변형된 AES-128

컴퓨터보안 002분반

12171678 이주호

Juho7668@naver.com / 01076687785

인하대학교
INHA UNIVERSITY

목록(함수별 정리)

0. Main & pch.h

Main 함수는 exe 파일의 입력에 따라서 encoding 객체 또는 decoding 객체를 생성한다. e 입력 시 encrypt를, d 입력 시 decrypt를 수행하며 각각 매개변수로 plaintext 경로, ciphertext 경로, Key 경로, 암호화 및 복호화 과정 출력 여부를 받는다.

```
encoding encode("plain.bin", "cipher.bin", "key.bin", printFlag);
decoding decode("cipher.bin", "plain2.bin", "key.bin", printFlag);
```

Pch.h에서 우리가 사용할 polynomial 0xE7, mix_col_y, mix_col_inv_y가 선언되어 있으며, Binary field 상에서의 곱셈연산 함수도 포함되어 있다.

곱셈연산은 다음과 같다.

Code

```
static int XTime(int X){
    return ((X << 1) & 0xFF) ^ (((X >> 7) & 1) * polynomial[ver]);
}
static int Multiply(int X, int Y){
    return ((Y >> 0 & 1) * X) ^
        ((Y >> 1 & 1) * XTime(X)) ^
        ((Y >> 2 & 1) * XTime(XTime(X))) ^
        ((Y >> 3 & 1) * XTime(XTime(XTime(X)))) ^
        ((Y >> 4 & 1) * XTime(XTime(XTime(XTime(X))))) ^
        ((Y >> 5 & 1) * XTime(XTime(XTime(XTime(XTime(X)))))) ^
        ((Y >> 6 & 1) * XTime(XTime(XTime(XTime(XTime(XTime(X))))))) ^
        ((Y >> 7 & 1) * XTime(XTime(XTime(XTime(XTime(XTime(XTime(X))))))));
}
```

XTime은 곱셈 연산 중 자리 올림이 있을 경우 $x^8 + x^7 + x^6 + x^5 + x^2 + x + 1$의 mod 연산을 수행해준다.

Multiply는 두 인자를 받고 각 비트 별 곱셈 연산을 수행하고 그 값을 리턴한다.

1. Encryption

1-0 Encoding 멤버 변수 및 멤버 함수

- 멤버 변수
  - Plaintext와 Ciphertext 파일 경로와 입출력 객체, 파일 내용 저장할 변수
    - `ofstream cipher; ifstream plain;`
    - `const char* cipher_Path; const char* plain_Path;`
    - `char ciphertext[CipherSize]; char plaintext[PlainSize];`
  - 암호화 과정 출력 여부 flag
    - `bool printFlag;`
  - 현재 round에 해당하는 key를 저장할 변수
    - `Key curKey[KeySize];`
- 멤버 함수
  - 암호화 수행 함수 (doEncoding(): Encryption 시작 함수)
    - `void Substitute();`
    - `void ShiftRows();`
    - `errno_t MixColumns();`
    - `void AddRoundKey();`
  - 현재 round에 해당하는 key를 계산 후 curKey에 저장 함수
    - `void getCurKey(int);`
      - 인자로 현재 round를 받고 해당하는 index를 계산하고 curKey에 저장한다.
      - 계산식: round * 16 + index
  - plaintext <- ciphertext 복사 함수
    - `void Copy();`

1-1 Add Round Key

함수명: void AddRoundKey();

Add round key 단계는 현재 round에 해당하는 key와 plaintext를 xor 연산을 수행 후 ciphertext에 저장한다.

Code

```
        for (size_t i = 0; i < KeySize; i++)
                ciphertext[i] = plaintext[i] ^ curKey[i];
```

1-2 Substitute Bytes

함수명: void Substitute();

Sbox를 통해서 plaintext 단순 문자 치환을 하고, ciphertext에 저장한다.

Code

```
        for (size_t i = 0; i < PlainSize; i++)
                ciphertext[i] = sbox.my_aes_sbox[(unsigned char)plaintext[i]];
```

1-3 Shift Rows

함수명: void ShiftRows();

현재 plaintext의 상태를 저장할 4 x 4 배열 state를 생성하여 각 행마다 shift를 수행한다.

State는 다음과 같이 저장한다.

Plaintext:

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

State:

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

그 후 shift를 수행하는데 i번째 해당하는 행은 i번 만큼의 **왼쪽** shift를 진행한다.
즉, 0번째 해당하는 행 1 5 9 13은 0번의 shift를 하며 마지막 3번째 행 4 8 12 16은 **왼쪽**으로 3만큼 shift를 진행한다.

Code

```
        // i행에 따라 i만큼 left shift진행
    char rotateState[4][4];
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++)
            rotateState[i][k] = state[i][(i + k) % 4];
```

그 후 shift 수행한 RotateState를 ciphertext에 저장한다.

전체 Code

```
    // 현재 plaintext의 상태를 4x4 행렬로 저장
    char state[4][4];
    size_t idx = 0;
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++, idx++)
            state[k][i] = plaintext[idx];

    // i행에 따라 i만큼 left shift진행
    char rotateState[4][4];
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++)
            rotateState[i][k] = state[i][(i + k) % 4];

    // left shift진행 후 다시 ciphertext에 저장
    idx = 0;
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++, idx++)
            ciphertext[idx] = rotateState[k][i];
```

1-4 Mix Columns

함수명: void MixColumns();

Shift rows와 동일하게 state에 plaintext를 저장한다.
해당 state의 하나의 열과 mix column의 행렬 mix_col_y와 행렬 곱셈 연산을 수행한다.
계산된 word가 ciphertext의 i번째 열로 삽입된다.

mix_col_y:

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

GF(2)상에서 덧셈연산은 XOR이므로 곱셈으로 계산된 결과들을 모두 XOR연산을 수행 후 해당 byte에 저장한다.

Code

```
char mixState[4][4];
for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++)
                mixState[k][i] = Multiply(state[0][i], mix_col_y[k][0])
                ^ Multiply(state[1][i], mix_col_y[k][1])
                ^ Multiply(state[2][i], mix_col_y[k][2])
                ^ Multiply(state[3][i], mix_col_y[k][3]);
```

그 후 행렬 연산을 수행한 mixState 를 ciphertext에 저장한다.

전체 Code

```
// 현재 plaintext의 상태를 4x4 행렬로 저장
char state[4][4];
size_t idx = 0;
for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++, idx++)
                state[k][i] = plaintext[idx];

// 각 열에 따라 state와 mix column을 위한 행렬(mix_col_y)을 곱하고 각 byte를 저장
char mixState[4][4];
for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++)
                mixState[k][i] = Multiply(state[0][i], mix_col_y[k][0])
                ^ Multiply(state[1][i], mix_col_y[k][1])
                ^ Multiply(state[2][i], mix_col_y[k][2])
                ^ Multiply(state[3][i], mix_col_y[k][3]);

// MixColumns진행 후 다시 ciphertext에 저장
idx = 0;
for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++, idx++)
                ciphertext[idx] = mixState[k][i];
```

2. Decryption

2-0 Decoding 멤버 변수 및 멤버 함수

Encryption과 과정이 동일하며, decoding을 위한 역행렬 등 일부만 수정된다.

- 멤버 변수
  - Plaintext와 Ciphertext 파일 경로와 입출력 객체, 파일 내용 저장할 변수
    - `ofstream cipher; ifstream plain;`
    - `const char* cipher_Path; const char* plain_Path;`
    - `char ciphertext[CipherSize]; char plaintext[PlainSize];`
  - 복호화 과정 출력 여부 flag
    - `bool printFlag;`
  - 현재 round에 해당하는 key를 저장할 변수
    - `Key curKey[KeySize];`
- 멤버 함수
  - 복호화 수행 함수 (doDecoding(): Decryption 시작 함수)
    - `void Substitute();`
    - `void ShiftRows();`
    - `errno_t MixColumns();`
    - `void AddRoundKey();`
  - 현재 round에 해당하는 key를 계산 후 curKey에 저장 함수
    - `void getCurKey(int);`
      - 인자로 현재 round를 받고 해당하는 index를 계산하고 curKey에 저장한다.
      - 계산식: round * 16 + index
  - ciphertext <- plaintext 복사 함수
    - `void Copy();`

2-1 Add Round Key

함수명: void AddRoundKey();

Add round key 단계는 현재 round에 해당하는 key와 ciphertext를 xor 연산을 수행 후 plaintext에 저장한다.

Code

```
        for (size_t i = 0; i < KeySize; i++)
                plaintext[i] = ciphertext[i] ^ curKey[i];
```

2-2 Inverse Substitute Bytes

함수명: void Substitute();

Inverse Sbox를 통해서 ciphertext단순 문자 치환을 하고, plaintext에 저장한다.

Code

```
        for (size_t i = 0; i < CipherSize; i++)
                plaintext[i] = sbox.my_inv_sbox[(unsigned char)ciphertext[i]];
```

2-3 Inverse Shift Rows

함수명: void ShiftRows();

현재 ciphertext의 상태를 저장할 4 x 4 배열 state를 생성하여 각 행마다 shift를 수행한다.

State는 다음과 같이 저장한다.

Plaintext:

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

State:

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

그 후 shift를 수행하는데 i번째 해당하는 행은 i번 만큼의 **오른쪽** shift를 진행한다.
즉, 0번째 해당하는 행 1 5 9 13은 0번의 shift를 하며 마지막 3번째 행 4 8 12 16은 **오른쪽**으로 3만큼 shift를 진행한다.

Code

```
        // i행에 따라 i만큼 left shift진행
    char rotateState[4][4];
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++)
            rotateState[i][k] = state[i][(k - i + 4) % 4];
```

그 후 shift 수행한 RotateState를 plaintext에 저장한다.

전체 Code

```
    // 현재 ciphertext의 상태를 4x4 행렬로 저장
    char state[4][4];
    size_t idx = 0;
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++, idx++)
            state[k][i] = ciphertext[idx];

    // i행에 따라 i만큼 right shift진행
    char rotateState[4][4];
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++)
            rotateState[i][k] = state[i][(k - i + 4) % 4];

    // right shift진행 후 다시 plaintext에 저장
    idx = 0;
    for (size_t i = 0; i < 4; i++)
        for (size_t k = 0; k < 4; k++, idx++)
            plaintext[idx] = rotateState[k][i];
```

2-4 Inverse Mix Columns

함수명: void MixColumns();

Inverse Shift rows와 동일하게 state에 ciphertext를 저장한다.
해당 state의 하나의 열과 mix column의 행렬 mix_col_inv_y와 행렬 곱셈 연산을 수행한다.
계산된 word가 plaintext의 i번째 열로 삽입된다.

mix_col_inv_y:

0x0E	0x0B	0x0D	0x09
0x09	0x0E	0x0B	0x0D
0x0D	0x09	0x0E	0x0B
0x0B	0x0D	0x09	0x0E

Code

```
        char mixState[4][4];
        for (size_t i = 0; i < 4; i++)
                for (size_t k = 0; k < 4; k++)
                        mixState[k][i] = Multiply(state[0][i], mix_col_inv_y[k][0])
                        ^ Multiply(state[1][i], mix_col_inv_y[k][1])
                        ^ Multiply(state[2][i], mix_col_inv_y[k][2])
                        ^ Multiply(state[3][i], mix_col_inv_y[k][3]);
```

그 후 행렬 연산을 수행한 mixState를 plaintext에 저장한다.

전체 Code

```
        // 현재 ciphertext의 상태를 4x4 행렬로 저장
        char state[4][4];
        size_t idx = 0;
        for (size_t i = 0; i < 4; i++)
                for (size_t k = 0; k < 4; k++, idx++)
                        state[k][i] = ciphertext[idx];

        // 각 열에 따라 state와 mix column을 위한 역행렬(mix_col_inv_y)을 곱하고 각 byte를 저장
        char mixState[4][4];
        for (size_t i = 0; i < 4; i++)
                for (size_t k = 0; k < 4; k++)
                        mixState[k][i] = Multiply(state[0][i], mix_col_inv_y[k][0])
                        ^ Multiply(state[1][i], mix_col_inv_y[k][1])
                        ^ Multiply(state[2][i], mix_col_inv_y[k][2])
                        ^ Multiply(state[3][i], mix_col_inv_y[k][3]);

        // Inverse MixColumns진행 후 다시 plaintext에 저장
        idx = 0;
        for (size_t i = 0; i < 4; i++)
                for (size_t k = 0; k < 4; k++, idx++)
                        plaintext[idx] = mixState[k][i];
```

3. Key Expansion

3-0 KeyExpansion 멤버 변수 및 멤버 함수

- 멤버 변수
  - Key 파일 경로와 입출력 객체, key value 저장할 변수
    - `ifstream ReadKey;`
    - `Key key[KeySize * (Round + 1)];`
    - `const char* key_path;`
  - 복호화 과정 출력 여부 flag
    - `bool printFlag;`
  - 현재 round에 해당하는 g function value를 저장할 변수
    - `Key gValue[4];`
- 멤버 함수
  - 키 확장 수행 함수 (do_KeyExpansion(): Expansion 시작 함수)
  - G function
    - `void G_function(int);`

3-1 XOR Calculate(do_KeyExpansion)

do_KeyExpansion 함수에서 각 열의 word 단위로 XOR연산을 수행한다.

각 round에서 다음의 과정을 수행한다.

- 첫번째 word는 이전 round의 w0와 이전 round의 w3의 g function value, gValue를 XOR 연산을 하여 저장한다.
- 나머지 i번째 word는 이전 round의 w_i와 현재 round의 $w_{(i-1)}$를 XOR 연산을 하여 저장한다.

또한 각 byte들의 index 계산 식은 다음과 같다.
계산식: round * 16 + index

Code

```
size_t curIdx = 16;
for (size_t round = 1; round <= Round; round++) {
        // 현재 round의 gvalue를 얻는다.
        G_function(round-1);

        // gvlaue와 이전 round의 key값을 xor한다. (w0~w3)
        for (size_t w0 = 0; w0 < 4; w0++)
                key[round * KeySize + w0] = (key[(round - 1) * KeySize + w0] ^ gValue[w0]); //XOR

        // 바로 직전의 word와 현재 word를 xor한다. (w12~w15)
        for (size_t i = 1; i < 4; i++)
                for (size_t k = 0; k < 4; k++)
                        key[round * KeySize + i * 4 + k] =
                         (key[(round - 1) * KeySize + i * 4 + k] ^
                           key[round * KeySize + (i - 1) * 4 + k]); //XOR
}
```

3-2 g function

함수명: void G_function(int);

현재 round를 받고 계산한 idx를 저장하여 이전 round의 key w3에 접근한다.

G function은 다음의 3가지 작업을 수행한다.

- Left shift
  - gValue에 w3요소가 left shift 과정을 거친 값을 저장
- Substitute byte
  - Sbox를 통해서 단순 문자 치환을 한다.
- XOR RCj
  - 각 round에 해당하는 RCj 상수와 XOR 연산을 수행한다.
    - RCj는 8 round 이후부터 polynomial 즉, 0x01E7 과 곱셈 연산을 수행한다.

Code

```
int idx = (KeySize * round) + 12;

// left shift
for (size_t i = 0; i < 4; i++)
        gValue[(3 + i) % 4] = key[idx + i];

// sbox 통과
for (size_t i = 0; i < 4; i++)
        gValue[i] = sbox.my_aes_sbox[(unsigned char)gValue[i]];

// RCj XOR
uint8_t RCj[4] = { 0x01, 0, 0, 0 };
if (round >= 8) {
        uint8_t temp = polynomial[ver];
        RCj[0] = Multiply(temp, round-7);
}
else
        RCj[0] = (RCj[0] << (round));

for (size_t i = 0; i < 4; i++)
         gValue[i] ^= RCj[i];
```

4. S-box

4-0 S-box 멤버 변수 및 멤버 함수

- 멤버 변수
  - 계산식에 사용할 행렬과 역행렬
    - `uint8_t forward[8];`
    - `uint8_t inverse[8];`
  - 역원 저장 변수
    - `int dgh[3];`
  - sbox와 inverse sbox 저장할 변수
    - `uint8_t my_aes_sbox[256];`
    - `uint8_t my_inv_sbox[256];`
- 멤버 함수
  - 확장된 유클리드 알고리즘 구현 메소드와 역원 반환 메소드
    - `void bin_ext_euclid(uint8_t);`
    - `uint8_t bin_inv(uint8_t);`
  - Sbox 생성 메소드
    - `void make_Sbox();`
      - 전체의 sbox와 inverse sbox를 구한다.
    - `uint8_t get_Sbox(uint8_t);`
      - 입력 받은 값의 sbox 연산 결과 값을 반환한다.
    - `uint8_t get_inv_Sbox(uint8_t);`
      - 입력 받은 값의 inverse sbox 연산 결과 값을 반환한다.

4-1 Extended Euclid Algorithm in Binary Field

함수명: void bin_ext_euclid (uint8_t);

Binary field 상에서 확장된 유클리드 알고리즘을 구현한 메소드이다.

a * g[0] + b * h[0] = u

a * g[1] + b * h[1] = v

우리는 $x^8 + x^7 + x^6 + x^5 + x^2 + x + 1$에 해당하는 a의 역원을 구하는 것이 목적이다.
따라서, a는 입력 값이며, b는 항상 0x01E7의 값을 가진다.

Code

```
        int b = (0x100 | polynomial[ver]);
        int u = a, v = b;
        int g[2] = { 1, 0 };
        int h[2] = { 0, 1 };

        while (u != 0){
                int j = (deg(u) - deg(v));
                if (j < 0) {
                        swap(u, v);
                        swap(g[0], g[1]);
                        swap(h[0], h[1]);
                        j = -j;
                }
                u ^= v << j;
                g[0] ^= g[1] << j;
                h[0] ^= h[1] << j;
        }
        dgh[0] = v;
        dgh[1] = g[1];
        dgh[2] = h[1];
```

3-2 Inverse in Binary Field

함수명: uint8_tbin_inv(uint8_t);

입력 값 a에 해당하는 역원 g의 값을 리턴한다.

Code

```
bin_ext_euclid(a);
        return dgh[1];
```

4-2 Get sbox and inverse sbox

- make_Sbox는 단순히 0부터 255까지의 값들을 get_Sbox와 get_inv_Sbox에 대입하여 그에 상응하는 값들을 일일이 생성한다.

  - Code

```
for (size_t i = 0; i < 256; i++) {
        my_aes_sbox[i] = get_Sbox(i);
        my_inv_sbox[i] = get_inv_Sbox(i);
}
```

4-2-1 get_Sbox

Get sbox 과정은 다음과 같다.

1. 입력 값 e에 해당하는 역원을 구한다.
2. 해당 역원에 대해서 sbox 계산할 행렬 forward와 bit 단위의 행렬 곱셈을 한다.
   A. bit단위의 행렬 곱을 하기 전에 먼저 byte 단위로 and 연산을 수행한다.
      - Bit 단위의 곱셈은 and 연산을 하는 것과 동일하며
      - 계산된 Byte의 전체 bit를 더하는 것은 XOR연산을 하는 것과 동일하다.
   B. 그 후 각 bit별로 XOR 연산을 수행하고 b_output에 bit값을 저장한다.
3. 행렬 곱셈의 결과와 상수 0x63을 XOR 연산 수행한다.
   A. temp 변수 각 비트에 상수 c와 계산된 bit b_output를 xor 연산을 하여 저장한다.
   B. 이는 bit별 XOR 연산을 수행한다.

Code

```
uint8_t inv_e = bin_inv(e);// 입력받은 요소의 역원 구하기
uint8_t temp = 0x00, c = 0x63;
uint8_t b_Matrix_Cal[8] = { 0, };
uint8_t b_output[8] = { 0, };
// sbox 연산 중 행렬 연산
for (size_t k = 0; k < 8; k++)
        b_Matrix_Cal[k] = (forward[k] & inv_e);
for (size_t k = 0; k < 8; k++) {
        for (size_t t = 0; t < 8; t++)
                temp ^= ((b_Matrix_Cal[k] >> t) & 0x01);

        temp ^= ((c >> k) & 0x01);
        b_output[k] = temp;
        temp = 0x00;
}

// 행렬 연산결과와 0x63 상수 XOR연산
for (size_t k = 0; k < 8; k++)
        temp |= (b_output[k] << k);

// 연산된 결과 값 리턴
return temp;
```

4-2-2 get_inv_Sbox

Get inverse sbox 과정은 다음과 같다.

1. 입력 값 d에 대해서 inverse sbox 계산할 행렬 inverse와 bit 단위의 행렬 곱셈을 한다.
   A. bit단위의 행렬 곱을 하기 전에 먼저 byte 단위로 and 연산을 수행한다.
      - get_sbox 연산과 동일한 이유이다.
   B. 그 후 각 bit별로 XOR 연산을 수행하고 b_output에 bit값을 저장한다.
2. 행렬 곱셈의 결과와 상수 0x63의 역원 0x05를 XOR 연산 수행한다.
   A. temp 변수 각 비트에 상수 c와 계산된 bit b_output를 xor 연산을 하여 저장한다.
   B. 이는 bit별 XOR 연산을 수행한다.
3. 계산된 temp 값의 역원을 구하고 그 값을 리턴한다.

Code

```
uint8_t temp = 0x00, c = 0x05;
uint8_t b_Matrix_Cal[8] = { 0, };
uint8_t b_output[8] = { 0, };

// Inverse sbox 연산 중 행렬 연산
for (size_t k = 0; k < 8; k++)
        b_Matrix_Cal[k] = (inverse[k] & d);
for (size_t k = 0; k < 8; k++) {
        for (size_t t = 0; t < 8; t++)
                temp ^= ((b_Matrix_Cal[k] >> t) & 0x01);

        temp ^= ((c >> k) & 0x01);
        b_output[k] = temp;
        temp = 0x00;
}

// 행렬 연산결과와 0x63 상수의 역원 0x05 XOR연산
for (size_t k = 0; k < 8; k++)
        temp |= (b_output[k] << k);

// 연산된 요소의 역원 구하고 리턴
return bin_inv(temp);
```

5. 예제 출력

- Sbox & Inverse Sbox

```
----------------------------------------
sbox
63 7c 18 31 2a a 4a fa c7 eb 23 ad 3 3a 5b bb
c5 d4 d3 e8 43 50 4 54 a7 1d cf 1b 8b 7e fb f9
c4 ea 4c 85 3b 86 52 97 87 fd e b6 d0 4b f8 1c
1 f2 5c f1 c1 1a ab 6b 17 d6 19 14 db 9f da 8d
44 c6 53 a1 f4 9b e4 37 4f ee 65 57 f 4e ed 71
e5 21 2c b4 d5 b5 89 c9 ba 3c 83 69 5a cd dc ec
a6 26 5f bc fc 99 de 12 32 68 2b 60 f3 ef 67 98
59 11 4d dd aa 29 d8 84 3f 10 e9 b1 bf 77 e0 82
f0 c3 45 ce 8f 90 f6 6f a8 6 1f 15 a0 2d bd af
75 b8 a5 b2 94 9 79 a3 55 3e f5 80 24 e3 6a 2
20 51 42 7 30 8e 88 c2 cc b3 8 96 16 61 36 ca
7b 6d 38 22 13 ff 66 40 b b7 c0 3d 48 62 a4 d9
81 7f 35 0 7d 7a 8c 9c ac f7 1e 6e 49 a2 2f 6c
cb 92 e6 28 47 39 e2 78 df 27 25 e7 95 d7 9e 34
8a 41 ae 70 74 33 c8 5e 73 91 46 a9 be 9a 64 e1
b9 58 2e 5d d2 d1 fe 72 d 5 9d c 56 b0 93 76

inverse_sbox
c3 30 9f c 16 f9 89 a3 aa 95 5 b8 fb f8 2a 4c
79 71 67 b4 3b 8b ac 38 2 3a 35 1b 2f 19 ca 8a
a0 51 b3 a 9c da 61 d9 d3 75 4 6a 52 8d f2 ce
a4 3 68 e5 df c2 ae 47 b2 d5 d 24 59 bb 99 78
b7 e1 a2 14 40 82 ea d4 bc cc 6 2d 22 72 4d 48
15 a1 26 42 17 98 fc 4b f1 70 5c e 32 f3 e7 62
6b ad bd 0 ee 4a b6 6e 69 5b 9e 37 cf b1 cb 87
e3 4f f7 e8 e4 90 ff 7d d7 96 c5 b0 1 c4 1d c1
9b c0 7f 5a 77 23 25 28 a6 56 e0 1c c6 3f a5 84
85 e9 d1 fe 94 dc ab 27 6f 65 ed 45 c7 fa de 3d
8c 43 cd 97 be 92 60 18 88 eb 74 36 c8 b e2 8f
fd 7b 93 a9 53 55 2b b9 91 f0 58 f 63 8e ec 7c
ba 34 a7 81 20 10 41 8 e6 57 af d0 a8 5d 83 1a
2c f5 f4 12 11 54 39 dd 76 bf 3e 3c 5e 73 66 d8
7e ef d6 9d 46 50 d2 db 13 7a 21 9 5f 4e 49 6d
80 33 31 6c 44 9a 86 c9 2e 1f 7 1e 64 29 f6 b5
----------------------------------------
```

- Key

```
k0: 0 1 2 3 4 5 6 7 8 9 a b c d e f
k1: 3b 5a ffb9 0 3f 5f ffbf 7 37 56 ffb5 c 3b 5b ffbb 3
k2: 50 67 ff88 14 6f 38 37 13 58 6e ff82 1f 63 35 39 1c
k3: 4e ffb1 3 ffa8 21 ff89 34 ffbb 79 ffe7 ffb6 ffa4 1a ffd2 ff8f ffb8
k4: ffa0 1e 8 67 ff81 ff97 3c ffdc fff8 70 ff8a 78 ffe2 ffa2 5 ffc0
k5: fff2 14 ff89 ffc9 73 ff83 ffb5 15 ff8b fff3 3f 6d 69 51 3a ffad
k6: fff3 d ffe8 ffa1 ff80 ff8e 5d ffb4 b 7d 62 ffd9 62 2c 58 74
k7: 63 ffb7 42 fffe ffe3 39 1f 4a ffe8 44 7d ff93 ff8a 68 25 ffe7
k8: ffd1 31 1c ffe1 32 8 3 ffab ffda 4c 7e 38 50 24 5b ffdf
k9: d 58 28 4 3f 50 2b ffaf ffe5 1c 55 ff97 ffb5 38 e 48
k10: 33 3 67 fffb c 53 4c 54 ffe9 4f 19 ffc3 5c 77 17 ff8b
```

- Encrypt

```
plaintext: 0 11 22 33 44 55 66 77 ff88 ff99 ffaa ffbb ffcc ffdd ffee ffff
AR: 0 10 20 30 40 50 60 70 ff80 ff90 ffa0 ffb0 ffc0 ffd0 ffe0 fff0
1 round
BS: 63 ffc5 ffc4 1 44 ffe5 ffa6 59 fff0 75 20 7b ff81 ffcb ff8a ffb9
SR: 63 ffe5 20 ffb9 44 75 ff8a 1 fff0 ffcb ffc4 59 ff81 ffc5 ffa6 7b
MR: ff97 ff97 ffea fff5 ff9c ffd6 ffc1 31 20 73 ffbf 4a ff90 ff9a 62 fff1
AR: ffac ffcd 53 fff5 ffa3 ff89 7e 36 17 25 a 46 ffab ffc1 ffd9 fff2
2 round
BS: 16 ffa2 ffb4 ffd1 7 6 ffe0 ffab 54 ff86 23 ffe4 ff96 7f 27 2e
SR: 16 6 23 2e 7 ff86 27 ffd1 54 7f ffb4 ffab ff96 ffa2 ffe0 ffe4
MR: 2b 51 24 43 ff95 54 5b ffed 36 3a ffbe ff86 ffce 16 ffd8 30
AR: 7b 36 ffac 57 fffa 6c 6c fffe 6e 54 3c ff99 ffad 23 ffe1 2c
3 round
BS: ffb1 ffab 16 ffc9 ff9d fff3 fff3 ff93 67 ffd5 ffdb 3e 61 ff85 41 ffd0
SR: ffb1 fff3 ffdb ffd0 ff9d ffd5 41 ffc9 67 ff85 16 ff93 61 ffab fff3 3e
MR: 7c ffea ff84 5b ffcd ffda 76 ffa1 23 23 ff9c fffb 15 1c ff89 ff87
AR: 32 5b ff87 fff3 ffec 53 42 1a 5a ffc4 2a 5f f ffce 6 3f
4 round
BS: 5c 69 6f 5d ffbe ffb4 53 ffcf ff83 7d e ffec ffbb 2f 4a ff8d
SR: 5c ffb4 e ff8d ffbe 7d 4a 5d ff83 2f 6f ffcf ffbb 69 53 ffec
MR: 0 4c ff84 ffa3 b ffc7 ffb0 ffa8 30 ffa3 ffc4 5b ff95 70 ffa7 2f
AR: ffa0 52 ff8c ffc4 ff8a 50 ff8c 74 ffc8 ffd3 4e 23 77 ffd2 ffa2 ffef
5 round
BS: 20 2c ffa0 7d 1f ffe5 ffa0 ffaa ffac 28 ffed ff85 ff84 ffe6 42 ffe1
SR: 20 ffe5 ffed ffe1 1f 28 42 7d ffac ffe6 ffa0 ffaa ff84 2c ffa0 ff85
MR: ff84 3c 3c 4d 79 fff4 34 ffb1 78 2a fff4 ffe6 ffbe 5e 67 a
AR: 76 28 ffb5 ff84 a 77 ff81 ffa4 fff3 ffd9 ffcb ff8b ffd7 f 5d ffa7
6 round
BS: ffd8 ff87 ffff ff8f 23 ff84 ffc3 30 5d 27 6e 15 78 ffbb ffcd ffc2
SR: ffd8 ff84 6e ffc2 23 27 ffcd ff8f 5d ffbb ffff 30 78 ff87 ffc3 15
MR: ff90 47 21 6 6d 52 f 76 5f 1a ffaf ffc3 48 26 ffa1 ffe6
AR: 63 4a ffc9 ffa7 ffed ffdc 52 ffc2 54 67 ffcd 1a 2a a fff9 ff92
7 round
BS: ffbc 65 fff7 ffc2 ff9a ff95 2c 35 ffd5 12 ffa2 ffcf e 23 5 ffa5
SR: ffbc ff95 ffa2 ffa5 ff9a 12 5 ffc2 ffd5 23 fff7 35 e 65 2c ffcf
MR: ffc0 ffd5 ff82 ffb9 22 73 23 3d ffea 58 ffa0 26 50 7f ff85 22
AR: ffa3 62 ffc0 47 ffc1 4a 3c 77 2 1c ffdd ffb5 ffda 17 ffa0 ffc5
8 round
BS: 7 5f ff81 37 7f 65 ffdb ff84 18 ff8b ffd7 ffff 25 54 20 7a
SR: 7 65 ffd7 7a 7f ff8b 20 37 18 54 ff81 ff84 25 5f ffdb ffff
MR: c 29 ffa5 4f ff93 ffd9 ffed 44 ffc9 50 ffc2 12 ff8f ffee ffcd fff2
AR: ffdd 18 ffb9 ffae ffa1 ffd1 ffee ffef 13 1c ffbc 2a ffdf ffca ff96 2d
9 round
BS: ffd7 ffa7 ffb7 36 51 ff92 64 ffe1 ffe8 ff8b 48 e 34 1e 79 4b
SR: ffd7 ff92 48 4b 51 ff8b 79 36 ffe8 1e ffb7 ffe1 34 ffa7 64 e
MR: 1b ff87 8 ffd2 ff97 1d 72 6d 43 b ffbb 53 c 3f 49 ff83
AR: 16 ffdf 20 ffd6 ffa8 4d 59 ffc2 ffa6 17 ffee ffc4 ffb9 7 47 ffcb
10 round
BS: 4 34 ffc4 ffe2 ffcc 4e 3c 35 ff88 54 64 7d ffb7 fffa 37 6e
SR: 4 4e 64 6e ffcc 54 37 ffe2 ff88 fffa ffc4 35 ffb7 34 3c 7d
AR: 37 4d 3 ff95 ffc0 7 7b ffb6 61 ffb5 ffdd fff6 ffeb 43 2b fff6
encrypte: 37 4d 3 ff95 ffc0 7 7b ffb6 61 ffb5 ffdd fff6 ffeb 43 2b fff6
```

- Decrypt

```
ciphertext: 37 4d 3 ff95 ffc0 7 7b ffb6 61 ffb5 ffdd fff6 ffeb 43 2b fff6
AR: 37 4d 3 ff95 ffc0 7 7b ffb6 61 ffb5 ffdd fff6 ffeb 43 2b fff6
9 round
Inv SR: 4 4e 64 6e ffcc 54 37 ffe2 ff88 fffa ffc4 35 ffb7 34 3c 7d
Inv BS: 4 34 ffc4 ffe2 ffcc 4e 3c 35 ff88 54 64 7d ffb7 fffa 37 6e
AR: 16 ffdf 20 ffd6 ffa8 4d 59 ffc2 ffa6 17 ffee ffc4 ffb9 7 47 ffcb
MR: 1b ff87 8 ffd2 ff97 1d 72 6d 43 b ffbb 53 c 3f 49 ff83
8 round
Inv SR: ffd7 ff92 48 4b 51 ff8b 79 36 ffe8 1e ffb7 ffe1 34 ffa7 64 e
Inv BS: ffd7 ffa7 ffb7 36 51 ff92 64 ffe1 ffe8 ff8b 48 e 34 1e 79 4b
AR: ffdd 18 ffb9 ffae ffa1 ffd1 ffee ffef 13 1c ffbc 2a ffdf ffca ff96 2d
MR: c 29 ffa5 4f ff93 ffd9 ffed 44 ffc9 50 ffc2 12 ff8f ffee ffcd fff2
7 round
Inv SR: 7 65 ffd7 7a 7f ff8b 20 37 18 54 ff81 ff84 25 5f ffdb ffff
Inv BS: 7 5f ff81 37 7f 65 ffdb ff84 18 ff8b ffd7 ffff 25 54 20 7a
AR: ffa3 62 ffc0 47 ffc1 4a 3c 77 2 1c ffdd ffb5 ffda 17 ffa0 ffc5
MR: ffc0 ffd5 ff82 ffb9 22 73 23 3d ffea 58 ffa0 26 50 7f ff85 22
6 round
Inv SR: ffbc ff95 ffa2 ffa5 ff9a 12 5 ffc2 ffd5 23 fff7 35 e 65 2c ffcf
Inv BS: ffbc 65 fff7 ffc2 ff9a ff95 2c 35 ffd5 12 ffa2 ffcf e 23 5 ffa5
AR: 63 4a ffc9 ffa7 ffed ffdc 52 ffc2 54 67 ffcd 1a 2a a fff9 ff92
MR: ff90 47 21 6 6d 52 f 76 5f 1a ffaf ffc3 48 26 ffa1 ffe6
5 round
Inv SR: ffd8 ff84 6e ffc2 23 27 ffcd ff8f 5d ffbb ffff 30 78 ff87 ffc3 15
Inv BS: ffd8 ff87 ffff ff8f 23 ff84 ffc3 30 5d 27 6e 15 78 ffbb ffcd ffc2
AR: 76 28 ffb5 ff84 a 77 ff81 ffa4 fff3 ffd9 ffcb ff8b ffd7 f 5d ffa7
MR: ff84 3c 3c 4d 79 fff4 34 ffb1 78 2a fff4 ffe6 ffbe 5e 67 a
4 round
Inv SR: 20 ffe5 ffed ffe1 1f 28 42 7d ffac ffe6 ffa0 ffaa ff84 2c ffa0 ff85
Inv BS: 20 2c ffa0 7d 1f ffe5 ffa0 ffaa ffac 28 ffed ff85 ff84 ffe6 42 ffe1
AR: ffa0 52 ff8c ffc4 ff8a 50 ff8c 74 ffc8 ffd3 4e 23 77 ffd2 ffa2 ffef
MR: 0 4c ff84 ffa3 b ffc7 ffb0 ffa8 30 ffa3 ffc4 5b ff95 70 ffa7 2f
3 round
Inv SR: 5c ffb4 e ff8d ffbe 7d 4a 5d ff83 2f 6f ffcf ffbb 69 53 ffec
Inv BS: 5c 69 6f 5d ffbe ffb4 53 ffcf ff83 7d e ffec ffbb 2f 4a ff8d
AR: 32 5b ff87 fff3 ffec 53 42 1a 5a ffc4 2a 5f f ffce 6 3f
MR: 7c ffea ff84 5b ffcd ffda 76 ffa1 23 23 ff9c fffb 15 1c ff89 ff87
2 round
Inv SR: ffb1 fff3 ffdb ffd0 ff9d ffd5 41 ffc9 67 ff85 16 ff93 61 ffab fff3 3e
Inv BS: ffb1 ffab 16 ffc9 ff9d fff3 fff3 ff93 67 ffd5 ffdb 3e 61 ff85 41 ffd0
AR: 7b 36 ffac 57 fffa 6c 6c fffe 6e 54 3c ff99 ffad 23 ffe1 2c
MR: 2b 51 24 43 ff95 54 5b ffed 36 3a ffbe ff86 ffce 16 ffd8 30
1 round
Inv SR: 16 6 23 2e 7 ff86 27 ffd1 54 7f ffb4 ffab ff96 ffa2 ffe0 ffe4
Inv BS: 16 ffa2 ffb4 ffd1 7 6 ffe0 ffab 54 ff86 23 ffe4 ff96 7f 27 2e
AR: ffac ffcd 53 fff5 ffa3 ff89 7e 36 17 25 a 46 ffab ffc1 ffd9 fff2
MR: ff97 ff97 ffea fff5 ff9c ffd6 ffc1 31 20 73 ffbf 4a ff90 ff9a 62 fff1
0 round
Inv SR: 63 ffe5 20 ffb9 44 75 ff8a 1 fff0 ffcb ffc4 59 ff81 ffc5 ffa6 7b
Inv BS: 63 ffc5 ffc4 1 44 ffe5 ffa6 59 fff0 75 20 7b ff81 ffcb ff8a ffb9
AR: 0 10 20 30 40 50 60 70 ff80 ff90 ffa0 ffb0 ffc0 ffd0 ffe0 fff0
decrypt: 0 11 22 33 44 55 66 77 ff88 ff99 ffaa ffbb ffcc ffdd ffee ffff
```

- Bin file

D:₩인하대학교₩3학년 1학기(2021-1)₩컴보₩과제₩10w AES₩AES128₩AES-128₩x64₩Debug₩plain.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	..”3DUfw^™ª»ÌÝîÿ
00000010	23	85	96	82	83	74	72	91	20	28	57	67	03	04	98	85	#…–‚ƒtr‘ (Wg..˜…
00000020	76	88	58	67	30	40	06	80	83	02	00	37	86	63	74	37	vˆXg0@.€ƒ..7†ct7

D:₩인하대학교₩3학년 1학기(2021-1)₩컴보₩과제₩10w AES₩AES128₩AES-128₩x64₩Debug₩cipher.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	37	4D	03	95	C0	07	7B	B6	61	B5	DD	F6	EB	43	2B	F6	7M.•À.{¶aµÝöëC+ö
00000010	A5	68	65	22	E1	41	CE	85	C7	9C	D1	3F	E1	AF	37	67	¥he"áAÎ…ÇœÑ?á¯7g
00000020	FD	3D	08	88	23	3E	DD	69	89	AD	5D	34	D0	8F	1F	C5	ý=.ˆ#>Ýi‰.]4Ð..Å

D:₩인하대학교₩3학년 1학기(2021-1)₩컴보₩과제₩10w AES₩AES128₩AES-128₩x64₩Debug₩plain2.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	..”3DUfw^™ª»ÌÝîÿ
00000010	23	85	96	82	83	74	72	91	20	28	57	67	03	04	98	85	#…–‚ƒtr‘ (Wg..˜…
00000020	76	88	58	67	30	40	06	80	83	02	00	37	86	63	74	37	vˆXg0@.€ƒ..7†ct7