공학석사 학위논문

산술 시프트 레지스터를 이용한 블록암호 운영모드에 관한 연구

2009년 2 월

부경대학교 산업대학원

컴퓨터공학과

양 상 근

공학석사 학위논문

산술 시프트 레지스터를 이용한 블록암호 운영모드에 관한 연구

지도교수 조 경 연

이 논문을 공학석사 학위 논문으로 제출함

2009년 2 월

부경대학교 산업대학원

컴퓨터공학과

양 상 근

양상근의 공학석사 학위논문을 인준함

2009년 2월

주　심 :　공학박사　서 경 룡 (인)

위　원 :　공학박사　김 종 남 (인)

위　원 :　공학박사　조 경 연 (인)

목 차

그 림 목 차

표 목 차

Study for Block Cipher Operating Mode Using Arithmetic Shift Register

Sang-Keun Yang

Dept of Computer Engineering

Graduate School

Pukyong National University

ABSTRACT

This thesis suggests block cipher operating mode using ASR(Arithmetic Shift Register). ASR is called arithmetic shift register which is sequence that is not 0 but initial value A_0 multiplies not 0 or 1 but free number D on $GF(2^n)$. This thesis proposes ASR mode which changes output multiplying d and Floating ASR mode which has same function but having strengthened stability altering d. If we use ASR's output as a counter,there's advantage that it has higher stability and better speed than CTR. Also, ASR mode and FASR mode have advantage of Random access which is not being functioned on CTR mode,they can be widely used to any part which Random access is needed.

제1장 서론

현대에는 컴퓨터와 정보통신 기술의 비약적 발전으로 컴퓨터 통신망의 보급이 대단히 많이 이루어졌고, 특히 1990년대 이후 더욱 빠른 속도로 발전함에 따라 이제는 고도의 정보화 사회에 접어들었다고 할 수 있다. 정보화 사회란 컴퓨터와 정보통신 기술의 결합으로 정보의 가치가 산업 사회에서 매우 중요해 지는 사회를 말한다. 정보화 사회가 됨으로써 정보처리의 방식이 종이문서에서 컴퓨터와 정보통신을 이용하는 방식으로 변화하였다. 이러한 변화에 따라 많은 양의 정보를 먼 곳으로 쉽게 전달할 수가 있어, 도중의 정보보보 조치가 없으면 전송도중 또는 저장장치에 저장된 상태에서 불법적으로 유출, 삭제 또는 수정될 수가 있어 정보보호가 필수의 과제로 대두되었다[1].

1990년대에 들어와 컴퓨터의 발달로 그동안 널리 사용되던 DES의 해독 가능성이 높아져 새로운 암호 알고리즘이 필요하게 되었고, 1997년 미국 상무 기술 표준국(National Institute of Standard Technology)에서 AES(Advance Encryption Standard)를 공모하였고, J. Deamen과 V. Rijemen이 제안한 Rijindael이 AES암호 알고리즘으로 채택되어 현재까지 쓰이고 있다[2]. 따라서, 본 논문에서는 운영모드에 따른 암호 알고리즘을 소개하고 암호화를 하는 모든 과정을 AES를 중심으로 확인 하였다.

AES가 지금까지 알려진 블록암호 알고리즘에 대한 모든 공격방법들에 대해 안전하도록 설계되었다고 한다[3]. 하지만 공격자들은 암호 알고리즘 자체를 공격하기보다는 그 알고리즘을 운영하는 운영모드에 대

해서 공격을 하여 자신이 원하는 짧은 문장만을 해독하는 경우가 발생할 수 있다. 이와 같이 암호 알고리즘보다 그 알고리즘을 운영하는 모드가 취약한 경우가 많음에도 불구하고 수많은 연구자들은 암호 알고리즘 자체만을 연구할 뿐 운영모드에 대한 연구는 활발히 진행되지 못하고 있는 실정이다.

현재까지 알려진 암호 알고리즘 운영모드는 ECB(Electronic Codebook Mode), CBC(Cipher Block Chaining), CFB(Cipher FeedBack), OFB(Output FeedBack), CTR(CounTeR) 모드가 있다. 그러나 각각의 모드는 보안성, 암호문의 오류, 비트삽입 및 손실 등의 약점을 가지고 있다. 이에 각각의 모드를 개선하고 보안성에 더욱 강력한 모드의 개발이 필요하다.

본 논문에서는 ASR(Arithmetic Shift Register)를 이용한 블록 암호 운용모드를 제안한다. 본 논문에서 제안한 ASR Mode는 LFSR(Linear Feedback Shift Register)을 이용한 CTR Mode보다 안정성이 높을 것으로 기대되며, CTR Mode 및 OFB Mode에서 불편한 Random Access가 더욱 편리해 질것으로 예상된다.

본 논문의 구성은 2장에서 AES암호 알고리즘을 간단히 소개 및 암호 알고리즘 운용 모드의 종류와 각각의 구조, 암호화 및 복호화 과정의 장, 단점을 살펴본다. 3장에서는 ASR을 소개하고, ASR 모드의 알고리즘을 C언어로 구현하고 암호화 및 복호화 과정을 살펴본다. 4장에서는 ASR 모드의 안전성을 검정하고, 기존 모드와 암호화 및 복호화 과정의 속도를 테스트하여 결과를 비교 분석하며, 5장에서는 결론을 내린다.

제2장 블록 암호의 운영모드

2.1 AES 암호 알고리즘 소개

AES는 미국 국립 표준 기술 연구소에서 DES를 대체하기 위해 전 세계적으로 공모하였다. 그 후 공개 검사와 평가, 공개 경쟁을 통하여 2000년 10월에 Joan Daemen과 Vincent Rijmen이 개발한 Rijndael을 AES로 선정하였다. AES는 지금까지 알려진 블록 암호 알고리즘에 대한 공격에 안전하게 설계 되었으며, 하드웨어나 소프트웨어로 구현 했을 때 속도와 코드 면에서 빠르고 안정적인 특징을 가진다. AES 알고리즘의 입력 평문의 길이는 128 비트로 고정이고, 암호화키의 길이는 128 비트, 196비트, 256비트 중에서 선택해서 사용할 수 있다.

AES의 암호화 과정은 평문이 입력되면 각 라운드 동작이 수행되기 전 평문과 라운드 키의 배타적 논리합 연산(XOR)이 수행된다. 이 과정이 AddRoundKey() 연산이다. 이 후에는 각 라운드마다 4가지 연산을 동일하게 수행하며, 전체 라운드가 r이면 r-1라운드까지 수행한다. 각 라운드마다 수행되는 4가지 연산은 아래와 같다.

- SubBytes() : S 박스를 이용하여 바이트 단위로 치환을 수행
- ShiftRows() : god 단위로 순환 시프트를 수행
- MixColumns() : 열 단위로 혼합을 수행
- AddRoundKey() : 라운드 키와 배타적 논리합 연산을 수행

암호화 과정의 마지막 라운드는 위의 4가지 연산중에서

MixColumns() 연산을 수행하지 않고, 세 가지 연산만을 수행하여 암호문이 출력된다. 복호화 과정은 암호화 과정을 반대로 수행하면 된다. 암호화 및 복호화 과정은 [그림 2-1]과 같다[1].

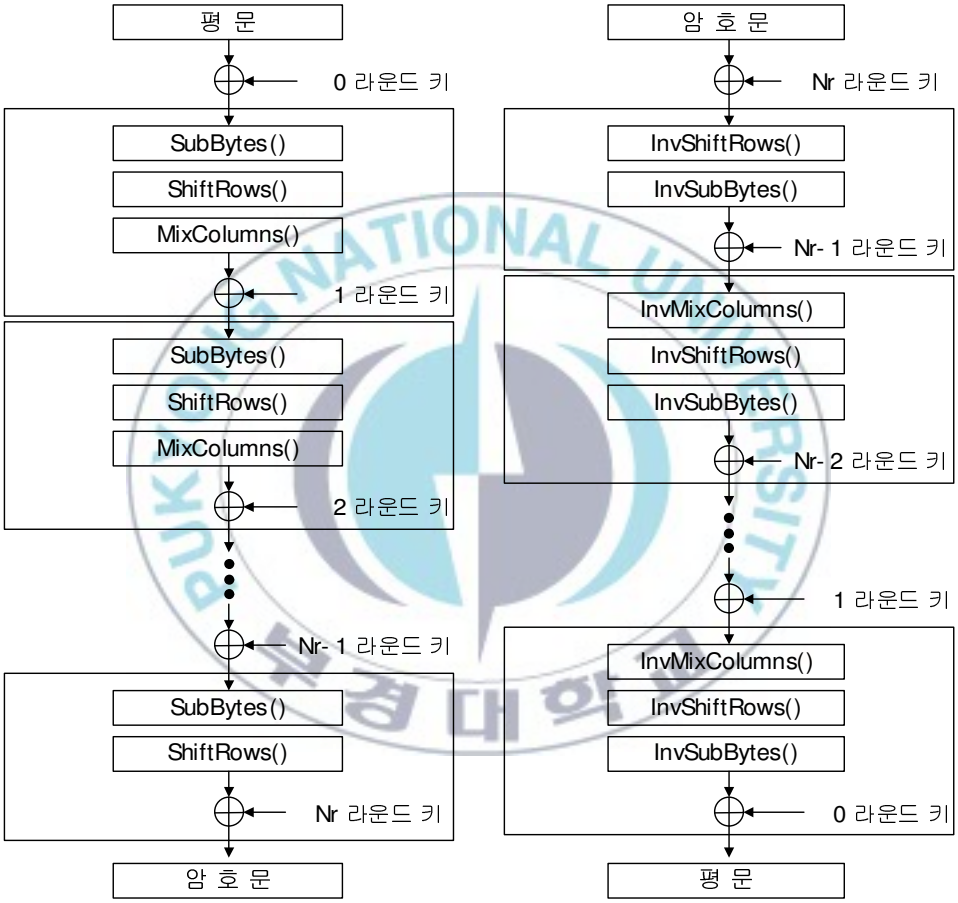

그림 2-1. AES의 암호화 및 복호화 과정

SubBytes() 연산은 S 박스를 사용하여 치환을 하는 단계이다. S 박스는 두 가지 연산을 하여 만들어지는데, 첫 번째 연산은 $GF(2^8)$에 대

한 곱셈의 역원을 구한다. 입력이 00인 경우에는 자기 자신으로 치환한다. 두 번째 연산은 $GF(2)$상의 아핀 변환을 구하는 것이다. 아핀 변환은 아래 행렬식과 같다.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \bullet \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

SubBytes()연산을 그림으로 나타내면 [그림 2-2]와 같다.

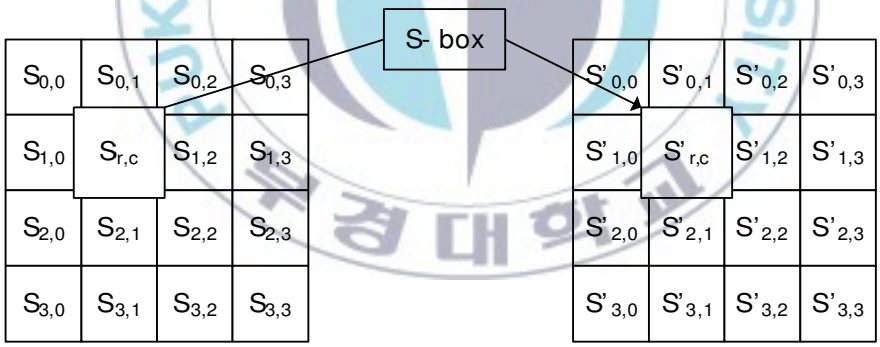

그림 2-2. SubBytes() 연산

이 변환은 가역이며, 역변환 테이블도 미리 만들어 복호에 사용한다.

ShiftRows() 연산은 4x4 행렬로 표시되는 상태의 각 행을 위에서부터 0, 1, 2, 3 만큼씩 좌로 순환시킨다. ShiftRows() 연산은 [그림 2-3]과 같다.

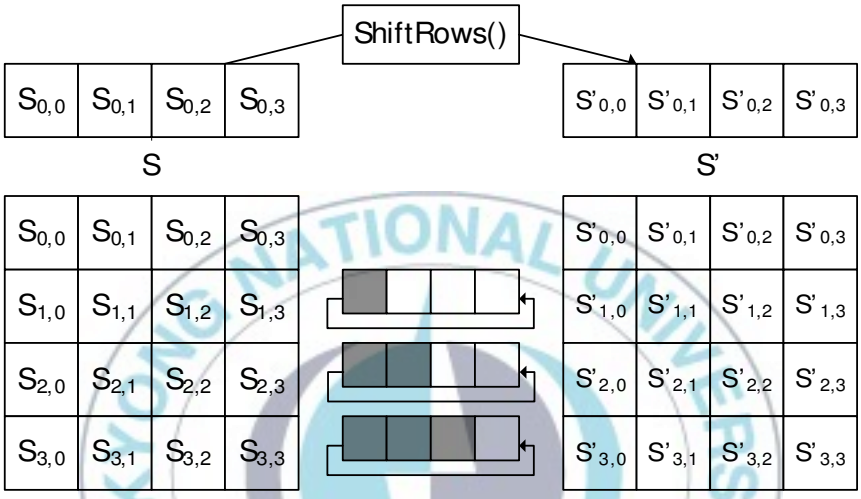

그림 2-3. ShiftRow() 연산

MixColumns() 연산은 상태의 각 열에 해당하는 바이트 블록들이 서로 영향을 받도록 변환시켜 주는 단계이다. 행렬의 각 열을 $GF(2^8)$ 상의 3차이하 다항식으로 표현하여 $c(x)=03x^3+01x^2+01x+02$를 곱하고 x^4+1로 나눈 나머지를 결과로 선택한다. 이 연산은 다음과 같은 행렬식으로 표현할 수 있다.

$$\begin{vmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{vmatrix} = \begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \times \begin{vmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{vmatrix}, \quad 0 \le c \le 4$$

MixColumns() 연산을 그림으로 표현하면 [그림 2-4]와 같이 나타낼 수 있다.

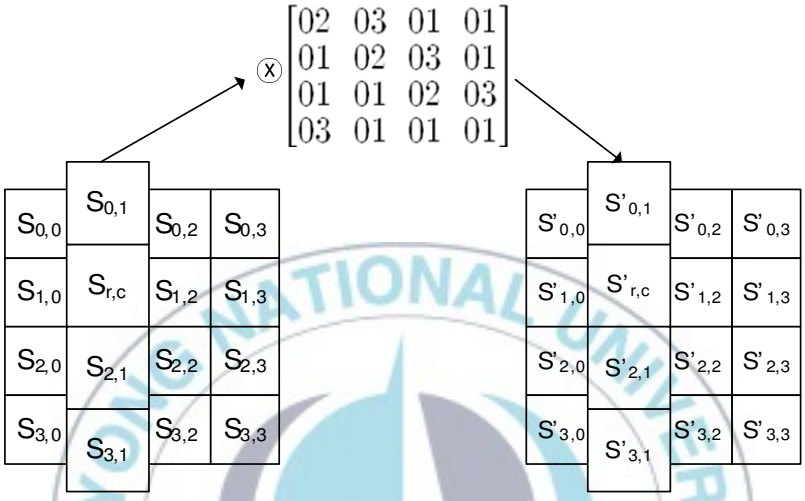

그림 2-4. MixColumns() 연산

AddRoundKey() 연산은 현재 상태의 각 바이트와 라운드 키를 XOR연산을 수행한다. AddRoundKey() 연산은 [그림 2-5]와 같다.

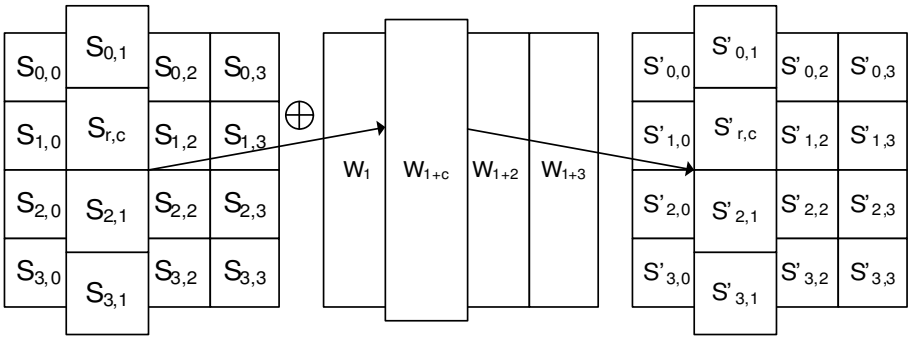

그림 2-5. AddRoundKey() 연산

2.2 블록 암호의 운영 모드

블록 암호화는 오직 고정된 크기의 블록만을 암호화 한다. 평문의 길이가 블록 암호의 블록 크기보다 클 경우 어떻게 블록 암호를 적용할 것인가를 해결하기 위해 다양한 응용 환경하에 적절한 암호화 도구로 사용할 수 있는 여러 유형의 효율적인 운영 방식들을 제시하고 있다. 이것을 암호화 모드(Block Cipher Mode)라고 한다.

일반적으로 블록 암호화 모드는 임의의 길이 평문 M을 임의의 길이 암호문 C로 암호화 하는 방법이다. 대부분의 모드는 평문 M의 길이가 정확히 블록 크기의 배수가 되어야 한다. 이 원칙 때문에 채우기(Padding)가 필요하다. 평문을 채우는 데는 여러 가지 방법이 있다. 그러나 가장 중요한 법칙은 채우기는 되돌릴 수 있어야 한다는 것이다. 채워진 메시지로부터 원래의 메시지를 얻는 것이 가능해야 한다.

블록 암호의 대표적인 운영모드에는 ECB 모드(Electronic Code Book Mode), CBC모드(Cipher Block Chaining Mode), CFB 모드(Cipher Feed Back Mode), OFB모드(Output Feed Back Mode), 그리고 CTR모드(CounTeR Mode)가 있다.

2.2.1 ECB 모드 (Electronic Code Book Mode)

ECB 모드는 암호 운영방식 중 가장 간단한 방법으로 평문을 64 비트씩 나누어 암호화 하는 방식이다. 평문을 64비트씩 나눌 때 마지막 블록이 64비트가 되지 않을 때는 임의의 약속된 비트 모양을 패딩(Padding)하게 된다[4]. ECB 모드의 수식 표현은 [식-1] 과 같이 정의되며 [그림 2-2]와 같이 나타낼 수 있다.

$$C_i = E(K, P_i) \quad (1)$$
$$(\text{단}, i = 1, 2, \ldots, k)$$

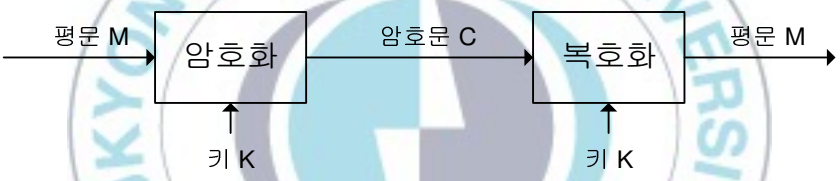

그림 2-6. ECB 모드의 암호화 및 복화화 과정

이 방식은 동일한 평문 블록 모양에 따라 항상 동일한 암호문이 출력되므로 암호 해독자 들의 해독 가능성을 높게 만든다. 따라서 암호문을 살펴보는 것만으로도 평문 속에 패턴의 반복이 있다는 것을 알게 되며, 이것을 실마리로 암호 해독을 할 수 있게 된다. ECB 모드는 굉장히 취약하므로 암호화 모드로 절대 사용하면 안 된다.

2.2.2 CBC 모드(Cipher Block Chaining Mode)

CBC모드는 출력 암호문이 다음 평문 블록에 영향을 미치게 하여 각 암호문 블록이 전단의 암호문의 영향을 받도록 만든 방식으로 ECB모드에서 발생하는 동일한 평문에 의한 동일한 암호문이 발생하지 않도록 구성한 동작 모드이다.

CBC모드의 동작은 [그림 2-3]과 같이 처음 입력된 평문 블록 M_1은 초기벡터 IV_0(Innitial Vector)와 XOR되어 암호기에 입력된다. 암호기 출력 암호문 C_1은 다음단 평문 블록 M_2와 XOR되어 암호기에 다시 입력된다. CBC모드의 수식 표현은 [식-2] 와 같이 정의되며 [그림 2-3]과 같이 나타낼 수 있다.

$$\begin{aligned} &\text{암호화}: C_i = E_k(M_i \oplus C_{i-1}) \\ &\text{복호화}: M_i = D_k(C_i) \oplus C_{i-1} \\ &\quad (\text{단}, C_0 = IV_0) \end{aligned} \tag{2}$$

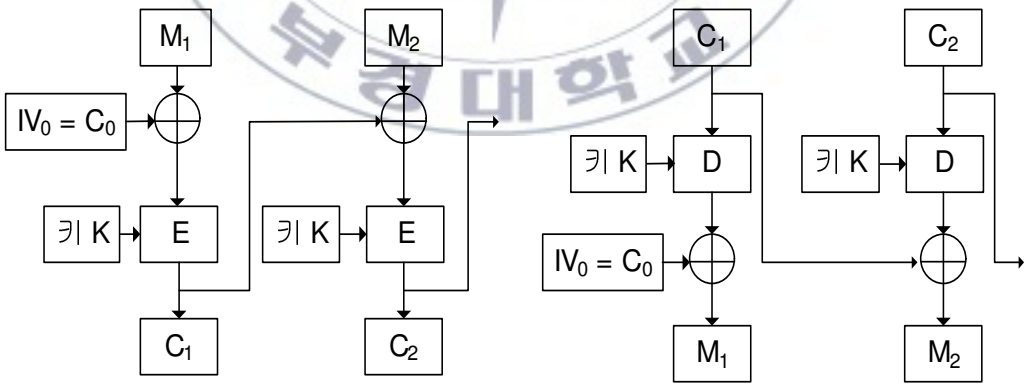

그림 2-7. CBC 모드의 암호화 및 복호화 과정

2.2.3 CFB 모드(Cipher Feed Back Mode)

CFB 모드도 CBC 모드와 마찬가지로 평문 블록이 동일한 경우 동일한 암호문이 나타나지 않도록 전단의 암호문이 다음 단의 평문에 영향을 미치게 구성하는 방식이다. CFB 모드의 동작은 [그림 2-4]와 같이 CBC를 연상시키게 구성되어 있으나, 다른점은 암호문이 수신자의 암호기 입력으로 사용되는 것이다. CFB모드는 현재 생성된 암호문이 이후에 생성되는 암호문에도 영향을 미친다는 측면에서 CBC모드와 유사하다. 따라서 암호문에 발생되는 채널상의 잡음에 의한 오류는 해당 암호문의 복호화 뿐만 아니라 연속적으로 발생되는 암호문의 복호화에도 영향을 미친다. CFB모드의 수식 표현은 [식-3] 와 같이 정의되며 [그림 2-4]과 같이 나타낼 수 있다.

$$\begin{aligned} &\text{암호화}: C_i = E_k(C_{i-1}) \oplus M_i \\ &\text{복호화}: M_i = D_k(C_{i-1}) \oplus C_i \\ &(\text{단}, C_0 = IV_0) \end{aligned} \tag{3}$$

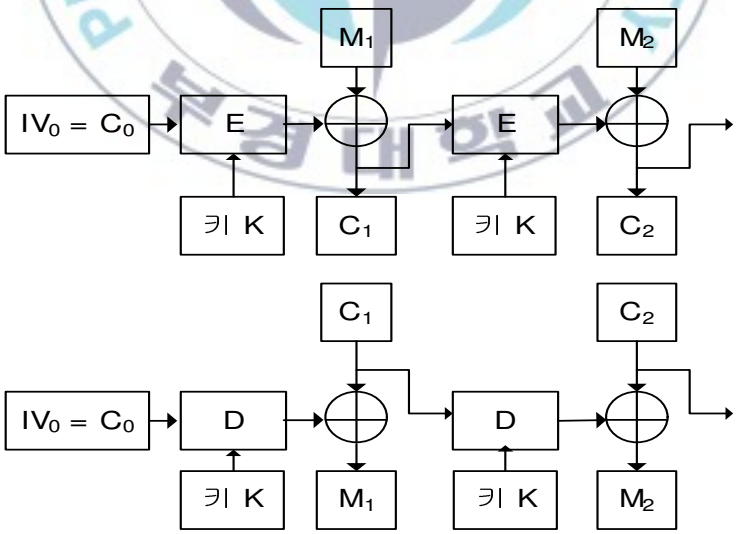

그림 2-8. CFB 모드의 암호화 및 복호화과정

2.2.4 OFB 모드(Output Feed Back Mode)

OFB모드 동작은 평문을 서로 독립적으로 암호화 하는 ECB모드의 단점과 오류전파가 발생하는 CBC모드와 CFB모드를 개선한 동작모드이다. OFB모드 동작은 암호기의 출력과 평문을 XOR하여 암호문을 생성하고 있으므로 오류 전파가 발생하지 않고 단지 해당 암호문만이 영향을 받는다. 그러나 암호문 송신자와 수신자 사이에 동기를 조절해야 한다. 즉, 전송중인 암호문의 비트 손실이나 삽입등에 유의해야 하는 방식이다. OFB모드의 수식 표현은 [식-4] 와 같이 정의되며 [그림 2-5]과 같이 나타낼 수 있다.

$$\begin{aligned} &\text{암호화}: C_i = E_k(Z_{i-1}) \oplus M_i \\ &\text{복호화}: M_i = D_k(Z_{i-1}) \oplus C_i \\ &(\text{단}, C_0 = IV_0) \end{aligned} \tag{4}$$

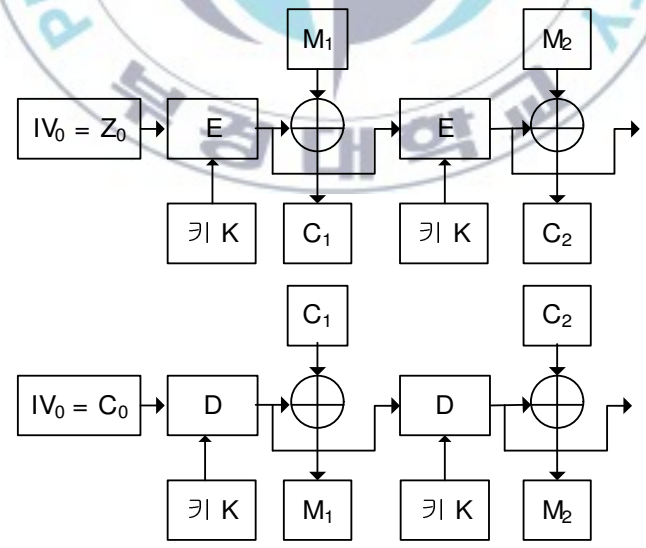

그림 2-9. OFB모드의 암호화 및 복호화 과정

2.2.5 CTR 모드(Counter Mode)

CTR모드는 1씩 증가해 가는 카운터를 암호화 해서 키 스트림을 만들어내는 스트림 암호이다. 블록을 암호화 할 때마다 1씩 증가해 가는 카운터를 암호화해서 키 스트림을 만든다. 즉 카운터를 암호화한 비트열과 평문 블록과의 XOR연산을 취한 결과가 암호문 블록이 된다. CTR모드의 장점은 동일한 평문이 순서에 따라서 다른 암호문을 생성하기 때문에 안전도를 높일 수 있다는 것이다. 그러나 Counter의 특성상 대다수의 Bit가 변화하지 않는 단점이 있다. CTR모드의 수식 표현 [식-5] 와 같이 정의되며 [그림 2-6]과 같이 나타낼 수 있다.

$$\begin{aligned} &\text{암호화}: C_i = E(K, P_i \oplus (N \parallel T_i)) \\ &\text{복호화}: M_i = D(K, P_i \oplus (N \parallel T_i)) \end{aligned} \tag{5}$$

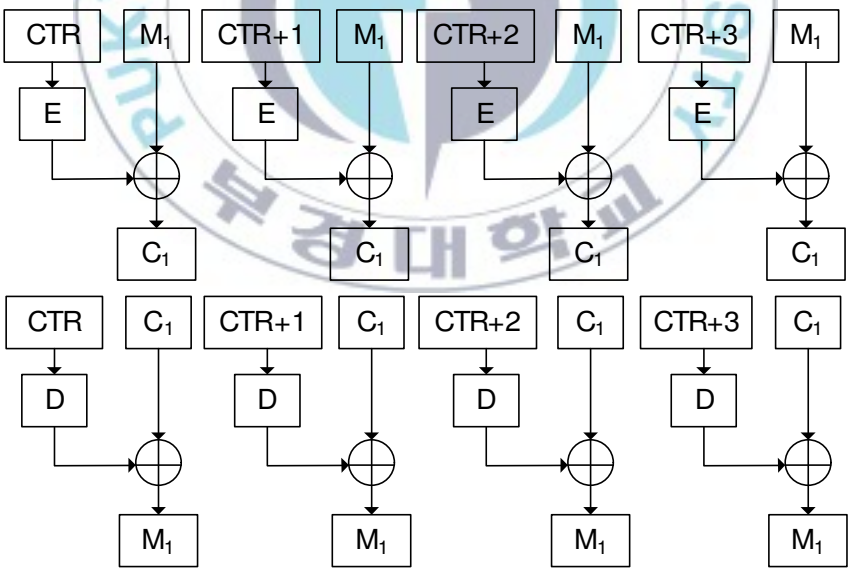

그림 2-10. CTR 모드의 암호화 및 복호화 과정

제3장 산술 시프트 레지스터
(Arithmetic Shift Register)

3.1 갈로이 선형 궤환 시프트 레지스터

갈로이 선형 궤환 시프트 레지스터는 소프트웨어로 구현인 곤란한 피보나치 선형 궤환 시프트 레지스터의 단점을 보완하기 위해서 구성한 것이다. 갈로이 선형 궤환 시프트 레지스터는 마지막 단의 출력이 동시에 궤환 함수를 통과한 후 앞단의 출력과 모듈로 2 덧셈을 하여 다음 단으로 입력되며, 순환 방정식은 [식-6]이 되며, 하드웨어로 구현을 하면 [그림3-1]과 같이 나타낼 수 있다.

$$\begin{aligned} a_i' &= a_{i+1} \oplus p_{i+1} a_0 \quad \text{for} \, 0 \le i \le n-2 \\ a'_{n-1} &= p_n a_0 \end{aligned} \tag{6}$$

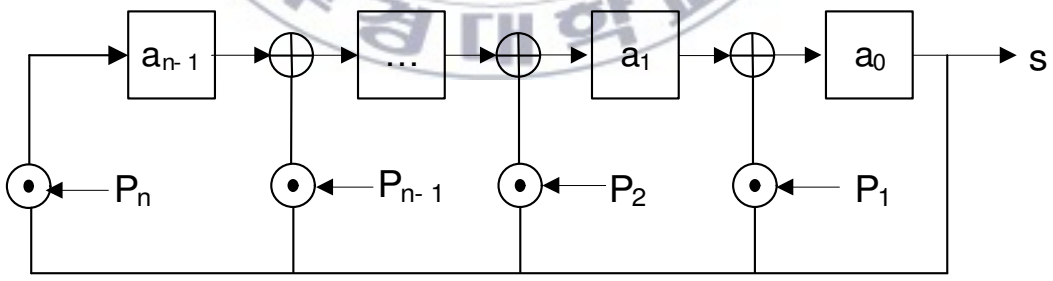

그림 3-1 갈로이 선형 궤환 시프트 레지스터

3.2 산술 시프트 레지스터

ASR(Arithmetic Shift Register)[6]이란 $GF(2^n)$상에서 0이 아닌 초기 값 A_0에 0또는 1이 아닌 임의의 수 D를 곱하는 수열을 산술 시프트 레지스터(ASR-D)로 정의한다. ASR-D의 i번째 값(상태) A_i는 A_0D^i가 된다. 따라서 갈로이 선형 궤환 시프트 레지스터는 '$D=2^{-1}$'인 산술 시프트 레지스터다. 다시말해서 산술 시프트 레지스터는 갈로이 선형 궤환 시프트 레지스터의 일반형이다. 갈로이 선형 궤환 시프트 레지스터의 순환 방정식인 [식-6]은 $GF(2^n)$상에서 비복원다항식(irreducible polynomial)이 $P(x)$인 경우에 '$A_{i+1}=\frac{A_i}{2}$'를 나타낸다. 따라서 갈로이 선형 궤환 시프트 레지스터는 ASR-2^{-1}이다.

$GF(2^n)$상에서 0또는 1이 아닌 임의의 수 D에 대하여 '$D^k=1$'이 되는 t가 '$t=2^n-1$'로 유일하면 0을 제외한 $R\in\{1,2,\ldots,2^n-1\}$은 $D^u\in\{1,2,\ldots 2^n-1\}$로 표현할 수 있으며, 이때 A는 $A=A_0D^u\in\{1,2,\ldots,2^n-1\}$로 표현 할 수 있다.

또한, $GF(2^n)$상의 비복원 다항식 $P(x)$가 0 또는 1이 아닌 임의의 수 D에 대하여 '$D^k=1$'이 되는 t가 '$t=2^n-1$'로 유일하면 $P(x)$는 ASR-D의 특성다항식 (Characteristic polynomial)이 된다. 따라서 $GF(2^n)$에서 특성다항식으로 표현되는 ASR-D의 주기는 '2^n-1'이다.

32비트 컴퓨터에서 $GF(2^{32})$상에서 임의 값 D에 대한 ASR-D를 C언어로 프로그램 한 것을 [표-1]에 보이고, $GF(2^{32})$상의 ASR-2와 갈로이 및 피보나티 선형 궤환 시프트 레지스터를 C언어로 프로그램한 것을 [표-2]에 보인다.

표 1. $GF(2^{32})$상의 ASR-D의 C 프로그램

```
unsigned int P;//Characteristic polynomial
unsigned int A, D;
for (B=0 ; D ; D >>= 1 )
  { if ( D & 1 ) B ^= A;
    A = ( A << 1 ) ^ ( (int)A >> 31 ) & P );}
```

표 2. $GF(2^{32})$상의 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터의 C 프로그램

```
unsigned int P;//Characteristic polynomial
unsigned int A, D;

/* ASR-2 */
A = (A << 1) ^ ((int)A >> 31) & P);

/* Galois-LFSR */
A = ((A ^ ( - (A & 1) & P)) >> 1) | (A << 31);

/* Fibonacci-LFSR */
B = A ^ P;
B ^= B >> 16;
B ^= B >> 8;
B ^= B >> 4;
B ^= B >> 2;
B ^= B >> 1;
A = (A >> 1) | (B << 31);
```

[표-2]에서 32비트 산술 시프트 레지스터는 & 연산 1회, XOR 연산 1회, 시프트 연산 2회 만을 수행하므로 총 4번의 연산만이 필요하다. (int) 형변환 연산자는 해당하는 기계어를 선정하는 기능만을 수행하

는 것으로 연산시간을 소요하지 않는다.

이와 비교하여 갈로이 선형 궤환 시프트 레지스터는 & 연산 2회, OR연산 1회, XOR 연산 1회, 시프트 연산 2회를 수행하므로 총 6번의 연산이 필요하다. 그리고 피보나치 선형 궤환 시프트 레지스터는 XOR 연산 6회, OR 연산 1회, 시프트 연산 7회 수행해서 총 14회 연산을 필요로 한다.

32 비트 컴퓨터에서 $GF(2^{32})$상과 $GF(2^{64})$상에서 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터를 C언어로 프로그램하는 경우에 각각 수행되는 동작을 비교하여 [표 3]에 보인다.

표 3. $GF(2^{32})$상과 $GF(2^{64})$상에서 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터의 동작속도 비교

	ASR-2				
	&	OR	XOR	Shift	Total
$GF(2^{32})$	1		1	2	4
$GF(2^{64})$	2	1	2	4	9
	Galois-LFSR				
	&	OR	XOR	Shift	Total
$GF(2^{32})$	2	1	1	2	6
$GF(2^{64})$	3	2	2	4	11
	Fibonacci-LFSR				
	&	OR	XOR	Shift	Total
$GF(2^{32})$		1	6	7	14
$GF(2^{64})$		2	8	9	19

[표 3]으로부터 ASR-2가 소프트웨어 구현시에 $GF(2^{32})$상에서는

필요한 연산의 수가 종전의 갈로이 선형궤환 시프트 레지스터보다 33%, 피보나치 선형궤환 시프트 레지스터 보다 71%적게 필요함을 알 수 있다. 또한 $GF(2^{64})$상에서는 각각 18%와 53%가 적게 필요하다.

ASR-2를 하드웨어로 구현하면 아래 [그림 3-2]와 같으며, [그림 3-2]와 [그림 3-1]을 비교하면 시프트되는 방향만 다른 것을 제외하고는 동일한 것을 알 수 있다. 따라서 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터의 하드웨어 복잡도는 동일하다. 그러므로 ASR-2의 선형복잡도는 n이며, ASR-D의 선형복잡도는 $'n \le LC \le (n^2+n)/2'$ 으로 종래의 궤환 시프트 레지스터 보다 높아서 안전도가 높다.

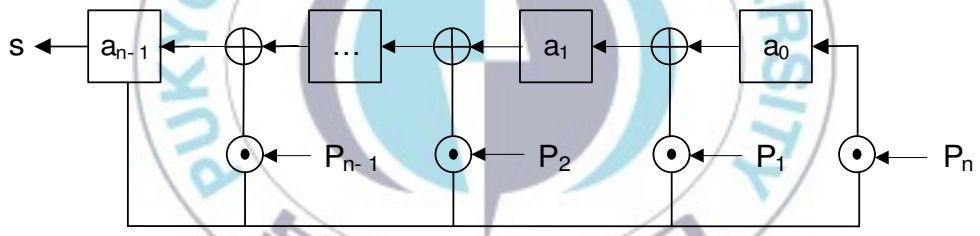

그림 3-2. ASR-2의 하드웨어 구현

제4장 산술 시프트 레지스터를 이용한 암호 운영모드

4.1 ASR 모드

ASR 모드의 동작방법은 CTR 모드의 동작방법과 거의 유사한 형태로 구현을 하였는데, CTR 모드의 수식표현인 $C_i = E(K, P_i \oplus (N \| T_i))$ 에서 N은 nonce 즉 random number이고, T_i는 $'T_ = T_{i-1} + 1'$인 Counter이다. 예를 들어 128비트 블록에서 N은 96 bit가 되고, T는 32 bit Counter이다. CTR 모드의 장점은 동일한 평문이 순서에 따라서 다른 암호문을 생성하기 때문에 안전도를 높일 수 있다는 것이다. 그러나 Counter의 특성상 대다수의 bit가 변화하지 않는 단점이 있으므로 이를 보완할 필요가 있다. 즉, 32 bit Counter에서 bit 0은 매번 '0'에서 '1' 또는 '1'에서 '0'으로 규칙적으로 변화하고, 31 bit는 50%의 빈도로 변화하지 않는다. 나아가서 상위 30 bit는 75%의 빈도로 변화하지 않는다. 이렇게 고정된 변화하지 않는 bit가 많으면 이를 이용한 공격의 가능성이 늘어나게 된다.

이를 보완하여 CTR 모드에서는 Counter를 LFSR로 사용하기도 하는데, Fibonacci-LFSR은 msb를 제외하고 항상 $b_i = b_{i+1}$ 이 된다. 즉, msb 1 bit를 제외하고 단순 shift 동작이므로 이를 이용한 공격이 가능하게 된다. 이를 보완하기 위하여 Galois-LFSR을 사용할 수 있는데, 이것은 75%dml 확률로 $b_i = b_{i+1}$이 되다. 즉, tab 위치가 아닌 bit는 항상 $b_i = b_{i+1}$이고, tab 위치에서는 50%의 확률로 $b_i = b_{i+1}$이 된다. 또한

LFSR을 사용한 CTR 모드에서는 random access가 불편하게 된다.

위와같은 단점을 보완하기 위해 ASR을 이용한 블록암호 운영모드를 생각할 수 있는데 이를 수식으로 표현하면 [식 7] 과같이 표현할 수 있다.

ASR모드는 d를 곱해가면서 나타나는 산술 시프트 레지스터의 출력으로 키 스트림을 만드는 스트림 암호의 형태라 할 수 있다. ASR모드는 산술 시프트 레지스터의 출력을 카운터로 사용하는 방식으로 동작방법은 ASR의 첫 번째 출력 A_0가 초기벡터가 되며, A_0와 평문블록 M_1을 XOR 연산한 후 이를 암호기에 입력하여 초기화 키 K와 함께 암호화한다. 이 때 ASR의 첫 번째 출력이 정해지면 두 번째, 세 번째 등은 첫 번째 출력으로부터 d를 곱하여서 출력을 구한다.

예를 들어 128비트 블록에서 산술 시프트 레지스터의 특성방정식은 “00000002 00000004 00000005 00000041”이고, d는 2^{19}을 선택할 수 있다. 이 모드는 d값이 2^{19}이므로 블록 마다 각각 19비트씩 회전하게 되므로 변화하는 비트의 수가 많아서 CTR 모드보다 안정성이 높을 것으로 기대된다. 또한 A_0가 CTR 모드의 nonce 기능을 가질 수 있다. 따라서 A_0의 i번째 블록에 대한 A_i는 ‘$A_i = A_0 * D^i$’ 가 되므로 Random Access가 가능하다.

$$
\begin{aligned}
&A_0 = IV \\
&A_i = A_{i-1} \times d \quad (i = 1, 2, 3, \cdots) \\
&C_j = E_k(M_j \oplus A_j) \quad (j = 0, 1, 2, 3, \cdots)
\end{aligned}
\tag{7}
$$

(단, A_0는 초기벡터 IV)

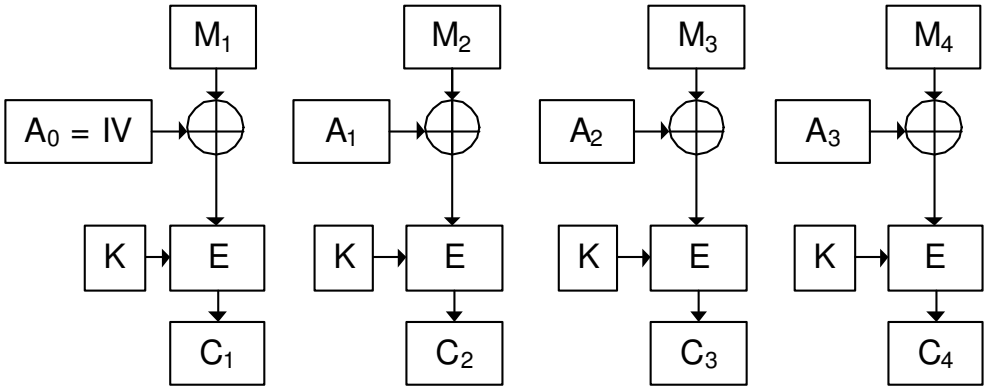

그림 4-1 ASR 모드의 구현

즉 ASR모드는 위의 [식-7]과 같이 정의되며, 그림으로 표현하면 [그림 4-1]과 같이 나타낼 수 있다.

4.2 FASR 모드

ASR의 d값을 변경시켜 안정성을 더욱 강화한 방식으로, Floating ASR 모드라고 가칭을 한 모드가 FASR 모드이다.

예로, 4.1절의 ASR 모드에서 제시한 “00000002 00000004 00000005 00000041”의 특성 방정식에서는 d의 값을 “$2^{19}, 2^{19}+1, 2^{13}, 2^{13}+1$” 의 4가지 경우를 선택할 수 있다. 이와 같이 d를 선택하여 사용을 하게 되면, 안정성은 강화되지만 ASR의 장점인 Random Access가 불편해지는 단점이 있다.

FASR 모드는 아래의 [식 8]과 같이 정의 되며 그림으로 나타내면 ASR 모드의 그림인 [그림 4-1]과 동일하게 나타낼 수 있다.

$$\begin{aligned} A_0 &= IV \\ A_i &= A_{i-1} \times d(M_{i-1} \cap 3) \ (i = 1, 2, 3, \cdots) \\ C_j &= E_k(M_j \oplus A_j) \ \ (j = 0, 1, 2, 3, \cdots) \end{aligned} \tag{8}$$

(단, A_0는 FASR의 첫 번째 출력)

제5장 구현 및 비교

Ⅳ장에서 제시한 ASR 모드와 FASR 모드를 각각 C언어로 구현한 내용을 아래의 [표 4]와 [표 5]에 보인다.

표 4. C로 구현한 ASR 모드

```
C = (word32 *)(kbuf + BUFP2) ;
wa = C[3]>>19 ;
C[3] = ((C[3]<<13) | (C[2]>>19) )^(wa<<1) ;
C[2] = ((C[2]<<13) | (C[1]>>19) )^(wa<<2) ;
C[1] = ((C[1]<<13) | (C[0]>>19) )^(wa<<2)^wa ;
C[0] = (C[1]<<13) ^ (wa<<6) ^ wa ;
```

표 5. C로 구현한 FASR 모드

```
C = (word32 *)(kbuf + BUFP2); a = C[3]>>i ;
if (pch & 0x8) i = 19 ;
    else i = 13 ;
if (pch & 0x80) b = -1 ;
    else b = 0 ;
C[3] = ((C[3]<<(32-i)) | (C[2]>>i))^(a<<1)^ (C[3] & b);
C[2] = ((C[2]<<(32-i)) | (C[1]>>i))^(a<<2)^ (C[2] & b);
C[1] = ((C[1]<<(32-i)) | (C[0]>>i))^(a<<2)^ w ^ (C[1] & b);
C[0] = (C[1]<<(32-i))^(a<<6)^a^(C[0] & b);
```

CTR 모드는 Counter의 특성상 대다수의 비트가 변화하지 않는 단점이 있으므로, 변화하지 않는 비트를 공격할 가능성이 있다. 본 논문에서 제시한 ASR 모드와 FASR모드는 각 블록마다 다수의 비트만큼 회전하므로 변화하는 비트수가 많아지므로 CTR 모드보다 안정성이 높을 것으로 예상된다.

다음으로 기존 암호운영모드와 ASR 모드의 수행속도를 비교하기 위해, 각 운영모드를 이용해서 암호화 해보았다. 용량이 작은 파일은 수행속도 비교에 있어서 어려움이 있기 때문에, 용량이 큰 동영상 파일을 이용해서 각 모드의 수행 속도를 측정해 보도록 한다. 아래의 Test는 약 860MB정도의 크기를 가진 동영상 파일을 각 모드별로 암호화 하는데 걸리는 시간을 측정한 결과이다.

Test환경은 WindosXP Professional OS가 설치되고, AMD Athlon 2600+ (1.91 GHz)의 CPU와 1GB의 Memory가 사양의 PC를 이용하여 Visual Studio 2005를 사용하여 구현하였으며, 기본 암호화 알고리즘으로는 AES 알고리즘을 사용하였다.

표 6. 각 모드의 수행시간 비교

(단위 : 초)

모드	암호화 시간
CBC	573.031000
CFB	575.156000
OFB	574.765000
CTR	573.688000
ASR	573.062000
FASR	573.047000

[표 6]에서 알 수 있듯이 ASR 모드의 암호화시간과 다른 모드들의 암호화시간을 비교해 보면 ASR 모드의 암호화시간이 다른 모드들의 암호화 시간에 비해서 뒤지지 않으며, 오히려 몇 모드에 비교해서는 암호화 시간이 단축 됨을 알 수 있다.

또한, ASR 모드는 [그림 4-1]에서 알 수 있듯이 앞단 블록의 암호문이 다음 블록의 암호화과정에 영향을 미치지 않기 때문에 부분적으로 암/복호화가 가능하다. 이것은 큰 파일을 암/복호화 해야할 때, 유용하게 사용될 수 있다. 예를 들어서 1GB의 암호화된 파일을 읽으려 할 때에 필요한 부분을 알고 있다면, 그 부분만 복호화를 해서 필요한 정보를 얻는 것이 가능하다. 따라서, Random Access가 필요한 부분에서 ASR 모드를 사용할 수 있을 것으로 사료된다.

본 논문에서 암호화 과정을 직접 보여줄 수는 없기 때문에 그림파일을 암호화 하여 이것을 열었을 때의 결과를 보여 주고자 한다. 동일한 그림을 동일한 PC에서 CBC 모드 및 ASR 모드로 암호화 하여 그림파일을 읽을 수 있는지 확인 하였다. 그림 파일은 용량이 작기 때문에 별도로 시간을 측정하는 작업은 하지 않았으며, 암호화 전의 그림파일과 CBC모드로 암호화 한 그림파일, ASR 모드로 암호화한 그림파일을 각각 WindowsXP에서 지원하는 이미지 미리보기를 이용하여 그림 파일을 확인한 결과를 [그림 5-1][그림5-2]에 보인다. 그림파일을 암호화 한 결과 CBC 모드와 마찬가지로 ASR 모드도 정상적으로 암호화가 됨을 알 수 있다.

그림 5-1. 원본그림파일과 CBC모드로 암호화한 그림파일

그림 5-2. 원본그림파일과 ASR모드로 암호화한 그림파일

제6장 결 론

정보화 사회로 접어든 현재에 정보보호는 중요한 연구 분야로 자리 잡았다. 여러 가지 암호 알고리즘이 발명되고 또 이것을 공격하는 공격자들도 나타나고 있다. 따라서 공격자들의 공격으로부터 안전한 암호 알고리즘을 만드는 것이 암호학 연구의 가장 중요한 부분이다. 현재까지 알려져 있는 블록암호 알고리즘 중 AES는 공격자들의 공격으로부터 안전하게 설계되었다고 한다. 그러나 실제로 공격자들은 블록암호 알고리즘이 아닌 그 알고리즘을 운영하는 모드를 공격하여 자신이 원하는 정보를 경우가 있다. 그러나 블록암호 알고리즘의 연구보다 그 알고리즘을 운영하는 모드에 관한 연구는 미흡한 실정이다. 현재까지 알려진 모드들은 각각의 특징에 따라 보안상 취약한 부분들을 가지고 있다.

본 논문에서는 현재까지 알려진 블록암호 알고리즘의 운영모드보다 더욱 안정성이 강화된 모드들을 제안하였다.

산술 시프트 레지스터(Arithmetic Shift Register)를 이용한 블록암호 운영모드인 ASR 모드와 ASR 모드에서 d값을 변경시켜 안정성을 더욱 강화한 FASR모드가 그것이다.

ASR 모드의 동작방법은 CTR 모드의 동작방법과 거의 유사한 형태로 구현을 하였는데, CTR 모드의 수식표현인 $C_i = E(K, P_i \oplus (N \| T_i))$ 에서 N은 nonce 즉 random number이고, T_i는 $'T_ = T_{i-1} + 1'$인 Counter이다. 예를 들어 128비트 블록에서 N은 96 bit가 되고, T는 32 bit Counter이다. CTR 모드의 장점은 동일한 평문이 순서에 따라서 다른 암호문을 생성하기 때문에 안전도를 높일 수 있

다는 것이다. 그러나 Counter의 특성상 대다수의 bit가 변화하지 않는 단점이 있으므로 이를 보완할 필요가 있다. 즉, 32 bit Counter에서 bit 0은 매번 '0'에서 '1' 또는 '1'에서 '0'으로 규칙적으로 변화하고, 31 bit는 50%의 빈도로 변화하지 않는다. 나아가서 상위 30 bit는 75%의 빈도로 변화하지 않는다. 이렇게 고정된 변화하지 않는 bit가 많으면 이를 이용한 공격의 가능성이 늘어나게 된다.

이를 보완하여 CTR 모드에서는 Counter를 LFSR로 사용하기도 하는데, Fibonacci-LFSR은 msb를 제외하고 항상 $b_i = b_{i+1}$ 이 된다. 즉, msb 1 bit를 제외하고 단순 shift 동작이므로 이를 이용한 공격이 가능하게 된다. 이를 보완하기 위하여 Galois-LFSR을 사용할 수 있는데, 이것은 75%dml 확률로 $b_i = b_{i+1}$이 되다. 즉, tab 위치가 아닌 bit는 항상 $b_i = b_{i+1}$이고, tab 위치에서는 50%의 확률로 $b_i = b_{i+1}$이 된다. 또한 LFSR을 사용한 CTR 모드에서는 random access가 불편하게 된다.

ASR모드는 d를 곱해가면서 나타나는 산술 시프트 레지스터의 출력으로 키 스트림을 만드는 스트림 암호의 형태라 할 수 있다. ASR모드는 산술 시프트 레지스터의 출력을 카운터로 사용하는 방식으로 동작 방법은 ASR의 첫 번째 출력 A_0가 초기벡터가 되며, A_0와 평문블록 M_1을 XOR 연산한 후 이를 암호기에 입력하여 초기화 키 K와 함께 암호화 한다. 이 때 ASR의 첫 번째 출력이 정해지면 두 번째, 세 번째 등은 첫 번째 출력으로부터 d를 곱하여서 출력을 구한다. 또한 A_0가 CTR 모드의 nonce 기능을 가질 수 있다. 따라서 A_0의 i번째 블록에 대한 A_i는 '$A_i = A_0 * D^i$' 가 되므로 Random Access가 가능하다.

ASR 모드의 암호화 과정은 평문블록과 ASR의 출력을 XOR한

뒤 암호 키와 함께 암호 알고리즘을 통과하는 방식으로 구성되며, [식 7]와 같이 '$C_j = E_k(M_j \oplus A_j)$' 로 표현 가능하다. 특히 이 모드는 CTR모드와 같이 Counter를 이용하는 방식이지만, Counter의 특성상 대다수의 비트가 변하지 않는 단점을 가진 CTR 모드를 보완하여, 각 블록마다 ASR의 출력에 따라서 다수의 비트가 변화하는 구조를 가지므로 안정성이 더욱 향상되었을 뿐 아니라 Test결과 암호화시간도 단축 되었음을 알 수 있었다.

제안한 ASR 모드는 안정성이 뛰어나고, Random Access가 가능할 뿐 아니라 수행속도 또한 빠르므로 앞으로 암호 알고리즘을 구성함에 있어 많이 사용될 수 있을 것으로 기대 된다.

또한 ASR 모드를 변화시킨 FASR 모드는 ASR모드의 암호화 과정을 그대로 따르면서, ASR 모드의 d값을 변경시켜 안전성을 더욱 강화시켰다. 그러나 FASR 모드는 ASR 모드의 최대 장점인 Random Access가 불편하게 구성이 되어있다. 그러나 불편한 Random Access라는 단점이 있지른 그만큼 보안성이 강화 되었기 때문에 강한 보안을 요구하는 부분에 많은 활용이 가능할 것으로 사료 된다.

[1] 박창수, “블록 암호 알고리즘 NSEED에 관한 연구”, 부경대학교 대학원 컴퓨터공학과 박사학위논문, 2007.
[2] 서명룡, “하드웨어에 적합한 AES알고리즘에 관한 연구”, 부경대학교 대학원 컴퓨터공학과 석사학위논문, 2006.
[3] 원동호, “현대 암호학”, 도서출판그린, 2003.
[4] 김길호, “데이터 의존 셀룰라 오토마타를 사용한 블록 암호 알고리즘”, 부경대학교 산업대학원 컴퓨터공학과 석사학위논문, 2002.
[5] Niels Ferguson, Bruce Schneier, "Practical Cryptography", John Wiley & Sons, Inc., 2003.
[6] 박창수, 조경연, “갈로이 선형 궤환 레지스터의 일반화”, 전자공학회 논문지 제 43권 CI편 제 1호, 2006

감사의 글

많은 기대를 안고 대학원 과정을 시작한지가 엊그제 같은 데 제가 벌써 논문 심사를 마치고 이런 감사의 글을 쓸 수 있게 된 것을 보면 시간은 참으로 빨리 흐른다는 것을 알 수 있습니다. 시간만 흐르면 무사히 졸업을 할 수 있을 것이라는 안일한 생각을 가지고 있던 저에게 논문이라는 것은 너무나 큰 두려움이 있었습니다. 그런 저에게 논문을 작성하는 순서부터 시작해서 논문의 내용까지 너무나도 많은 도움을 주신 분들이 계십니다.

먼저 제가 이 논문을 작성 할 수 있도록 모든 지원을 아끼지 않으며 항상 저를 믿고 아껴주고 이해해주었으며 힘이 들 때마다 저를 위로해주었던 세상에서 가장 사랑하는 어머님과 가족들, 또한 언제나 저와 함께 웃어주고 저와 함께 울어주었던 가장 아름다운 내 친구들에게 감사의 말씀을 전합니다.

또한, 회사 업무로 자주 학교에 얼굴을 비추지 못하는 저에게도 물심양면으로 지원을 아끼지 않으신 조경연 교수님, 논문을 작성함에 있어서 세세한 과정부터 시작하여 전체적인 내용까지 지도를 해주신 박창수 선배님, 논문을 작성하는 과정에 제가 무척 힘든 시기가 있었는데 그때마다 제게 용기를 북돋워 주시고 힘을 낼 수 있도록 도움을 주신 김길호 선배님께 먼저 감사의 말씀을 전합니다.

마지막으로, 회사에서 새롭게 시작한 업무에 적응을 하지도 못했던 제가 빈번히 논문과 학교를 핑계로 회사 업무에 약간 소홀한 부분이 있었음에도 불구하고 제가 논문에 집중 할 수 있도록 도와주신 과장님 이하 사무실 식구들께도 감사의 말씀을 전합니다.

저에게 있어서 부경대학교 2년간의 생활은 평생 잊지 못할 좋은 기억으로 남게 해주신 모든 분들 다시 한 번 감사의 말씀을 드립니다.

2009년 2월 양 상 근 드림