ARIA 블록 암호 알고리즘 구현

한재수*, 최진구**
*(주)데이터스트림즈
**한국산업기술대학교 컴퓨터공학과
e-mail:hjsdy@nate.com,jkchey@kpu.ac.kr

Implementation of ARIA Block Encryption Algorithm

Jae-Su Han*, Jin-Ku Choi**
*DataStreams Corp.
**Dept of Computer Engineering, Korea-Polytechnic University

요 약

최근 개인 프라이버시 보호에 대한 중요성이 제기되면서, 보안에 대한 관심이 증가하게 되었다. 본 논문에서는 한국 산업규격 KS 표준 블록 암호 알고리즘인 ARIA의 핵심논리를 유지하면서, 동종 경쟁 알고리즘(AES, Camellia 등)과의 차별성으로 강조되어온 16 x 16 이진 행렬을 이용한 확산 계층을 (4 x 4) x 4의 이진 행렬 형태로 수정한 개선 ARIA를 구현하였다. 개선 설계된 ARIA를 검증하기 위해, 파일 암·복호화 시스템을 적용하였고, 보안 영상 시스템을 개발하였다. 기존의 ARIA의 장점을 유지하기 때문에, 초경량 환경이나 많은 데이터를 초고속으로 처리에 필요한 응용에 더 효과적으로 적용될 수 있다.

1. 서론

최근 사회전반의 인터넷사용의 보편화와 스마트폰의 보급으로 인해 지속적으로 제기되어 온 산업 핵심기술 관리와 개인 프라이버시 보호에 대한 중요성이 다시 대두되고 있다. 국가보안기술연구소에서는 정보 통신 서비스의 다변화 및 전자 정부 구현 등으로 인한 국가기관과 민간과의 안정성과 효율성을 제공할 수 있는 소통자료로 ARIA 알고리즘을 이미 2004년 제안하여, 민간에서도 ARIA 알고리즘을 적용한 응용 환경적용 연구가 이루어질 수 있는 환경을 제공하였다. 하지만, 2000년 128비트 미국 표준 블록 암호 알고리즘으로 ARIA의 동급 경쟁 기술인 AES은 하드웨어와 소프트웨어 설계에 관한 많은 연구가 이루어져, 실제 운용 환경에서 알고리즘의 성능을 제시하고, 그에 맞게 개선·적용되는 효과를 거두고 있는데 반해, ARIA 알고리즘의 경우 실용화 연구가 아직은 활발히 이루어지지 못하고 있어, 응용 환경의 실용화 연구와 최적의 구조를 찾는 알고리즘 개선이 더디게 이루어지고 있는 현실이다.

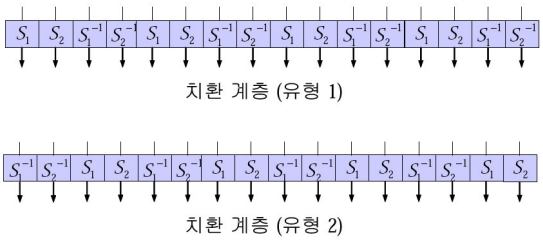
본 논문은 국가보안연구소에서 제안한 ARIA 블록 암호 알고리즘에 대한 분석을 통해 개선점을 도출하여, 암·복호화 처리속도 및 사용되는 프로그램에 대한 메모리 효율성의 개선을 목표로 한다. 논문의 구성은 2장에서는 관련 사항 연구로, ARIA 알고리즘의 소개와, 적용하여 구현되어지는 어플리케이션의 개발환경을 소개 한다. 3장에서는 ARIA 알고리즘에 대한 개선점 제안과 파일 암·복호화 시스템을 이용한 성능결과를 분석한다. 4장에서는 개선 설계된 ARIA 알고리즘을 적용한 보안 영상 시스템을 구현을 소개 한다. 마지막으로 5장에서는 이를 토대로 결론을 맺는다.

2. 관련연구

2.1 ARIA 블록 암호 알고리즘[1]

ARIA 알고리즘은 암호화와 복호화를 수행하는 라운드 함수와 키 스케줄러로 구성되어 있다. ARIA 라운드 함수의 기본 구조는 Involution SPN 구조이다. 입, 출력의 크기는 128비트이고, 키의 크기는 128비트, 192비트 그리고 256비트의 3종류 키를 사용해 암·복호화를 하며, 키의 크기에 따라 가변라운드(12, 14, 16)가 적용된다.

내부 함수는 세단계로 라운드 키 삽입, 치환계층, 확산계층으로 이루어져 있다. 아래 (그림 1)와 같이 치환계층은 8비트 입/출력을 가지는 S-box S_1 과 S_2 와 각각의 그의 역으로 구성되어 있다. 치환계층은 테이블 참조로 구현하는 것이 일반적이고, 최적의 선택이라 할 수 있다.



(그림 1) ARIA의 S-box 계층

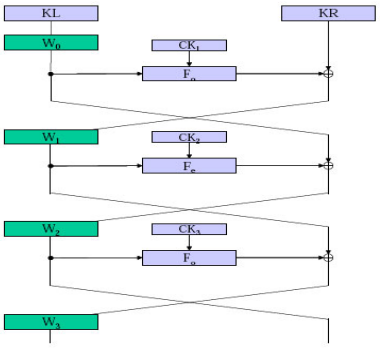
전체 구조가 대합 형태를 갖추기 위해서, 치환계층의 (유형1)은 홀수, (유형2)는 짝수라운드에 사용된다.

확산계층(Diffusion layer)은 ARIA 알고리즘을 다른 동종 블록암호 알고리즘(AES, Camellia 등)과 구별 짓는 주요부분으로 간단한 16x16 대합이진행렬을 사용한다. 확산함수의 사용은 16바이트의 입력에 대해서, 바이트 단위의 행렬 곱을 수행한 결과의 16바이트를 출력 값으로 한다. 아래 (그림 2)는 확산계층을 나타낸 것 이다.

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

(그림 2) ARIA의 16x16 확산 계층

라운드 키 삽입 단계부분은 키 스케줄에서 생성된 라운드 키와 라운드 함수 입력간의 XOR연산으로 이루어져 있다. 키 스케줄은 (그림 3)과 같이 암호화와 사용되는 라운드 함수를 응용한 3 라운드 256비트 Feistel 구조를 이용하여, 4개의 마스터 키(MK)로 128비트 초기 값 4개를 생성한다. 생성된 초기 마스터 키(MK)를 이용하여, 라운드 키를 생성하게 된다.



(그림 3) ARIA의 키 초기화 과정

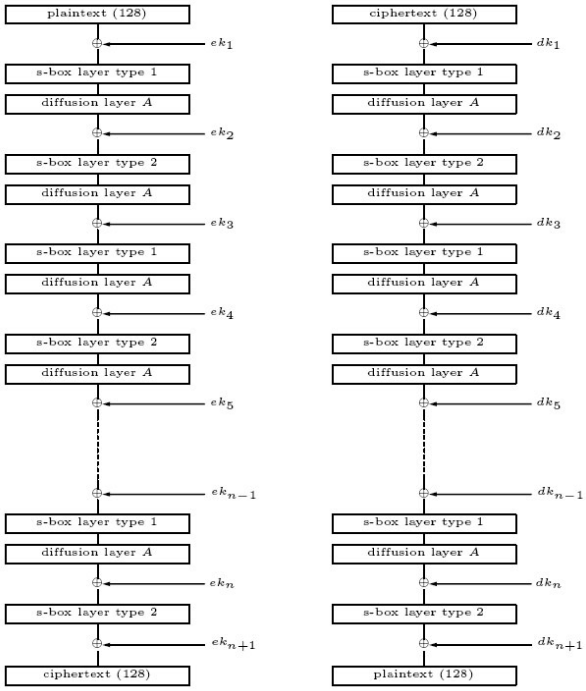
각 암호화 라운드 키는 위에서 생성된 마스터 키(MK)를 이용하여, (그림 4)와 같은 방법으로 생성된다.

$$\begin{aligned} ek_1 &= (W_0) \oplus (W_1^{\gg 19}), & ek_2 &= (W_1) \oplus (W_2^{\gg 19}), \\ ek_3 &= (W_2) \oplus (W_3^{\gg 19}), & ek_4 &= (W_0^{\gg 19}) \oplus (W_3), \\ ek_5 &= (W_0) \oplus (W_1^{\gg 31}), & ek_6 &= (W_1) \oplus (W_2^{\gg 31}), \\ ek_7 &= (W_2) \oplus (W_3^{\gg 31}), & ek_8 &= (W_0^{\gg 31}) \oplus (W_3), \\ ek_9 &= (W_0) \oplus (W_1^{\ll 61}), & ek_{10} &= (W_1) \oplus (W_2^{\ll 61}), \\ ek_{11} &= (W_2) \oplus (W_3^{\ll 61}), & ek_{12} &= (W_0^{\ll 61}) \oplus (W_3), \\ ek_{13} &= (W_0) \oplus (W_1^{\ll 31}), & ek_{14} &= (W_1) \oplus (W_2^{\ll 31}), \\ ek_{15} &= (W_2) \oplus (W_3^{\ll 31}), & ek_{16} &= (W_0^{\ll 31}) \oplus (W_3), \\ ek_{17} &= (W_0) \oplus (W_1^{\ll 19}) \end{aligned}$$

(그림 4) ARIA 라운드 키 생성식

복호화 라운드 키는 암호화 라운드 키와 다르며, 암호화 라운드 키로부터 유도된다. 암호화 라운드 키의 순서를 바꾸고, 첫 번째와 마지막 라운드 키를 제외한 암호화키에 대한 확산함수와의 결과가 복호화 라운드의 키가 된다.

이렇게 생성된 각 라운드의 키에 대한 암·복호화 과정은 (그림 5)와 같이 최종적으로 진행된다[2].



(그림 5) ARIA의 암·복호화 과정

ARIA 알고리즘은 차분 공격과 선형 공격과 같은 공격방법 등에 비교적 안전하다. 2003년 벨기에 루벤 대학에서의 성능평가에서 우리나라의 SEED나 일본의 암호화 알고리즘인 Camellia 보다 데이터 처리속도 등 성능 면에서 2배가량 앞선 것으로 나타났다[3].

2.2 개발환경

본 논문에서 제시되어질 ARIA 알고리즘의 구현과 파일 암·복호화 시스템, 보안 영상 시스템은 Java SE 1.6.0_18을 이용하여 구현하였으며, 보안 영상 시스템은 JMF와 My SQL 5.1을 추가적으로 이용하여 구현하였다.

JMF(Java Media Framework)는 다양한 환경의 플랫폼에서 구동 가능한 멀티미디어 애플리케이션을 개발하려는 목적 하에 개발되어진 툴로, Time-Based Media의 전송과 Processing, 획득을 위한 메시지 프로토콜과 규격화된 구조를 가진다. JMF는 대부분의 표준 미디어 타입을 지원하고, 객체에 대한 클래스화로 언제든지 사용할 수 있는 특징을 가지고 있기 때문에, 영상 전송프로그램에 적합하다고 판단되어, 채택하게 되었다[4].

3. 제안된 ARIA 구조와 성능 평가

3.1 제안된 ARIA

ARIA 알고리즘의 16 x 16 이진행렬을 이용한 확산계층(그림 2 참조)은 ARIA를 다른 동종 블록 암호 알고리즘 (AES, Camellia 등)과 차별되는 부분으로, 암·복호화를 실행하는 라운드에서 마지막 라운드를 제외하고 사용하는 ARIA의 핵심연산부분이다. 본 논문에서 제안하는 개선방향은 고정적으로 잡혀져 있는 행렬의 크기를 줄이면, 알고리즘이 가지고 있는 고정메모리 공간을 줄일 수 있게 된다는 제안에서 시작된다. (그림 6)은 16 x 16 이진행렬에서 행렬을 줄일 수 있는 연산의 규칙을 찾아 변경한 것이다.

$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y6 \\ y7 \\ y4 \\ y5 \\ y9 \\ y8 \\ y11 \\ y10 \\ y15 \\ y14 \\ y13 \\ y12 \end{bmatrix} = \begin{bmatrix} 0001\ 1010\ 1100\ 0110 \\ 0010\ 0101\ 1100\ 1001 \\ 0100\ 1010\ 0011\ 1001 \\ 1000\ 0101\ 0011\ 0110 \\ 1010\ 0001\ 0110\ 1100 \\ 0101\ 0010\ 1001\ 1100 \\ 1010\ 0100\ 1001\ 0011 \\ 0101\ 1000\ 0110\ 0011 \\ 1100\ 0110\ 0001\ 1010 \\ 1100\ 1001\ 0010\ 0101 \\ 0011\ 1001\ 0100\ 1010 \\ 0011\ 0110\ 1000\ 0101 \\ 0110\ 1100\ 1010\ 0001 \\ 1001\ 1100\ 0101\ 0010 \\ 1001\ 0011\ 1010\ 0100 \\ 0110\ 0011\ 0101\ 1000 \end{bmatrix} \cdot \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \\ x9 \\ x10 \\ x11 \\ x12 \\ x13 \\ x14 \\ x15 \end{bmatrix}$$

(그림 6) 변경한 16 x 16 확산 계층

(그림 2)의 확산계층과 비교해 보았을 때, 각 행(16바이트)과 각 행에 맞는 출력 값의 위치를 변경할 때에, 4행 간격으로 규칙이 나타나는 것을 확인할 수 있다.

아래와 같이 4 x 4 행렬 A, B, C, D로 나타내보면,

$$A = \begin{bmatrix} 0001 \\ 0010 \\ 0100 \\ 1000 \end{bmatrix} \quad B = \begin{bmatrix} 1010 \\ 0101 \\ 1010 \\ 0101 \end{bmatrix} \quad C = \begin{bmatrix} 1100 \\ 1100 \\ 0011 \\ 0011 \end{bmatrix} \quad D = \begin{bmatrix} 0110 \\ 1001 \\ 1001 \\ 0110 \end{bmatrix}$$

(그림 6)의 변경된 16 x 16 이진행렬은 아래와 같이 나타낼 수 있다.

$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y6 \\ y7 \\ y4 \\ y5 \\ y9 \\ y8 \\ y11 \\ y10 \\ y15 \\ y14 \\ y13 \\ y12 \end{bmatrix} = \begin{bmatrix} ABCD \\ BADC \\ CDAB \\ DCBA \end{bmatrix} \cdot \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \\ x9 \\ x10 \\ x11 \\ x12 \\ x13 \\ x14 \\ x15 \end{bmatrix}$$

따라서, (그림 7)의 4 x (4 x 4)행렬이면, 확산계층을 표현할 수 있게 되어, 고정 행렬의 크기를 줄일 수 있게 된다.

$$\begin{bmatrix} 0001\ 1010\ 1100\ 0110 \\ 0010\ 0101\ 1100\ 1001 \\ 0100\ 1010\ 0011\ 1001 \\ 1000\ 0101\ 0011\ 0110 \end{bmatrix}$$

(그림 7) 변경한 4 x (4 x 4) 확산 계층의 배열

3.2 제안된 ARIA의 성능 평가

본 장에서 제안된 ARIA에 대한 암·복호화의 효율성을 측정 및 평가하기 위해서, 파일에 대한 암·복호화 시스템을 만들어 사용하였다.

파일 암·복호화 시스템의 데이터 흐름은 다음과 같다.
(1) 입력된 파일을 16바이트 단위로 분리한다.
(2) 분리된 데이터를 암호화를 실시한다.
(3) 암호화된 데이터를 복호화를 실시한다.
(4) 복호화된 데이터를 묶어, 파일로 출력한다.

파일 암·복호화 시스템은 처리의 효율성을 위해서, 버퍼를 준비하여, 처리된 데이터를 넣을 수 있게 하며, 스레드를 사용하여 처리 할 수 있게 설계 및 구현 되었으며, 최종적인 측정은 Java의 System.currentTimeMillis()를 사용하여 시작시간과 마침시간을 측정하였다. 알고리즘의 정확성을 알기 위해서, ARIA 알고리즘 테스트 벡터를 이용하여, 별도로 검증하였다[5].

테스트는 Intel Core2Duo P8700(2.53GHz)에서 진행하였다.

<표 1>은 ARIA와 제안되어 개선된 ARIA의 파일 암·복호화 속도를 비교한 표로, 66.2KB JPEG의 경우 ARIA는 718.4ms, 제안된 ARIA는 527.3ms의 속도를 보였다. 다른 종류의 파일의 암·복호화의 수치와 비교했을 때, 제안된 ARIA가 평균 30%정도 속도가 개선된 것으로 나타났다.

<표 1> 파일 암·복호화 속도 측정결과(측정 평균값)

파일 종류	TYPE / 용량	암 · 복호화 속도(ms)		비 고
		ARIA	제안된 ARIA	
400*267 Image	JPEG / 66.2KB	718.4	527.3	34%
1920*1200 Image	JPEG / 1.06MB	10529.7	8209.4	28%
640*480 Video	WMV / 15.3MB	142729	110864.3	29%

또, Java의 Runtime.getRuntime()을 이용하여, 현재 사용하고 있는 메모리의 양을 측정한 결과를 <표 2>에서 보여준다. 기존의 ARIA보다, 제안된 ARIA의 메모리 사용량이 적은 것으로 나타나지만, 측정결과가 오차 범위 안에 있으므로, ARIA와 비슷하거나 조금 효율적인 것으로 판단된다.

<표 2> 파일 암·복호화 메모리 사용량(측정 평균값)

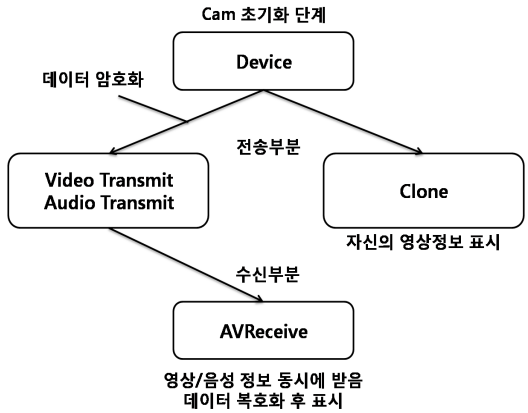
메모리 사용량 (Byte)		비 고
ARIA	제안된 ARIA	
203884760	202613568	1271192 1.21MB

위의 결과와 같이 제안된 ARIA는 기존의 ARIA보다 메모리 효율 및 성능 면에서 개선되었다고 볼 수 있다[6].

4. 보안 영상 시스템

제안된 ARIA를 검증하기 위해, 영상 정보를 암호화하여 인터넷 회선을 통해 주고 받을 수 있는 시스템인 보안 영상 시스템을 Java 전용 멀티미디어 플랫폼인 JMF를 사용하여 개발하였다.

보안 영상 시스템은 서로간의 영상 및 음성뿐만 아니라, 채팅도 가능하게 설계되었다. 효과적인 영상전송을 위해서, 두 컴퓨터의 통신은 UDP를 사용하였고, 전송하는 패킷의 구조는 RTP의 형식을 사용하였다. (그림 8)은 보안 영상 시스템의 간단한 구성도를 나타낸 것이다.



(그림 8) 보안 영상 시스템 구성도

보안 영상 시스템의 데이터 흐름은 다음과 같다.

(1) 사용자 A가 방을 만들고, Cam을 활성화 시킨다.

(2) 사용자 B가 A가 만든 방에 들어오면 Cam을 활성화 시키며, A와 영상 및 음성을 위한 RTP 프로토콜을 사용한 연결을 시도한다.

(3) 각 사용자의 영상데이터는 실시간으로 한 프레임씩 버퍼에 저장되고, 암호화 되며 전송된다.

(4) 전송된 데이터는 복호화 되어, 각 사용자에게 보여진다.

(그림 9)는 640*480의 해상도를 갖는 영상을 전송하고 있는 시연장면을 포착한 것으로, 제안된 ARIA모듈은 실시간으로 처리하기에 충분한 것을 보여준다.

(그림 9) 보안 영상 시스템 화면

5. 결론 및 고찰

본 논문에서는 ARIA 알고리즘의 핵심논리를 유지하면서, 동종 알고리즘과의 차이점으로 제시되고 있는 확산계층을 16 x 16 이진행렬에서 4 x (4 x 4)의 이진행렬로의 구조를 제안하였다. 제안된 ARIA 알고리즘은 파일 암·복호화 시스템에서의 성능측정에서 알고리즘 성능과 효율성 측면에서 기존의 ARIA 알고리즘과 비슷하거나 약간 우수한 것으로 나타났고, 실제 환경의 검증을 위한 보안 영상 시스템을 구축하기에도 적당하다고 판단된다.

제안된 구조는 ARIA 알고리즘의 기본 특징을 유지하기 때문에, ARIA가 가지고 있는 장점을 유지할 수 있다. 이는 ARIA에 대한 지속적인 개선 가능성을 보여주며, 메모리가 비교적 적게 사용되어야 하는 부분, 고속 데이터 처리가 필요한 관련 부분에서 ARIA를 이용한 응용 적용이 더 효과적으로 이루어 질 것으로 기대된다.

앞으로 ARIA 알고리즘의 메모리 사용에 좀 더 최적화된 알고리즘 구조 설계와 구현방법에 대한 연구가 더 필요할 것으로 판단된다.

참고문헌

[1] 국가보안기술연구소, 민관겸용 블록 암호 알고리즘 ARIA 알고리즘 명세서(Version 1.0), 2004.

[2] 국가보안기술연구소, New Block Cipher ; ARIA (Version 0.8), 2003.

[3] 국가보안기술연구소, Security and Performance Analysis of ARIA, 2003.

[4] SUN, Java Media Framework API, http://java.sun.com

[5] 국가보안기술연구소, ARIA 테스트 벡터(Version 1.0), 2004.

[6] K.Aoki, Optimized Software Implementations of E2, 1999.