256 비트 대칭 SPN 블록 암호 XSB

(256 bit Symmetric SPN Block cipher XSB)

조 경 연*
(Gyeong-Yeon Cho)

요 약 본 논문에서는 암호와 복호 과정이 동일한 SPN 구조 256 비트 블록 암호 알고리즘인 가칭 XSB(eXtended SPN Block cipher)를 제안한다. XSB는 짝수 N 라운드로 구성하고, 1 라운드부터 N/2-1 라운드까지는 전함수를 적용하고, N/2+1 라운드부터 N 라운드까지는 후함수를 적용한다. 각 라운드는 키 합산층, 치환층, 바이트 교환층 및 확산층의 네 단계로 구성한다. 또한 전함수단과 후함수단 사이에 대칭 블록을 구성하는 대칭단을 삽입한다. 대칭단은 간단한 비트 슬라이스 대합 S-박스로 구성한다. 비트 슬라이스 대합 S-박스는 Square 공격, 부메랑 공격, 불능 차분 공격 등의 공격을 어렵게 한다.

핵심주제어 : AES, ARIA, SHACAL-2, SPN, 암호, 복호

Abstract In this paper, we propose a SPN 256 bit block cipher so called XSB(eXtended SPN Block cipher) which has a symmetric structure in encryption and decryption. The proposed XSB is composed of the even numbers of N rounds where the first half of them, 1 to N/2-1 round, applies a pre-function and the last half of them, N/2+1 to N round, employs a post-function. Each round consists of a round key addition layer, a substiution layer, a byte exchange layer and a diffusion layer. And a symmetry layer is located in between the pre-function layer and the post-function layer. The symmetric layer is composed with a multiple simple bit slice involution S-Boxes. The bit slice involution S-Box symmetric layer increases difficult to attack cipher by Square attack, Boomerang attack, Impossible differentials cryptanalysis etc.

Key Words : AES, ARIA, SHACAL-2, SPN, Cipher, Decipher

1. 서 론

비밀키 블록 암호는 암호 알고리즘 E에 비밀키 K를 적용하여 평문 P로부터 암호문 $C=E_K(P)$를 생성하며, 복호는 복호 알고리즘 D에 동일한 비밀키 K를 적용하여 암호문 C로부터 평문 $P=D_K(C)$를 얻는다. 이러한 비밀키 블록 암호의 암호 및 복호 알고리즘은 암호 및 복호 함수를 여러 라운드 반복 수행하는 구조이다. 비밀키 블록 암호는 라운드 함수의 구조에 따라서 Feistel 구조[1]와 SPN(Substitution Permutation Network) 구조로 나눌 수 있다.

SPN 구조는 C. E Shannon의 혼돈(Confusion)과 확산(Diffusion)[2] 이론을 바탕으로 하였다. AES[3-4], ARIA[5] 등이 SPN 구조로 블록 크기는 128 비트, 키 길이는 128-256 비트이다. SPN 구조에서의 암호 라운드 함수는 키합산층(Key addition layer)과 혼돈을 수행하는 치환층(Substitution layer) S 및 확산층(Permutation layer) P의 세 단계로 구성된다. 복호 라운드 함수는 역확산층(Inverse Permutation layer)

* 부경대학교 공과대학 IT융합응용공학과

P^{-1}과 역치환층(Inverse Substitution layer) S^{-1} 및 키합산층(Key addition layer)의 세 단계로 구성한다. SPN 구조는 암호와 복호 알고리즘이 다르지만 각 라운드에서 입력이 전부 비선형변환되므로 Feistel 구조에 비하여 라운드의 수가 적어지므로 하드웨어의 동작 속도가 빠른 장점을 가진다.

블록 암호는 블록 크기와 키 길이에 따라서 분류하기도 하는데, 일반적으로 블록 크기는 키 길이와 같거나 작다. 암호의 안전성은 비밀 키를 찾아내는 시도의 횟수 또는 한 회의 시도로 키를 찾아낼 확률로 정의한다. 가장 단순한 공격으로 전수검사가 있다. n 비트 키를 전수검사로 찾아내기 위해서는 2^{n-1}회 시도하면 50% 확률로 키를 찾아낼 수 있다. 어떤 공격으로도 전수검사보다 적은 횟수의 시도로 키를 찾아낼 수 없으면 이를 안전하다고 평가한다. 향후 20-30년 내에서는 키 길이가 128 비트이면 충분하다고 인정하고 있으므로 현재는 128 비트 키 블록 암호가 주로 사용되고 있다. 256 키 블록 암호는 활발하게 연구되고 있지 않으나 SHACAL-2[6-7] 등이 제안되어 있다.

암복호가 동일한 SPN 블럭 암호 알고리즘은 조경연 등의 연구[8-10]가 있다. 본 논문에서는 이들 연구를 확장하여 256 비트 블록과 키를 가지는 암복호가 동일한 SPN 블럭 암호 알고리즘 XSB(eXtended SPN Block cipher)를 제안한다.

XSB는 짝수 N 라운드로 구성하고, 1 라운드부터 N/2-1 라운드까지의 라운드 함수는 치환층(Substitution layer) S, 바이트를 교환하는 교환(Exchange) 계층, 확산층(Permutation layer) P와 키합산층(Key addition layer)의 네 단계로 구성하고 이를 전함수단이라 가칭한다. 후반부 (N/2)+1 라운드부터 N-1 라운드까지는 역치환층(Inverse Substitution layer) S^{-1}, 교환(Exchange) 계층, 확산층(Permutation layer) P 및 키합산층(Key addition layer)의 네 단계로 구성하고 이를 후함수단이라 칭한다. 확산층 P는 MDS(Maximum Distance Separated) 대합(involution) 행렬을 사용하여 전함수단과 후함수단에서 동일하게 구성한다.

전함수단과 후함수단 사이에 규칙적인 라운드의 반복을 피하고, 바이트 단위로 구성된 S-박스의 공격을 어렵게 하기 위하여 비트 슬라이스 대합 S-박스로 구성한 대칭단(Symmetry layer)을 삽입한다.

XSB는 블록 크기가 256 비트이고 14 라운드를 수행한다. 한편 AES는 128 비트 블록이며 256 비트 키에서 14 라운드를 수행한다. 하드웨어로 구현시에 지연 시간은 라운드 수에 비례하므로 XSB와 AES의 하드웨어 지연 시간은 동일하다. 반면에 XSB의 블록 크기가 AES의 두 배이므로 암호 및 복호 속도는 XSB가 AES보다 두 배 빠르다. 한편 소프트웨어로 구현시에는 순차적으로 수행하므로 XSB와 AES의 속도 차이는 크지 않다.

본 논문의 구성은 2장에서 확산 계층을 구성하는 MDS 대합 행렬의 구조를 기술하고, 3장에서 XSB의 구조를 기술한다. 4장에서는 XSB의 안전성을 검증하고, 5장에서 결론을 맺는다.

2. MDS 대합 행렬

MDS 코드[11]를 암호 구조의 선형 변환 확산층에 사용하는 것은 Vaudenay[12]에 의하여 제안되었고, SHARK[13], SQUARE[14]에 채용되었으며 AES의 MixColumn() 및 InvMixColumn()도 MDS 코드를 구성한다. 선형 코드 (n, k, d)에서 'd=n-k+1'일 때 이 코드가 MDS 코드를 구성하는 조건은 lemma-1로 정의된다.

Lemma-1[12] : 생성 행렬 $G=[I|A]$를 가지는 (n, k, d) 코드가 MDS이기 위한 필요충분조건은 모든 부분 정방 행렬(square submatrix)이 비특이(nonsingular)인 경우이다. 단, A는 'k * (n-k)' 행렬이다.

MDS 대합 행렬은 lemma-2로 구성할 수 있다.

Lemma-2[12] : $x_0,\cdots,x_{n-1}$과 $y_0,\cdots,y_{n-1}$이 주어졌을 때, 행렬 $A=[a_{ij}]$, $0\le i,j\le n-1$, $a_{ij}=\frac{1}{x_i\oplus y_j}$을 Cauchy 행렬이라고 한다. 단. 모든 i, j에 대하여 '$x_i\neq x_j$', '$y_i\neq y_j$', '$x_i\neq y_j$'이다. Cauchy 행렬 A는 $A^2=c^2I$, $c=\bigoplus_{i=1}^{n} a_{ij}^2$이 된다. 따라서 행렬 A의 모든 원소 a_{ij}를 $\sqrt{c}=\bigoplus_{i=1}^{n} a_{1i}$로 나누어 생성한 행렬은 MDS 대합 행렬이 된다.

본 논문에서는 lemma-2에 의하여 4X4 MDS 대합 행렬을 다음과 같이 정의한다.

$$M=\begin{bmatrix} 0x1b & 0x1c & 0x14 & 0x12 \\ 0x1c & 0x1b & 0x12 & 0x14 \\ 0x14 & 0x12 & 0x1b & 0x1c \\ 0x12 & 0x14 & 0x1c & 0x1b \end{bmatrix}$$

3. XCB의 구조

본 논문에서는 AES에서 정의한 기호를 사용한다. 기약다항식은 $m(x)=x^8+x^4+x^3+x+1$을 사용한다. 블록 크기는 256 비트이며, 이는 32 바이트 $\{I_0, \cdots, I_{15}\}$로 구성한다. S-박스 및 S^{-1}-박스는 AES와 동일하게 구성한다.

내부 연산은 '4 X 8' 바이트 정방 행렬인 스테이트(state) 단위로 수행한다. 입력 $\{I_0 \cdots I_{15}\}$과 스테이트 그리고 출력 $\{O_0 \cdots O_{15}\}$을 그림 1에 보인다.

I_0	I_4	I_8	I_{12}	I_{16}	I_{20}	I_{24}	I_{28}
I_1	I_5	I_9	I_{13}	I_{17}	I_{21}	I_{25}	I_{29}
I_2	I_6	I_{10}	I_{14}	I_{18}	I_{22}	I_{26}	I_{30}
I_3	I_7	I_{11}	I_{15}	I_{19}	I_{23}	I_{27}	I_{31}

==>

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	$s_{0,4}$	$s_{0,5}$	$s_{0,6}$	$s_{0,7}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,4}$	$s_{1,5}$	$s_{1,6}$	$s_{1,7}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	$s_{2,4}$	$s_{2,5}$	$s_{2,6}$	$s_{2,7}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$	$s_{3,5}$	$s_{3,6}$	$s_{3,7}$

==>

O_0	O_4	O_8	O_{12}	O_{16}	O_{20}	O_{24}	O_{28}
O_1	O_5	O_9	O_{13}	O_{17}	O_{21}	O_{25}	O_{29}
O_2	O_6	O_{10}	O_{14}	O_{18}	O_{22}	O_{26}	O_{30}
O_3	O_7	O_{11}	O_{15}	O_{19}	O_{23}	O_{27}	O_{31}

<그림 1> XSB의 입력, 스테이트 및 출력

스테이트는 CPU 워드에 따라서 16개의 16 비트 워드, 8개의 32 비트 워드 또는 4개의 64 비트 워드로 구성한다. 32비트 워드의 구성은 $SW_n=\{s_{0,n}, s_{1,n}, s_{2,n}, s_{3,n}\}$, $n=\{0,1,\cdots 7\}$과 같다. 워드내의 바이트 순서 즉 엔디안은 CPU에 따라서 빅 엔디안 또는 리틀 엔디안을 취하며, 이는 알고리즘에 영향을 주지 않는다.

키는 256 비트이며 14 라운드를 수행한다. 키 길이 및 라운드 수는 변경이 가능한데, 본 논문에서는 설명의 편의상 256 비트와 14 라운드를 가정한다.

그림 2에 XSB 암호 알고리즘과 그림 3에 복호 알고리즘의 의사 코드를 보인다. NR은 라운드 수이다.

```
XSB_Encyption(byte in[32], byte out[32], byte
erk[NR+1][32])
  begin
    byte state[8][4]

    state = in ;

    Pre_stage(state, erk)
    Symm_Funct(state, erk[NR/2])
    Post_state(state, erk)

    out = state ;
  end
```

<그림 2> XSB 암호 알고리즘

```
XSB_Decyption(byte in[32], byte out[32], byte
drk[NR+1][32])
  begin
    byte state[8][4]

    state = in ;

    Pre_stage(state, drk)
    Symm_Funct(state, drk[NR/2])
    Post_state(state, drk)

    out = state ;
  end
```

<그림 3> XSB 복호 알고리즘

그림 2와 그림 3은 동일하며 단지 사용하는 라운디 키만이 다르다. 따라서 XSB의 암호와 복호 알고리즘은 동일하다. 전함수단과 후함수단을 각각 그림 4와 그림 5에 보인다.

```
Pre_Stage
    (byte state[8][4], byte key[NR+1][32])

  begin
    AddRoundKey(state, key[0])

    for round = 1 to NR/2-1 step 1
      SubBytePre(state)
      ExgRow(state)
      MixColumn(state)
      AddRoundKey(state, key[round])

    SubBytePre(state)
    ExgRow(state)
  end
```

<그림 4> 전함수단 알고리즘

```
Post_Stage
  (byte state[8][4], byte key[NR+1][32])

  begin
   for round = NR/2+1 to NR-1 step 1
       SubBytePost(state)
       ExgRow(state)
       MixColumn(state)
       AddRoundKey(state, key[round])

     SubBytePost(state)
     ExgRow(state)
     AddRoundKey(state, key[NR])
  end
```

<그림 5> 후함수단 알고리즘

그림 4와 그림 5에서 사용하는 각각의 함수는 다음과 같다.

- AddRoundKey() 함수는 라운드 키 덧셈 함수이다.
- SubBytePre() 함수는 전함수단의 라운드에서 8비트 S-박스를 이용한 비선형 바이트 치환 함수이다. 그림 6에 함수의 동작을 보인다. 그림 6에서 $S[x]$와 $S^{-1}[x]$는 입력 x를 S-박스 및 S^{-1}-박스에 의하여 각각 비선형변환한 것을 나타낸다.

$s_{0,0}$	$s_{0,1}$	⋯	$s_{0,7}$
$s_{1,0}$	$s_{1,1}$	⋯	$s_{1,7}$
$s_{2,0}$	$s_{2,1}$	⋯	$s_{2,7}$
$s_{3,0}$	$s_{3,1}$	⋯	$s_{3,7}$

==>

$S[s_{0,0}]$	$S[s_{0,1}]$	⋯	$S[s_{0,7}]$
$S^{-1}[s_{1,0}]$	$S^{-1}[s_{1,1}]$	⋯	$S^{-1}[s_{1,7}]$
$S[s_{2,0}]$	$S[s_{2,1}]$	⋯	$S[s_{2,7}]$
$S^{-1}[s_{3,0}]$	$S^{-1}[s_{3,1}]$	⋯	$S^{-1}[s_{3,7}]$

<그림 6> SubBytePre() 함수 동작

- SubBytePost() 함수는 후함수단의 라운드에서 8비트 S-박스를 이용한 비선형 바이트 치환 함수이다. 그림 7에 함수의 동작을 보인다.

$s_{0,0}$	$s_{0,1}$	⋯	$s_{0,7}$
$s_{1,0}$	$s_{1,1}$	⋯	$s_{1,7}$
$s_{2,0}$	$s_{2,1}$	⋯	$s_{2,7}$
$s_{3,0}$	$s_{3,1}$	⋯	$s_{3,7}$

==>

$S^{-1}[s_{0,0}]$	$S^{-1}[s_{0,1}]$	⋯	$S^{-1}[s_{0,7}]$
$S[s_{1,0}]$	$S[s_{1,1}]$	⋯	$S[s_{1,7}]$
$S^{-1}[s_{2,0}]$	$S^{-1}[s_{2,1}]$	⋯	$S^{-1}[s_{2,7}]$
$S[s_{3,0}]$	$S[s_{3,1}]$	⋯	$S[s_{3,7}]$

<그림 7> SubBytePost() 함수의 동작

- ExgRow() 함수는 스테이트의 바이트를 그림 8과 같이 교환한다.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	$s_{0,4}$	$s_{0,5}$	$s_{0,6}$	$s_{0,7}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,4}$	$s_{1,5}$	$s_{1,6}$	$s_{1,7}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	$s_{2,4}$	$s_{2,5}$	$s_{2,6}$	$s_{2,7}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$	$s_{3,5}$	$s_{3,6}$	$s_{3,7}$

==>

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	$s_{0,4}$	$s_{0,5}$	$s_{0,6}$	$s_{0,7}$
$s_{1,1}$	$s_{1,0}$	$s_{1,3}$	$s_{1,2}$	$s_{1,5}$	$s_{1,4}$	$s_{1,7}$	$s_{1,6}$
$s_{2,7}$	$s_{2,2}$	$s_{2,1}$	$s_{2,4}$	$s_{2,3}$	$s_{2,6}$	$s_{2,5}$	$s_{2,0}$
$s_{3,4}$	$s_{3,5}$	$s_{3,6}$	$s_{3,7}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

<그림 8> ExgRow() 함수 동작

첫 번째 행은 교환하지 않으며, 두 번째 행은 (0,1), (2,3), (4,5), (6,7)을 서로 교환하고, 세 번째 행은 (0,7), (1,2), (3,4), (5,6)을 서로 교환하며, 네 번째 행은 (0,4), (1,5), (2,6), (3,7)을 서로 교환한다.

- MixColumn() 함수는 열 단위로 혼합을 수행하는 4 8-비트 선형 변환 함수이다. 2장에서 정의한 MDS 대합 행렬을 사용하여, 그림 9와 같이 동작한다.

$$\begin{bmatrix} S'_{0,n} \\ S'_{1,n} \\ S'_{2,n} \\ S'_{3,n} \end{bmatrix} = \begin{vmatrix} 0x1b & 0x1c & 0x14 & 0x12 \\ 0x1c & 0x1b & 0x12 & 0x14 \\ 0x14 & 0x12 & 0x1b & 0x1c \\ 0x12 & 0x14 & 0x1c & 0x1b \end{vmatrix} \begin{vmatrix} S_{0,n} \\ S_{1,n} \\ S_{2,n} \\ S_{3,n} \end{vmatrix}$$

<그림 9> MixColumn() 함수 동작

XSB를 소프트웨어로 효과적으로 구현하기 위해서 256 4-바이트 테이블 T0-T7를 다음과 같이 정의한다.

$$T0[a] = \begin{bmatrix} S[a]*0x1b \\ S^{-1}[a]*0x1c \\ S[a]*0x14 \\ S^{-1}[a]*0x12 \end{bmatrix} \qquad T1[a] = \begin{bmatrix} S[a]*0x1c \\ S^{-1}[a]*0x1b \\ S[a]*0x12 \\ S^{-1}[a]*0x14 \end{bmatrix}$$

$$T2[a] = \begin{bmatrix} S[a]*0x14 \\ S^{-1}[a]*0x12 \\ S[a]*0x1b \\ S^{-1}[a]*0x1c \end{bmatrix} \qquad T3[a] = \begin{bmatrix} S[a]*0x12 \\ S^{-1}[a]*0x14 \\ S[a]*0x1c \\ S^{-1}[a]*0x1b \end{bmatrix}$$

$$T4[a] = \begin{bmatrix} S^{-1}[a]*0x1b \\ S[a]*0x1c \\ S^{-1}[a]*0x14 \\ S[a]*0x12 \end{bmatrix} \qquad T5[a] = \begin{bmatrix} S^{-1}[a]*0x1c \\ S[a]*0x1b \\ S^{-1}[a]*0x12 \\ S[a]*0x14 \end{bmatrix}$$

$$T6[a] = \begin{bmatrix} S^{-1}[a]*0x14 \\ S[a]*0x12 \\ S^{-1}[a]*0x1b \\ S[a]*0x1c \end{bmatrix} \qquad T7[a] = \begin{bmatrix} S^{-1}[a]*0x12 \\ S[a]*0x14 \\ S^{-1}[a]*0x1c \\ S[a]*0x1b \end{bmatrix}$$

T0-T3 테이블은 전함수단의 라운드에서 사용하여 SubBytePre(), ExgRow(), MixColumn()을 4번의 테이블 참조와 3번의 XOR 연산으로 구현할 수 있다. T4-T7 테이블은 후함수단의 라운드에서 사용하여 SubBytePost(), ExgRow(), MixColumn()을 4번의 테이블 참조와 3번의 XOR 연산으로 구현할 수 있다.

XSB의 대칭단 구조의 의사코드를 그림 10에 보인다.

```
Symm_Funct(word sw[8], word k[8])
  begin
   word t[2]

   t[0] = sw[4]⊕k[0]
   t[1] = sw[6]⊕k[1]
   sw[4] = sw[2]⊕((t[0]⊕sw[0])∨t[1])
   sw[6] = sw[0]⊕~(sw[4]∧t[1])
   sw[0] = t[1]⊕~(t[0]∧sw[6])
   sw[2] = t[0]⊕((sw[0]⊕sw[4])∨sw[6])
   sw[4] = sw[4]⊕k[4]
   sw[6] = sw[6]⊕k[5]

   t[0] = sw[5]⊕k[2]
   t[1] = sw[7]⊕k[3]
   sw[5] = sw[3]⊕((t[0]⊕sw[1])∨t[1])
   sw[7] = sw[1]⊕~(sw[5]∧t[1])
   sw[1] = t[1]⊕~(t[0]∧sw[7])
   sw[3] = t[0]⊕((sw[1]⊕sw[5])∨sw[7])
   sw[5] = sw[5]⊕k[6]
   sw[7] = sw[7]⊕k[7]

  end
```

<그림 10> XSB 대칭 함수

그림 10의 대칭단은 워드 단위로 연산을 수행하므로 CPU의 엔디안(endianness)과 무관하게 구현할 수 있다.

x 워드의 i번 비트를 x^i로 표현하면, 대칭단은 $F: \{a[3]^i, a[2]^i, a[1]^i, a[0]^i\} \mapsto \{b[3]^i, b[2]^i, b[1]^i, b[0]^i\}$인 함수로 4 비트 S-박스에 대응한다. 따라서 대칭단은 64개의 4비트 S-박스로 해석할 수 있다. 표-1에 대칭단 4비트 S-박스의 입출력을 보인다.

<표-1> 대칭단 S-박스의 입출력(16 진수)

in	0	1	2	3	4	5	6	7
out	b	5	f	3	c	1	8	7

in	8	9	a	b	c	d	e	f
out	6	e	a	0	4	d	9	2

대칭단 S-박스의 최대 선형근사 확률(maximum linear approximation probability)은 1/4, 최대 차분(maximum difference)은 1/4이다. 그러므로 대칭단 S-박스는 선형공격 및 차분공격에 최대의 저항성을 가진다. 대칭단 S-박스는 함수이며, $F(A) \mapsto B$ 이면 $F(B) \mapsto A$이다. 따라서 그림 10의 비트 슬라이스 S-박스는 대합(involution)이다.

암호 라운드 키와 복호 라운드 키는 교환해서 적용한다. 즉, 암호시에에 k[0]-k[3]은 복호시에 각각 k[4]-k[7]이 되며, 암호시에 k[4]-k[7]은 복호시에 각각 k[0]-k[3]이 된다.

라운드 키는 AES와 유사하게 생성한다. AES에서는 4개의 S-박스를 사용했으나, XSB에서는 $\{S, S^{-1}, S, S^{-1}\}$를 사용한다. AES와 동일하게 생성한 라운드 키 {erk[0], erk[1],..., rk[14]}가 암호 라운드 키이다.

MixColumn()은 선형 변환 함수이므로 '$M(A \oplus B) = MA \oplus MB$'가 성립한다. 따라서 drk[7]을 제외한 복호 라운드 키 {drk[0], drk[1], ... drk[6], drk[8], ..., drk[14]}는 다음과 같이 생성한다.

```
drk[0] = erk[14],
drk[1] = MixCloumn(erk[13]),
    .....
drk[13] = MixColumn(erk[1]),
drk[14] = erk[0].
```

drk[7]은 대칭 함수의 라운드 키로 다음과 같이 생성한다.

drk[7][0] = erk[7][16],
drk[7][1] = erk[7][17],
.....
drk[7][15] = erk[7][31],
drk[7][16] = erk[7][0],
drk[7][17] = erk[7][1],
.....
drk[7][31] = erk[7][15],

4. XSB의 성능 및 안전성

XSB를 소프트웨어로 구현하는 경우에 속도를 빠르게 하기 위하여 T0-T7의 256 4-바이트 테이블을 사용하는데 이는 AES와 동일한 형태의 테이블이다. 한편 XSB에서는 AES에 없는 대칭단이 추가되어 있으므로 이만큼 동작 속도가 느리다. ARIA는 암호와 복호 알고리즘이 동일한 SPN 블록 암호이다. ARIA는 확산 계층을 16X16 바이트 대합 행렬로 구성했다. [5]에서 제안한 32 비트 프로세서에서 소프트웨어 구현 방식에 의한 16 라운드 ARIA와 14 라운드 AES 및 XSB의 소프트웨어 속도 비교를 표-2에 보인다.

<표-2> 소프트웨어 수행 시간 비교

XSB	SHACAL-2	AES-256	ARIA-256
100	130	94	320

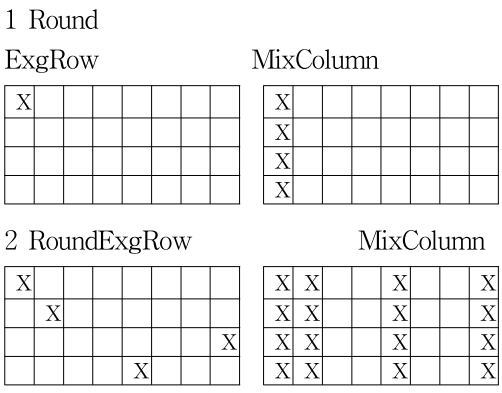
GNU-C 컴파일러를 사용하였으며 Windows XP, 셀러론 2.8GHz, 2G RAM 환경에서 16억 바이트에 대하여 XSB와 14 라운드 AES, 16 라운드 ARIA 및 64 라운드 SHACAL-2를 비교하였다. XSB는 AES에 비하여 6% 정도 소프트웨어 속도가 느리며, ARIA에 비하여 3.2배 정도 빠르다. 256 비트 블록 암호인 SHACAL-2에 비하여 XSB가 30% 정도 소프트웨어 동작 속도가 빠르다.

XSB를 하드웨어로 구현하면 T0-T7의 256 4-바이트 테이블을 사용하지 않으며 8 비트 S-박스를 사용하는데 이는 AES의 암호 부분과 동일한 구조가 된다. XSB와 256 비트 AES는 모두 14 라운드이므로 하드웨어 지연 시간은 동일하다. 반면에 XSB의 블록 크기가 AES의 두 배이므로 암복호 속도는 2배이다. 한편 XSB는 암호와 복호 알고리즘이 동일하므로 블록 크기가 두 배이지만 하드웨어의 복잡도는 유사하다.

XSB는 AES와 동일한 구조를 가지고 있으므로 안전성 또한 AES와 동일하다. AES에 대한 안전성 분석은 차분 공격[15], 선형 공격[16], Square 공격[13], 부메랑 공격[17], 불능 차분 공격(Impossible Differentials Cryptanalysis)[18], 부정차분공격(Truncated Differentials Cryptanalysis) [19] 등이 수행되었다. 이들 공격은 공격 패스를 설정하고, 설정한 패스에서 활동성을 가지는 S-박스의 차분/선형 확률에 의하여 공격 복잡도를 계산한다. 그러므로 XSB의 안전성을 검증하기 위해서는 최적의 패스에서 활동성을 가지는 S-박스의 수를 산출하고 이를 AES와 비교하여 상대적인 안전성을 분석할 수 있다. 그림 11에 6 라운드까지 XSB에서 최소로 활동성을 가지는 크리티칼 패스(critical path)를 보인다.

1 라운드 초기 상태에서 한 개의 S-박스만이 활동성을 가지도록 평문을 설정한다. 그림 11에서는 활동성을 가지는 S-박스의 위치를 'x'로 표시했다. 1 라운드에서는 ExgRow()를 수행해도 여전히 한 개의 S-박스만이 활성화된다. MixColumn() 함수를 수행하면 하나의 열에 속한 모든 S-박스가 활성화된다. 이는 XSB의 MixColumn()이 MDS 행렬을 이루기 때문이다.

2 라운드의 입력은 1 라운드의 MixColumn() 결과이다. 라운드 키 덧셈은 선형변환이므로 S-박스의 활성 상태에 영향을 주지 않는다. ExgRow()를 수행하면 활성 S-박스의 위치가 변화한다. 2 라운드에서는 네 개의 S-박스가 활성화된다. MixColumn() 함수를 수행하면 각 열의 모든 S-박스가 활성화된다.



3 Round

ExgRow

X	X			X			X
X	X				X	X	
X		X	X				X
X			X	X	X		

MixColumn

X	X	X		X		X	X
		X	X	X	X	X	X
	X	X	X	X	X	X	
	X	X	X		X	X	X

4 Round

ExgRow

X	X	X		X		X	X
		X	X	X	X	X	X
	X	X	X	X	X	X	
	X	X	X		X	X	X

MixColumn

X	X			X			X
X	X				X	X	
X		X	X				X
X			X	X	X		

5 Round

ExgRow

X	X			X			X
X	X			X			X
X	X			X			X
X	X			X			X

MixColumn

X							
	X						
							X
				X			

6 Round

ExgRow

X							
X							
X							
X							

MixColumn

X							

<그림 11> XSB의 최소 활성 S-박스

3 라운드에서는 16개의 S-박스가 활성화된다. MixColumn() 함수를 수행하면 24개의 S-박스가 활성화된다. 이는 MixColumn()이 MDS 행렬이므로 두 라운드에서 최소 5개의 S-박스가 활성화되기 때문이다.

4 라운드의 입력에는 24개의 S-박스가 활성화되는데, 다음 라운드에서 활성화되는 최소 S-박스의 수는 16개이다. 이는 공격이 가장 용이한 경우를 가정한 것이다. 3 라운드와 4 라운드와 같은 경우가 실제로 발생하는 입력을 찾지 못했으나 공격이 가장 용이한 패스로 설정하였다.

5-6 라운드는 1-2 라운드를 역으로 진행하는 구조로 이것이 최소 활성화되는 경우이다.

XSB에서는 S-박스가 최소 활성화되는 주기가 6 라운드이다. 한편 전함수단과 후함수단에 마지막 라운드가 하나씩 추가되므로 전체적으로 14 라운드가 필요하다.

SPN 구조 암호 알고리즘의 안전성 평가는 Hong [20] 등에 의해서 제안된 바 있다. 차분/선형 분석법은 차분/선형 활동성을 가지는 S-박스의 수를 구해서 안전성을 분석할 수 있다. XSB S-박스의 최대 차분/선형 공격 확률은 각각 2^{-6} 이다. S-박스와 S^{-1}-박스는 일대일 대응되는 전단사 함수로 같은 차분/선형 확률 값을 가진다.

그림 11에서 6 라운드 XSB에서 활성화 되는 S-박스의 수는 총 65개이다. 차분/선형 공격은 마지막 라운드의 라운드 키를 탐색하므로 마지막 라운드의 S-box는 안전도에 영향을 주지 않는다. XSB에서 13 라운드까지 최소 131개의 S-박스가 활성화된다. 따라서 S-박스에 의한 XSB의 공격 확률은 $(2^{-6})^{131} = 2^{-786}$이다. 대칭단 4비트 S-박스에서 n(0<n<9) 비트가 활성화 되어있다면 대칭단의 차분/선형 확률은 2^{-2n}이다. 그러므로 XSB의 키를 찾기 위해서는 2^{786+2n} 회의 공격이 필요하다.

유사한 방법으로 AES를 분석하면 AES는 4 라운드에 최소 25개의 S-박스가 활성화된다. 14 라운드 AES에서는 마지막 라운드를 제외하면 76개의 S-박스가 활성화된다. ARIA는 확산 계층의 branch number가 8이다. 즉, 최악의 경우에 두 라운드에서 8개의 S-박스만이 활성화된다. 그러므로 16 라운드 ARIA에서 마지막 라운드를 제외하면 57개의 S-박스가 활성화된다.

<표-3> 차분 및 선형 공격 비교

	Differential attack	Linear attack
XSB	$2^{-786-2n}$	$2^{-780-2n}$
AES-256	2^{-456}	2^{-456}
ARIA-256	2^{-342}	2^{-342}

표-3에 XSB, 14 라운드 AES 및 16 라운드 ARIA의 차분 및 선형 공격 확률을 보인다.

부정차분공격은 바이트 단위로 변환되는 암호에서 바이트 패턴이 서로 다른 경우 '1', 같은 경우 '0'으로 정의된 차이 값을 가지고 분석하는 것으로 입력차분과 출력차분의 전부를 고려해야하는 차분분석보다 용이하게 공격할 수 있다. 불능차분공격은 차분확률이 '0'에 수렴하거나, 차분이 존재하지 않는 경우 평문쌍에 이와 같은 차분을 가지는 라운드 키는 제외한 후

남은 라운드 키를 가지고 틀린 키를 제외하는 분석방법으로 가능한 후보 서브키에 한정해서 적용하는 방법이다. Square 공격 및 부매랑 공격은 바이트 패턴이 각각의 라운드 사이에서 전파되는 특성을 이용한 공격이다.

이들 공격은 S-박스의 크기가 모두 동일하고 확산계층이 동일한 AES 및 ARIA에 유효한 공격 방법이다. 그러나 본 논문에서 제안하는 XSB의 대칭단은 64개의 비트 슬라이스 4 비트 S-박스로 구성되어 있다. 즉, 대칭단에서 바이트 패턴이 완전히 바뀌므로 부정차분공격, 불능차분공격, Square 공격 및 부매랑 공격 등은 XSB의 효율적인 공격 방법이 아니다.

5. 결 론

본 논문에서는 암호와 복호 과정이 동일한 SPN 구조 블록 암호 알고리즘인 가칭 XSB를 제안했다. XSB는 짝수 N 라운드로 구성하고, 1 라운드부터 N/2-1 라운드까지의 라운드 함수는 치환층(Substitution layer) S, 바이트를 교환하는 교환(Exchange) 계층, 확산층(Permutation layer) P와 키합산층(Key addition layer)의 네 단계로 구성하고 이를 전함수단이라 가칭한다. 후반부 (N/2)+1 라운드부터 N-1 라운드까지는 역치환층(Inverse Substitution layer) S^{-1}, 교환(Exchange) 계층, 확산층(Permutation layer) P 및 키합산층(Key addition layer)의 네 단계로 구성하고 이를 후함수단이라 칭한다. 전함수단과 후함수단 사이에 비트 슬라이스 대합 S-박스로 구성한 대칭단(Symmetry layer)을 삽입한다. XSB는 블록 크기가 256 비트이고 14 라운드를 수행한다.

치환 계층의 S-박스는 안전성이 검증된 AES의 S-박스를 사용했다. 전함수단 라운드의 치환 계층은 $\{S, S^{-1}, \ldots S, S^{-1}\}$으로 구성하고, 후함수단 라운드의 치환 계층은 이와 반대 순서로 구성했다. 확산층 P는 MDS 대합 행렬을 사용하여 전함수단과 후함수단에서 동일하게 구성한다. XSB는 암호와 복호 알고리즘이 동일하며, 적용하는 라운드 키는 상호 역순으로 되어 있다.

XSB의 소프트웨어 수행 속도는 14 라운드 AES보다 6% 정도 느린데, 이는 대칭단의 추가에 따른 것이다. ARIA는 XSB와 유사하게 암복호가 동일한 SPN 블록 암호인데, XSB는 16 라운드 ARIA과 비교하여 소프트웨어 수행 속도가 약 3.2배 정도 빠르다. 또한 256 비트 블록 암호인 SHACAL-2에 비하여 30% 정도 소프트웨어 동작 속도가 빠르다.

XSB와 256 비트 AES는 모두 14 라운드이므로 하드웨어 지연 시간은 동일하다. 반면에 XSB의 블록 크기가 AES의 두 배이므로 암복호 속도는 두 배 빠르다. 한편 XSB는 암호와 복호 알고리즘이 동일하므로 블록 크기가 두 배이지만 하드웨어의 복잡도는 AES와 유사하다.

XSB는 AES와 동일한 구조를 가지고 있으므로 안전성은 AES와 동일한 방식으로 검증할 수 있다. 블록 암호의 안전성 분석은 차분 공격, 선형 공격, Square 공격, 부매랑 공격, 불능 차분 공격, 부정차분공격 등이 있다. 이들 공격은 공격 패스를 설정하고, 설정한 패스에서 활동성을 가지는 S-박스의 차분/선형 확률에 의하여 공격 복잡도를 계산한다. XSB의 차분/선형 격 확률은 $2^{-768-2n}$, $1 \le n \le 9$로 14 라운드 AES의 2^{-456}보다 충분히 작아서 차분/선형 공격으로부터 안전하다.

부정차분공격, 불능차분공격, Square 공격 및 부매랑 공격 등은 S-박스의 크기가 모두 동일하고 바이트 패턴이 각각의 라운드 사이에서 전파되는 특성이 동일한 경우에 유용한 공격이다. 이들 공격은 AES, ARIA 등에 유효한 공격 방법이다. 본 논문에서 제안한 XSB의 대칭단은 64개의 비트 슬라이스 4 비트 S-박스로 구성되어 있다. 즉, 대칭단에서 바이트 패턴이 완전히 바뀌므로 이들 공격은 XSB의 효율적인 공격 방법이 아니다.

참 고 문 헌

[1] H. Feistel, “Cryptography and Computer Privacy.” Scientific American, Vol.228, No.5, pp. 15-23, 1973.

[2] C.E. Shannon, “Communication Theory of Secrecy System” Bell System Technical Journal, Vol. 28, No. 4, page 656-715, 1949.

[3] “Report on the Development of the Advanced Encryption Standard(AES)”, http://www.csrc.nist.gov/

encryption/aes/.
[4] J. Daemen, and V. Rijmen, “AES Proposal: Rijndael,” http://www.csrc.nist.gov/ encryption/ aes/rijndael/Rijndael.pdf, 1999.
[5] Daesung Kwon, et. al., "New block cipher : ARIA," Information security and cryptology - ICISC 2003, 6th international, pp. 432-445, 2003
[6] H. Handschuh and D. Naccache, “SHACAL, " Primitive submitted to NESSIE by Gemplus, Sep. 2000.
[7] “New European Schemes for Signatures. Integrity. and Encryption(NESSIE).” http:// cryptonessie.org/.
[8] 조경연, 송홍복, “암호와 복호가 동일한 변형 AES,” 한국산업정보학회논문지, 제15권, 2호, pp. 1-9, 6월 2010.
[9] 조경연, “암호와 복호가 동일한 SPN 블록 암호 SSB,” 한국해양정보통신학회논문지, 제15권, 4호, pp. 860-868, 2011.
[10] 조경연, 송홍복, “비트 슬라이스 대합 S-박스에 의한 대칭 SPN 블록암호,” 한국전자통신학회논문지, 제6권, 2호, pp. 171-179, 2011.
[11] A. M. Youssef, S. Mister, and S. E. Tavares, “On the Design of linear Transformation for Substitution and Permutation Encryption Networks,” in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC '97), pp. 40-48, Aug. 1997.
[12] S. Vaudenay, "On the need for multipermutations: Cryptanalysis of MD4 and SAFER," Proc. of Fast Software Encryption(2), LNCS 1008, Springer-Verlag, pp. 286-297, 1995
[13] V. Rijmen, J. Daemen, B. Preneel, A. Bosselares, and E. De Win, "The cipher SHARK," Fast Software Encryption, LNCS 1-39, D. Gollmann Ed., Springer-Verlag, pp. 99-112, 1996
[14] J. Daemen, L. Knudsan, and V. Rijmen, “The Block Cipher Square," Proceeding of FSE'97, LNCS Vol.1267, pp. 149-165, 1997.
[15] E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES", LNCS 537, page 2-21, 1990.
[16] M. Matsui, "Linear Cryptanalysis Method for DES", LNCS 765, page 386-397, 1994.
[17] A. Birynkov, "The Boomerang attack on 5 and 6-round reduced AES", LNCS 3373, page 42-57, 2005.
[18] J. Cheon, M. Kim, K. Kim, J. Lee and S. Kang, "Improved impossible differential cryptanalysis of Rijndael and Crypton", LNCS 2288, page 39-49, 2001.
[19] L. R. Knudsen, “Truncated and higher order differential,” Fast Software Encryption-Second International Workshop, LNCS Vol.1008, pp. 196-211, 1995.
[20] S. Hong, S. Lee, J. Lim, J. Sung, and D. Cheon, “Provable security against differential and linear cryptanalysis for the SPN structure,” In Fast Software Encryption 2000, LNCS Vol.1978, pp. 273-283, 2001.

조 경 연 (Gyeong-Yeon Cho)

- 1990년 2월 인하대학교 전자공학과 박사
- 1983년~1991년 삼보컴퓨터 기술연구소 책임연구원
- 1991년~2003년 삼보컴퓨터 기술연구소 기술고문
- 1998년~현재 에이디칩스(주) 기술고문
- 1991년~현재 부경대학교 공과대학 IT융합응용공학과 교수
- 관심분야 : 전산기구조, 반도체회로 설계, 암호 알고리즘

논 문 접 수 일 : 2011년 12월 21일
1 차 수 정 완 료 일 : 2012년 02월 01일
2 차 수 정 완 료 일 : 2012년 04월 23일
게 재 확 정 일 : 2012년 05월 24일