R_{I-1}과 라운드 키 K_I로 L_I, R_I 생성
    - 혼합기(Mixer)는 L_{I-1}에 $f(R_{I-1}, K_I)$를 XOR하고 R_{I-1}은 그대로 전달
    - 스와퍼(Swapper)는 혼합기 출력의 왼쪽과 오른쪽 32비트를 교환
    - 혼합기와 스와퍼는 한 번 더 적용하면 원래 값이 되는 구성요소



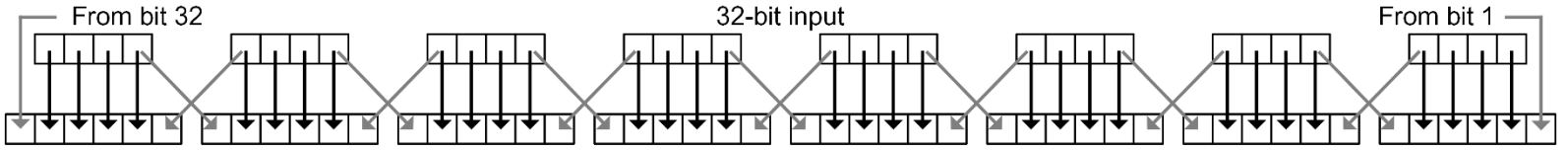
DES 구조

- 라운드 (2/7)
  - DES 함수 (1/6)
    - 정의
      - R_{I-1}에 48비트 라운드 키 K_I를 적용하여 32비트를 출력하는 함수 $f(R_{I-1}, K_I)$
    - 특징
      - 그대로 전달된 R_{I-1}로 다시 계산되므로 역함수가 없어도 복호화 가능
      - 확장 P-박스가 32비트 중 16비트를 이웃한 두 S-박스에 넣어 1비트 변화를 두 S-박스로 전달
    - 구성 요소
      - 확장 P-박스(Expansion P-box)
      - 키 XOR
      - S-박스(S-box)
      - 단순 P-박스(Straight P-box)

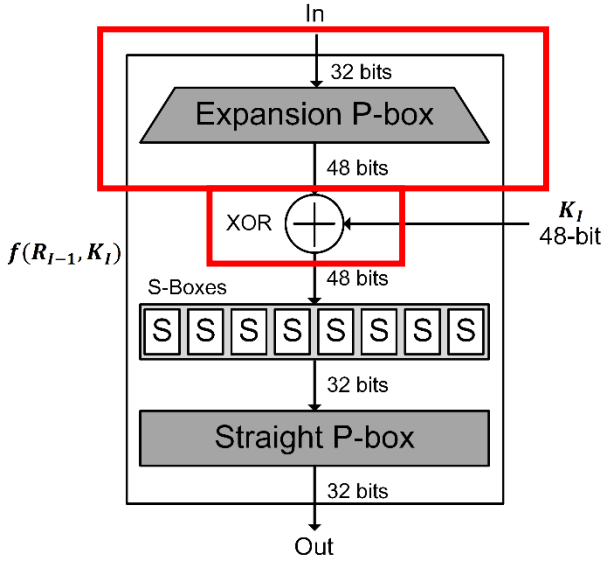


DES 구조

- 라운드 (3/7)
  - DES 함수 (2/6)
    - 과정 (1/3)
      1. 확장 P-박스를 통해 32비트 R_{I-1}을 48비트로 확장
         - R_{I-1}을 8개의 4비트로 분할 후 미리 정의된 규칙에 따라 6비트로 확장
         - 16개 입력 비트가 두 출력 비트로 복사되어 32비트가 48비트로 증가



32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

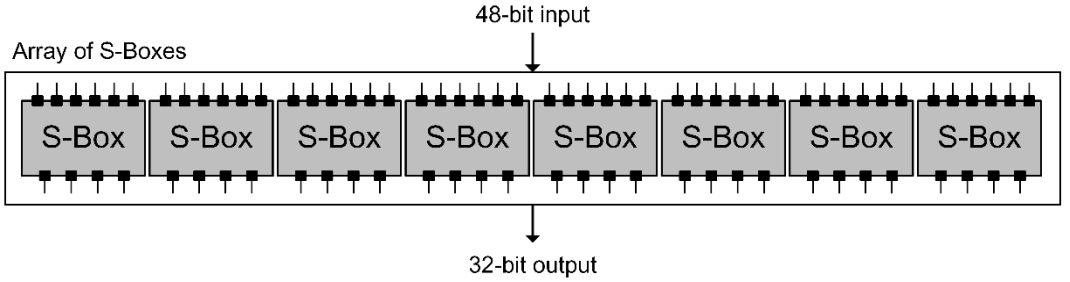


      2. 확장된 48비트와 라운드 키 K_I를 XOR 연산

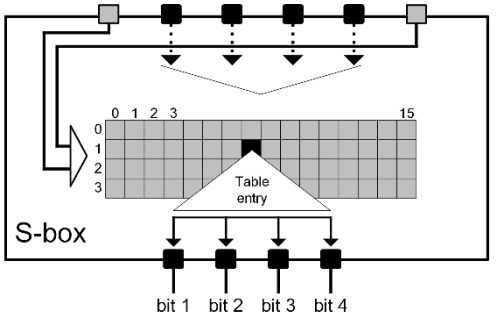
DES 구조

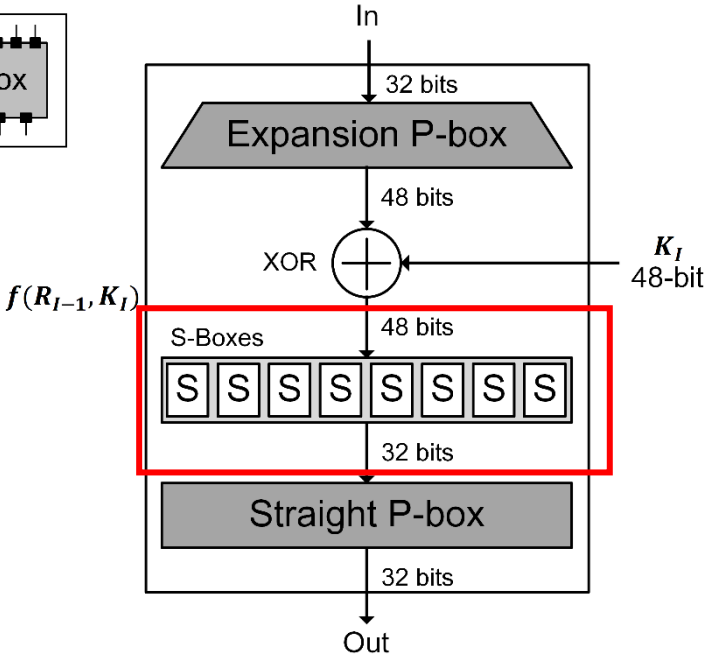
- 라운드 (4/7)
  - DES 함수 (3/6)
    - 과정 (2/3)

3. 8개의 S-박스를 통해 48비트를 32비트로 대치
   - 8개의 6비트로 분할하며 각 6비트 값은 하나의 S-박스에 삽입
   - S-박스는 섞어주는 역할을 하므로, 혼돈(Confusion) 발생



- 4행 16열을 갖는 표를 기반으로 하는 미리 정의된 규칙에 따라 대치



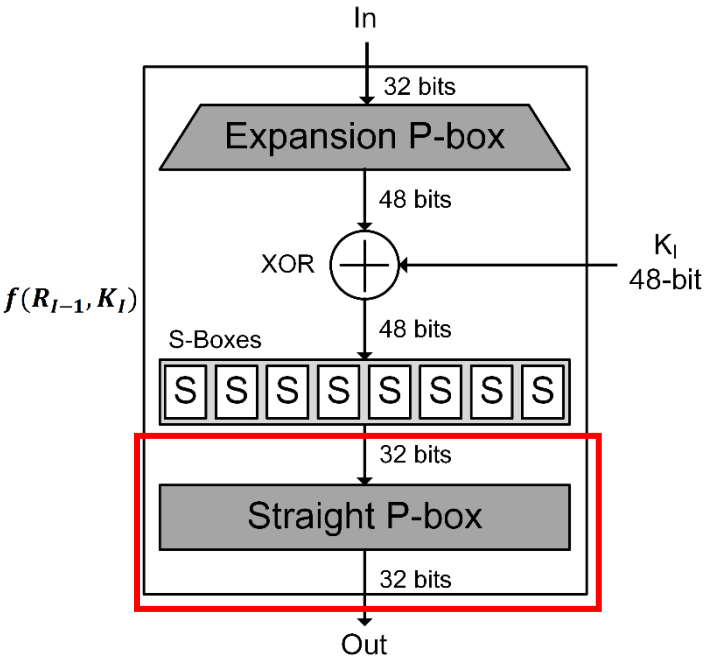


DES 구조

- 라운드 (5/7)
  - DES 함수 (4/6)
    - 과정 (3/3)

4. 단순 P-박스를 통해 32비트를 재배열하여 출력
   - 치환표들은 이전의 치환표와 동일한 규칙이며, 치환표에 따라 치환
   - S-박스 출력 비트를 다음 라운드의 여러 S-박스로 보내 평문 비트의 영향을 퍼뜨리는 확산(Diffusion) 발생

단순 P-박스	1	2	3	4	5	6	7	8
1~8	16	07	20	21	29	12	28	17
9~16	01	15	23	26	05	18	31	10
17~24	02	08	24	14	32	27	03	09
25~32	19	13	30	06	22	11	04	25



DES 구조

- 라운드 (6/7)
  - DES 함수 (5/6)
    - 예제 6.3

S-박스 1의 입력 값이 100011일 때 출력 값을 구하시오.

- 풀이
  1. 입력 비트 1과 6을 이은 2진수 11은 10진수 3이므로 3행 선택
  2. 입력 비트 2~5를 이은 2진수 0001은 10진수 1이므로 1열 선택
  3. S-박스 1 표의 3행 1열 값 12를 4비트로 바꾸면 2진수 1100

S-박스 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

∴ 출력 값 1100

DES 구조

- 라운드 (7/7)
  - DES 함수 (6/6)
    - 예제 6.4

S-박스 8의 입력 값이 000000일 때 출력 값을 구하시오.

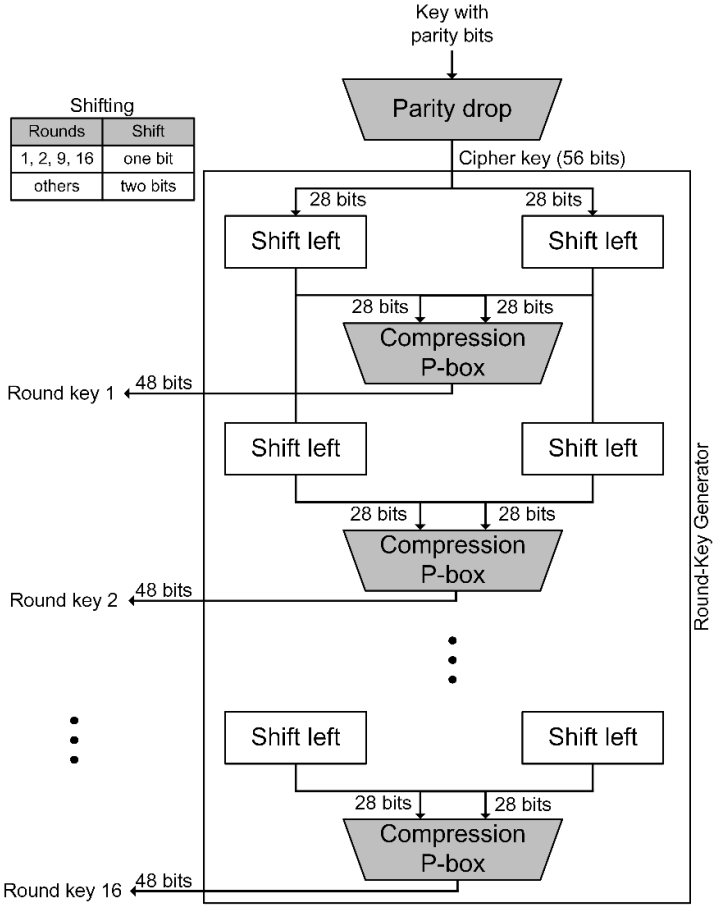
- 풀이
  1. 입력 비트 1과 6을 이은 2진수 00은 10진수 0이므로 0행 선택
  2. 입력 비트 2~5를 이은 2진수 0000은 10진수 0이므로 0열 선택
  3. S-박스 8 표의 0행 0열 값 13을 4비트로 바꾸면 2진수 1101

S-박스 8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

∴ 출력 값 1101

DES 구조

- 라운드 키 생성 (1/5)
  - 정의
    - 64비트 암호 키로 48비트 라운드 키 16개를 생성하는 과정
  - 특징
    - 패리티 비트를 뺀 56비트를 28비트씩 나누어 각각 순환 이동
    - 이전 라운드의 이동 결과를 다시 이동하여 다음 라운드 키 생성
  - 구성요소
    - 패리티 비트 제거
    - 좌측 순환 이동
    - 축소 P-박스



DES 구조

- 라운드 키 생성 (1/5)
  - *패리티 비트 제거
    - 정의
      - 64비트 키에서 패리티 비트 8개를 버리고 남은 56비트를 재배열하는 P-박스
    - 특징
      - 표에 8의 배수 번째 비트가 없어 패리티 비트는 출력에서 제외
      - 출력 56비트가 라운드 키 생성에 쓰는 실제 암호 키

*패리티 비트(Parity Bit)
바이트마다 1의 개수를 홀수로 맞추도록 덧붙여 키 입력 오류를 검사하는 비트

패리티 비트 제거	1	2	3	4	5	6	7	8
1~8	57	49	41	33	25	17	09	01
9~16	58	50	42	34	26	18	10	02
17~24	59	51	43	35	27	19	11	03
25~32	60	52	44	36	63	55	47	39
33~40	31	23	15	07	62	54	46	38
41~48	30	22	14	06	61	53	45	37
49~56	29	21	13	05	28	20	12	04

DES 구조

- 라운드 키 생성 (2/5)
  - 좌측 순환 이동(Shift Left)
    - 정의
      - 28비트를 왼쪽으로 1비트 혹은 2비트 이동하고 밀려난 비트를 오른쪽 끝에 붙이는 연산
    - 특징
      - 1, 2, 9, 16번째 라운드는 1비트, 나머지 라운드는 2비트 이동
      - 16라운드 동안 이동한 비트 수의 합이 28이므로 16라운드 후 두 28비트가 원래 값으로 복귀

라운드	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
이동 비트 수	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

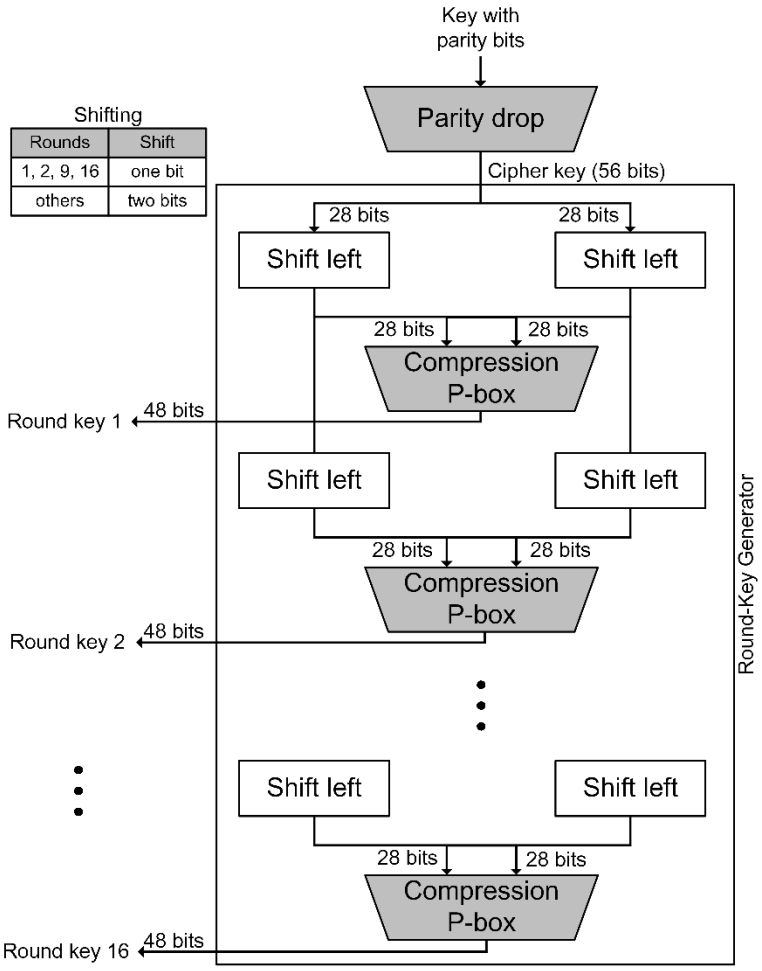
DES 구조

- 라운드 키 생성 (3/5)
  - 축소 P-박스(Compression P-box)
    - 정의
      - 56비트 입력을 48비트 라운드 키로 줄이는 P-박스
    - 특징
      - 왼쪽 28비트에서 24비트, 오른쪽 28비트에서 24비트를 골라 라운드 키 구성
      - 표에 없는 9, 18, 22, 25, 35, 38, 43, 54번째 비트는 라운드 키에서 제외

축소 P-박스	1	2	3	4	5	6	7	8
1~8	14	17	11	24	01	05	03	28
9~16	15	06	21	10	23	19	12	04
17~24	26	08	16	07	27	20	13	02
25~32	41	52	31	37	47	55	30	40
33~40	51	45	33	48	44	49	39	56
41~48	34	53	46	42	50	36	29	32

DES 구조

- 라운드 키 생성 (5/5)
  - 과정
    1. 패리티 비트 8개를 제거하여 64비트 키를 56비트로 변환
    2. 56비트를 왼쪽 28비트와 오른쪽 28비트로 분할
    3. 라운드마다 두 28비트를 각각 1비트 혹은 2비트 좌측 순환 이동
    4. 두 28비트를 합친 56비트를 축소 P-박스로 48비트 라운드 키로 변환
    5. 3~4단계를 16라운드 동안 반복하여 라운드 키 16개 생성
  - 라운드 키 생성 코드는 부록#2 참고

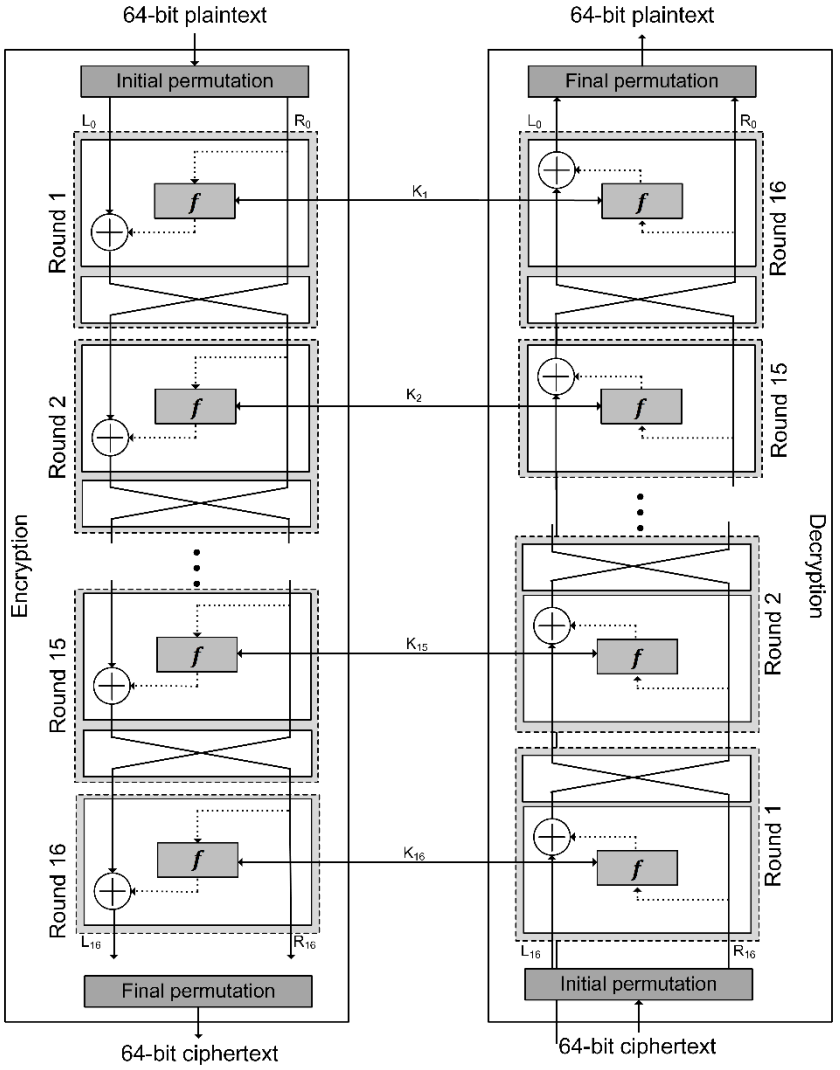


DES 구조

- 암호 알고리즘과 복호 알고리즘 (1/7)
  - 개요
    - 혼합기와 스와퍼를 사용하여 16라운드 암호 알고리즘과 복호 알고리즘 설계
    - 암호화와 복호화 코드는 부록#3 참고
  - 분류
    - 첫 번째 접근방법
    - 대체 접근방법

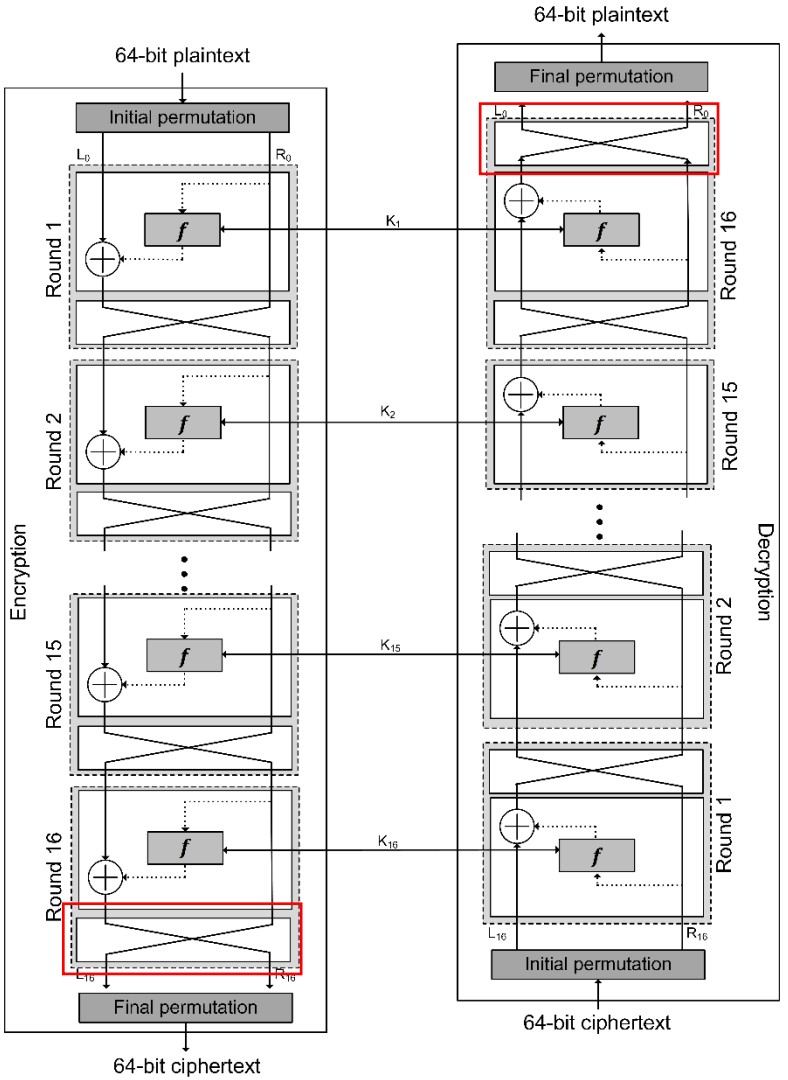
DES 구조

- 암호 알고리즘과 복호 알고리즘 (2/7)
  - 첫 번째 접근방법
    - 구조
      - 1~15라운드는 혼합기와 스와퍼, 16라운드는 스와퍼 없이 혼합기만 사용
      - 복호 알고리즘은 같은 구조에 라운드 키를 K_{16}부터 K_1까지 역순으로 입력
    - 특징
      - 암호화 I번째 라운드와 복호화 $(17-I)$번째 라운드의 혼합기가 같은 K_I를 써서 서로 상쇄
      - 초기 치환과 최종 치환도 서로 역관계이므로 라운드 키 순서만 바꾸어 원래 평문 복원



DES 구조

- 암호 알고리즘과 복호 알고리즘 (3/7)
  - 대체 접근방법
    - 구조
      - 16개 라운드를 모두 혼합기와 스와퍼로 같게 구성하고 16라운드 뒤에 스와퍼를 하나 더 추가
      - 복호 알고리즘은 같은 구조에 라운드 키를 역순으로 입력
    - 특징
      - 16라운드 스와퍼와 추가한 스와퍼가 연속하여 서로 상쇄되므로 첫 번째 접근방법과 같은 결과
      - 모든 라운드가 같은 구조이므로 같은 라운드 함수를 16번 반복 사용



DES 구조

- 암호 알고리즘과 복호 알고리즘 (4/7)
  - 예제 6.5 (1/2)

16진수 평문 123456ABCD132536을 16진수 키 AABB09182736CCDD로 암호화하여 라운드별 왼쪽 32비트, 오른쪽 32비트, 라운드 키와 암호문을 구하시오.

- 풀이
  1. 평문 123456ABCD132536을 초기 치환하면 14A7D67818CA18AD
  2. 64비트를 나누면 L_0은 14A7D678, R_0은 18CA18AD
  3. 라운드마다 라운드 키로 혼합기와 스와퍼를 적용한 1~8라운드 값은 오른쪽 표
     - 각 라운드의 R_I는 다음 라운드 혼합기를 그대로 통과한 후 스와퍼로 L_{I+1}이 되는 구조
     - e.g., 1라운드의 R_1인 5A78E394가 2라운드의 L_2

라운드	왼쪽 32비트	오른쪽 32비트	라운드 키
1	18CA18AD	5A78E394	194CD072DE8C
2	5A78E394	4A1210F6	4568581ABCCE
3	4A1210F6	B8089591	06EDA4ACF5B5
4	B8089591	236779C2	DA2D032B6EE3
5	236779C2	A15A4B87	69A629FEC913
6	A15A4B87	2E8F9C65	C1948E87475E
7	2E8F9C65	A9FC20A3	708AD2DDB3C0
8	A9FC20A3	308BEE97	34F822F0C66D

DES 구조

- 암호 알고리즘과 복호 알고리즘 (5/7)
  - 예제 6.5 (2/2)

16진수 평문 123456ABCD132536을 16진수 키 AABB09182736CCDD로 암호화하여 라운드별 왼쪽 32비트, 오른쪽 32비트, 라운드 키와 암호문을 구하시오.

- 풀이
  4. 9~16라운드 값은 오른쪽 표
     - 16라운드는 스와퍼가 없어 R_{15}인 CF26B472가 L_{16}이 아닌 R_{16}으로 출력
  5. L_{16}과 R_{16}을 결합하면 19BA9212CF26B472
  6. 최종 치환하면 암호문 C0B7A8D05F3A829C

  ∴ 라운드별 값은 1~8라운드 표와 9~16라운드 표, 암호문은 C0B7A8D05F3A829C

라운드	왼쪽 32비트	오른쪽 32비트	라운드 키
9	308BEE97	10AF9D37	84BB4473DCCC
10	10AF9D37	6CA6CB20	02765708B5BF
11	6CA6CB20	FF3C485F	6D5560AF7CA5
12	FF3C485F	22A5963B	C2C1E96A4BF3
13	22A5963B	387CCDAA	99C31397C91F
14	387CCDAA	BD2DD2AB	251B8BC717D0
15	BD2DD2AB	CF26B472	3330C5D9A36D
16	19BA9212	CF26B472	181C5D75C66D

DES 구조

- 암호 알고리즘과 복호 알고리즘 (6/7)
  - 예제 6.6 (1/2)

16진수 암호문 C0B7A8D05F3A829C를 같은 키 AABB09182736CCDD로 복호화하여 평문을 구하고, 복호화에 쓰는 라운드 키의 순서를 구하시오.

- 풀이
  1. 암호문을 초기 치환하면 19BA9212CF26B472이므로 L_0은 19BA9212, R_0은 CF26B472
     - 암호화 16라운드 출력 L_{16}, R_{16}과 같은 값
  2. 라운드 키를 역순으로 넣어 1라운드에 K_{16}, 16라운드에 K_1을 사용한 라운드별 값은 오른쪽 표
     - 복호화 1라운드 키 181C5D75C66D는 암호화 16라운드 키와 같은 값

라운드	왼쪽 32비트	오른쪽 32비트	라운드 키
1	CF26B472	BD2DD2AB	181C5D75C66D
2	BD2DD2AB	387CCDAA	3330C5D9A36D
…	…	…	…
15	5A78E394	18CA18AD	4568581ABCCE
16	14A7D678	18CA18AD	194CD072DE8C

DES 구조

- 암호 알고리즘과 복호 알고리즘 (7/7)
  - 예제 6.6 (2/2)

16진수 암호문 C0B7A8D05F3A829C를 같은 키 AABB09182736CCDD로 복호화하여 평문을 구하고, 복호화에 쓰는 라운드 키의 순서를 구하시오.

- 풀이
  3. 복호화 I번째 라운드 출력은 암호화 $(16-I)$번째 라운드 출력의 왼쪽과 오른쪽을 바꾼 값이며 아래 표에서 확인 (단, $1 \leq I \leq 15$)
  4. 16라운드 출력을 결합하면 암호화의 초기 치환 결과와 같은 14A7D67818CA18AD
  5. 최종 치환하면 평문 123456ABCD132536

∴ 평문 123456ABCD132536, 라운드 키 순서는 $K_{16}, K_{15}, \cdots, K_1$

출력 단계	암호화 왼쪽	암호화 오른쪽	복호화 왼쪽	복호화 오른쪽
초기 치환 후	14A7D678	18CA18AD	19BA9212	CF26B472
1라운드 후	18CA18AD	5A78E394	CF26B472	BD2DD2AB
15라운드 후	BD2DD2AB	CF26B472	5A78E394	18CA18AD
16라운드 후	19BA9212	CF26B472	14A7D678	18CA18AD

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

DES 분석

- 성질 (1/2)
  - 쇄도 효과(Avalanche Effect)
    - 평문 혹은 키의 작은 변화가 암호문의 많은 비트 변화로 이어지는 성질
    - S-박스가 입력 1비트 변화를 출력 2비트 이상 변화로 늘리고, P-박스가 바뀐 비트를 다음 라운드의 여러 S-박스로 퍼뜨려 DES는 쇄도 효과가 강한 블록 암호
  - 완전성 효과(Completeness Effect)
    - 암호문의 각 비트가 평문의 많은 비트에 의존하는 성질
    - P-박스의 확산과 S-박스의 혼돈으로 DES는 완전성 효과가 강한 블록 암호

DES 분석

- 성질 (2/2)
  - 예제 6.7

16진수 키 22234512987ABB23으로 평문 0000000000000000과 0000000000000001을 각각 암호화하여 두 암호문의 다른 비트 수와 라운드별 다른 비트 수를 구하시오.

- 풀이
  1. 두 평문은 마지막 1비트만 다른 64비트 블록
  2. 두 암호문은 4789FD476E82A5F1과 0A4ED5C15A63FEA3
  3. 두 암호문을 XOR한 4DC7288634E15B52에서 1의 개수를 세면 29
  4. 라운드마다 두 64비트 출력의 다른 비트 수는 아래 표
     - 3라운드에서 이미 20비트가 다르고 4라운드부터 28~39비트 차이 유지

∴ 두 암호문은 64비트 중 29비트(약 45%) 차이, 라운드별 다른 비트 수는 아래 표

라운드	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
다른 비트 수	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

DES 분석

- 설계 기준 (1/2)
  - S-박스에 대한 설계기준
    - 각 라운드의 출력이 다음 라운드까지 혼돈과 확산을 만족하도록 정한 기준
      - 각 행은 0~15를 한 번씩 포함하는 치환
      - 출력을 입력 비트들의 XOR와 상수 XOR로 나타낼 수 없는 비선형 함수
      - 입력 1비트를 바꾸면 출력이 2비트 이상 변화
      - 가운데 두 비트(비트 3, 4)만 다른 두 입력의 출력은 2비트 이상 차이
      - 앞 두 비트(비트 1, 2)가 다르고 뒤 두 비트(비트 5, 6)가 같은 두 입력의 출력은 서로 다른 값
      - 두 입력의 XOR인 입력 차분이 같은 입력 쌍 32개 중 출력 차분이 같은 쌍은 최대 8개
      - 입력 1비트를 고정하고 나머지 비트를 바꿀 때 출력의 0과 1 개수 차이가 최소

DES 분석

- 설계 기준 (2/2)

*알려진 평문 공격(Known-plaintext Attack)
공격자가 같은 키로 만든 평문과 암호문 쌍을 알고 키를 찾는 공격

  - P-박스에 대한 설계기준
    - 연속된 두 라운드의 S-박스 사이에 있는 단순 P-박스와 확장 P-박스가 확산을 만들도록 정한 기준
      - S-박스의 각 입력 비트는 이전 라운드의 서로 다른 S-박스에서 오고 같은 번호의 S-박스에서는 오지 않는 구조
      - 한 S-박스의 출력 4비트는 다음 라운드의 서로 다른 6개 S-박스로 입력
      - S_j의 앞 두 비트에 S_{j-2}, 뒤 두 비트에 S_{j-1}, 가운데 두 비트에 S_{j+1}의 출력 비트가 하나씩 입력 (단, S_j는 j번째 S-박스)
      - S_j의 출력 비트가 S_k의 가운데 비트로 가면 S_k의 출력 비트는 S_j의 가운데 비트로 가지 않는 구조
  - 라운드 수에 대한 설계기준
    - 8라운드 이후 암호문의 모든 비트가 평문과 키의 모든 비트에 의존하여 8라운드로 충분해 보이는 구조
    - 16라운드보다 적은 DES는 전수조사 공격보다 적은 계산의 *알려진 평문 공격에 해독되므로 16라운드 사용

DES 분석

- 알고리즘 설계의 취약점
  - S-박스
    - 4번째 S-박스 출력의 하위 3비트는 일부 입력 비트의 0과 1을 반전하여 첫 번째 출력 비트와 같은 방법으로 계산 가능
    - 특별히 선택한 두 입력이 한 S-박스에서 같은 출력을 만드는 경우 존재
    - 이웃한 3개의 S-박스 입력 비트만 바꾸어 한 라운드의 출력을 같게 만드는 경우 존재
  - P-박스
    - 초기 치환과 최종 치환은 사용 이유가 공개되지 않았고 안전성에 기여하지 않는 구성요소
    - 확장 P-박스는 입력을 4비트씩 나누었을 때 첫 번째와 네 번째 비트를 두 번씩 출력하는 반복 구조

DES 분석

- 암호 키의 취약점 (1/11)

*전수조사 공격(Brute-force Attack)
가능한 모든 키를 하나씩 대입하여 올바른 키를 찾는 공격

  - 키 크기
    - 패리티 비트를 뺀 실제 암호 키가 56비트이므로 *전수조사 공격 시 최대 2^{56}개의 키 검사
    - 56비트 키로 전수조사 공격에 취약하므로, 112비트 혹은 168비트 키의 삼중 DES로 공격 방어
      - e.g., 초당 10^6개 키를 검사하는 칩 10^6개로 병렬 처리하면 $2^{56}/10^{12} \approx 7.2\times10^4$초, 약 20시간 소요
      - e.g., 인터넷에 연결된 컴퓨터 3,500대가 키 공간을 나누어 전수조사하여 120일 만에 키 탐색

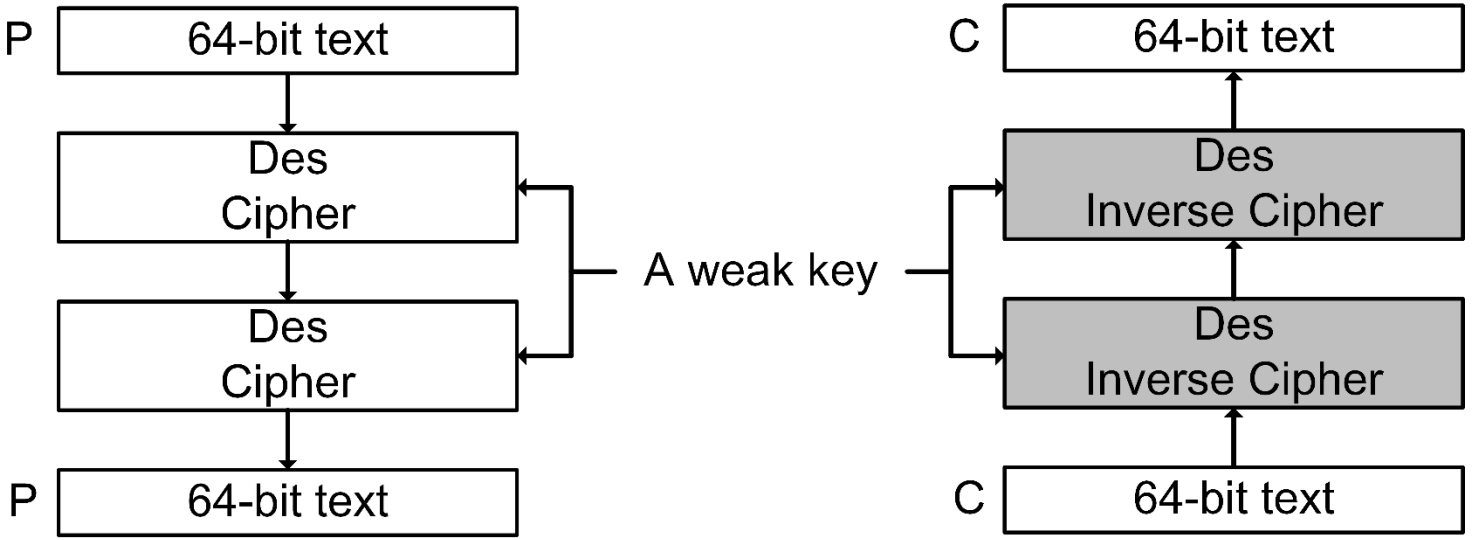
DES 분석

- 암호 키의 취약점 (2/11)
  - 취약 키(Weak Key) (1/2)
    - 정의
      - 패리티 비트 제거 후 56비트가 모두 0, 모두 1, 혹은 한쪽 28비트는 0이고 다른 쪽 28비트는 1인 4개의 키
    - 특징 (1/2)
      - 두 28비트가 각각 한 가지 비트로만 이루어져 좌측 순환 이동과 축소 P-박스를 거쳐도 16개 라운드 키가 모두 같은 값
      - 라운드 키가 암호 키와 같은 패턴
        - e.g., 0101010101010101의 라운드 키는 모두 0, 1F1F1F1F0E0E0E0E의 라운드 키는 절반이 0이고 절반이 1

패리티 비트 제거 전 키(64비트)	실제 키(56비트)
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

DES 분석

- 암호 키의 취약점 (3/11)
  - 취약 키(Weak Key) (2/2)
    - 특징 (2/2)
      - 같은 키로 두 번 암호화하면 원래 평문으로 복원되는 역관계
        - $E_k(E_k(P)) = P$로 표기 (단, E_k는 키 k로 암호화하는 함수)
      - 취약 키가 4개뿐이라 공격자가 가로챈 암호문에 4개 키를 바로 시험할 수 있어 취약하므로, 키 생성 시 취약 키를 제외하여 공격 방어



DES 분석

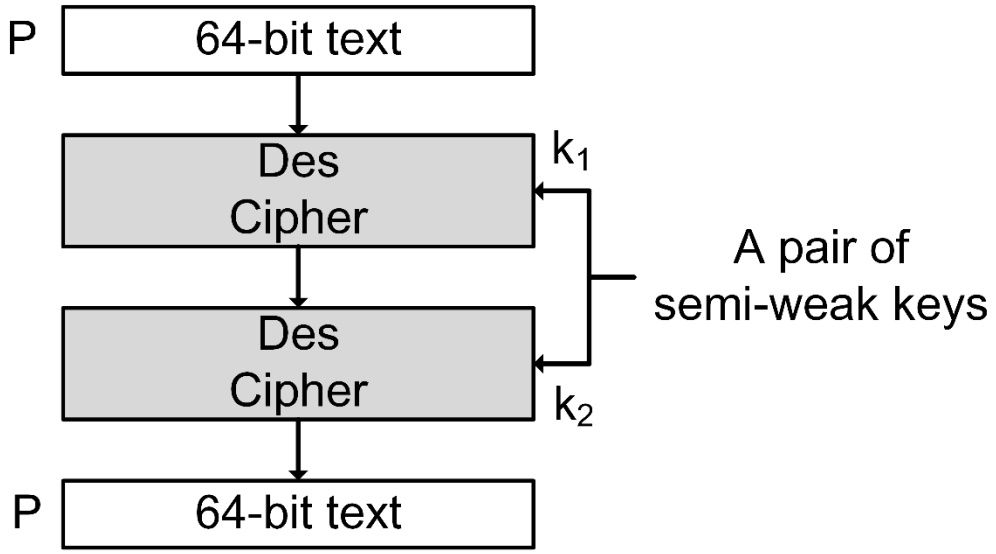
- 암호 키의 취약점 (4/11)
  - 준-취약 키(Semi-weak Key) (1/2)
    - 정의
      - 두 가지 라운드 키만 8번씩 생성하는 키로, 아래 6쌍 12개 존재
    - 특징 (1/2)
      - 16개 라운드 키가 두 값으로만 구성되며 각 값이 8번씩 반복
      - 쌍을 이루는 두 키는 같은 라운드 키를 반대 순서로 생성

쌍의 첫 번째 키	쌍의 두 번째 키
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E0 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

키	1라운드	2~8라운드	9~15라운드	16라운드
01FE01FE01FE01FE	9153E54319BD	6EAC1ABCE642	9153E54319BD	6EAC1ABCE642
FE01FE01FE01FE01	6EAC1ABCE642	9153E54319BD	6EAC1ABCE642	9153E54319BD

DES 분석

- 암호 키의 취약점 (5/11)
  - 준-취약 키(Semi-weak Key) (2/2)
    - 특징 (2/2)
      - 한 키로 암호화한 블록을 쌍의 다른 키로 다시 암호화하면 원래 평문으로 복원되는 역관계
        - $E_{k_2}(E_{k_1}(P)) = P$로 표현 가능 (단, k_1, k_2는 쌍의 두 키)
      - 쌍의 다른 키로 암호화하는 것이 복호화와 같아 취약하므로, 키 생성 시 준-취약 키를 제외하여 공격 방어



DES 분석

- 암호 키의 취약점 (6/11)
  - 가능한 취약 키(Possible Weak Key)
    - 정의
      - 네 가지 라운드 키만 4번씩 생성하는 48개의 키
    - 특징
      - 16개 라운드 키가 같은 라운드 키 4개씩 4개 그룹으로 구성
      - 같은 라운드 키가 4번씩 반복되어 라운드마다 다른 키를 쓰는 효과가 줄어 취약하므로, 키 생성 시 가능한 취약 키를 제외하여 공격 방어
  - 취약 키, 준-취약 키, 가능한 취약 키 비교

구분	취약 키	준-취약 키	가능한 취약 키
개수	4개	12개 (6쌍)	48개
16개 라운드 키 구성	1가지 값이 16번	2가지 값이 8개씩	4가지 값이 4번씩
두 번 암호화하여 평문이 되는 키	같은 키	쌍의 다른 키	없음

DES 분석

- 암호 키의 취약점 (7/11)
  - 키 보수(Key Complement)
    - 정의
      - 키의 모든 비트를 반전한 키
    - 특징
      - 키와 평문을 모두 보수로 바꾸어 암호화하면 원래 암호문의 보수 생성
        - $C = E_K(P)$일 경우, $\bar{C} = E_{\bar{K}}(\bar{P})$인 관계 (단, $\bar{K}$, $\bar{P}$, $\bar{C}$는 K, P, C의 보수)
      - 시험한 키의 보수 키 결과도 함께 얻어 전수조사할 키가 2^{55}개로 줄어 취약하므로, 112비트 혹은 168비트 키의 삼중 DES로 공격 방어

DES 분석

- 암호 키의 취약점 (8/11)
  - 키 클러스터링(Key Clustering)
    - 정의
      - 서로 다른 2개 이상의 키가 같은 평문을 같은 암호문으로 암호화하는 현상
    - 특징
      - 준-취약 키의 각 쌍을 하나의 키 클러스터로 간주
      - 준-취약 키 쌍 외의 키 클러스터는 아직 발견되지 않은 상태

DES 분석

- 암호 키의 취약점 (9/11)
  - 예제 6.8

16진수 취약 키 0101010101010101로 평문 1234567887654321을 암호화한 결과를 같은 키로 다시 암호화하여 원래 평문으로 복원되는지 확인하시오.

- 풀이
  1. 평문 1234567887654321을 키 0101010101010101로 암호화하면 814FE938589154F7
  2. 814FE938589154F7을 같은 키로 다시 암호화하면 1234567887654321
  3. 복호 알고리즘이 아니라 암호 알고리즘을 두 번 사용하여 원래 평문으로 복원

  ∴ 두 번 암호화한 결과 1234567887654321이 원래 평문과 같아 복원 확인

DES 분석

- 암호 키의 취약점 (10/11)
  - 예제 6.9

DES 키 2^{56}개 중 임의로 고른 키가 취약 키, 준-취약 키, 가능한 취약 키 중 하나일 확률을 구하시오.

- 풀이
  1. 취약 키 4개, 준-취약 키 12개, 가능한 취약 키 48개를 더하면 64개
  2. 전체 키 2^{56}개 중 64개를 고를 확률은 $64/2^{56} = 2^{-50}$
  3. $2^{-50} \approx 8.88\times10^{-16}$이므로 임의로 고른 키가 이 키들일 가능성은 무시할 수 있는 수준

$\therefore 2^{-50} \approx 8.88\times10^{-16}$

DES 분석

- 암호 키의 취약점 (11/11)
  - 예제 6.10

16진수 키 1234123412341234로 평문 12345678ABCDEF12를 암호화한 암호문과, 키와 평문의 보수로 암호화한 암호문을 비교하시오.

- 풀이
  1. 원래 키와 평문으로 암호화하면 암호문 E112BE1DEFC7A367
  2. 키의 보수 EDCBEDCBEDCBEDCB와 평문의 보수 EDCBA987543210ED로 암호화하면 암호문 1EED41E210385C98
  3. 1EED41E210385C98은 E112BE1DEFC7A367의 각 비트를 반전한 값
     - 16진수 자리마다 15에서 뺀 값이므로 E112는 1EED로 반전

구분	원래 값	보수
키	1234123412341234	EDCBEDCBEDCBEDCB
평문	12345678ABCDEF12	EDCBA987543210ED
암호문	E112BE1DEFC7A367	1EED41E210385C98

∴ 키와 평문의 보수로 만든 암호문 1EED41E210385C98은 원래 암호문 E112BE1DEFC7A367의 보수

목 차

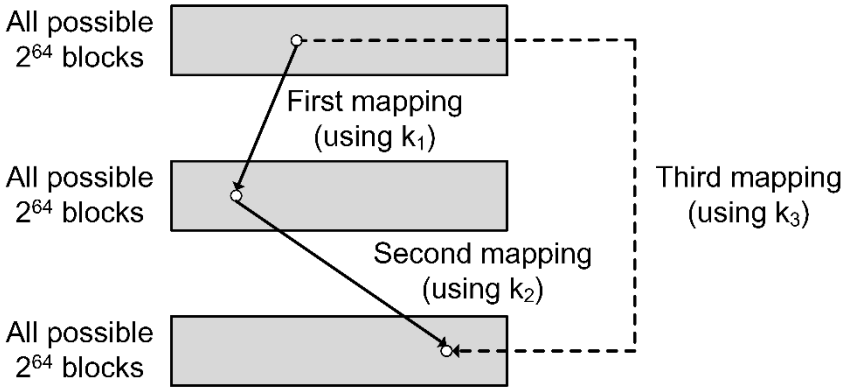
- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

다중 DES

- 정의
  - 서로 다른 여러 키로 DES를 여러 번 적용하여 키 길이를 늘린 암호

*군(Group)
연산 결과가 다시 집합 안에 있고 결합 법칙, 항등원, 역원을 만족하는 집합

- 특징
  - DES를 그대로 쓰므로 새 소프트웨어와 하드웨어 없이 56비트 키의 짧은 길이를 보완
  - DES는 *군이 아니므로 두 키로 두 번 암호화한 결과를 한 키로 얻을 수 없어 키 길이 확장 가능
- 종류
  - 이중 DES (2DES, Double DES)
  - 삼중 DES (3DES, Triple DES)

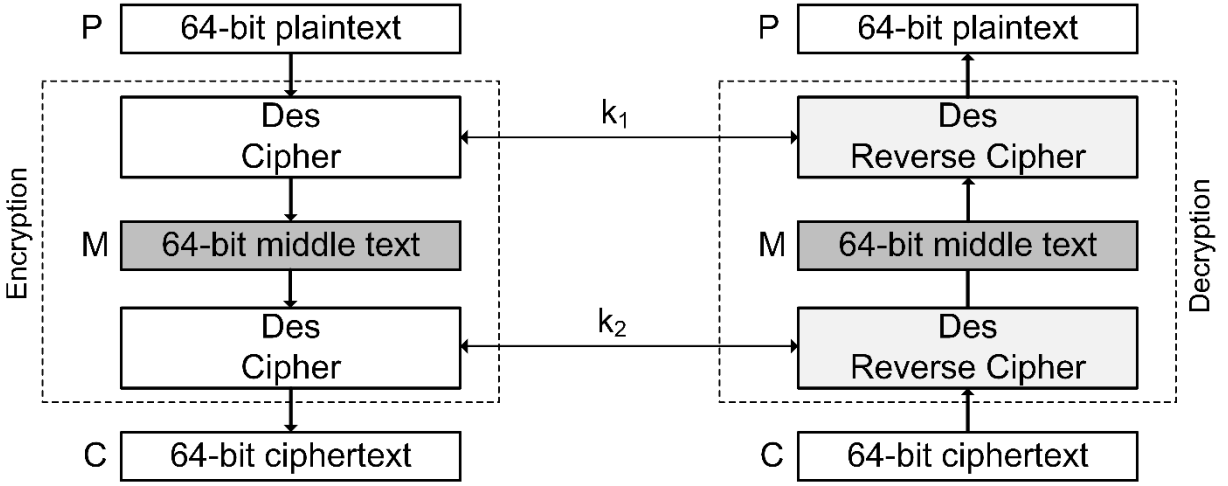


다중 DES

- 이중 DES (2DES, Double DES) (1/3)
  - 정의
    - 서로 다른 두 키 k_1, k_2로 DES를 두 번 적용하는 다중 DES
  - 특징
    - 평문 P를 k_1로 암호화한 중간 텍스트 M을 k_2로 다시 암호화하여 암호문 C 생성
    - 복호화는 C를 k_2, k_1 순서로 두 번 복호화하여 P 복원
    - 키가 112비트로 늘지만 중간일치 공격(Meet-in-the-Middle Attack)에 취약하므로, 삼중 DES로 공격 방어

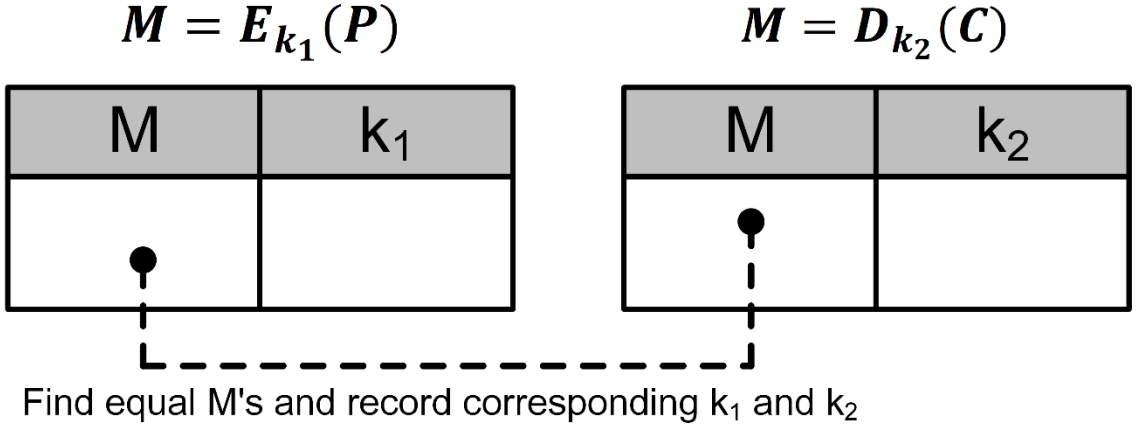
다중 DES

- 이중 DES (2DES, Double DES) (2/3)
  - 중간일치 공격(Meet-in-the-Middle Attack) (1/2)
    - 정의
      - 평문을 암호화한 중간 텍스트와 암호문을 복호화한 중간 텍스트가 같아지는 키 쌍을 찾는 알려진 평문 공격
    - 특징
      - k_1과 k_2를 따로 탐색하여 2^{112}번 대신 2^{56}번 탐색 두 번, 약 2^{57}번만에 키 탐색
      - 이중 DES의 안전도는 단일 DES의 2^{56}번에서 2^{57}번으로 2배 증가한 수준



다중 DES

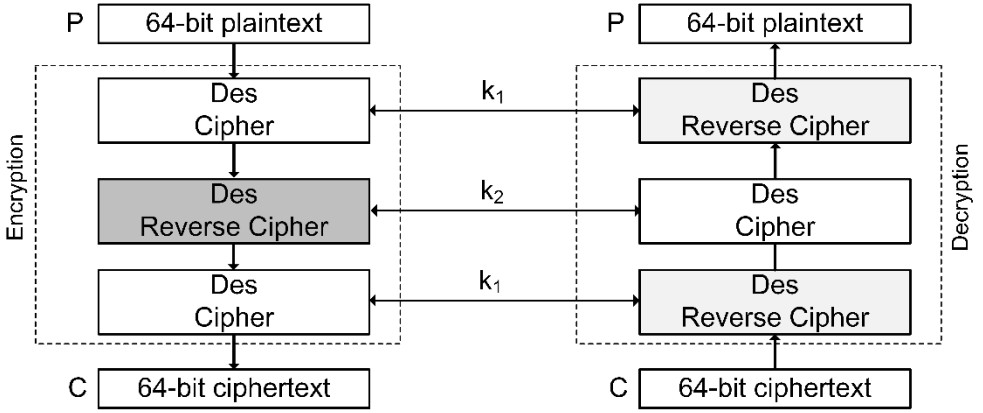
- 이중 DES (2DES, Double DES) (3/3)
  - 중간일치 공격(Meet-in-the-Middle Attack) (2/2)
    - 과정
      1. 같은 키로 만든 평문 P와 암호문 C 쌍 확보
      2. k_1후보 2^{56}개로 P를 암호화한 중간 텍스트 M과 k_1을 표로 기록
      3. k_2후보 2^{56}개로 C를 복호화한 M과 k_2를 표로 기록



      4. 두 표를 M 값으로 정렬하여 M이 같은 k_1, k_2 쌍을 후보로 기록
      5. 후보가 2개 이상이면 다른 평문과 암호문 쌍으로 검사하여 하나가 남을 때까지 반복

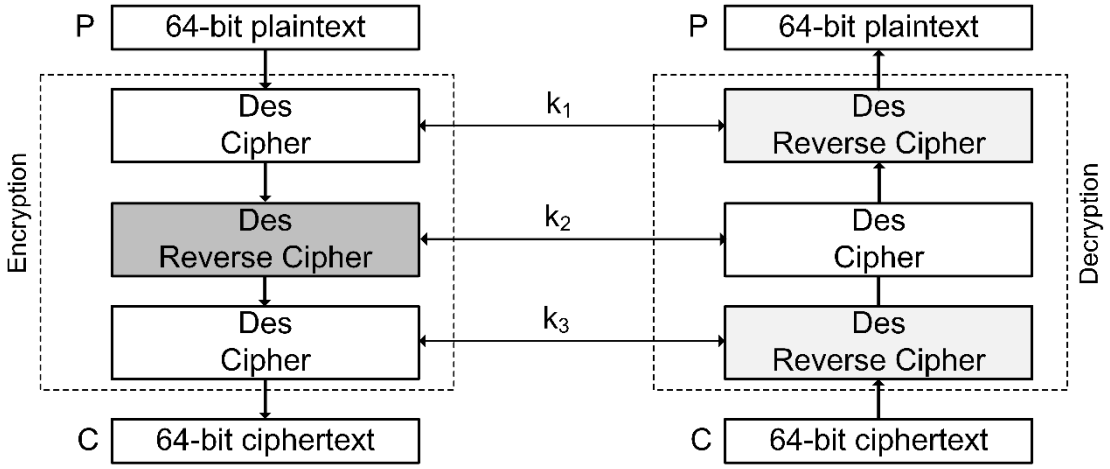
다중 DES

- 삼중 DES (3DES, Triple DES) (1/2)
  - 두 개의 키를 갖는 삼중 DES
    - 정의
      - 첫 번째와 세 번째 단계에 k_1, 두 번째 단계에 k_2를 쓰는 112비트 키의 삼중 DES
    - 특징
      - 암호화는 k_1 암호화, k_2 복호화, k_1 암호화 순서
      - $k_1 = k_2$일 경우, 두 번째 단계가 첫 번째 단계를 상쇄하여 단일 DES와 호환
      - 알려진 평문 공격에 취약하므로, 세 개의 키를 갖는 삼중 DES로 공격 방어
    - 활용
      - 은행 같은 금융기관에서 사용



다중 DES

- 삼중 DES (3DES, Triple DES) (2/2)
  - 세 개의 키를 갖는 삼중 DES
    - 정의
      - 세 키 k_1, k_2, k_3으로 DES를 세 번 적용하는 168비트 키의 삼중 DES
    - 특징
      - 암호화는 k_1 암호화, k_2 복호화, k_3 암호화 순서
      - $k_2 = k_3$이면 두 번째와 세 번째 단계가 서로 상쇄되어 키 k_1의 단일 DES와 호환
      - 세 번째 키를 추가하여 두 개의 키를 갖는 삼중 DES의 알려진 평문 공격 가능성을 방어
    - 활용
      - PGP (Pretty Good Privacy) 같은 응용 프로그램에서 사용



목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

DES의 안전성

- 전수조사 공격
  - 56비트 키는 현재 기술로 전수조사가 가능한 길이
  - 키 보수 성질로 2^{55}번의 암호화만으로 키를 찾을 수 있어 취약하므로, 112비트 혹은 168비트 키의 삼중 DES로 공격 방어 가능

- *통계적인 공격
  - 평문을 64비트 블록 단위로 암호화하므로 문자 하나의 빈도가 암호문에 그대로 드러나지 않는 구조
  - S-박스의 혼돈과 P-박스의 확산으로 평문 1비트 차이도 암호문 전체에 퍼뜨려 통계적 특징을 숨겨 공격 방어 가능

*통계적인 공격(Statistical Attack)
평문 언어의 문자 빈도 같은 통계적 특징을 이용하여 평문이나 키를 찾는 공격

DES의 안전성

- 차분 분석
  - 설계자가 차분 분석을 알고 S-박스와 16라운드를 설계하여 저항하는 구조
  - 2^{47}개의 선택 평문이 필요하므로, 같은 키로 만든 평문과 암호문 쌍이 2^{47}개 쌓이기 전에 키를 바꾸어 공격 방어 가능

*차분 분석(Differential Cryptanalysis)
두 평문의 XOR 차이와 두 암호문의 XOR 차이 사이의 관계로 키를 찾는 선택 평문 공격
*선택 평문 공격(Chosen-plaintext Attack)
공격자가 고른 평문의 암호문을 얻어 키를 찾는 공격

- 선형 분석
  - 차분 분석보다 나중에 개발되어 DES의 S-박스 설계에 반영되지 않은 공격
  - 2^{43}개의 알려진 평문 쌍으로 해독할 수 있어 취약하므로, 같은 키로 만든 쌍이 2^{43}개 쌓이기 전에 키를 바꾸어 공격 방어 가능

*선형 분석(Linear Cryptanalysis)
평문, 암호문, 키 비트 사이의 선형 근사식이 성립하는 확률의 치우침으로 키를 찾는 알려진 평문 공격

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

개요

- AES (Advanced Encryption Standard) (1/2)
  - 정의
    - 미국 국립기술표준원이 DES를 대체하기 위하여 공표한 대칭-키 블록 암호
  - 역사
    - 1997년 미국 국립기술표준원이 128비트 블록과 128비트, 192비트, 256비트 키를 쓰는 공개 알고리즘을 공모
    - 2000년 최종 후보 5개 중 벨기에 학자 Joan Daemen과 Vincent Rijmen이 설계한 Rijndael을 AES로 선정
    - 2001년 미국 연방 표준으로 공표

개요

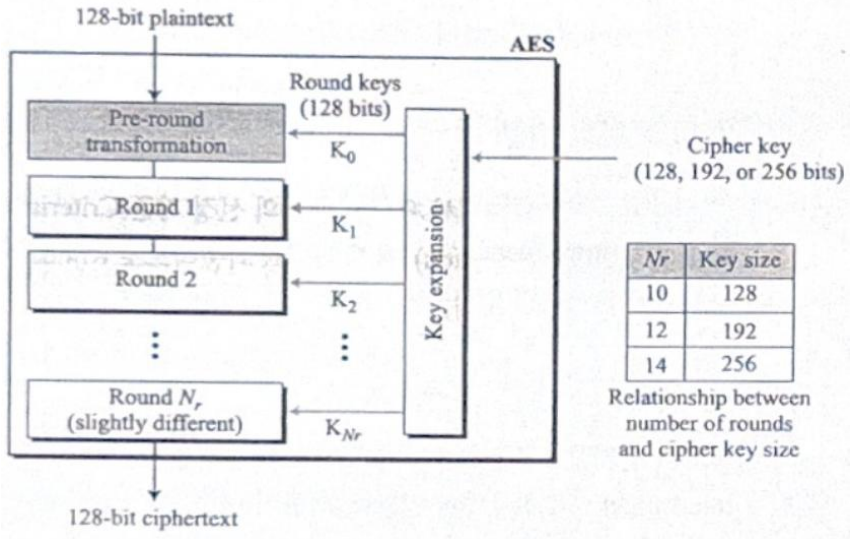
- AES (Advanced Encryption Standard) (2/2)
  - 특징
    - 128비트 블록을 16바이트로 보고 바이트 단위로 변환하는 바이트 기반(Byte-oriented) 암호
    - 128비트 평문 블록을 128비트, 192비트, 256비트 중 하나의 암호 키로 암호화하여 128비트 암호문 블록 생성
    - 블록을 두 절반으로 나누어 처리하는 Feistel 구조가 아닌, 역변환이 있는 변환을 차례로 적용하는 non-Feistel 암호
    - 바이트를 $GF(2^8)$ 체의 원소로 보고 바이트 값을 바꾸는 변환을 체의 곱셈과 덧셈으로 정의

개요

- 선정 기준(Criteria)
  - 안전성(Security)
    - 128비트 키를 요구하여 모든 키를 대입하는 전수조사 공격에 대비하고 그 밖의 암호 분석에 대한 저항성도 평가
  - 비용(Cost)
    - 하드웨어, 소프트웨어, 스마트카드 환경에서의 계산 효율성과 필요한 저장 공간
  - 구현(Implementation)
    - 어느 플랫폼에서도 구현할 수 있는 유연성과 알고리즘 구조의 단순성

개요

- 라운드
  - 개요
    - 128비트 평문을 라운드 전 변환(Pre-round Transformation)과 여러 라운드로 처리하여 128비트 암호문 생성
    - 키 확장(Key Expansion)이 암호 키로 라운드 전 변환과 각 라운드에 쓸 128비트 라운드 키 생성
    - 암호 키 길이에 따라 라운드 수 N_r이 다른 AES-128, AES-192, AES-256의 세 가지 버전

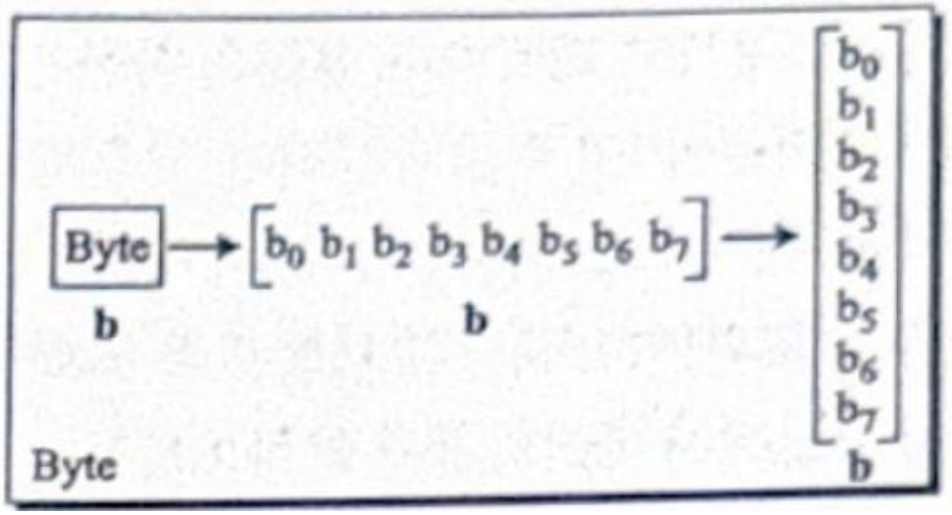


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

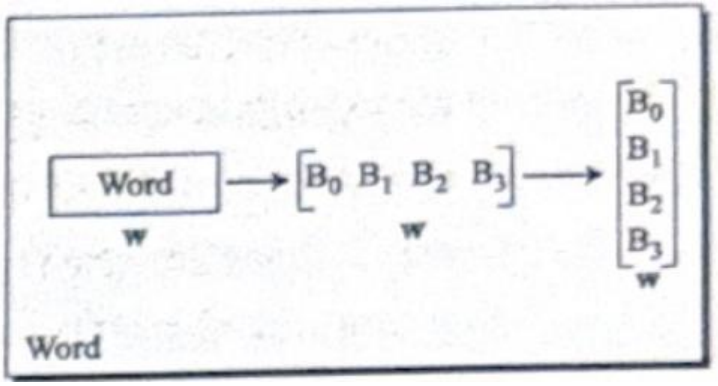
개요

- 데이터 단위 (1/4)
  - 비트(Bit)
    - 0 혹은 1의 2진 값으로 다른 데이터 단위를 이루는 최소 요소
    - 표현 시, b_1, b_2 와 같이 소문자로 표현
  - 바이트(Byte)
    - 8비트를 원소로 하는 1×8 행 행렬 및 8×1 열 행렬로 표현
    - 표현 시, 볼드 체 소문자 **b**로 표현

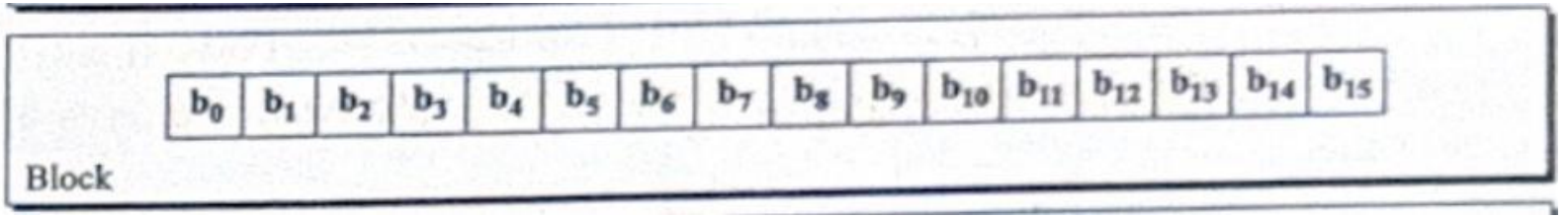


개요

- 데이터 단위 (2/4)
  - 워드(Word)
    - 4바이트를 원소로 하는 1×4 행 행렬 및 4×1 열 행렬로 표현
    - 표현 시, 볼드 체 소문자 **w**로 표현

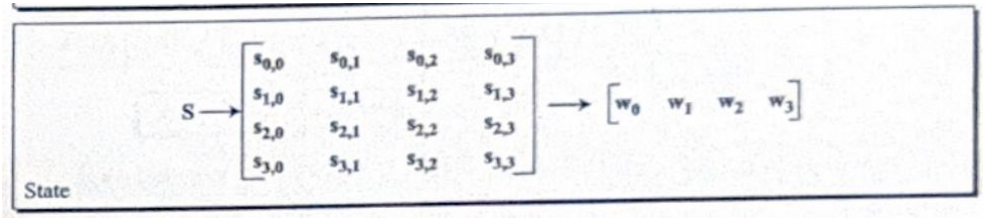


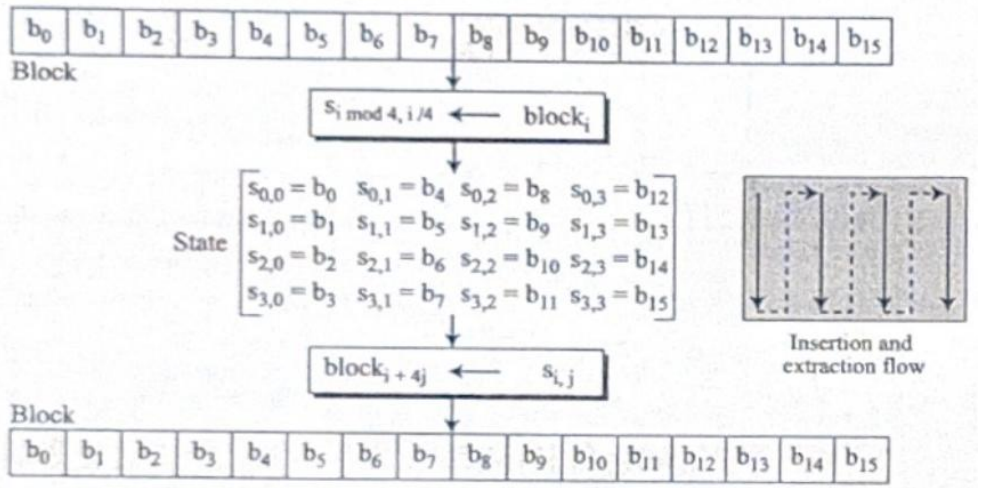
  - 블록(Block)
    - 16바이트를 원소로 하는 1×16 행 행렬로 표현



개요

- 데이터 단위 (3/4)
  - 상태(State)
    - 라운드 안의 각 변환 단계 전후에 있는 128비트 데이터 블록
    - $s_{0,0}$ ~ $s_{3,3}$의 16바이트를 원소로 하는 4×4 행렬
    - 블록의 바이트를 열 단위로 위에서 아래로 넣어 블록의 연속한 4바이트, 즉 워드 하나가 상태의 한 열
      - e.g., 블록의 i번째 바이트는 $i = 4c + r$인 $s_{r,c}$에 저장



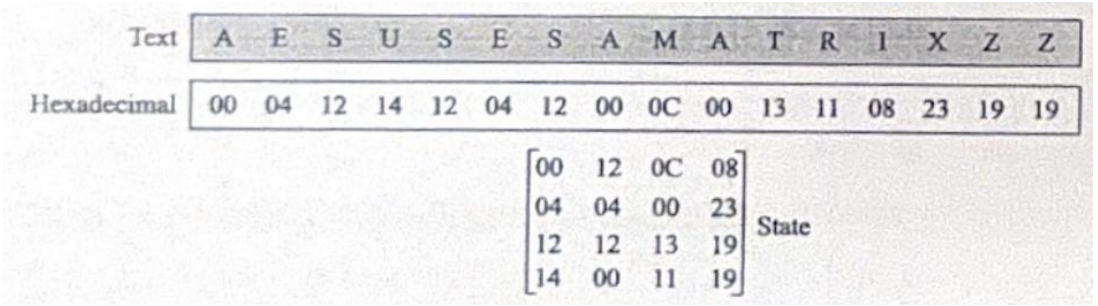


개요

- 데이터 단위 (4/4)
  - 예제 7.1

텍스트 "AES uses a matrix."의 공백과 마침표를 빼고 끝에 가짜 문자 ZZ를 덧붙인 16문자 블록 AESUSESAMATRIXZZ를 상태로 나타내시오. 단, 각 문자는 A ~ Z를 0 ~ 25의 정수로 바꾼 후 16진수 1바이트로 나타낸다.

- 풀이
  1. 문자를 차례로 0 ~ 25의 정수로 바꾼 후 16진수 1바이트로 변환
     - e.g., S는 $18 = 12_{16}$, X는 $23 = 17_{16}$
     - 00 04 12 14 12 04 12 00 0C 00 13 11 08 17 19 19
  2. 16바이트를 4바이트씩 열 단위로, 각 열은 위에서 아래로 상태에 삽입



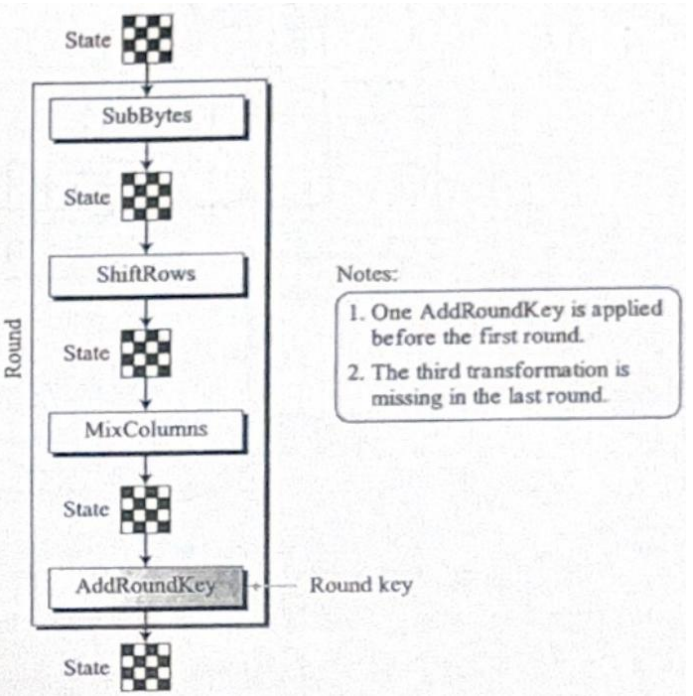
∴ 상태의 0~3행은 차례로 00 12 0C 08, 04 04 00 17, 12 12 13 19, 14 00 11 19

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

변환

- 라운드 함수
  - 개요
    - 라운드 전에는 *AddRoundKey만 적용
    - $1 \sim (N_r - 1)$번째 라운드는 4가지 변환을 차례로 적용
      - *SubBytes, *ShiftRows, *MixColumns, AddRoundKey 순서
    - 마지막 라운드는 MixColumns를 뺀 3가지 변환만 적용
      - 네 변환과 역변환의 코드는 부록 #5 참고
  - 구조
    - 대치(Substitution)
    - 치환 또는 이동 연산(Shifting)
    - 뒤섞음(Mixing)
    - 키 덧셈(Key Adding)



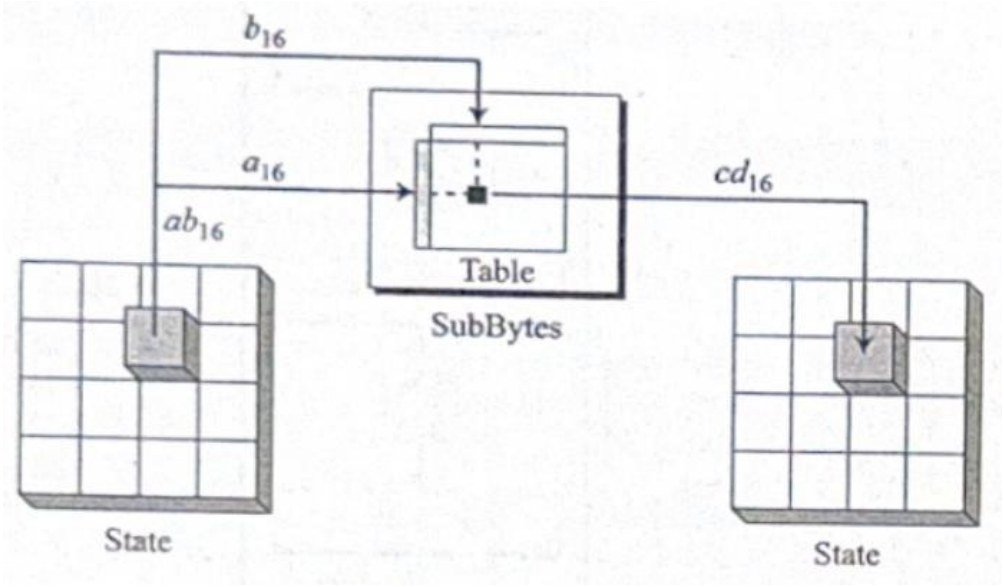
*AddRoundKey
상태에 라운드 키를 XOR하는 변환
*SubBytes
S-박스 표로 상태의 각 바이트를 대치하는 변환
*ShiftRows
상태의 각 행을 왼쪽으로 순환 이동하는 변환
*MixColumns
상수 행렬을 곱하여 상태의 각 열을 뒤섞는 변환

변환

- 대치(Substitution) (1/7)
  - 부분 바이트(SubBytes) (1/2)
    - 정의
      - SubBytes 표로 상태의 각 바이트를 다른 바이트로 대치하는 변환
        - SubBytes 표는 부록#4 참고
    - 특징
      - 16바이트 모두 같은 표 하나로 독립적으로 대치하고 바이트 위치는 그대로 유지
        - 8개의 서로 다른 S-박스를 쓰는 DES와 다른 구조
      - 입력과 출력 사이에 선형 관계가 없는 대치로 혼돈 효과 제공
        - e.g., 한 비트만 다른 5A와 5B는 4비트가 다른 BE와 39로 대치
      - 복호화는 InvSubBytes (Inverse SubBytes) 표로 같은 방법으로 대치하여 원래 바이트를 복원하는 InvSubBytes 사용
        - InvSubBytes 표는 부록#4 참고

변환

- 대치(Substitution) (2/7)
  - 부분 바이트(SubBytes) (2/2)
    - 과정
      1. 바이트의 왼쪽 4비트를 행 번호, 오른쪽 4비트를 열 번호로 하여 SubBytes 표에서 교차하는 칸 선택
         - e.g., 바이트 ab_{16}은 a행 b열 칸의 값으로 대치
      2. 선택한 칸의 바이트로 대치하고 상태의 16바이트마다 반복



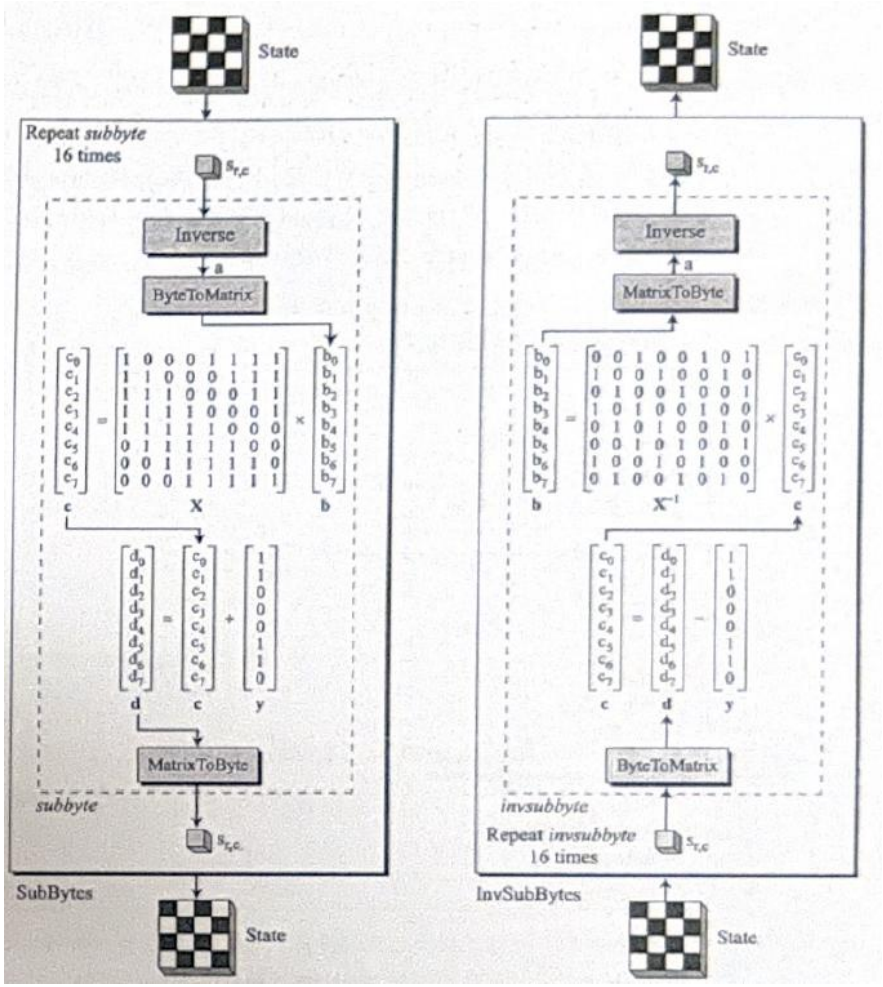
변환

- 대치(Substitution) (3/7)
  - $GF(2^8)$를 이용한 변환 (1/2)
    - 정의
      - SubBytes 표의 값을 표 없이 *기약 다항식 $x^8 + x^4 + x^3 + x + 1$을 가진 체 $GF(2^8)$의 연산으로 계산하는 방법
    - 특징
      - SubBytes와 InvSubBytes는 바이트 하나를 처리하는 subbyte, invsubbyte 루틴을 상태의 16바이트에 한 번씩 반복
      - 곱셈 역원을 구한 뒤 상수 행렬 X를 곱하고 상수 y를 더하는 계
      - $GF(2)$에서 덧셈과 뺄셈이 모두 XOR이므로 invsubbyte가 subbyte를 되돌리는 역변환

*기약 다항식(Irreducible Polynomial)
차수가 1 이상인 두 다항식의 곱으로 인수분해되지 않는 다항식

변환

- 대치(Substitution) (4/7)
  - $GF(2^8)$를 이용한 변환 (2/2)
    - 과정
      - Subbyte
        1. 바이트 $s_{r,c}$의 곱셈 역원 계산 (단, 00의 역원은 00)
        2. 역원을 최하위 비트가 맨 위인 8×1 열 행렬 b로 변환
        3. $d = Xb \oplus y$ 계산 (단, y는 0x63)
        4. 열 행렬 d를 바이트로 바꾸어 출력
      - Invsubbyte
        1. 바이트를 열 행렬 d로 바꾸어 $b = X^{-1}(d \oplus y)$ 계산
        2. b를 바이트로 바꾸고 곱셈 역원을 구하여 출력
        - $d \oplus y = Xb \oplus y = Xb$, $X^{-1}Xb = b$이므로 b의 역원인 $s_{r,c}$ 복원

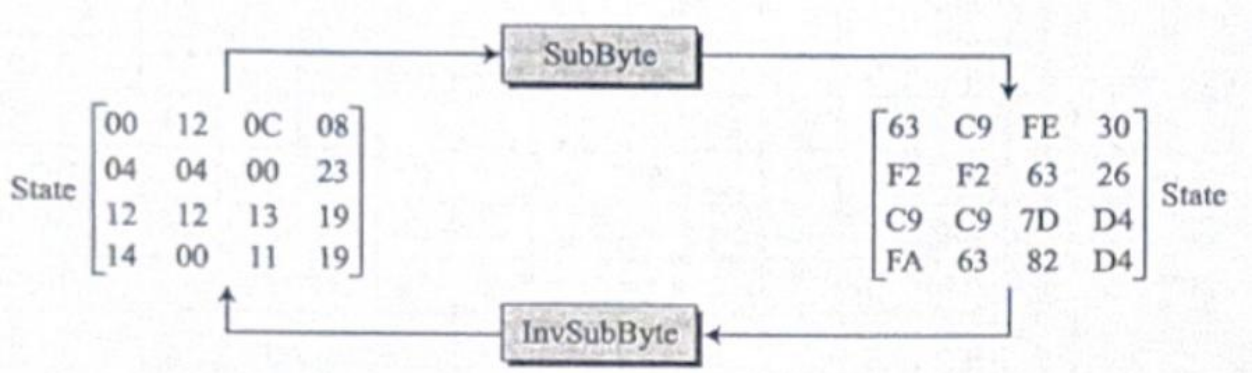


변환

- 대치(Substitution) (5/7)
  - 예제 7.2 (1/2)

상태 S에 SubBytes를 적용한 결과 상태를 구해보아라. 또한 값이 같은 두 바이트 04의 대치 결과를 비교하고, InvSubBytes로 원래 상태가 복원되는지 확인해보아라. 단, S의 0~3행은 00 12 0C 08, 04 04 00 17, 12 12 13 19, 14 00 11 19이다.

- 풀이
  1. 바이트마다 왼쪽 16진수를 행, 오른쪽 16진수를 열로 하여 SubBytes 표에서 대치
     - 00은 0행 0열의 63, 12는 1행 2열의 C9, 17은 1행 7열의 F0
  2. 16바이트를 모두 대치한 결과 상태의 0~3행은 63 C9 FE 30, F2 F2 63 F0, C9 C9 7D D4, FA 63 82 D4



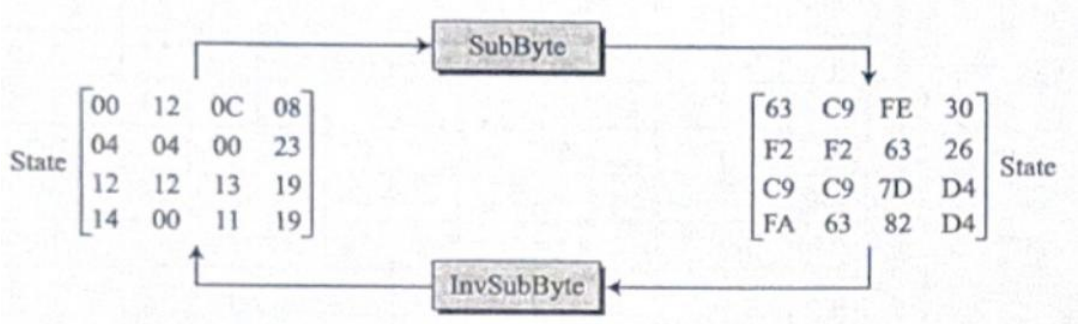
변환

- 대치(Substitution) (6/7)
  - 예제 7.2 (2/2)

상태 S에 SubBytes를 적용한 결과 상태를 구해보아라. 또한 값이 같은 두 바이트 04의 대치 결과를 비교하고, InvSubBytes로 원래 상태가 복원되는지 확인해보아라. 단, S의 0~3행은 00 12 0C 08, 04 04 00 17, 12 12 13 19, 14 00 11 19이다.

- 풀이
  3. $s_{1,0}$과 $s_{1,1}$의 04는 같은 표로 대치되므로 둘 다 F2
  4. 결과 상태를 InvSubBytes 표로 대치하면 63은 00, F2는 04, F0은 17로 원래 상태 복원
     - 부록#2 참고

∴ 결과 상태의 0~3행은 63 C9 FE 30, F2 F2 63 F0, C9 C9 7D D4, FA 63 82 D4,
두 바이트 04는 모두 F2로 대치, InvSubBytes로 원래 상태 복원



변환

- 대치(Substitution) (7/7)
  - 예제 7.3

16진수 바이트 0C를 subbyte 루틴으로 FE로 대치하는 과정과, invsubbyte 루틴으로 FE를 0C로 복원하는 과정을 보이시오.

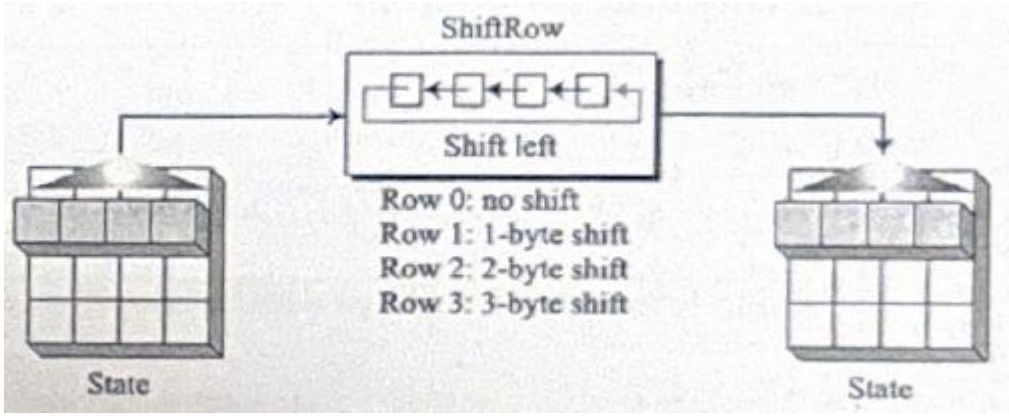
- 풀이
  - subbyte
  1. $GF(2^8)$에서 0C의 곱셈 역원은 B0이며 최상위 비트부터 나타내면 $\boldsymbol{b} = 10110000$
     - 역원 검산 - $GF(2^8)$에서 0C와 B0의 곱은 01
  2. 행렬 X를 곱하면 $\boldsymbol{c} = 10011101$
  3. $y = 01100011$(0x63)과 XOR하면 $\boldsymbol{d} = 11111110$이므로 FE

  - invsubbyte
  1. invsubbyte는 FE에 y를 XOR하여 $\boldsymbol{c} = 10011101$ 계산
  2. X^{-1}을 곱하면 10110000이므로 B0
  3. $GF(2^8)$에서 B0의 곱셈 역원은 0C

  ∴ subbyte는 0C를 역원 B0, 행렬 곱 9D, XOR 결과 FE 순서로 대치하고, invsubbyte는 FE를 9D, B0, 0C 순서로 복원

변환

- 치환(Permutation) 또는 이동 연산(Shifting) (1/3)
  - ShiftRows
    - 정의
      - 상태의 각 행을 행 번호만큼 왼쪽으로 순환 이동하는 이동 변환
    - 특징
      - 0행은 이동하지 않고 1행은 1바이트, 2행은 2바이트, 3행은 3바이트 이동
      - 바이트 단위로 이동하므로 바이트 값은 그대로 두고 위치만 변경
      - 복호화는 같은 바이트 수만큼 오른쪽으로 순환 이동하여 원래 위치로 복원하는 InvShiftRows (Inverse ShiftRows) 사용

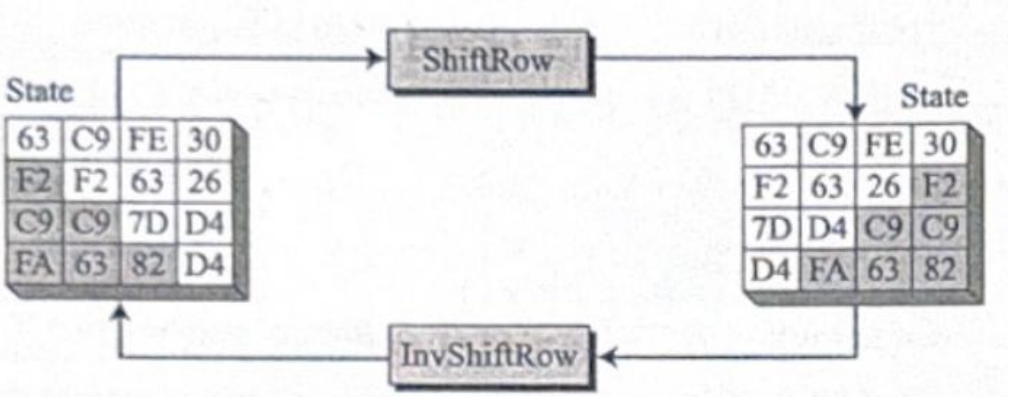


변환

- 치환(Permutation) 또는 이동 연산(Shifting) (2/3)
  - 예제 7.4 (1/2)

상태 S에 ShiftRows를 적용한 결과 상태를 구하고, InvShiftRows로 원래 상태가 복원되는지 확인하시오. 단, S의 0~3행은 63 C9 FE 30, F2 F2 63 F0, C9 C9 7D D4, FA 63 82 D4이다.

- 풀이
  1. 행 번호만큼 왼쪽으로 순환 이동
     - 0행 63 C9 FE 30은 이동 없이 그대로
     - 1행 F2 F2 63 F0은 1바이트 이동하여 F2 63 F0 F2
     - 2행 C9 C9 7D D4는 2바이트 이동하여 7D D4 C9 C9
     - 3행 FA 63 82 D4는 3바이트 이동하여 D4 FA 63 82



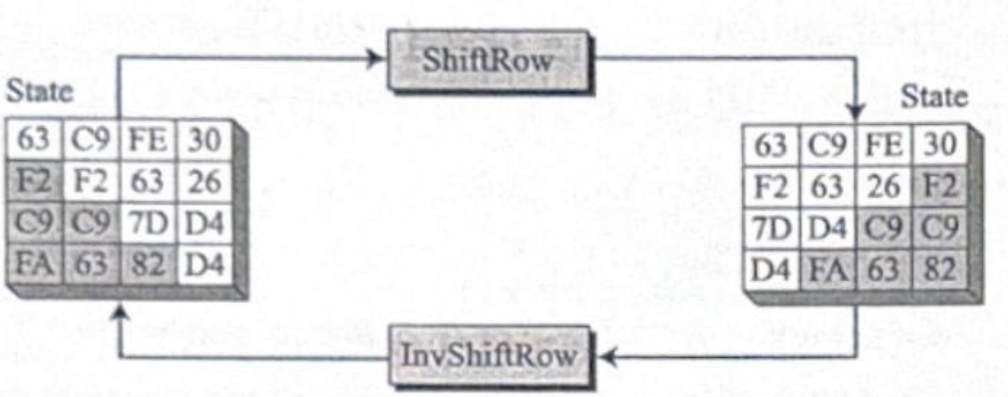
변환

- 치환(Permutation) 또는 이동 연산(Shifting) (3/3)
  - 예제 7.4 (2/2)

상태 S에 ShiftRows를 적용한 결과 상태를 구하고, InvShiftRows로 원래 상태가 복원되는지 확인하시오. 단, S의 0~3행은 63 C9 FE 30, F2 F2 63 F0, C9 C9 7D D4, FA 63 82 D4이다.

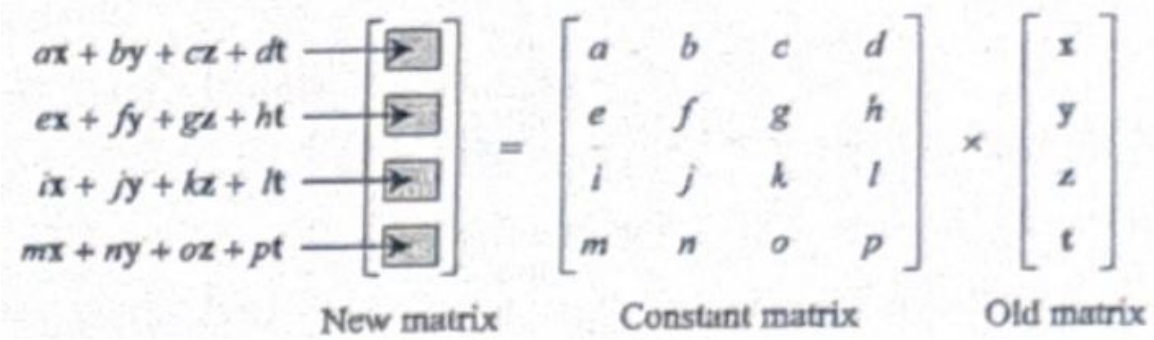
- 풀이
  2. InvShiftRows로 각 행을 같은 바이트 수만큼 오른쪽으로 이동하면 원래 상태 복원
     - e.g., 1행 F2 63 F0 F2를 오른쪽으로 1바이트 이동하면 F2 F2 63 F0

∴ 결과 상태의 0~3행은 63 C9 FE 30, F2 63 F0 F2, 7D D4 C9 C9, D4 FA 63 82, InvShiftRows로 원래 상태 복원



변환

- 뒤섞음(Mixing) (1/3)
  - MixColumns (1/2)
    - 정의
      - 상태의 각 열에 상수 정방 행렬 C를 곱하여 새로운 열로 바꾸는 변환
    - 특징 (1/2)
      - 새 열의 각 바이트가 이전 열의 네 바이트 모두에 의존하여 비트 단위의 확산 효과 제공
        - SubBytes와 ShiftRows만으로는 한 바이트의 비트가 다른 바이트에 영향을 주지 않는 구조
      - 네 입력 바이트가 같아도 출력 바이트가 서로 다르도록 각 바이트에 서로 다른 상수를 곱하여 결합

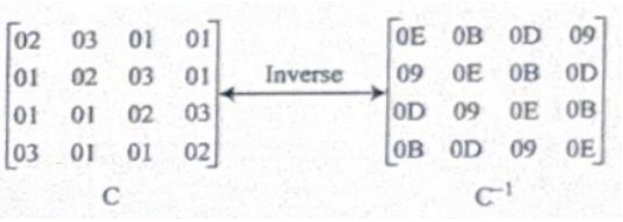


변환

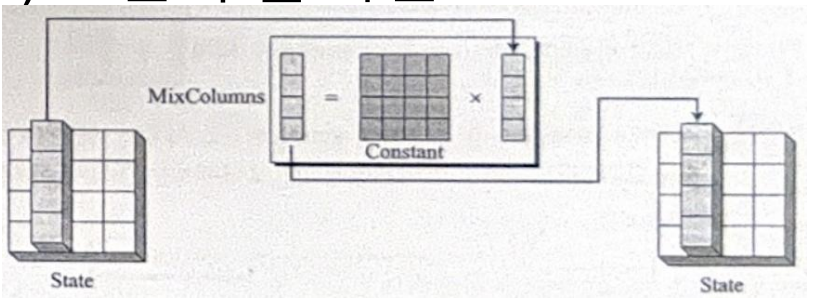
- 뒤섞음(Mixing) (2/3)

*법(Modulus)
연산 결과를 나누어 나머지만 결과로 남길 때 나누는 값

  - MixColumns (2/2)
    - 특징 (2/2)
      - 새 열의 r번째 바이트 $s'_{r,c}$는 C의 r행과 이전 열의 곱
        - e.g., C의 0행이 02, 03, 01, 01이므로 $s'_{0,c} = (02 \bullet s_{0,c}) \oplus (03 \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$ (단, •는 $GF(2^8)$ 곱셈)



      - 바이트 곱셈 •는 $GF(2^8)$에서 $x^8 + x^4 + x^3 + x + 1$을 *법으로, 덧셈 ⊕는 XOR 덧셈으로 계산
      - 복호화는 C의 역행렬 C^{-1}을 곱하는 InvMixColumns (Inverse MixColumns)로 원래 열 복원



변환

- 뒤섞음(Mixing) (3/3)
  - 예제 7.5

상태 S에 MixColumns를 적용한 결과 상태를 구하고, InvMixColumns로 원래 상태가 복원되는지 확인하시오. 단, S의 0~3행은 63 C9 FE 30, F2 63 F0 F2, 7D D4 C9 C9, D4 FA 63 82이다.

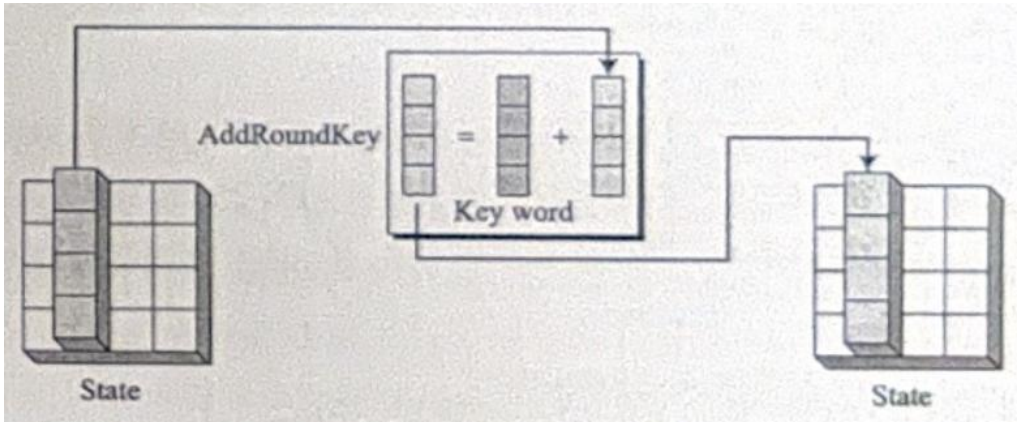
- 풀이
  1. 0열 63, F2, 7D, D4에 C의 각 행을 곱하여 새 0열 계산
     - 0행은 $(02 \bullet 63) \oplus (03 \bullet F2) \oplus 7D \oplus D4 = C6 \oplus 0D \oplus 7D \oplus D4 = 62$
     - 1~3행도 같은 방법으로 CF, 0C, 99
  2. 1~3열도 같은 방법으로 계산
  3. 결과 상태에 InvMixColumns를 적용하면 원래 상태 복원

- $$\begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & 63 & F0 & F2 \\ 7D & D4 & C9 & C9 \\ D4 & FA & 63 & 82 \end{bmatrix} \leftrightarrow \begin{bmatrix} 62 & 02 & 46 & 26 \\ CF & 92 & 26 & 0D \\ 0C & 0C & 22 & D6 \\ 99 & 18 & E6 & 74 \end{bmatrix}$$

∴ 결과 상태의 0~3행은 62 02 46 26, CF 92 26 0D, 0C 0C 22 D6, 99 18 E6 74, InvMixColumns로 원래 상태 복원

변환

- 키 덧셈(Key Adding)
  - AddRoundKey
    - 정의
      - 상태의 각 열에 라운드 키의 워드 하나를 XOR하는 변환
    - 특징
      - 상태의 c번째 열에 라운드 키 워드 w_{4R+c}를 XOR 연산
        - R은 라운드 번호
      - 라운드 키 없이는 공개된 세 역변환만으로 평문이 복원되므로 라운드마다 키를 더하여 평문 복원을 방어
      - $GF(2^8)$에서 덧셈과 뺄셈이 같은 XOR이므로 AddRoundKey 자신이 역변환



목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

키 확장

- 키 확장(Key Expansion) (1/12)
  - 정의
    - 암호 키 하나로 $N_r + 1$개의 128비트 라운드 키를 만드는 과정
  - 특징
    - 라운드 키를 32비트 워드 단위로 만들어 $4(N_r + 1)$개의 워드 $w_0, w_1, \cdots$ 생성
      - e.g., AES-128은 44개, AES-192는 52개, AES-256은 60개 워드
    - R번째 라운드는 연속한 워드 4개 w_{4R} ~ w_{4R+3}을 라운드 키로 사용

라운드	첫 번째 워드	두 번째 워드	세 번째 워드	네 번째 워드
라운드 전	w_0	w_1	w_2	w_3
1	w_4	w_5	w_6	w_7
2	w_8	w_9	w_{10}	w_{11}
…	…	…	…	…
N_r	w_{4N_r}	w_{4N_r+1}	w_{4N_r+2}	w_{4N_r+3}

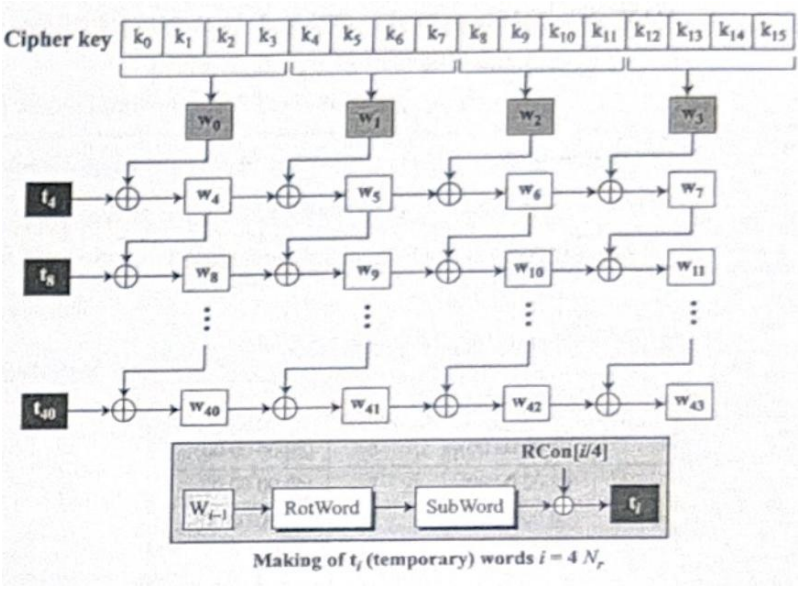
키 확장

- 키 확장(Key Expansion) (2/12)
  - AES-128의 키 확장 (1/5)
    - 구조
      - RotWord
        - 워드의 4바이트를 왼쪽으로 1바이트 순환 이동하는 루틴
      - SubWord
        - 워드의 4바이트를 SubBytes 표로 각각 대치하는 루틴
      - 라운드 상수(RCon , Round Constant)
        - 맨 왼쪽 바이트가 x^{R-1}을 $x^8 + x^4 + x^3 + x + 1$로 나눈 나머지이고 나머지 3바이트가 0인 워드
        - e.g., 9라운드는 x^8 mod $(x^8 + x^4 + x^3 + x + 1) = x^4 + x^3 + x + 1$이므로 1B

라운드 R	x^{R-1} 나머지	RCon	라운드 R	x^{R-1} 나머지	RCon
1	1	01 00 00 00	6	x^5	20 00 00 00
2	x	02 00 00 00	7	x^6	40 00 00 00
3	x^2	04 00 00 00	8	x^7	80 00 00 00
4	x^3	08 00 00 00	9	$x^4 + x^3 + x + 1$	1B 00 00 00
5	x^4	10 00 00 00	10	$x^5 + x^4 + x^2 + x$	36 00 00 00

키 확장

- 키 확장(Key Expansion) (3/12)
  - AES-128의 키 확장 (2/5)
    - 과정
      1. 암호 키 16바이트를 4바이트씩 나누어 w_0 ~ w_3 생성
      2. i mod 4가 0이 아니면 $w_i = w_{i-1} \oplus w_{i-4}$ 계산 (단, $4 \leq i \leq 43$)
      3. i mod 4가 0이면 $w_i = t \oplus w_{i-4}$ 계산
         - 임시 워드 t는 w_{i-1}에 RotWord와 SubWord를 적용한 후 RCon[$i/4$]를 XOR 연산한 워드
    - 따라서 라운드 키 하나가 바로 앞 라운드 키의 워드 4개로 만들어지는 구조
    - 키 확장 코드는 부록#6 참고



키 확장

- 키 확장(Key Expansion) (4/12)
  - AES-128의 키 확장 (3/5)
    - 예제 7.6 (1/3)

AES-128 키 확장으로 128비트 암호 키 K의 라운드 키를 만들 때, 라운드마다 임시 워드 t와 라운드 키를 구하시오.
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  1. 암호 키를 4바이트씩 나누면 w_0 ~ w_3은 2475A2B3, 34755688, 31E21200, 13AA5487
  2. 1라운드의 t는 w_3에 RotWord와 SubWord를 적용한 후 RCon[1]과 XOR
     - 13AA5487에 RotWord를 적용하면 AA548713, SubWord를 적용하면 AC20177D
     - t=AC20177D⊕01000000=AD20177D
  3. $w_4 = t \oplus w_0$=8955B5CE, $w_5 = w_4 \oplus w_1$=BD20E346, $w_6 = w_5 \oplus w_2$=8CC2F146, $w_7 = w_6 \oplus w_3$=9F68A5C1

라운드	t	첫 번째 워드	두 번째 워드	세 번째 워드	네 번째 워드
라운드 전	-	2475A2B3	34755688	31E21200	13AA5487
1	AD20177D	8955B5CE	BD20E346	8CC2F146	9F68A5C1

키 확장

- 키 확장(Key Expansion) (5/12)
  - AES-128의 키 확장 (4/5)
    - 예제 7.6 (2/3)

AES-128 키 확장으로 128비트 암호 키 K의 라운드 키를 만들 때, 라운드마다 임시 워드 t와 라운드 키를 구하시오.
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  4. 2~5라운드도 같은 방법으로 계산한 결과는 아래 표

라운드	t	첫 번째 워드	두 번째 워드	세 번째 워드	네 번째 워드
라운드 전	-	2475A2B3	34755688	31E21200	13AA5487
1	AD20177D	8955B5CE	BD20E346	8CC2F146	9F68A5C1
2	470678DB	CE53CD15	73732E53	FFB1DF15	60D97AD4
3	31DA48D0	FF8985C5	8CFAAB96	734B7483	13920E57
4	47AB5B7D	B822DEB8	34D8752E	479301AD	54010FFA
5	6C762D20	D454F398	E08C86B6	A71F871B	F31E88E1

키 확장

- 키 확장(Key Expansion) (6/12)
  - AES-128의 키 확장 (5/5)
    - 예제 7.6 (3/3)

AES-128 키 확장으로 128비트 암호 키 K의 라운드 키를 만들 때, 라운드마다 임시 워드 t와 라운드 키를 구하시오.
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  5. 6~10라운드도 같은 방법으로 계산한 결과는 아래 표

라운드	t	첫 번째 워드	두 번째 워드	세 번째 워드	네 번째 워드
6	52C4F80D	86900B95	661C8D23	C1030A38	321D82D9
7	E4133523	62833EB6	049FB395	C59CB9AD	F7813B74
8	8CE29268	EE61ACDE	EAFE1F4B	2F62A6E6	D8E39D92
9	0A5E4F61	E43FE3BF	0EC1FCF4	21A35A12	F940C780
10	3FC6CD99	DBF92E26	D538D2D2	F49B88C0	0DDB4F40

∴ 라운드 전부터 10라운드까지의 t와 라운드 키는 두 표의 값

키 확장

- 키 확장(Key Expansion) (7/12)
  - AES-192와 AES-256의 키 확장
    - 개요
      - AES-128과 같은 방법이지만 AES-192는 6개, AES-256은 8개 워드 단위로 계산
      - 처음 6개 혹은 8개 워드를 암호 키로 채우고 단위 워드 수의 배수 위치마다 임시 워드 t 사용
    - 과정
      - AES-192
        - i mod 6이 0이 아니면 $w_i = w_{i-1} \oplus w_{i-6}$, 0이면 $w_i = t \oplus w_{i-6}$ 계산
      - AES-256
        - i mod 8이 0이면 $w_i = t \oplus w_{i-8}$ 계산
        - i mod 8이 4이면 SubWord를 적용한 w_{i-1}과 w_{i-8}을 XOR하여 w_i 계산
        - 나머지 i는 $w_i = w_{i-1} \oplus w_{i-8}$ 계산

키 확장

- 키 확장(Key Expansion) (8/12)
  - 키 확장 과정 분석 (1/5)
    - 특징
      - SubWord의 S-박스 값도 $GF(2^8)$ 연산으로 계산하므로 표를 저장하지 않고 $GF(2^8)$와 $GF(2)$ 연산만으로 구현
      - 라운드 상수 RCon이 다른 변환에서 생길 수 있는 라운드 키 사이의 대칭성 제거
    - 안전성
      - 암호 키나 라운드 키의 일부를 얻어도 SubWord의 비선형성 때문에 나머지 라운드 키를 알려면 남은 암호 키 복구 필요
      - 한 비트만 다른 두 암호 키로도 적은 수의 라운드 만에 서로 다른 라운드 키 생성
      - DES와 달리 같은 라운드 키가 반복되는 심각한 취약 키가 없는 구조

키 확장

- 키 확장(Key Expansion) (9/12)
  - 키 확장 과정 분석 (2/5)
    - 예제 7.7

AES-128 키 확장에서 다음 라운드 키는 이전 라운드 키의 워드로 만들어진다. 이전 라운드 키와 다음 라운드 키가 같은 규칙으로 반복되지 않는 이유를 설명하시오.

- 풀이
  1. 각 라운드의 첫 번째 워드는 t를 거쳐 만들고, 나머지 세 워드는 바로 앞 워드와 이전 라운드 워드를 XOR하여 생성
  2. t의 SubWord는 곱셈 역원을 쓰는 비선형 대치이므로 다음 라운드 키를 이전 라운드 키 워드의 XOR 조합으로 표현할 수 없는 구조
  3. t에 라운드마다 다른 RCon을 XOR하므로 이전 라운드와 같은 계산이 반복되지 않는 구조

  ∴ SubWord의 비선형성과 라운드마다 다른 RCon 덧셈 때문에 라운드 키가 서로 다른 값으로 생성

키 확장

- 키 확장(Key Expansion) (10/12)
  - 키 확장 과정 분석 (3/5)
    - 예제 7.8 (1/2)

한 비트만 다른 두 암호 키로 각각 라운드 키를 만들고, 라운드마다 두 라운드 키의 비트 차이를 구하시오.
K_1 = 12 45 A2 A1 23 31 A4 A3 B2 CC AA 34 C2 BB 77 23
K_2 = 12 45 A2 A1 23 31 A4 A3 B2 CC AB 34 C2 BB 77 23

- 풀이
  1. 두 암호 키는 11번째 바이트 AA와 AB의 최하위 비트만 다른 값
  2. 라운드 전 키는 암호 키와 같으므로 비트 차이 1, 1라운드 키는 w_6과 w_7이 1비트씩 달라 비트 차이 2

라운드	K_1의 라운드 키	K_2의 라운드 키	비트 차이
라운드 전	1245A2A1 2331A4A3 B2CCAA34 C2BB7723	1245A2A1 2331A4A3 B2CCAB34 C2BB7723	1
1	F9B08484 DA812027 684D8A13 AAF6FD30	F9B08484 DA812027 684D8B13 AAF6FC30	2
2	B9E48028 6365A00F 0B282A1C A1DED72C	B9008028 6381A00F 0BCC2B1C A13AD72C	17
3	A0EAF11A C38F5115 C8A77B09 6979AC25	3D0EF11A 5E8F5115 55437A09 F479AD25	30
4	1E7BCEE3 DDF49FF6 1553E4FF 7C2A48DA	839BCEA5 DD149FB0 8857E5B9 7C2E489C	31
5	EB2999F3 36DD0605 238EE2FA 5FA4AA20	A2C910B5 7FDD8F05 F78A6ABC 8BA42220	34

키 확장

- 키 확장(Key Expansion) (11/12)
  - 키 확장 과정 분석 (4/5)
    - 예제 7.8 (2/2)

한 비트만 다른 두 암호 키로 각각 라운드 키를 만들고, 라운드마다 두 라운드 키의 비트 차이를 구하시오.
K_1 = 12 45 A2 A1 23 31 A4 A3 B2 CC AA 34 C2 BB 77 23
K_2 = 12 45 A2 A1 23 31 A4 A3 B2 CC AB 34 C2 BB 77 23

- 풀이
  3. 2라운드부터 t의 SubWord를 거친 차이가 모든 워드로 퍼져 비트 차이 증가
  ∴ 비트 차이는 라운드 전부터 10라운드까지 차례로 1, 2, 17, 30, 31, 34, 56, 50, 44, 51, 52

라운드	K_1의 라운드 키	K_2의 라운드 키	비트 차이
6	82852E3C B4582839 97D6CAC3 C87260E3	CB5AA788 B487288D 430D4231 C8A96011	56
7	82553FD4 360D17ED A1DBDD2E 69A9BDCD	588A2560 EC0D0DED AF004FDC 67A92FCD	50
8	D12F822D E72295C0 46F948EE 2F50F523	0B9F98E5 E7929508 4892DAD4 2F3BF519	44
9	99C9A438 7EEB31F8 38127916 17428C35	F2794CF0 15EBD9F8 5D79032C 7242F635	51
10	83AD32C8 FD460330 C5547A26 D216F613	E83BDAB0 FDD00348 A0A90064 D2EBF651	52

키 확장

- 키 확장(Key Expansion) (12/12)
  - 키 확장 과정 분석 (5/5)
    - 예제 7.9

AES-128 키 확장으로 모든 비트가 0인 암호 키의 라운드 키를 만들고, DES의 취약 키처럼 같은 라운드 키가 반복되는지 확인하시오.

- 풀이
  1. 라운드 전 키는 00000000 4개이고, 1라운드 t는 00000000에 RotWord와 SubWord를 적용한 63636363과 RCon[1]의 XOR인 62636363
  2. 이전 라운드 워드가 모두 0이므로 1라운드 키는 62636363 4개
  3. 2라운드 키는 첫 번째와 세 번째, 두 번째와 네 번째 워드가 같은 값
  4. 3라운드부터 네 워드가 모두 다른 값

라운드	첫 번째 워드	두 번째 워드	세 번째 워드	네 번째 워드
라운드 전	00000000	00000000	00000000	00000000
1	62636363	62636363	62636363	62636363
2	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
3	90973450	696CCFFA	F2F45733	0B0FAC99
…	…	…	…	…
10	B4EF5BCB	3E92E211	23E951CF	6F8F188E

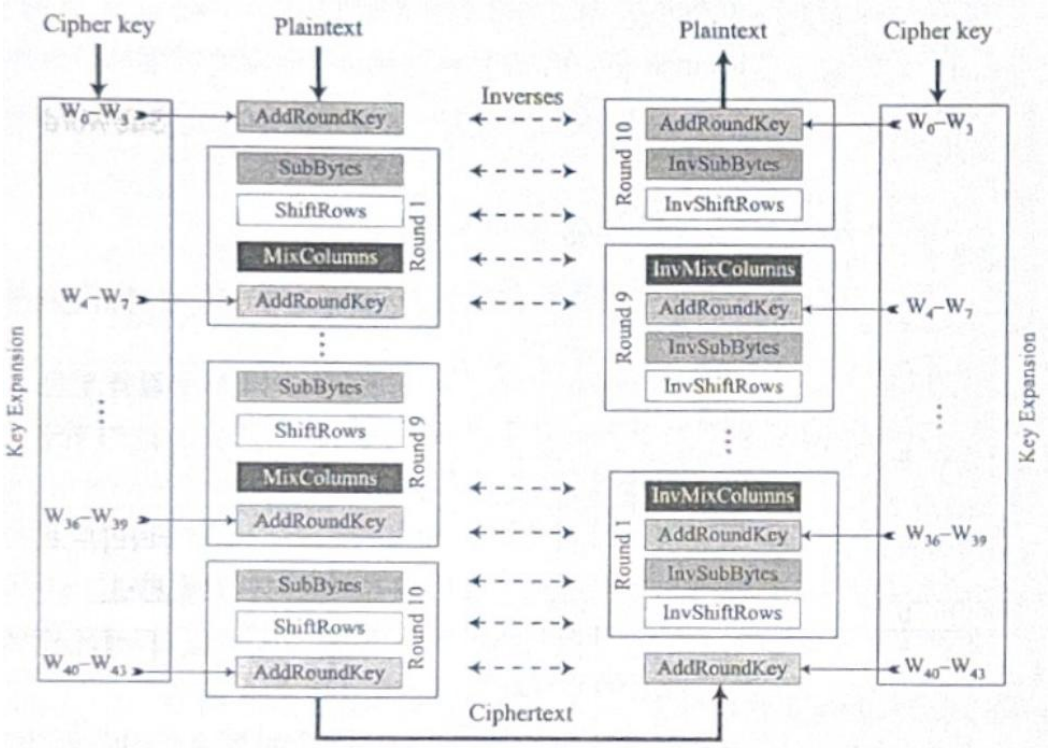
∴ 같은 라운드 키가 반복되지 않으므로 DES의 취약 키 성질이 없는 키

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

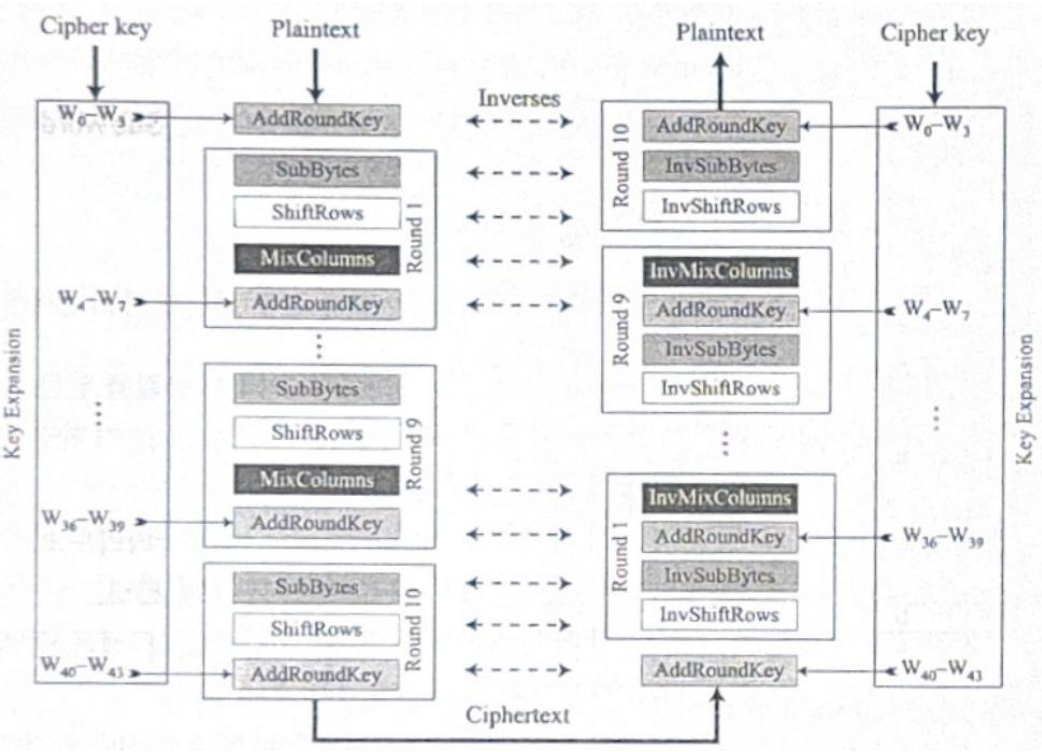
암호와 역암호

- 기본 설계 (1/2)
  - 구조 (1/2)
    - 암호와 역암호(Inverse Cipher)는 각각 라운드 전 AddRoundKey와 10개 라운드로 구성
    - 역암호는 각 변환을 역변환으로 바꾸고 라운드 키를 역순으로 사용
    - AES-128 기준이며 AES-192와 AES-256도 같은 설계
    - 암호의 1~9라운드는 SubBytes, ShiftRows, MixColumns, AddRoundKey 순서



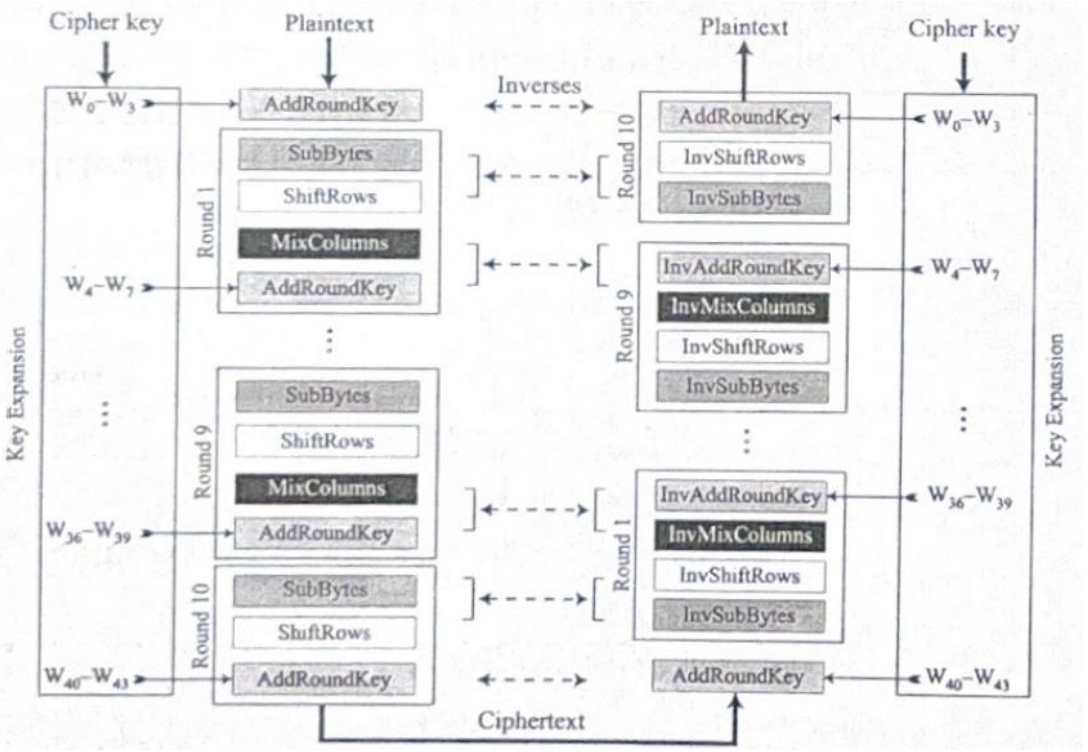
암호와 역암호

- 기본 설계 (2/2)
  - 구조 (2/2)
    - 역암호의 1~9라운드는 InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns 순서
  - 특징
    - 암호의 각 변환이 역암호에서 같은 위치의 역변환과 서로상쇄되도록 변환 순서 변경
    - 라운드 안 변환 순서가 달라 암호와 역암호를 같은 구조로구현할 수 없는 설계



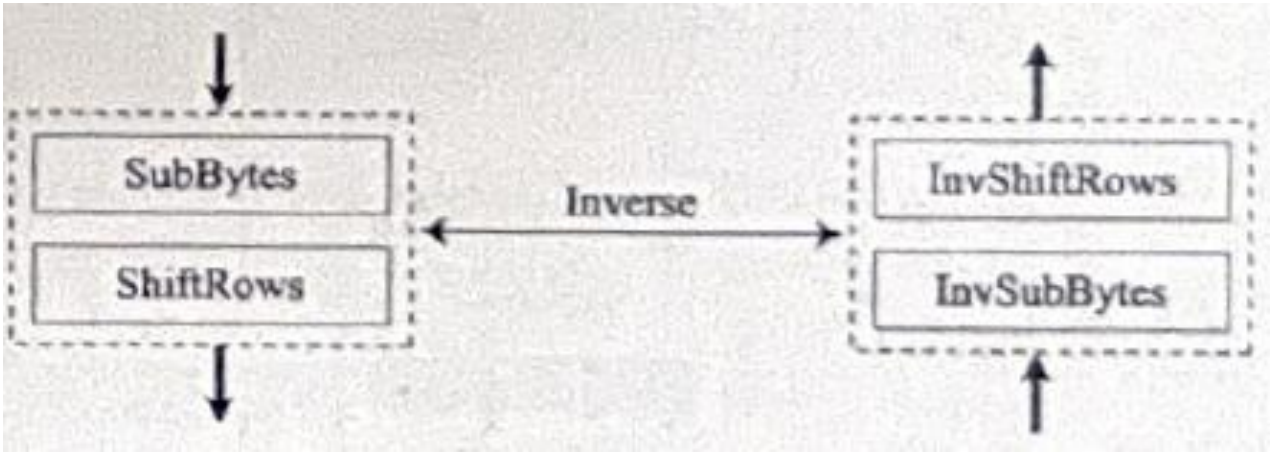
암호와 역암호

- 대체 설계 (1/3)
  - 구조
    - 역암호의 1~9라운드도 암호와 같은 순서로 역변환 적용
      - InvSubBytes, InvShiftRows, InvMixColumns, InvAddRoundKey 순서
    - 10라운드는 InvMixColumns를 빼고 AddRoundKey 적용
  - 특징
    - 암호와 역암호를 같은 구조로 구현해야 하는 환경에서 사용
    - 변환 하나가 아니라 이웃한 두 변환의 쌍 단위로 가역성 (Invertibility) 성립



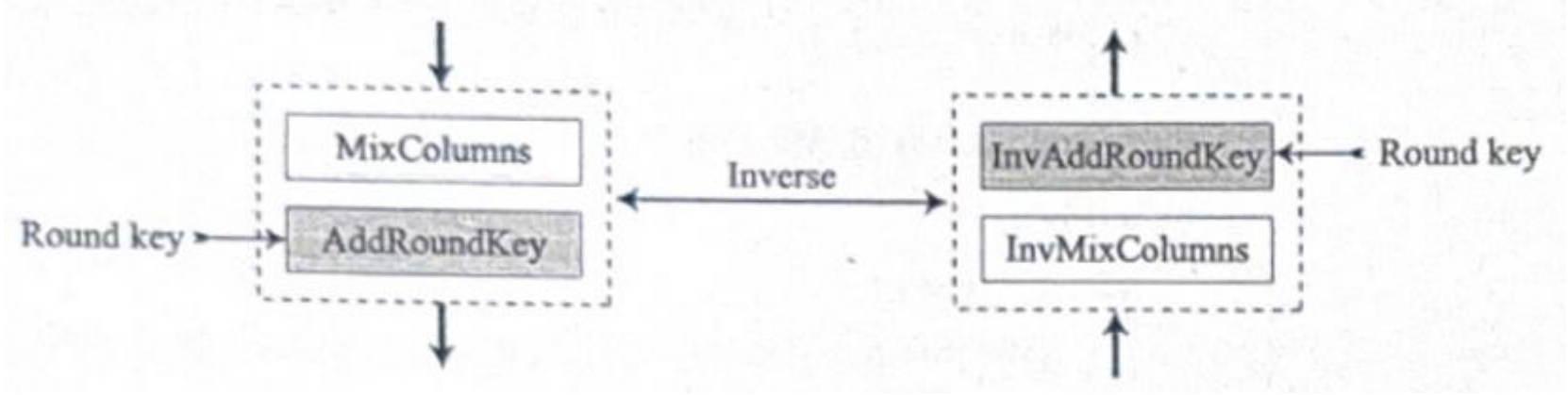
암호와 역암호

- 대체 설계 (2/3)
  - SubBytes와 ShiftRows 쌍
    - SubBytes는 값만, ShiftRows는 위치만 바꾸므로 두 변환은 순서를 바꾸어도 같은 결과
    - 역암호에서 InvSubBytes, InvShiftRows 순서로 적용해도 암호의 두 변환 조합과 역함수 관계



암호와 역암호

- 대체 설계 (3/3)
  - MixColumns와 AddRoundKey 쌍
    - 역암호에서 InvMixColumns, InvAddRoundKey 순서로 적용해도 암호의 두 변환 조합과 역함수 관계
      - InvAddRoundKey는 라운드 키에 C^{-1}을 곱한 키를 더하는 변환
      - 암호의 두 변환은 $T = CS \oplus K$이고 역암호에서 $C^{-1}T \oplus C^{-1}K = S$로 입력 상태 복원 (단, S는 입력 상태, K는 라운드 키)
    - 키 확장에서 1~9라운드 키에 미리 C^{-1}을 곱할 경우, 역암호도 InvAddRoundKey 대신 AddRoundKey 사용



암호와 역암호

- AES-128 암호화 (1/7)
  - 예제 7.10 (1/3)

AES-128로 평문 P를 암호 키 K로 암호화하여 라운드마다 입력 상태, 출력 상태, 라운드 키와 암호문을 구하시오.
P = 00 04 12 14 12 04 12 00 0C 00 13 11 08 23 19 19
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  1. 평문 16바이트를 열 단위로 상태에 넣어 라운드 전 입력 상태 생성
  2. 라운드 전에는 암호 키와 같은 라운드 키로 AddRoundKey만 적용
     - e.g., $s_{0,0}$은 00⊕24=24

라운드	입력 상태	출력 상태	라운드 키
라운드 전	00 12 0C 08 04 04 00 23 12 12 13 19 14 00 11 19	24 26 3D 1B 71 71 E2 89 B0 44 01 4D A7 88 11 9E	24 34 31 13 75 75 E2 AA A2 56 12 54 B3 88 00 87
1	24 26 3D 1B 71 71 E2 89 B0 44 01 4D A7 88 11 9E	6C 44 13 BD B1 9E 46 35 C5 B5 F3 02 5D 87 FC 8C	89 BD 8C 9F 55 20 C2 68 B5 E3 F1 A5 CE 46 46 C1
2	6C 44 13 BD B1 9E 46 35 C5 B5 F3 02 5D 87 FC 8C	1A 90 15 B2 66 09 1D FC 20 55 5A B2 2B CB 8C 3C	CE 73 FF 60 53 73 B1 D9 CD 2E DF 7A 15 53 15 D4
3	1A 90 15 B2 66 09 1D FC 20 55 5A B2 2B CB 8C 3C	F6 7D A2 B0 1B 61 B4 B8 67 09 C9 45 4A 5C 51 09	FF 8C 73 13 89 FA 4B 92 85 AB 74 0E C5 96 83 57

암호와 역암호

- AES-128 암호화 (2/7)
  - 예제 7.10 (2/3)

AES-128로 평문 P를 암호 키 K로 암호화하여 라운드마다 입력 상태, 출력 상태, 라운드 키와 암호문을 구하시오.
P = 00 04 12 14 12 04 12 00 0C 00 13 11 08 23 19 19
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  3. 1~9라운드는 SubBytes, ShiftRows, MixColumns, AddRoundKey를 차례로 적용
     - 각 라운드의 출력 상태가 다음 라운드의 입력 상태

라운드	입력 상태	출력 상태	라운드 키
4	F6 7D A2 B0 1B 61 B4 B8 67 09 C9 45 4A 5C 51 09	CA E5 48 BB D8 42 AF 71 D1 BA 98 2D 4E 60 9E DF	B8 34 47 54 22 D8 93 01 DE 75 01 0F B8 2E AD FA
5	CA E5 48 BB D8 42 AF 71 D1 BA 98 2D 4E 60 9E DF	90 35 13 60 2C FB 82 3A 9E FC 61 ED 49 39 CB 47	D4 E0 A7 F3 54 8C 1F 1E F3 86 87 88 98 B6 1B E1
6	90 35 13 60 2C FB 82 3A 9E FC 61 ED 49 39 CB 47	18 0A B9 B5 64 68 6A FB 5A EF D7 79 8E B2 10 4D	86 66 C1 32 90 1C 03 1D 0B 8D 0A 82 95 23 38 D9
7	18 0A B9 B5 64 68 6A FB 5A EF D7 79 8E B2 10 4D	01 63 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03	62 04 C5 F7 83 9F 9C 81 3E B3 B9 3B B6 95 AD 74

암호와 역암호

- AES-128 암호화 (3/7)
  - 예제 7.10 (3/3)

AES-128로 평문 P를 암호 키 K로 암호화하여 라운드마다 입력 상태, 출력 상태, 라운드 키와 암호문을 구하시오.
P = 00 04 12 14 12 04 12 00 0C 00 13 11 08 23 19 19
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  4. 10라운드는 MixColumns 없이 SubBytes, ShiftRows, AddRoundKey만 적용
  5. 10라운드 출력 상태를 열 단위로 꺼내 암호문 생성

  ∴ 라운드마다 상태와 라운드 키는 세 표, 암호문은 BC 02 8B D3 E0 E3 B1 95 55 0D 6D F8 E6 F1 82 41

라운드	입력 상태	출력 상태	라운드 키
8	01 63 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	EE EA 2F D8 61 FE 62 E3 AC 1F A6 9D DE 4B E6 92
9	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	0A D9 F1 3C 95 63 9F 35 2A 80 29 00 16 76 09 77	E4 0E 21 F9 3F C1 A3 40 E3 FC 5A C7 BF F4 12 80
10	0A D9 F1 3C 95 63 9F 35 2A 80 29 00 16 76 09 77	BC E0 55 E6 02 E3 0D F1 8B B1 6D 82 D3 95 F8 41	DB D5 F4 0D F9 38 9B DB 2E D2 88 4F 26 D2 C0 40

암호와 역암호

- AES-128 암호화 (4/7)
  - 예제 7.11

예제 7.10의 7라운드 입력 상태 S에 SubBytes, ShiftRows, MixColumns, AddRoundKey를 차례로 적용한 후의 상태를 각각 구하시오.
단, S의 0~3행은 18 0A B9 B5, 64 68 6A FB, 5A EF D7 79, 8E B2 10 4D이다.

- 풀이
  1. SubBytes로 바이트마다 대치하고 ShiftRows로 1~3행을 왼쪽으로 순환 이동
     - e.g., SubBytes에서 18은 AD, 0A는 67
  2. MixColumns로 열마다 C를 곱하여 새 열 계산
  3. 예제 7.10의 키 확장으로 만든 7라운드 키와 XOR하여 출력 상태 생성

  ∴ 단계별 상태는 아래 표

입력 상태	SubBytes 후	ShiftRows 후	MixColumns 후	7라운드 키	출력 상태
18 0A B9 B5	AD 67 56 D5	AD 67 56 D5	63 67 34 61	62 04 C5 F7	01 63 F1 96
64 68 6A FB	43 45 02 0F	45 02 0F 43	D6 BB A6 E3	83 9F 9C 81	55 24 3A 62
5A EF D7 79	BE DF 0E B6	0E B6 BE DF	CA 39 67 76	3E B3 B9 3B	F4 8A DE 4D
8E B2 10 4D	19 37 CA E3	E3 19 37 CA	7A 2F 25 77	B6 95 AD 74	CC BA 88 03

- AES-128 암호화 (5/7)
  - 예제 7.12

AES-128로 모든 바이트가 00인 평문을 암호 키 K로 암호화한 암호문을 구하시오.
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  1. 라운드 전 AddRoundKey에서 평문이 모두 0이므로 상태는 암호 키와 같은 값
     - 상태의 0~3행은 24 34 31 13, 75 75 E2 AA, A2 56 12 54, B3 88 00 87
  2. 1~10라운드를 예제 7.10과 같은 방법으로 적용

  ∴ 암호문은 63 2C D4 5E 5D 56 ED B5 62 04 01 A0 AA 9C 2D 8D

암호와 역암호

- AES-128 암호화 (6/7)
  - 예제 7.13

AES-128로 마지막 바이트의 한 비트만 다른 두 평문 P_1, P_2를 암호 키 K로 암호화하여 두 암호문의 비트 차이를 구하고 쇄도 효과를 확인하시오.
P_1 = 00 00 ⋯ 00, P_2 = 00 00 ⋯ 01
K = 24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87

- 풀이
  1. P_1의 암호문은 63 2C D4 5E 5D 56 ED B5 62 04 01 A0 AA 9C 2D 8D
  2. P_2의 암호문은 26 F3 9B BC A1 9C 0F B7 C7 2E 7E 30 63 92 73 13
  3. 두 암호문을 XOR하면 45 DF 4F E2 FC CA E2 02 A5 2A 7F 90 C9 0E 5E 9E이며 1인 비트는 128비트 중 67개로 약 절반
     - SubBytes, ShiftRows, MixColumns의 혼돈과 확산이 라운드마다 차이를 퍼뜨린 결과

∴ 평문의 1비트 차이가 암호문 67비트의 차이로 퍼지는 쇄도 효과

암호와 역암호

- AES-128 암호화 (7/7)
  - 예제 7.14

AES-128로 평문 P를 모든 비트가 0인 암호 키로 암호화한 암호문을 구하시오.
P = 00 04 12 14 12 04 12 00 0C 00 13 11 08 23 19 19

- 풀이
  1. 모든 비트가 0인 암호 키를 키 확장하여 라운드 키 생성
     - 1라운드 키는 62636363 4개, 2라운드 키는 9B9898C9, F9FBFBAA, 9B9898C9, F9FBFBAA
  2. 라운드 전 AddRoundKey는 0과 XOR하므로 상태가 평문 그대로
  3. 1~10라운드에서 라운드마다 다른 라운드 키로 변환

  ∴ 암호문은 5A 6F 4B 67 57 B7 A5 D2 C4 30 91 ED 64 9A 42 72

목 차

- DES
  - 개요
  - DES 구조
  - DES 분석
  - 다중 DES
  - DES의 안전성
- AES
  - 개요
  - 변환
  - 키 확장
  - 암호와 역암호
  - AES의 분석

AES의 분석

- 안전성
  - 전수조사 공격
    - AES-128은 2^{128}번의 시험이 필요하여 2^{56}번인 DES보다 2^{72}배 많은 계산이 필요하므로 공격 방어
      - e.g., DES의 키를 t초에 찾으면 AES-128은 $2^{72} \times t \approx 4.7 \times 10^{21} \times t$초 소요
  - 통계적인 공격
    - SubBytes, ShiftRows, MixColumns의 혼돈과 확산이 평문의 빈도 패턴을 없애 공격 방어
  - 차분 분석과 선형 분석
    - DES 이후에 알려진 두 공격에 대한 안전성을 설계 단계부터 고려하여 공격 방어
      - 현재까지 AES를 해독하는 차분 분석과 선형 분석이 알려지지 않은 상황

AES의 분석

- 구현
  - 소프트웨어, 하드웨어, 펌웨어로 구현하는 암호
  - 변환을 테이블 참조(Table Lookup) 혹은 대수 구조를 사용하는 루틴으로 구현 가능
    - 변환은 바이트 기반이나 워드 기반으로 설계 가능
    - 바이트 기반 설계는 8비트 프로세서, 워드 기반 설계는 32비트 프로세서에서 동작
- 단순성과 비용
  - 알고리즘이 단순하여 저렴한 프로세서와 최소량의 메모리로 구현 가능

Thanks!

서 진 우(jinwoo@pel.sejong.ac.kr)

부록#1

- DES S-박스 표 (1/4)
  - 행 번호는 입력 비트 1과 6, 열 번호는 입력 비트 2~5로 정하고 칸 값을 4비트로 출력

S-박스 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-박스 2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

부록#1

- DES S-박스 표 (2/4)
  - S-박스 3과 S-박스 4의 4행 16열 대치 표

S-박스 3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

S-박스 4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

부록#1

- DES S-박스 표 (3/4)
  - S-박스 5와 S-박스 6의 4행 16열 대치 표

S-박스 5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

S-박스 6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13

부록#1

- DES S-박스 표 (4/4)
  - S-박스 7과 S-박스 8의 4행 16열 대치 표

S-박스 7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

S-박스 8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

부록#2

- DES 라운드 키 생성 알고리즘 (1/2)
  - 과정
    1. 패리티 비트 제거 후 두 28비트로 분할
    2. 16라운드 동안 좌측 순환 이동과 축소 P-박스로 라운드 키 생성

```
1.  Key_Generator (keyWithParities[64], RoundKeys[16, 48], ShiftTable[16])
2.  {
3.      permute (64, 56, keyWithParities, cipherKey, ParityDropTable)
4.      split (56, 28, cipherKey, leftKey, rightKey)
5.      for (round = 1 to 16)
6.      {
7.          shiftLeft (leftKey, ShiftTable[round])
8.          shiftLeft (rightKey, ShiftTable[round])
9.          combine (28, 56, leftKey, rightKey, preRoundKey)
10.         permute (56, 48, preRoundKey, RoundKeys[round], KeyCompressionTable)
11.     }
12. }
```

부록#2

- DES 라운드 키 생성 알고리즘 (2/2)
  - 함수
    - shiftLeft는 맨 앞 비트를 맨 뒤로 옮기는 1비트 이동을 라운드별 이동 비트 수만큼 반복

```
1. shiftLeft (block[28], numOfShifts)
2. {
3.     for (i = 1 to numOfShifts)
4.     {
5.         T ← block[1]
6.         for (j = 2 to 28)
7.         {
8.             block[j – 1] ← block[j]
9.         }
10.         block[28] ← T
11.     }
12.}
```

부록#3

- DES 암호 알고리즘과 복호 알고리즘 (1/4)
  - 과정
    1. 초기 치환 후 두 32비트로 분할
    2. 혼합기를 16번, 스와퍼를 15번 적용
    3. 두 32비트를 결합하여 최종 치환

```
1.  Cipher (plainBlock[64], RoundKeys[16, 48], cipherBlock[64])
2.  {
3.      permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
4.      split (64, 32, inBlock, leftBlock, rightBlock)
5.      for (round = 1 to 16)
6.      {
7.          mixer (leftBlock, rightBlock, RoundKeys[round])
8.          if (round != 16) swapper (leftBlock, rightBlock)
9.      }
10.     combine (32, 64, leftBlock, rightBlock, outBlock)
11.     permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
12. }
```

부록#3

- DES 암호 알고리즘과 복호 알고리즘 (2/4)
  - 함수 (1/3)
    - 복호 알고리즘은 암호 알고리즘과 같은 과정에서 라운드 키만 K_{16}부터 역순으로 입력

```
1.  ReverseCipher (cipherBlock[64], RoundKeys[16, 48], plainBlock[64])
2.  {
3.      permute (64, 64, cipherBlock, inBlock, InitialPermutationTable)
4.      split (64, 32, inBlock, leftBlock, rightBlock)
5.      for (round = 1 to 16)
6.      {
7.          mixer (leftBlock, rightBlock, RoundKeys[17 – round])
8.          if (round != 16) swapper (leftBlock, rightBlock)
9.      }
10.     combine (32, 64, leftBlock, rightBlock, outBlock)
11.     permute (64, 64, outBlock, plainBlock, FinalPermutationTable)
12. }
```

부록#3

- DES 암호 알고리즘과 복호 알고리즘 (3/4)
  - 함수 (2/3)
    - function은 확장 P-박스, 화이트너, S-박스, 단순 P-박스 순서로 적용

```
1. function (inBlock[32], RoundKey[48], outBlock[32])
2. {
3.     permute (32, 48, inBlock, T1, ExpansionPermutationTable)
4.     exclusiveOr (48, T1, RoundKey, T2)
5.     substitute (T2, T3, SubstitutionTables)
6.     permute (32, 32, T3, outBlock, StraightPermutationTable)
7. }
8. mixer (leftBlock[32], rightBlock[32], RoundKey[48])
9. {
10.    copy (32, rightBlock, T1)
11.    function (T1, RoundKey, T2)
12.    exclusiveOr (32, leftBlock, T2, T3)
13.    copy (32, T3, leftBlock)
14.}

15.swapper (leftBlock[32], rightBlock[32])
16.{
17.    copy (32, leftBlock, T)
18.    copy (32, rightBlock, leftBlock)
19.    copy (32, T, rightBlock)
20.}
```

부록#3

- DES 암호 알고리즘과 복호 알고리즘 (4/4)
  - 함수 (3/3)
    - S-박스 함수 substitute는 6비트마다 행 번호와 열 번호를 계산하여 표 값을 4비트로 출력

```
1. substitute (inBlock[48], outBlock[32], SubstitutionTables[8, 4, 16])
2. {
3.     for (i = 1 to 8)
4.     {
5.         row ← 2 × inBlock[(i – 1) × 6 + 1] + inBlock[(i – 1) × 6 + 6]
6.         col ← 8 × inBlock[(i – 1) × 6 + 2] + 4 × inBlock[(i – 1) × 6 + 3] + 2 × inBlock[(i –
   1) × 6 + 4] + inBlock[(i – 1) × 6 + 5]
7.         value ← SubstitutionTables[i][row][col]
8.         outBlock[(i – 1) × 4 + 1] ← value / 8; value ← value mod 8
9.         outBlock[(i – 1) × 4 + 2] ← value / 4; value ← value mod 4
10.        outBlock[(i – 1) × 4 + 3] ← value / 2; value ← value mod 2
11.        outBlock[(i – 1) × 4 + 4] ← value
12.    }
13.}
```

부록#4

- AES S-박스 표 (1/2)
  - SubBytes 표

SubBytes	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

부록#4

- AES S-박스 표 (2/2)
  - InvSubBytes 표

InvSubBytes	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

부록#5

- AES 변환 알고리즘 (1/6)
  - 과정
    1. SubBytes는 상태의 16바이트마다 subbyte 루틴 호출
    2. subbyte는 곱셈 역원을 비트 행렬로 바꾸어 행렬 X 곱과 0x63 덧셈을 계산한 후 바이트로 출력
    - 14행은 행렬 X의 i행에서 1인 $i, i+4 \sim i+7$번째 비트만 XOR (단, 비트 번호는 mod 8로 계산)

```
1.  SubBytes (S)
2.  {
3.      for (r = 0 to 3)
4.          for (c = 0 to 3)
5.              S[r][c] ← subbyte (S[r][c])
6.  }
```

```
7.  subbyte (byte)
8.  {
9.      a ← inverse (byte)
10.     ByteToMatrix (a, b)
11.     ByteToMatrix (0x63, y)
12.     for (i = 0 to 7)
13.     {
14.         c[i] ← b[i] ⊕ b[(i+4) mod 8] ⊕
    b[(i+5) mod 8] ⊕ b[(i+6) mod 8] ⊕ b[(i+7)
    mod 8]
15.         d[i] ← c[i] ⊕ y[i]
16.     }
17.     MatrixToByte (d, byte)
18.     return byte
19. }
```

부록#5

- AES 변환 알고리즘 (2/6)
  - 함수 (1/5)
    - invsubbyte는 0x63을 먼저 XOR하고 X^{-1} 곱을 비트 3개의 XOR로 계산한 후 곱셈 역원 출력

```
1. InvSubBytes (S)
2. {
3.     for (r = 0 to 3)
4.         for (c = 0 to 3)
5.             S[r][c] ← invsubbyte (S[r][c])
6. }
7. invsubbyte (byte)
8. {
9.     ByteToMatrix (byte, d)
10.    ByteToMatrix (0x63, y)
11.    for (i = 0 to 7)
12.        c[i] ← d[i] ⊕ y[i]
13.    for (i = 0 to 7)
14.        b[i] ← c[(i+2) mod 8] ⊕ c[(i+5) mod 8] ⊕ c[(i+7) mod 8]
15.    MatrixToByte (b, a)
16.    return inverse (a)
17.}
```

부록#5

- AES 변환 알고리즘 (3/6)
  - 함수 (2/5)
    - shiftrow는 t_c를 $(c-n)$ mod 4번째 칸에 저장
    - invshiftrow는 t_c를 $(c+n)$ mod 4번째 칸에 저장

```
1. ShiftRows (S)
2. {
3.     for (r = 1 to 3)
4.         shiftrow (Row (S, r), r)
5. }
6. shiftrow (row[4], n)
7. {
8.     CopyRow (row, t)
9.     for (c = 0 to 3)
10.        row[(c – n) mod 4] ← t[c]
11.}
```

```
1. InvShiftRows (S)
2. {
3.     for (r = 1 to 3)
4.         invshiftrow (Row (S, r), r)
5. }
6. invshiftrow (row[4], n)
7. {
8.     CopyRow (row, t)
9.     for (c = 0 to 3)
10.        row[(c + n) mod 4] ← t[c]
11.}
```

부록#5

- AES 변환 알고리즘 (4/6)
  - 함수 (3/5)
    - mixcolumn은 상수 행렬 C의 각 행과 열을 곱하며 •는 $GF(2^8)$ 곱셈

```
59. MixColumns (S)
60. {
61.     for (c = 0 to 3)
62.         mixcolumn (Column (S, c))
63. }
64. mixcolumn (col[4])
65. {
66.     CopyColumn (col, t)
67.     col[0] ← (0x02 • t[0]) ⊕ (0x03 • t[1]) ⊕ t[2] ⊕ t[3]
68.     col[1] ← t[0] ⊕ (0x02 • t[1]) ⊕ (0x03 • t[2]) ⊕ t[3]
69.     col[2] ← t[0] ⊕ t[1] ⊕ (0x02 • t[2]) ⊕ (0x03 • t[3])
70.     col[3] ← (0x03 • t[0]) ⊕ t[1] ⊕ t[2] ⊕ (0x02 • t[3])
71. }
```

부록#5

- AES 변환 알고리즘 (5/6)
  - 함수 (4/5)
    - invmixcolumn은 C^{-1}의 상수 0E, 0B, 0D, 09로 계산

```
72. InvMixColumns (S)
73. {
74.     for (c = 0 to 3)
75.         invmixcolumn (Column (S, c))
76. }
77. invmixcolumn (col[4])
78. {
79.     CopyColumn (col, t)
80.     col[0] ← (0x0E • t[0]) ⊕ (0x0B • t[1]) ⊕ (0x0D • t[2]) ⊕ (0x09 • t[3])
81.     col[1] ← (0x09 • t[0]) ⊕ (0x0E • t[1]) ⊕ (0x0B • t[2]) ⊕ (0x0D • t[3])
82.     col[2] ← (0x0D • t[0]) ⊕ (0x09 • t[1]) ⊕ (0x0E • t[2]) ⊕ (0x0B • t[3])
83.     col[3] ← (0x0B • t[0]) ⊕ (0x0D • t[1]) ⊕ (0x09 • t[2]) ⊕ (0x0E • t[3])
84. }
```

부록#5

- AES 변환 알고리즘 (6/6)
  - 함수 (5/5)
    - AddRoundKey는 상태의 c열에 라운드 키의 c번째 워드를 XOR
      - Cipher가 R라운드에서 w_{4R} ~ w_{4R+3}을 roundKey로 전달

```
85. AddRoundKey (S, roundKey[4])
86. {
87.     for (c = 0 to 3)
88.         Column (S, c) ← Column (S, c) ⊕ roundKey[c]
89. }
```

부록#6

- AES 키 확장 알고리즘
  - 과정
    1. 암호 키를 4바이트씩 연결(||)하여 $w_0 \sim w_3$ 생성
    2. i mod 4가 0이면 t로, 아니면 w_{i-1}로 w_i 계산

```
1.  KeyExpansion (key[16], w[44])
2.  {
3.      for (i = 0 to 3)
4.          w[i] ← key[4i] || key[4i+1] || key[4i+2] || key[4i+3]
5.      for (i = 4 to 43)
6.      {
7.          if (i mod 4 != 0)
8.              w[i] ← w[i – 1] ⊕ w[i – 4]
9.          else
10.         {
11.             t ← SubWord (RotWord (w[i – 1])) ⊕ RCon[i / 4]
12.             w[i] ← t ⊕ w[i – 4]
13.         }
14.     }
15. }
```

부록#7

- AES 암호와 역암호 알고리즘 (1/2)
  - 과정
    1. 블록을 상태로 바꾸고 라운드 전 AddRoundKey 적용
    2. 1~10라운드 변환을 적용하고 상태를 블록으로 변환

```
1.  Cipher (InBlock[16], OutBlock[16], w[0 … 43])
2.  {
3.      BlockToState (InBlock, S)
4.      S ← AddRoundKey (S, w[0 … 3])
5.      for (round = 1 to 10)
6.      {
7.          S ← SubBytes (S)
8.          S ← ShiftRows (S)
9.          if (round != 10) S ← MixColumns (S)
10.         S ← AddRoundKey (S, w[4 × round … 4 × round + 3])
11.     }
12.     StateToBlock (S, OutBlock)
13. }
```

부록#7

- AES 암호와 역암호 알고리즘 (2/2)
  - 함수
    - 역암호는 라운드 키를 w_{40} ~ w_{43}부터 역순으로 쓰고 라운드마다 InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns 순서로 적용

```
14. InvCipher (InBlock[16], OutBlock[16], w[0 … 43])
15. {
16.     BlockToState (InBlock, S)
17.     S ← AddRoundKey (S, w[40 … 43])
18.     for (round = 1 to 10)
19.     {
20.         S ← InvShiftRows (S)
21.         S ← InvSubBytes (S)
22.         S ← AddRoundKey (S, w[4 × (10 – round) … 4 × (10 – round) + 3])
23.         if (round != 10) S ← InvMixColumns (S)
24.     }
25.     StateToBlock (S, OutBlock)
26. }
```