공 학 박 사 학 위 논 문

블록 암호 알고리즘 NSEED에 관한 연구

2007년 8월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

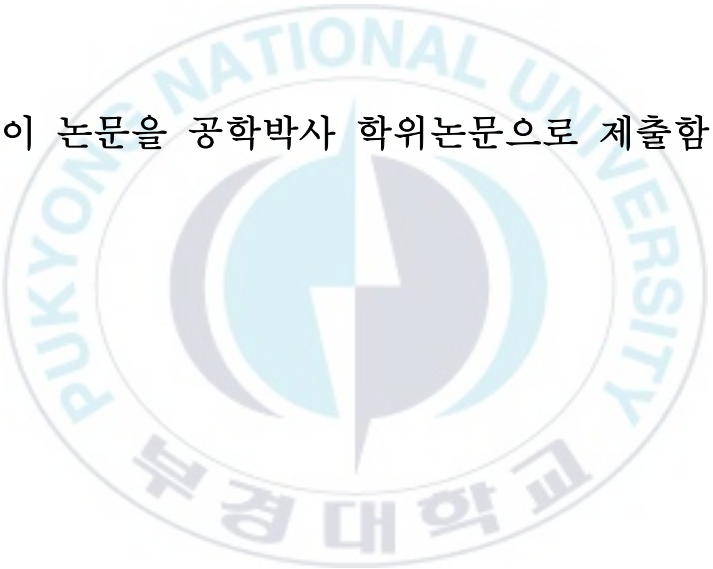
박 창 수

공 학 박 사 학 위 논 문

블록 암호 알고리즘 NSEED에 관한 연구

지도교수 조 경 연

이 논문을 공학박사 학위논문으로 제출함



2007년 8월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

박 창 수

박창수의 공학박사 학위논문을 인준함

2007년 6월 15일

주 심 이학박사 이 경 현 인

위 원 공학박사 정 목 동 인

위 원 공학박사 송 하 주 인

위 원 공학박사 송 홍 복 인

위 원 공학박사 조 경 연 인

이 논문을 사랑하는 가족에게
바칩니다.

목 차

그 림 목 차

표 목 차

A Study on Block Cipher Algorithm NSEED

Chang-Soo Park

Department of Computer Engineering, The Graduate School
Pukyong National University

Abstract

With the modern improvements in computer and information communication technologies, the computer network has been gradually expanding. As a result, a variety of information is being processed using the internet. Consequently, difficulties in information security are rising as a crucial issue. The advanced countries started to build and use cipher algorithms. In Korea, the KISA (Korea Information Security Agency) has been leading the development of SEED, a 128 bit block cipher algorithm.

In this dissertation, we will analyze SEED, the standard 128 bit block cipher algorithm, suggest a new composition that complements the disadvantages by using a new S box and G function, and propose the use of NSEED, the block cipher algorithm that uses S box and the G function.

NSEED has a Feistel network structure, and inputs a 128 bit plain text to yield a 128 bit coded text. The keys can be selected between 128 bits and 256 bits. The number of execution rounds is 4 rounds or 6 rounds, depending on the key. The F function, or the round function, is formed by connecting three series of G functions. The inner function G is composed of a SPS (Substitution Permutation Substitution) function. The Substitution Layer is

arranged into four S boxes, and the permutation layer uses a proliferating sector conversion queue to proliferate.

The S box is composed of a nonlinear function and an affine transformation. The nonlinear function has a unique feature using the differential analysis method and the sector analysis method. It is composed of a fixed point which has the same value for the input and output (with the exception of '0'and '1'), and a $GF(2^8)$ reciprocal which does not have an unfixed point where the output is the complement of 1 of the input. The affine transformation is structured so as to minimize the correlation of input and output without a fixed or unfixed point. The G function uses the $GF(2^8)$ 4×4 determinant to proliferate and transform the four S boxes.

The G function is structured to produce a MDS (Maximum Distance Separable) code, satisfy the SAC (Strict Avalanche Criterion) in order to improve the disseminating capacity, ensure no weak inputs are given in which the fixed point, unfixed point and output are not a complement of 2, and simplify the circuit when embodied into a hardware.

The NSEED, composed of the S box and the G function, has a maximum differential probability and sector probability of 2^{-6}, which makes it stronger in differential and sector analysis methods. Moreover, the invasion probability is 2^{-180} and 2^{-300} respectively, when using 128 and 256 bits. This level is determined to be sufficiently secure. When embodied as software, fewer S boxes pass all rounds, which speeds the processing rate. It is easily embodied into hardware, has fewer weak signals, and has a superior disseminating capacity.

The NSEED block cipher algorithm suggested in this thesis is expected to be

useful in systems with a constraint on the hardware structure such as the smart card and RFID. Also, the newly suggested S box and G function structure can be utilized in coded structure, which requires a high level of security.

제 1 장 서 론

현대에는 컴퓨터와 정보통신 기술의 비약적 발전으로 컴퓨터 통신망의 보급이 대단히 많이 이루어졌다. 특히 1990년대 이후에 더욱 빠른 속도로 발전하였고, 이제는 고도의 정보화 사회에 접어 들었다고 할 수 있다. 정보화 사회란 컴퓨터와 정보통신 기술의 결합으로 정보의 저장, 처리, 전송 능력이 획기적으로 증대되면서 정보의 가치가 산업 사회에서 매우 중요해지는 사회를 말한다[1]. 정보화 사회가 됨으로써 종이문서를 이용하여 정보를 처리하던 방식에서 벗어나 컴퓨터와 정보통신을 이용하는 방식으로 변화하였다. 이러한 변화는 많은 양의 정보를 먼 곳으로 쉽게 전달할 수가 있어, 적절한 정보보호 조치가 없으면 전송 도중, 또는 저장장치에 저장된 상태에서 불법적으로 유출되거나 삭제, 수정될 수가 있어 정보보호가 필수의 과제로 대두되었다.

컴퓨터와 정보통신을 이용하여 정보를 처리할 때 정보를 보호하는 가장 좋은 방법은 암호화를 이용하는 것이다. 암호(Cryptography)는 현대에 들어와서 사용된 것이 아니라 고대부터 사용되어 왔다. 고대에 사용된 암호는 주로 문자를 전치(Transposition) 시키거나 다른 문자로 치환(Substitution)하는 방법을 사용하여 전달하려는 정보가 들어있는 평문(Plaintext)을 암호문(Ciphertext)으로 변환하였다. 치환을 이용한 최초의 암호는 로마 시대에 줄리어스 시저(Julius Caesar)가 사용한 시저 암호(Caesar Cipher)이다[2].

근대에는 수학이 발달한 결과, 암호에도 수학이 접목되면서 암호학이

라는 학문의 분야가 새롭게 발달하기 시작하였다. 근대 암호의 기초는 C. E. Shannon[3]이 1940년대 후반에 확률을 이용한 정보이론을 발표함으로써 성립되었다. 그리고 이 시기의 컴퓨터의 등장은 암호 연구에 새로운 계기가 되었다. 즉, 암호화(Encryption) 과정과 암호문을 다시 평문으로 변환하는 복호화(Decryption) 과정을 컴퓨터를 이용하여 수행하게 된 것이다. 따라서 암호화 과정과 복호화 과정의 수행 속도가 빨라지게 되어 암호 해독도 더욱 쉬워졌다. 그러므로 더욱 복잡한 암호 체계가 필요하였고, 이에 관련된 연구도 활발해졌다.

암호 체계는 크게 대칭 암호 체계(Symmetric Cipher System)와 공개키 암호 체계(Public-key Cipher System)로 분류된다. 대칭 암호 체계는 고대부터 사용한 전치나 치환을 이용한 암호 체계에 Shannon의 이론을 접목 발전시킨 것으로 비밀키 암호 체계(Secret-key Cipher System)라고도 한다. 이 암호 체계는 암호화, 복호화할 때 동일한 키를 사용한다. 그래서 송신자와 수신자가 암호화된 문서를 서로 교환하기 전에 키를 먼저 교환하여 비밀리에 보관하여야 한다. 이것은 동작 속도가 빠르고, 키 크기가 상대적으로 작으며, 역사가 길다는 장점을 가진 반면에 정보를 교환하려는 상대방과 사전에 키를 공유해야 하고, 정보 교환자가 많아질수록 키의 수가 늘어나며, 키를 자주 바꾸어 주어야 하는 단점도 가지고 있다[4].

대칭 암호 체계는 데이터를 변환하는 방법에 따라 블록 암호(Block Cipher)와 스트림 암호(Stream Cipher)로 나누어진다. 블록 암호 알고리즘은 암호화할 때 고정된 크기의 입력을 받아서 고정된 크기의 출력으로 변환하는 암호 알고리즘이다. 복호화를 수행할 때는 암호화의 역과정을 수행한다. 스트림 암호는 암호화할 평문을 이진 유한 수열로 나타내고 키를 이

용해서 키 수열을 생성하고, 그 다음 이 두 수열의 배타적 논리합(XOR) 연산으로 암호문을 형성하는 암호 알고리즘이다. 스트림 암호는 비트 단위로 암호화를 진행하므로 에러 전파가 없고, 속도가 빠르다는 장점을 가지고 있으며, 1970년대부터 유럽을 중심으로 연구가 활발히 진행되고 있다. 키 수열은 의사 난수 발생기를 사용하여 생성한다. 이러한 의사 난수 발생기로 많이 사용되는 것이 선형 궤환 시프트 레지스터(LFSR, Linear Feedback Shift Register)이다. 많이 사용되는 선형 궤환 시프트 레지스터로는 갈로이 선형 궤환 시프트 레지스터(Galois Linear Feedback Shift Register)와 피보나치 선형 궤환 시프트 레지스터(Fibonacci Linear Feedback Shift Register)가 있다[5, 6]. 선형 궤환 시프트 레지스터가 많이 사용되는 이유는 구조가 간단하고, 동작속도가 빠르며 수학적으로 잘 정의되어 있기 때문이다.

현대 암호 연구의 새로운 계기는 공개키 암호 체계가 소개되면서부터라고 할 수 있다. 공개키 암호 체계는 다른 명칭으로 비대칭 암호 체계(Asymmetric Cipher System)라고도 한다. 이 암호 체계는 1976년 Diffie와 Hellman이 처음으로 제안한 것으로 암호화, 복호화할 때 단일 키를 사용하는 대칭 암호 체계와는 달리 키를 쌍으로 가지고 암호화, 복호화를 수행한다[7]. 다시 말하면 암호화 시에 사용한 키는 공개를 하고 복호화 시에 사용하는 키는 비밀리에 보관하는 것이다. 공개키 암호 체계는 사용자마다 공개하는 키와 비밀로 하는 키를 가지므로 여러 사람들과 키를 교환하여 공유할 필요가 없으므로 대칭 암호 체계에서 문제가 되었던 키 공유 문제를 해결하는 계기가 되었다.

위와 같이 정보를 보호하기 위한 암호에 관한 연구가 최근에 들어오면

서 더욱 활발해지고 있으며 전세계 많은 나라들이 국가 표준 암호 체계를 만들고 있다. 미국은 1977년 국가 표준 암호 체계로 블록 암호 알고리즘 구조로 되어 있는 DES(Data Encryption Standard)[8]를 채택하였다. DES는 64 비트 블록으로 나누어서 암호화를 하고, 56개의 키를 가지며 피스탈(Feistel) 구조를 채택하고 있는 블록 암호이다. 이것은 미국 연방정부의 데이터 보호용으로 출발하여 금융 및 다른 상업용으로 널리 사용되어 세계 표준 암호 알고리즘으로 인식되었다. 1980년대를 지나면서 DES의 특성을 분석하는 연구가 활발해졌고, 컴퓨터의 성능이 좋아지게 되자 암호 알고리즘의 성능을 보완하고 좀더 복잡하게 만든 알고리즘들이 개발되었는데, 미국의 삼중 DES(Triple DES), 유럽의 IDEA(International Data Encryption Algorithm)[9], 일본의 FEAL(Fast Data Encipherment Algorithm)[10] 등이 대표적이다. 그리고 스트림 암호로 실용화되어 널리 사용된 것으로는 RC4[11]가 있다.

공개키 암호 체계로써 처음으로 실용화된 알고리즘은 1978년 Rivest, Shamir, Adleman이 제안한 RSA[12]이다. 이 알고리즘은 아주 큰 정수의 인수분해 문제에 기초를 둔 것이다. 그 외 다른 알고리즘으로는 1985년에 발표된 유한체 위의 이산대수 문제의 어려움에 기초를 둔 ElGamal[13] 과 최근에 제안된 유한체 위에서 정의된 타원곡선 상에서의 이산대수 문제에 기초를 둔 타원 곡선 암호 체계[14]가 있다.

암호 연구가 활발히 진행되는 것만큼이나 컴퓨터의 성능도 나날이 발전하였다. 그 결과 1998년 DES가 해독, 공개되어 DES는 표준 암호 알고리즘으로서의 그 생명을 다하게 되었다. 따라서, 새로운 암호 알고리즘이 필요하게 된 미국은 전 세계적으로 AES(Advanced Encryption

Standard)[15]를 공모하였다. 그 결과 공개 경쟁을 통해 2000년에 AES로 J. Daemen 과 V. Rijmen 이 제안한 Rijndeal이 선정되었다[16].

암호 알고리즘에 대한 대부분의 연구는 미국, 유럽 국가 등 몇몇 암호 선진국에서 주도하고 있으며, 정보의 유출을 방지하기 위하여 암호 기술 및 제품에 대한 수출을 규제하고 있다. 이에 자국의 정보를 보호하기 위해 각 나라들은 자체적인 암호 알고리즘을 연구하고 있으며, 우리 나라에서는 한국정보보호진흥원이 주축이 되어 관련 전문가들과 공동으로 128 비트 블록 암호 알고리즘인 SEED[17, 18, 19, 20, 21]를 개발하여 공개하였다.

SEED는 16회전을 수행하는 피스탈 네트워크(Feistel Network) 구조로, 64 비트 평문을 2개의 32 비트 평문으로 나누어 주고, 하나의 32 비트 블록은 G 함수를 사용하여 변환한다. 이 단계를 거친 후에 변환한 32 비트 블록과 변환하지 않은 32 비트 블록을 연산하여 32 비트 출력을 얻는다. SEED에서 하나의 F 함수는 위의 과정을 3회 반복하여 구성한다. 여기서 G 함수는 4개의 8 비트 S 박스와 S 박스 출력을 선형 변환하는 부분으로 구성되어 있는 32 비트 치환 블록이다.

SEED의 G 함수는 4개의 S 박스의 출력을 간단한 수식으로 결합하여 입력의 변화를 출력에 확산시키므로 충분한 확산이 이루어지지 않는 단점이 존재한다.

본 논문에서는 이러한 SEED의 단점을 개선하기 위하여 먼저, 블록 암호 알고리즘을 공격하는 가장 뛰어난 공격 방법인 차분 분석법과 선형 분석법에 강한 특성을 가지기 위해 안전성이 높은 S 박스를 구성한다. 차분 분석법에 강한 특성을 지닌 S 박스를 구성하기 위한 세 가지 조건은 아래와 같다[22, 23, 24].

- 차분 XOR 테이블에서 모든 첫 번째 컬럼이 '0'이 되어야 한다.
- '0'인 성분의 수가 작아야 한다.
- 최대 성분 값이 작아야 한다.

첫째 조건을 만족하기 위해서는 S 박스는 전단사함수가 되어야 하며, 둘째 조건을 만족하기 위해서는 '0' 또는 '2'인 성분이 많아야 한다. 끝으로 셋째 조건을 만족하기 위해서는 최대성분 값이 '4'인 전단사함수를 선정해야 한다. 선형 분석법에 강한 S 박스를 구성하기 위해서는 입력과 출력의 상관계수가 작아야 한다.

위 조건을 만족하도록 S 박스는 비선형함수와 아핀 변환으로 구성한다. 비선형함수는 차분 분석법[25]과 선형 분석법[26]에 강한 $nl(X) \Rightarrow X^{-1}$ over $GF(2^8)$ 변환[27]을 채택하고, 원시 다항식은 '0'과 '1' 이외의 약한 입력을 가지지 않으면서 입출력의 상관계수가 작은 것을 선정한다. 아핀 변환은 차분 분석법과 선형 분석법의 특성에 영향을 주지는 않지만 [28], 수식의 복잡도를 증가 시켜서 보간공격(Interpolation Attack)[29]에 강하도록 한다. 본 논문에서는 입력과 출력이 같거나 출력이 입력의 1의 보수가 되는 약한 입력을 가지지 않으면서 입출력의 상관계수가 가장 작은 아핀 변환을 구성한다.

다음으로 F 함수의 특성을 분석하여 G 함수는 전단사함수가 되어야 하고, MDS(Maximum Distance Separable) 코드[30, 31, 32]를 생성해야 하며, SAC(Strict Avalanche Criterion)[33]을 충족시켜야 함을 제시한다. 그리고 G 함수의 입력 가운데 고정점과 역고정점 및 출력이 입력의 2의 보수가 되는

입력이 약한 입력임을 보인다. 그리고 이러한 G 함수의 특성을 고려하여 아래 4가지 조건을 만족하는 G 함수를 생성하는 알고리즘을 제안한다.

- MDS 코드를 생성한다.
- SAC을 만족하여 확산 특성이 우수하도록 한다.
- 약한 입력을 가지지 않는 전단사함수이다.
- 하드웨어 구현이 용이하다.

본 논문에서는 위에서 제시한 결과를 바탕으로 새로 생성한 S 박스와 G 함수를 사용하여 SEED를 수정한다. 그리고 수정한 블록 암호 알고리즘을 NSEED라고 명명한다. NSEED는 NEW SEED라는 의미이다.

본 논문에서 제안하는 NSEED 블록 암호 알고리즘은 SEED, ARIA[34] 그리고 AES와 비교했을 때 적은 라운드를 수행하고도 동등한 안전성을 지니며, 소프트웨어로 구현 시에도 더 빨리 동작한다. 그리고 하드웨어로의 구현도 용이하다. 또한 본 논문에서 제안하는 S 박스와 G 함수는 차분 분석법과 선형 분석법에 강하고, 약한 입력이 없으며, 확산 특성이 우수하므로 비교적 안전성이 높은 암호 방식의 구성 요소로 활용할 수 있다.

본 논문의 구성은 2장에서 관련 연구로 SEED, ARIA, 그리고 AES 알고리즘을 소개하고, 블록 암호 알고리즘에 대한 공격법을 알아보며, 3장에서는 MDS 코드 생성 확인 방법을 알아본다. 4장에서는 산술 시프트 레지스터를 소개하고, 5장에서는 S 박스의 구성과 G 함수의 특성을 분석하여 G 함수를 생성하며, 6장에서는 본 논문에서 제안하는 NSEED 블록 암호 알고리즘의 구조를 소개하고 성능 평가를 한다. 7장에서는 NSEED 블록

암호 알고리즘의 안전성을 평가하며, 마지막으로 8장에서 결론을 맺는다.

제 2 장 관련 연구

이 장에서는 본 논문의 연구와 관련이 되는 블록 암호 알고리즘들을 소개한다. 그리고 블록 암호알고리즘을 공격하기 위한 가장 좋은 방법으로 알려진 차분 분석법과 선형 분석법에 대해 알아보고, 그 외 관련된 내용들도 소개한다.

2.1 SEED 암호 알고리즘

현대에 들어오면서 컴퓨터와 정보통신이 비약적으로 발전하고 인터넷이 대중화되었다. 그 결과 일상 생활의 많은 활동들이 컴퓨터와 인터넷을 통해서 이루어지게 되어, 개인의 중요한 정보나 문서 등이 유출되지 않도록 정보를 보호하는 것이 중요한 문제로 대두되었다. 그래서 선진국을 비롯한 많은 나라들이 고유의 암호 알고리즘을 연구, 개발하여 사용한다. 이에 우리 나라도 정보보호진흥원이 주도하여 SEED 암호 알고리즘을 개발하여 공개하였다. 이 절에서는 SEED 알고리즘에 대해서 알아보겠다.

2.1.1 암호화 과정

SEED는 대칭키 암호 알고리즘으로써 블록 단위로 메시지를 처리하는

블록 암호 알고리즘이다. 전체 구조는 피스탈 네트워크(Feistel Network) 구조로 이루어져 있다. 그리고 암호화를 위한 입력으로 128 비트의 평문을 입력 받고, 128 비트의 키로부터 생성된 64 비트의 라운드 키를 입력 받아 총 16 라운드를 거쳐 128 비트 암호문을 생성한다. SEED의 전체 구조도는 그림 1과 같다.

SEED는 128 비트의 평문을 64 비트씩 R_0, L_0로 나누어서 입력을 받는다. 우측 64 비트인 R_0는 128 비트 키로부터 생성된 64 비트의 라운드 키와 함께 F 함수에 입력된다. 또한 다음 라운드의 좌측 입력인 L_1이 된다. 좌측 64 비트인 L_0는 F 함수의 출력인 $F(R_0, K_1)$과 비트 별로 배타적 논리합(Exclusive-Or) 연산을 수행하여 다음 라운드의 우측 입력인 R_1이 생성된다. 이 과정이 암호화를 수행할 때 한 라운드이다. SEED는 이 과정을 총 16회 반복 수행하여 출력 암호문을 생성한다. 전 암호화 과정을 대수적으로 표현하면 식 (1)과 같이 된다.

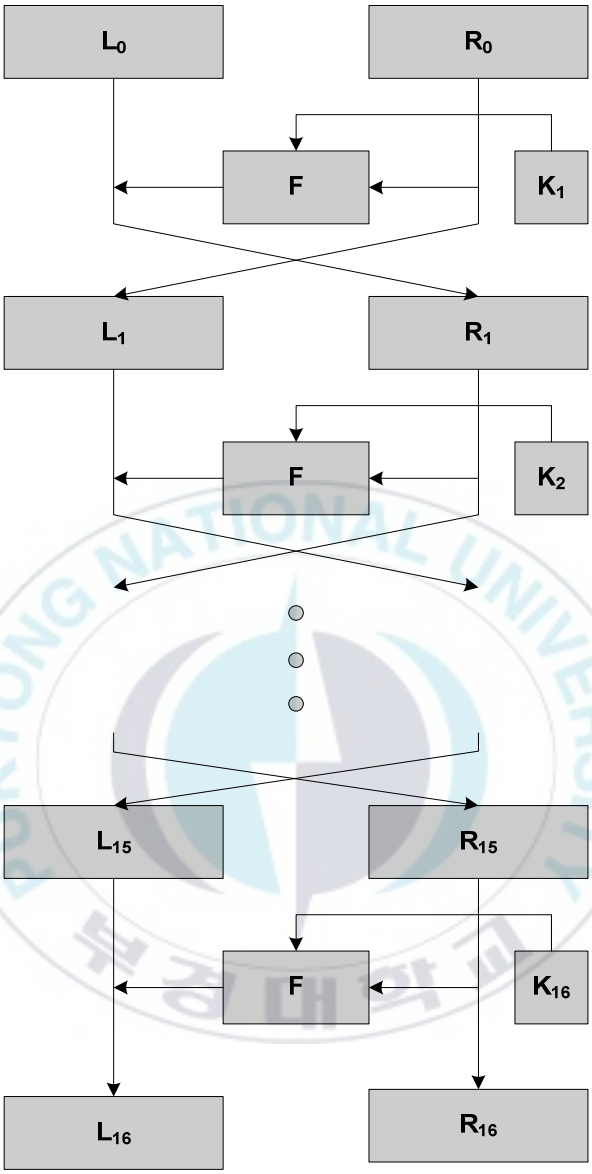


그림 1. SEED 전체 구조도

$$
\begin{aligned}
R_1 &= F\left(R_0, K_1\right) \oplus L_0 \\
L_1 &= R_0 \\
R_2 &= F(R_1, K_2) \oplus L_1 \\
L_2 &= R_1 \\
R_3 &= F(R_2, K_3) \oplus L_2 \\
L_3 &= R_2 \\
&\vdots \\
R_{15} &= F(R_{14}, K_{15}) \oplus L_{14} \\
L_{15} &= R_{14} \\
R_{16} &= F(R_{15}, K_{16}) \oplus L_{15} \\
L_{16} &= R_{15}
\end{aligned}
\qquad (1)
$$

2.1.2 F 함수

F 함수는 64 비트의 입력을 받아 각 32 비트 2개의 블록 $(R_0,\ R_1)$으로 나누고, 64 비트 라운드 키도 2개의 32 비트 라운드 키로 나누어 입력 받아서 배타적 논리합(Exclusive-Or), 32 비트 모듈로 연산을 수행하고, G 함수 처리를 하여 2개의 32 비트 블록 $(R_0',\ R_1')$을 출력한다. F 함수의 구조도는 아래 그림 2와 같다.

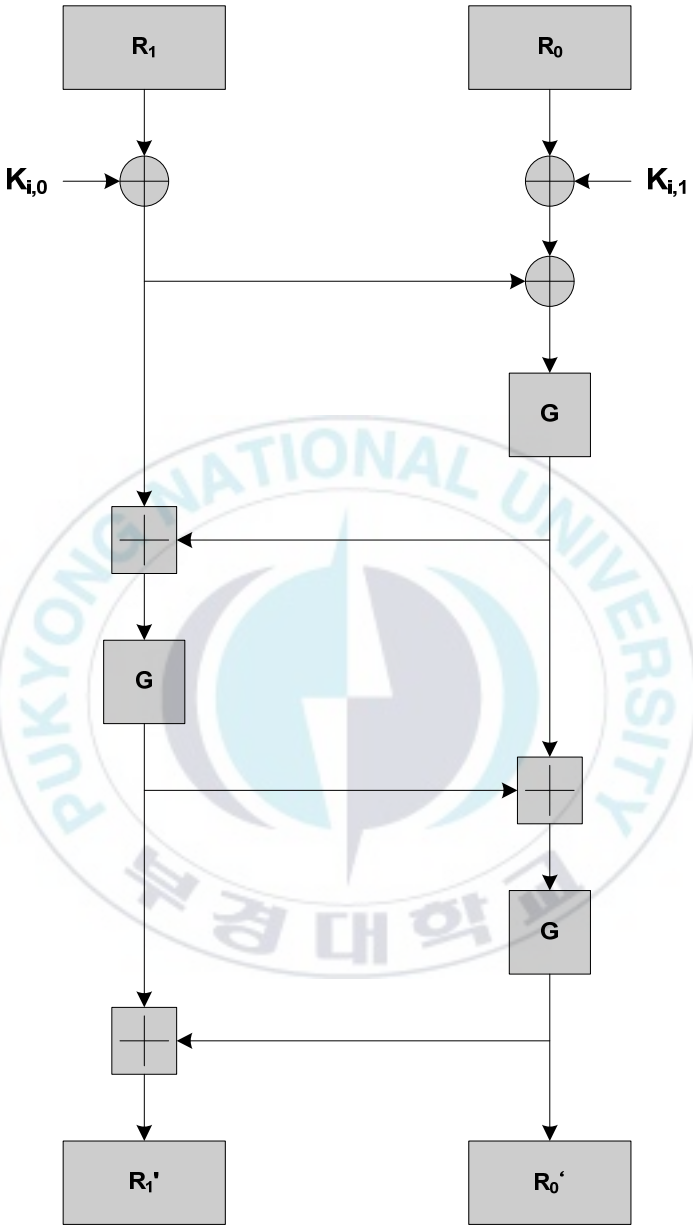


그림 2. F 함수 구조도

F 함수도 두 부분으로 나누어져서 동작을 한다. 먼저, 2개의 32 비트 블록 R_0, R_1은 64 비트 라운드 키를 32 비트씩 둘로 나눈 k_{i0}, k_{i1}과 각각 배타적 논리합(Exclusive-Or) 연산을 수행하여 $R_0 \oplus k_{i0}$, $R_1 \oplus k_{i1}$을 형성한다. 그 다음에 이 값을 다시 배타적 논리합(Exclusive-Or) 연산을 수행하고 G 함수 처리를 한다. 여기까지의 결과가 $G\{(R_0 \oplus k_{i0}) \oplus (R_1 \oplus k_{i1})\}$이다. 이 값과 $R_1 \oplus k_{i1}$을 모듈로 32 덧셈을 수행한 후 그 결과를 두 번째로 G 함수 처리를 한다. 첫 번째 G 함수 처리를 한 결과와 두 번째 G 함수 처리를 한 결과를 두 번째로 모듈로 32 덧셈을 수행하고, 결과를 세 번째로 G 함수 처리를 한다. 이 결과가 F 함수의 한 부분 출력인 R_1'이다. 두 번째 G 함수 처리 결과와 세 번째 G 함수 처리 결과를 세 번째로 모듈로 32 덧셈을 수행한다. 이 결과가 F 함수의 다른 한 부분 결과인 R_0'이다. F 함수의 출력은 식 (2)로 표현된다.

$$
\begin{aligned}
R_0' = G\Big[(R_0 \oplus k_{i0}) \boxplus G\big\{(R_1 \oplus k_{i1}) \oplus (R_0 \oplus k_{i0})\big\}\Big] \boxplus G\Big[\big\{G(R_1 \oplus k_{i1}) \oplus \\
(R_0 \oplus k_{i0})\big\} \boxplus G\Big[G\big\{(R_1 \oplus k_{i1}) \oplus (R_0 \oplus k_{i0})\big\} \boxplus G\Big[(R_0 \oplus k_{i0}) \boxplus \\
G\big\{(R_1 \oplus k_{i1}) \oplus (R_0 \oplus k_{i0})\big\}\Big]\Big]\Big] \\
R_1' = G\Big[G\big\{(R_1 \oplus k_{i1}) \oplus (R_0 \oplus k_{i0})\big\} \boxplus G\Big[(R_0 \oplus k_{i0}) \boxplus G\big\{(R_1 \oplus k_{i1}) \oplus \\
(R_0 \oplus k_{i0})\big\}\Big]\Big]
\end{aligned}
\tag{2}
$$

2.1.3 G 함수

G 함수는 32 비트 입력을 4개의 8 비트 블록 (X_3, X_2, X_1, X_0)으로 분할하여 2 개의 S 박스를 (S_2, S_1, S_2, S_1) 순서로 적용시켜 다음과 같이 출력을 생성한다.

$$Y_3 = S_2(X_3), \quad Y_2 = S_1(X_2)$$
$$Y_1 = S_2(X_1), \quad Y_0 = S_1(X_0)$$

S 박스의 출력에 마스킹 바이트(masking byte) $m_3 m_2 m_1 m_0$을 정하여 아래와 같이 연산을 하여 G 함수의 최종 출력 $Z_3 Z_2 Z_1 Z_0$을 생성한다.

$$Z_3 = (Y_0 \,\&\, m_3) \oplus (Y_1 \,\&\, m_0) \oplus (Y_2 \,\&\, m_1) \oplus (Y_3 \,\&\, m_2)$$
$$Z_2 = (Y_0 \,\&\, m_2) \oplus (Y_1 \,\&\, m_3) \oplus (Y_2 \,\&\, m_0) \oplus (Y_3 \,\&\, m_1)$$
$$Z_1 = (Y_0 \,\&\, m_1) \oplus (Y_1 \,\&\, m_2) \oplus (Y_2 \,\&\, m_3) \oplus (Y_3 \,\&\, m_0)$$
$$Z_0 = (Y_0 \,\&\, m_0) \oplus (Y_1 \,\&\, m_1) \oplus (Y_2 \,\&\, m_2) \oplus (Y_3 \,\&\, m_3)$$

여기서, $m_0 = 0xfc, m_1 = 0xf3, m_2 = 0xcf, m_3 = 0x3f$

& 는 bit-wise AND

G 함수의 구조는 그림 3과 같다.

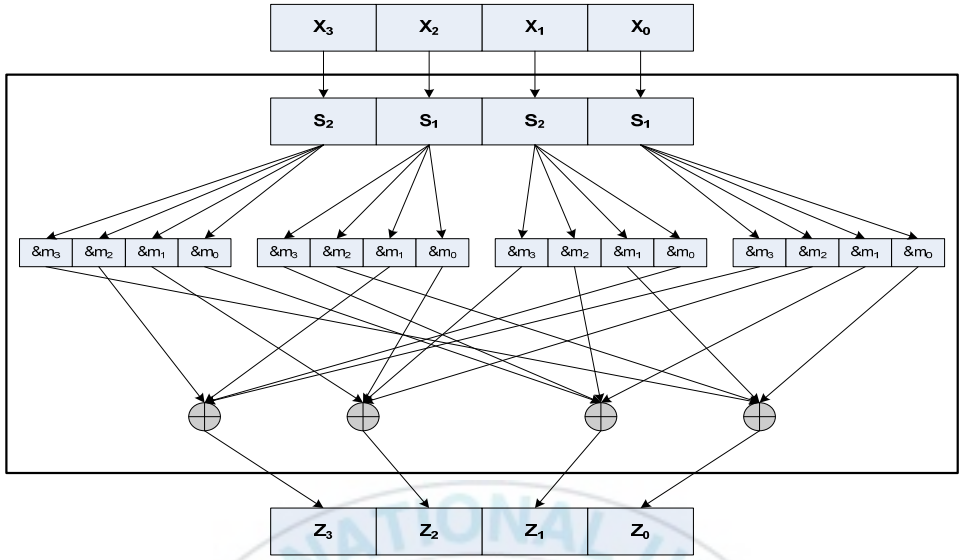


그림 3. G 함수 구조도

위의 G 함수는 4개의 확장된 4 바이트 SS 박스들(4Kbytes)의 배타적 논리합(Exclusive-Or)으로 구현할 수 있다. 이 때는 다음과 같은 4개의 SS 박스들을 저장해야 한다.

$$SS_0 = S_1(x_0)\,\&\,m_3 \,\|\, S_1(x_0)\,\&\,m_2 \,\|\, S_1(x_0)\,\&\,m_1 \,\|\, S_1(x_0)\,\&\,m_0$$
$$SS_1 = S_2(x_1)\,\&\,m_0 \,\|\, S_2(x_1)\,\&\,m_3 \,\|\, S_2(x_1)\,\&\,m_2 \,\|\, S_2(x_1)\,\&\,m_1$$
$$SS_2 = S_1(x_2)\,\&\,m_1 \,\|\, S_1(x_2)\,\&\,m_0 \,\|\, S_1(x_2)\,\&\,m_3 \,\|\, S_1(x_2)\,\&\,m_2$$
$$SS_3 = S_2(x_3)\,\&\,m_2 \,\|\, S_2(x_3)\,\&\,m_1 \,\|\, S_2(x_3)\,\&\,m_0 \,\|\, S_2(x_3)\,\&\,m_3$$

여기서, $\|$ 는 concatention

이 확장 SS 박스들을 이용하면 G 함수는 다음과 같이 구현할 수 있다.

$$Z = SS_0(x_0) \oplus SS_1(x_1) \oplus SS_2(x_2) \oplus SS_3(x_3)$$

2.1.4 S 박스

S 박스는 8 비트가 입력되어 8 비트가 출력되는 치환과정으로서 G 함수에서 사용되며 S_1, S_2 두 개를 사용한다. S 박스는 전단사함수 $x^n\,(0 \le n \le 255)$에서 차분 분석법(Differential Cryptanalysis) 및 선형 분석법(Linear Cryptanalysis) 특성이 가장 우수한 두 개의 n 값 247, 251을 선택하고, $GF(2^8)$상에서 지수승을 구하기 위하여 $GF(2^8)$상에서의 모든 원소를 원시원소 α의 멱승으로 표현한다. 이 때 사용한 원시원소는

$$\alpha = p(x) = x^8 + x^6 + x^5 + x + 1$$

이다.

지수 함수의 입력 $x = 0$이 출력 0, 입력 $x = 1$이 출력 1로 고정되는 것을 방지하기 위하여, 지수 함수를 $S(x) = (A \bullet x^n) \oplus b$로 선형변환한다. 이 때 선형변환행렬 A는 다음과 같다.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

각 S 박스에 사용된 행렬 $A^{(1)}$, $A^{(2)}$는 A의 행을 교환하여 사용한다. $A^{(1)}$은 A의 2행과 3행을 교환하고, 4행과 6행을 교환한다. $A^{(2)}$는 A의 1행과 5행을 교환하고, 6행과 7행을 교환한다. A는 행과 열의 수가 같은 정칙 행렬(Nonsingular matrix)이므로 서로 다른 두 개의 입력 x, x'에 대하여 $Ax \neq Ax'$이다.

다음으로 b는 다음과 같다.

$$b_1 = (1,0,1,0,1,0,0,1)^T = 169, \quad b_2 = (0,0,1,1,1,0,0,0) = 56$$

두 개의 S 박스는 아래와 같이 구성한다.

$$\left(A^{(1)} \bullet x^{247}\right) \oplus b_1 = \left(P_7, P_6, P_5, P_4, P_3, P_2, P_1, P_0\right)^T \rightarrow$$
$$S_1(x) = P_7 \times 2^7 + P_6 \times 2^6 + P_5 \times 2^5 + P_4 \times 2^4 + P_3 \times 2^3 + P_2 \times 2^2 + P_1 \times 2^1 + P_0$$
$$\left(A^{(2)} \bullet x^{251}\right) \oplus b_2 = \left(Q_7, Q_6, Q_5, Q_4, Q_3, Q_2, Q_1, Q_0\right)^T \rightarrow$$
$$S_2(x) = Q_7 \times 2^7 + Q_6 \times 2^6 + Q_5 \times 2^5 + Q_4 \times 2^4 + Q_3 \times 2^3 + Q_2 \times 2^2 + Q_1 \times 2^1 + Q_0$$

단, $x = \left(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0\right)$ 이고, x_i는 2^i의 계수임

2.2 ARIA 블록 암호 알고리즘

2.2.1 알고리즘 구조

ARIA 알고리즘의 구조를 요약 하면 아래와 같다.

- 기본 구조는 ISPN(Involution SPN) 구조이다.
- 입·출력문의 크기는 128 비트이다.
- 키의 크기는 128 비트, 192 비트, 256 비트이다.
- 라운드 키의 크기는 128 비트이다.
- 라운드 수는 키의 크기에 따라 12, 14, 16 라운드이다.

이 사양을 8 비트 블록 단위로 나타내면 표 1과 같다. 여기서 입·출력 블록의 크기는 Nb, 입력 키 블록의 크기는 Nk, 그리고 라운드 수를 Nr로 나타내었다.

표 1. ARIA 사양

구분	Nb	Nk	Nr
ARIA-128	16	16	12
ARIA-192	16	24	14
ARIA-256	16	32	16

ARIA의 라운드 함수는 다음과 같은 세 부분으로 구성되어 있다.

1. 라운드 키 덧셈(AddRoundKey) 부분은 128 비트 라운드 키를 라운드 입력 128 비트와 비트별 XOR 연산을 수행한다.
2. 치환을 수행하는 치환 계층이 있다. 치환 계층은 두 가지 유형이 있으며, 각각은 2종의 8비트 입·출력 S 박스와 그들의 역변환으로 구성된다.
3. 확산 계층이 있는데, 이 계층은 간단한 16×16 involution 행렬을 사용한 바이트 간의 확산 함수로 구성되어 있다.

ARIA의 마지막 라운드는 확산 계층이 라운드 키 덧셈으로 대치된다. 암호화 과정과 복호화 과정은 라운드 키를 제외하고는 일치한다. ARIA의 암호화 과정과 복호화 과정은 그림 4와 같다.

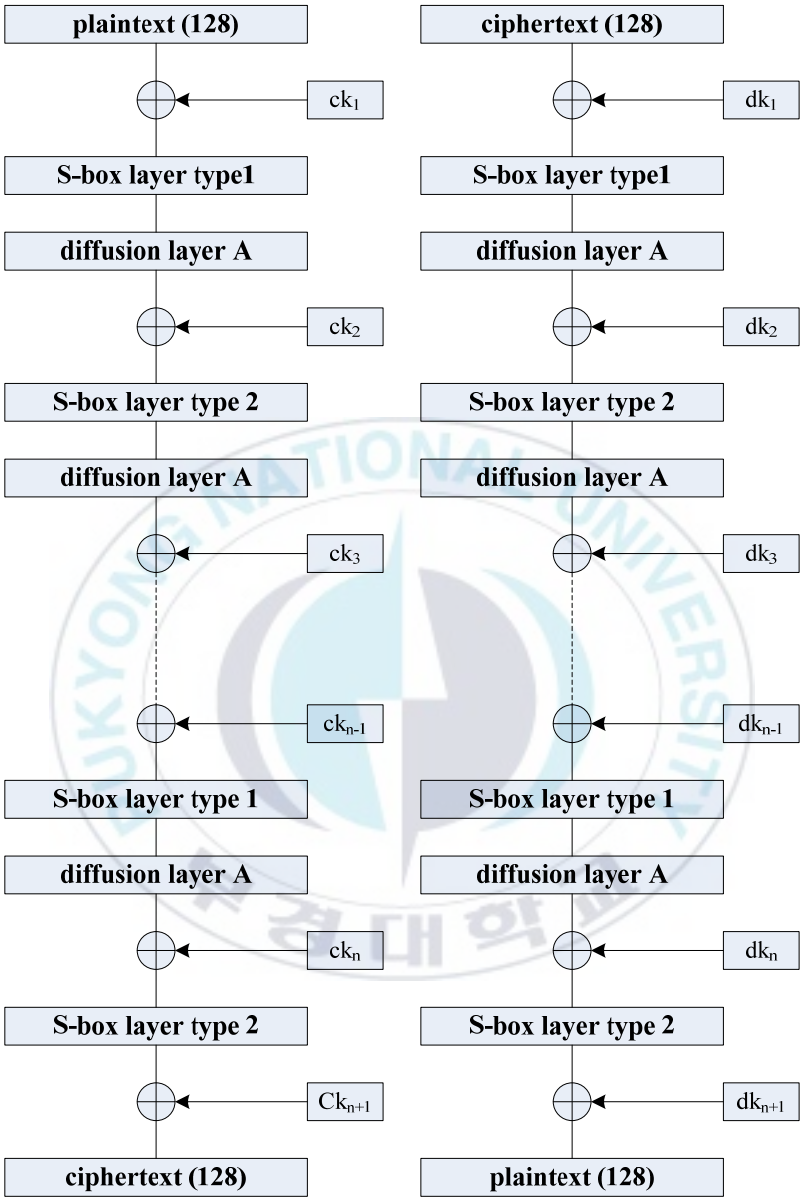


그림 4. ARIA의 암호화, 복호화 과정

2.2.2 치환 계층

치환 계층은 8 비트 입·출력 S 박스들로 구성되어 있다. 이 S 박스들은 다음의 성질을 만족하도록 선택되었다.

- 최대 차분/선형 확률이 2^{-6}
- 대수적 차수는 7
- 고정점, 반고정점이 없을 것

위의 성질을 만족하기 위해서 많이 사용되는 것이 유한체 $GF(2^8)$ 상의 함수 x^{-1}에 아핀 변환을 사용한 형태이다. ARIA에서는 x^{-1}와 x^{247}에 아핀 변환을 하여 S 박스를 생성하였다. 따라서, 두 S 박스 S_1과 S_2는

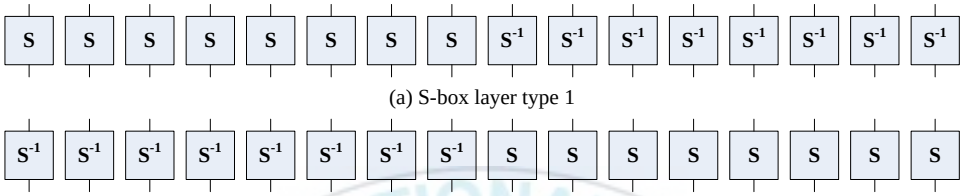
$$S_1(x) = Bx^{-1} \oplus b, \quad S_2(x) = Cx^{247} \oplus c$$

의 형태를 가진다. 여기서 B, C는 8×8 정칙행렬이며, b,c는 8×1 행렬이다. ARIA는 위에서 생성한 S_1, S_2와 그 역치환 S_1^{-1}, S_2^{-1}를 사용한다.

치환 계층은 두 유형으로 되어 있는데 다음 사항을 만족하도록 구성되어 있다.

- 두 유형 모두 S 박스 S_1, S_2, S_1^{-1}, S_2^{-1}로 구성
- 두 유형의 치환 계층은 서로 역의 관계 ((유형 1)$^{-1}$ = 유형 2)
- 32 비트 단위로 4 종류의 S 박스를 사용

두 유형의 치환 계층은 그림 5와 같다. 두 유형의 치환 계층은 교대로 사용된다. 즉, 유형 1은 홀수 번째 라운드에 사용되고, 유형 2는 짝수 번째 라운드에 사용된다. 전체 구조가 역으로도 동작이 가능한 구조로 되어 있다.



(a) S-box layer type 1

(b) S-box layer type 2

그림 5. 두 종류의 바이트 치환 계층

2.2.3 확산 계층

ARIA와 다른 블록 암호 알고리즘들과 차이가 나는 부분이 확산 계층이다. 이 부분은 역행렬이 가능한 16×16 이진 행렬을 사용한다. 확산 함수는 입력 16 바이트에 대해서 바이트 단위의 행렬 곱을 수행한 결과의 16 바이트를 출력으로 한다.

ARIA의 확산 계층의 설계 방향은 다음과 같다.

- AES에 강력한 분석 방법들에 대해서 내성을 가져야 한다. 이를 위해서 AES의 확산 함수 처리 단위(32 비트) 보다 큰 단위의 확산 함수를 사용한다.
- 8 비트, 32 비트 소프트웨어 및 하드웨어 구현에 적합해야 한다.
- 동종의 확산 함수 중에서 안전성과 효율성을 고려할 때 제일 우수

해야 한다.

위 설계 방향을 만족하도록 선택된 ARIA의 확산 함수 $A : GF(2^8)^{16} \rightarrow GF(2^8)^{16}$ 는 입력을 $(x_0, x_1, x_2, \cdots, x_{15})$ 라 하고 출력을 $(y_0, y_1, y_2, \cdots, y_{15})$라 하면, 아래 식과 같은 행렬의 곱으로 표현할 수 있다.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix}$$

위 식은 다음에 나타낸 것과 같은 성질을 가진다.

- A는 involution 구조이다. 즉, $A^{-1} = A$ 이다.
- A의 가지수(Branch number)는 8이다. 여기서 가지수 $\beta(A)$는 다음과 같이 정의된다.

$$\beta(A) = \min\left\{wt(x) + wt(A \bullet x) \mid x \in GF(2^8)^{16}, x \neq 0\right\}$$

여기서 $wt(x)$는 해밍 웨이트(Hamming Weight)이다.

2.3 AES(Advanced Encryption Standard)

AES는 미국 국립 표준 기술 연구소(NIST, National Institute Standards & Technology)에서 DES를 대체하기 위해 전 세계적으로 공모하였다. 그 후 공개 검사와 평가, 그리고 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, 암호화 키의 길이는 128 비트, 196 비트, 256 비트 중에서 선택해서 사용할 수 있다.

AES는 입력되는 평문을 4×4 행렬로 표현하여 연산을 수행하는데, 4×4 행렬로 표현된 암호화, 복호화 중간 과정을 상태(State)라 하며, 행렬의 각 열의 32 비트를 워드(Word)라 한다.

AES의 암호화 과정은 평문이 입력되면 각 라운드 동작이 수행되기 전에 평문과 라운드 키의 배타적 논리합 연산이 먼저 수행된다. 이 과정이 AddRoundKey() 연산이다. 다음부터는 각 라운드 마다 4가지 연산을 동일하게 수행하면서, 전체 라운드가 r이라면 r – 1 라운드까지 수행한다. 각 라운드 마다 수행되는 4가지 연산은 아래와 같다.

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

암호화 과정의 마지막 라운드는 위의 4가지 연산 중에서 MixColumns() 연산을 수행하지 않고, 세가지 연산만 수행하여 암호문이 출력된다. 암호화 과정은 그림 6과 같다.

AES는 사용하는 키의 길이에 따라 암·복호화가 수행되는 라운드 수가 달라진다. AES의 키 길이에 따른 블록 크기와 라운드 수는 표 2와 같다.

표 2. 키 길이와 블록 크기에 따른 라운드 수

	키 길이 (Nk words)	블록 길이 (Nb words)	라운드 수 (Nr)
AES - 128	4	4	10
AES - 196	6	4	12
AES - 256	8	4	14

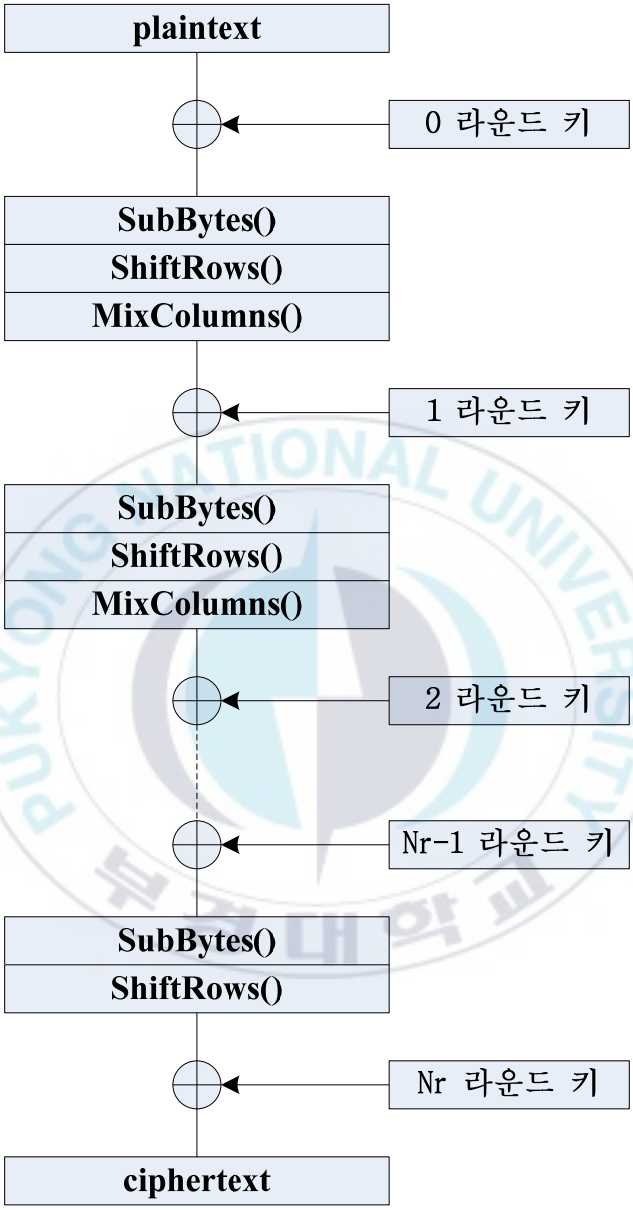


그림 6. AES의 암호화 과정

SubBytes() 연산은 S 박스를 사용하여 치환을 하는 단계이다. S 박스는 두 가지 연산을 하여 만들어지는데, 첫 번째 연산은 $GF(2^8)$에 대한 곱셈의 역원을 구한다. 입력이 00인 경우에는 자기 자신으로 치환한다. 두 번째 연산은 $GF(2)$ 상의 아핀 변환을 구하는 것이다. 아핀 변환은 그림 7과 같다.

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

그림 7. AES의 아핀 변환

SubBytes() 연산을 그림으로 나타내면 그림 8과 같다.

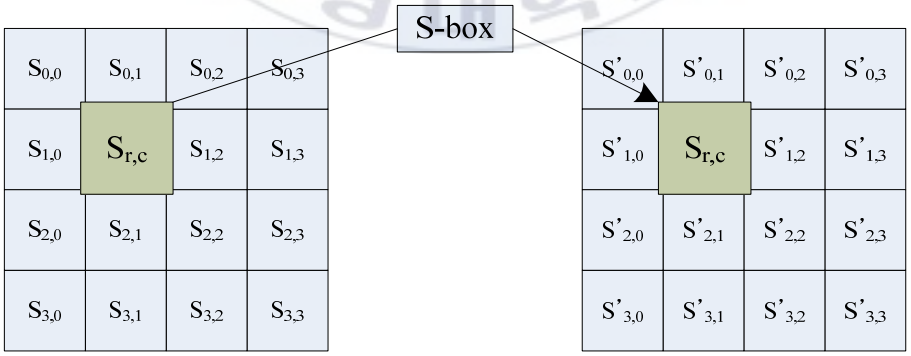


그림 8. SubBytes() 연산

이 변환은 가역이며, 역변환 테이블도 미리 만들어 복호에 사용한다.

ShiftRows() 연산은 4×4 행렬로 표시되는 상태의 각 행을 위에서부터 0, 1, 2, 3 만큼씩 좌로 순환시킨다. ShiftRows() 연산은 그림 9와 같다.

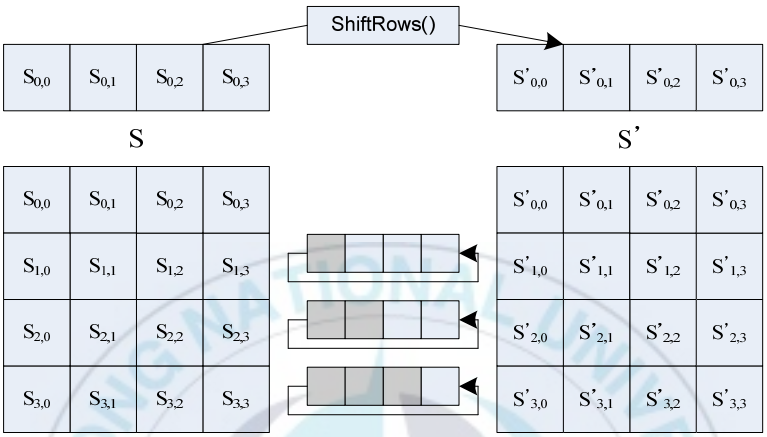


그림 9. ShiftRows() 연산

MixColumns() 연산은 상태의 각 열에 해당하는 바이트 블록들이 서로 영향을 받도록 변환시켜 주는 단계이다. 행렬의 각 열을 $GF(2^8)$ 상의 3차 이하 다항식으로 표현하여 $c(x) = 03x^3 + 01x^2 + 01x + 02$를 곱하고 $x^4 + 1$로 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 10과 같이 나타낼 수 있다.

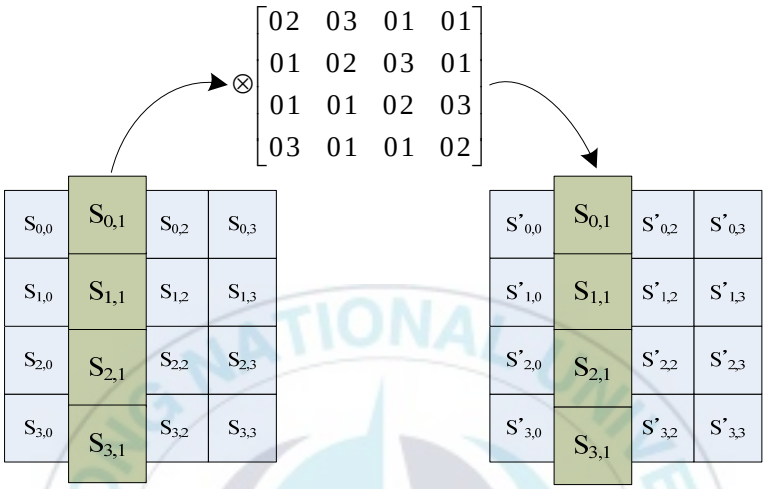


그림 10. MixColumns() 연산

AddRoundKey() 연산은 현재 상태의 각 바이트와 라운드 키를 XOR 연산을 수행한다. AddRoundKey() 연산은 그림 11과 같다.

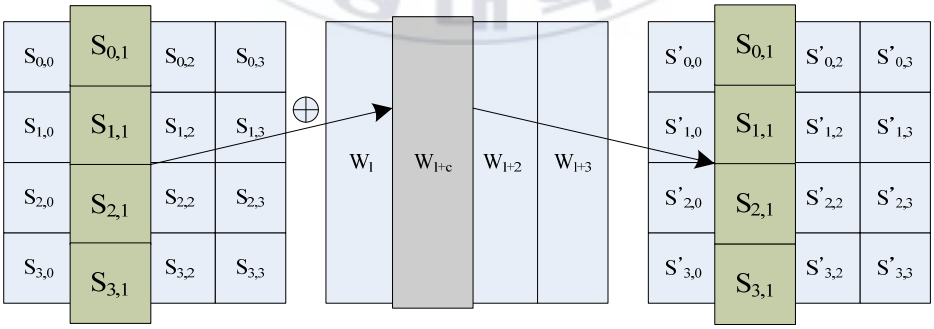


그림 11. AddRoundKey() 연산

2.4 블록 암호 알고리즘 공격 방법

블록 암호 알고리즘을 해독하기 위해 공격하는 방법 중에서 가장 강력한 것이 차분 분석법[25]과 선형 분석법[26]이다. 이 장에서는 이 두 암호 분석법에 대해서 H. M. Heys의 튜토리얼(Tutorial)을 기반으로 소개를 하도록 하겠다[35].

2.4.1 차분 분석법

차분 분석법은 FEAL-4를 공격하는 방법으로 Murphy에 의해 처음으로 소개되었다[36]. 그리고 Biham과 Shamir가 DES와 같은 블록 암호 알고리즘을 공격하기 위하여 이 방법을 완성하여, DES 공격에 성공하였다. 이 후에 여러 블록 암호들이 이 분석법에 의하여 해독되었다[37, 38].

이 암호 분석법은 입력 평문의 변화에 대응되는 출력 암호문의 변화를 높은 확률로 예측 가능 할 때, 이 특성을 이용하여 공격하는 방법이다. 이 때 입력 평문의 변화와 출력 암호문의 변화를 차분(Differential)이라고 하고, 각각의 차분을 구하는 연산은 비트간의 배타적 논리합(XOR) 연산이다. 입력 $X = [X_1 \quad X_2 \quad \cdots \quad X_n]$ 이고, 출력 $Y = [Y_1 \quad Y_2 \quad \cdots \quad Y_n]$인 암호 시스템에서 두 입력을 X_1, X_2 그에 대응되는 출력을 각각 Y_1, Y_2라 두자. 입력 비트들 간의 차분은

$$\Delta X = X_1 \oplus X_2$$

에 의해서 주어진다. 여기서 "⊕"는 비트간의 XOR 연산을 나타낸다. 따라

서 입력 차분

$$\Delta X = [\Delta X_1 \quad \Delta X_2 \quad \cdots \quad \Delta X_n]$$

이다. 여기서 두 입력 X_1 과 X_2 의 i 번째 비트를 X_{1i}, X_{2i} 로 나타내면 그 비트들 사이의 차분

$$\Delta X_i = X_{1i} \oplus X_{2i}$$

이다. 같은 방법을 사용하면 출력 차분은

$$\Delta Y = Y_1 \oplus Y_2$$

에 의해서 구할 수 있고, 출력 차분

$$\Delta Y = [\Delta Y_1 \quad \Delta Y_2 \quad \cdots \quad \Delta Y_n]$$

이다. 여기서 두 출력의 i 번째 비트의 차분

$$\Delta Y_i = Y_{1i} \oplus Y_{2i}$$

이다.

이상적으로 무작위로 구성되는 암호 알고리즘에서 특정 입력 차분 ΔX 가 주어지고 특정 출력 차분 ΔY 가 발생할 확률은 $\frac{1}{2^n}$ 이다. 여기서 n

은 입력 X의 비트 수이다. 차분 분석법은 특정 입력 차 ΔX가 주어졌을 때 특정 출력 차 ΔY의 발생이 매우 높은 확률 p_D를 가지도록 구성해서 이용하는 것이다. 여기서 확률 p_D는 $\frac{1}{2^n}$보다 훨씬 큰 값이다.

차분 분석법은 선택된 평문 공격법이다. 이것은 공격자가 키를 알아내기 위한 시도에서 입력을 선택하고 출력을 시험하는 것이 가능하다는 의미이다. 즉, 차분 분석법에서 공격자는 자신이 알고 있는 어떤 특정 ΔX를 만족하도록 입력 쌍 X_1, X_2를 선택해서 특정 출력 차 ΔY가 높은 확률을 가지고 발생되도록 해서 공격을 한다.

차분 분석법이 소개된 이후 이 공격에 대해서 암호 알고리즘을 보호하는 연구들이 Lai, Massey, Murphy 와 Nyberg, Kundsen에 의해 이루어졌다[39, 40]. 차분 분석법은 또한 해쉬 함수와 같은 다른 암호 알고리즘들에 대한 공격에도 유용하게 이용되고 있다.

2.4.2 선형 분석법

선형 분석법은 FEAL을 공격하기 위해 Matsui와 Yamagishi에 의해 처음으로 소개되었다[41]. 그 후 DES를 공격하기 위해 Matsui가 확장하여 공격에 성공하였다[26].

선형 분석법은 알려진 평문 공격법이다. 즉, 공격자가 평문과 그에 대응하는 암호문의 집합에 대한 정보를 가지고 있다는 것을 전제로 한다. 그러나 공격자는 그 정보들 중에서 이용할 수 있는 평문과 대응하는 암호문을 선택하기 위한 방법은 모르는 것이다.

이 공격법의 기본 개념은 암호 알고리즘의 비선형 부분의 동작을 선형

근사 시키는 것이다. 여기서 선형은 모듈러-2 비트간 연산을 의미한다. 즉, "⊕"로 표현되는 배타적 논리합(XOR) 연산을 뜻한다. 이것을 식으로 표현하면

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_m} \oplus Y_{i_1} \oplus Y_{i_2} \oplus \cdots \oplus Y_{i_n} = 0 \qquad (3)$$

이고, 여기서 X_i는 입력 비트 $X = [X_1,\ X_2,\ \cdots]$의 i번째 비트를 나타내고, Y_j는 출력 비트 $Y = [Y_1,\ Y_2,\ \cdots]$의 j번째 비트를 나타낸다.

식 (3)의 오른쪽 편을 하위 키(즉, 각 라운드에 사용되는 키) 비트들의 합을 가지도록 동등하게 재구성할 수 있다. 그런데 식 (3)의 오른쪽 편이 "0"을 가지면, 그 식은 하위 키 비트들을 함축적으로 포함하고 있다. 이 비트들은 위치가 고정되어 있으나 키에 의해서 결정되었기 때문에 알려져 있지 않고, 식 (3)의 오른쪽 편에 있는 "0"에 포함되어 있고 그 선형식은 확률 p_L을 가진다. 만약 하위 키 비트들의 합이 "0"이면, 식 (3)의 편향은 그 식이 포함하고 있는 하위 키 합의 편향 때문에 똑 같은 부호(+ 혹은 -)를 가질 것이고, 만약 포함된 서브 키 비트들의 합이 "1"이면 식 (3)의 편향은 반대 부호를 가질 것이다.

확률 "$p_L = 1$"은 선형식이 암호 동작과 암호가 지닌 최대의 약점을 완전히 나타낸다는 것을 의미한다. 만약 확률 "$p_L = 0$"이면 식 (3)은 암호에서 아핀 관계를 나타내고, 또한 최대 약점을 가리킨다. 모듈러-2 덧셈 연산 체계에서 아핀 함수는 선형 함수를 간단하게 보완하는 것이다.
선형 근사와 아핀 근사가 각각 $p_L > \frac{1}{2}$과 $p_L < \frac{1}{2}$에 의해 나타나면 똑같이

선형 분석법에 영향을 받기 쉽다.

높은 선형성을 가지는 식을 구성하기 위해서는 블록 암호에서 유일한 비선형 요소인 S 박스의 특성을 고려해야 한다. S 박스의 비선형 특성들이 계산되면, S 박스의 입력과 출력 비트들의 집합 사이의 선형 근사치를 전개하는 것이 가능하다. 그 결과, S 박스들의 선형 근사치들을 연결하는 것이 가능하므로 중간 비트 들은 생략되고, 단지 평문과 마지막 라운드의 입력 비트들만 포함하고 큰 편향을 가진 선형식을 구할 수 있다.

선형 분석법의 원리인 Matsui가 알고리즘 2라고 이름 붙인 **Pilling-Up 보조정의**[26]를 소개한다. 이것을 소개하기 전에 확률의 기본을 먼저 알아보자. 두 개의 임의의 이진 변수 X_1과 X_2가 있다. $X_1 \oplus X_2 = 0$은 선형식이고 $X_1 = X_2$와 같다. $X_1 \oplus X_2 = 1$은 아핀식이고 $X_1 \neq X_2$와 같다.

확률 분포가

$$\Pr(X_1 = i) = \begin{cases} p_1 & , i = 0 \\ 1 - p_1 & , i = 1 \end{cases}$$

과

$$\Pr(X_2 = i) = \begin{cases} p_2 & , i = 0 \\ 1 - p_2 & , i = 1 \end{cases}$$

에 의해 주어졌다고 가정하자.

만약 두 임의의 변수가 독립적이면

$$\Pr\left(X_1 = i, X_2 = j\right) = \begin{cases} p_1 p_2 & , i = 0, j = 0 \\ p_1\left(1 - p_2\right) & , i = 0, j = 1 \\ \left(1 - p_1\right) p_2 & , i = 1, j = 0 \\ \left(1 - p_1\right)\left(1 - p_2\right) & , i = 1, j = 1 \end{cases}$$

이고 이것은 다음과 같이 나타낼 수도 있다.

$$\begin{aligned} \Pr\left(X_1 \oplus X_2 = 0\right) &= \Pr\left(X_1 = X_2\right) \\ &= \Pr\left(X_1 = 0, X_2 = 0\right) + \Pr\left(X_1 = 1, X_2 = 1\right) \\ &= p_1 p_2 + \left(1 - p_1\right)\left(1 - p_2\right) \end{aligned}$$

$p_1 = \frac{1}{2} + \varepsilon_1$ 그리고 $p_2 = \frac{1}{2} + \varepsilon_2$ 라 두자. 여기서 ε_1 과 ε_2 는 확률 편향이고, $-\frac{1}{2} \le \varepsilon_1,\ \varepsilon_2 \le +\frac{1}{2}$ 이다. 그러면 확률 분포

$$\Pr\left(X_1 \oplus X_2 = 0\right) = \frac{1}{2} + 2\varepsilon_1\varepsilon_2$$

이고, $X_1 \oplus X_2 = 0$ 의 편향 $\varepsilon_{1,2}$ 는

$$\varepsilon_{1,2} = 2\varepsilon_1\varepsilon_2$$

이다.

이것은 확률 $p_1 = \frac{1}{2} + \varepsilon_1$ 에서 $p_n = \frac{1}{2} + \varepsilon_n$ 를 가지는 더 많은 임의의 이진 변수들인 X_1 에서 X_n 으로 확장될 수 있다. $X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0$ 이

가지는 확률은 모든 임의의 이진 변수들 n 이 독립적이라고 가정하면 **Piling-Up 보조정의**에 의해서 결정될 수 있다.

Piling-Up 보조정의[26]

서로 독립인 n 개의 임의의 이진 변수들 X_1, X_2, …, X_n 에 대해서

$$\Pr\left(X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0\right) = \frac{1}{2} + 2^{n-1}\prod_{i=1}^{n}\varepsilon_i$$

혹은, 동등하게

$$\varepsilon_{1,2,\cdots,n} = 2^{n-1}\prod_{i=1}^{n}\varepsilon_i$$

이다. 여기서, $\varepsilon_{1,2,\cdots,n}$ 은 $X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0$ 의 확률 편향을 나타낸다. □

만약 모든 i 에 대해서 $p_i = 0$ 혹은 $p_i = 1$ 이면, $\Pr\left(X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0\right) = 0$ 혹은 $\Pr\left(X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0\right) = 1$ 임을 주목해야 한다. 만약 단지 하나의 $p_i = \frac{1}{2}$ 이면, $\Pr\left(X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0\right) = \frac{1}{2}$ 이다.

암호 알고리즘의 선형 근사치를 전개하는데 있어 X_i 의 값들은 실제 S 박스들의 선형 근사치를 나타낸다.

선형 분석법은 평문과 그에 대응하는 암호문의 쌍이 충분히 주어졌을 때 키에 대한 정보를 가진 비트들을 얻을 수 있으며, 증가된 데이터의 양은 성공 확률을 더 높여준다.

선형 분석법은 기본적인 공격 방법을 바탕으로 다양하게 향상되어 왔다. Langford와 Hellman은 차분 선형 암호 공격[42]이라고 불리는 공격을 연구 하였는데, 이것은 차분 공격의 원소들과 선형 공격의 원소들을 결합한 것이다. 또한 Kaliski와 Robshow는 다중 근사화를 이용한 선형 공격은 요구되는 데이터의 양을 감소 시킨다고 소개하였다[43].

이 두 가지 공격 방법 이외에 최근에 나온 공격법으로 스퀘어(Square) 공격법[44]이 있다. 스퀘어 공격법은 SQUARE 알고리즘을 공격하기 위해 처음으로 적용된 방법이다. 스퀘어 공격은 바이트 단위 혹은 워드 단위의 선택 평문 패턴이 전파되는 특성을 이용한 공격이다. 최근에는 AES 알고리즘을 이 공격법을 사용하여 공격하는 연구가 진행되고 있다[45]. 그리고 전 동현 등[46]이 제안한 Difference Distribution 공격이 있다. Difference Distribution 공격은 선택된 평문 공격법으로 입력 차이가 고정되었을 때 출력 차이의 분포를 사용해서 공격한다. 2 라운드로 구성되었다면 공격은 다음과 같이 시작한다. 먼저 고정된 입력 차이 Δ를 선택하고, 그 입력 차이 Δ를 가지는 2^{64} 개의 평문 쌍을 선택한다. 그리고 대응되는 암호문 쌍을 얻고, 입력 차이와 같은 출력 차이가 존재하도록 결정한다. 이것이 발생할 확률은 2^{-16} 이다. 그래서 만약 2^{16} 의 입력 차이를 고려하면 입력 차이 Δ와 같은 값을 가지는 출력 차이 Δ'을 얻을 수 있다. 이런 상태가 발생되면 마지막 라운드 함수를 통과하고 나온 출력 차이가 $(\alpha, 0)$ 형이 된다. 이것은 마지막 라운드 함수가 입력 차이에 의한 영향을 받지 않아 안전성에 영향을 미치지 않는다는 것을 의미한다.

2.5 MDS 코드

블록 암호 알고리즘은 암호문을 충분히 섞기 위해 치환을 수행한다. 이때 사용하는 것이 선형변환이다. 그리고 선형변환은 선형변환행렬을 이용해서 수행한다. 선형변환이 차분 분석법과 선형 분석법의 특성에 강하기 위해서는 선형변환행렬이 MDS(Maximum Distance Separable) 코드를 생성해야 한다. 이 절에서는 선형변환행렬이 MDS코드를 생성하기 위한 방법에 대해서 살펴보도록 하겠다.

2.5.1 치환 계층과 교환 계층의 특성

치환(Substitution)과 교환(Permutation)을 반복하는 구조를 가지고 있는 블록 암호 알고리즘은 치환을 수행하는 치환 계층과 교환을 수행하는 교환 계층의 특성이 안전성에 영향을 준다. 이 구조를 SPN 구조라고 한다. SPN 구조의 특성에 대해서 강 주성 등[47]과 Tavares 등[48] 그리고 Heys[49]의 연구결과가 있다.

치환 계층은 비선형 특성을 가지는 S 박스들로 구성된다. 이 S 박스들은 교환 계층의 특성에 따라 차분 및 선형 활동성을 가지게 된다. 차분 활동성을 가지는 S 박스는 입력 값이 '0'이 아닌 S 박스이며, 선형 활동성을 가지는 S 박스는 입력과 출력의 마스크 값이 '0'이 아닌 S 박스이다.

치환 계층과 교환 계층을 가지는 구조에서 입력과 출력을 각각 x, y 라 두면 차분 및 선형 활동성을 가지는 S 박스의 최소수는 아래 식과 같다.

$$\beta_d(P) = \min_{\Delta x \neq 0} \{W_c(\Delta x) + W_c(\Delta y)\}$$
$$\beta_l(P) = \min_{b \neq 0} \{W_c(a) + W_c(b)\}$$
$$while, \quad \Delta x, \Delta y, a, b \in Z_2^n$$

여기서, W는 해밍 가중치이다.

차분 및 선형 활동성을 가지는 S 박스의 최소수를 Branch number라고 하며, 교환 계층의 선형변환행렬의 특성에 의하여 결정된다. 교환 계층의 선형변환행렬을 M이라 하면 행렬 M의 출력 값과 입력 마스크 값 사이의 관계는 전치된 행렬인 M^t로 나타낼 수 있다. 따라서

$$\Delta y = M\Delta x, \quad a = M^t b$$

이다[50]. 여기서 전치행렬은 원래 행렬의 열은 행으로 행은 열로 바꾼 행렬이다. 이 식을 차분 및 선형 활동성을 가지는 S 박스의 최소수를 구하는 식에 적용하면 차분 및 선형 활동성을 가지는 S 박스의 최소수인 Branch number는 다음 식과 같이 된다.

$$\beta_d(P) = \min_{\Delta x \neq 0} \{W_c(\Delta x) + W_c(M\Delta x)\}$$
$$\beta_l(P) = \min_{b \neq 0} \{W_c(M^t b) + W_c(b)\}$$

이 식으로부터 행렬 M이 대칭행렬 또는 직교행렬이면 차분 Branch number및 선형 Branch number가 동일하게 됨을 알 수 있다. 여기서 대칭행렬은 주 대각선을 기준으로 대칭인 행렬이고, 직교행렬은 어떤 행렬에 그 행렬의 전치행렬을 곱했을 때 단위행렬이 되는 행렬이다.

치환 계층의 S 박스에 대한 차분 및 선형 공격 최고 확률을 각각 D 및 L 이라고 정의하면 SPN 구조의 차분 및 선형 공격 최고 확률 D_f 와 L_f 는

$$D_f \le D^\beta, \quad L_f \le L^\beta$$

이 된다.

차분 분석법 및 선형 분석법에 강하기 위해서는 D_f 및 L_f 가 작아야 한다. 그런데 D 와 L 은 '1'보다 항상 작으므로, Branch number β 가 최대로 되어야 차분 분석법과 선형 분석법에 강한 특성을 지니게 된다.

임의의 필드상의 선형 $(n,\ k,\ \beta)$ 코드에서 $\beta \le n-k+1$ 이다. 여기서 $\beta = n-k+1$ 이 최대값이 되는 코드를 Maximum Distance Separable 코드 또는 줄여서 MDS 코드라고 한다[51]. SPN 구조를 가지는 블록 암호 알고리즘에서 교환 계층의 선형변환행렬 M 은 $GF(2^n)$ 상의 $(2m,\ m,\ \beta)$ 코드가 된다. 이때 생성행렬 $G = [I \mid M]$ 이라 하면, M 은 $m \times m$ 정칙행렬이고, I 는 $m \times m$ 항등행렬이다. 행렬 M 이 MDS코드를 생성하면, Branch number 가 $\beta = m+1$ 로 최대값을 가진다[48].

따라서 SPN 구조에서 차분 분석법과 선형 분석법에 강한 특성을 지니기 위해서는 교환 계층의 선형변환행렬이 MDS 코드를 생성해야 한다. 선형변환행렬이 MDS 코드를 생성하기 위해서는 직교행렬 또는 대칭행렬로 구성되어야 한다.

2.5.2 선형 변환 행렬 구성 방법

교환 계층의 선형변환행렬이 차분 분석법과 선형 분석법에 강하기 위해서는 MDS 코드를 생성하는 대칭행렬 또는 직교행렬로 구성되어야 한다. 여기서 Tavares 등[48]이 소개한 선형변환행렬의 구성 방법을 알아보겠다.

보조정의 2.1[51]

생성 행렬이 $G=[I \mid A]$ 인 어떤 $(n,\ k,\ d)$ 코드에서 A의 모든 정방부분행렬(임의의 i행, i열로 구성되고, $i=1,2,3,\cdots,\min\{k,n-k\}$)이 정칙이면, 이 코드는 MDS이다. 여기서 A는 $k\times(n-k)$ 행렬이다.

MDS 코드를 생성하는 선형변환행렬을 구성하는 방법은 행렬의 원소를 무작위로 선정하는 방식과 Cauchy 행렬을 사용하는 방식이 있다.

먼저 무작위로 선정하는 방식은 $M\times M$ 선형변환행렬 A를 $GF(2^n)$상의 임의의 원소로 구성한 다음 **보조정의 2.1**의 조건을 만족하는지 검사하는 방법이다. 무작위로 선정하는 방식으로 선형변환행렬 A를 구성할 때는 $n=8,\ M\le 6$인 짝수 값들인 M에 대해서는 쉽게 구성할 수 있다. 그러나 M이 커지면 행렬의 구성이 어려워지고, MDS 코드를 생성할 확률도 낮아지게 된다.

다음으로 Cauchy 행렬을 이용해서 구성하는 방식이다.

보조정의 2.2[51]

$x_0,\ x_1,\ x_2,\ \cdots,\ x_{n-1}$ 과 $y_0,\ y_1,\ y_2,\cdots,\ y_{n-1}$ 이 주어진 행렬 $A=[a_{ij}](0\le i,j\le n-1)$ 에서, $a_{ij}=\dfrac{1}{x_i+y_j}$ 이고, 모든 $i,\ j$ 에 대해서 $x_i+y_j\neq 0$ 인 행렬을 Cauchy 행렬이라고 부른다. Cauchy 행렬은 다음 식을 만족한다.

$$\det(A)=\frac{\prod\limits_{0\le i<j\le n-1}(x_j-x_i)(y_j-y_i)}{\prod\limits_{0\le i,j\le n-1}(x_i+y_j)}$$

따라서 Cauchy 행렬의 모든 정방부분행렬이 정칙이므로 Cauchy 행렬은 MDS 코드를 생성한다. 그런데 Cauchy 행렬은 항상 특정한 원소들로 행렬을 구성하기 때문에 차분 분석법과 선형 분석법에 강한 특성을 유지하기 위한 행렬의 각 원소들 간의 곱셈 특성이 [48]의 연구결과를 만족시키지 못한다. 따라서 선형변환행렬은 행렬의 원소를 $GF(2^n)$ 상의 원소들 가운데 무작위로 선정하는 방식을 사용해서 구성해야 한다. 그런데 무작위로 선정하는 방식을 사용하게 되면 행렬의 차수가 높아지면 선형변환행렬이 MDS 코드를 생성할 확률이 낮아지게 된다. 그래서 구성한 선형변환행렬이 MDS 코드를 생성하는지 확인할 필요가 있다. MDS 코드를 생성하는지 확인하는 알고리즘으로는 선형변환행렬의 모든 정방부분행렬이 정칙이면 MDS 코드를 생성한다는 것이 연구되어 있다[48].

제 3 장 MDS 코드 생성 확인 알고리즘

블록 암호 알고리즘의 교환 계층에서 사용되는 선형변환행렬을 구성하는 방법은 2장에서 소개한대로 무작위로 구성하는 법과 Cauchy 행렬을 이용하는 방법이 있다.

무작위로 구성하는 법은 행렬의 차수가 커질수록 행렬의 구성이 어려워지고, MDS 코드를 생성할 확률이 낮아진다. 또한, Cauchy 행렬을 이용한 방법은 항상 특정한 원소로만 행렬이 구성되므로 [47]과 [52]의 연구 결과를 만족하도록 선형변환행렬을 구성하기가 어렵다. 따라서 선형변환행렬은 먼저 무작위 선정 방식을 사용해서 구성한 후, 그 결과 선형변환행렬이 MDS 코드를 생성하는지 확인해야 한다.

이 장에서는 블록 암호 알고리즘에서 치환을 할 때 사용하는 선형변환행렬이 MDS 코드를 생성하는지 판단하는 알고리즘을 제안한다.

3.1 MDS 코드 생성 확인 알고리즘

블록 암호 알고리즘에서 선형변환행렬 M 은 $GF(2^n)$ 상의 m 개의 n 비트 코드 X 를 m 개의 n 비트 코드 Y 로 선형변환 한다. 이 때 선형변환행렬을 M 이라 하면 선형변환식은 식 (4)와 같다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ \cdots \\ Y_{m-1} \end{bmatrix} = \begin{bmatrix} M_{0,0} & M_{0,1} & M_{0,2} & \cdots & M_{0,m-1} \\ M_{1,0} & M_{1,1} & M_{1,2} & \cdots & M_{1,m-1} \\ M_{2,0} & M_{2,1} & M_{2,2} & \cdots & M_{2,m-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ M_{m-1,0} & M_{m-1,1} & M_{m-1,2} & \cdots & M_{m-1,m-1} \end{bmatrix} \times \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \cdots \\ X_{m-1} \end{bmatrix} \quad (4)$$

식 (4)에서 선형변환의 입력코드 X 의 해밍 가중치와 선형변환 후의 출력코드 Y 의 해밍 가중치를 각각 $W_h(X)$, $W_h(Y)$ 라 두면, 선형변환행렬 M 의 Branch number는 식 (5)와 같이 주어진다.

$$\beta = \min\left(W_h(X) + W_h(Y)\right) \quad \text{while} \quad X \neq 0 \quad (5)$$

Branch number $\beta = m+1$ 이 되면 선형변환행렬 M 은 MDS 코드를 생성하게 된다. 따라서 선형변환행렬 M 이 MDS 코드를 생성하는지 판단하기 위해서는 입력코드 X 의 모든 원소에 대해 Branch number $\beta = m+1$을 만족하는지 확인하면 된다.

입력코드 X 의 원소 $x_i\,(0 \le i \le m-1)$을 변수로 취급하여 이들을 소거하면서 연산하여 MDS 코드를 생성하는지 판단한다.

보조정의 3.1

m 개의 원소를 가진 입력코드의 해밍 가중치 '$W_h(X)=1$'이면, 선형변환행렬 M 의 모든 원소가 '0'이 아닌 경우 MDS 코드를 생성한다.

증명

입력코드 X 의 해밍 가중치 $W_h(X)=1$ 이므로 X 의 원소 중 하나만이 '0'이 아니다. 그 원소를 $x_i\,(0 \le i \le m-1)$ 라 하면 식 (4)는 식 (6)과 같이 된다.

$$\begin{aligned} Y_0 &= M_{0,i} \times X_i \\ Y_1 &= M_{1,i} \times X_i \\ Y_2 &= M_{2,i} \times X_i \\ &\vdots \\ Y_{m-1} &= M_{m-1,i} \times X_i \end{aligned} \tag{6}$$

식 (6)에서 $X_i \neq 0$ 이므로 선형변환행렬 M 의 모든 원소가 '0'이 아니며 출력코드 Y 의 모든 원소는 '0'이 아니고, 해밍 가중치 '$W_h(Y)=m$'이 된다. 따라서 선형변환행렬 M 의 Branch number $\beta = m+1$ 이 되어 MDS 코드를 생성한다. □

보조정의 3.2

입력코드 X의 해밍 가중치 '$W_h(X)=2$'이면 출력코드 Y의 해밍 가중치 최소값이 '$W_h(Y)=m-1$'인 경우에 MDS 코드를 생성한다. 입력코드의 원소 가운데 두 개의 원소만이 '0'이 아니고, 그 원소를 X_i, X_j라 하면 출력코드 원소 가운데 $Y_p(0 \le p \le m-2)$ 원소를 '0'으로 만드는 X_j가 반드시 존재한다. X_j를 X_i의 함수로 표현하고, 이것을 $Y_q(0 \le q \le m-1)$에 대입하여 '$Y_q \neq 0$'가 되면 선형변환행렬 M은 MDS 코드를 생성한다.

증명

입력코드 X의 해밍 가중치 '$W_h(X)=2$'이므로 X의 원소 가운데 두 개만이 '0'이 아니다. 그 원소를 X_i, X_j라 두면 식 (4)는 식 (7)과 같이 표현된다.

$$\begin{aligned} Y_0 &= \left(M_{0,i} \times X_i\right) + \left(M_{0,j} \times X_j\right) \\ Y_1 &= \left(M_{1,i} \times X_i\right) + \left(M_{1,j} \times X_j\right) \\ Y_2 &= \left(M_{2,i} \times X_i\right) + \left(M_{2,j} \times X_j\right) \\ &\vdots \\ Y_{m-1} &= \left(M_{m-1,i} \times X_i\right) + \left(M_{m-1,j} \times X_j\right) \end{aligned} \tag{7}$$

식 (7)에서 X_j를 소거시키기 위하여 '$Y_0=0$'으로 가정하고, X_j에 대해서 정리하면 식 (8)이 된다.

$$X_j = -\frac{M_{0,i} \times X_i}{M_{0,j}} \tag{8}$$

식 (8)을 식 (7)에 대입하여 정리하면 식 (9)와 같이 계산된다.

$$\begin{aligned}
Y_1 &= \left(M_{1,i} - \frac{M_{1,j} \times M_{0,i}}{M_{0,j}} \right) \times X_i \\
Y_2 &= \left(M_{2,i} - \frac{M_{2,j} \times M_{0,i}}{M_{0,j}} \right) \times X_i \\
Y_3 &= \left(M_{3,i} - \frac{M_{3,j} \times M_{0,i}}{M_{0,j}} \right) \times X_i \\
&\vdots \\
Y_{m-1} &= \left(M_{m-1,i} - \frac{M_{m-1,j} \times M_{0,i}}{M_{0,j}} \right) \times X_i
\end{aligned} \tag{9}$$

식 (9)의 출력코드 Y_1에서 Y_{m-1}의 식에서 X_i의 모든 계수 값이 '0'이 아니면 $X_i \neq 0$이므로 Y_1에서 Y_{m-1}는 모두 '0'이 아니다. 따라서 출력 Y의 해밍 가중치 최소값은 '$W_h(Y) = m-1$'이다.

이어서 식 (7)에서 $Y_1 = 0$이라고 가정하면 앞에서 $Y_0 = 0$이면서 $Y_1 = 0$인 경우가 없음을 확인하였으므로 $Y_1 \neq 0$이다. 이 경우 X_j는 식 (10)이 된다.

$$X_j = -\frac{M_{1,i} \times X_i}{M_{1,j}} \tag{10}$$

X_j를 소거하기 위하여 식 (10)을 식 (7)에 대입하여 정리하면 식 (11)과 같이 계산된다.

$$\begin{aligned}
Y_2 &= \left(M_{2,i} - \frac{M_{2,j} \times M_{1,i}}{M_{1,j}} \right) \times X_i \\
Y_3 &= \left(M_{3,i} - \frac{M_{3,j} \times M_{1,i}}{M_{1,j}} \right) \times X_i \\
Y_4 &= \left(M_{4,i} - \frac{M_{4,j} \times M_{1,i}}{M_{1,j}} \right) \times X_i \\
&\vdots \\
Y_{m-1} &= \left(M_{m-1,i} - \frac{M_{m-1,j} \times M_{1,i}}{M_{1,j}} \right) \times X_i
\end{aligned} \tag{11}$$

식 (11)의 출력코드 Y_2에서 Y_{m-1}의 식에서 X_i의 모든 계수 값이 '0'이 아니면, $Y \neq 0$이므로 Y의 해밍 가중치 최소값은 '$W_h(Y) = m-1$'이 된다.

이와 같은 과정을 $Y_i (2 \leq i \leq m-2) = 0$에 대하여 반복 적용하여 모든 출력코드 Y_{i+1}에서 Y_{m-1}의 식에서 X_i의 계수 값이 '0'이 아니면, Y의 해밍 가중치 최소값은 '$W_h(Y) = m-1$'이 된다. □

보조정의 3.3

입력코드 X의 해밍 가중치 '$W_h(X)=3$'이면 출력코드 Y의 해밍 가중치 '$W_h(Y)=m-2$'인 경우에 MDS 코드를 생성한다. 입력코드 X의 원소 가운데 세 개만이 '0'이 아니고, 그 원소를 X_i, X_j 그리고 X_k라 하면 출력코드의 원소 $Y_p(0\le p\le m-3)$를 '0'으로 가정하여 X_k를 X_i와 X_j의 함수로 표현하고, 이것을 $Y_q(0\le q\le m-2)$에 대입하여 X_k를 소거한다. Y_q는 '$W_h(X)=2$'인 $m-q$차원의 행렬이 되며, $Y_q(0\le q\le m-2)$가 **보조정의 3.2**를 만족하면 선형변환행렬 M은 MDS 코드를 생성한다.

증명

입력코드 X의 해밍 가중치 '$W_h(X)=3$'이므로 X의 원소 가운데 세 개만이 '0'이 아니다. 그 원소를 X_i, X_j 그리고 X_k라 두면 식 (4)는 식 (12)와 같이 계산된다.

$$\begin{aligned}
Y_2 &= \left(M_{2,i}-\frac{M_{2,j}\times M_{1,i}}{M_{1,j}}\right)\times X_i \\
Y_3 &= \left(M_{3,i}-\frac{M_{3,j}\times M_{1,i}}{M_{1,j}}\right)\times X_i \\
Y_4 &= \left(M_{4,i}-\frac{M_{4,j}\times M_{1,i}}{M_{1,j}}\right)\times X_i \\
&\vdots \\
Y_{m-1} &= \left(M_{m-1,i}-\frac{M_{m-1,j}\times M_{1,i}}{M_{1,j}}\right)\times X_i
\end{aligned} \tag{12}$$

식 (12)에서 X_k를 소거시키기 위하여 $Y_0 = 0$이라 가정하고 X_k에 대하여 정리하면 식 (13)이 된다.

$$
\begin{aligned}
Y_1 &= \left\{\left(M_{1,i} - \frac{M_{0,i} \times M_{1,k}}{M_{0,k}}\right) \times X_i\right\} + \left\{\left(M_{1,j} - \frac{M_{0,j} \times M_{1,k}}{M_{0,k}}\right) \times X_j\right\} \\
Y_2 &= \left\{\left(M_{2,i} - \frac{M_{0,i} \times M_{2,k}}{M_{0,k}}\right) \times X_i\right\} + \left\{\left(M_{2,j} - \frac{M_{0,j} \times M_{2,k}}{M_{0,k}}\right) \times X_j\right\} \\
Y_3 &= \left\{\left(M_{3,i} - \frac{M_{0,i} \times M_{3,k}}{M_{0,k}}\right) \times X_i\right\} + \left\{\left(M_{3,j} - \frac{M_{0,j} \times M_{3,k}}{M_{0,k}}\right) \times X_j\right\} \\
&\vdots \\
Y_{m-1} &= \left\{\left(M_{m-1,i} - \frac{M_{0,i} \times M_{m-1,k}}{M_{0,k}}\right) \times X_i\right\} + \left\{\left(M_{m-,j} - \frac{M_{0,j} \times M_{m-1,k}}{M_{0,k}}\right) \times X_j\right\}
\end{aligned}
\tag{13}
$$

여기서 식을 간소화하기 위하여

$$
\begin{aligned}
&\left(M_{1,i} - \frac{M_{0,i} \times M_{1,k}}{M_{0,k}}\right) = M'_{1,i}, \quad \left(M_{1,j} - \frac{M_{0,j} \times M_{1,k}}{M_{0,k}}\right) = M'_{1,j} \\
&\left(M_{2,i} - \frac{M_{0,i} \times M_{2,k}}{M_{0,k}}\right) = M'_{2,i}, \quad \left(M_{2,j} - \frac{M_{0,j} \times M_{2,k}}{M_{0,k}}\right) = M'_{2,j} \\
&\cdots \\
&\left(M_{m-1,i} - \frac{M_{0,i} \times M_{m-1,k}}{M_{0,k}}\right) = M'_{m-1,i}, \quad \left(M_{m-1,j} - \frac{M_{0,j} \times M_{m-1,k}}{M_{0,k}}\right) = M'_{m-1,j}
\end{aligned}
$$

로 치환하면 식 (13)은 식 (14)로 나타낼 수 있다.

$$
\begin{aligned}
Y_1 &= \left(M'_{1,i} \times X_i\right) + \left(M'_{1,j} \times X_j\right) \\
Y_2 &= \left(M'_{2,i} \times X_i\right) + \left(M'_{2,j} \times X_j\right) \\
Y_3 &= \left(M'_{3,i} \times X_i\right) + \left(M'_{3,j} \times X_j\right) \\
&\vdots \\
Y_{m-1} &= \left(M'_{m-1,i} \times X_i\right) + \left(M'_{m-1,j} \times X_j\right)
\end{aligned} \tag{14}
$$

식 (14)는 식 (7)과 같은 형태이다. 따라서 식 (14)가 **보조정의 3.2**를 만족하면 $Y_1 - Y_{m-1}$의 해밍 가중치의 최소값은 '$m-2$'이다. 한편 $Y_0 = 0$이므로 Y의 해밍 가중치의 최소값은 '$W_h(Y) = m-2$'이다.

이어서 식 (12)에서 $Y_1 = 0$이라고 가정하고 식 (12)가 **보조정의 3.2**를 만족하면 $Y_2 - Y_{m-1}$의 해밍 가중치의 최소값은 '$m-3$'이 된다. 앞에서 $Y_0 = Y_1 = 0$ 이면서 또 다른 $Y_i = 0$ 인 경우가 없음을 확인 하였으므로, $Y_0 \neq 0$이다. 따라서 Y의 해밍 가중치의 최소값은 '$W_h(Y) = m-2$'이다.

이와 같은 과정을 $Y_i (3 \leq i \leq m-3)$에 대하여 반복 수행하여 모두 **보조정의 3.2**를 만족하면 $Y_i - Y_{m-1}$의 해밍 가중치의 최소값은 '$m-i-1$'이 된다. 그리고 동시에 세 개의 $Y_i = 0$이 되는 경우가 없음을 확인하였으므로, Y의 해밍 가중치 최소값은 '$W_h(Y) = m-2$'이 된다. □

보조정의 3.4

입력코드 X의 해밍 가중치 '$W_h(X)=k(4\le k\le m)$'이면, 출력코드 Y의 해밍 가중치 최소값이 '$W_h(Y)=m-k+1$'이면 MDS 코드를 생성한다. 입력코드 X의 원소 가운데 k개의 원소가 '0'이 아니라면, 출력코드 Y의 원소 $Y_p(0\le p\le m-k)$를 '0'으로 가정하여 X의 원소 가운데 하나를 다른 X의 원소로 표현하고 이것을 $Y_q(p<q\le m-k+1)$에 대입하여 X의 원소 하나를 소거하면 Y_q는 '$W_h(X)=k-1$'인 $m-1$차원의 행렬이 된다. 이러한 과정을 재귀적으로 반복하여 $W_h(X)=2$인 $m-k+2$차원의 행렬이 되고, 이 행렬이 **보조정의 3.2**를 만족하면 선형변환행렬 M은 MDS 코드를 생성한다.

증명

보조정의 3.3의 증명을 재귀적으로 적용하여 증명할 수 있다. □

3.2 구현 및 비교

본 논문에서 제안한 알고리즘은 m개의 원소를 가진 입력코드 X의 해밍 가중치가 '$W_h(X)=1$'이면 **보조정의 3.1**에 의하여 곱셈 연산을 수행하지 않는다. 그리고 '$W_h(X)=k(2\le k\le m)$'이면 m개의 원소 중에서 '0'이 아닌 k개의 원소를 선정해서 연산을 수행한다. 따라서 원소를 선정하는

경우의 수는 $\binom{m}{k}$가 되고, 각 경우에 있어서 k의 값에 따라서 **보조정의 3.2**부터 **보조정의 3.4**를 적용한다. 이를 수식으로 나타내면 식 (15)가 된다.

$$F(m)=\sum_{k=2}^{m}\left\{\binom{m}{k}Y_P(m,k)\right\} \tag{15}$$

식 (15)의 Y_p에서 $k=2$이면 **보조정의 3.2**에 의하여 곱셈 연산의 횟수가 결정된다. $k>2$이면 **보조정의 3.3** 및 **보조정의 3.4**에서 k값을 '1'씩 감소시켜가면서 재귀적으로 연산해서 $k=2$가 되면 **보조정의 3.2**를 적용한다. 이를 수식화하면 식 (15)의 Y_p는 식 (16)과 같이 된다.

$$Y_P(m,k)=\begin{cases}\sum_{i=2}^{m} i & \text{if } k=2\\ \sum_{i=0}^{m-k}\left[\{(m-i)\times(k-1)\}+Y_P(m-i-1,k-1)\right] & \text{if } k>2\end{cases} \tag{16}$$

동일한 방법으로 역수 연산 횟수를 수식화하면 식 (17)이 된다.

$$G(m)=\sum_{k=2}^{m}\left\{\binom{m}{k}Y_R(m,k)\right\}$$
$$\text{where } Y_R(m,k)=\begin{cases}m-1 & \text{if } k=2\\ \sum_{i=0}^{m-k}\{1+Y_R(m-i-1,k-1)\} & \text{if } k>2\end{cases} \tag{17}$$

선형변환행렬이 MDS 코드를 생성 하는지 판단하는 기존 알고리즘은

행렬의 모든 정방부분행렬이 정칙인지를 확인하는 것이다. 행렬이 정칙이기 위한 필요조건은 행렬의 determinant(det)가 '0'이 아니어야 한다. 본 논문에서는 행렬을 상삼각행렬로 변환하고, 그 대각선 원소들이 모두 '0'이 아니면 det 또한 '0'이 아니라는 점을 이용하여 정칙행렬을 판단하였다. 기존 알고리즘에서 곱셈 연산(F_S)과 역수 연산(G_S)의 횟수를 행렬의 차수(m)에 대한 함수로 나타내면 다음과 같다.

$$F_S(m) = \sum_{k=2}^{m} \left\{ \binom{m}{k} \binom{m}{k} Y_S(k) \right\} \quad \text{where} \quad Y_S(k) = \sum_{i=0}^{k-2} \sum_{j=i+1}^{k-1} (k-1)$$

$$G_S(m) = (m-1) + \sum_{k=1}^{m-2} \left\{ k \binom{m}{k+1} \binom{m}{k+1} \right\}$$

제안한 알고리즘과 기존의 알고리즘을 C 언어를 사용하여 프로그램을 작성해서 동작을 검증하였다. $GF(2^8)$ 상에서 선형변환행렬 M 의 차수가 $m = 8,12,16$ 인 경우에 곱셈 연산과 역수 연산의 횟수를 측정하여 수식과 일치함을 확인하였다. 행렬의 차수가 24와 32인 경우는 연산 시간이 많이 걸려 동작을 프로그램으로는 확인하지 못하였으나 차수가 8, 12, 16인 경우를 봤을 때 수식과 일치할 것으로 판단된다.

표 3에 $GF(2^8)$ 상에서 선형변환행렬 M 의 차수 m 에 따른 곱셈 및 역수 연산 횟수를 비교하여 보였다. 이를 그림 12에 그래프로 나타내었다. 두 알고리즘의 연산 수를 비교하여 보면 행렬의 차수가 커질수록 본 논문에서 제안한 알고리즘의 연산 수가 크게 감소하는 것을 알 수 있다. 연산 수가 감소함으로써 연산 수행 시간을 많이 단축할 수 있다.

표 3. 행렬의 차수에 따른 연산 횟수

연산 \ m		8	12	16	24	32
기존 알고리즘	곱셈	3.12×10^{5}	2.15×10^{8}	1.11×10^{11}	1.96×10^{16}	2.61×10^{21}
	역수	3.86×10^{5}	1.35×10^{7}	4.21×10^{9}	3.55×10^{14}	2.75×10^{19}
제안 알고리즘	곱셈	7.75×10^{4}	2.03×10^{7}	4.87×10^{9}	2.79×10^{14}	1.64×10^{19}
	역수	1.12×10^{4}	2.49×10^{6}	5.66×10^{8}	3.10×10^{13}	1.78×10^{18}

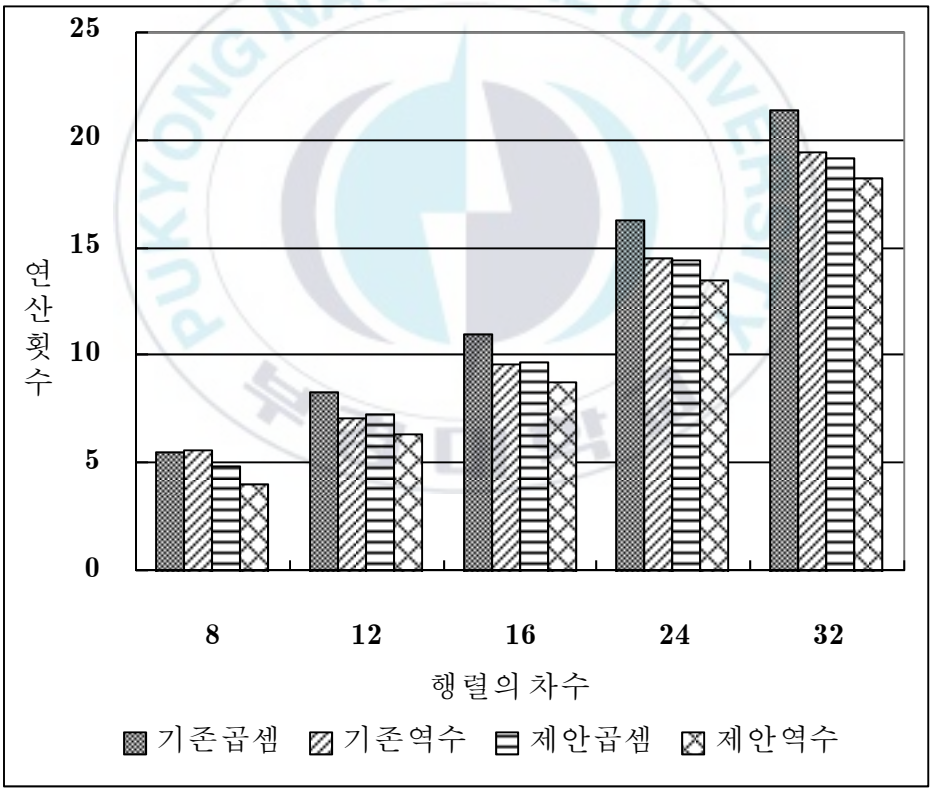


그림 12. 연산 횟수 그래프

제 4 장 산술 시프트 레지스터

이 장에서는 의사 난수 발생기로 사용할 수 있는 새로운 구조의 산술 시프트 레지스터(ASR, Arithmetic Shift Register)를 소개한다. 산술 시프트 레지스터는 갈로이 선형 궤환 시프트 레지스터를 일반화한 것으로 $GF(2^n)$에서 일정한 상수 D를 연속적으로 곱하는 구조이며, 이를 본 논문에서는 ASR-D로 기술한다. 이 산술 시프트 레지스터는 소프트웨어 및 하드웨어 구현이 용이하며, 상수 D를 변경시켜서 갈로이 선형 궤환 시프트 레지스터와 동등 이상의 선형복잡도를 가지는 다양한 종류의 난수를 얻을 수 있는 장점을 가진다.

4.1 산술 시프트 레지스터

정의 4.1

$GF(2^n)$상에서 0이 아닌 초기 값 A_0에 0 또는 1이 아닌 임의의 수 D를 곱하는 수열을 산술 시프트 레지스터(ASR-D: Arithmetic Shift Register-D)로 정의한다. ASR-D의 i번째 값(상태) A_i는 A_0D^i가 된다.

보조정의 4.1

갈로이 선형 궤환 시프트 레지스터는 '$D = 2^{-1}$'인 산술 시프트 레지스터이다. 즉, 산술 시프트 레지스터는 갈로이 선형 궤환 시프트 레지스터의 일반형이다.

증명

갈로이 선형 궤한 시프트 레지스터의 순환 방정식

$$a_i^{'} = a_{i+1} \oplus p_{i+1} a_0 \quad \text{for} \quad 0 \le i \le n-2$$
$$a_{n-1}^{'} = p_n a_0$$

은 $GF(2^n)$ 상에서 비복원다항식(Irreducible polynomial)이 $P(x)$ 인 경우에 '$A_{i+1} = \frac{A_i}{2}$'를 나타낸다. 따라서 정의 4.1에 의하여 갈로이 선형 궤환 시프트 레지스터는 ARS-2^{-1}이다. □

보조정의 4.2

$GF(2^n)$상에서 0 또는 1이 아닌 임의 수 D에 대하여 '$D^k = 1$'이 되는 t가 '$t = 2^n - 1$'로 유일하면 0을 제외한 모든 수 $R \in \{1,\ 2,\ \cdots,\ 2^n - 1\}$는 $D^u \in \{1,\ 2,\ \cdots,\ 2^n - 1\}$로 표현할 수 있다.

증명

$D^u \in \{1,2,\cdots,2^n - 1\}$ 로 표현할 수 없는 수 R_{NO} 가 존재한다면 '$\#(R - R_{NO}) \le 2^n - 1$'이 된다. 이를 만족하기 위해서는 다음 식이 성립되어

야 한다.

$$(\exists p),(\exists q), \quad D^p = D^q$$

위 식의 양변을 D^q 로 나누면 ' $D^{p-q} = 1$ '이 된다. 정의에 의하면 ' $p - q = 2^n - 1$ '이 되며, ' $p = 2^n - 1 + q$ '가 되어야 한다. 그러나 $p, q \in \{1,\ 2,\ \cdots,\ 2^n - 1\}$ 로 정의하였으므로 이 식을 만족하는 p, q 는 존재하지 않는다. □

보조정의 4.3

$GF(2^n)$ 상에서 0 또는 1이 아닌 임의 수 D 가 있고, D 에 대하여 ' $D^k = 1$ '이 되는 t 가 ' $t = 2^n - 1$ '로 유일하면 0을 제외한 모든 수 A_i 는 $A = A_0 D^u \in \{1,\ 2,\ \cdots,\ 2^n - 1\}$ 로 표현할 수 있다. A_0 는 0이 아닌 임의 수이다.

증명

보조정의 4.1로부터 D^u 는 0을 제외한 모든 수 $R \in \{1,\ 2,\ \cdots,\ 2^n - 1\}$ 을 생성할 수 있다. 따라서 $RA_0 \in \{1,\ 2,\ \cdots,\ 2^n - 1\}$ 이다. □

정의 4.2

비복원 다항식 $P(x)$ 로 표현되는 $GF(2^n)$ 상에서 0 또는 1이 아닌 임의 수 D 에 대하여 ' $D^k = 1$ '이 되는 t 가 ' $t = 2^n - 1$ '로 유일하면 $P(x)$ 를 ASR-D 의 특성다항식(Characteristic polynomial)이라 한다.

정리

$GF(2^n)$ 상에서 특성다항식(Characteristic polynomial)으로 표현되는 ASR-D의 주기는 '2^n-1'이다

증명

보조정의 4.1과 **보조정의 4.2**로부터 증명할 수 있다. □

보조정의 4.4

$GF(2^n)$ 상에서 '2^n-1'의 모든 소수 인수 p에 대하여 '$U=\frac{2^n-1}{p}$, $D^U \neq 1$'인 비복원다항식이 ASR-D의 특성다항식이다.

증명

'$2^n-1=pU$'이고 '$C^p=D$'인 C가 존재하면, '$D^U=C^{pU}=1$'이 된다. 따라서 '$D^k=1$'이 되는 t가 '$t=2^n-1$'로 유일하기 위해서는 '2^n-1'의 모든 소수 인수 p에 대하여 '$U=\frac{2^n-1}{p}$, $D^U \neq 1$'이 되어야 한다. □

예 1 '$2^{32}-1=3\times5\times17\times257\times65537$'이다. $GF(2^n)$ 상의 비복원다항식 '$Pa(X)=0\text{x}197943fc9$'에서 $D=2$인 경우에 '$2^{5\times17\times257\times65537}=1$'이다. 따라서 $Pa(X)$는 ASR-2의 특성다항식이 아니다. 비복원다항식 '$Pb(X)=0\text{x}19fa0ff27$'에서는 $D=2$인 경우에 **보조정의 4.3**을 만족하므로 $Pb(X)$는 ASR-2의 특성다항식이다.

보조정의 4.5

$GF(2^n)$상에서 '2^n-1'이 소수이면 모든 비복원다항식은 ASR-D의 특성다항식이다.

증명

Fermat의 little theorem에 의하여 '$t=2^n-1,\ D^k=1$'이 된다. t가 소수이므로 '$D^k=1$'이 되는 t는 '$t=2^n-1$'로 유일하다. □

소수 가운데서 '2^n-1'형태의 소수를 Mersenne 소수라고 하며, $i=\{\cdots,13,17,19,31,61,89,107,127,521,\cdots\}$이 알려져 있다.

32 비트 컴퓨터에서 $GF(2^{32})$ 상의 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터를 C언어로 프로그램 한 것은 표 4와 같다.

표 4에서 32 비트 산술 시프트 레지스터는 AND 연산 1회, XOR 연산 1회, 시프트 연산 2회만을 수행하므로 총 4회의 연산이 필요하다. (int) 형변환 연산자는 해당하는 기계어를 선정하는 기능만을 수행하는 것으로 연산 시간을 소요하지 않는다. 이와 비교하여 갈로이 선형 궤환 시프트 레지스터는 AND 연산 2회, OR 연산 1회, XOR 연산 1회, 시프트 연산 2회를 수행하므로 총 6회의 연산이 필요하다. 그리고 피보나치 선형 궤환 시프트 레지스터는 XOR 연산 6회, OR 연산 1회, 시프트 연산 7회를 수행해서 총 14회 연산을 필요로 한다.

표 4. $GF(2^{32})$ **상의 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터의 C프로그램**

```
unsigned int P ; /* Characteristic polynomial */
unsigned int A, B ;

/* 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) ;
```

32 비트 컴퓨터에서 $GF(2^{64})$ 상의 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터를 C언어로 프로그램 한 것은 표 5와 같다.

표 5. $GF(2^{64})$ **상의 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터의 C프로그램**

```
unsigned int P[2] ; /* Characteristic polynomial */
unsigned int A[2], B, C ;

/* ASR-2 */
B = (int)A[1] >> 31 ;
A[1] = ((A[1] << 1) | (A[0] >> 31)) ^ (B & P[1]) ;
A[0] = (A[0] << 1) ^ (B & P[0]) ;

/* Galois-LFSR */
B = -(A[0] & 1) ;
C = A[1] ^ (B & P[1]) ;
A[0] = ((A[0] ^ (B & P[0])) >> 1) | (C << 31) ;
A[1] = (C >> 1) | (B << 31) ;

/* Fibonacci-LFSR */
B = A[1] ^ P[1] ^ A[0] ^ P[0] ;
B ^= B >> 16 ;
B ^= B >> 8 ;
B ^= B >> 4 ;
B ^= B >> 2 ;
B ^= B >> 1 ;
A[0] = (A[0] >> 1) | (A[1] << 31) ;
A[1] = (A[1] >> 1) | (B << 31) ;
```

32 비트 컴퓨터에서 $GF(2^{32})$ 상과 $GF(2^{64})$ 상에서 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터 각각이 수행하는 동작을 비교하여 나타낸 연산수는 표 6과 같다.

표 6. $GF(2^{32})$ **상과** $GF(2^{64})$ **상에서 ASR-2와 갈로이 및 피보나치 선형 궤환 시프트 레지스터의 동작속도 비교**

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

표 6으로부터 ASR-2가 갈로이 및 피보나치 선형 궤환 시트프 레지스터보다 소프트웨어로 구현시에 $GF(2^{32})$ 상에서는 필요한 연산수가 갈로이 선형 궤환 시프트 레지스터보다는 33%, 피보나치 선형 궤환 시프트 레지스터보다는 71% 적게 필요함을 알 수 있다. 그리고 $GF(2^{64})$ 상에서는 각각 18%, 53%가 적게 필요하다.

따라서 ASR-2는 기존의 갈로이 및 피보나치 선형 궤환 시프트 레지스터보다 동작속도가 빠르므로 효율적임을 알 수 있다.

ASR-D에서 D값이 작으면 소프트웨어 구현이 용이하다.

$GF(2^{32})$ 상에서 임의의 값 D에 대한 ASR-D를 C언어로 프로그램 한 것을 표 7과 같다.

표 7. $GF(2^{32})$ 상의 ASR-D의 C 프로그램

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

4.2 선형 복잡도

이 절에서는 산술 시프트 레지스터의 안전성을 검증하기 위해 선형 복잡도를 구하고, 갈로이 및 피보나치 선형 궤환 시프트 레지스터와 비교한다.

$GF(2^n)$ 상에서 ASR-D의 초기값 $A_0(x)$와 i번째의 값 $A_i(x)$ 및 특성 방정식 $P(x)$는 다음 식과 같이 표현할 수 있다.

$$A_0(x) = \sum_{k=0}^{n-1} a_{0,k} x^k, \qquad A_i(x) = \sum_{k=0}^{n-1} a_{i,k} x^k$$

$$P(x) = x^n \oplus \sum_{k=0}^{n-1} p_k x^k$$

ASR-2에서 출력 s_i와 $A_{i+1}(x)$의 계수는 다음과 같이 된다.

$$\begin{aligned} s_i &= a_{i,n-1} \\ a_{i+1,j} &= a_{i,j-1} \oplus (s_i \bullet p_j) \quad \text{for } j \in \{1,2,\cdots,n-1\} \\ a_{i+1,0} &= s_i \bullet p_0 \end{aligned}$$

위 식은 $2n$원 1차 연립 방정식이므로 $2n$개 출력 s를 알면 $A_0(x)$와 $P(x)$의 모든 계수를 풀이할 수 있다. 즉, ASR-2의 선형 복잡도는 n으로 갈로이 선형 궤환 시프트 레지스터 및 피보나치 선형 궤환 시프트 레지스터와 동일하다.

ASR-D에서 출력 s_j와 $A_{j+1}(x)$의 계수는 **보조정의 4.1**의 순환 행렬식에 의하여 다음과 같이 된다.

$$s_i = a_{i,n-1}$$

$$a_{i+1,j} = \sum_{k=0}^{n-1} m_{jk} \bullet a_{i,k} \qquad \text{for} \quad j \in \{0,1,2,\cdots,n-1\}$$

위 식은 미지수가 m_{ij}와 $a_{0,i}$인 'n^2+n'원 1차 연립 방정식이다. 그러므로 $GF(2^n)$상의 ASR-D의 선형 복잡도 LC는 다음 식과 같이 된다.

$$n \le LC \le \frac{n^2+n}{2}$$

제 5 장 S 박스와 G 함수 구성

SEED와 같은 형식을 가지는 블록 암호 알고리즘에서 사용할 수 있는 안전성이 높은 S 박스를 구성하고, G 함수의 특성을 분석하여 차분 분석법과 선형 분석법에 강한 G 함수를 구성한다.

5.1 S 박스 구성

SEED와 같은 형식을 가지는 대칭키 블록 암호 알고리즘을 공격하는 가장 뛰어난 공격 방법은 차분 분석법과 선형 분석법이다. 따라서 블록 암호 알고리즘에서 이런 공격에 강한 특성을 가지기 위해서는 안전성이 높은 S 박스를 구성해야 한다.

차분 분석법에 강한 특성을 지닌 S 박스를 구성하기 위해서는 만족해야 할 세가지 조건이 있는데 다음과 같다.

- 차분 XOR 테이블에서 모든 첫 번째 칼럼이 '0'이 되어야 한다.
- '0'인 성분의 수가 작아야 한다.
- 최대 성분 값이 작아야 한다.

이 세가지 조건을 만족하기 위해서 S 박스는 전단사함수가 되어야 하며, '0' 또는 '2'인 성분이 많아야 하고, 끝으로 최대성분 값이 '4'인 전단사

함수를 선정해야 한다. 또한, 선형공격에 강한 S 박스를 구성하기 위해서는 입력과 출력의 상관계수가 작아야 한다.

그 외에 S 박스는 $S(A)=A$ 인 고정점과 $S(A)=\overline{A}$ 인 역고정점이 없어야 한다. 왜냐하면 고정점과 역고정점은 S 박스가 치환기능을 수행하지 않는 입력으로 약한 입력이기 때문이다.

이들 조건을 만족하는 비선형 전단사함수를 찾기 위하여

$$nl(X) \Rightarrow X^{-1} \quad \text{over} \quad GF(2^8)$$

을 계산하였다. 단, 입력 '0'과 '1'은 예외적인 경우로 고정점 판단에서 제외한다. 조건을 만족하는 역수 계산 결과는 표 8과 같다.

표 8. 고정점과 역고정점을 가지지 않는 $GF(2^8)$ 상의 역수 특성

원시다항식	입출력 상관계수	원시다항식	입출력 상관계수
0x1a3	0.010	0x1cf	0.080
0x163	0.015	0x15f	0.080
0x18b	0.019	0x1a9	0.081
0x12b	0.020	0x165	0.111
0x171	0.037	0x17b	0.124
0x1dd	0.047	0x12d	0.133
0x19f	0.052	0x11d	0.182
0x1c3	0.066	0x1b1	0.196

표 8은 입출력 상관계수가 작은 순서대로 나열하였다.

$GF(2^8)$ 상의 역수는 다음과 같은 성질을 가진다.

- 차분 XOR 테이블의 최대값은 4이며, 각 행에서 오직 하나의 성분만이 4이며, 나머지는 0과 2로만 구성되어 차분공격에 강하다.
- 아핀 변환과 최소거리는 2^4 이다.
- 출력 비트들의 선형결합에 대한 비선형계수(Nonlinear order) 는 7이다.

S 박스는 비선형함수와 아핀 변환을 결합하여 식 (18)과 같이 표현된다.

$$S(X) = \left((X^{-1} \bmod Q) \times M \right) \bmod P + C \tag{18}$$

여기서, M 과 P 는 $GF(2^8)$ 상에서 서로소(Relatively prime) 이다.

아핀 변환은 차분공격과 선형공격의 특성에 변화를 주지는 않지만 수식의 복잡도를 증가시켜서 보간공격에 강한 특성을 나타내며, 비선형함수의 고정점을 없애는 기능을 수행한다.

식 (18)에서 고정점과 역고정점을 가지지 않으면서 상관계수를 최소로 하는 아핀함수를 구하여 정리한 것을 표 9에 나타내었다.

식 (18)에서 비선형함수와 아핀 변환의 원시다항식은 상호 다르게 선정한다.

표 9. 최적화된 S 박스 계수표

비선형함수 원시다항식 Q	아핀변환 곱항 M	법연산 생성다항식 P	아핀변환 상수 C	S 박스 입출력 상관계수	비고
0x1a3	0x38	0x1c5	0x94	0.000056	G1-S1
0x163	0xc8	0x159	0x3f	0.000303	G1-S2
0x18b	0xba	0x1cd	0x11	0.000112	G2-S1
0x12b	0x71	0x182	0xd9	0.000296	G2-S2
0x171	0xc1	0x1a4	0x40	0.000411	G3-S1
0x1dd	0x4d	0x1fc	0x2f	0.000149	G3-S2
0x19f	0x84	0x17b	0x09	0.000095	
0x1c3	0xf4	0x1e1	0x08	0.000130	
0x1cf	0x76	0x127	0x63	0.000400	
0x15f	0x44	0x16f	0x2c	0.000202	
0x1a9	0x43	0x1bf	0x2e	0.000518	
0x165	0x06	0x11b	0x18	0.000450	
0x17b	0xc8	0x1bf	0x67	0.000503	
0x12d	0x45	0x167	0x28	0.000318	
0x11d	0x35	0x1e1	0xdd	0.000442	
0x1b1	0x08	0x187	0x56	0.000020	

5.2 G 함수의 특성

G 함수는 32 비트 입력 X를 32 비트 출력 Z로 변환하는

$$G : X \rightarrow Z$$

함수로 정의 할 수 있으며 다음과 같은 특징을 가진다.

5.2.1 전단사함수

SEED의 F 함수를 변수 T0과 T1에 아래바 첨자를 붙여서 변화하는 모습을 알기 쉽도록 표현하면 다음과 같이 된다.

$$\begin{aligned}
&\mathrm{T0_0} = \mathrm{R0} \ \hat{} \ \mathrm{roundkey[Ki][0]};\\
&\mathrm{T1_0} = \mathrm{R1} \ \hat{} \ \mathrm{roundkey[Ki][1]};\\
&\\
&\mathrm{T1_1} = \mathrm{T0_0} \ \hat{} \ \mathrm{T1_0} \ ;\\
&\mathrm{T1_2} = \mathrm{G(T1_1)};\\
&\mathrm{T0_1} = \mathrm{T0_0} + \mathrm{T1_2} \ ;\\
&\mathrm{T0_2} = \mathrm{G(T0_1)};\\
&\mathrm{T1_3} = \mathrm{T0_2} + \mathrm{T1_2} \ ;\\
&\mathrm{T1_4} = \mathrm{G(T1_3)};\\
&\mathrm{T0_3} = \mathrm{T0_2} + \mathrm{T1_4} \ ;\\
&\mathrm{R0}' = \mathrm{T0_3} \ ;\\
&\mathrm{R1}' = \mathrm{T1_4} \ ;
\end{aligned}$$

G 함수의 출력 Z의 상태수 N_Z가

$$N_Z < 2^{32}$$

이면, T1_2와 T0_2의 상태수가 N_Z가 되며, 이어서 T1_3, T1_4 그리고 T0_3의 상태수가 2^{32}보다 작아진다. 이러한 현상을 방지하기 위해서는 G 함수의 출력 Z의 상태수는 반드시 2^{32}이 되어야 한다.

한편 G 함수는 X입력을 Z출력으로 변환하는 기능을 수행하는 함수로 차분공격에 강하기 위해서는 단사함수가 되어야 한다. 따라서 G 함수는 전단사함수가 되어야 한다.

5.2.2 MDS 코드

G 함수는 그림 12와 같이 S 박스와 $GF(2^8)$상의 4×4 행렬식으로 표현되는 확산선형변환 부분의 결합으로 표현할 수 있다.

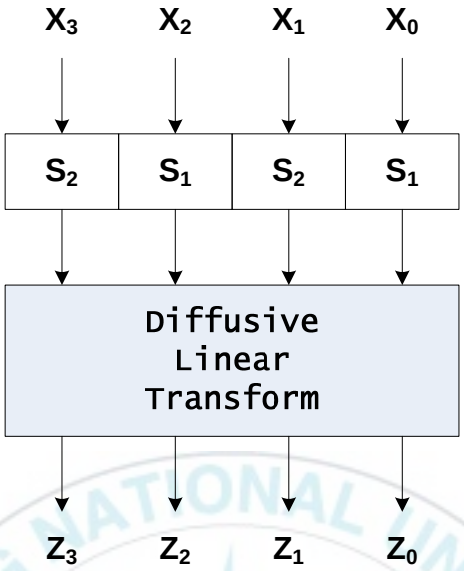


그림 13. G 함수 구조도

그림 13의 확산선형변환 부분은 식 (19)의 행렬식으로 표현된다.

$$\begin{vmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{vmatrix} = \begin{vmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{23} & A_{33} \end{vmatrix} \cdot \begin{vmatrix} S_1(X_0) \\ S_2(X_1) \\ S_1(X_2) \\ S_2(X_3) \end{vmatrix} \tag{19}$$

32 비트 입력 X는 4개의 8 비트

$$Z_i \in \{Z_3, Z_2, Z_1, Z_0\}$$

로 출력된다. X_3와 Z_3가 최상위 바이트이다.

식 (19)의 확산선형변환 행렬식은 차분공격과 선형공격에 강하도록 설계되어야 한다. 이를 위해서는 확산선형변환 행렬식의 차분 확산 계수와 선형 확산 계수가 최대값을 가지도록 설계 해야 한다.

식 (19)의 G 함수에서 S 박스의 8 비트 단위 출력을

$$P_i \in \{P_3, P_2, P_1, P_0\}$$

라 하면, 확산선형변환 행렬식은 P_i를 확산시켜서 출력

$$Z_i \in \{Z_3, Z_2, Z_1, Z_0\}$$

를 생성한다.

$W_h(a)$를 $GF(2^8)$상의 a의 해밍 가중치라고 하면,

$$W_h(a) = \begin{cases} 0 & \text{if } a = 0 \\ 1 & \text{if } a \neq 0 \end{cases}$$

이 된다.

확산선형변환 행렬식의 차분 확산 계수 (Differential Branch Number)를 B_d라 하면, B_d는 식 (20)과 같이 정의할 수 있다.

$$B_d = \min\left(\sum_{i=0}^{3} W_h(P_i) + \sum_{i=0}^{3} W_h(Z_i)\right) \quad (20)$$
여기서, $P \neq 0$

식 (20)에서 $\sum W_h(P_i)$와 $\sum W_h(Z_i)$의 최대값은 4이고, 최소값은 1이 된다. 따라서 차분 확산 계수는

$$B_d \leq 5$$

가 된다. 차분 확산 계수는 입력 P_i가 변화하였을 때 출력 Z_i가 변화하는 $GF(2^8)$ 상의 코드 수의 최소값을 나타낸다. 차분공격에 있어서 비활성 S 박스 입력의 차분은 '0'이다. 따라서 출력의 차분도 '0'이다. SEED의 F 함수는 그림 2와 같이 3개의 G 함수를 직렬로 연결한 구조이다. 따라서 차분공격에 강하기 위해서는 G 함수의 차분 확산 계수는 최고값인 5가 되어야 한다.

한편 확산선형변환 행렬식의 선형 확산 계수 (Linear Branch Number) B_l은 식 (21)과 같이 정의 할 수 있다.

$$B_l = \min\left(\sum_{i=0}^{3} W_h(A_i) + \sum_{i=0}^{3} W_h(B_i)\right) \quad (21)$$
여기서, $B \neq 0$

식 (21)에서 A_i와 B_i는 각각 입력 마스트 값과 출력 마스크 값을 나타

내며, 식 (21)의 확산선형변환 행렬식을 M 이라고 하면 입력 마스크 값 A 는 다음 식과 같다.

$$A = M^t \times B$$

식 (21)에서 $\sum W_h(A_i)$ 와 $\sum W_h(B_i)$ 의 최대값은 4이고, 최소값은 1이 된다. 따라서 선형 확산 계수는

$$B_l \leq 5$$

가 된다. 선형 확산 계수는 출력 Z_i 에 대하여 선형 결합된 입력 P_i 의 $GF(2^8)$ 상의 코드 수의 최소값을 나타낸다. SEED의 F 함수는 그림 2와 같이 3개의 G 함수를 직렬로 연결한 구조이다. 따라서 선형공격에 강하기 위해서는 G 함수의 선형 확산 계수가 최고값인 5가 되어야 한다.

차분 확산 계수와 선형 확산 계수가 최고값을 가지는 코드가 제 3 장에서 소개한 Maximum Distance Separable 코드 또는 줄여서 MDS 코드 이다. 이상에서 본대로 G 함수는 MDS 코드를 생성해야 차분 분석법과 선형 분석법의 특성에 강하다.

확산선형변환 행렬식 M 이 MDS 코드를 생성하려면 대칭행렬이 되어야 한다. 식 (20)과 식 (21)로부터 확산선형변환 행렬식 M 이 대칭행렬이면

$$B_d = B_l$$

이 된다.

본 논문에서는 확산선형변환 행렬식을 대칭행렬로 구성하기 위하여 한 행의 성분 4개를 선정하고, 아래 행은 윗 행의 성분을 왼쪽으로 회전시켜서 구성한다. 이렇게 구성한 G 함수를 식 (22)에 나타내었다.

$$\begin{vmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{vmatrix} = \begin{vmatrix} A_0 & A_1 & A_2 & A_3 \\ A_1 & A_2 & A_3 & A_0 \\ A_2 & A_3 & A_0 & A_1 \\ A_3 & A_0 & A_1 & A_2 \end{vmatrix} \bullet \begin{vmatrix} S_1(X_0) \\ S_2(X_1) \\ S_1(X_2) \\ S_2(X_3) \end{vmatrix} \tag{22}$$

5.2.3 SAC

식 (19)의 G 함수에서 두 개의 입력 X 와 X_i 에 대하여, X_i 는 $i(0 \le i \le 31)$ 비트만이 X 와 다르다고 할 때, V_{ij} 를 Z 의 $j(0 \le j \le 31)$ 비트가 변경될 확률이라고 정의하면, 입력 $X \in \{0,1,2,\cdots,2^{32-1}\}$ 에서 V_{ij} 가 평균값 0.5를 가지는 정규분포를 이루면 SAC를 만족시킨다고 한다.

선형공격 및 차분공격에 강하기 위해서는 입력의 작은 비트의 변화가 출력의 많은 비트에 나타나야 하며, 또한 입력의 많은 비트의 변화가 출력의 적은 비트의 변화로 나타나야 한다.

SEED의 F 함수는 그림 2와 같이 G 함수의 출력이 다음 G 함수의 입력이 되는 구조이다. 또한 입력은 3개의 G 함수를 거쳐서 출력이 된다. 이와 같은 구조에서 G 함수가 SAC를 만족하지 않으면 입력의 변화가 일부 출력에만 나타나고, 이러한 과정이 3번 반복되면 출력의 특정 부분에 만 변화가 집중될 수 있다.

SEED에서는 G 함수의 출력에 대하여 덧셈 연산을 하여서, 캐리 전파에 의한 확산을 하고 있다. 그런데 캐리 전파는 더하는 두 개의 비트가 '0'과 '1'인 경우에만 발생하므로 캐리 전파 확률은 50%이다. 즉, 덧셈 연산의 캐리 전파만으로는 충분한 확산을 기대할 수 없다. 따라서 입력의 변화를 출력에 충분히 확산시키기 위해서는 G 함수가 SAC를 만족시켜야 한다.

5.2.4 약한 입력

G 함수는 X 입력을 Z 출력으로 변환하는 32 비트 치환 블록이다. 따라서 '$G(X) = X$'인 고정점과 '$G(X) = \overline{X}$'인 역고정점은 약한 입력이 된다. 또한 F 함수는 G 함수 출력에 대하여 덧셈 연산을 수행한다. '$G(X) = -X$'가 되는 입력 X에 대해서 F 함수는

$$
\begin{aligned}
&\mathrm{T0\_0 = R0 \hat{} roundkey[Ki][0];}\\
&\mathrm{T1\_0 = R1 \hat{} roundkey[Ki][1];}\\
&\\
&\mathrm{T1\_1 = T0\_0 \hat{} T1\_0\ ;}\\
&\mathrm{T1\_2 = G(T1\_1);}\\
&\mathrm{T0\_1 = T0\_0 + T1\_2\ ;}\\
&\mathrm{T0\_2 = G(T0\_1) = -T0\_1 = -T0\_0 - T1\_2\ ;}\\
&\mathrm{T1\_3 = T0\_2 + T1\_2 = -T0\_0\ ;}\\
&\mathrm{T1\_4 = G(T1\_3) = T0\_0\ ;}\\
&\mathrm{T0\_3 = T0\_2 + T1\_4}\\
&\qquad \mathrm{= -T0\_0 - T1\_2 + T0\_0 = -T1\_2\ ;}\\
&\mathrm{R0' = T0\_3 = -(T0\_0 \hat{} T1\_0);}\\
&\mathrm{R1' = T1\_4 = T0\_0\ ;}
\end{aligned}
$$

가 된다. 따라서 '$G(X) = -X$'가 되는 입력은 약한 입력이다.

G 함수가 약한 입력을 가지게 되면 약한 입력에 대해서는 치환기능을 수행하지 않는다. 따라서 안전성이 좋은 G 함수가 되기 위해서는 이런 약한 입력을 가지지 않아야 한다.

5.2.5 간단한 하드웨어

그림 13을 하드웨어로 구현할 때 S 박스는 8×8 ROM으로 구성하고, 확산선형변환 부분은 2 입력 XOR 게이트 어레이로 구현할 수 있다. 확산선형변환 부분은 비규칙적인 회로가 되므로 반도체 상에서 넓은 면적을 차지하게 된다. 따라서 G 함수의 회로를 간단하게 구성하기 위해서는 확산선형변환 행렬식을 구성하는 2 입력 XOR 게이트 어레이 부분의 회로가 간단해야 된다.

5.3 G 함수의 생성

SEED의 G 함수는 두 종류의 S 박스를 각각 두 개씩 사용하여 그림 2와 같이 G 함수를 세 번 반복 사용하여 연산한다. 본 논문에서는 F 함수에서 서로 다른 세 개의 G 함수를 사용하여 수식의 복잡도를 높이는 경우를 고려하여 세 개의 G 함수, G_1, G_2 그리고 G_3 함수를 생성한다. 각 G 함수에 사용되는 두 종류의 S 박스는 S 박스의 비선형함수의 입출력 상관계수가 낮은 순서대로 선정하였다. 각 G 함수에 사용한 S 박스를 표 9의 비고란에 표시하였다.

식 (19)의 확산선형변환 행렬식은 Z_i에 대한 함수로 Z_0는 다음 식과

같이 표현된다.

$$Z_0 = \sum \left(A_i \times S_i \left(X_i \right) \right) \mod A_p$$

여기서, A_i는 $GF(2^s)$상에서 곱항으로 확산선형변환 행렬식의 한 성분이고, A_p는 $GF(2^s)$상의 원시다항식이다. $GF(2^s)$상의 곱셈은 곱항과 원시다항식이 정해지면 $GF(2)$상의 행렬식으로 표현할 수 있다.

$GF(2)$상의 n 비트 수 A와 B 및 원시다항식 P는 다음과 같이 다항식으로 표현 할 수 있다.

$$A = \sum_{i=0}^{n-1} a_i x^i, \quad B = \sum_{i=0}^{n-1} b_i x^i, \quad P = \sum_{i=0}^{n-1} p_i x^i$$

단위함수 $u(\,)$를 다음과 같이 정의하면,

$$u(i,j,m) = \begin{cases} 1 & \text{for} \quad m = i + j \\ 0 & \text{for} \quad m \neq i + j \end{cases}$$

A와 B의 곱 C는

$$C = A \times B = \sum_{i=0}^{2n-2} c_i x^i$$
$$c_m = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} u(i,j,m) \bullet a_i \bullet b_j$$
단, $m \in \{0,1,2,\cdots 2n-2\}$

이다.

곱 C를 $GF(2^s)$상의 원시다항식 P로 나눈 나머지는 다음과 같은 방식으로 계산할 수 있다.

$$\text{for } (\text{i} = 2\text{n-2} \text{ ; i} > \text{n-1} \text{ ; i}--)$$
$$\text{for } (\text{j} = 0 \text{ ; j} < \text{n+1} \text{ ; j}++)$$
$$c_{ij} = c_i \bullet p_{n-j} + c_{i-j}$$

이렇게 계산된 $c_i \in \{c_{n-1}, c_{n-2}, \cdots, c_0\}$는 식 (23)과 같이 표현할 수 있다.

$$c_i = \sum_{j=0}^{n-1} f_j(b_{n-1}, b_{n-2}, \cdots, b_0, p_{n-1}, p_{n-2}, \cdots, p_0) \bullet a_j \quad (23)$$

식 (23)에서 B와 P가

$$b_i \in \{0,1\}, \quad p_i \in \{0,1\}$$

로 주어지면

$$f_j(\cdots) \in \{0,1\}$$

가 된다. 따라서 $GF(2)$ 상의 A 와 B 의 곱은 $f_j(\cdots)$ 를 성분으로 하는 $GF(2)$ 상의 행렬식으로 표현할 수 있다. 그림 14와 그림 15에 행렬식의 예를 나타내었다.

$$\begin{vmatrix} C_7 \\ C_6 \\ C_5 \\ C_4 \\ C_3 \\ C_2 \\ C_1 \\ C_0 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{vmatrix} \times \begin{vmatrix} A_7 \\ A_6 \\ A_5 \\ A_4 \\ A_3 \\ A_2 \\ A_1 \\ A_0 \end{vmatrix}$$

그림 14. $GF(2^8)$**상 곱셈** $C = (A \times 0\text{xe1}) \bmod 0\text{x1c3}$**의** $GF(2)$**상 행렬식 표**

$$\begin{vmatrix} C_7 \\ C_6 \\ C_5 \\ C_4 \\ C_3 \\ C_2 \\ C_1 \\ C_0 \end{vmatrix} = \begin{vmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} A_7 \\ A_6 \\ A_5 \\ A_4 \\ A_3 \\ A_2 \\ A_1 \\ A_0 \end{vmatrix}$$

그림 15. $GF(2^8)$**상 곱셈** $C = (A \times 0\text{x33}) \bmod 0\text{x1cf}$**의** $GF(2^8)$**상 행렬식 표**

그림 14는 간단한 선형변환행렬식의 예이고, 그림 15는 복잡한 선형변환행렬식의 예를 나타내고 있다. 그림 14와 그림 15의 행렬식에서 각 행의 '1' 성분은 c_i 생성식의 항을 나타내며, '1'의 성분수가 N 개이면, c_i 를 생성하기 위해서는 '$N-1$'개의 2 입력 XOR 게이트가 필요하다. 또한 전달 지연 시간은

$$\lfloor \log_2 N \rfloor$$

가 된다.

하드웨어를 최소로 하는 G 함수의 확산선형변환 행렬식을 생성하는 알고리즘을 표 10에 나타내었다.

표 10. G 함수의 확산선형변환 행렬식 생성 알고리즘

단계 1) 2 입력 XOR 하드웨어를 최소로 필요로 하는 $GF(2^8)$ 상의 행렬식 성분 4개를 무작위로 발췌하여 확산선형변환 행렬식을 구성한다.
단계 2) 행렬식의 역행렬식을 구한다. 역행렬식이 구해지지 않으면 전단사함수가 되지 않으므로 단계 1로 되돌아간다.
단계 3) 역행렬식의 모든 성분이 '0'이 아닌 것을 확인한다. '0'인 성분이 있으면 확산이 제대로 이루어지지 않으므로 단계 1로 되돌아간다.
단계 4) 확산산선형변환 행렬식이 입력 $P \in \{1,2,3,\cdots,2^{32}-1\}$ 에 대하여 식 16의 차분 확산 계수가 5인가를 판단한다. 5미만이면 MDS 코드를 생성하지 않으므로 단계 1로 되돌아간다.
단계 5) $X \in \{1,2,3,\cdots,2^{32}-1\}$ 에 대하여 고정점 '$G(X)=X$'와 역고정점 '$G(X)=\overline{X}$' 및 '$G(X)=-X$'가 되는 약한 입력점을 가지지 않는 것을 확인한다. 약한 입력을 가지면 단계 1로 되돌아간다.

표 10의 알고리즘을 적용하여 생성한 G 함수를 표 11에 보인다.

표 11. S 박스와 확산선형변환 행렬식의 G 함수

	S 박스	확산선형변환 행렬식
G1	$S_1 = \{0x1a3, 0x38, 0x1c5, 0x94\}$ $S_2 = \{0x163, 0xc8, 0x159, 0x3f\}$	$A_i = \{0x04, 0x01, 0x01, 0xc3\}$ $A_p = 0x187$
G2	$S_1 = \{0x18b, 0xba, 0x1cd, 0x11\}$ $S_2 = \{0x12b, 0x71, 0x182, 0xd9\}$	$A_i = \{0x02, 0x01, 0xa2, 0x02\}$ $A_p = 0x187$
G3	$S_1 = \{0x171, 0xc1, 0x1a4, 0x40\}$ $S_2 = \{0x1dd, 0x4d, 0x1fc, 0x2f\}$	$A_i = \{0x02, 0x02, 0x01, 0x91\}$ $A_p = 0x187$

표 11에서 $S_n = \{Q, M, P, C\}$는 식 (18)의 Q, M, P, 및 C를 각각 나타낸다.

$A_i = \{A_0, A_1, A_2, A_3\}$는 식 (22)에서 $GF(2^8)$ 상의 4×4 행렬식의 각 성분을 나타내며, A_p는 행렬식의 원시다항식이다.

제 6 장 NSEED 구조 및 평가

이 장에서는 제 5 장에서 제안한 결과를 바탕으로 구성된 S 박스와 G 함수를 사용해서, 새로 제안한 NSEED 블록 암호 알고리즘의 구조를 기술하고 NSEED의 성능을 평가한다. NSEED는 NEW SEED라는 의미이다.

6.1 NSEED 블록 암호 알고리즘의 구조

기본 구조 새로운 블록 암호 알고리즘 NSEED의 전체 구조는 피스탈 네트워크(Fesitel Network)이다. 128 비트 평문 블록과 암호화 키로부터 생성된 라운드 키를 입력 받아서 128 비트 암호문 블록으로 변환한다. NSEED는 128 비트, 256 비트 크기의 키 중에서 선택해서 사용할 수 있고, 라운드 수는 키의 길이에 따라 4, 6 라운드로 수행된다. 각 라운드에서 사용되는 라운드 키는 암호키를 사용하여 산술 시프트 레지스터와 G 함수를 적용하여 생성한다. 따라서 하드웨어 자원이 제한적인 시스템도 암호키를 사용하여 암호화와 복호화에 필요한 라운드 키를 간단히 생성해서 사용할 수 있다. 내부 함수인 G 함수는 SPS(Substitution Permutation Substitution) 함수로 구성하였다. 블록의 크기를 8 비트 블록단위로 했을 때 키 크기에 따른 블록의 크기와 라운드 수는 표 12와 같다. NSEED의 전체 구조는 그림 16과 같다.

표 12. NSEED의 키 크기에 따른 라운드 수

구분	블록 크기(Byte)	키 크기(Byte)	라운드 수
NSEED-128	16	16	4
NSEED-256	16	32	6

라운드 함수 F NSEED의 라운드 함수인 F 함수는 세 개의 G 함수와 두 개의 산술 시프트 레지스터로 구성되어 있다. F 함수의 입력 64 비트는 32 비트씩 나누어져서 세 개의 G 함수를 통과하면서 변환을 한다. G 함수는 수식의 복잡도를 높이는 경우를 생각해서 서로 다른 세 개의 G 함수, G1, G2 그리고 G3로 구성하였다. 세 개의 G 함수를 통과한 후에 한 쪽은 산술 시프트 레지스터 2(ASR-2, Arithmetic Shift Register 2)를 사용하고 다른 쪽은 산술 시프트 레지스터 3(ASR-3, Arithmetic Shift Register3)을 사용해서 출력을 낸다. F 함수에서 사용된 산술 시프트 레지스터의 원시다항식은 0x19fa0ff27이다. F 함수의 구조는 그림 17과 같다.

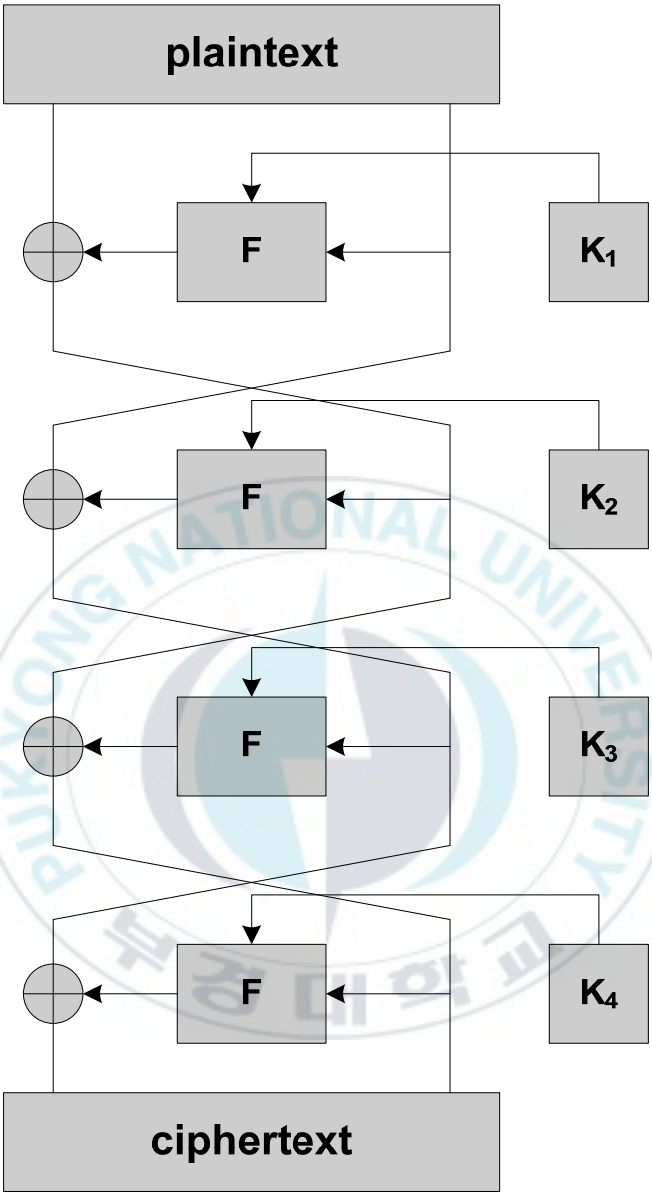


그림 16. NSEED 전체 구조도

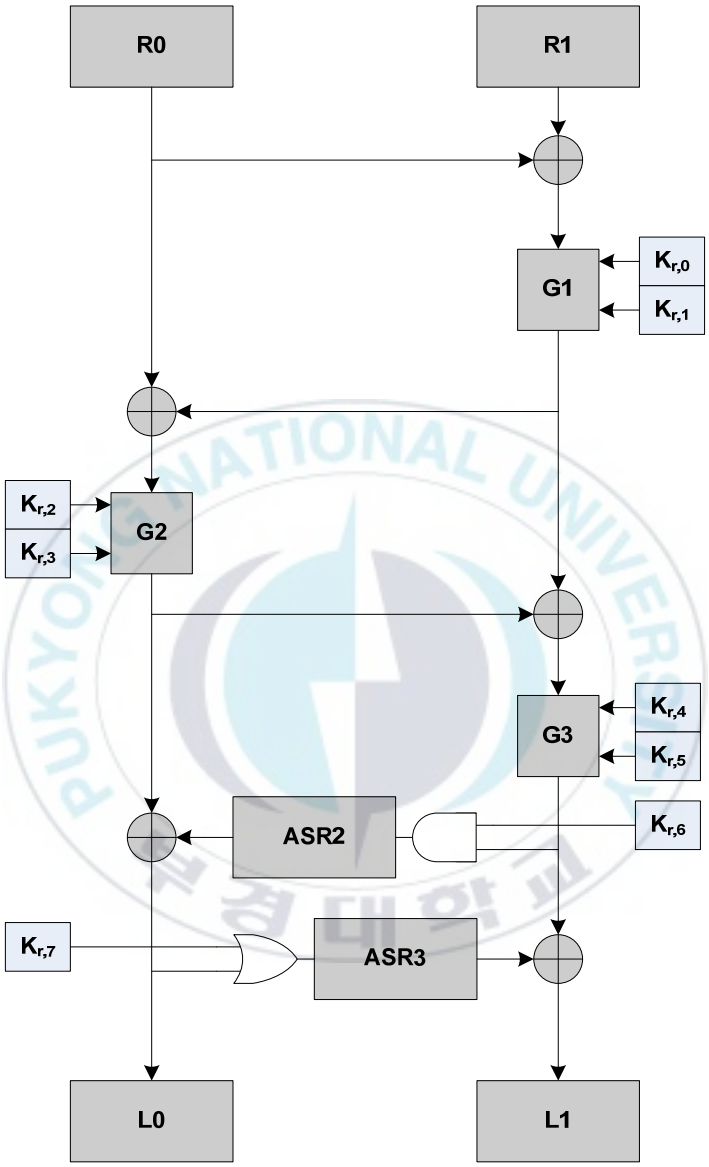


그림 17. F 함수 구조도

내부 함수 G 내부 함수인 G 함수는 다음과 같이 3 개의 층으로 이루어진 SPS(Substitution Permutation Substitution) 함수 구조로 되어 있다.

- 첫 번째 치환 계층(Substitution Layer)
- 교환 계층(Permutation Layer)
- 두 번째 치환 계층(Substitution Layer)

첫 번째 치환 계층(Substitution Layer)은 4개의 8 비트 S 박스로 구성되어 있고, 32 비트의 입력이 들어와서 이 곳을 통과하면서 변환을 한 후 출력된다. 여기서 사용된 S 박스는 차분 분석법과 선형 분석법에 강하도록 전단사함수로 구성하고, 역고정점과 고정점이 없도록 구성하기 위하여 기약 다항식(Irreducible Polynomial) $x^8 + x^4 + x^3 + x^2 + 1$ (hex: 0x11d)의 역수를 취하고, 아핀변환을 하여 구성하였다. S 박스를 통과해 나온 출력은 교환 계층(Permutation Layer) 에서 선형변환행렬을 사용해서 확산을 한다. 이때 사용하는 선형변환행렬은 차분 분석법과 선형 분석법에 강한 특성을 가지기 위해 차분 확산 계수 B_d와 선형 확산 계수 B_l을 $B_d = B_l = 5$가 되도록 구성하였다. 따라서 선형변환행렬은 MDS 코드를 생성한다. 선형변환행렬을 사용해서 확산을 한 결과는 다시 4개의 8 비트 S 박스로 구성되어 있는 두 번째 치환 계층을 통과하여 출력을 낸다. 첫 번째 치환 계층과 두 번째 치환 계층에 사용하는 S 박스는 서로 다르게 구성하였다. 새로 구성한 S 박스와 선형변환행렬은 표 11에 나타낸 것과 같다. G 함수의 구조는 그림 18과 같다.

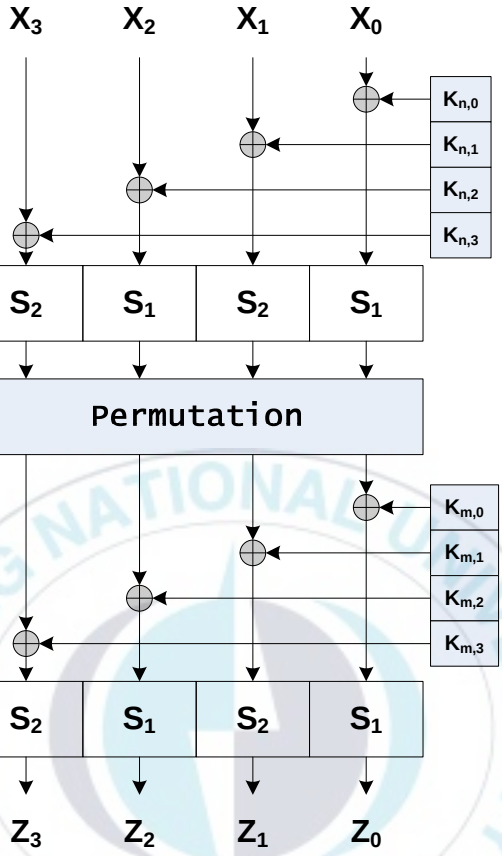


그림 18. G 함수 구조도

키 스케줄링 NSEED는 가변적 키를 가진다. 키의 크기는 128 비트, 256 비트 중에서 선택할 수 있다. 키 스케줄링 알고리즘은 128 비트 키를 선택했을 때 키를 32 비트씩 4개로 나눈 후, 이들 가운데 교대로 두 개씩을 산술 시프트 레지스터 (ASR, Arithmetic Shift Register)를 통과 시킨 후 8 비트씩 좌/우로 회전이동을 하여 다음 라운드 키 생성을 위한 입력으로 사용한다. 그리고 이 4 워드들을 두 개씩 XOR 연산을 한 후 G 함수를 적

용하여 라운드 키를 생성한다. 256 비트 키를 사용할 경우에는 32 비트씩 8개로 나눈 후 같은 방법을 사용해서 라운드 키를 생성한다. 키 스케줄링 알고리즘은 자원이 제약적인 시스템도 간편하게 키를 생성하기 위하여, 암호화나 복호화 때 암호키로부터 필요한 라운드 키를 간단히 계산할 수 있도록 하였다.

라운드 키는 다음 과정을 통해서 생성된다.

① 128 비트 입력키를 32 비트씩 4개의 조각으로 나눈다. 4개의 조각을 K_0, K_1, K_2, K_3이라고 하자.

② $K_{1,0} = G(K_1 \oplus K_3 \oplus CK_0)$, $K_{1,1} = G(K_0 \oplus K_2 \oplus CK_0)$ 로 1라운드 키를 생성한다. 여기서 CK_0는 라운드 상수이다.

③ K_2, K_3을 산술 시프트 레지스터에 적용한 후, 그 결과를 논리적 OR 연산한 다음 좌측으로 8 비트 회전 이동한다.

④ $K_{2,0} = G(K_1 \oplus K_3 \oplus CK_1)$, $K_{2,1} = G(K_0 \oplus K_2 \oplus CK_1)$ 로 2라운드 키를 생성한다.

⑤ K_0, K_1을 산술 시프트 레지스터에 적용한 후 그 결과를 논리적 OR 연산한 다음 우측으로 8 비트 회전 이동한다.

⑥ $K_{3,0} = G(K_1 \oplus K_3 \oplus CK_2)$, $K_{3,1} = G(K_0 \oplus K_2 \oplus CK_2)$ 로 3라운드 키를 생성한다.

⑦ K_2, K_3을 산술 시프트 레지스터에 적용한 후 그 결과를 논리적 OR 연산한 다음 좌측으로 8 비트 회전 이동한다.

⑧ $K_{4,0} = G(K_1 \oplus K_3 \oplus CK_3)$, $K_{4,1} = G(K_0 \oplus K_2 \oplus CK_3)$ 로 4라운드 키를 생성한다.

키 스케줄링 알고리즘에서 사용된 산술 시프트 레지스터는 ASR-2와 ASR-3이다. 그리고 키 스케줄링 알고리즘에서 사용한 산술 시프트 레지스터의 원시 다항식은 $0\times13f258cdb$ 이다. 여기서 CK_i 는 라운드 상수이다. 첫 라운드의 라운드 상수값은 $0\times9e3779b9$ 로 정했다. 다음 라운드의 라운드 상수는 이전 라운드의 상수를 산술 시프트 레지스터를 통과시킨 결과를 사용한다. 이때 사용한 산술 시프트 레지스터는 ASR-2를 사용한다. 그리고 라운드 상수를 생성하는데 사용한 산술 시프트 레지스터의 원시다항식은 $0\times1cf9fc95f$ 이다. 각 라운드에 사용한 라운드 상수는 표 13에 나타낸 것과 같다. 키 스케줄링의 구조도는 그림 19와 같다.

표 13. 각 라운드에 사용된 라운드 상수 값

라운드 상수 (CK_i)	
CK_0 = 0x9e3779b9	CK_3 = 0x50fb7a0a
CK_1 = 0xf3f13a2d	CK_4 = 0xa1f6f414
CK_2 = 0x287dbd05	CK_5 = 0x8c722177

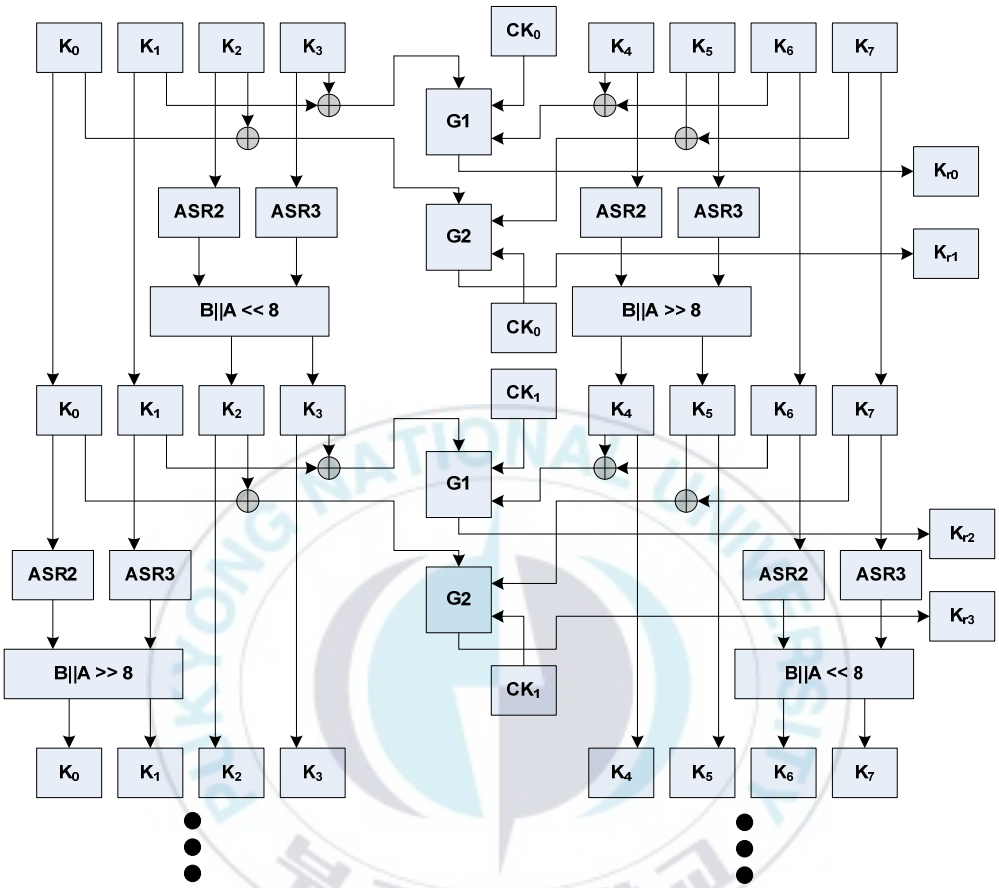


그림 19. 키 스케줄링 구조도

6.2 NSEED의 안전성 평가

이 절에서는 블록 암호 알고리즘의 안전성을 평가하는 다른 연구들의 결과를 NSEED에 적용하여 NSEED가 지니고 있는 차분 분석법과 선형 분석법에 대한 저항성을 평가한다.

m 개의 입력 그리고 출력 비트를 가진 S 박스를 S 라 하자. 즉, $S : Z_2^m \rightarrow Z_2^m$ 이다. 그러면 차분 그리고 선형 확률은 다음과 같이 정의된다.

정의 1[46]

임의의 주어진 값 Δx, Δy, Γx, $\Gamma y \in \mathbb{Z}_2^m$ 에 대해서 각 S 박스들의 차분 그리고 선형 확률은

$$DP^s(\Delta x \rightarrow \Delta y) = \frac{\#\{x \in \mathbb{Z}_2^m \mid S(x) \oplus S(x \oplus \Delta x) = \Delta y\}}{2^m}$$

그리고

$$LP^s(\Gamma x \rightarrow \Gamma y) = \left(\frac{\#\{x \in \mathbb{Z}_2^m \mid \Gamma x \bullet x = \Gamma y \bullet S(x)\}}{2^{m-1}} - 1 \right)^2$$

에 의해서 정의된다. 여기서 $\Gamma x \bullet x$ 는 Γx 와 x 의 비트간 XOR 연산의 패리티를 나타낸다.

S 박스가 차분 분석법과 선형 분석법에 충분히 강하기 위해서는 DP^s 와 LP^s 가 임의의 입력 차 $\Delta x \neq 0$ 그리고 출력 마스크 값 $b \neq 0$ 에 대해

충분히 작아야 한다. 따라서 S 박스의 저항성을 나타내는 최대 차분 그리고 선형 확률은 다음과 같이 정의된다.

정의 2[46]

S 박스의 최대 차분 그리고 선형 확률은

$$DP^{S}_{\max} = \max_{\Delta x \neq 0, \Delta y} DP^{S}(\Delta x \rightarrow \Delta y)$$

그리고

$$LP^{S}_{\max} = \max_{\Gamma x, \Gamma y \neq 0} LP^{S}(\Gamma x \rightarrow \Gamma y)$$

로 각각 정의된다.

NSEED의 G 함수에서 사용하는 S 박스는 8×8 이고, 차분 분석법에 강하기 위해 S 박스의 차분 XOR 테이블의 최대 성분 값이 4이며, 오직 하나의 성분만 4이고, 나머지 성분은 0과 2로 구성되어 있다. 그리고 선형 분석법에도 강하기 위해 비선형함수와 아핀변환을 이용하였는데, 아핀변환의 최소거리는 2^4 이다. 따라서 NSEED의 최대 차분 그리고 선형 확률은 각각

$$DP^{S}_{\max} = 2^{-6}, \quad LP^{S}_{\max} = 2^{-6}$$

이다.

S 박스에 대한 최대 차분 그리고 선형 확률뿐만 아니라 확산도 SPS 함수의 확률 안전성을 측정하기 위해 주어진 중요한 요소이다.

SPS 함수에서 차분 그리고 선형으로 활동하는 S 박스의 최소수는

$$n_d(D) = \min_{\Delta x \neq 0}\{Hw(\Delta x) + Hw(\Delta y)\}$$

그리고

$$n_l(D) = \min_{\Gamma y \neq 0}\{Hw(\Gamma x) + Hw(\Gamma y)\}$$

로 각각 정의된다.

정리 1[53]

확산층 D의 $n \times n$ 선형변환행렬을 M 이라 하자. 만약, M 의 각 $k \times k$ 정방부분행렬의 계수가 k 이면 $n_d(D) = n + 1$ 이다. 여기서 모든 k 는 $1 \leq k \leq n$ 이다.

정리 1의 확산층에 해당하는 것이 NSEED의 교환 계층이다. NSEED의 교환 계층에서 구성한 8×8 행렬은 **정리 1**을 만족하므로, NSEED의 차분 그리고 선형 활동성을 가지는 S 박스의 최소수는 $n_d(P) = 5,\ n_l(P) = 5$ 이다.

따라서 NSEED에서 사용한 G 함수의 안전성은 다음의 두 정리에 의해서 증명될 수 있다.

정리 2[53]

각 라운드의 입력 데이터와 XOR 연산을 수행한 라운드 키들은 의존적이지 않고 균일하게 분포되어 있다고 가정하자. 만약, $n_d(D) = n + 1$ 이면 SDS 함수의 각 차분 확률은 $(DP^s_{\max})^n$ 에 의해서 결정된다.

정리 3[53]

각 라운드의 입력 데이터와 XOR 연산을 수행한 라운드 키들은 의존적이지 않고 균일하게 분포되어 있다고 가정하자. 만약, $n_l(D) = n + 1$ 이면 SDS 함수의 각 선형 hull 확률은 $(LP^s_{\max})^n$ 에 의해서 결정된다.

만약, NSEED의 라운드 키가 의존적이지 않고 균일하게 분포한다고 가정하면, G 함수의 최대 차분 그리고 선형 확률은 위 정리들에 의해서

$$n_d(G) = n_l(G) = (2^{-6})^5 = 2^{-30}$$

에 의해서 결정된다.

다음으로 Difference Distribution 공격을 NSEED에 적용해 보도록 하겠다. 먼저 이 공격을 NSEED의 F 함수에 적용한다. 고정된 입력 차이 Δ 를

가지는 평문 쌍을 선택하고, 입력 차이와 같아지는 출력 차이 Δ'을 얻고, 세 번째 G 함수를 통과하여 나온 출력 차이가 $(\alpha, 0)$ 형이라고 하자. 그러면 세 개의 G 함수를 통과하는 구조인 NSEED의 F 함수에서 마지막 G 함수는 입력 차이에 의한 영향을 받지 않아 안전성에 영향을 주지 않게 된다. 따라서, NSEED의 F 함수의 최대 차분 그리고 선형 확률은 두 번째 G 함수를 통과한 것까지에 의해서 결정되므로 다음과 같이 나타낼 수 있다.

$$n_d(F) = n_l(F) = (n_l(G))^{i-1} = (2^{-30})^2 = 2^{-60}$$

여기서 i는 G함수의 수

이 과정은 그림 20과 같다.

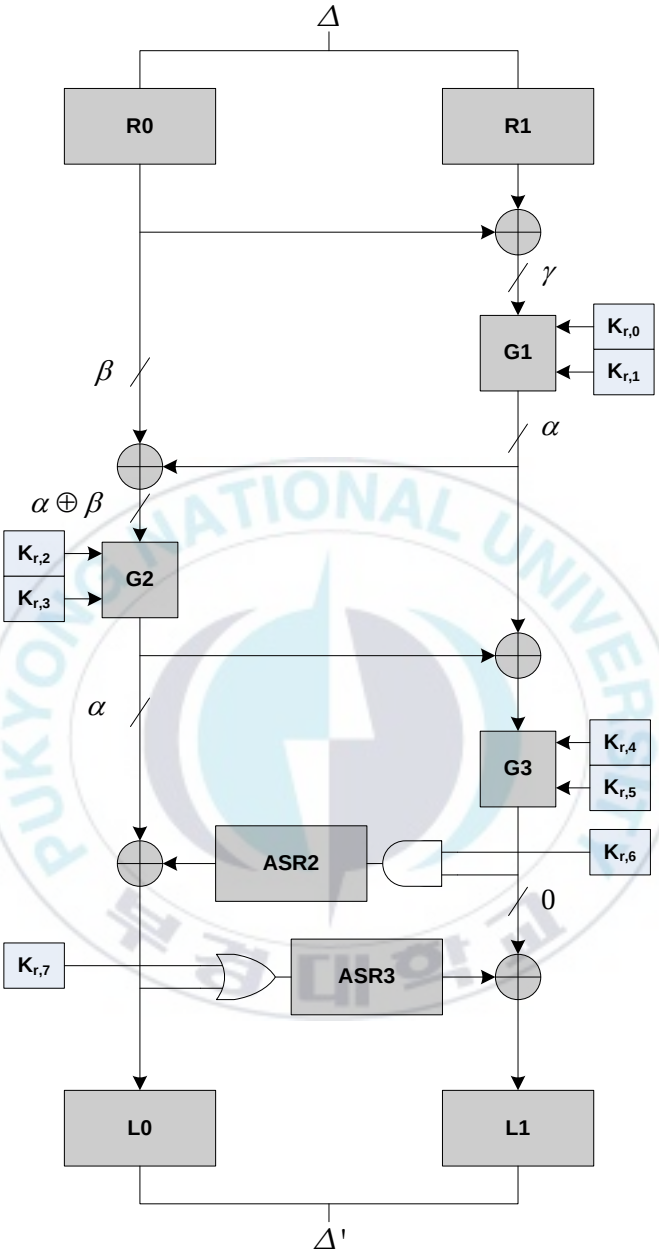


그림 20. F 함수에 대한 Difference Distribution 공격

이제, 이 공격을 4 라운드 NSEED 전체에 적용해 보겠다. 같은 방법으로 고정된 입력 차이 Δ를 가지는 평문 쌍을 선택하고, 입력 차이와 출력 차이가 같아지는 출력 차이 Δ'을 얻는다. 이 입력이 4 라운드 NSEED를 통과해서 네 번째 F 함수를 통과하여 나온 출력의 차이가 $(\alpha, 0)$ 형이라면, 이 F 함수는 입력 차이에 의한 영향을 받지 않는다. 따라서, 마지막 F 함수는 안전성에 영향을 주지 않는다. NSEED 전체에 대한 최대 차분 그리고 선형 확률은

$$n_d(NSEED) = n_l(NSEED) = (n_l(F))^{r-1} = (2^{-60})^3 = 2^{-180}$$

여기서 r은 라운드 수

으로 결정된다. 그리고 6 라운드 NSEED의 최대 차분 그리고 선형 확률은 다음과 같이 결정된다.

$$n_d(NSEED) = n_l(NSEED) = (n_l(F))^{r-1} = (2^{-60})^5 = 2^{-300}$$

여기서 r은 라운드 수

이 과정은 그림 21과 같다.

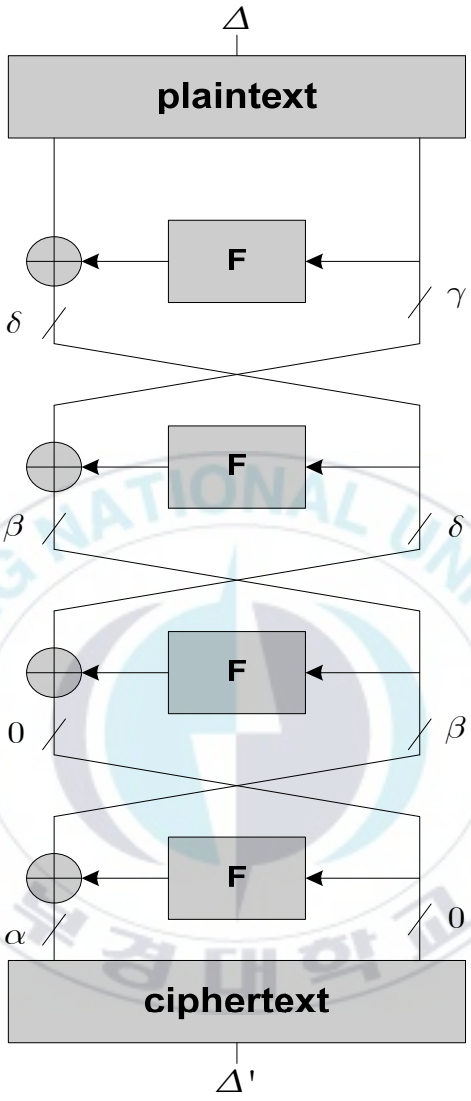


그림 21. NSEED에 대한 Difference Distribution 공격

이 결과들에 의해서 NSEED는 128 비트, 256 비트 키에서 충분히 안전한 것으로 판단된다.

이 결과들로부터 NSEED에서 최소 차분 활동성과 선형 활동성을 갖는 S 박스의 수를 계산하면 다음과 같다. NSEED의 최대 차분 확률과 선형 확률은 2^{-6} 을 만족한다. 그리고 차분 확산 계수 B_d 와 선형 확산 계수 B_l 은 $B_d = B_l = 5$ 를 만족한다. 이것으로부터 하나의 G 함수에서 차분 그리고 선형 활동성을 가지는 S 박스의 최소 개수는 5개이다. 그리고 1 라운드의 최소 차분 그리고 선형 활동성을 갖는 S 박스의 수는 G 함수를 세 번 통과하므로 15개이다. 따라서 128 비트 키를 가지고 전체 4 라운드를 수행하면 최소 차분 그리고 선형 활동성을 갖는 S 박스의 수는 60개가 된다. 그리고 256 비트의 키를 가지고 전체 6 라운드를 수행하면 90개이다.

제 7 장 NSEED의 성능 평가

이 장에서는 NSEED의 성능과 안전성을 NSEED와 같은 블록 암호 알고리즘인 SEED, ARIA, AES와 비교해서 평가한다.

7.1 소프트웨어적 성능 평가

블록 암호 알고리즘을 소프트웨어로 구현 하였을 때 알고리즘의 수행 시간은 S 박스를 통과하는데 소요되는 시간이 가장 큰 비중을 차지한다. 따라서 알고리즘을 수행하는 동안 통과하는 전체 S 박스의 수를 알아보면 알고리즘의 성능이 얼마나 좋은지 알 수 있다. NSEED 블록 암호 알고리즘에서 S 박스는 라운드 함수인 F 함수 안에 있는 내부함수인 G 함수에 위치하고 있다. NSEED의 F 함수는 세 개의 G 함수로 구성되어 있고, G 함수는 8개의 S 박스로 구성되어 있다. 따라서 한 라운드당 24개의 S 박스를 통과한다. 전체 4라운드를 수행하면 총 96개의 S 박스를 통과한다. NSEED와 비교했을 때, SEED는 한 라운드당 12개씩 16라운드를 수행하므로 총 192개를 통과한다. AES는 암호화 알고리즘과 복호화 알고리즘이 다르게 구성되어 있어 암호화할 때 한 라운드당 16개씩 10라운드를 수행하여 160개, 복호화할 때도 똑같이 160개를 통과해서 총 320개를 통과한다. ARIA는 한 라운드당 16개씩 12라운드를 수행하여 192개를 통과한다. 결과적으로 NSEED는 이들보다 훨씬 적은 S 박스를 통과한다. 따라서 소프트

웨어로 구성했을 때 다른 알고리즘들에 비해서 빨리 동작을 한다.

NSEED의 성능을 비교하기 위해 SEED, AES, ARIA와 함께 같은 파일을 암호화하여 알고리즘의 수행 시간을 비교 측정하였다. 수행 시간을 측정하는데 사용한 시스템은 다음과 같다.

- CPU: Intel Core2 DUO Conroe 6300
- 동작속도: 2.13 GHz
- 메모리: DDR2 PC6400 1 GB
- 운영체제: Windows XP

수행 시간은 알고리즘 내에서 암호화를 수행하는 부분만 측정하여 객관성을 가지도록 하였다. 알고리즘의 수행 시간을 측정한 결과는 표 14와 같다. 알고리즘 수행 시간은 측정하는 시스템의 사양 또는 사용 환경에 따라 달라질 수 있어 시간을 측정하여 NSEED의 수행 시간을 100으로 놓고 정규화하였다.

표 14. 각 알고리즘의 수행시간

키 크기	알고리즘			
	NSEED	SEED	ARIA	AES
128 비트	100	108	167	280
256 비트	100	-	185	306

알고리즘의 수행시간을 측정한 결과 NSEED가 AES보다는 3배, ARIA 보다는 1.9배 정도 빠른 시간에 수행을 끝냈다.

7.2 하드웨어적 평가

NSEED 블록 암호 알고리즘은 소프트웨어적 평가에서 살펴본 대로 다른 블록 암호 알고리즘에 비해 S 박스의 수를 적게 가지고 있다. 따라서 하드웨어로 구성했을 때 다른 알고리즘에 비해 간단하게 구성할 수가 있다.

블록 암호 알고리즘을 하드웨어로 구성했을 때 반도체의 면적을 가장 많이 차지하는 것이 S 박스이다. 왜냐하면 S 박스는 ROM으로 구성하기 때문이다. 그러므로 S 박스의 수가 적을수록 하드웨어가 간단해지고, 반도체 면적을 적게 차지한다. 본 논문에서 제안하는 NSEED는 96개의 S 박스가 사용된다. 이에 반해 SEED는 192개, ARIA는 192개, AES는 암호화 과

정과 복호화 과정이 분리되어 있어서 각 과정당 160개씩 총 320개의 S 박스가 사용된다. 따라서 하드웨어로 구성할 때 NSEED가 다른 알고리즘에 비해서 훨씬 간단하고, 반도체 면적도 적게 차지한다. 각 알고리즘을 하드웨어로 구성했을 때 사용되는 S 박스의 수는 표 15와 같다.

표 15. 하드웨어로 구성했을 때 사용되는 S 박스의 수

알고리즘	S 박스의 수
NSEED	96
SEED	192
ARIA	192
AES	320

하드웨어로 구성했을 때의 동작속도는 입력할 때부터 출력할 때까지 S 박스를 직렬로 통과하는 속도에 비례를 한다. 본 논문에서 제안하는 NSEED 블록 암호 알고리즘의 F 함수는 세 개의 G 함수가 직렬로 연결된 구조이고, 하나의 G 함수에 8개의 S 박스를 가지고 있다. 그래서 직렬로 통과하는 S 박스의 수가 24개이다. 이에 비해 다른 알고리즘인 SEED는 48개, AES는 10개, ARIA는 12개를 통과한다. 따라서 SEED 보다는 빠르게

동작을 하지만 AES와 ARIA에 비해서는 동작 속도가 늦을 것으로 판단된다.

NSEED 블록 암호 알고리즘은 하드웨어적인 성능을 요구하는 곳 보다는 하드웨어 자원이 제한적인 분야인 스마트 카드, RFID 등에서 유용하게 사용할 수 있을 것으로 예상된다.

7.3 안전성 비교 평가

NSEED와 SEED, ARIA 그리고 AES의 안전성 비교는 차분 분석법과 선형 분석법에 강한 특성을 지니고 있는가를 평가해서 알아보았다. 이 공격에 대한 안전성은 차분 확률과 선형 확률에 의존하므로 각 알고리즘의 차분 확률과 선형 확률을 비교하고, 이에 따르는 차분 활동성과 선형 활동성을 가지는 S 박스의 수를 비교하였다.

NSEED는 제 5 장에서 알아본 것과 같이 최대 차분 확률과 선형 확률이 2^{-6} 을 만족하고, 차분 확산 계수 B_d 와 선형 확산 계수 B_l 은 $B_d = B_l = 5$ 를 만족한다. 따라서 한 라운드에서 하나의 G 함수에 최소 5 개의 차분 활동성과 선형 활동성을 가지는 S 박스가 있다. F 함수 하나에는 15개의 S 박스가 차분 그리고 선형 활동성을 가진다. 따라서, 128 비트 키를 가지고 전체 4 라운드로 구성된 NSEED에서 차분 그리고 선형 활동성을 가지는 S 박스의 수는 $15 \times 4 = 60$ 개이고, 256 비트의 키를 가지고 전체 6 라운드로 구성한 NSEED에서는 $15 \times 6 = 90$ 개이다.

ARIA는 최대 차분 확률과 선형 확률이 2^{-6} 을 만족한다. 그리고 차분

특성의 Branch number β_d와 선형근사의 Branch number β_l은 $\beta_d = \beta_l = 8$을 만족하며, r 라운드로 구성되었을 때 차분 그리고 선형 활동성을 가지는 S 박스의 수는 다음 식으로 구해진다.

$$8 \bullet \lfloor r/2 \rfloor + 2 \bullet (r/2 - \lfloor r/2 \rfloor)$$

이 식에 의해서 ARIA는 1 라운드에 최소 4개의 S 박스가 활동성을 가지게 된다. ARIA의 한 라운드는 S 박스에 의해서 치환을 하는 치환 계층(Substitution Layer)과 선형변환행렬에 의해서 확산을 하는 확산 계층(Diffusion Layer)으로 구성되어 있다. ARIA의 두 라운드를 하나의 함수로 묶으면 '치환 계층 – 확산 계층 – 치환 계층 – 확산 계층'으로 구성된다. 여기서 두 번째 확산 계층은 선형부분이므로 안전성에 영향을 미치지 않으므로 제외할 수 있다. 따라서 ARIA의 두 라운드는 하나의 SDS (Substitution-Diffusion-Substitution) 함수로 나타낼 수 있다. 이것은 그림 22에 나타낸 것과 같다. 하나의 SDS 함수에는 32개의 S 박스가 있고, 어떤 S 박스가 활성화될지 알 수 없지만 위 식에 의해서 최소 8개의 S 박스가 차분 그리고 선형 활동성을 가지게 된다. 따라서 12 라운드로 구성했을 때는 6개의 SDS 함수로 구성되어 $8 \times 6 = 48$ 개, 16 라운드로 구성했을 때는 8개의 SDS 함수로 구성되어 $8 \times 8 = 64$ 개의 S 박스가 차분 그리고 선형 활동성을 가진다.

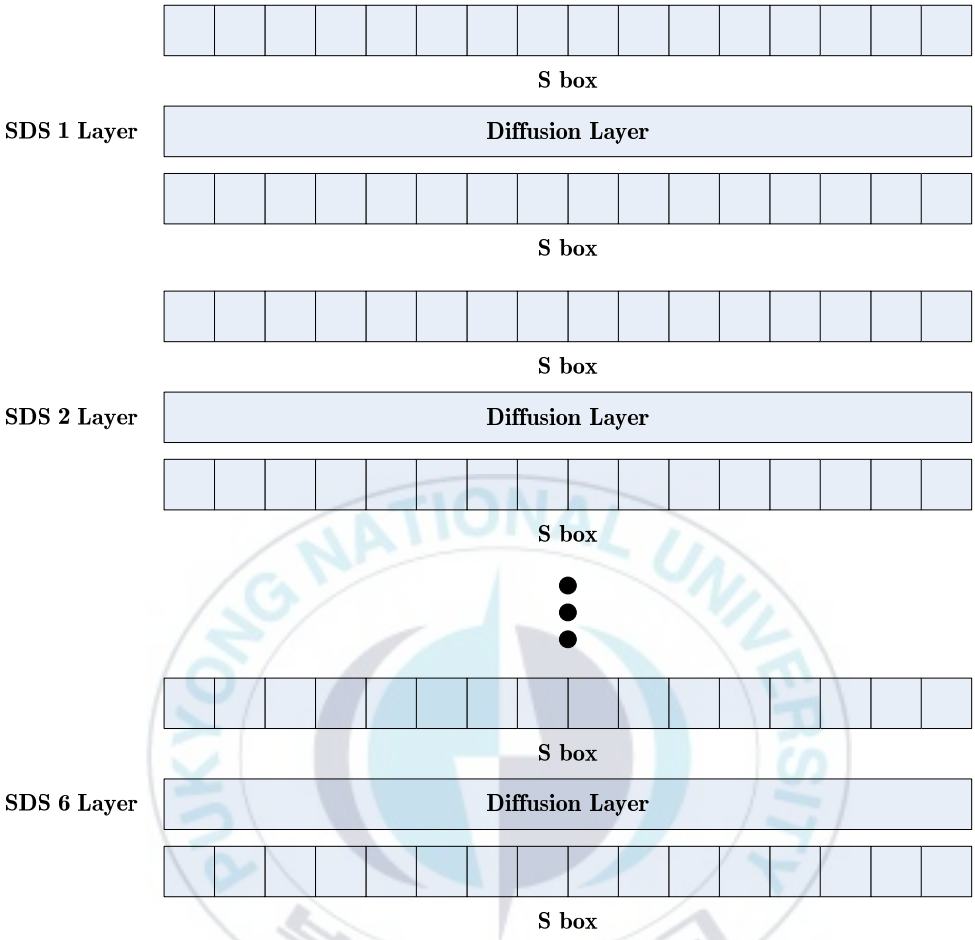


그림 22. SDS 함수로 구성한 ARIA

AES는 암호화할 때 SubBytes(), ShiftRows(), Mixcolumns(), AddRoundKey()의 단계를 수행한다. 이 단계를 수행할 때의 전달 특성에 의해서 차분 그리고 선형 활동성을 가지는 S 박스의 수를 구한다. 그런데 SubBytes()와 AddRoundKey()는 전달 특성에 영향을 미치지 않는다. 따라서 이 두 단계는 차분 그리고 선형 활동성을 가지는 S 박스의 수를 구하

는 과정에서 제외한다. 또한 MixColumns()는 Branch number 5를 만족한다.

각 라운드 별로 AES의 전달 특성을 살펴보도록 하자. 128 비트 블록 단위로 암호화를 하는 AES의 S 박스는 4×4 행렬로 나타낼 수 있다. 이 S 박스에서 가장 오른쪽 칼럼이 한 바이트에 의해서 활동성을 갖는다고 하자. 그리고 다른 세 칼럼은 고정되어 있다. 1 라운드를 수행하면 ShiftRows()와 MixColumns()에 의해 가장 오른쪽 칼럼의 전 바이트가 활동성을 가지게 된다. 그 결과 활동성을 가지는 S 박스는 1개가 된다. 이 결과를 가지고 2 라운드를 수행하여 ShiftRows()에 의해서 각 칼럼의 한 바이트만이 활동성을 가지게 된다. 이어서 MixColumns()에 의해 전 바이트가 활동성을 가지게 된다. 따라서 2 라운드 수행 후의 활동성을 가지는 S 박스의 수는 4개가 된다. 3 라운드에서는 ShiftRows()를 수행한 결과 역시 모든 바이트가 활동성을 가진다. 그 후, MixColumns()에 의해 각 칼럼의 한 바이트만이 활동성을 가진다. 그래서 활동성을 가지는 S 박스의 수가 16개이다. 4 라운드에서는 ShiftRows() 후에 가장 오른쪽 칼럼의 모든 바이트가 활동성을 가지고, MixColumns() 후에는 가장 오른쪽 칼럼의 한 바이트만이 활동성을 가져 처음 상태로 돌아오게 된다. 4 라운드에서 활동성을 가지는 S 박스의 수는 4개이다. 4 라운드 이후는 지금까지의 과정을 반복하게 된다. 4 라운드까지의 차분 그리고 선형 활동성을 가지는 S 박스의 수를 구하면 25개이다. 따라서 128 비트 키를 가지고 수행하는 경우에는 전체 10 라운드이므로 차분 그리고 선형 활동성을 가지는 S 박스의 수는 $(25\times2)+5=25$ 개이다. 256 비트 키로 수행하는 경우에는 전체 14 라운드이므로 $(25\times3)+5=25$ 개이다. AES의 전달 특성은 그림 23과 같다.

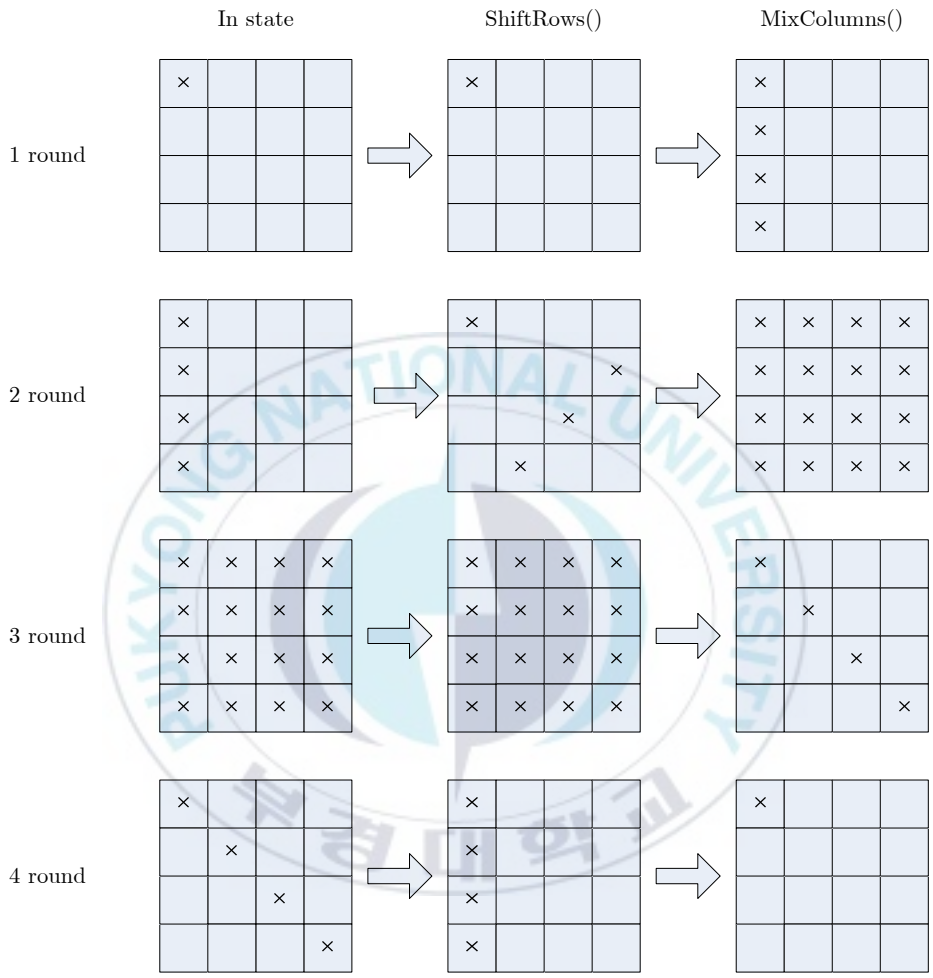


그림 23. AES의 전달 특성

각 알고리즘에 대한 활동 S 박스의 수를 표 16에 나타내었다.

표 16. 알고리즘별 활동 S 박스의 수

키 크기	NSEED	ARIA	AES
128 비트	60 개	48 개	55 개
256 비트	90 개	64 개	80 개

SEED는 F 함수에 들어오는 어떤 입력 차이에 대해서 3개의 G 함수 중 최소한 2개의 G 함수가 활성화된다. 즉, '0'이 아닌 입력 차이가 최소한 2개의 G 함수에 입력으로 들어가게 된다. 그리고 SEED G 함수의 diffusion order가 최소한 4 이므로 두 개의 G 함수를 연속해서 통과하면 최소 4개의 S 박스가 활성화 된다. 그런데 SEED의 F 함수는 세 개의 G 함수로 구성되어 있다. 따라서, 처음 두 개의 G 함수를 연속해서 통과하는 경우에는 최소 4개의 S 박스가 활성화 된다. 그 후, 세 번째 G 함수를 통과하고 다음 라운드의 첫 번째 G 함수를 통과하는 경우에는 덧셈 연산 후에 다시 XOR 연산을 한 후 G 함수를 통과하므로 앞의 경우와 같은 전달 특성을 갖지 않는다. 왜냐하면 덧셈연산이 비선형부분이기 때문이다. 그래서 차분 그리고 선형 활동성을 갖는 S 박스의 수가 최소 4개라고 할 수 없다. 그 결과 차분 그리고 선형 활동성을 가지는 S 박스의 수를 구할 수 없어 비교에서 SEED를 제외하였다. 만약 덧셈 연산을 XOR 연산으로 바

꾸면 같은 전달 특성을 갖게 되어 두 라운드 당 12개의 S 박스가 활성화 된다. 따라서 전체가 16 라운드 이므로 $12 \times 8 = 96$ 개의 S 박스가 차분 그리고 선형 활동성을 가진다. 그러나 덧셈 연산을 XOR 연산으로 바꾸게 되면 비선형 요소가 선형 요소로 바뀌게 되므로 안전성에 영향을 주게 된다. 즉, 안전성이 약하게 되는 단점이 발생된다.

이상에서 살펴 본 대로 NSEED 블록 암호 알고리즘은 ARIA, AES와 비교했을 때 적은 라운드를 수행하고도 동등 수준의 안전성을 가지고 있는 것으로 판단된다.

제 8 장 결 론

컴퓨터와 정보 통신이 발달할수록 더욱 많은 정보들이 이들을 통해서 처리되고 있다. 그 결과 처리되는 정보에 대한 보안이 중요한 문제로 대두되면서 정보를 보호하고, 불법적인 유출을 방지하기 위한 암호의 필요성이 증대되었다. 그러므로 선진 각국은 자국의 정보를 보호하기 위하여 독자적으로 암호 알고리즘을 개발하여 사용하고 있다. 우리나라도 한국정보보호진흥원이 주축이 되어 SEED를 개발하여 사용 중이다. 그런데 SEED는 확산이 덜 되는 단점이 있어 보완이 다소 필요하였다.

그러므로 본 논문에서는 첫째, SEED와 같은 구조를 가지는 블록 암호 알고리즘에서 사용할 수 있는 더욱 안전성이 높은 S 박스와 G 함수를 생성하는 방법을 제안하였다.

S 박스는 $GF(2^8)$ 상에서 비선형함수와 아핀 변환으로 구성하였다. 비선형함수는 블록 암호의 대표적 공격법인 차분 분석법과 선형 분석법에 강한 특성을 가지는 $nl(X) \Rightarrow X^{-1}$ over $GF(2^8)$ 변환을 채택하고, 원시 다항식은 '0'과 '1' 이외의 약한 입력을 가지지 않으면서 입력과 출력의 상관계수가 작은 것으로 선정하였다. 아핀 변환은 입력과 출력이 같은 고정점과 출력이 입력의 1의 보수가 되는, 역고정점인 약한 입력을 가지지 않으면서 입력과 출력의 상관계수가 가장 작게 되도록 구성하였다.

G 함수는 4개의 S 박스 출력을 $GF(2^8)$ 상에서 4×4 행렬식으로 확산 선형변환하여 구성하였다. G 함수는 MDS(Maximum Distance Separable) 코드를 생성하고, SAC(Strict Avalanche Criterion) 를 만족하여 확산 기능이

좋도록 하였다. 또한 고정점과 역고정점 및 출력이 입력의 2의 보수가 되는, 약한 입력을 가지지 않으며 하드웨어로 구현했을 때 회로가 간단해지도록 구성하였다.

둘째, 본 논문에서는 새로운 S 박스와 G 함수를 사용하여 SEED를 수정한 NSEED 블록 암호 알고리즘을 제안하였다. NSEED 블록 암호 알고리즘의 전체 구조는 피스탈 네트워크 구조로 되어 있고 128 비트 블록 단위로 암호화, 복호화를 수행한다. 암호화, 복호화할 때 사용하는 키는 128 비트, 256 비트의 키 중에서 선택해서 사용할 수 있고, 128 비트 키를 사용할 때는 4 라운드, 256 비트 키를 사용할 때는 6 라운드로 수행된다. F 함수는 세 개의 G 함수와 산술 시프트 레지스터로 구성하였고, 내부함수인 G 함수는 SPS(Substitution Permutation Substitution) 함수로 구성하여 안전성을 강화하였다. 그리고 키 생성 알고리즘은 암호 키로부터 산술 시프트 레지스터와 내부함수인 G 함수를 통하여 생성해서 사용하도록 하였다.

NSEED의 안전성은 블록 암호 알고리즘에 대한 공격법 가운데 강한 공격법인 차분 분석법과 선형 분석법에 대한 공격 확률이 4 라운드일 때 2^{-180}, 6 라운드일 때 2^{-300}으로 충분히 안전할 것으로 판단된다. 또한 차분 그리고 선형 활동성을 가지는 S 박스의 수가 4 라운드와 6 라운드일 때 각각 60개, 90개로 ARIA, AES와 비교 했을 때 동등한 안전성을 가진다. NSEED 알고리즘의 내부함수인 G 함수에서 사용된 S 박스의 총수는 96개로 다른 블록 암호 알고리즘인 ARIA, AES와 비교했을 때 훨씬 적은 수의 S 박스를 사용한다. 따라서 소프트웨어로 구현했을 때 ARIA, AES보다 더 빨리 동작한다. 그리고 하드웨어로 구현했을 때 반도체의 면적을 가장 많이 차지하는 것이 S 박스이다. NSEED는 S 박스의 수가 적으므로 반도체

의 면적을 적게 차지해 하드웨어로 구현할 때 간단하다는 장점이 있다. 그러나 하드웨어로 구현했을 때 수행 속도는 직렬로 통과하는 S 박스의 수에 의해서 결정된다. NSEED는 직렬로 통과하는 S 박스의 수가 ARIA와 AES에 비해 많아 하드웨어로 구현 시에는 속도가 늦어지는 단점이 있다.

본 논문에서 제안하는 NSEED 블록 암호 알고리즘은 소프트웨어로 사용할 때에 적합하며, 스마트 카드, RFID 등과 같은 하드웨어 자원을 적게 요구하는 분야에 적합하다. 또한 본 논문에서 새로 구성한 S 박스와 G 함수는 차분 분석법과 선형 분석법에 강하고, 약한 입력이 없으며, 하드웨어 구현이 용이하고, 확산 특성이 우수하므로 안전성이 높은 암호 방식의 구성 요소로도 활용할 수 있을 것이다.

참 고 문 헌

[1] 이만영, 원동호, 이민섭, 송주석, 임종인, 박춘식, 현대 암호학 및 응용, 생능출판사, 2002.

[2] 원동호, 현대 암호학, 도서출판그린, 2003.

[3] C. E. Shannon, “Communication theory of secrecy system”, BSTJ, vol. 28, pp. 656-715, 1949.

[4] 강주성, 김재헌, 박상우, 박춘식, 지성택, 하길찬, 한재우, 현대 암호학, 경문사, 2000.

[5] H. J. Becker, “Nonlinear feedback shift register sequences”, In Allen Gersho, Editor, Advances in Cryptology: A Report on CRYPTO ’81, pp. 121 – 123, U. C. Santa Barbara Dept. of Elec. and Computer Eng., 1982. Tech Report 82–04.

[6] S. W. Golomb, Shift Register Sequences, Aegean Park Press, Laguna Hills, 1982, Revised Edition.

[7] W. Diffie, M. E. Hellman, “New direction in Cryptography”, IEEE Transactions on Information Theory IT-26, 1976, no. 6, pp. 644 – 654.

[8] National Bureau of Standards, “Announcing the data encryption standard”, Technical Report FIPS Publication 46, National Bureau of Standards, January 1977.

[9] X. Lai, J. Massey, “Hash functions based on block ciphers”, In R. A. Rueppel, editor, Advances in Cryptology – Eurocrypt ’92, pp. 55-70, Berlin, Springer-Verlag, 1993.

[10] Shimizu, S. Miyaguchi, “Fast Data encipherment algorithm FEAL”, in Advances in Cryptology – EUROCRYPT ’87, vol. 304 of Lecture Note in Computer Science, Springer-Verlag, pp. 267 – 278, 1998.

[11] R. L. Rivest, “The RC4 Encryption Algorithm”, RSA Data Security Inc., Mar. 1992.

[12] R. L. Rivest, A. Shamir, L. M. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, Communications of the ACM 21, 1978, pp. 120 – 126.

[13] T. ElGamal, “A public key Cryptosystem and a Signature scheme based on discrete logarithms”, Advances in Cryptology – CRYPTO ’84, Lecture Note of Computer Science, vol. 196, Springer-Verlag, 1985, pp. 10 – 18.

[14] N. Koblitz, “Elliptic curves cryptosystems”, Math. of Comp. 48, 1987, no. 177, pp. 203 – 209.

[15] Joan Daeman, Vincent Rijman, AES proposal: Rijndael, 1999.

[16] NIST, Advanced Encryption Standard Development Effort, http://csrc.nist.gov/encryption/aes.

[17] 한국정보보호센터, 128 비트 블록 암호 알고리즘 (SEED) 개발 및 분석 보고서, DEC. 1998.

[18] Young-Ho Seo, Jong-Hyeon Kim and Dong-Wook Kim, “Hardware Implementation of 128-bit Symmetric Cipher SEED”, The Second IEEE Asia Pacific Conference on ASICs, pp. 183-186, Aug. 2000.

[19] 전신우, 정용진, “128 비트 SEED 암호 알고리즘의 고속처리를 위한 하드웨어 구현,” 통신정보보호학회지, Vol. 11, No. 1, pp. 13-23, Feb.

2001.

[20] 이명동, “SEED 암호 알고리즘의 FPGA 구현을 위한 RTL 수준 VHDL 설계”, 한남대학교 대학원 컴퓨터공학과 석사학위논문, 2001.

[21] 정찬호, “SEED에 대한 효과적인 Brute-Force 공격 알고리즘”, 한국항공대학교 컴퓨터공학과 석사학위논문, 2001.

[22] A. M. Youssef, Z. G. Chen, and S. E. Tavares, “Construction of Highly Nonlinear Injective S-boxes With Application to CAST-like Encryption Algorithms”, Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE ’97), 1997.

[23] Jennifer Seberry, Xian-Mo Zhang and Yuliang Zheng, “Systematic Generation of Cryptographically Robust S-boxes,” The preceedings of the First ACM Conference on Computer and Communications Security, pp. 172-182, Nov. 1993.

[24] Nyberg, K., “Perfect nonlinear S-boxes,” In Advances in Cryptology, EUROCRYPT'91, Vol. 547, Lecture Notes in Computer Science, Springer-Verlag, pp. 378-386, 1991.

[25] E. Biham, A. Shamir, “Differential cryptanalysis of DES-like cryptosystems”, Journal of Cryptology, Vol. 4, No. 1, pp. 3 – 72, 1991.

[26] M. Matsui, “The first experimental cryptanalysis of the Data Encryption Standard”, Advances in Cryptology, Proceedings of CRYPTO ‘94, Springer-Verlag, Berlin, pp. 1 – 11, 1994.

[27] K. Nyberg, “Differentially uniform mappings for Cryptography”, Advances in Cryptology, Proceedings of Eurocrypt ’93, LNCS 765, T.

Helleseth, Ed., Springer-Verlag, pp. 55 – 64, 1994.

[28] S. Mister, C. Adams, “Practical S-box Design”, Workshop record of the workshop on selected area in Cryptography (SAC ‘96), Queen’s University, pp. 61 – 76, Aug. 1996.

[29] T. Jakobsen, L. R. Knudsen, “ The interpolation attack on block cipher”, Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, pp. 28 – 40, 1997.

[30] S. Vaudenay, “On the need for multipermutations: Cryptanalysis of MD4 and SAFER”, Proceedings of Fast Software Encryption (2), LNCS 1008, Springer-Verlag, pp. 286 – 297, 1995.

[31] V. Rijmem, J. Daemen, B. Preneel, A. Bosselaers and E. De Win, “The Cipher SHARK”, Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-verlag, pp. 99 – 112, 1996.

[32] J. Daemen, L. Knudsen and V. Rijmen, “The Block cipher SQUARE”, Proceedings of Fast Software Encryption (4), LNCS, Springer-Verlag, 1997.

[33] A. Webster, S. Tavares, “On the Design of S-boxes”, Advances on Cryptology, CRYPTO ’85, pp. 523 – 534, 1985.

[34] ARIA 개발팀, 블록 암호 알고리즘 ARIA 알고리즘 명세서, May. 2004.

[35] H. M. Heys, "A Tutorial on Linear and Differential Cryptanalysis", Technical Report CORR 2001-17, Centre for Applied Cryptographic Research, Department of Combinatorics and Optimization, University of Waterloo, Mar. 2001. (Also appears in Cryptologia, vol. XXVI, no. 3, pp.

189-221, 2002.)

[36] S. Murphy, "The Cryptanalysis of FEAL-4 with 20 chosen plaintexts", Journal of Cryptology, Vol. 2, No. 3, pp. 145 – 154, 1990.

[37] E. Biham, A. Shamir, "Differential cryptanalysis of FEAL and N-Hash", In D. W. Davies, Ed., Advances in Cryptology – Eurocrypt '91, pp. 1 – 16, Berlin, Springer-Verlag, 1991.

[38] E. Biham, A. Shamir, "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer", In J. Feigenbaum, Ed., Proc. CRYPTO '91, pp. 156 – 171, LNCS No. 576, Springer-Verlag, 1992.

[39] X. Lai, J. Massey, and S. Murphy, "Markov Cipher and Differential Cryptanalysis", Advances in Cryptology – EUROCRYPT 91, Proceedings, Springer-Verlag, Springer-Verlag, pp. 17 – 38, 1991.

[40] K. Nyberg, L. R. Knudsen, "Provable Security against Differential Cryptanalysis", Advances in Cryptology – CRYPTO 92, Proceedings, Springer-Verlag, pp. 566 – 574, 1993.

[41] M. Matsui, A Yamagishi, "A New Method for Known Plaintext Attack of FEAL cipher", Advances in Cryptology – EUROCRYPT '92, Porceedings, Springer-Verlag, pp. 81 – 91, 1993.

[42] S. K. Langford, M. E. Hellman, "Cryptanalysis of DES", presented at 1994 RSA Data Security conference, Redwood Shores, CA, 12 – 14 Jan., 1994.

[43] B. S. Kaliski, M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations", Advances in Cryptology – CRYPTO '94, Proceedings,

Springer-Verlag, pp. 26 – 39, 1994.

[44] Joan Daemen, Lars Knudsen, and Vincent Rijmen, "The Block Cipher Square", In Eli Biham, Ed., Fast Software Encryption, 4th International Workshop, Vol. 1267 of Lecture Note in Computer Science, pp. 149-165, 1997.

[45] Baodian Wei, Dongsu Liu and Xinmei Wang, "Activity Attack on Rijndael", *aina*, pp. 803, 17th International Conference on Advanced Information Networking and Applications (AINA'03), 2003.

[46] Dong Hyeon Cheon, Seok Hie Hong, Sang Jin Lee, Sung Jae Lee, Kyung Hwan Park and Seon Hee Yoon, "Difference Distribution Attack on DONUT and Improved DONUT", In D. Won Ed., Information Security and Cryptology – ICISC 2000, LNCS 2015, pp. 37-48, 2001.

[47] Ju-Sung Kang, Choonsik Park, Sangjin Lee, and Jong-In Lim, "On the Optimal Diffusion Layers with Practical Security against Differential and Linear Cryptanalysis", Proceedings of ICISC '99, LNCS 1787, Springer-Verlag, pp. 32 – 52, 1999.

[48] A. M. Youssef, S. Mister, and S. E. Tavares, "On the Design of Linear Transformation for Substitution Permutation Encryption Networks", in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC '97), pp. 40 – 48, Aug. 11 – 12, 1997.

[49] H. M. Heys and S. E. Tavares, "The Design of Product Ciphers Resistant to Differential and Linear Cryptanalysis", Journal of Cryptology, Vol. 9, no. 1, pp. 1-19, 1996.

[50] J. Daemen, R. Govaerts, and J. Vandewalle, “Correlation Matrixes”, Fast Software Encryption, LNCS 1008, Spring-Verlag, pp. 275-285, 1994.

[51] F. J. MacWilliams, N.J.A. Sloane, The theory of error correcting codes, North-Holland Publishing Company, 1977.

[52] 박창수, 송홍복, 조경연, “SEED 형식 암호에서 공격에 강한 S 박스와 G 함수의 실험적 설계”, 한국해양정보통신학회 제8권 제1호, pp. 123-136, 2004.

[53] S. H. Hong, S. J. Lee, J. I. Lim, J. C. Sung, D. H. Cheon and I. H. Cho, “Provable security against Differential and Liner Cryptanalysis for the SPN Structure”, In B. Schneier Ed., Fast Software Encryption 2000, LNCS 1978, pp. 273-283, 2001.

감사의 글

그 동안 힘들었지만 나에게 새로움을 일깨워준 7년 6개월간의 대학원 생활을 마치면서 언제나 나를 도와주고, 옆에서 격려해 주신 모든 분들께 감사의 인사를 드립니다. 지난 대학원 생활 동안 항상 웃으시는 얼굴로 아낌없는 격려와 인자하신 모습으로 언제나 성의껏 지도해 주셔서 오늘 새로운 결실을 맺을 수 있도록 해주신 조경연 교수님께 진심으로 감사의 인사를 드립니다. 교수님께서 지난 기간 동안 보여주신 연구자세와 열정적으로 가르치시는 모습은 공학도로서의 앞으로의 제 인생에 큰 지침이 될 것입니다. 부족한 저의 논문을 심사하시면서 세심한 부분까지 지도해 주시고 깊은 관심을 보여주신 이경현 교수님, 정목동 교수님, 송하주 교수님 그리고 송홍복 교수님께도 깊은 감사를 드립니다. 대학원 생활을 하는 동안 저에게 많은 가르침을 주신 조우현 교수님, 서경룡 교수님, 신봉기 교수님 그리고 권오흠 교수님께도 인사를 드립니다.

지난 기간 동안 같은 연구실에서 동고동락하며 같이 지냈었고, 나에게 힘이 되어 주었던 최명룡, 전희성군과 장현지양에게도 이 자리를 빌어 고마움을 표시합니다. 그리고 같이 박사과정 공부를 하면서 항상 나의 버팀목이 되어 주었던 김성기씨와 김길호씨에게도 따뜻한 나의 인사를 전합니다. 박사과정 동기생인 박희숙선생님께도 항상 도와주신 것에 인사를 드립니다. 논문을 준비하면서 서로 도와주고 격려를 해주었던 석흥일군에게도 고마움을 표시합니다.

나를 지금까지 키워주신 어머님과 항상 뒤에서 도와주었던 동생 창현이게도 감사의 말을 전합니다. 저의 집에서 언제나 말없이 도와주신 장모님께도 고마움을 전합니다. 끝으로 지금까지 믿음과 사랑으로 나를 믿고 항상 지원해 준 이 세상에서 제일 사랑하는 아내 류희숙과 늘 바쁘다는 핑계로 같이 있어 주지 못하고 공부도 도와주지 못해서 미안한 또한 이 세상에서 제일 사랑하는 아들 준성이에게 언제나 사랑하고 고맙다는 인사를 전합니다.

2007년 7월

박 창 수 올림