제 3 장 대칭 암호알고리즘: 암호화 모드

3.1 암호화 모드 개요

대칭 암호알고리즘은 크게 블록 암호방식과 스트림 암호방식으로 분류된다. 블록 암호방식은 메시지를 일정한 크기로 나누어 각 블록을 암호화하는 방식이고, 스트림 암호방식은 한 번에 한 비트 또는 한 바이트를 암호화하는 방식이다. 2장에서 살펴본 단순치환 암호방식은 한 문자 단위로 암호화하기 때문에 스트림 암호방식이고, 컴퓨터에서 사용하는 자리바꿈 맵을 이용한 자리바꿈 암호방식은 어떤 블록 단위로 동일 맵을 반복적으로 적용하기 때문에 블록 암호방식이다. 블록 암호방식의 한 가지 문제점은 같은 평문 블록은 항상 같은 암호문 블록으로 암호화된다는 것이다. 이것을 극복하기 위해 등장한 기술이 **암호화 모드** (cryptographic mode)이다. 반면에 스트림 암호방식에서 각 바이트 또는 비트는 암호화될 때마다 보통 항상 다른 바이트 또는 비트로 암호화된다. 물론 2장에서 살펴본 단순치환이나 다중치환 암호방식은 이와 같은 특성을 가지지 않거나 주기적으로 반복된다. 하지만 현대 스트림 암호방식은 영구적으로 이와 같은 특성을 가지지는 못하지만 안전성에 문제가 될 정도로 주기적인 반복이 일어나지는 않는다.

블록 암호방식의 문제점을 극복하기 위한 방법으로 사용되는 암호화 모드는 모드를 도입하여 암호화하더라도 기존 알고리즘의 성능에 큰 영향을 주지 않아야 한다. 따라서 보통 XOR 연산과 피드백을 활용한다. 여기서 피드백이라는 것은 한 블록을 독립적으로 암호화하지 않고, 다른 블록이나 다른 블록의 암호화한 결과를 활용하는 것을 말한다.

3.2 ECB 모드

ECB(Electronic CodeBook) 모드는 동일 블록은 동일 암호문으로 암호화된다는 문제점을 극복할 수 있는 모드는 아니다. 보다 정확하게 말하면 ECB 모드는 암호화 모드를 사용하지 않는 모드이며, 피드백을 사용하지 않고, 각 블록을 독립적으로 암호화하는 모드이다. ECB 암호화 모드는 다음과 같이 나타낼 수 있다.

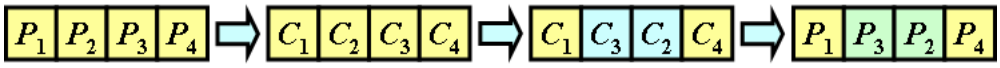
- 암호화: $C_i = E_K(P_i)$
- 복호화: $P_i = D_K(C_i)$

여기서 P_i는 i번째 평문 블록을 말한다. 암호화 모드의 특성을 논할 때 항상 다음 사항들을 분석한다.

- 첫째, 암호화된 암호문에서 두 암호블록을 바꾼 경우에 복호화 결과
- 둘째, 암호화하기 전에 P_i에 오류가 발생하였을 때 그것의 파급 효과
- 셋째, 암호화한 후에 C_i에 오류가 발생하였을 때 그것의 파급 효과
- 넷째, 암호화 모드 사용에 따른 보안 문제의 유무

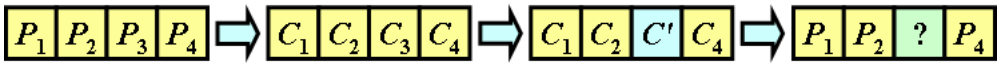
ECB 모드에 대해 위 네 가지에 대해 분석하여 보자.

- 암호화된 암호문에서 두 암호블록을 바꾼 경우에 복호화 결과: 그림 3.1처럼 암호화된 메시지의 블록들의 위치를 바꾸면 복호화된 평문 메시지의 블록 위치들도 동일하게 바꾸게 된다. 이것은 각 블록이 독립적으로 암호화되기 때문이다.



<그림 3.1> ECB 모드에서 두 암호블록을 바뀐 상태로 복호화한 경우

- 암호화하기 전에 P_i에 오류가 발생한 경우: 각 평문 블록은 독립적으로 암호화되기 때문에 한 평문 블록에 오류가 있는 상태로 암호화되어도 그것은 오직 해당 암호문 블록에만 영향을 준다.
- 암호화한 후에 C_i에 오류가 발생한 경우: 각 암호문 블록은 독립적으로 복호화되기 때문에 한 암호문 블록에 오류가 발생한 상태로 복호화하면 그림 3.2처럼 그 블록의 복호화에만 영향을 준다.



<그림 3.2> ECB 모드에서 한 암호문에 오류가 발생한 경우

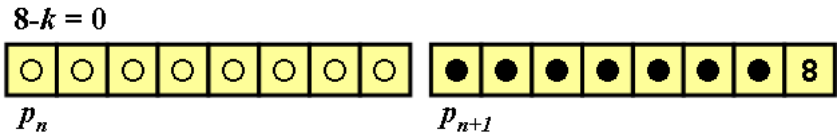
ECB 모드는 특별한 피드백을 적용하지 않기 때문에 모드 사용에 따른 추가적인 보안 문제는 없다. 다만. 피드백을 사용하지 않기 때문에 같은 평문 블록은 같은 암호문 블록으로 암호화된다는 문제점을 지니고 있다.

3.3 채우기

피드백을 사용하는 암호화 모드를 설명하기 전에 먼저 블록 암호방식에서 꼭 필요한 또 다른 요소인 **채우기**(padding)에 대해 알아볼 필요가 있다. 평문 메시지의 크기가 정확하게 블록 크기의 배수가 아닌 경우에는 완전하지 않은 마지막 블록을 완전한 블록으로 만들어 암호화하여야 한다. 완전하지 않은 마지막 블록 끝에 규칙적인 일련의 비트를 추가하여 완전한 블록을 만드는 것을 채우기라 한다. 채우기의 한 가지 요구사항은 나중에 암호문을 복호화한 사용자가 채우기를 한 부분을 정확하게 제외시킬 수 있어야 한다. 이를 위해 보통 두 가지 방법을 사용한다. 하나는 암호문과 별도로 평문의 길이를 따로 보내는 것이다. 다른 하나는 채우기를 한 부분에 채운 부분을 제거할 수 있는 요소를 포함하는 것이다. 후자의 방법은 채우기 끝에 채운 크기를 기록하거나 끝 표시 문자 등 다양한 방법이 사용된다. 후자의 방법 중에는 평문의 크기가 정확하게 블록 크기의 배수이어도 채우기를 해야 하는 경우가 있다. 이것은 원래 평문을 채우기로 생각하여 제거할 수 있기 때문이다.

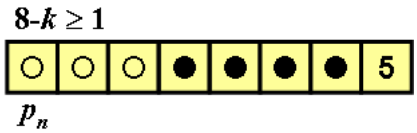
블록 크기가 8바이트이고 마지막 평문 블록의 크기가 k일 때 채우기 끝에 채운 크기를 기록하는 방법으로 채우기를 하는 방법은 다음과 같다. 그림 3.3부터 3.6에서 빈 원으로 표시된 바이트는 원래 평문이고, 검은색으로 채워진 원으로 표시된 바이트는 채우기를 한 바이트이다.

- $8-k=0$: 평문의 크기가 정확하게 블록 크기의 배수인 경우



<그림 3.3> 평문의 크기가 정확하게 블록 크기의 배수일 때 채운 크기를 기록하는 채우기 방법

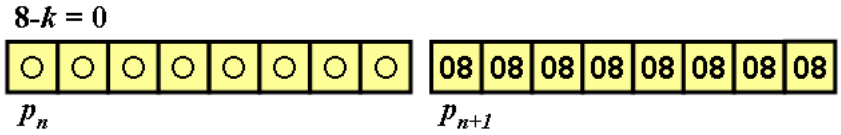
- $8-k \geq 1$: 평문의 크기가 블록 크기의 배수가 아닌 경우



<그림 3.4> 평문의 크기가 블록 크기의 배수가 아닐 때 채운 크기를 기록하는 채우기 방법

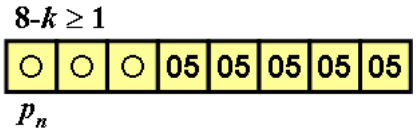
PKCS #5, PKCS #7, RFC3369 표준에 제시된 채우기 방법은 그림 3.3과 그림 34에 제시된 방법과 유사하다. 다만, 채워지는 바이트들을 마지막 블록에 기록한 크기와 동일한 값으로 채운다.

- $8-k=0$: 평문의 크기가 정확하게 블록 크기의 배수인 경우



<그림 3.5> 평문의 크기가 정확하게 블록 크기의 배수일 때 표준에 제시된 채우기 방법

- $8-k \geq 1$: 평문의 크기가 블록 크기의 배수가 아닌 경우



<그림 3.6> 평문의 크기가 정확하게 블록 크기의 배수일 때 표준에 제시된 채우기 방법

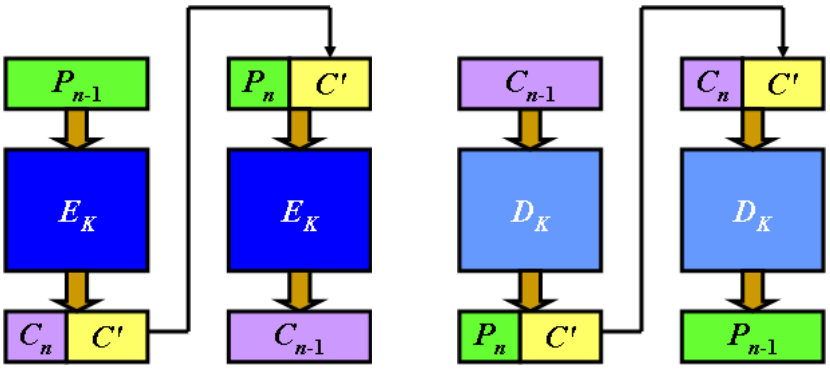
이 외에도 다양한 채우기 방법이 있지만 모두 원래의 평문 크기보다 암호문의 크기가 커지는 단점을 가지고 있다. 특히, 앞서 살펴본 두 방법은 평문의 크기가 블록의 크기의 정확한 배수이어도 채우기를 해야 하는 단점을 지니고 있다. 이와 같은 모든 문제를 해결할 수 있는 방법이 **암호문 훔침**(ciphertext stealing) 기법이다.

암호문 훔침 기법은 바로 전 암호문을 블록을 이용하여 채우기를 하는 방식으로 암호문의 크기가 평문의 크기와 같아진다는 장점을 지니고 있다. 암호문 훔침 기법에서 암호화와 복호화는 그림 3,7과 같으며, 식으로 표현하면 다음과 같다.

- 암호화: $E_K(P_{n-1}) = C_n \| C'$, $E_K(P_n \| C') = C_{n-1}$

- 복호화: $D_K(C_{n-1}) = P_n \| C'$, $D_K(C_n \| C') = P_{n-1}$

암호문을 복호화해야 하는 사용자는 암호문의 크기가 블록의 배수가 아니면 끝에서 두 번째 암호문 블록을 복호화한 다음에 마지막 암호문 블록의 크기만큼을 P_n으로 간주하고, 남은 부분을 마지막 블록과 결합하여 복호화하여 P_{n-1}를 얻는다.



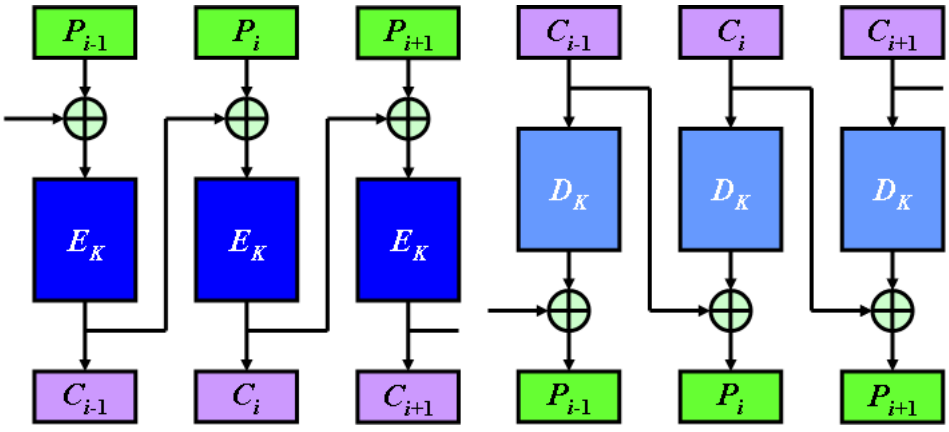
(ㄱ) 암호문 훔침 기법을 이용한 암호화 (ㄴ) 암호문 훔침 기법을 이용한 복호화

<그림 3.7> 암호문 훔침 기법

3.4 CBC 모드

CBC(Cipher Block Chaining) 모드는 이전 블록의 암호문을 현재 평문 블록을 암호화할 때 사용하는 모드이다. 따라서 같은 평문 블록들도 서로 다른 암호문 블록으로 암호화된다. 그림 3.8에 기술된 CBC 모드를 식으로 표현하면 다음과 같다.

- 암호화: $E_K(P_i \oplus C_{i-1}) = C_i$
- 복호화: $D_K(C_i) \oplus C_{i-1} = P_i$



(ㄱ) CBC 모드를 이용한 암호화 (ㄴ) CBC 모드를 이용한 복호화

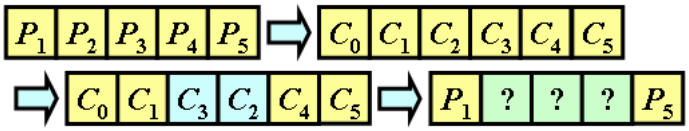
<그림 3.8> CBC 모드

평문을 암호화할 때 이전 암호문 블록을 이용하기 때문에 첫 평문 블록은 이용할 수 있는 암호문 블록이 없다. 따라서 랜덤 블록을 하나 생성하여 이를 이용한다. 복호화하는 측에서도 첫 번째 평문 블록을 얻기 위해서는 사용된 랜덤 블록이 필요하다. 따라서 이 블록을 암호문의 첫 블록을 사용하며, 이 블록을 **초기화 벡터**(IV, Initialization Vector)라 한다. 이 벡터를 사용하므로 같은 메시지를 다시 암호화하여도 IV가 다르면 결과가 다르게 된다. 하지만 초기 벡터가 공개된다고 하여 공격자들이 첫 평문 블록을 복호화할 수 있는 것은 아니므로 비밀성을 유지할 필요는 없다. 하지만 IV가 사용되므로 암호문의 크기는 평문의 크기보다 한 블록이 증가하게 된다. 또 채우기까지 고려할 경우 최대 두 블록이 커질 수 있다. 물론 CBC 모드에서도 다음과 같이 암호문 훔침 기법을 활용할 수 있다.

- CBC 모드에서 암호문 훔침 기법을 이용한 암호화:
$$E_K(P_{n-1} \oplus C_{n-2}) = C_n \| C',\ E_K(\{P_n \| 0\} \oplus \{C_n \| C'\}) = C_{n-1}$$
- CBC 모드에서 암호문 훔침 기법을 이용한 복호화:
$$D_K(C_{n-1}) \oplus \{C_n \| 0\} = P_n \| C',\ D_K(C_n \| C') \oplus C_{n-2} = P_{n-1}$$

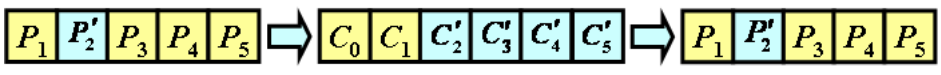
CBC를 이용한 암호화의 특성은 다음과 같다.

- 암호화된 암호문에서 두 암호블록을 바꾼 경우에 복호화 결과: 그림 3.9처럼 암호화된 메시지의 블록들의 위치를 바꾸면 3개의 평문 블록에 영향을 준다. CBC 모드에서는 한 블록에 오류가 발생하면 두 평문 블록에 영향을 주기 때문에 연속된 두 개 암호문 블록에 오류가 발생한 것과 동일하므로 세 평문 블록에 영향을 준다.



<그림 3.9> CBC 모드에서 인접한 두 암호블록을 바뀐 상태로 복호화한 경우

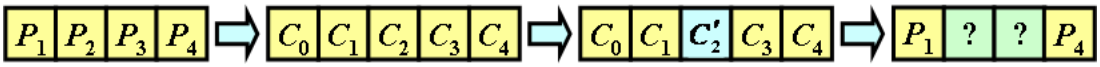
- 암호화하기 전에 P_i에 오류가 발생한 경우: 암호화하기 전에 P_i에 오류가 발생하면 C_i에 영향을 줄 뿐만 아니라 C_i 이후 모든 암호문 블록에 영향을 준다. 하지만 복호화하면 오직 P_i에만 영향을 준다. 이것은 당연한 현상이다. 발생된 오류가 실제 오류가 아니라 원래 평문이 그와 같은 형태이었다고 생각하면 쉽게 이해할 수 있다. 여기서 우리가 주목해야 할 것은 복호화하였을 때의 결과가 아니라 평문의 일부가 바뀌었을 때 그 평문 블록 이후 모든 암호문 블록이 변한다는 것이다. 따라서 CBC 모드로 암호화된 암호문 블록의 마지막 블록은 그 평문을 대표하는 블록으로 사용될 수 있다. 즉, 마지막 블록을 MAC 값으로 사용할 수 있다.



<그림 3.10> CBC 모드에서 암호화하기 전에 평문 블록에 오류가 있는 상태로 암호화된 경우

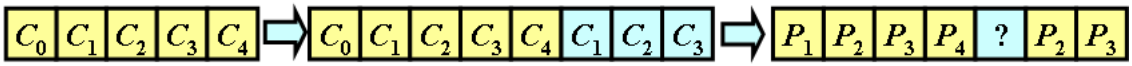
- 암호화한 후에 C_i에 오류가 발생한 경우: 복호화하면 P_i와 P_{i+1}에만 영향을 준다. C_2에 오류가 발생하였다고 가정하고 이것을 수식으로 살펴보면 다음과 같다.
  - $P_2' = D_K(C_2') \oplus C_1 = ? \oplus C_1 = ?$
  - $P_3' = D_K(C_3) \oplus C_2' = P_3 \oplus C_2 \oplus C_2' = ?$
  - $P_4' = D_K(C_4) \oplus C_3 = P_4 \oplus C_3 \oplus C_3 = P_4$

  즉, C_2에 오류가 발생하면 P_2와 P_3 복호화에 영향을 준다. 보다 자세히 살펴보면 P_2에 발생되는 오류는 예측이 가능하지 않지만 P_3는 예측이 가능한 오류가 발생한다. 특히 C_2의 한 비트를 바꾸면 P_3에 동일한 위치에 한 비트만 오류가 발생한다.



<그림 3.11> CBC 모드에서 한 암호문에 오류가 발생한 경우

- 보안 문제: CBC 모드를 사용할 때 추가되는 보안 문제는 크게 다음과 같은 네 가지가 있다.
  - 문제 1. 암호화된 메시지 끝에 임의의 블록을 추가할 수 있다. 예를 들어 그림 3.12처럼 같은 키로 암호화된 암호블록을 추가하면 추가된 블록 중 첫 블록만 제대로 복호화되지 않고 나머지는 모두 제대로 복호화된다.



<그림 3.12> CBC 모드에서는 암호문 끝에 임의의 블록을 추가가 가능하다는 문제점

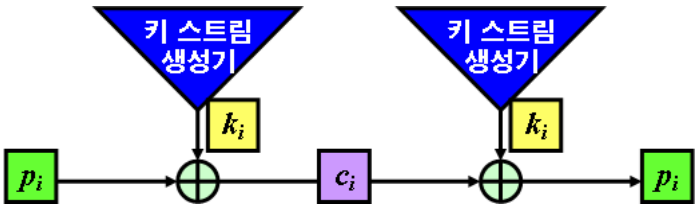
  - 문제 2. 암호문 블록을 변경하여 예측할 수 있는 변경을 도입할 수 있다. 즉, C_i의 한 비트를 변경하면 P_{i+1}은 같은 위치 비트만 변경된다. 물론 이 때 P_i는 예측 가능하지 않은 오류가 발생한다.
  - 문제 3. 평문의 패턴은 CBC 모드를 통해 은닉되지만 영구적인 것은 아니다. 즉, 메시지가 매우 길어질 경우에는 패턴이 다시 등장할 수 있다.
  - 문제 4. 임의의 두 암호문 블록 C_i와 C_j가 같으면 다음이 성립한다.

$$C_{i-1} \oplus C_{j-1} = P_i \oplus P_j$$

3.5 스트림 암호방식

스트림 암호방식은 평문의 비트 또는 바이트를 하나씩 암호화하는 방식이다. 스트림 암호방식을 위해 전용 스트림 암호알고리즘을 개발할 수 있지만 일반 블록 암호알고리즘을 이용하여 스트림 암호방식으로 암호화할 수 있다. 스트림 암호방식의 핵심 요소는 **키 스트림 생성기**(key stream generator)이며, 암호화하는 사용자와 복호화하는 사용자는 동일한 키 스트림 생성기를 가지고 있어야 한다. 이 생성기는 평문을 암호화/복호화하기 위해 필요한 비트 또는 바이트를 지속적으로 생성한다. 보통 그림 3.13과 같이 평문과 이 생성기에서 생성한 비트 또는 바이트를 XOR 연산하여 암호화한다. 따라서 이 방식의 안전성은 키 스트림 생성

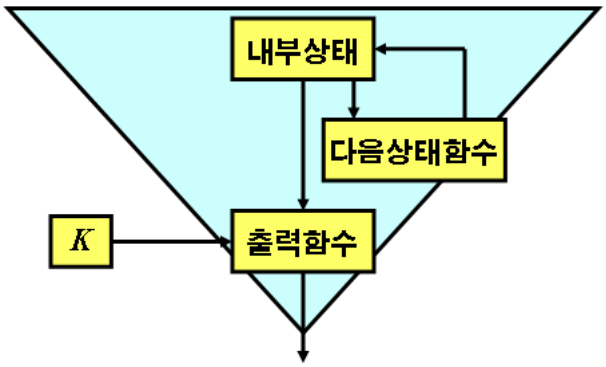
기의 출력에 의해 결정된다.



<그림 3.13> 키 스트림 암호방식의 기본 구조

이와 같은 방식으로 암호화하므로 공격자가 평문과 그것에 해당하는 암호문을 얻으면 암호화 과정에서 사용된 키 스트림을 얻을 수 있다. 또 같은 키 스트림으로 암호화된 두 개의 암호문을 가지고 있으면 이들을 서로 XOR하여 두 개의 평문을 XOR한 데이터를 얻을 수 있다. 따라서 키 스트림은 매번 다른 것을 사용해야 한다. 또 스트림 암호방식도 암호키를 사용한다. 암호키를 사용할 경우 생성되는 키 스트림은 암호키에 의해 결정된다. 따라서 키 스트림의 안전성이 떨어질 때 암호키만 바꾸면 새로운 안전한 키 스트림을 얻을 수 있다.

3.5.1 키 스트림 생성기



<그림 3.14> 키 스트림 생성기의 구성 요소

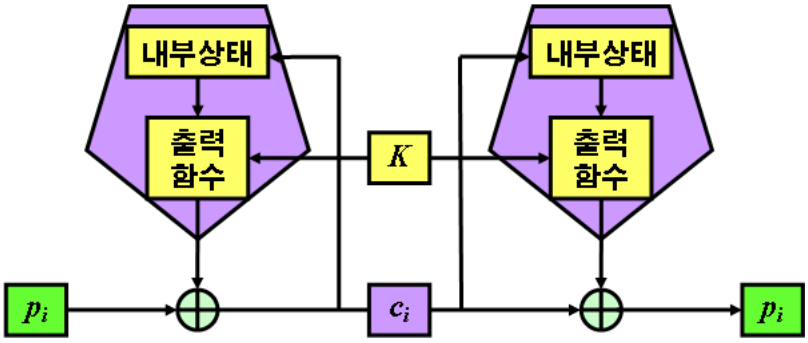
키 스트림 생성기는 그림 3.14에 기술된 것과 같이 보통 다음과 같은 세 가지 구성요소를 가진다.

- **내부 상태**(internal state): 키 스트림 생성기의 현재 상태
- **다음 상태 함수**(next-state function): 내부 상태를 입력으로 받아 새로운 상태를 출력하여 주는 함수
- **출력 함수**(output function): 키와 내부 상태를 입력으로 받아 키 스트림 바이트(비트)를 생성하여 주는 함수

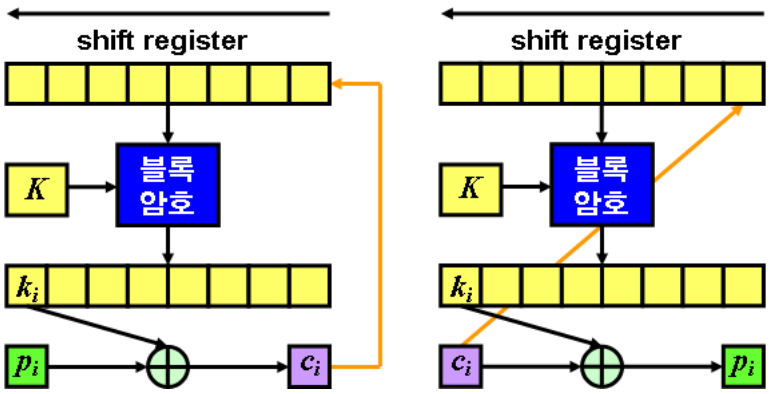
3.5.2 자체 동기화 스트림 암호방식

키 스트림 생성기의 구성 요소 중에 다음 상태 함수는 외부 값에 의해 결정될 수 있다.

가장 대표적인 키 스트림 생성 방식인 **자체 동기화 스트림 방식**은 그림 3.15와 같이 출력되는 키 스트림 바이트(비트)가 고정된 개수의 이전 암호문 바이트(비트)에 의해 결정된다. 이와 같은 방식을 사용할 경우 암호문을 구성하는 한 바이트에 오류가 발생하더라도 그 바이트가 더 이상 내부 상태에 영향을 주지 않는 상황이 되면 자동적으로 다시 암호화 측의 키 스트림과 복호화 측의 키 스트림이 동기화된다. 따라서 이 방식을 자체 동기화 스트림 암호방식이라 한다.



<그림 3.15> 자체 동기화 스트림 암호방식



<그림 3.16> 블록방식의 암호알고리즘을 이용한 자체 동기화 스트림 암호방식

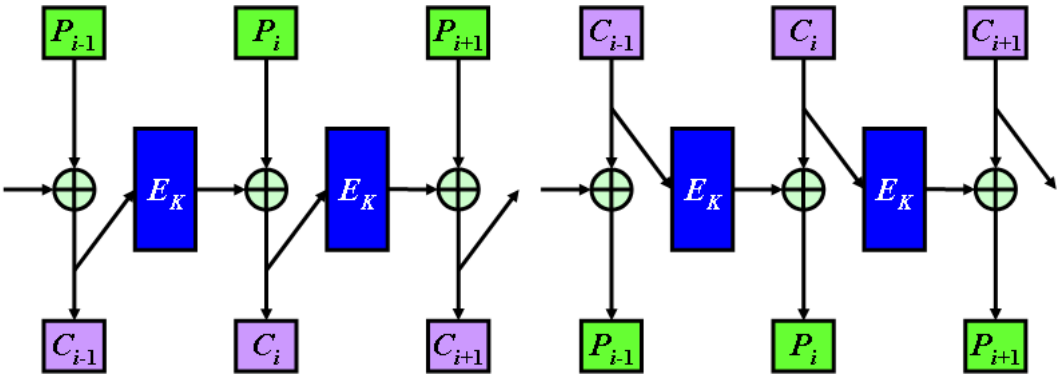
다시 말하면 내부 상태가 이전 암호문의 n 바이트에 의존하면 의미 없는 데이터를 n 바이트 전송하여 상호 키 스트림을 다시 동기화할 수 있다. 같은 이유에서는 어떤 암호문 바이트가 전달되는 과정에서 변경되었으면 총 $n+1$ 평문 바이트에 영향을 준다. 자체 동기화 스트림 암호 방식의 한 가지 문제점은 전달되는 암호문의 일부를 보관한 후에 나중에 재전송하면 처음 동기화하는 동안은 쓸모없는 데이터를 얻게 되지만 다음 상태 함수에 따라 정해진 수의 바이트가 지나면 올바르게 복호화된다.

일반 블록방식의 암호알고리즘을 키 스트림 생성기 내에 출력함수로 사용하면 블록 암호방식을 스트림 암호방식으로 전환할 수 있다. 예를 들어 그림 3.16은 64비트 블록 암호방식을 이용한 8비트 자체 동기화 스트림 암호방식을 보여주고 있다. 이와 같은 방식의 한 가지 문제점은 64비트 블록을 암호화하는데 총 8번의 64비트 블록 암호화를 수행한다는 것이다.

3.6 CFB 모드

CFB(Ciphertext Feedback) 모드는 앞서 살펴본 자체 동기화 스트림 암호방식을 이용하는 암호화 모드이다. 따라서 블록보다 작은 단위로 암호화가 가능하며, 복호화 연산 없이 암호화 연산과 XOR 연산만 사용한다. 이 때문에 구현이 효율적이다. CFB 모드를 수식으로 표현하면 다음과 같다.

- 암호화: $P_i \oplus E_K(C_{i-1}) = C_i$
- 복호화: $C_i \oplus E_K(C_{i-1}) = P_i$



(ㄱ) CFB 모드를 이용한 암호화 (ㄴ) CFB 모드를 이용한 복호화

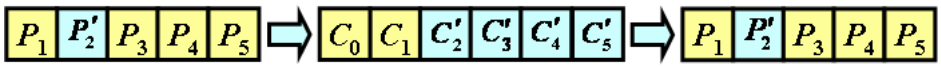
<그림 3.16> CFB 모드

CFB 모드도 CBC 모드와 마찬가지로 초기 벡터가 필요하다. 하지만 CBC와 달리 동일한 IV를 두 번 사용하면 다음과 같은 이유에서 심각한 보안 문제를 초래할 수 있다.

$$C_1 = P_1 \oplus E_K(C_0),\ C_1' = P_1' \oplus E_K(C_0)$$
$$C_1 \oplus C_1' = P_1 \oplus P_1'$$

즉, 동일한 IV를 사용하는 두 암호문의 첫 블록들을 XOR하면 두 평문 블록을 XOR한 값을 얻을 수 있다. 따라서 CFB 모드에서는 반드시 매 번 다른 IV를 사용해야 한다. 또 CFB 모드를 사용할 경우에는 블록 크기보다 작은 크기로 암호화할 수 있으며, 이 때문에 채우기가 필요 없다. CFB 모드의 그 밖의 특성은 다음과 같다.

- 암호화된 암호문에서 두 암호블록을 바꾼 경우에 복호화 결과: CBC 모드와 마찬가지로 세 블록 복호화에 영향을 준다.
- 암호화하기 전에 P_i에 오류가 발생한 경우: CBC 마찬가지로 암호화하기 전에 P_i에 오류가 발생하면 C_i에 영향을 줄 뿐만 아니라 C_i 이후 모든 암호문 블록에 영향을 준다. 따라서 CFB 모드로 암호화된 암호문의 마지막 블록도 MAC 값으로 사용할 수 있다.



<그림 3.17> CBC 모드에서 암호화하기 전에 평문 블록에 오류가 있는 상태로 암호화된 경우

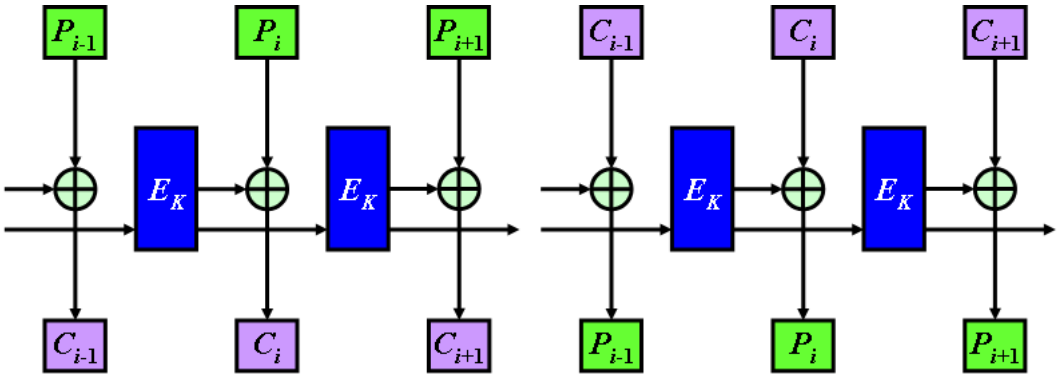
- 암호화한 후에 C_i에 오류가 발생한 경우: 복호화하면 P_i와 P_{i+1}에만 영향을 준다. C_2에 오류가 발생하였다고 가정하고 이것을 수식으로 살펴보면 다음과 같다.
  - $P_2' = C_2' \oplus E_K(C_1) = ? \oplus E_K(C_1) = ?$
  - $P_3' = C_3 \oplus E_K(C_2') = P_3 \oplus E_K(C_2) \oplus E_K(C_2') = ?$
  - $P_4' = C_4 \oplus E_K(C_3) = P_4 \oplus E_K(C_3) \oplus E_K(C_3) = P_4$

  즉, C_2에 오류가 발생하면 P_2와 P_3 복호화에 영향을 준다. 보다 자세히 살펴보면 P_2는 예측이 가능한 오류가 발생되며, P_3는 예측이 가능하지 않는 오류가 발생한다. 특히 C_2의 한 비트를 바꾸면 P_2에 동일한 위치에 한 비트만 오류가 발생한다.

3.7 OFB 모드

OFB(Output Feedback) 모드는 CFB 모드와 마찬가지로 암호화 연산만 사용한다. 실제로는 꼭 암호화 연산을 사용할 필요는 없고, 암호화 연산 대신에 MAC을 사용할 수도 있다. OFB를 수식으로 표현하면 다음과 같다.

- 암호화: $P_i \oplus S_i = C_i,\ S_i = E_K(S_{i-1})$
- 복호화: $C_i \oplus S_i = P_i,\ S_i = E_K(S_{i-1})$



(ㄱ) OFB 모드를 이용한 암호화 (ㄴ) OFB 모드를 이용한 복호화

<그림 3.18> OFB 모드

OFB 모드도 CFB 모드와 마찬가지로 블록 단위보다 작은 단위로 암호화할 수 있으므로 채우기가 필요 없다. 이 방식에서도 초기 벡터가 필요하며, CFB와 마찬가지로 항상 다른 초기 벡터를 사용해야 한다. CFB 방식에서는 동일한 초기 벡터를 사용하면 첫 블록에 대해서는 두 평문 블록을 XOR한 값을 얻을 수 있었지만 OFB 방식에서는 동일한 초기 벡터를 사용하면 모든 블록에 대해 두 평문 블록을 XOR한 값을 얻을 수 있다. 따라서 OFB 방식에서

는 동일한 초기 벡터를 사용하면 더 심각한 보안 문제를 야기 시킬 수 있다. OFB의 그 밖의 특성은 다음과 같다.

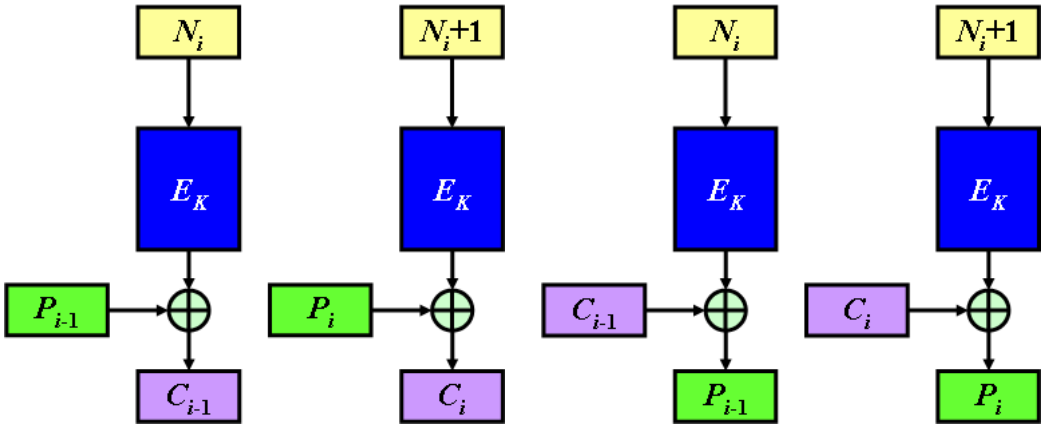
- 암호화된 암호문에서 두 암호블록을 바꾼 경우에 복호화 결과: 두 블록의 복호화에만 영향을 준다.
- 암호화하기 전에 P_i에 오류가 발생한 경우: P_i에 오류가 발생하면 C_i에만 영향을 준다. 따라서 OFB 모드로 암호화된 암호문의 마지막 블록은 MAC 값으로 사용할 수 없다.
- 암호화한 후에 C_i에 오류가 발생한 경우: 복호화하면 P_i에만 영향을 준다. C_2에 오류가 발생하였다고 가정하고 이것을 수식으로 살펴보면 다음과 같다.
  - $P_2' = C_2' \oplus E_K(S_1) = ? \oplus E_K(S_1) = ?$
  - $P_3' = C_3 \oplus S_3 = P_3 \oplus E_K(S_2) \oplus E_K(S_2) = P_3$

  즉, C_2에 오류가 발생하면 P_2에만 영향을 주며, P_2에는 예측이 가능한 오류가 발생한다.

3.8 카운터 모드

이 모드는 최근에 많이 사용되는 모드로 CFB, OFB와 마찬가지로 암호화 연산만 사용한다. 이 모드에서는 각 평문을 암호화할 때 OFB와 마찬가지로 평문과 독립적으로 생성된 랜덤 블록과 XOR하여 암호화한다. 차이점은 OFB 모드에서는 한 블록에 사용된 랜덤 블록이 다음 랜덤 블록을 생성할 때 사용되는 반면에 카운터 모드에서는 카운터 값을 정해진 증가 함수를 이용하여 증가시켜 이것을 입력으로 사용하여 랜덤 블록을 생성한다. 즉, S_i를 계산하는 방법을 제외하고는 동일하다. 카운터 모드를 수식으로 나타내면 다음과 같다. 여기서 N_i는 해당 평문 블록을 암호화할 때 사용되는 카운터 값이다.

- 암호화: $P_i \oplus S_i = C_i,\ S_i = E_K(N_i)$
- 복호화: $C_i \oplus S_i = P_i,\ S_i = E_K(N_i)$



(ㄱ) 카운터 모드를 이용한 암호화 (ㄴ) 카운터 모드를 이용한 복호화

<그림 3.19> 카운터 모드

앞서 언급한 바와 같이 S_i를 계산하는 방법을 제외하고는 동일하다. 따라서 카운터 모드의 특성은 OFB 모드의 특성과 동일하다. 그러면 이 차이를 통해 얻을 수 있는 이득은 있는가? OFB의 경우에는 S_i를 알아도 키를 모르면 S_{i+1}를 계산할 수 없다. 하지만 카운터의 경우에는 N_i를 알면 N_{i+1}를 계산할 수 있지만 마찬가지로 키를 모르면 S_i나 S_{i+1}를 계산할 수 없다. 즉, 안전성 측면에는 큰 차이가 없다. 다만, OFB는 각 S_i들을 순차적으로 계산해야 하지만 카운터 모드에서는 각 S_i들을 독립적으로 계산할 수 없다.

3.9 암호화 모드의 비교

<표 3.1> 암호화 모드의 비교

	ECB	CBC	CFB	OFB	Counter
평문패턴의 은닉	×	○	○	○	○
평문의 조작	○	△	△	△	△
전처리 여부	×	×	△	○	○
병렬 수행	ENC:(○) DEC:(○)	ENC:(×) DEC:(○)	ENC:(×) DEC:(○)	ENC:(×) DEC:(×)	ENC:(○) DEC:(○)
오류 확산	단일블록	P_i 전체 P_{i+1} 특정	P_i 특정 P_{i+1} 전체	P_i 특정	P_i 특정
블록보다 작은 단위의 암호화	×	×	○	○	○

다섯 가지 암호화 모드를 비교하면 표 3.1과 같다. ECB는 동일한 두 평문 블록은 항상 동일한 암호문 블록으로 암호화되는 문제점을 가지고 있지만 나머지 모드는 이것을 극복하기 위해 사용되는 모드이다. ECB 모드의 경우에는 암호문 블록을 교체, 삽입, 삭제가 가능하다. CBC 모드와 CFB 모드는 앞과 뒤에서 블록을 삭제할 수 있다. 반면에 OFB와 카운터 모드는 초기벡터가 없으면 복호화 과정을 수행할 수 없기 때문에 앞에서 블록을 삭제할 수는 없다. 하지만 이 두 모드에서는 특정 암호문 블록을 조작하면 해당 평문에 영향을 준다. 이것은 C_i에 한 비트를 조작하면 그것이 그대로 P_i에 영향을 준다. CBC와 CFB의 경우에도 C_i를 조작하였을 때 P_{i+1} 또는 P_i가 그대로 영향을 받지만 다른 한 평문 블록 전체가 엉뚱하게 복호화되므로 평문을 조작할 수 있는 정도가 다르다.

OFB와 카운터 모드에서는 S_i 값들을 미리 계산해 놓을 수 있다. CFB는 현 암호문 블록을 보기 전에 이전 암호문 블록을 미리 복호화해 놓을 수 있다. 병렬 수행이란 여러 평문의 암호화나 여러 암호문의 복호화를 동시에 수행할 수 있는지 여부를 말한다. 특히 병렬 프로세서를 사용할 수 있는 환경에서 병렬 수행이 가능하면 매우 효율적으로 암호화나 복호화를 수행할 수 있다. CBC 같은 경우에는 모든 암호문 블록을 수신한 다음에 이들을 병렬로 복호화한 다음 XOR 연산을 통해 복호화를 수행할 수 있다. CFB도 모든 암호문 블록을 수신한 다음에 이들을 병렬로 암호화한 다음 XOR 연산을 통해 복호화를 수행할 수 있다. OFB에서 암호화 또는 복호화를 하기 위해서는 S_i를 계산해야 한다. 하지만 S_i는 순차적으로만 계산할 수 있어 병렬 수행이 가능하지 않다. 하지만 카운터 모드에서는 N_0가 결정되면 나

머지 N_i를 결정할 수 있고, 이 값이 결정되면 S_i를 결정할 수 있다. 그러므로 카운터 모드는 암호화와 복호화를 모두 병렬로 수행할 수 있다.

3.10 암호화 모드의 비교

이 장에서 소개한 다섯 가지 암호화 모드 중 가장 성능이 좋은 것은 ECB 모드이다. 하지만 이 모드가 암호해독하기 가장 유리하다. 따라서 보통 일반 메시지들은 ECB 모드로 암ㄹ호화하지 않는다. 다만, 암호키와 같이 크기가 작은 랜덤한 값은 ECB로 암호화하여도 문제가 없다. ECB 대신 가장 많이 사용되는 암호화 모드는 CBC 모드이지만 최근에는 카운터 모드를 많이 사용한다. 또 블록암호 알고리즘을 이용하여 스트림 암호방식으로 암호화하고 싶을 경우에는 보통 CFB 모드를 많이 사용한다.

연습문제

1. 다음 각 질문에 해당하는 암호화 모드를 나열하고, 그 이유를 설명하시오.
   ㄱ) 채우기가 필요 없는 암호화 모드
   ㄴ) 암호화와 복호화를 모두 병렬로 수행할 수 있는 모드
   ㄷ) 마지막 암호문 블록을 MAC으로 사용할 수 있는 모드