Steganography

LSB 변조

https://youtu.be/3ZY_-M5SWec

Contents

CryptoCraft LAB

01. Steganography

Steganography

❖ Cover data에 의미 있는 비밀 정보를 숨기는 기술

- 비밀 정보의 존재 유무를 숨김

❖ 스테가노그래피 & 암호화

- 스테가노그래피의 비밀 정보는 암호화된 상태로도 은닉 가능 → 상호보완적 관계

❖ Cover data

- host file (문자, 이미지, 음악파일, 동영상 등) → 비밀 정보를 Cover image에 숨김

  ex) 이미지 파일에 비밀 메시지가 담긴 .txt파일을 숨김

❖ 디지털 스테가노그래피

- Cover image 유형에 따라 image, text, audio, video steganography로 분류

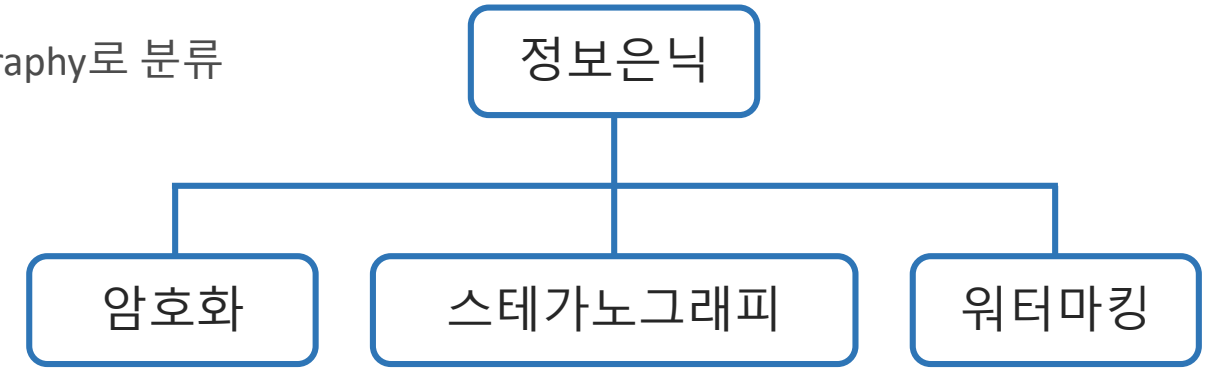


Image Steganography

❖ 공간 도메인 기법 : Bitmap

- cover image의 pixel을 조작하여 비밀 데이터 은닉 → lsb steganography
- pixel 값에 직접 데이터를 숨기고 추출

  → 비교적 빠르게 많은 데이터 은닉 가능 (cover image의 최대 50%까지도 가능)
- 이미지 변환에 약함

❖ 변환 도메인 기법 (주파수 도메인 기법) : jpeg

- 주파수 또는 변환 영역에 비밀 데이터 내장
- DCT 변환 등을 이용 → 이미지 변환 등으로 인한 비밀 정보 훼손 X
- 삽입 가능 용량 → cover image의 약 5 ~ 15%
- Steganalysis 공격에 강함

❖ 왜곡 기법 / 마스킹 & 필터링 기법

- 이미지 신호를 왜곡 / 밝기 등을 수정하여 은닉

*DCT 변환 : 공간영역 → 주파수 영역 변환
낮은 주파수(DC)로 색상이 몰리고
색상 변화가 있는 경우 높은 주파수(AC)에 위치
AC 성분은 생략해도 화질 차이에 영향 X

Image Steganography 조건

❖ 데이터 용량 (Data capacity)

- 데이터 용량이 클수록 더 많은 데이터 내장 가능
- 은닉 데이터 多 → 노이즈 등의 시각적 이상 현상, 비정상적 히스토그램 발생

❖ 비검출성 (Undetectable)

- 삽입한 메시지가 검출되지 않아야 함

❖ 비인지성 (Imperceptibility)

- 삽입된 비밀 정보에 의한 원본 데이터의 변형이 없어야 함

❖ 견고성 (Robustness)

- 삽입된 비밀 정보는 데이터 변형에도 삭제 불가능 해야 함

➢ 목표

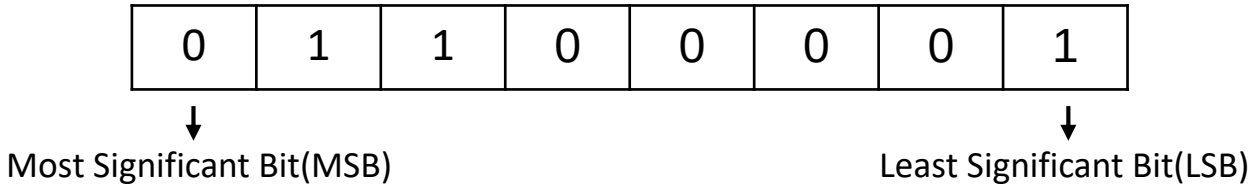
- 가능한 많은 데이터를 은닉하는 동시에 stego image와 cover image간의 시각적 및 통계적 유사성 유지

  → 인지할 수 없게 하여 공격을 피함

LSB Steganography

❖ Cover image의 LSB를 숨기려고 하는 비밀 데이터 1bit로 대체

- 비인지성 확보 위해 MSB 값이 큰 픽셀의 LSB에 데이터 은닉



❖ Bitmap image에 가장 많이 적용

- 24-bit BMP 파일에 주로 사용
- 24 bit → RGB(8 bit, 8 bit, 8 bit) → 한 pixel으로 나타낼 수 있는 색상의 수 = 2^{24}개
- MSB는 변화가 크지만 하위 비트로 갈수록 육안으로 인식 불가

  → 0xFFFFFF >> 0xFEFEFE (11111111…1111 → 11111110….1110)

*127 → 126 : LSB 변조 시 육안으로 인식 불가

02. New LSB-based colour image steganography

New LSB-based colour imag

Sender

1. 비밀 데이터(x)
2. CRC-32 checksum
3. 압축
4. AES 암호화
5. 암호화 된 데이터에 대한 헤더 암호화
6. 다음 픽셀 위치를 정하는 시드키를 사용하여
   암호화 된 코드워드와 헤더정보를 커버이미지에 삽입
   → Fisher-Yayes Shuffle 알고리즘 활용

Receiver

1. 스테고이미지로부터 암호화된 헤더 정보 추출 (Fisher-Yayes Shuffle 알고리즘)
   → sender와 동일한 픽셀 선택하는 과정
   해당 픽셀의 LSB로부터 정보 추출
2. 추출한 길이 정보를 사용하여 암호화 된 데이터 추출
3. AES 복호화
4. CRC-32 checksum 통한 무결성 검사

*Fisher-Yayes Shuffle 알고리즘 : 유한한 시퀀스의 랜덤 순열을 만드는 알고리즘

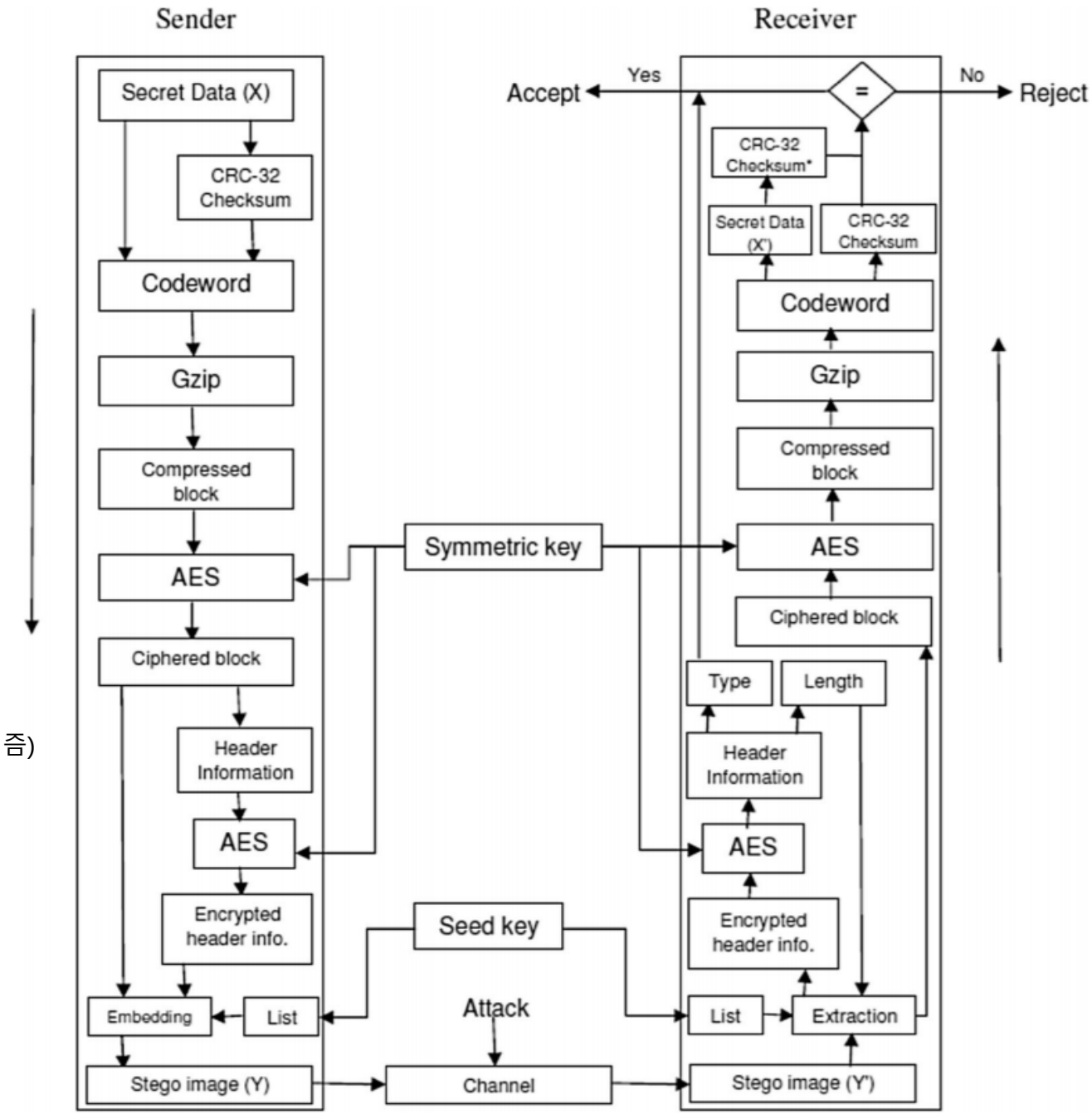
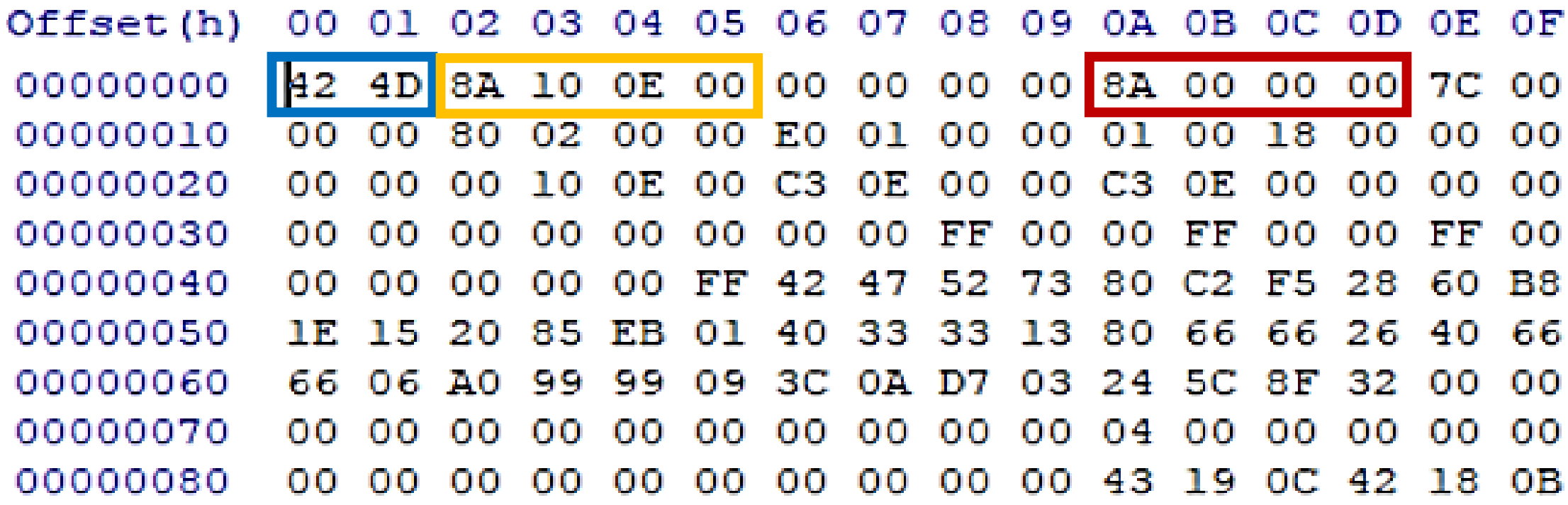


Figure 1. Proposed framework.

03. LSB Steganography 구현

LSB Steganography



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	42	4D	8A	10	0E	00	00	00	00	00	8A	00	00	00	7C	00
00000010	00	00	80	02	00	00	E0	01	00	00	01	00	18	00	00	00
00000020	00	00	00	10	0E	00	C3	0E	00	00	C3	0E	00	00	00	00
00000030	00	00	00	00	00	00	00	00	FF	00	00	FF	00	00	FF	00
00000040	00	00	00	00	00	FF	42	47	52	73	80	C2	F5	28	60	B8
00000050	1E	15	20	85	EB	01	40	33	33	13	80	66	66	26	40	66
00000060	66	06	A0	99	99	09	3C	0A	D7	03	24	5C	8F	32	00	00
00000070	00	00	00	00	00	00	00	00	00	00	04	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	43	19	0C	42	18	0B

파일 타입

bmp → 42 4D

파일 크기

리틀앤디안 → 00 0E 10 8A

헤더 크기

리틀앤디안 → 00 00 00 8A
header size = 138

*리틀앤디안 : 낮은(시작) 주소에 하위 바이트부터 기록

LSB Steganography

moveEndofHeader

*BMP file에서 **header size 정보가 있는 위치를 읽은 후,**
해당 크기만큼 이동해서 **header 뒤로 이동**

hideDatalen

***숨겨야 할 데이터 : 데이터 길이 정보** (for recovery) : 2진수
*1byte씩 읽고, **각 byte의 LSB에 1 또는 0을 삽입**
*최대 32글자 입력 가정 : 256bits : 2^8 : 8번만 반복
*비밀 데이터와 파일에서 읽어온 LSB가
다르면 읽어온 **lsb ^ 0x01 연산 통해 원하는 정보 삽입**
같으면 그대로 사용

hideData

***숨겨야 할 데이터 : 입력한 데이터**
*1byte씩 읽고, 각 byte의 LSB에 1 또는 0을 삽입
*hideDatalen과 동일 but **비밀데이터의 길이 * 8 만큼 반복**
→ 한글자 당 8bits : 3글자면 24bits 필요하고,
한 번 읽어오고 변조할 때 1bit씩 은닉하므로

main

```
fopen("image.bmp", "r+b");
moveEndofHeader (fp,&input_bit,input,inputlengtharr);
hideDatalen (fp,inputlengtharr);
hideData (fp,input_bit,input, dec_data);
decData (fp, dec_inputlength);
```

decData

*BMP file에서 header size 정보가 있는 곳으로 이동하여 읽고,
*해당 크기만큼 이동해서 **header 뒤로 이동**
*해당 위치부터 **8bytes의 LSB를 추출하여 길이 정보 파악**
*해당 위치부터 추출한 **길이 정보/8 만큼 LSB 추출 반복**

LSB Steganography

```
void moveEndofHeader(FILE *fp, int *input_bit, u8 input[32], int inputlengtharr[8]){

        int h;
        u8 header = 0;

        *input_bit = 8 * strlen(input); // 입력값 길이 * 8 --> 총 필요한 바이트 수 : 이진수 변환 : 8bits당 1bit 숨기니까

        // 입력값 길이 2진수로 변환하는 부분 (길이도 삽입할 거라서 제대로 들어가는지 보여주려고)
        printf("\n============== data length ==============\n\ninput bit : %d bits= ", *input_bit);

        for(int i = 7; i >= 0; i--){ // 최상위 비트부터 배열에 순서대로 대입 : >> 7부터 해야 최상위부터 ..
            inputlengtharr[7-i] = (*input_bit >> i) & 0x01;// 해당 자리의 비트와 1과 & -> 1이면 1, 0이면 0
            printf("%d",inputlengtharr[7-i]);
        }


        fseek(fp, 10, SEEK_SET); // header size 읽기 위해 10bytes 이동
        fread(&header, 1, 1, fp); // 해당 위치에 저장된 header size 읽음
        h = ((header >> 4) * 16) + (header & 0x0f); // header size를 10진수로 바꿔줌
        fseek(fp, h, SEEK_SET); // header 뒤쪽으로 이동
}
```

Header size info : fseek(fp, 10, SEEK_SET)

Offset(h)	0A	0B	0C	0D
00000000	8A	00	00	00

header size = 138

LSB Steganography

삽입할 비밀 정보 (data 길이 정보)

```
void hideDatalen(FILE *fp, int inputlengtharr[8]){

u8 buffer1 = 0, buffer2 = 0, temp = 0;

printf("\n\n===== hide data length info to LSB =====\n\n");

    for(int i = 0 ; i < 8 ; ++i){
        fread(&buffer1, 1, 1, fp); // buffer1에 1byte를 1번 읽어옴 : 커서가 1byte 이동
        printf("original data : %02x", buffer1);

        if((buffer1 & 0x01) != inputlengtharr[i]){ // 숨겨야할 데이터와 파일에서 읽어온 LSB가 다르면 ^0x01(원하는 정보 삽입), 같으면 그대로 사용
            temp = buffer1 ^ 0x01; // 연산한 값을 temp에 저장
            fseek(fp, -1, SEEK_CUR); // 읽었던 자리에 LSB 수정하기 위해 움직인 커서를 다시 1byte 앞으로 옮김
            fwrite(&temp, sizeof(u8), 1, fp); // temp에 저장된 값을 1byte로 씀
        }
        // check re-write : 이부분은 사실 필요 없음
        fseek(fp, -1, SEEK_CUR); // 제대로 쓰여졌는지 다시 출력해보기 위해 1byte 앞으로 이동
        fread(&buffer2, 1, 1, fp); // 정보가 써진 부분 읽어옴
        printf("\nmodified data : %02x", buffer2);
        printf("\n=======================================\n");
    } // 헤더 입력 완료
}
```

① offset 138

```
                                                  0A
00000070  00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 43 19 0C 42 18 0B  ..........C..B..
```

u8 buffer1

1byte씩 읽고 파일포인터 이동

② Buffer로 읽어온 각 byte의 LSB값 != 삽입할 data → temp에 저장 후 바꿔씀

buffer1 : $43_{(16)}$

0	1	0	0	0	0	1	1

^ 0x01 → 0x01 → temp

LSB Steganography

```
void hideData(FILE *fp, int *input_bit, u8 input[32], int dec_data[256]){

    u8 buffer1 = 0, buffer2 = 0, temp = 0;
    int data[256];

    for(int i = 0; i < strlen(input) ; ++i){
        for(int j = 0; j < 8 ; ++j){
            data[(8*(i+1)-1) -j] = (input[i] >> j) & 0x01;
        }
    }
    printf("input bit : ", *input_bit);

    printf("\n\n========= hide data to LSB ==========\n\n");

        for(int i = 0 ; i < *input_bit ; ++i ){ // 총 변조에 필요한 비트 수

            fread(&buffer1, 1, 1, fp); // buffer1에 1byte를 1번 읽어옴 : 커서가 1byte 이동

            printf("original data : %02x", buffer1);

            if((buffer1 & 0x01) != data[i]){ // 숨겨야할 데이터와 파일에서 읽어온 LSB가 다르면 ^
                temp = buffer1 ^ 0x01; // 연산한 값을 temp에 저장
                fseek(fp, -1, SEEK_CUR); // 읽었던 자리에 LSB 수정하기 위해 움직인 커서를 다시 1by
                fwrite(&temp, sizeof(u8), 1, fp); // temp에 저장된 값을 1byte로 씀 // 처
            }
            // check data : 이부분은 사실 필요 없음
            fseek(fp, -1, SEEK_CUR); // 제대로 쓰여졌는지 다시 출력해보기 위해 1byte 앞으로 이동
            fread(&buffer2, 1, 1, fp); // 정보가 써진 부분 읽어옴

            dec_data[i] = buffer2;
            printf("\nmodified data: %02x", dec_data[i]);
            printf("\n=======================================\n");
        } // 숨기려고 하는 데이터 입력 완료
}
```

숨겨야 할 데이터 input[i]을 j번 오른쪽 shift 한 후 &0x01

→ 각 자리의 비트를 구해서 data배열에 저장 (동일한 순서 : lsb->lsb)

buffer1을 통해 읽어온 데이터의 lsb와 data배열에 저장된 요소(숨길 데이터)가 다르면 정보 삽입, 같으면 안바꿈

→ 정보를 삽입하는 경우 : buffer와 0x01을 XOR하여 값을 바꿔주고 파일에 씀

② buffer1 : $43_{(16)}$ data[i] : 0

buffer1

0	1	0	0	0	0	1	1	& 0x01

!= 0 → if문 진입

buffer1

0	1	0	0	0	0	1	1	^ 0x01

→ 0 → data[i] 숨김

LSB Steganography

```
void decData(FILE *fp, int dec_inputlength){

    u8 header = 0, buffer1 = 0, temp = 0,  buffer2 = 0, temp2 = 0;
    u8 dec_origindata[32];
    int h;

    printf("\n\n======= recovery data from LSB ========\n");
        fseek(fp, 10, SEEK_SET); // header size 읽기 위해 10bytes 이
        fread(&header, 1, 1, fp); // 해당 위치에 저장된 h
        h = ((header >> 4) * 16) + (header & 0x0f);
        fseek(fp, h, SEEK_SET); // header 뒤쪽으로 이동

        for(int i = 0; i < 8; ++i){
            fread(&buffer1, 1, 1, fp);
            dec_inputlength ^= ((buffer1 & 0x01) << (7-i));
        }
    printf("\n\nextracted data length : %d bits", dec_inputlength);

    for(int i = 0; i < dec_inputlength/8  ; ++i){ // 입력한 문자의 개수 번
        temp2=0; // 문자마다 0으로 다시 초기화
        for(int j = 0; j < 8 ; ++j){ // 8bit씩 이라서 8번 반복
            fread(&buffer2, 1, 1, fp);
            temp2 ^= ((buffer2 & 0x01) << (7-j)); // 0x01과 &연산해서 버
                                                  // 맨 앞부터 채움 -> 0
                                                  // buffe가 011000

        }
        dec_origindata[i] = temp2; // 각 문자의 lsb들을 모은 temp2를 하나의 배
    } // 데이터 추출 완료
     printf("\n\nrecovered data : ");

     for(int i = 0; i < dec_inputlength/8 ; ++i) {// 글자수만큼 출력
         printf("%c", dec_origindata[i]);
     }
         printf("\n=========================================\n\n");
     fclose(fp);
}
```

Bmp는 **Offset 10에 헤더정보**가 있음 → fseek으로 10바이트 이동→ 헤더정보를 읽어서 헤더의 뒤로 이동

해당 위치로부터 8바이트를 읽어서 숨긴 데이터의 길이 추출

→ 파일에서 읽은 값을 & 0x01하여 읽어온 값의 lsb를 추출

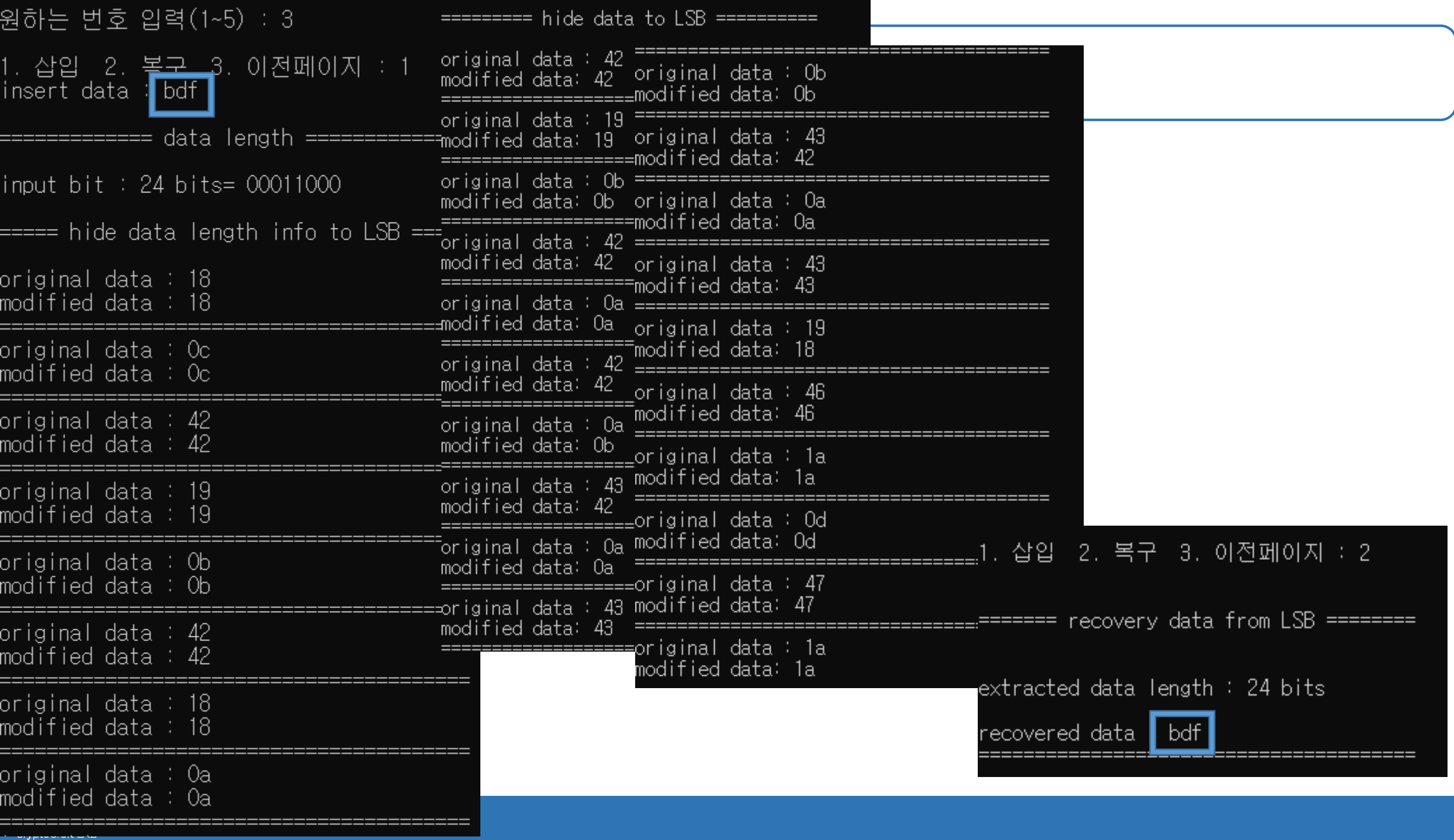
→ **dec_inputlength에 XOR → 8byte의 lsb값들을 모두 더해줌**

각 문자별로 데이터 복구 → **dec_inputlength / 8 (글자수)만큼 반복**

1. 파일에서 읽어온 **1byte마다 &0x01 → lsb 추출**
2. lsb를 7 - j번 left shift해서 **temp와 XOR**

   → temp = 128 + 32 + …

      → XOR로 더해주어서 char형태로 **dec-origindata배열에 저장**


원하는 번호 입력(1~5) : 3
1. 삽입 2. 복구 3. 이전페이지 : 1
insert data : bdf
========== data length ==========
input bit : 24 bits= 00011000
===== hide data length info to LSB ===
original data : 18
modified data : 18
original data : 0c
modified data : 0c
original data : 42
modified data : 42
original data : 19
modified data : 19
original data : 0b
modified data : 0b
original data : 42
modified data : 42
original data : 18
modified data : 18
original data : 0a
modified data : 0a
======== hide data to LSB ==========
original data : 42
modified data: 42
original data : 19
modified data: 19
original data : 0b
modified data: 0b
original data : 42
modified data: 42
original data : 0a
modified data: 0a
original data : 42
modified data: 42
original data : 0a
modified data: 0b
original data : 43
modified data: 42
original data : 0a
modified data: 0a
original data : 43
modified data: 43
original data : 0b
modified data: 0b
original data : 43
modified data: 42
original data : 0a
modified data: 0a
original data : 43
modified data: 43
original data : 19
modified data: 18
original data : 46
modified data: 46
original data : 1a
modified data: 1a
original data : 0d
modified data: 0d
original data : 47
modified data: 47
original data : 1a
modified data: 1a
1. 삽입 2. 복구 3. 이전페이지 : 2
======= recovery data from LSB ========
extracted data length : 24 bits
recovered data bdf

Q & A