1. 이 문제는 1.1 절에서 설명된 내용에 대한 질문이다. 16byte(128bit) 블록 방식의 대칭 암호알고리즘을 이용하여 키 K로 $A||K_{AB}||padding$을 ECB 모드를 사용하여 암호화하였다고 하자. 여기서 A는 20byte, K_{AB}는 16byte이다. 수신자는 이 메시지의 내부 형태를 알고 있다고 가정하고, 채우기는 표준 채우기 이용하였다고 하자. 즉, 채워야 하는 12byte의 각 바이트를 정수 12(0x0C)로 채웠다고 하자. 실제 암호화된 평문을 C/C++의 구조체로 표현하면 다음과 같다.

```
struct Msg{
    char id[20];
    char key[16];
};
```

수신자가 K로 복호화하였을 때 3개의 블록에 대해 무엇을 확신할 수 있는지 설명하시오. 여기서 id에는 널문자로 끝나는 C문자열이 유지된다. 이 문자열의 실제 크기가 답에 어떤 영향을 주는지 여부도 답에 포함해야 하며, 영향을 줄 경우에는 크기가 16보다 작은 경우와 16이상인 경우를 나누어 답하시오.

A. 첫 번째 블록: A의 일부분과 KAB의 일부분이 포함되어 있으므로, A의 일부분과 KAB의 일부분을 확신할 수 있습니다. 그러나, 이 블록에는 padding이 포함되어 있지 않으므로, padding의 내용에 대해서는 확신할 수 없습니다.

두 번째 블록: KAB의 나머지 부분과 padding이 포함되어 있으므로, KAB의 나머지 부분과 padding의 내용을 확신할 수 있습니다.

세 번째 블록: 모두 padding으로 채워졌으므로, padding의 내용을 확신할 수 있습니다.

만약 id의 길이가 16보다 작다면, 첫 번째 블록에서 남은 바이트는 0x0C로 채워져있을것이다. 따라서 수신자는 첫 번째 블록에서 id의 길이를 실제 파악을 할 수 있다. 하지만 id의 길이가 16바이트 이상이라면, 첫 번째 블록에서 id의 끝에 0x0C가 포함되어 있을 수 있다. 따라서 이 경우, 첫 번째 블록만으로 id의 실제 길이를 정확히 파악할 수 없다. 따라서 수신자는 두 번째 블록에서 padding을 제외한 나머지 내용을 복호화하여 메시지의 내용을 파악하고 길이를 계산할 수 있다.

---

8. 16비트 카운터(unsigned)를 사용할 때 1초에 1 증가한다고 가정하자. 그러면 이 카운터의 값이 순환되어 0이 될 때까지 소요되는 시간을 계산하시오. 32비트를 사용할 경우에도 동일한 시간을 계산하시오. 실제 컴퓨터 프로그래밍을 이용하여 소요되는 시간을 년, 일, 시간, 분, 초로 제시하시오.

A. 16비트의 경우 최대값 65535에 도달하기까지 걸리는 시간은 [시간 = (최대값 + 1) / 증가량 = 65536 / 1 = 65536초]이다. 32비트의 경우 최대값 429496725에 도달하기까지 걸리는 시간은 [시간 = (최대값 + 1) / 증가량 = 429496726 / 1 = 429496726초]이다.

노트북 프로세서의 클럭이 1.7GHz이므로 이를 바탕으로 c++코드를 작성해보면 다음과 같다.

```c++
// 16비트의 경우
#include <iostream>
using namespace std;

int main() {
    unsigned int counter = 0;
```

```
    double clockSpeed = 1.7e9; // 1.7GHz

    while (counter < 65535) {
        counter++;
    }

    double timeTaken = counter / clockSpeed;
    int year = (int)(timeTaken / 31536000); // 1년 = 31536000초
    int day = (int)((timeTaken - year * 31536000) / 86400); // 1일 = 86400초
    int hour = (int)((timeTaken - year * 31536000 - day * 86400) / 3600); // 1시간 = 3600초
    int minute = (int)((timeTaken - year * 31536000 - day * 86400 - hour * 3600) / 60); // 1분 = 60초
    int second = (int)(timeTaken - year * 31536000 - day * 86400 - hour * 3600 - minute * 60);

    cout << "총 소요 시간: " << year << "년 " << day << "일 " << hour << "시간 " << minute << "분 " << second << "초" << endl;

    return 0;
}
```

```c++
// 32비트의 경우
#include <iostream>
using namespace std;

int main() {
    unsigned int counter = 0;
    double clockSpeed = 1.7e9; // 1.7GHz

    while (counter < 4294967295) {
        counter++;
    }

    double timeTaken = counter / clockSpeed;
    int year = (int)(timeTaken / 31536000); // 1년 = 31536000초
    int day = (int)((timeTaken - year * 31536000) / 86400); // 1일 = 86400초
    int hour = (int)((timeTaken - year * 31536000 - day * 86400) / 3600); // 1시간 = 3600초
```

```
    int minute = (int)((timeTaken - year * 31536000 - day * 86400 - hour * 3600) / 60); // 1분 = 60초
    int second = (int)(timeTaken - year * 31536000 - day * 86400 - hour * 3600 - minute * 60);

    cout << "총 소요 시간: " << year << "년 " << day << "일 " << hour << "시간 " << minute << "분 " << second << "초" << endl;

    return 0;
}
```

---

13. 다음과 같은 인증 프로토콜이 있다고 가정하자.

$$\begin{aligned}&\text{Msg 1. } A \rightarrow B: \quad A, N_A\\&\text{Msg 2. } B \rightarrow A: \quad B, N_B, \text{Sig.}\text{–}K_B(N_B||N_A)\\&\text{Msg 3. } A \rightarrow B: \quad \text{Sig.}\text{–}K_A(N_B)\end{aligned}$$

이 프로토콜은 강한 개체 인증을 제공하지 못한다. 강한 개체 인증을 제공하도록 수정하시오.

A.

Msg 1. A -> B: A, sig.A(+KA), NA

Msg 2. B -> A: sig.B(+KB), NB, sig.-KB(NB||sig.A(+KA)||NA)

Msg 3. A -> B: sig._KA(NB||sig.B(+KB))

---

14. 문제 13에서 강한 개체 인증을 제공하도록 수정한 프로토콜에 대해 프로토콜의 시작자로써 A가 C와 하는 프로토콜을 작성하고, A 입장에서 두 프로토콜의 수행이 구분할 수 있는지 설명하시오. 또 프로토콜의 반응자로써 C와 B가 하는 프로토콜을 작성하고, B 입장에서 두 프로토콜의 수행이 구분할 수 있는지 설명하시오.

A.

C와 하는 프로토콜:

Msg 1. A -> C: A, sig.A(+KA), NA

Msg 2. C -> A: sig.C(+KC), NC, sig.-KC(NC||sig.A(+KA)||NA)

Msg 3. A -> C: sig._KA(NC||sig.C(+KC))

A 입장에서 두 프로토콜의 수행은 구분할 수 있다. 첫 번째 프로토콜에서는 메시지 1을 C에게 보내고, 두 번째 프로토콜에서는 메시지 1을 B에게 보내기 때문이다.

C와 B가 하는 프로토콜:

Msg 1. C -> B: A, sig.A(+KA), NA

Msg 2. B -> C: sig.B(+KB), NB, sig.-KB(NB||sig.A(+KA)||NA)

Msg 3. C -> B: sig._KC(NB||sig.B(+KB))

B 입장에서 두 프로토콜의 수행은 구분할 수 있다. 첫 번째 프로토콜에서는 메시지 1을 A에게 보내고, 두 번째 프로토콜에서는 메시지 1을 C에게 보내기 때문이다.