情報保護學會論文誌
第16卷 第6號, 2006. 12

경량화된 확산계층을 이용한 32-비트 구조의 소형 ARIA 연산기 구현

유 권 호,[†‡] 구 본 석, 양 상 운, 장 태 주
국가보안기술연구소

Area Efficient Implementation of 32-bit Architecture of ARIA Block Cipher Using Light Weight Diffusion Layer

Gwonho Ryu,[†‡] Bonseok Koo, Sangwoon Yang, Taejoo Chang,
National Security Research Institute(NSRI)

요 약

최근 휴대용 기기의 중요성이 증가하면서 이에 적합한 암호 구현이 요구되고 있으나, 기존의 암호 구현 방식이 속도에 중점을 두고 있어 휴대용 기기에서 요구하는 전력 소모나 면적을 만족하지 못하고 있다. 따라서 휴대용 기기에 적합한 암호 알고리즘의 경량 구현이 매우 중요한 과제로 떠오르고 있다. 이 논문에서는 국내 KS 표준 알고리즘인 ARIA 알고리즘을 32-비트 구조를 이용하여 경량화하는 방법을 제안한다. 확산 계층의 새로운 설계를 이용하여 구현된 결과는 아남 0.25um 공정에서 11301 게이트를 차지하며, 128-비트 키를 이용할 때 87/278/256 클락 (초기화/암호화/복호화)을 소모한다. 그리고 128-비트 키만을 지원하는 기존의 구현과 달리, 256-비트 키까지 지원하도록 구성하여 ARIA 알고리즘의 표준을 완벽히 구현하였다. 이를 통해 지금까지 알려진 가장 경량화된 구현 결과와 비교하면 면적은 7% 감소, 속도는 13% 향상된 결과이다.

ABSTRACT

Recently, the importance of the area efficient implementation of cryptographic algorithm for the portable device is increasing. Previous ARIA(Academy, Research Institute, Agency) implementation styles that usually concentrate upon speed, are not suitable for mobile devices in area and power aspects. Thus in this paper, we present an area efficient ARIA processor which use 32-bit architecture. Using new implementation technique of diffusion layer, the proposed processor has 11301 gates chip area. For 128-bit master key, the ARIA processor needs 87 clock cycles to generate initial round keys, 278 clock cycles to encrypt, and 256 clock cycles to decrypt a 128-bit block of data. Also the processor supports 192-bit and 256-bit master keys. These performances are 7% in area and 13% in speed improved results from previous cases.

Keywords : *ARIA processor, Diffusion, Light Weight, gate count*

I. 서 론

최근의 암호 구현의 경향은 기존의 파이프라인 구조를 이용한 고속 구현에서 경량 구현 쪽으로 선회하

접수일: 2006년 6월 16일 ; 채택일: 2006년 10월 23일
† 주저자: jude@etri.re.kr
‡ 교신저자: jude@etri.re.kr

고 있다. 그 이유는 핸드폰, RFID와 같은 휴대용 기기의 사용이 증가하면서 이에 대한 보안의 중요성이 강조되고 있으나, 기존의 고속 연산을 위해 구현된 암호 회로는 휴대용 기기에 적용하기에는 면적이나 전력 소모 측면에서 적합하지 않기 때문이다. 따라서 이에 적용하기 위한 경량 암호 구현이 중요한 요소로 자리잡고 있다.

ARIA 블록 암호 알고리즘[1]은 국내 표준 블록 알고리즘인 SEED 알고리즘을 대체하기 위해 개발된 알고리즘으로, SPN(Substitution-Permutation Network) 구조를 가진 128-비트 블록암호 알고리즘이며 속도와 안전성 측면에서 AES(Advanced Encryption Standard) 알고리즘[2]과 유사하다는 평가를 받고 있다. 현재 KS 표준 블록 암호 알고리즘으로 제정되어 있어 광범위하게 사용될 것으로 예측된다.

이 논문에서는 ARIA 블록 암호 알고리즘을 32-비트 구조로 경량 구현하였다. SBOX와 Diffusion 블록을 경량으로 설계하였고, 보다 효율적인 구조를 채용하여 동부-아남 0.25um 공정에서 11301 게이트의 면적과 128-비트 키를 이용하는 경우 87/278/256 클락 (초기화/암호화/복호화)를 소모한다. 이는 지금까지 알려진 가장 경량화된 구현과 비교하여 면적은 7% 감소, 속도는 13% 향상된 결과이다.

이 논문은 다음과 같이 구성되었다. 2장에서는 ARIA 연산기의 구현에 관해 발표된 기존 연구 결과를 살펴보고, 3장에서는 ARIA 알고리즘의 개요를 설명하며, 4장에서는 제안하는 32비트 구조의 ARIA 연산기에 대해서 상세히 설명한다. 5장에서는 구현된 결과를 기존의 결과와 비교하여 서술하며, 마지막 6장에서 결론을 맺는다.

Ⅱ. 관련분야의 연구

ARIA 알고리즘은 2004년 12월 한국 표준 블록 암호 알고리즘으로 선정되었으며, SPN 구조를 가지는 128-비트 블록 암호 알고리즘이다. 기존의 한국 표준 블록암호 알고리즘인 SEED에 비하여 2배 이상의 효율성을 가지고 있으며, AES 와 유사한 효율성을 가지는 것으로 알려져 있다.[3]

지금까지의 ARIA 알고리즘에 대한 구현은 주로 128-비트 구조를 가지는 고속 연산을 중심으로 이루어져 1 cycle/round 구조를 설계하고 이를 반복적으로 사용하는 방법을 통해 ARIA를 설계하는 방식이 제안되었고,[4] 현재 파이프라인 구조를 이용하여 보다 고속 연산을 수행하도록 하는 방법이 연구되고 있다.

최근에는 ARIA 알고리즘의 경량 구현에 대한 연구가 이루어져 SBOX를 구현하기 위해 메모리를 이용하던 방식을 개선하여 EBGA(Extended Binary GCD Algorithm)를 기초로 $GF(2^8)$ 역함수와 비트의 선형 부울함수를 이용하여 구현하는 방법이[5] 제안되었으며, 128-비트 키만을 사용하며 12501 게이트의 크기를 가지고 32-비트 구조의 ARIA 구현 논문이 발표되었다.[6] 이 논문의 ARIA 연산기는 초기화 연산에 65 클락을 소모하며, 암복호화 시 357 클럭을 소모한다.

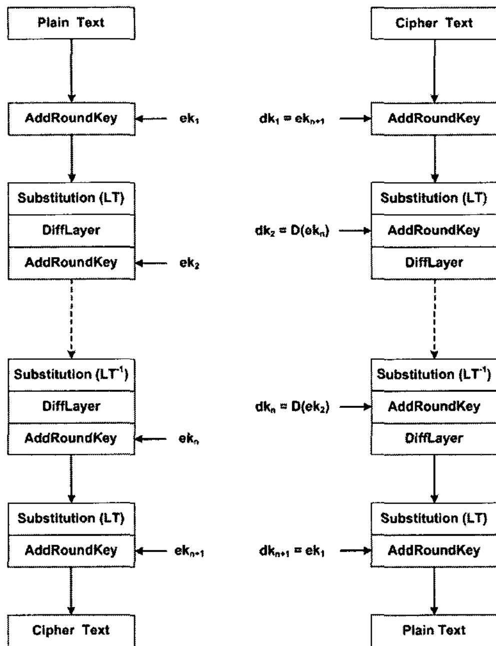
또한 ARIA 알고리즘과 유사한 형태를 지니는 AES의 경우 SBOX 구조 부분에서는 Fermat 정리를 이용한 역원 계산 방법이 제안되었으며,[6] D. Canright[7]가 과거 V. Rijmen[8]과 A. Satoh[9]에 의해 연구되어온 복합체(Composite field) 연산을 이용한 경량화를 더욱 발전시켜, 정규기저(Normal Basis)를 사용하여 $GF(2^8)$의 상의 원소를 $GF(((2^2)^2)^2)$로 매핑하는 방법을 제안하였다.

Ⅲ. ARIA 알고리즘

ARIA 알고리즘은 128-비트의 데이터 블록을 처리하는 알고리즘으로 128/192/256-비트 암호키를 사용한다. AES와 유사한 정도의 암호학적 안전성을 가지고 있으며, 현재 KS 표준 암호 알고리즘으로 제정되어 있어 향후 광범위하게 사용될 것으로 예상되고 있다. ARIA 알고리즘은 키 길이에 따라 다른 라운드 수를 가지게 되며, 각 라운드 함수는 다음과 같은 세 부분으로 구성되어 있다.

- 라운드 키 덧셈(AddRoundKey) : 128-비트 라운드 키를 입력 128-비트와 XOR
- 치환 계층(SubstLayer) : 두 유형의 치환계층을 가지며, 각각은 2종의 SBOX와 그 역함수로 구성
- 확산 계층(DiffLayer) : 16x16 involutional 이진 행렬을 사용한 바이트간의 확산함수

전체적인 ARIA 알고리즘의 암호화 및 복호화 과정은 그림 1과 같다.



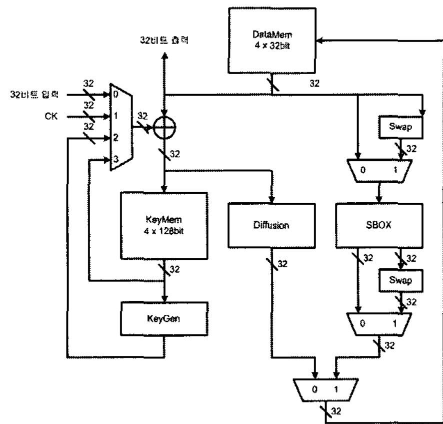
〔그림 1〕 ARIA 알고리즘의 암호화 및 복호화 과정

ARIA의 암호화 과정은 첫 라운드와 마지막 라운드를 제외하면 모두 동일한 형태를 가지고 있으며, 홀수와 짝수의 라운드에 각기 다른 치환 계층을 사용하게 된다. 복호화의 경우 암호화의 역 과정으로 이루어져 있으며, 확산 계층과 라운드 키 덧셈의 위치가 바뀌어 있기 때문에 확산 함수를 통과한 암호화 라운드 키를 복호화 라운드 키로 사용하게 된다.

Ⅳ. 32-비트 구조 ARIA 연산기의 아키텍쳐

ARIA 알고리즘은 연산 순서의 변경이 불가능하며, 두 개의 치환 계층과 128-비트 전체를 사용하는 확산 함수로 인해서 AES 알고리즘에 비해 경량 구현에 불리하다. ARIA의 알고리즘의 경량 구현으로는 8, 16, 32-비트 구조를 생각해 볼 수 있는데, 치환 계층이 32-비트 단위로 이루어지므로 32-비트 구조를 이용하여 ARIA 알고리즘을 구현하였다.

그림 2는 제안하는 32-비트 ARIA 연산기의 구조를 보여주고 있다. ARIA 연산기는 4개의 128-비트 키를 저장하는 키 메모리(KeyMem)와 연산 중간 결과를 저장하기 위한 4개의 32-비트의 데이터 메모리(DataMem), 확산 연산을 수행하는 Diffusion 블록과 치환 연산을 수행하는 SBOX 블록, 그리고 라운드 키를 생성하는 KeyGen 블록으로 구성된다.



〔그림 2〕 제안하는 32-비트 ARIA 연산기

ARIA 연산기를 동작시키기 위해서는 32-비트 입력을 통해 키와 데이터를 전달 받아야 한다. ARIA 연산기는 128/192/256-비트 암호키를 지원하므로, 어떤 크기의 키를 사용하는 지를 알려주기 위한 키 길이의 설정과 암호화 및 복호화 연산중 어느 연산을 수행할 것인지를 알려주는 연산 설정, 그리고 암호키를 동시에 입력받아 암호키를 키 메모리에 저장한다. 암호키 입력 후 Diffusion 블록과 SBOX 블록을 이용하여 4개의 128-비트 초기 라운드 키를 생성하고 이를 다시 키 메모리에 저장하는 것으로 암/복호화 준비를 완료한다. 초기 라운드 키 생성과정은 키 길이와 무관하게 동일하며 유일한 차이는 XOR 과정에서 필요한 상수값 뿐이므로 키 길이 설정을 참조하여 필요한 상수값을 사용한다.

암/복호화를 위한 128-비트 데이터를 32-비트 입력을 통해 데이터 메모리에 저장한 후 암/복호화 연산을 수행하게 된다. 암호화 및 복호화 과정에서는 각 라운드 연산을 라운드 키 덧셈과 치환 계층, 확산 계층 연산으로 분리하여 수행한다. 라운드 키 덧셈을 위해서는 키 메모리에 저장된 키를 KeyGen 블록을 이용하여 라운드 키를 생성하고 데이터 메모리에 저장된 데이터와 XOR 연산을 통해 결과를 얻게 된다. 이 결과는 Diffusion 블록을 통과하도록 설정하여, 라운드 키 덧셈과 Diffusion 블록의 첫 연산을 동시에 이루어지도록 구성하였다. 확산 계층의 연산은 데이터 메모리에 저장된 데이터를 16번 반복하여 Diffusion 블록을 통과함으로써 이루어진다. 데이터 메모리의 데이터가 Diffusion 블록으로 변경없이 전송하기 위하여 '0'을 출력하고 있는 KeyGen 블록의 결과를 MUX에서 선택하도록 하여 결과적으로

중간에 위치한 XOR 연산의 한 쪽 입력이 '0'이 되도록 설정한다. 라운드 연산의 횟수는 키 길이에 따라 달라지며, 몇 번째 라운드인지 여부에 따라 KeyGen에 사용되는 연산이 다르기는 하지만 각 블록이 이를 적절히 처리하도록 구성하였으므로 전체적인 암/복화화 과정은 라운드 연산의 반복이 된다.

1. 메모리 및 라운드 키 생성

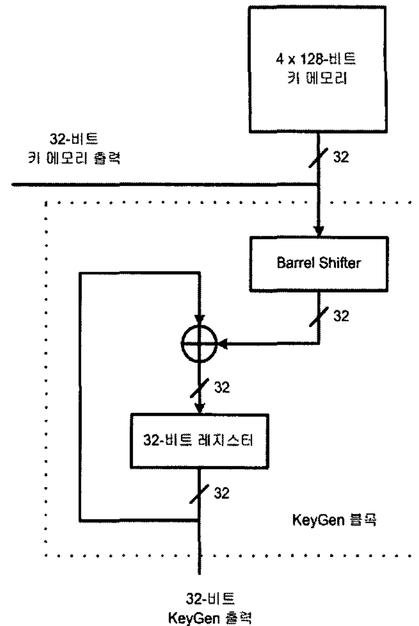
데이터 메모리는 라운드 키 덧셈이나 확산 및 치환을 통해 얻어진 중간 결과를 저장하기 위해 사용되며, 키 메모리는 마스터 키로부터 초기화를 통해 얻어진 4개의 128-비트의 초기 라운드 키를 저장하기 위해 사용된다. 데이터 메모리와 키 메모리는 듀얼 포트 메모리로 이루어져 읽는 주소와 쓰는 주소를 다르게 지정할 수 있도록 구성되었다.

4개의 128-비트 키에서 라운드 키 생성하는 KeyGen 블록은 기존에 알려진 4 클락에 32-비트 라운드 키를 생성하는 구조를 이용하였다.[6]

KeyGen 블록은 그림 3과 같은 구조를 가지고 있으며 주어진 32-비트 키를 쉬프트 시키기 위한 Barrel Shifter와 XOR 연산기, 그리고 중간값을 저장하기 위한 32-비트 레지스터로 구성된다.

32-비트 라운드 키를 생성하기 위해서는 두 개의 32-비트 초기 라운드 키를 필요로 하며, 그 중 하나의 키는 비트 쉬프트 된 값을 요구하므로, 3번의 클락을 소모하여 32-비트 라운드 키를 생성하게 된다. 일례로 ARIA 암호화의 1 라운드 키는 $W_0 \oplus W_1 \ggg 19$ 연산을 통해 얻어지는 데, 첫 32-비트를 얻기 위해서는 $W_{0,0}$, $W_{1,0}$, $W_{1,3}$ 을 순차적으로 받아서 Barrel Shifter를 이용하여 $W_{0,0}$, $W_{1,0} \gg 19$, $W_{1,3} \ll 12$ 로 변환하고 이들을 XOR 연산을 이용하여 계산한다. 이 과정이 3 클락을 필요로 하게 되며, 레지스터를 초기화 하는데 1 클락을 추가적으로 소모하게 되므로 결과적으로 32-비트 라운드 키를 얻기 위해서 4 클락을 필요로 하게 된다.

128/192/256-비트 키 길이의 차이에 따른 연산의 변화를 생각해보면 암호키에서 초기 라운드키를 생성하는 과정에서는 초기 암호키 입력 시간과 CK 상수값의 차이만 존재할 뿐 나머지 과정은 동일하며, 암호화와 복호화 과정에서는 라운드 수 차이만 존재할 뿐 KeyGen 블록의 사용은 동일하다. 따라서 동일한 KeyGen 블록을 이용하여 128/192/256-비트 키 연산을 처리할 수 있다.



〔그림 3〕 KeyGen 블록의 구조

2. 치환 계층

ARIA의 치환계층은 32-비트 입력에 대해서 4 종류의 8-비트 SBOX를 이용해 치환을 수행하도록 구성되어 있으며, 이들은 유한체 $GF(2^8)$상의 함수 x^{-1}과 x^{247} 에 아핀(Affine) 변환을 취한 형태로 이루어져 있다. 이를 하드웨어로 구현하는 경우 표 참조(Table Lookup) 방식이 일반적으로 이용되고 있으나, 차지하는 면적이 큰 관계로 이를 줄이기 위한 연구가 계속되고 있다. 면적을 줄이기 위한 방법으로 x^{-1}을 테이블이 아닌 조합논리 회로(Combinational logic)로 설계하는 방법이 있으며, 현재까지 가장 효율적으로 x^{-1}을 구현한 사례는 D. Canright가 제안한 정규기저를 사용하여 $GF(2^8)$의 상의 원소를 $GF(((2^2)^2)^2)$로 매핑하는 방법[7]이므로, 이를 이용하여 x^{-1} 연산기를 구현하였다. 구현된 x^{-1} 연산기를 이용하면, S_1 SBOX는 x^{-1}만을 사용하는 형태를 취하고 있으므로 행렬 연산을 통해 쉽게 정리할 수 있다.

S_2 SBOX의 경우 x^{247}을 사용하고 있는데, 이는 x^8 연산을 나타내는 8x8 이진 행렬 F를 이용하여 x^{-1}로 변환할 수 있다. 마찬가지로 S_2 SBOX의 역연산도 x^{32} 연산을 처리하는 8x8 이진 행렬 G를 통해 x로 변환할 수 있다.[5] 이 두 이진 행렬을 사용하여 식을 정리하면 다음과 같다.

$$S_1(x) = B \cdot x^{-1} \oplus b$$

$$= \begin{bmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{bmatrix} \cdot (x^{-1}) \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad (1)$$

$$S_1^{-1}(x) = (B^{-1} \cdot (x \oplus b))^{-1}$$
$$= (B^{-1} \cdot x \oplus B^{-1} \cdot b)^{-1}$$

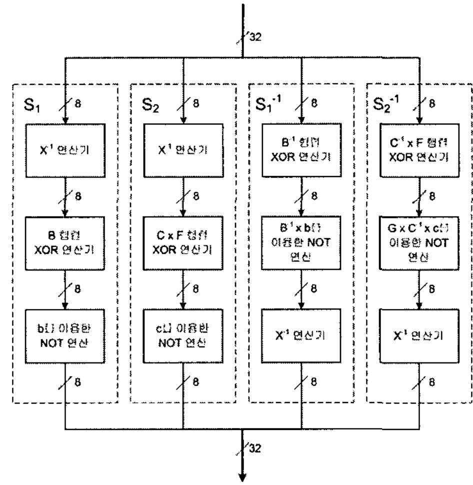
$$= \left(\begin{bmatrix} 00100101 \\ 10010010 \\ 01001001 \\ 10100100 \\ 01010010 \\ 00101001 \\ 10010100 \\ 01001010 \end{bmatrix} \cdot x \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)^{-1} \qquad (2)$$

$$S_2(x) = C \cdot x^{247} \oplus c$$
$$= C \cdot x^{-8} \oplus c$$
$$= C \cdot (x^{-1})^8 \oplus c$$
$$= C \cdot F \cdot (x^{-1}) \oplus c$$

$$= \begin{bmatrix} 01010111 \\ 00111111 \\ 11101101 \\ 11000011 \\ 01000011 \\ 11001110 \\ 01100011 \\ 11110110 \end{bmatrix} \cdot (x^{-1}) \oplus \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad (3)$$

$$(S_2^{-1}(x))^{-8} = (C^{-1} \cdot (y \oplus c))$$
$$S_2^{-1}(x) = (S_2^{-1}(x))^{-8 \cdot -32} = (C^{-1} \cdot (y \oplus c))^{-32}$$
$$= ((C^{-1} \cdot (y \oplus c))^{32})^{-1}$$
$$= (G \cdot (C^{-1} \cdot (y \oplus c))^{-1}$$
$$= (G \cdot C^{-1} \cdot y \oplus G \cdot C^{-1} \cdot c)^{-1}$$

$$= \left(\begin{bmatrix} 00011000 \\ 00100110 \\ 00001010 \\ 11100011 \\ 11101100 \\ 01101011 \\ 10111101 \\ 10010011 \end{bmatrix} \cdot y \oplus \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right)^{-1} \qquad (4)$$



〔그림 4〕 SBOX 블록의 구조

이렇게 구성된 SBOX 블록은 그림 4와 같고, 수식에서 보이는 XOR 연산중에서 하나의 열을 사용하는 부분은 면적 감소를 위해 NOT 게이트 연산으로 처리하였다.

치환 계층에 사용하는 SBOX의 적용 순서는 홀수 라운드의 경우 $\{S_1, S_2, S_1^{-1}, S_2^{-1}\}$로, 짝수 라운드는 $\{S_1^{-1}, S_2^{-1}, S_1, S_2\}$로 적용하고 있어 서로 다른 형태를 가지고 있으나, 상하위 16-비트의 순서를 바꿔주는 것으로 하나의 32-비트 SBOX 블록만을 사용하여 구현이 가능하다. 그림 2의 SWAP 블록이 순서를 바꿔주는 기능을 수행하게 된다.

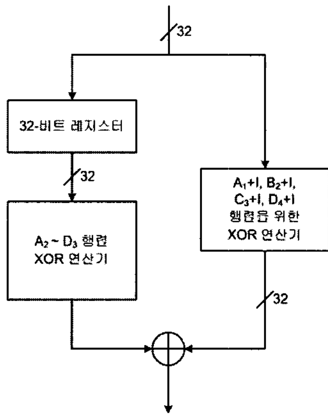
3. 라운드 키 확장

ARIA의 확산 계층은 16x16 이진 행렬과 바이트 입력의 곱셈으로 이루어지므로, 128-비트 데이터 전체가 영향을 받게 된다. 이는 32-비트 단위로 데이터에 영향을 주는 AES 알고리즘의 MixColumn 연산에 비해 경량 하드웨어 설계에 어려움을 안겨주고 있다. 지금까지 32-비트 구조의 확산 계층 구현은 16x16 이진행렬을 4x16 이진행렬 4개로 분리한 후, 32-비트 데이터를 이용하여 연산을 수행하고 얻어진 중간 결과를 추가된 128-비트 레지스터에 저장하는 방식을 사용하고 있다.[6] 이러한 방식은 구현이 간결하다는 장점이 있지만, 128-비트 레지스터로 인해 면적이 증가하는 단점을 가지게 된다.

이러한 단점을 줄이기 위하여 32-비트 임시 레지스터만을 사용하는 경우, 연산의 중간 결과를 기존의

데이터 메모리에 저장하게 되면, 그 결과가 향후 연산에 영향을 끼치게 되므로 이를 고려한 새로운 이진 행렬을 사용하여 확산 연산을 수행해야 한다. 128-비트 데이터 X가 확산 계층을 통과하여 128-비트의 결과값 Y를 얻는 과정을 살펴보면 식 (5)와 같으며 이를 정리하면 식 (6)을 얻을 수 있다.

이때 $X^{(j)}_i$는 j 번 갱신된 i 번째 32-비트 데이터이며, A_i, B_i, C_i, D_i는 연산에 사용되는 4x4 이진 행렬이고, I는 4x4 단위 행렬이며, Y_i는 i 번째 32-비트 최종 결과값이다.



〔그림 5〕 Diffusion 블록의 구조

$$\begin{bmatrix}Y_1\\Y_2\\Y_3\\Y_4\end{bmatrix}=\begin{bmatrix}D_1\\D_2\\D_3\\D_4\end{bmatrix}\cdot X_4^{(3)}+\begin{bmatrix}X_1^{(3)}\\X_2^{(3)}\\X_3^{(3)}\\X_4^{(3)}\end{bmatrix}\quad \begin{bmatrix}X_1^{(1)}\\X_2^{(1)}\\X_3^{(1)}\\X_4^{(1)}\end{bmatrix}=\begin{bmatrix}A_1\\A_2\\A_3\\A_4\end{bmatrix}\cdot X_1+\begin{bmatrix}X_1\\X_2\\X_3\\X_4\end{bmatrix}$$

$$\begin{bmatrix}X_1^{(2)}\\X_2^{(2)}\\X_3^{(2)}\\X_4^{(2)}\end{bmatrix}=\begin{bmatrix}B_1\\B_2\\B_3\\B_4\end{bmatrix}\cdot X_2^{(1)}+\begin{bmatrix}X_1^{(1)}\\X_2^{(1)}\\X_3^{(1)}\\X_4^{(1)}\end{bmatrix}\quad \begin{bmatrix}X_1^{(3)}\\X_2^{(3)}\\X_3^{(3)}\\X_4^{(3)}\end{bmatrix}=\begin{bmatrix}C_1\\C_2\\C_3\\C_4\end{bmatrix}\cdot X_3^{(2)}+\begin{bmatrix}X_1^{(2)}\\X_2^{(2)}\\X_3^{(2)}\\X_4^{(2)}\end{bmatrix} \tag{5}$$

식 (6)의 결과가 원래의 확산 계층 연산과 동일하므로, 이를 이용하여 A_i, B_i, C_i, D_i를 구할 수 있다. 그리고 A_1, B_2, C_3, D_4에서는 임시 레지스터에 저장된 데이터와 입력 데이터가 동일하므로 이를 하나로 정리해서 사용할 수 있다. 이렇게 얻어진 16개의 4x4 이진 행렬은 식 (7)과 같으며, 이를 이용하여 구성한 Diffusion 연산기는 〔그림 5〕와 같다.

Diffusion 연산기는 32-비트 데이터 입력을 4 클락에 1번 32-비트 레지스터에 저장하고, 이 저장된 데이터와 입력 데이터를 XOR 연산 처리하여 결과를 얻게 된다. 수식 (7)의 16개의 4x4 이진행렬 중에서 A_1+I, B_2+I, C_3+I, D_4+I를 사용하는 부분에서 32-비트 레지스터의 데이터를 갱신하게 되는데, 레지스터에 데이터를 저장하는 데 걸리는 1 클락의 지연을 방지하기 위하여 A_1+I, B_2+I, C_3+I,

$$\begin{aligned}
Y_1 &= D_1X_4+(D_1C_4+C_1)X_3+(D_1B_4+(D_1C_4+C_1)B_3+B_1)X_2\\
&\quad +(D_1A_4+(D_1C_4+C_1)A_3+(D_1B_4+(D_1C_4+C_1)B_3+B_1)A_2+(A_1+I))X_1\\
Y_2 &= D_2X_4+(D_2C_4+C_2)X_3+(D_2B_4+(D_2C_4+C_2)B_3+B_2+I)X_2\\
&\quad +(D_2A_4+(D_2C_4+C_2)A_3+(D_2B_4+(D_2C_4+C_2)B_3+B_2+I)A_2)X_1\\
Y_3 &= D_3X_4+(D_3C_4+C_3+I)X_3+(D_3B_4+(D_3C_4+C_3+I)B_3)X_2\\
&\quad +(D_3A_4+(D_3C_4+C_3+I)A_3+(D_3B_4+(D_3C_4+C_3+I)B_3)A_2)X_1\\
Y_4 &= (D_4+I)X_4+(D_4+I)C_4X_3+((D_4+I)B_4+(D_4+I)C_4B_3)X_2\\
&\quad +((D_4+I)A_4+(D_4+I)C_4A_3+((D_4+I)B_4+(D_4+I)C_4B_3)A_2)X_1
\end{aligned} \tag{6}$$

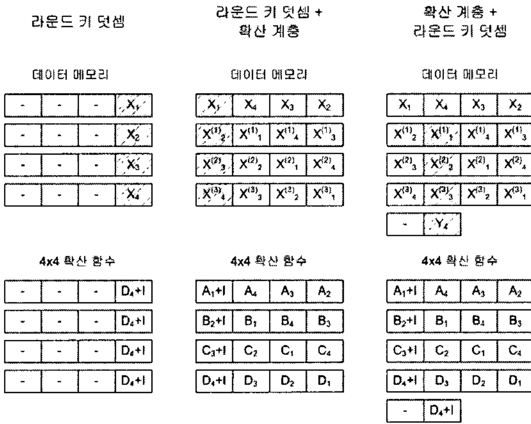
$$A_1+I=\begin{bmatrix}0001\\0010\\0100\\1000\end{bmatrix}\quad B_1=\begin{bmatrix}1010\\0101\\1010\\0101\end{bmatrix}\quad C_1=\begin{bmatrix}1100\\1100\\0011\\0011\end{bmatrix}\quad D_1=\begin{bmatrix}0110\\1001\\1001\\0110\end{bmatrix}$$

$$A_2=\begin{bmatrix}0101\\1010\\0101\\1010\end{bmatrix}\quad B_2+I=\begin{bmatrix}0100\\1000\\0001\\0010\end{bmatrix}\quad C_2=\begin{bmatrix}1001\\0110\\0110\\1001\end{bmatrix}\quad D_2=\begin{bmatrix}0011\\0011\\1100\\1100\end{bmatrix}$$

$$A_3=\begin{bmatrix}0011\\0011\\1100\\1100\end{bmatrix}\quad B_3=\begin{bmatrix}1001\\0110\\0110\\1001\end{bmatrix}\quad C_3+I=\begin{bmatrix}0010\\0001\\1000\\0100\end{bmatrix}\quad D_3=\begin{bmatrix}0101\\1010\\0101\\1010\end{bmatrix}$$

$$A_4=\begin{bmatrix}0110\\1001\\1001\\0110\end{bmatrix}\quad B_4=\begin{bmatrix}1100\\1100\\0011\\0011\end{bmatrix}\quad C_4=\begin{bmatrix}0101\\1010\\0101\\1010\end{bmatrix}\quad D_4+I=\begin{bmatrix}1000\\0100\\0010\\0001\end{bmatrix} \tag{7}$$

D_4+I행렬을 위한 XOR 연산기를 추가하였다.



〔그림 6〕 라운드 키 덧셈과 확산 계층의 연산 과정

이전 32-비트 구조 ARIA 연산기[6]의 확산 계층은 4개의 4x16 확산 함수를 사용하므로 4번의 연산과 4번의 데이터 전송으로 8 클락이 소모되는데 비하여, 위에서 얻어진 16개의 4x4 확산 함수를 사용하는 경우 16번의 연산이 필요하므로 16 클락이 소모된다. 확산 계층에서 소모되는 16 클락이 32-비트 라운드 키 생성에 소모되는 시간과 동일하므로 라운드 키 덧셈 연산과 동시에 확산 계층의 연산을 수행하여 연산 시간을 단축시킬 수 있다. 특히, 마지막 4x4 확산 함수가 입력이 출력으로 그대로 전송될 수 있어서 라운드 키 덧셈이 단독으로 필요한 경우에도 결과 값을 데이터 메모리로 전송할 수 있다. 확산 계층과 라운드 키 덧셈을 한꺼번에 처리하도록 구성하는 경우 연산 과정에 따라 선택된 메모리 데이터와 와 4x4 이진 행렬은 그림 6과 같다.

라운드 키 덧셈만을 수행하는 경우 입력이 그대로 출력으로 나가도록 확산 함수를 선택한다. 4 클락마다 32-비트 라운드 키가 생성되므로, 이를 이용하면 16 클락을 소모하여 128-비트 라운드 키 덧셈이 이루어진다. 복호화 과정에서는 라운드 키 덧셈 후에 확산 연산을 수행하는 데 이를 한꺼번에 수행하기 위해서는 라운드 키 덧셈이 완료된 데이터를 32-비트 임시 레지스터에 저장하는 방법을 사용한다. 그림에서 빗금으로 표시된 부분이 라운드 키 덧셈이 완료된 데이터가 확산 함수의 입력으로 들어오는 부분이다. 16 클락을 소모하여 라운드 키 덧셈과 확산을 동시에 수행하게 된다. 암호화 과정에서는 확산 연산을 수행한 후 라운드 키 덧셈이 이루어진다. 두 연산을 한꺼번에 처리하기 위해서는 확산 연산의 결과가 다른 데이터에 더 이상 영향을 주지 않는 값에 라운드 키 덧셈을 수행해야 하며, 그림과 같은 순서로 데이터를 전송하면 18 클락을 소모하여 처리할 수 있다.

V. 구현결과

제안하는 ARIA 연산기는 Synopsys사의 DC Compiler를 이용하여 동부-아남 반도체의 0.25um standard CMOS cell 공정으로 합성하였다. 합성 결과는 128/192/256-비트 키를 사용하는 암/복호화가 모두 가능하면서도 11301 게이트의 크기를 가진다. 연산 속도의 측면에서 초기화의 경우 한 라운드에 128-비트 XOR 연산 4클락, 치환 연산 4 클락, 확산 연산 후 128-비트 XOR 연산 17 클락으로 이루어져 모두 25 클락을 소모하게 된다. 암호화의 한 라운드는 4 클락의 치환 연산과 18 클락의 확산 연산 후 라운드 키 덧셈으로 이루어져 22 클락이 소모되며, 복호화의 한 라운드는 4 클락의 치환 연산과 16 클락의 라운드 키 덧셈 후 치환 연산으로 20 클락이 소모된다.

```
1  ****************************************
2  Report : reference
3  Design : ARIA
4  Version: X-2005.09-SP3
5  Date   : Fri Jun  9 15:08:49 2006
6  ****************************************
7
8  Reference          Library     Unit Area   Count    Total Area   Attributes
9  -----------------------------------------------------------------------------
10 DataMem                       1028.099976      1   1028.099976   d, h, n
11 KeyMem                        4124.100098      1   4124.100098   d, h, n
12 Diff                          1929.000000      1   1929.000000   d, h, n
13 SBOX                          1684.500122      1   1684.500122   d, h
14 AddKey                         358.799988      1    358.799988   d, h
15 KeyGen                         561.400024      1    561.400024   d, h, n
16 Control                       1606.699829      1   1606.699829   d, h, n
17 AIN01D1            anam          0.700000      4      2.800000
18 AIN01D4            anam          1.200000      1      1.200000
19 AIN01D7            anam          1.500000      1      1.500000
20 ANI01OA            anam          2.700000      1      2.700000
21 -----------------------------------------------------------------------------
22
23 Total 11 references                                 11300.799805
```

〔그림 7〕 제안하는 ARIA 연산기의 합성 결과 (동부-아남 0.25um 공정)

〔표 1〕 제안하는 ARIA 연산기의 특징

암호 알고리즘	ARIA
키 길이	128/192/256-비트
입/출력 길이	32-비트
키 확장 방식	on-the-fly
게이트 수	11301
최대 주파수	467 MHz
라운드당 평균 클락 수	22 클락(암호화) 20 클락(복호화)

기존에 발표되었던 32-비트 구조의 ARIA 연산기를 동일한 공정으로 구현하여 크기의 비교 결과를 표에 정리하였다. 제안된 연산기는 기존 ARIA 연산기보다 작은 확산 계층과 치환 계층을 가지게 되며, 128/192/256-비트 키에 대한 암복호화 지원으로 인해 컨트롤 블록의 크기가 증가하였음을 알 수 있다. 이 결과 기존의 ARIA 연산기에 비해 7% 정도 줄어든 11301 게이트의 크기를 소모하며 이는 지금까지 알려진 ARIA 연산기 중에서 가장 작은 크기이다.

〔표 2〕 ARIA 연산기의 게이트 카운트

구성 요소	기존의 ARIA 연산기[6]	제안하는 ARIA 연산기
DataMem	1028	1028
KeyMem	4124	4124
Diff	2233	1929
SBOX	2733	1685
AddKey	359	359
KeyGen	538	561
Control	1136	1606
Etc	0	8
Total	12154	11301

연산기의 정확성 및 수행 시간에 대한 검증을 위해 ARIA 표준문서에 포함된 테스트 벡터를 입력으로 하여, Cadence 사의 NCverilog 시뮬레이터를 이용하였다. 〔그림 8〕은 1MHz 클락을 기준으로 128-비트 키를 이용한 암호화의 시뮬레이션 결과이며, 〔그림 9〕는 1MHz 클락을 기준으로 256-비트 키를 이용한 복호화의 시뮬레이션 결과이다.

그림에서 din은 32-비트 입력을, dout은 32-비트 출력을 나타내며, nrd는 데이터 읽기 신호, nwr은 데이터 쓰기 신호이다. 제안하는 ARIA 연산기는 128/192/256-비트 키를 선택적으로 적용할 수 있는데, 이는 mode 입력을 선택하여 적용한다. 〔그림 8〕은 128-비트 암호화 연산이므로 mode를 '0' 으로 설정하였고, 〔그림 9〕는 256-비트 암호화 연산이므로 mode를 '7' 로 설정하였다. 연산기의 동작 상태를 알려주는 신호로 init_done은 초기화의 완료 여부를 알려주며, out_valid는 암/복호화 연산이 완료되었음을 알려준다. 시뮬레이션의 결과값과 테스트 벡터의 결과값이 일치하는 지 여부는 Test 신호를 통해 확인하였다.

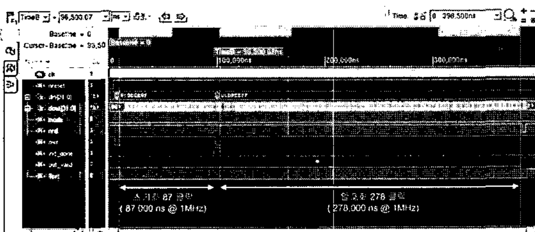
기존의 128-비트 키만을 지원하는 32-비트 구조의 ARIA 연산기가 복호화 시 348 클락을 소모한 것과 비교하여 26% 정도의 속도 상승이 있으며, 초기화를 포함한 암호화의 경우에도 7%의 속도 상승이 있음을 알 수 있다.

VI. 결 론

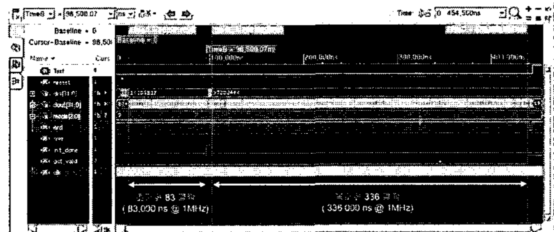
〔표 3〕 ARIA 연산기의 수행 시간 (클락 사이클)

	키 길이	초기화	암호화	복호화
기존 ARIA 연산기[6]	128-비트	64	348	348
제안하는 ARIA 연산기	128-비트	87	278	256
	192-비트	85	318	296
	256-비트	83	358	336

제안하는 32-비트 구조의 ARIA 연산기는 128/192/256-비트 키 모두에 동작이 가능하며, 11301 게이트 카운트의 크기를 가지고 있다. 라운드 키 덧셈과 확산 계층의 연산을 동시에 수행할 수 있는 구조로 인해 128-비트 키의 암호화에 278 클락을 소



〔그림 8〕 128-비트 암호화 연산의 시뮬레이션 출력 파형



〔그림 9〕 256-비트 복호화 연산의 시뮬레이션 출력 파형

모하게 된다. 이는 기존에 발표된 128-비트 키만을 사용하는 경량 ARIA 연산기에 비해 면적에서 7% 이상, 속도에서 13% 이상 개선된 수치이다.

참 고 문 헌

[1] ARIA Algorithm Specification, May. 2004, available at: http://www.nsri.re.kr/ARIA/doc/ARIA-specification.pdf

[2] FIPS Pub. 197: Specification for the AES, Nov. 2001, available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[3] Security and Performance Analysis of ARIA, Jan. 2003, available at: http://www.nsri.re.kr/ARIA/doc/ARIA-COSICreport.pdf

[4] 박진섭, 윤연상, 김용대, 양상운, 장태주, 유영갑, "ARIA 암호 알고리듬의 하드웨어 설계 및 구현," *전자공학회논문지*, vol. 42-SD, no. 4, pp. 26-36, Apr. 2004.

[5] 전기원, 전은아, 지재덕, 박익수, 정석원, 서재현, 오병균, "유비쿼터스 환경에 적합한 블록암호 ARIA의 S-Box FPGA 구현," *CISC 2005*, pp. 621-625, Jun. 2005.

[6] Jinsup Park, Yong Dae Kim and Younggap You, "A Scale Down Structure of ARIA Processor," *Journal of the Research Institute for Computer and Information Communication*, pp. 79-84, Dec. 2005.

[7] D. Canright, "A Very Compact S-Box for AES," *CHES'05*, LNCS 3659, pp. 441-455, Springer-Verlag, 2005.

[8] Vincent Rijmen, "Efficient implementation of the Rijndael S-box," available at: http://www.esat.kuleuven.ac.be/ ~rijmen/rijndael/sbox.pdf, 2001.

[9] A. Satoh, S. Morioka, K. Takano, and Seiji Munetoh, "A compact Rijndael hardware architecture with S-box optimization," *Advances in Cryptology - ASIACRYPT 2001*, LNCS 2248, pp. 239-254, Springer-Verlag, 2001.

[10] 안하기, 신경욱, "AES Rijndael 블록 암호 알고리듬의 효율적인 하드웨어 구현," *정보보호학회 논문지*, 제 12권, 제2호, pp. 57-67, 2002.

[11] 조용국, 송정환, 강성우, "AES(Advanced Encryption Standard) 안전성 평가에 대한 고찰," *정보보호학회 논문지*, 제11권, 제6호, pp. 67-76, 2001.

[12] 최병윤, 박영수, 전성익, "모듈화된 라운드 키 생성회로를 갖는 AES 암호 프로세서의 설계," *정보보호학회 논문지*, 제12권, 제5호, pp. 15-25, 2002.

[13] 서정갑, 김창균, 하재철, 문상재, 박일환, "블럭 암호 ARIA에 대한 차분전력분석공격," *정보보호학회 논문지*, 제15권, 제1호, pp. 99-107, 2005.

〈著者紹介〉

유 권 호 (Gwonho Ryu) 정회원
1999년 2월: 포항공과대학교 전자공학과 학사
2001년 2월: 포항공과대학교 전자공학과 석사
2002년 9월～현재: 국가보안기술연구소 연구원
<관심분야> 암호칩 설계, 블록 암호, 부채널 공격

구 본 석 (Bonseok Koo) 정회원
1998년 2월: 경북대학교 전자공학과 학사
2000년 2월: 포항공과대학교 전자공학과 석사
2000년 3월~2000년 9월: LG 정보통신 중앙 연구소 연구원
2000년 10월～현재: 국가보안기술연구소 선임연구원
<관심분야> 암호칩 설계, 공개키 암호, 부채널 공격

양 상 운 (Sangwoon Yang) 정회원
1992년 2월: 충북대학교 정보통신공학과 학사
1998년 2월: 충북대학교 정보통신공학과 석사
1992년~2000년: 국방과학연구소 연구원
2000년～현재: 국가보안기술연구소 선임연구원
<관심분야> 암호칩 설계, Computer Arithmetic, 정보보호, 반도체

장 태 주 (Taejoo Chang) 정회원
1982년 2월: 울산대학교 전기공학과 학사
1990년: 한국과학기술원 전기 및 전자공학과 석사
1998년: 한국과학기술원 전기 및 전자공학과 공학박사
1982년~2000년: 국방과학연구소 선임연구원
2000년~현재: 국가보안기술연구소 책임연구원
<관심분야> 암호칩 설계, 정보보호, 통계학적 신호처리