韓國컴퓨터情報學會 論文誌
第16卷 第1號, 2011. 1.

2011-16-1-3-6

평문과 키의 종속관계를 이용한 혼합형 블록 암호시스템 설계에 관한 연구

이 선 근*

A Study on the design of mixed block crypto-system using subordinate relationship of plaintext and key

Seon-Keun Lee*

요 약

기존 블록 암호알고리즘은 평문과 키가 독립적이다. 또한 구조적인 특징을 이용하여 암/복호화를 수행한다. 이러한 특징은 해킹의 근거자료로 활용된다. 그러므로 제안된 혼합형 암호알고리즘은 외부환경은 기존과 동일하지만 내부적으로 평문과 키를 종속함수로 만들어 기존 블록암호알고리즘의 특징을 없애고자 하였다. 또한 인증처리에 대한 부하를 줄이기 위하여, 종속특징을 가진 인증 부가기능을 포함시켜 대칭형 암호알고리즘 활용을 증대시키고자 하였다. 제안된 혼합형 암호시스템을 칩 레벨로 구현하여 모의실험을 수행한 결과, 기존 시스템에 비하여 키 길이는 평문보다 작지만, 처리속도는 2배 높은 특징을 확인하였다.

▸ Keyword : 대칭형암호알고리즘, 혼합모드, 인증, 키관리, 반복연산

Abstract

Plaintext and key are independent in the existing block cipher. Also, encryption/decryption is performed by using structural features. Therefore, the external environment of suggested mixed cryptographic algorithm is identical with the existing ones, but internally, features of the existing block cipher were meant to be removed by making plaintext and key into dependent functions. Also, to decrease the loads on the authentication process, authentication add-on with dependent characteristic was included to increase the use of symmetric cryptographic algorithm. Through the simulation where the proposed cryptosystem was implemented in the chip level, we show that our system using the shorter key length than the length of the plaintext is two times faster than the existing systems.

▸ Keyword : Symmetric block Cryptographic algorithm, Mixed Mode, Authentication, Key management, iteration

• 제1저자 : 이선근
• 투고일 : 2010. 05. 28, 심사일 : 2010. 08. 22, 게재확정일 : 2010. 11. 03.
* 전북대학교 IT응용시스템공학과(Dept. of IT Applied System Engineering, Chonbuk national University)

Ⅰ. 서 론

정보보호분야는 정보보호 알고리즘 개발, 정보보호시스템 개발 등과 같이 전문성을 띄며 분야별로 발전되는 것이 현 추세이다. 그러므로 여러 종류의 플랫폼에 대한 확장성을 고려하여 정보보호 시스템들은 여러 가지 모양으로 변화한다.

대칭형 방식은 구조적인 메카니즘을 통하여 암호화를 수행하며 비대칭형 방식은 수학적인 해를 얻기가 어려운 문제를 이용하여 암호화를 수행한다. 그러므로 대칭형 암호방식은 암호화 수행속도가 비대칭형 방식에 비하여 매우 빠른 장점을 가진다. 그러나 이러한 구조적 암호화 과정은 구조자체가 공개되어 있어 DC(Differential Cryptanalysis) [1] 및 LC(Linear Cryptanalysis)[2] 등의 다양한 방법들에 의하여 해킹 및 크래킹의 기본 자료로 활용된다. 이러한 문제점들을 없애기 위하여 iteration 및 키 길이를 증대시키는 방식으로 연구가 진행되고 있다. 또한 블록방식은 안전이 확보되지 않은 채널을 통하여 데이터를 송수신하기 때문에 인증기능을 부가하여 사용된다. 그러므로 인증기능인 MAC, MDC에 대한 부담이 적지 않다.

그러므로 본 연구는 가장 보편적인 암호화 방식인 블록암호 시스템에서 문제점으로 제시된 인증기능과 키 관리, 그리고 DC/LC로부터 보다 안전하며 네트워크 통신에 적합하도록 하는 혼합형 암호알고리즘을 설계함으로써 다양한 네트워크 응용분야에 적응할 수 있는 해결책을 제시하고자 한다.

Ⅱ. 제안된 혼합형 암호알고리즘

제안된 혼합형 암호시스템은 데이터 재배열(permutation), 치환(substitution), 데이터 암호블록, 키 스케쥴(key schedule)로 구성되어 있다. 데이터 암호블록은 블록 암호시스템으로 구성되어 있으며 키 스케쥴러는 스트림 암호시스템으로 구성되어 있다.

데이터 암호화 과정은 128 비트 평문블록을 64 비트씩 2개의 블록으로 분할하고 확장재배열(expansion)을 거친 후 80 비트의 크기를 가지는 혼합형 키(hybrid key : HK)를 사용하여 암호화한다. 기존 블록 암호시스템(xDES)[3][4]인 경우 내부적으로 16라운드(16-round)의 암호화 과정을 거치고, 복호시에도 암호화에 사용된 동일한 키를 역순으로 사용하여 16라운드의 복호화 과정을 수행한다. 그러나 제안된 혼합형 암호화 시스템은 단일 라운드만을 사용하여 기존의 16 라운드에 해당하는 비도를 유지하기 위하여 입력문과 키를 혼합한 혼합형 키와 비선형 부분인 F 암호함수부분을 보다 더 복잡한 구조를 가지도록 하였다.

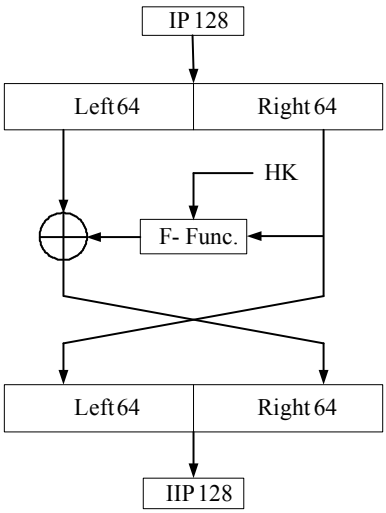

그림 1. 제안된 혼합형 Feistel 구조
Fig 1. Proposed mixed Feistel structure

제안된 혼합형 암호시스템은 그림 1과 같이 기존 Feistel 구조와 구조적으로 동일하다. 식 (1)과 같이 128 비트에 대하여 초기치환(initial permutation : IP)을 수행하고 IP 수행결과를 F 암호함수와 혼합 키(HK)를 이용하여 내부연산을 수행한 후 교번하여 출력을 내보낸다.

$$L_1 = R_0 \qquad \cdots\cdots (1)$$
$$R_1 = L_0 \oplus f(R_0, HK)$$

여기에서 L_1은 다음 stage의 왼쪽 데이터블록이며 R_0는 첫 stage의 오른쪽 데이터블록이다.

식 (1)은 기존 Feistel 구조와 동일하지만 iteration에 관한 i 파라미터가 없고, 대신 이전과 이후에 관한 방정식만 존재한다. 또한 특히, iteration에 대한 함수를 1회로 한정짓는 반면, 종속키를 이용하여 암호화를 수행한다. 여기에서 입력 128 비트는 IP를 거친 후 L/R로 좌우 64비트씩 분리된다. 각각 L/R의 좌우로 분리된 64 비트는 그림 1과 같이 오른쪽 64 비트는 왼쪽으로 이동하며 왼쪽 64 비트는 혼합형 키와 F 암호함수의 연산과정 후 XOR 연산을 통하여 오른쪽으로 이동하게 된다. 이러한 연산을 수행한 후 마지막으로 역 초기치환(inverse initial permutation : IIP)을 수행한 후 암호화된 데이터를 출력하게 된다.

F 암호함수는 일반적으로 사용되는 키 스케쥴러에 의해 발생된 값을 사용하는 것이 아니고 종속 키 값인 혼합키를 사용하여 연산을 수행함으로써 iteration은 감소하면서 비도는 유지한다.

암호함수 F의 비도 결정은 확장재배열과 재배열, 치환에 결정적으로 영향을 받으며 이러한 확장, 재배열, 치환은 기존 알고리즘에서 사용되는 미리 설정된 표에 의하여 결정된다.

혼합형 암호방식에서 다양한 통계적 분석기법들에 대한 저항성을 갖기 위하여 비선형 연산 못지않게 비선형 연산결과를 가능한 많은 다른 데이터 비트들과 섞어주는 효율적인 선형변환을 사용하였다. 이와 같이 선형과 비선형 연산을 섞어 사용하는 이유는 암호 해석기법인 DC 및 LC 암호분석에서의 성공확률은 상당부분 얼마나 선형변환을 사용하느냐에 의존하기 때문이다[5][6]. 그러므로 제안된 혼합형 암호시스템에서는 가장 널리 사용되는 선형변환들 중 bit permutation 개념과 본 논문에서 제안한 matrix 개념을 도입하였다.

그림 2는 혼합형 키를 사용하여 비선형 함수 및 선형함수를 생성하는 F 함수 발생 블록으로써 기능블록은 다음과 같다. E 함수는 확장재배열기능으로써 키와 데이터간의 비트수를 맞추어주며 비도를 증가시키기 위하여 사용되는 선형 암호블록이다.

E 함수에서는 16 비트의 데이터가 2번씩 반복되어 연산에 참여한다. 그러므로 64 비트들에 대해서 16비트가 첨가되므로 전체 80 비트의 데이터 값들이 산출되게 된다. C 함수는 압축(compression)을 수행하는 블록으로써, 그림 2에서 혼합형 키 80 비트와 80 비트로 확장된 입력데이터들끼리 bit-by-bit XOR 연산을 수행한 후 출력을 S 함수로 보내주는 기능을 수행한다. S 함수는 6 비트씩 모두 12개의 하부 S 함수로 구성되어 있기 때문에 6X12=72 비트의 값으로 재조정을 수행해야 한다. 80 비트 입력을 받아서 72 비트 크기의 값을 산출하기 위하여 미리 설정된 표에 의하여 압축과정을 수행하게 된다. 9, 19, 29, 39, 49, 59, 69, 79번째에 해당하는 데이터들은 소거시켜 72 비트의 데이터값을 얻는다. 비선형 S 함수는 그림 2에서와 같이 암호함수 F 내부에서 비선형 함수를 발생시키는 기능을 수행하게 된다. S 함수의 입력은 72 비트이고 출력은 48 비트의 크기를 가진다. S 함수 내부에는 입력 6 비트, 출력 4비트를 산출하는 하부 S 함수가 12개 존재하며 12개 하부 S 함수의 조합이 전체 S 함수를 구성하게 된다.

식 (2)는 비선형 S 함수를 표현한 것으로서 하부 S 함수가 12개로 구성되어 있다. 본 논문에서 사용된 S 함수는 기존의 S 함수와 기존 S 함수의 짝수에 해당하는 S 함수를 사용하게 되는데 S 함수의 내용을 변화시키지 않은 것은 기존 S 함수가 가지고 있는 비선형 특성을 유지하면서 내부구조를 더욱 복잡하게 하기 위함이다.

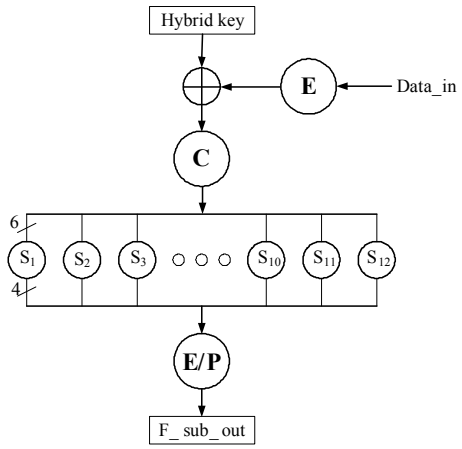

그림 2. F 암호함수
Fig 2. F crypto-function

$$S = \sum_{i=1}^{12} s_i \quad \cdots\cdots (2)$$

본 논문에서 제안한 혼합형 암호화 시스템은 iteration이 없고 단지 1회의 라운드만을 수행하기 때문에 기존 블록 암호시스템 S 함수의 설계방법과는 약간 다른 점이 있다. 즉 S 함수에 대한 재배열관계를 무시하여도 S 함수에 대한 비선형특성이 변화되지 않는다. 또한 기존 S 함수는 라운드에 대한 iteration을 수행할수록 비선형성이 증가되지만 혼합형 암호시스템에서는 iteration이 없이 단지 1회 연산만을 이용하여 암호화를 수행하기 때문에 하부 S 함수의 개수를 4개 더 증가시켜 설계하였다. E/P 함수는 확장과 치환(expansion and permutation) 기능을 수행하는 블록이다. S 함수의 출력 48 비트에 대하여 F 함수 64 비트로 변환하기 위하여 확장표를 사용한다.

그림 3은 스트림 암호방식을 적용한 혼합형 키 스케쥴러(Hybrid Key Scheduler : HKS) 전체블록이다. HKS의 구성은 Buffer 128, Block data format 128, Mixer, LFSR-128, Weight generator, Time generator, Key sequence, Replacement(replator), Merging & Expansion(Mexp)이다.

키 스케쥴러의 입력은 키 데이터가 별도로 존재하는 것이 아니고 암호화하고자 하는 데이터가 키 스케쥴러의 입력이 된다. 그러므로 그림 4와 같이 Mixer 입력은 128 비트의 입력데이터와 LFSR로부터 생성된 키 수열이 된다. 이 두 가지의 데이터는 16 비트씩 8개의 블록으로 분리되며 분리된 16 비트 8

블록은 각각 XOR 연산을 수행하게 된다. 동시에 16 비트의 데이터 8 블록들이 XOR 연산을 수행한 후 Matrix permutation을 수행한 후 Merging & Expansion 블록으로 유입된다.

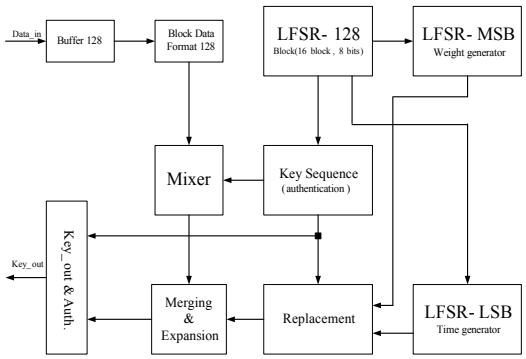

그림 3. 종속특성을 갖는 제안된 HKS
Fig 3. Proposed HKS with dependent characteristics

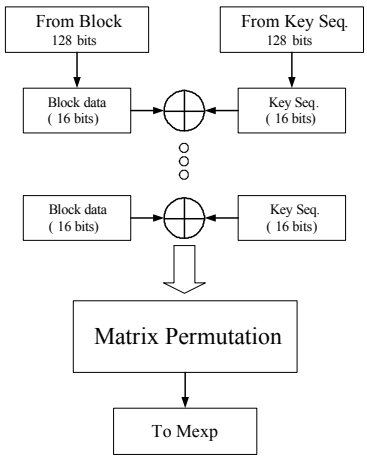

그림 4. Mixer 블록
Fig 4. Mixer block

Matrix permutation은 16 비트씩 8 블록들이 독립적으로 XOR 연산을 수행하게 되며 수행된 결과도 역시 16 비트 8 블록으로 구성된다. 이때 행 데이터들에 대한 연산결과는 열 데이터의 형태로 출력되어진다. 입력데이터의 집합을 I, 키 수열의 집합을 K라고 하였을 경우 I, K에 대한 표현은 식 (3)과 같다.

$$I= \{i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7\} \quad \cdots\cdots (3)$$
$$K= \{k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7\}$$

식 (3)에서 i 와 k는 각각 16 비트씩으로 구성된 스트림 데이터들이다. XOR 연산을 수행한 결과를 M이라 하였을 경우, XOR 연산을 수행한 입력 데이터들은 식 (4)와 같이 표현된다.

$$M= I \bigoplus K \quad \cdots\cdots (4)$$

식 (4)에서 M 역시 16 비트 8 블록이므로 식 (5)와 같은 수열을 가진다.

$$M= \{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7\} \quad \cdots\cdots (5)$$

그러므로 식 (4)에 식 (3)을 대입하고 식 (4)의 연산 수행을 행렬로 표현하면 식 (6)이 된다.

$$M= \begin{bmatrix} ik_{00} & ik_{01} & ik_{02} & ik_{03} & ik_{04} & ik_{05} & ik_{06} & ik_{07} \\ ik_{10} & ik_{11} & ik_{12} & ik_{13} & ik_{14} & ik_{15} & ik_{16} & ik_{17} \\ ik_{20} & ik_{21} & ik_{22} & ik_{23} & ik_{24} & ik_{25} & ik_{26} & ik_{27} \\ ik_{30} & ik_{31} & ik_{32} & ik_{33} & ik_{34} & ik_{35} & ik_{36} & ik_{37} \\ ik_{40} & ik_{41} & ik_{42} & ik_{43} & ik_{44} & ik_{45} & ik_{46} & ik_{47} \\ ik_{50} & ik_{51} & ik_{52} & ik_{53} & ik_{54} & ik_{55} & ik_{56} & ik_{57} \\ ik_{60} & ik_{61} & ik_{62} & ik_{63} & ik_{64} & ik_{65} & ik_{66} & ik_{67} \\ ik_{70} & ik_{71} & ik_{72} & ik_{73} & ik_{74} & ik_{75} & ik_{76} & ik_{77} \end{bmatrix} \quad \cdots\cdots (6)$$

여기에서 ik_{32}는 $i_3 \bigoplus k_2$ 의 연산결과임을 나타낸다. 식 (6)은 Mixer블록의 Matrix permutation의 입력으로 사용된다.

$$M^T = MP = \begin{bmatrix} ik_{00} & ik_{10} & ik_{20} & ik_{30} & ik_{40} & ik_{50} & ik_{60} & ik_{70} \\ ik_{01} & ik_{11} & ik_{21} & ik_{31} & ik_{41} & ik_{51} & ik_{61} & ik_{71} \\ ik_{02} & ik_{12} & ik_{22} & ik_{32} & ik_{42} & ik_{52} & ik_{62} & ik_{72} \\ ik_{03} & ik_{13} & ik_{23} & ik_{33} & ik_{43} & ik_{53} & ik_{63} & ik_{73} \\ ik_{04} & ik_{14} & ik_{24} & ik_{34} & ik_{44} & ik_{54} & ik_{64} & ik_{74} \\ ik_{05} & ik_{15} & ik_{25} & ik_{35} & ik_{45} & ik_{55} & ik_{65} & ik_{75} \\ ik_{06} & ik_{16} & ik_{26} & ik_{36} & ik_{46} & ik_{56} & ik_{66} & ik_{76} \\ ik_{07} & ik_{17} & ik_{27} & ik_{37} & ik_{47} & ik_{57} & ik_{67} & ik_{77} \end{bmatrix} \quad \cdots\cdots (7)$$

그러므로 식 (6)에 대한 매트릭스 치환은 식 (7)과 같은 결과를 가진다.

이상과 같은 연산을 통하여 생성된 64 비트는 Merging & Expansion의 입력으로 사용된다.

8단 LFSR의 초기상태가 모두 '0'일 경우 LFSR 탭 범위는 $1 \leq i \leq 8$이므로 지연소자의 초기 스트림값 s_i는 모두 영의 값을 가지게 된다. 그러므로 LFSR 키 스트림 생성수

열은 식 (8)과 같다.

$$\begin{bmatrix} z_8+k_8 \\ z_9+k_9 \\ z_{10}+k_{10} \\ z_{11}+k_{11} \end{bmatrix} = \begin{bmatrix} k_7 & k_6 & k_5 & k_4 & k_3 & k_2 & k_1 & k_0 \\ k_8 & k_7 & k_6 & k_5 & k_4 & k_3 & k_2 & k_1 \\ k_9 & k_8 & k_7 & k_6 & k_5 & k_4 & k_3 & k_2 \\ k_{10} & k_9 & k_8 & k_7 & k_6 & k_5 & k_4 & k_3 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} \quad \cdots\cdots (8)$$

$$\begin{bmatrix} z_{12}+k_{12} \\ z_{13}+k_{13} \\ z_{14}+k_{14} \\ z_{15}+k_{15} \end{bmatrix} = \begin{bmatrix} k_{11} & k_{10} & k_9 & k_8 & k_7 & k_6 & k_5 & k_4 \\ k_{12} & k_{11} & k_{10} & k_9 & k_8 & k_7 & k_6 & k_5 \\ k_{13} & k_{12} & k_{11} & k_{10} & k_9 & k_8 & k_7 & k_6 \\ k_{14} & k_{13} & k_{12} & k_{11} & k_{10} & k_9 & k_8 & k_7 \end{bmatrix} \begin{bmatrix} g_5 \\ g_6 \\ g_7 \\ g_8 \end{bmatrix}$$

$$G(x) = x^7 + x^5 + 1 \quad \cdots\cdots (9)$$

제안된 키 스케줄러의 생성다항식은 식 (9)와 같으며 식 (9)의 생성다항식에서 탭 계수$g_1 = g_6 = g_8 = 1$ 을 만족하며 초기값은 모두 0을 제외한 값을 사용하였다. 식 (8)과 같은 키 생성수열을 식 (9)의 생성다항식에 이용하면 키 출력수열 Z는 식 (10)을 얻는다.

식 (10)은 식 (9)의 생성다항식에 의한 키 수열을 나타내는 것으로써 본 논문에서는 식 (9)의 MSB와 LSB만을 이용하도록 하였다. 일반적으로 의사난수발생기의 주기는 $2^m - 1$로써 표현되어지며 이때 m값이 클수록, 주기가 길수록 선형특성에 안전하다. 그러므로 m값이 큰 의사난수발생기를 설계하게 되는데 이는 안전성에는 좋은 특성을 가지지만 주기에 비례하여 처리시간이 길어지며, 특히 동기시스템인 경우 오류발생시마다 동기 획득에 걸리는 시간이 길어지는 단점을 가진다.

그러므로 제안된 방식은 스트림 암호방식의 특징 중 하나인 난수발생은 한 주기 안에 절대적으로 존재한다는 것을 이용한다. 즉 한 주기 안에 존재하는 모든 의사난수를 이용하는 것이 아니고 초기 난수를 암호화에 이용하여 주기 시스템에 대한 주기시간의 단축을 달성한다. 즉 LFSR의 초기값에 대하여 m만큼의 시간이 지날 때 출력되는 의사난수출력 Z_m을 LFSR의 출력으로 사용한다.

그림 5는 8단 LFSR을 16개 동일하게 사용하여 LFSR 128 블록을 구성한 것이다.

LFSR Generator 0에서부터 LFSR Generator 15까지는 동일한 생성다항식을 사용하는 LFSR로써 모두 16개의 LFSR로써 구성되어 있다. 초기입력이 인가된 후 8T(period)까지 출력되는 출력수열은 모두 128 비트의 크기를 가진다. 이때 각각의 8 비트들에 대하여 MSB와 LSB로 분류하여 Weight generator, Time generator의 입력으로 사용한다. 또한 MSB 그룹(group)과 LSB 그룹을 모두 합한 128 비트는 Key sequence로 입력된다.

$$\begin{aligned} z_0 + k_0 &= g_1 s_1 + g_6 s_6 + g_8 s_8 \\ z_1 + k_1 &= g_1 k_0 + g_6 s_5 + g_8 s_7 \\ z_2 + k_2 &= g_1 k_1 + g_6 s_4 + g_8 s_6 \\ z_3 + k_3 &= g_1 k_2 + g_6 s_3 + g_8 s_5 \\ z_4 + z_4 &= g_1 k_3 + g_6 s_2 + g_8 s_4 \\ z_5 + k_5 &= g_1 k_4 + g_6 s_1 + g_8 s_3 \\ z_6 + k_6 &= g_1 k_5 + g_6 k_0 + g_8 s_2 \\ z_7 + k_7 &= g_1 k_6 + g_6 k_1 + g_8 s_1 \end{aligned}$$

$$\begin{aligned} z_8 + k_8 &= g_1 k_7 + g_6 k_2 + g_8 k_0 \quad \cdots\cdots (10) \\ z_9 + k_9 &= g_1 k_8 + g_6 k_3 + g_8 k_1 \\ z_{10} + k_{10} &= g_1 k_9 + g_6 k_4 + g_8 k_2 \\ z_{11} + k_{11} &= g_1 k_{10} + g_6 k_5 + g_8 k_3 \\ z_{12} + z_{12} &= g_1 k_{11} + g_6 k_6 + g_8 k_4 \\ z_{13} + k_{13} &= g_1 k_{12} + g_6 k_7 + g_8 k_5 \\ z_{14} + k_{14} &= g_1 k_{13} + g_6 k_8 + g_8 k_6 \\ z_{15} + k_{15} &= g_1 k_{14} + g_6 k_9 + g_8 k_7 \end{aligned}$$

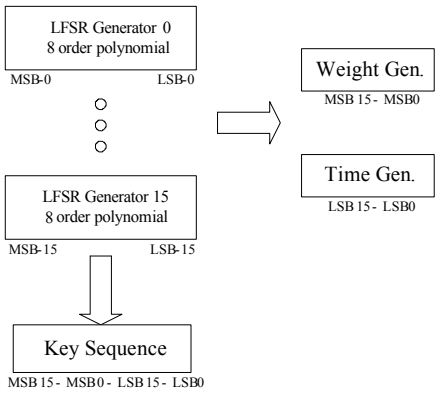

그림 5. LFSR 128 블록
Fig 5. LFSR 128 block

그림 6은 매트릭스 연산을 수행하는 블록으로써 유입되는 128 비트의 데이터들에 대하여 행렬로 표현하면 식 (11)과 같이 표현된다.

식 (11)은 LFSR 128에 대한 데이터들에 대한 행렬 표현식이다. 식 (11)의 행렬에서 대각행렬에 대한 데이터들을 별도의 데이터로 취급하기 위하여 별도의 데이터 포맷팅을 수행한다. 이때 생성되는 별도의 데이터는 rect와 orth이다. 각각의 데이터들은 16 비트의 크기를 가지고 있으며 rect와 orth에 대한 데이터값들의 복원 및 생성을 용이하도록 하기 위하여 128 비트의 유입데이터들에 대하여 정방행렬의 대각만을 데이터로 추출하여 사용하였다.

정방행렬의 대각에 대한 값을 추출하기 위하여 식 (11)에서 64 비트씩 두 블록으로 분리하여 rect와 orth값을 추출하였다.

식 (12)는 분리된 두 블록을 표시한다.

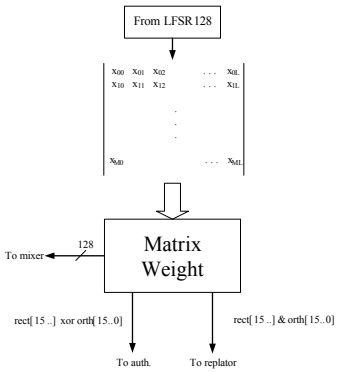

그림 6. 키 수열 블록
Fig 6. Key sequence block

$$\begin{vmatrix} x_{00} & x_{01} & x_{02} & \cdots & x_{07} \\ x_{10} & x_{11} & x_{12} & \cdots & x_{17} \\ & & \vdots & & \\ x_{150} & x_{151} & x_{152} & \cdots & x_{157} \end{vmatrix} \quad \cdots\cdots (11)$$

$$\begin{vmatrix} x_{0} & x_{1} & x_{2} & \cdots & x_{7} \\ x_{8} & x_{9} & x_{10} & \cdots & x_{15} \\ & & \vdots & & \\ x_{56} & x_{57} & x_{58} & \cdots & x_{63} \\ x_{64} & x_{65} & x_{66} & \cdots & x_{71} \\ x_{72} & x_{73} & x_{74} & \cdots & x_{79} \\ & & \vdots & & \\ x_{120} & x_{121} & x_{122} & \cdots & x_{127} \end{vmatrix} \quad \cdots\cdots 12)$$

여기에서 두 블록의 범위는 $ax = (x_0, ..., x_{63})$,
$bx = (x_{64}, ..., x_{127})$ 이다.

$ax + bx$는 mixer의 입력으로 사용되며, rect와 orth는 인증용으로 사용하기 위하여 XOR 연산을 수행하게 된다. 또한 비밀키로 사용하기 위하여 rect와 orth는 단순합인 &을 수행하여 replacement 기능을 수행하는 replator의 입력으로 유입된다.

그림 7의 Replator는 LFSR로부터 생성된 의사난수에 대하여 8T 동안에 새롭게 데이터들에 대한 재배치를 위한 포맷팅 기능을 수행한다. Replator는 Key sequence와 Weight generator 그리고 Time generator의 출력을 받아서 데이터를 재포맷(re-formating)하는 블록으로써 Key sequence에서 출력되는 key sequence_To replator 32 비트와 Time generator 16 비트 그리고 Weight generator 16 비트의 출력을 Time generator의 출력인 time stamp를 이용하여 64 비트 크기를 기지는 암호용 키 데이터를 생성시키기 위한 16 by 8 weight and time sequence를 생성한다.

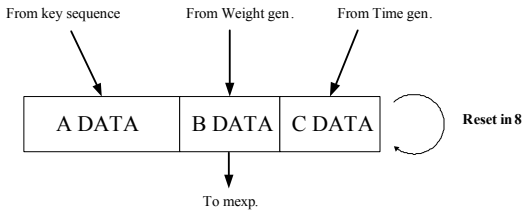

그림 7. Replator 블록
Fig 7. Replator block

Key sequence는 그림 8과 같이 interleaving된 mixer 블록의 출력과 replator에서 생성된 weight-time stamp를 이용하여 64 비트의 크기를 가지는 새로운 블록 데이터를 생성한다. mixer(M)의 MSB와 replator(R)의 MSB를 XOR 시키고 mixer의 LSB와 replator의 LSB를 XNOR 시킨후 MSB와 LSB를 합하여 출력한다. 이때 출력되는 데이터는 Auth. 데이터를 포함하여 80 비트의 크기를 가지게 된다. Auth. 데이터는 key sequence 블록에서 생성된 16 비트의 데이터로써 송신자의 인증용으로 사용한다.

출력되는 데이터의 프레임 구조는 그림 9와 같다. 그림 9에서 키 스케쥴러에서 출력되는 최종 출력값은 80 비트의 크기를 가지며 이중 64 비트는 데이터 암호화에 사용되며, 나머지 16 비트는 송신자에 대한 인증용으로 사용된다.

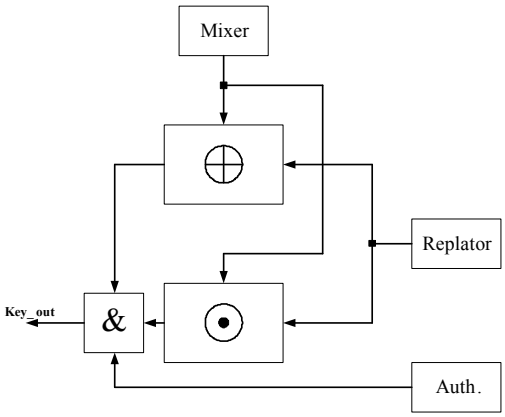

그림 8. Merging & Expansion 블록
Fig 8. Merging & Expansion block

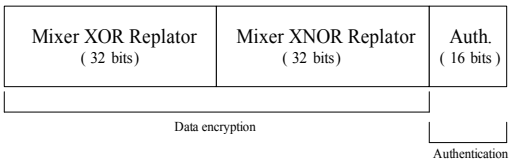

그림 9. Merging & Expansion 프레임 포맷
Fig 9. Merging & Expansion frame format

mixer의 데이터 $M=\{m_0,m_1,...,m_{62},m_{63}\}$, replator의 데이터 $R=\{r_0,r_1,...,r_{62},r_{63}\}$ 에 대하여 정리하면 식 (13)과 같다.

$$key-out = M(MSB) \bigoplus R(MSB) \text{ and } M(LSB) \odot R(LSB) \quad \cdots\cdots (13)$$

그림 8의 전체 출력 Key_out은 식 (13)의 결과식과 Auth. 데이터를 합한 것으로써 식 (14)와 같다.

$$key-out \text{ and } Auth. = M(MSB) \oplus R(MSB) \text{ and } M(LSB) \odot R(LSB) \text{ and } Auth. \quad \cdots\cdots (14)$$

여기에서 and는 &와 같이 단순합을 의미하며 ⊕는 비트들끼리의 XOR, ⊙는 비트들끼리의 XNOR 연산을 의미한다. 식 (14)를 행렬식으로 표현하면 다음 식 (15)와 같다.

$$MSB = \begin{bmatrix} m_{32} & m_{33} & \cdots & m_{39} \\ m_{40} & m_{41} & \cdots & m_{47} \\ & & \vdots & \\ m_{56} & m_{57} & \cdots & m_{63} \end{bmatrix} \oplus \begin{bmatrix} r_{32} & r_{33} & \cdots & r_{39} \\ r_{40} & r_{41} & \cdots & r_{47} \\ & & \vdots & \\ r_{56} & r_{57} & \cdots & r_{63} \end{bmatrix}$$

$$LSB = \begin{bmatrix} m_0 & m_1 & \cdots & m_7 \\ m_8 & m_9 & \cdots & m_{15} \\ & & \vdots & \\ m_{24} & m_{25} & \cdots & m_{31} \end{bmatrix} \odot \begin{bmatrix} r_0 & r_1 & \cdots & r_7 \\ r_8 & r_9 & \cdots & r_{15} \\ & & \vdots & \\ r_{24} & r_{25} & \cdots & r_{31} \end{bmatrix} \quad (15)$$

일반적으로 혼합형 암호시스템은 블록 암호시스템과 스트림 암호시스템의 결합을 의미하며 LFSR의 출력수열을 블록 암호시스템의 키 스케쥴로 사용한다. 이때 기존 혼합형 암호시스템은 입력문과 키가 독립적이며, 키 스케쥴러 부분이 아니라 라운딩 및 iteration을 수행하는 부분에 블록 및 스트림방식을 혼합하는 기법이었다. 그러므로 블록 암호시스템의 데이터 블록 크기가 m인 경우, LFSR 출력수열의 크기는 m을 가진다. 즉 블록 암호시스템의 블록 크기가 스트림 암호시스템의 키 수열의 크기와 같아야 한다. 그러므로 안전성을 위하여 블록크기를 매우 크게 해야 한다는 결론을 가진다. LFSR에 의한 의사난수발생은 LFSR의 출력수열에 대한 주기가 길수록 안전성을 가지게 된다. 그러나 LFSR의 출력수열의 안전성을 위하여 주기를 매우 크게 설정할 경우, 블록 암호시스템의 블록 데이터 크기가 따라서 증가해야 한다는 결론이다. 결국에는 많은 데이터의 암호화를 위하여 너무 많은 잉여분(redundancy)을 산출하게 된다. 그러므로 암호화에 대한 효율성이 떨어지게 된다.

대칭형 암호시스템의 특징 중의 하나는 비밀키를 가지고 있다는 것이다. 그러므로 사용자의 수가 증가하거나 암호화하여 전송해야할 데이터의 블록크기가 매우 길어진다면 대칭형 암호시스템의 장점인 처리속도 효율이 크게 저하될 것이다. 또한 암호화된 데이터와 키 데이터량이 방대하여 암호화 자체 성능도 저하된다.

그러므로 본 논문에서는 기존의 혼합형 암호시스템과 같이 데이터와 별개의 키 데이터를 가지고 암호화를 수행하는 것이 아니고 그림 10과 같이 암호를 수행할 데이터를 키 데이터로 사용하며, 키 수열 생성은 LFSR과 키 재포맷팅을 이용하여 블록 데이터의 크기에 상관없이 키 수열을 생성할 수 있도록 하였다.

그림 10에서 입력 데이터 X에 대하여 키 스케쥴러에 의하여 X에 의한 키 값 K를 생성하고 생성된 킷값 K와 입력데이터 X를 이용하여 암호화된 출력값 Y를 생성한다.

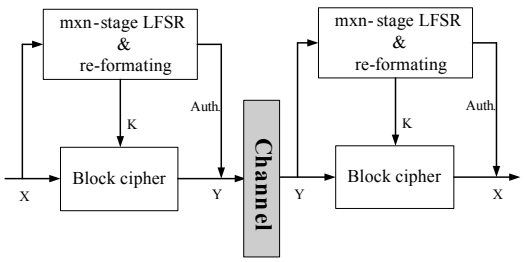

그림 10. 제안된 혼합형 암호시스템
Fig 10. Proposed hybrid cryptosystem

III. 모의실험 및 고찰

제안된 혼합형 암호알고리즘의 구현은 VHDL을 이용하여 Top-down 방식으로 설계하였으며, 회로합성은 Synopsys Design Analyser Ver. 1999.10, QUARTUS 7.0을 사용하였다. 모의실험에 사용된 툴은 Synopsys VHDL Debegger, ModelSim 5.8C이다.

본 논문에서 제안한 혼합형 암호시스템은 128 비트 데이터 길이, 80 비트 키 길이, 50MHz의 시스템 속도를 가진 상태에서 모의실험을 수행하였다.

모의실험 결과, 표 1과 같은 결과값을 얻을 수 있었다. 표 1은 제안된 알고리즘과 기존 블록 암호알고리즘을 5개의 파라미터로 비교분석한 것이다.

표 1. 기존 암호알고리즘과 제안된 암호알고리즘의 비교
Table 1. Comparison of existed & proposed cryptographic algorithms

@50MHz	DES	3DES	SEED	AES	hybrid
구 조	Feistel	Feistel	Feistel	SPN	Feistel & SPN
라운드수	16	48	16	10	1
데이터 길이 (bits)	64	64	128	128/192/256	128
키 길이 (bits)	56	112/168	128	128	80
처리율 (Mbps)	31.6	15.6	313.7	387.9	776.8

표 1에서 비교대상인 xDES[7]는 현재까지 현금인출기 등에서 사용되고 있으며, SEED[8]는 국내 블록암호알고리즘의 표준으로서 사용되고 있고, AES[9][10]는 현재 활성화 되고 있는 알고리즘이기 때문에 제안한 혼합형 알고리즘과 비교대상으로 결정한 것이다.

기존 xDES는 1 라운드, 16 iteration 또는 16라운드, 1 iteration을 수행하는 방식을 취한다. 본 논문에서 제안한 혼합형 방식은 키 스케쥴러를 기존 블록방식들에 비하여 비선형성을 증가시켰으며, 또한 입력문에 대하여 키 값을 종속화 시킴으로서 DC/LC 등에 의한 해석법으로부터 자유롭도록 하였다. 또한, 입력문:키:출력문≠1:1:1 관계 및 Feistel 구조와 SPN 구조를 동시에 사용하여 보다 복잡한 구조를 가지도록 하였다.

또한 기존 알고리즘들의 경우, 인증을 수행하기 위해서 비안전 채널에 대하여 해쉬함수 또는 MAC/MDC 등을 이용하여야 하지만 제안된 알고리즘은 인증을 위한 전처리를 수행하기 때문에 인증용 알고리즘을 처리하는데 부하가 덜 걸리도록 하였다.

IV. 결론

기존 혼합형 암호알고리즘은 평문/암호문과 키가 독립적이었으나 제안된 혼합형 암호알고리즘은 평문을 기준으로 키 스케쥴러에 의하여 종속변수로 변환되고 이를 이용하여 암호문을 생성하는 메카니즘으로 구성되어 있다. 그러므로 기존 혼합형 암호알고리즘의 단점이었던 비도 증가와 의사난수발생기의 주기와의 비례관계를 제안된 알고리즘에서는 적용 받지 않는다. 또한 기존 대칭형 암호시스템에서 사용되는 iteration을 제안된 암호시스템에서는 사용하지 않으므로 처리속도가 매우 증가된다. 또한 제안된 알고리즘의 경우, 내부에 인증을 위한 처리과정이 포함되어 있기 때문에 별도의 처리과정 없이 바로 인증용 알고리즘을 적용하여 인증기능을 수행할 수 있다는 장점이 있다.

그러므로 제안된 혼합형 암호시스템은 대용량 평문에 대한 암호화와 인증이 필요한 네트워크에서의 실시간 암호화가 가능하리라 생각된다.

참고문헌

[1] E. Biham, "On the Applicability of Differential Cryptanalysis to Hash Functions," Lecture at EIES Workshop on Cryptographic Hash Functions, Mar. 1992.

[2] E. Biham, "On Matsui's Linear Cryptanalysis," Advances in Cryptology- EURO-CRYPT'94 Proceedings, Springer- Verlag, pp. 398-412, 1995.

[3] A. G. Broscius and J. M. Smith, "Exploiting Parallelism in Hardware Implementation of the DES," Advances in Cryptology- CRYPTO'91 Proceeding, Springer-verlag, pp. 367-376, 1992

[4] E. Biham, A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," Advances in Cryptology-CRYPTO'90 Proceeding, Springer-verlag, pp. 2-21, 1991.

[5] Nicolas T. Courtois and Gregory V. Bard,

"Algebraic Cryptanalysis of the Data Encryption Standard," In Steven D. Galbraith, editor, Cryptography and Coding -11th IMA International Conference, volume 4887 of Lecture Notes in Computer Science, pp. 152-169, Berlin Heidelberg New York, Springer-Verlag, 2007.

[6] Meiqin Wang, "Differential Cryptanalysis of reduced-round PRESENT," In Serge Vaudenay, editor, Africacrypt 2008, volume 5023 of Lecture Notes in Computer Science, pp. 40-49, Springer- Verlag, 2008.

[7] http://opencores.org/project,3des_vhdl

[8] http://service2.nis.go.kr/pw_certified/seed.jsp

[9] Carlos Cid, Sean Murphy, and Matthew Robshaw, "Algebraic Aspects of the Advanced Encryption Standard," Springer-Verlag, 2006.

[10] *http://www.design-reuse.com/articles /13981 /fpga-implementation-of-aes-encryption -and-decryption.html(FPGA Implementation of AES Encryption and Decryption)*

저 자 소 개

이 선 근

1997 : 원광대학교 공학석사.

2003 : 원광대학교 공학박사.

2010 - 현재 : 전북대학교 IT응용시스템공학과

관심분야 : 정보보호 및 암호알고리즘 설계, 보안시스템 SoC 설계, 임베디드 시스템 설계

caiserrisk@googlemail.com