실시간 처리 응용에 적합한 고속 스트림 암호 AA128 구현

김길호* · 조경연* · 이경현* · 신상욱**

Implementation of fast stream cipher AA128 suitable for real time processing applications

Gil-ho Kim* · Gyeong-yeon Cho* · Kyung Hyune Rhee* · Sang Uk Shin**

이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2012-0001331)

요 약

최근 휴대폰과 같은 무선 인터넷 환경과 자원의 제약을 크게 받는 무선 센서 네트워크(wireless sensor network) 등에 대한 연구가 활발히 진행되고 있다. 또한 신뢰성과 안전성이 보장된 센서 네트워크 구축을 위해 전반적으로 보안에 관한 연구가 반드시 필요하다. 센서 네트워크 보안을 위한 한 가지 방안으로 암호학적으로 안전한 알고리즘 개발이 필요하며, 따라서 본 논문에서는 휴대폰과 같은 무선 인터넷 환경, 무선 센서 네트워크, DRM(Digital Right Management) 등과 같은 실시간처리가 필요한 분야에 사용할 목적으로 소프트웨어 및 하드웨어 구현이 쉬운 128비트 스트림 암호 AA128을 제안한다. AA128은 278비트 Arithmetic Shift Register(ASR)과 비선형 변환의 두 부분으로 구성된 스트림 암호이며, 비선형 변환은 혼잡함수(Confusion Function), 비선형변환(SF0 ~ SF3)과 표백(Whitening)을 사용하여 구성된다. 제안한 AA128은 AES와 Salsa20보다 수행 속도가 빠르고, 안전성 또한 현대 암호 알고리즘에서 요구하는 조건을 만족하고 있다. 하드웨어 시뮬레이션 결과를 통해 제안한 알고리즘이 실시간 처리가 필요한 어플리케이션의 속도 요구사항을 만족시키는 성능을 가지고 있음을 보였다.

ABSTRACT

Recently, wireless Internet environment with mobile phones and wireless sensor networks with severe resource restrictions have been actively studied. Moreover, an overall security issues are essential to build a reliable and secure sensor network. One of secure solution is to develop a fast cryptographic algorithm for data encryption. Therefore, we propose a 128-bit stream cipher, AA128 which has efficient implementation of software and hardware and is suitable for real-time applications such as wireless Internet environment with mobile phones, wireless sensor networks and Digital Right Management (DRM). AA128 is stream cipher which consists of 278-bit ASR and non-linear transformation. Non-linear transformation consists of Confusion Function, Nonlinear transformation(SF0 ~ SF3) and Whitening. We show that the proposed stream cipher AA128 is faster than AES and Salsa20, and it satisfies the appropriate security requirements. Our hardware simulation result indicates that the proposed cipher algorithm can satisfy the speed requirements of real-time processing applications.

키워드

스트림 암호, LFSR, ASR, Salsa20, DRM, 무선 센서 네트워크

Key word

Stream Cipher, LFSR, ASR, Salsa20, DRM, wireless sensor network

* 정회원 : 부경대학교

** 정회원 : 부경대학교 (교신저자, shinsu@pknu.ac.kr)

접수일자 : 2012. 05. 02

심사완료일자 : 2012. 07. 23

Open Access http://dx.doi.org/10.6109/jkiice.2012.16.10.2207

Ⅰ. 서 론

최근 휴대폰과 같은 무선 인터넷 환경과 물리적인 크기의 제한이 심한 무선 센서 네트워크(Wireless Sensor Network) 등에 대한 연구가 활발히 진행되고 있다. 특히 무선 센서 네트워크는 각 센서노드의 제한적인 상황 때문에 데이터의 저장, 계산 능력, 배터리, 대역폭 등에 분명한 한계를 보이고 있으며, 더불어 신뢰성과 안전성이 확보된 센서 네트워크 구축을 위해 추가적으로 보안에 관한 연구가 반드시 필요하다.

이는 센서노드 상의 정보의 저장 및 전송에 암호 알고리즘의 구현이 필요하며, 이 암호 알고리즘은 센서노드의 한정된 자원을 효과적으로 활용할 수 있도록 구현하는 것이 핵심이 된다. 현재 센서노드 상에서 구현된 암호 알고리즘은 공개 키 암호와 블록 암호인 AES[1,2]가 있으나, 이들은 계산량과 계산에 필요한 메모리도 많이 필요로 한다. 이는 센서노드의 한계점과 정면으로 반대되는 것으로 이를 극복할 대안으로 최근에 3단계로 진행된 eSTREAM[3,4] 프로젝트에서 소프트웨어와 하드웨어 부문에서 몇 개의 스트림 암호가 선택되었다.

그러나 이는 AES와 같은 블록 암호의 완전한 표준이 아니며 eSTREAM에서 채택된 스트림 암호가 센서노드 상에서 사용 가능성은 아직 명확하지 않다. 이에 본 논문에서는 휴대폰과 같은 무선 인터넷 환경과 무선 센서 네트워크 및 Digital Right Management(DRM) 등과 같은 실시간처리가 필요한 분야에 사용할 목적으로 소프트웨어 및 하드웨어 구현이 쉬운 128비트 스트림 암호 AA128을 제안한다. AA128은 AA32[5]를 확장 및 최적화한 것으로 AA128의 키 스트림 생성을 위한 내부 상태 비트는 조금 증가했지만 수행속도에서는 AA32보다 2배 정도 향상되었고, 안전성 또한 매우 강화되었다.

AA128은 의사 랜덤 수열 생성기(Pseudo-Random Number Generator: PRNG) 역할을 수행하는 선형 궤환 시프트 레지스터(Linear Feedback Shifter Register: LFSR)를 278비트 산술 쉬프트 레지스터(Arithmetic Shift Register)[6]로 적용하였고 비선형 여과기(Non-linear Filter Generator: NFG)는 간단한 논리 연산으로만 구성되어 있어 소프트웨어 및 하드웨어 구현이 용이하다. 특히 비선형 변환 연산은 S-박스나 계산 복잡도가 높은 곱셈, 나눗셈, 지수, 유한체 연산을 사용하지 않아 제한적인 하드웨어인 센서노드에 쉽게 적용할 수 있는 암호 알고리즘이다.

최적의 스트림 암호의 설계를 위한 방법으로는 첫째 LFSR의 수행 속도 향상을 위해 워드 단위 연산을 통한 다음 상태 변환이 필수이며 이를 ASR에서 구현하여 일반적인 LFSR보다 빠른 결과를 보여주고 있다. 둘째 비선형 변환을 위한 연산으로 낮은 계산 복잡도를 가지는 논리 연산의 사용이다. 이는 S-박스를 사용하지 않기 때문에 암호의 수행 속도를 향상 시킬 수 있으며 특히 소프트웨어 및 하드웨어 환경이 제한적인 무선 센서 네트워크 구축에 반드시 필요한 기술이다.

위와 같은 최적의 스트림 암호 설계를 바탕으로 제안한 AA128은 매우 간결한 구조의 128비트 출력을 만드는 스트림 암호이며, 278비트 ASR과 비선형 변환을 사용한 결합함수[7]로 구성되어 있다. AA128에서 비선형 변환은 혼잡함수(Confusion Function), 비선형변환(SF0 ~ SF3), 표백(Whitening)으로 구성된다.

제안한 AA128은 AES, Salsa20[8] 보다 소프트웨어 수행 속도 테스트결과 최소 60%에서 최대 200% 정도 빠른 결과를 보여주고 있으며, 안전성 또한 현대 암호 알고리즘이 필요로 하는 안전성을 만족하고 있다. 그리고 하드웨어 구현 역시 설명한 어플리케이션에 필요한 속도를 충분히 만족하고 있다.

본 논문의 구성은 2장에서 무선통신 및 무선 센서 네트워크를 위해 개발된 여러 스트림 암호와 전반적인 스트림 암호에 대해서 간략한 소개와 문제점을 중심으로 설명하고, 3장에서 제안한 AA128 알고리즘을 자세히 설명하고, 4장에서 AA128의 소프트웨어, 하드웨어 구현 결과 및 안전성에 대해 분석하고, 마지막으로 결론으로 끝맺는다.

Ⅱ. 관련연구

스트림 암호는 블록 암호와 다르게 국제적인 공모를 통해 현재까지 표준을 정하지는 못했다. 대표적인 국제 공모사업으로는 NESSIE(New European Schemes for Signatures, Integrity, and Encryption)[9], eSTREAM이 있었으며 특히 NESSIE 프로젝트에서 제안된 스트림 암호는 최종 평가에서 수행속도 등 여러 가지 면에서 만족할

결과를 얻지 못해 최종적으로 선택된 알고리즘 없이 종료하였고, 다음으로 진행된 eSTREAM에서는 3차에 걸쳐 소프트웨어[10] 및 하드웨어[11] 구현 부문으로 나누어 진행된 분석 및 평가의 결과로, HC-128, Rabbit, Salsa20, SOSEMANUK, Grain, MICKEY, Trivium 스트림 암호를 선택했다[12]. 현재까지 진행된 국제공모사업에서 스트림 암호가 선정되지 못한 주된 원인은 스트림 암호의 수행 속도가 블록 암호보다 월등히 앞서지 못했다는 것이다. 이는 LFSR 기반의 스트림 암호의 한계를 보여주는 것이며 그 대안으로 AES와 같은 빠른 블록 암호의 운영모드(OFB, CTR, CFB 등)를 이용하여 키 스트림을 생성하는 방법이 있다. 이와 같은 상황에서 앞으로의 스트림 암호의 개발 방향은 Gbps급 초고속 스트림 암호나 초경량 하드웨어 환경에 맞는 스트림 암호의 개발이 절실히 요구되고 있다.

다음은 본 논문에서 AA128과 수행 테스트 비교를 위한 알고리즘들을 간략히 소개한다.

AA32는 매우 간결한 구조의 32비트 출력을 만드는 스트림 암호이며, 구성은 151비트 ASR과 비선형 변환을 위한 함수는 S-박스를 사용하지 않고 간단한 논리 연산을 적용한 축소함수(Reduction Function)로 구성된 매우 간결한 스트림 암호이다.

무선 LAN 표준을 정의하는 IEEE 802.11 규약의 일부분으로 무선 LAN 운영간 보안을 위해 사용되는 WEP는 RC4를 사용한다. RC4는 1987년 개발된 스트림 암호로 바이트 단위 치환을 기본 동작으로 내부 상태를 갱신하는 방식으로 1994년 역어셈블리 방식으로 그 구조가 인터넷에 공개된 후 많은 암호학자들로부터 관심을 받았다. 현재까지 안전하다고 여겨지고 있지만 출력 키 수열과 랜덤함수를 구별하는 distinguishing 공격[13]과 WEP에 적용되는 경우 재동기 과정[14]에서 문제점이 발견되었다. 그리고 RC4는 256바이트의 메모리를 사용하므로 초경량 하드웨어 환경에서는 적합하지 않다.

Salsa20은 eSTREAM 프로젝트에서 소프트웨어 구현 부문에 3차까지 주목받은 알고리즘으로 128비트 키와 64비트 IV를 사용해 128비트 키 스트림을 생성한다. 키 생성을 위한 내부 상태공간은 32비트로 된 4 X 4 테이블로 총 64바이트이다. 내부 상태의 갱신은 64바이트 입력과 64바이트 출력을 위한 해시 함수를 통해 이루어지고 이러한 해시 함수는 여러 개의 quarterround로 구성 되어 있으며, 각 quarterround는 XOR, 덧셈, 회전 연산으로 구성되어 있다.

참고문헌[12]은 eSTREAM 3차 평가 결과로 채택된 스트림 암호 알고리즘들의 다양한 CPU 구조에서 소프트웨어 수행속도 결과를 나타내고 있다. Intel Core2 Quad Q6600 6fb에서 40바이트 암호 수행에서 바이트 당 클록 수는 표 1과 같다.

표 1. 알고리즘 수행 속도
Table. 1 algorithm execution speed

알고리즘	바이트 당 클록 수
Salsa20	16.19
Rabit	17.68
AES	18.60
SoSemanuk	23.99
HC	498.93

본 논문에서는 이와 가장 유사한 환경에서 Salsa20, AES와 비교 수행을 한 결과 AA128의 소프트웨어 성능이 훨씬 빠르다는 것을 4장에서 설명하고 있다.

Ⅲ. AA128

AA128의 키 생성과정을 설명하기 위해 사용된 그림이나 수식 등에서 논리 및 산술 연산자들은 일반적인 표기법을 그대로 적용하고, 대문자로 표기된 변수는 32비트이며, 소문자는 8비트이다. 그리고 변수 명에 위첨자는 상태 값을 나타내며, 아래첨자는 워드 또는 바이트 단위 순서 번호이다.

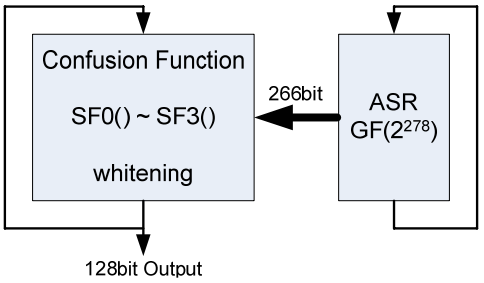

그림 1. AA128 블록도
Fig. 1 AA128 block diagram

그림 1은 초기화 과정을 수행한 다음 278비트의 ASR을 새로 갱신한 후 혼잡함수, 비선형변환(SF0 ~ SF3), 표백에 ASR의 8개의 워드와 최상위 워드에서 10비트(266bit)를 적용하여 최종적으로 128비트의 키를 생성한다. AA128은 내부 상태가 406비트이며 그림 1과 같이 수행 후 128비트의 스트림 키를 생성한다. 이어지는 다음 절부터 그림 1의 순서대로 진행과정을 자세히 설명한다.

3.1. 초기화 과정

초기화 과정은 ASR 278비트(32비트 8개와 마지막 최상위 워드는 22비트만 사용)와 혼잡함수 128비트의 초기상태를 만드는 과정으로 스트림 암호에서는 최종적인 키 생성 과정에서 초기상태는 매우 중요하며[15], 초기상태 값의 노출은 스트림 암호의 안전성과 밀접한 관계가 있다. 초기상태 값을 만들기 위해 사용자 비밀 키와 IV(Initial Vector)를 암호학적으로 안전한 의사난수함수에 입력하여 필요한 406비트(278+128)를 초기화시켜야 한다.

암호학적으로 안전한 의사난수함수로 해시함수를 사용하는 방법이 있지만, 해시함수보다 간결한 블록 암호인 AES를 이용하여 AA128의 406비트(278+128)를 초기화시킬 수 있다. AES를 이용한 초기화 방법은 AES의 안전성에 대한 보장만큼 초기화의 안전성을 보장받게 된다. AES를 사용한 구체적인 초기화 방법의 예는 다음과 같다.

먼저 사용자가 입력한 128비트 키와 128비트 IV를 각각 AES의 마스터키와 평문으로 두고 AES의 10라운드를 적용한 후 그림 3의 혼잡함수 초기 값인 $WORD^0_0$ ~ $WORD^0_3$에 대입한다. 그리고 AES 7, 8, 9라운드의 384비트 결과 값을 순차적으로 ASR의 278비트에 대입한다. 이때 ASR^0_8은 22비트만 유효한 값이 된다. 그리고 ASR^0_0 ~ ASR^0_8을 ASR의 다음 상태 값인 ASR^1_0 ~ ASR^1_8로 그림 2와 같은 방법으로 새로 갱신한 값이 최종적인 AA128의 ASR 278비트의 초기 값이 된다. 만일 ASR^1_0 ~ ASR^1_7의 초기 값이 0이거나, ASR^1_8의 22비트가 0 또는 1인 경우는 초기화 과정을 현재의 값을 가지고 AES 알고리즘을 10라운드 더 수행한 후 처음부터 다시 적용한다.

3.2. ASR

PRNG로 사용할 수 있는 산술 쉬프트 레지스터(ASR)는 $GF(2^n)$상에서 0이 아닌 초기 값에 0 또는 1이 아닌 임의의 수 D를 곱하는 수열로 정의한다. ASR의 i번째 값(상태) ASR^i는 $ASR^0 \cdot D^i$가 된다. $D^k = 1$이 되는 t가 $t = 2^n - 1$로 유일하게 되는 기약 다항식(Irreducible Polynomial)이 ASR의 특성다항식(Characteristic Polynomial)이며, ASR의 주기는 $2^n - 1$로 최대 주기를 가진다. 그리고 ASR의 선형 복잡도는 기존의 LFSR의 선형 복잡도 보다 높아서 안전도가 높고, 다음 상태 갱신을 위한 연산이 32비트 단위로 처리되므로 수행속도도 기존의 LFSR보다 빠르다.

AA128에서는 $GF(2^{278})$상에서 특성다항식은 16진수로 0x00400000 0x00000001 0x00000001 0x00000001 0x00000001 0x00000001 0x00000001 0x00000002 0x00000001, $D = 2^{22}$를 적용한다. ASR의 동작 알고리즘은 그림 2이다. 그림 2의 진행과정에서 W는 32비트 임시 저장 변수이고, ASR의 위 첨자 i는 현재 상태를 나타내고 i+1은 다음 상태를 나타낸다.

$W = ASR^i_8$
$ASR^{i+1}_8 = (ASR^i_7 \gg 10)$
$ASR^{i+1}_7 = ((ASR^i_7 \ll 22) | (ASR^i_6 \gg 10)) \hat{} W$
$ASR^{i+1}_6 = ((ASR^i_6 \ll 22) | (ASR^i_5 \gg 10)) \hat{} W$
$ASR^{i+1}_5 = ((ASR^i_5 \ll 22) | (ASR^i_4 \gg 10)) \hat{} W$
$ASR^{i+1}_4 = ((ASR^i_4 \ll 22) | (ASR^i_3 \gg 10)) \hat{} W$
$ASR^{i+1}_3 = ((ASR^i_3 \ll 22) | (ASR^i_2 \gg 10)) \hat{} W$
$ASR^{i+1}_2 = ((ASR^i_2 \ll 22) | (ASR^i_1 \gg 10)) \hat{} W$
$ASR^{i+1}_1 = ((ASR^i_1 \ll 22) | (ASR^i_0 \gg 10)) \hat{} (W \ll 1)$
$ASR^{i+1}_0 = ((ASR^i_0 \ll 22) \hat{} W$

그림 2. ASR 다음 상태 알고리즘
Fig. 2 ASR next state algorithm

3.3. 혼잡함수(Confusion Function)

혼잡함수는 128비트 $WORD^0_0$ ~ $WORD^0_3$을 가지고 간단한 논리연산과 회전연산을 사용하여 비선형 변환 연산을 구현하였으며 그림 3과 같이 22비트 ASR^1_8에서 10비트만 적용한 후 최종적으로 128비트 중간 결과를 생성한다. 먼저 32비트 $WORD^0_1$과 $WORD^0_0$을 함수 b'에 적용하고, 그 결과 32비트를 ASR^1_8의 22비트 중 LSB 5비트 값에 의한 데이터의존 회전연산(Data Dependent

Rotation)을 수행한다. 그리고 그 결과 32비트와 $WORD^0_2$의 32비트를 함수 b에 적용한 후 $WORD^0_0$과 ASR^1_7과 XOR 연산 결과를 SF0에 보낸다. SF1의 입력으로는 함수 b 결과와 ASR^1_6과 XOR연산을 한 결과이다. 유사한 방법으로 $WORD^0_2$와 $WORD^0_3$을 사용하여 함수 b를 수행하며, 처음에 적용된 데이터의존 회전연산 값을 피하기 위해 ASR^1_8의 22비트 중 이번에는 MSB 5비트로 데이터의존 회전연산을 수행한 결과와 $WORD^0_1$을 함수 b‘ 입력으로 한다. SF2의 입력은 함수 b‘와 ASR^1_5를 XOR연산한 결과이다. SF3의 입력은 함수 b‘, $WORD^0_3$, ASR^1_4를 XOR 연산한 결과로 한다.

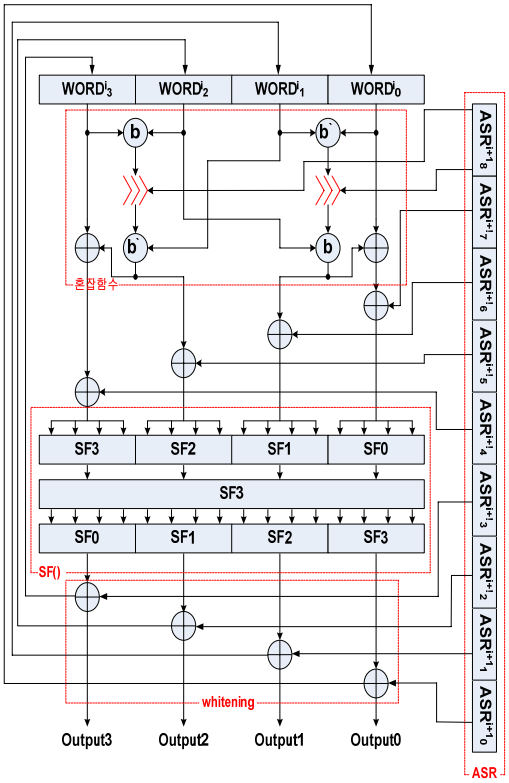

그림 3. AA128 흐름도
Fig. 3 AA128 flow diagram

혼잡함수의 비선형 연산을 수행하는 함수 b‘, b의 수행 알고리즘은 그림 4와 같다. 함수 b‘와 b를 공용체(union)로 구현하여 쉽게 8비트와 32비트 연산을 구현할 수 있다.

b‘(A, B), b(A, B)
입력 A, B는 32비트 값

$A = a_3 \| a_2 \| a_1 \| a_0$
$B = b_3 \| b_2 \| b_1 \| b_0$

b‘(A, B)
$a_3 = (a_3 \& b_3) \lll b_3$
$a_2 = (a_2 \mid b_2) \lll b_2$
$a_1 = (a_1 \& b_1) \lll b_1$
$a_0 = (a_0 \mid b_0) \lll b_0$

b(A, B)
$a_3 = (a_3 \mid b_3) \lll b_3$
$a_2 = (a_2 \& b_2) \lll b_2$
$a_1 = (a_1 \mid b_1) \lll b_1$
$a_0 = (a_0 \& b_0) \lll b_0$

출력 $A = a_3 \| a_2 \| a_1 \| a_0$ 한 32비트 값

그림 4. b‘(), b() 알고리즘
Fig. 4 b‘(), b() algorithm

3.4. SF0, SF1, SF2, SF3 함수

S-박스 대신 비선형변환을 위한 4개의 SF0 ~ SF3은 각각 32비트를 입력으로 받아 내부 처리는 8비트 단위로 수행 후 결과는 32비트를 만든다. 혼잡함수 수행결과 128비트를 LSB 워드부터 순서대로 SF0, SF1, SF2, SF3을 수행한 후 2번째 과정은 128비트 전체를 입력으로 받아 SF3을 수행한다. 이때 SF3의 내부 수행과정은 32비트 단위로 처리되고 마지막 단계는 다시 32비트 단위로 나누어 이번에는 순서를 LSB 워드부터 SF3, SF2, SF1, SF0을 수행하며, 내부처리는 8비트 단위로 한다.

그림 5는 비선형변환(SF0 ~ SF3)의 수행 알고리즘이다. 특히 SF3은 내부적으로 8비트와 32비트를 같이 수행하므로 모든 SF 함수는 공용체로 구현했다. 비선형변환(SF0 ~ SF3)은 4개의 입력 바이트 또는 워드에서 각 바이트 또는 워드의 동일한 위치의 4비트 입력으로 확산(Permutation)을 수행한 후 4비트 출력 값을 동일한 위치의 각각의 바이트 또는 워드로 보낸다. 예를 들어 SF0의 4개의 입력 바이트에서 각 바이트의 LSB가 “0000”이라면 SF0의 4개의 출력 바이트의 LSB는 “1111”이 된다. 이와 같은 변환은 4비트 입력에 대해서 4비트 출력을 만드는 일종의 S-박스와 같은 역할을 SF0이 수행한다.

이는 S-박스를 사용하지 않고 비선형 연산 및 확산을 동시에 수행하는 것과 동일한 효과를 보이며, 그림 5의 알고리즘을 수행하면 다음과 같은 출력 결과를 보인다.

SF0 = {0xF, 0x8, 0x7, 0x6, 0xB, 0xC, 0xE, 0xD,
0x0, 0x4, 0x9, 0xA, 0x5, 0x1, 0x3, 0x2}
SF1 = {0x2, 0x6, 0xB, 0x8, 0x7, 0x3, 0x1, 0x0,
0xD, 0xA, 0x4, 0x5, 0x9, 0xE, 0xF, 0xC}
SF2 = {0x3, 0x0, 0xD, 0x9, 0xA, 0xB, 0x8, 0xC,
0xF, 0xE, 0x1, 0x4, 0x5, 0x6, 0x7, 0x2}
SF3 = {0xE, 0xF, 0x2, 0x5, 0x6, 0x7, 0x4, 0x1,
0x0, 0x3, 0xC, 0xA, 0xB, 0x8, 0x9, 0xD}

```
SF0(pt3, pt2, pt1, pt0)

  pt2 ^= pt0 | pt1
  pt3 ^= ~(pt1 & pt2)
  pt0 ^= pt2 | (pt3 ^ pt1)
  pt1 ^= pt3 & pt0
  pt2 ^= pt3

SF1(pt3, pt2, pt1, pt0)

  pt2 ^= pt0 | pt1
  pt3 ^= pt1 & pt2
  pt0 ^= (pt2 ^ pt1) | pt3
  pt1 ^= ~(pt3 & pt0)
  pt2 ^= pt3

SF2(pt3, pt2, pt1, pt0)

  pt2 ^= pt0 & pt1
  pt3 ^= pt1 | pt2
  pt0 ^= ~(pt3 & pt2)
  pt1 ^= pt3 | (pt0 ^ pt2)
  pt2 ^= pt3

   SF3(pt3, pt2, pt1, pt0)

  pt2 ^= pt0 & pt1
  pt3 ^= ~(pt1 | pt2)
  pt0 ^= pt2 & pt3
  pt1 ^= (pt3 ^ pt2) | pt0
  pt2 ^= pt3
```

그림 5. SF0(), SF1(), SF2(), SF3() 알고리즘
Fig. 5 SF0(), SF1(), SF2(), SF3() algorithm

3.5. 표백(whitening)

3단계 비선형변환(SF0 ~ SF3)을 수행한 후 128비트 결과와 ASR^1_0 ~ ASR^1_3의 128비트를 32비트 단위로 각각 XOR연산으로 표백한 결과를 최종 128비트 키 출력으로 하며 또 다음 상태 128비트 혼잡함수의 입력 $WORD^1_0$ ~ $WORD^1_3$으로 갱신 한다.

다음 128비트 키는 초기화 과정을 제외하고 지금까지 설명한 순서대로 수행을 반복처리 하여 128비트 스트림 키를 생성한다.

Ⅳ. AA128의 성능 및 안전성 분석

제안한 AA128의 소프트웨어 구현은 Visual Studio 2008 C 컴파일러를 사용하였고 실행환경은 Windows Vista, Intel Core2 Duo CPU 2.26GHz, 2.27GHz, 2GB RAM의 환경에서 알고리즘의 수행 시간을 테스트했다. 결과는 표 2와 같다. 표 2에서 각 알고리즘은 초기화 과정은 제외하고, 약 12Gb의 키 생성 시간을 2번째 열에 표시했으며, 마지막 열은 각 알고리즘들의 소프트웨어 수행 중 내부 상태 변환을 위한 전역 변수 메모리양을 비교한 값이다.

표 2. 12Gb 스트림 암호 키 생성시간과 사용된 메모리 비교
Table. 2 12Gb key generation time of stream cipher and memory used compare

알고리즘	수행시간	메모리 사용
AA32[5]	43 sec	35 Byte
RC4	31 sec	256 Byte
salsa20[8]	49 sec	64 Byte
AES[2]	37 sec	1024 Byte
AA128	23 sec	51 Byte

4.1. 소프트웨어 분석

표 1의 분석을 위해 AA128과 비교된 각각의 알고리즘은 각 알고리즘의 개발자가 제공한 소스를 그대로 사용하였고 초기화과정은 제외하고 실제 키 생성 알고리즘만을 적용하여 얻은 결과이다. 스트림 암호는 생성된 키와 평문을 단순 XOR 연산 수행으로 암호문을 만들기

때문에 암호 복호 수행 시간보다는 키 생성 시간으로 스트림 암호의 성능을 평가할 수 있다.

AA32, RC4는 32비트 출력의 스트림 암호이고, 나머지 알고리즘들은 128비트 스트림 암호이다. 모든 알고리즘보다 AA128은 소프트웨어 수행 속도에서 앞서고 있다. 그리고 RC4, AES는 내부 상태 변환에 S-박스를 사용하여 메모리 사용이 AA128보다 많다. 따라서 초경량의 제한적인 하드웨어 환경에서 사용하기가 불가능하다.

그리고 AES와 Salsa20과의 비교는 AES의 경우 eSTREAM에서 제안된 스트림 암호의 성능 평가의 기준이었으며, Salsa20은 eSTREAM 3차까지 소프트웨어 구현 부문에서 가장 주목받은 알고리즘이다. 결과는 AES보다는 약 60%, Salsa20보다는 약 2배 이상 AA128이 빠른 결과를 보여 주고 있다. 이는 적은 메모리 사용이 실행 시간이 단축된 결과를 보인다.

4.2. 하드웨어 구현

하드웨어 구현은 Modelsim 6.5d를 활용하여 제안된 알고리즘의 기능을 검증하였고 그림 6은 하드웨어 블록도이다.

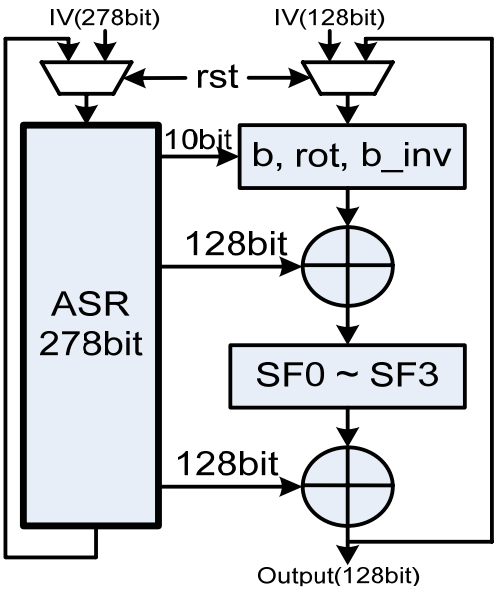

그림 6. 하드웨어 블록도
Fig. 6 Hard ware block diagram

그림 6에서 초기값 IV는 rst 신호에 의해 ASR(278bit)과 혼잡 함수(128bit)를 선택하고 이는 처음 1번만 사용되며 ASR은 3.2절의 알고리즘으로 클록 신호에 따라 갱신된다. b, rot, b_inv는 8비트 단위 혼잡함수처리로 3.3절과 같이 처리되고, SF0 ~ SF3은 3번에 걸쳐 8, 32비트 단위로 S-박스를 대용하는 연산을 처리한다.

성능은 Quartus II 11.1을 활용하여 분석한 결과, Altera Cyclone II FPGA에서 Compile한 결과 Total Logic Elements : 2674 / 68416(4%), Total Registers : 406, Total Pins: 259 / 622, 그리고 Worse Case에서 32.54MHz (4.16Gbps)의 성능을 보여주었다. 이는 무선 인터넷과 센서 네트워크 및 DRM 환경의 속도를 충분히 만족함을 보여준다. 그림 7은 Modelsim을 통한 AA128의 수행을 시뮬레이션한 결과로 AA128의 내부 상태 변환 및 128비트 출력 스트림이 정확히 생성되는 것을 보여주고 있다.

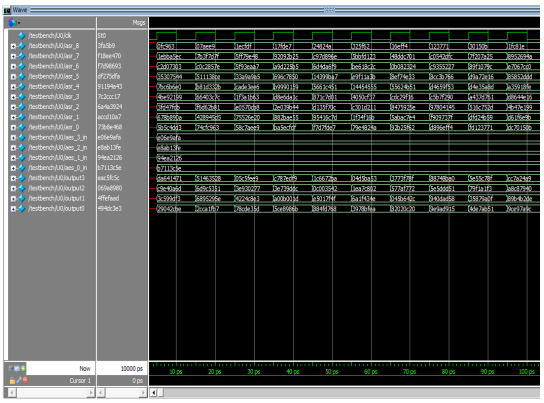

그림 7. 하드웨어 시뮬레이션 결과
Fig. 7 Hard ware simulation result

4.3. 안전성 분석

AA128의 안전성 분석은 그림 3에서 최종적인 128비트 출력 Output0 ~ Output3의 분석 확률을 구해서 전체적인 AA128의 안전성을 입증한다. 그림 3에서 혼잡함수, 비선형변환(SF0 ~ SF3), 표백에 ASR은 상태변화에 따른 일종의 라운드 키 형태로 계속해서 적용되고 있다. 따라서 ASR의 전체적인 선형 복잡도 분석 확률을 먼저 구하면, ASR의 최소 선형 복잡도는 278비트이다. 즉 556개 출력 값을 알면 ASR의 모든 상태를 분석할 수 있다.

1/556의 확률은 약 2^{-9}이다. 이는 ASR의 안전성이 약 2^{-9}이라고 추정할 수 있다. 그리고 ASR은 0을 제외한 최대 주기 수열을 생성한다. 이는 ASR의 특정 비트 또는 비트열(연속적이거나 연속적이지 않는 모든 경우)은 0을 제외한 모든 값들이 동일한 출현 빈도수를 보여줌으로 특정한 값을 추정한 어떤 공격[16]에도 내성이 있다.

혼잡함수 수행에서는 비선형 변환인 함수 b, b'의 안전성 분석확률은 바이트 단위 AND, OR, 데이터의존 회전연산의 분석을 통해 안전성을 입증할 수 있다. 먼저 논리연산(AND, OR)[17]의 안전성은 다음과 같은 분석확률을 구할 수 있다.

$$\frac{1}{2^N}\sum_{i=1}^{N}\left({}_NC_i\right) * 2^{-i} \quad (1)$$

수식 1에서 N은 바이트 단위이므로 8이고, i는 바이트 단위에서 출현한 1의 개수이다. 수식 1을 계산하면 약 0.1이며, $2^{-3.25}$이다. 다음은 데이터의존 회전연산[18]의 안전성은 가변적인 회전 연산량일 경우 하위 3비트만 회전연산에 적용되는 값이므로 분석 확률은 2^{-3}이 된다. 따라서 최종적인 바이트 단위 분석 확률은 AND, OR 연산과 데이터의존 회전연산이 서로 독립이므로 곱한 값 $2^{-6.25}$가 되고, 각각의 바이트 단위 또한 독립적으로 적용되므로 전체 워드의 분석확률은 2^{-25}가 된다. 그리고 그림 3에서 함수 b와 b' 사이에 워드 단위 데이터의존 회전연산이 독립적으로 적용되며 분석 확률은 2^{-5}이다. 따라서 비선형변환(SF0 ~ SF3)의 입력 워드는 최종적인 분석 확률이 각각 2^{-55}가 된다.

AA128의 또 다른 비선형 변환인 SF0 ~ SF3의 안전성은 4개의 입력 바이트에서 각 바이트의 동일한 위치의 4비트 입력으로 확산을 수행한 후 4비트 출력 값을 동일한 위치의 각각의 바이트로 보낸다.

그림 5의 알고리즘 진행으로 4비트 출력에 대한 최대 차분 및 선형 특성은 2^{-2}이고, 입력 비트에 대한 출력비트의 비선형 차수는 최대 3이다. 이는 비선형변환(SF0 ~ SF3)의 전체 입력 비트에 대한 분석 확률로 한 워드일 경우는 2^{-16}으로 128비트 전체는 2^{-64}가 된다. 이와 같은 연산을 3번 수행하므로 분석 확률은 2^{-192}가 된다. 따라서 최종적인 AA128의 분석 확률은 혼잡함수 분석 확률 2^{-220}과 비선형변환(SF0 ~ SF3)의 수행 후 분석 확률 2^{-192}, ASR의 분석 확률 2^{-9}를 곱한 값인 2^{-421}이 된다. 그리고 마지막 128비트인 $WORD^0_0$ ~ $WORD^0_3$과 ASR^1_0 ~ ASR^1_3을 XOR연산을 수행하여 표백하는 것은 AA128의 안전성에는 크게 영향을 미치지는 않지만 비선형변환(SF0 ~ SF3)의 단순한 출력은 SF0 ~ SF3의 내부 상태를 노출하여 분석에 사용될 수 있으며, 이는 AA128의 취약점이 될 수 있다. 이와 같이 비선형변환(SF0 ~ SF3)의 내부 상태를 숨기기 위해 XOR 연산을 통한 표백과정을 수행하는 것이다.

AA128의 출력은 128비트이지만 내부 상태 406비트에 대한 전수 조사 방법보다 더 낮은 확률이다. 따라서 AA128(2^{-421})의 안전성은 현대 암호에서 요구하는 안전성을 충분히 만족하고 있다. 계산적인 안전성으로 위와 같이 안전성이 입증되었지만 상관관계 분석(Correlation Analysis)[19]과 대수적 분석(Algebraic Analysis)[20]에 대한 안전성은 입증하지 못하였다. 제시한 알고리즘 내부의 13개 워드들이 비선형으로 확장 혼합 갱신되므로 ASR과의 상관관계를 이용하여 키를 찾는 상관 공격에 강하고, 갱신된 모든 비트가 출력에 영향을 미침으로 변수가 많아, 대수방정식을 이용하여 키를 복구하는 대수 공격에도 강하다고 보여진다.

Ⅴ. 결 론

안전하고 빠른 실시간처리가 가능한 128비트 출력의 스트림 암호 AA128을 제안하였다. AA128은 ASR 278비트와 128비트 비선형 변환 혼잡함수, 비선형변환(SF0 ~ SF3), 표백으로 구성된 결합 함수 스트림 암호 알고리즘이다. ASR은 워드 단위 피드백을 수행하므로 기존의 LFSR 보다 수행 속도가 빠르며 선형 복잡도 관점에서 안전성 또한 기존 LFSR 보다 양호하다. 또한 비선형 변환 혼잡함수, SF0 ~ SF3은 기존의 S-박스를 대치함으로써, 많은 계산량을 요구하는 산술 연산 대신 간단한 논리연산만으로 구성하여 소프트웨어 및 하드웨어 구현에서 빠르게 수행 되도록 설계하였다.

제안한 AA128은 AES, Salsa20 등과 수행 시간 테스트에서 최소 60%에서 최대 200%까지 수행 시간이 단축되었으며, 안전성 또한 현대 암호에서 요구하는 조건을 만족한다. 따라서 AA128은 휴대폰과 같은 무선 인터넷 환경과 DRM 등과 같은 실시간처리가 필요한 분야와 제한된 자원 환경인 무선 센서 네트워크(WSN)의 단말 센서

노드에 활용 가능한 고속 스트림 암호 알고리즘으로 평가된다.

추후 연구 과제로 제안한 알고리즘을 좀 더 확장 개선하여 더욱 고속인 Gbps급 스트림 암호 알고리즘을 개발함으로써 실시간으로 고속 이동 무선 통신 환경의 암호화에 적용하고자 한다.

감사의 글

이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2012-0001331)

참고문헌

[1] FIPS PUB 197, Advanced Encryption Standard(AES), NIST, 2001.

[2] D. J. Bernstein and P. Schwabe, “New AES Software Speed Records,” INDOCRYPT 2008, LNCS vol. 5365, pp. 322-336, 2008.

[3] http://www.ecrypt.eu.org/

[4] http://www.ecrypt.eu.org/stream/phase3list.html

[5] 김길호, 박창수, 김종남, 조경연, “소프트웨어 구현에 적합한 고속 스트림 암호 AA32,” 한국통신학회 논문지, 제35권, 제6호, 2010. 6.

[6] 박창수, 조경연, “갈로이 선형 궤환 레지스터의 일반화,” 전자공학회논문지, 제43권, C1편, 제1호, 2006. 1.

[7] L. Brynielsson, “On the linear complexity of combined shift register sequences,” Advances in Cryptology-Eurocrypt ‘85 pp. 156-166, 1986.

[8] D. J. Bernstein, Synchronous Stream Cipher Salsa20, http://www.ecrypt.eu.org/stream/salsa20p3.html

[9] “New European Schemes for Signatures. Integrity. and Encryption(NESSIE),” https://www.cosic.esat.kuleuven.be/nessie/

[10] http://www.ecrypt.eu.org/stream/sw.html

[11] http://www.ecrypt.eu.org/stream/hw.html

[12] Daniel J. Bernstein, “Which phase-3 eSTREAM ciphers provide the best software speeds?,” eSTREAM report 013, 2008.

[13] P. Souradyuti and B. Preneel, “Analysis of Non-fortuitous RC4 key stream generator,” Progress in Crytology-INDOCRYPT, 2003.

[14] L. of the IEEE CS, “Wireless LAN medium access control(MAC) and physical layer(PHY) specifications,” Technical Report, IEEE Standard 802.11, 1999.

[15] E. Zenner, “Why IV Setup for Stream Cipher is Difficult,” Proceedings of Dagstuhl Seminar on Symmetric Cryptography, 2007.

[16] P. Hawkes and G. Rose, “Guess-and-determine attacks on SNOW,” In Selected Areas in Cryptography - SAC 2002, LNCS vol. 2595, pp. 37 – 46, 2002.

[17] Y.L. Yin, “A Note on the Block Cipher Camellia,” a contribution for ISO/IEC JTC1/SC27, 2000.

[18] S. Contini, R. L. Rivest, M. J. B. Robshaw, and Y. L. Yin, “The Security of the RC6 Block Cipher,” 1998.

[19] P. Hawkes and G. Rose, “Correlation cryptanalysis of SSC2,” Presented at the Rump Session of CRYPTO, 2000.

[20] N. Courtois, “Fast Algebraic Attack on Stream Ciphers with Linear Feedback,” Advances in Cryptology-CRYPTO 2003, LNCS vol. 2729, pp. 176-194, 2003.

저자소개

김길호(Gil Ho Kim)

2000년 한국방송통신대학교 전자계산학과 (이학사)

2002년 부경대학교 컴퓨터공학과 (공학석사)

2010년 부경대학교 컴퓨터공학과 (공학박사)

※관심분야 : 반도체회로설계, 암호 알고리즘, 컴퓨터 구조

조경연(Gyeong Yeon Cho)

1990 인하대학교
공과대학전자공학과
정보공학전공 (공학박사)
1983-1991 삼보컴퓨터 기술연구소
책임연구원
1991-현재 부경대학교 IT융합응용공학과 교수
1991-2001 삼보컴퓨터 기술연구소 비상임기술 고문
1998-현재 에이디칩스 사외이사 겸 비상임기술고문
※관심분야 : 전산기구조, 반도체회로설계, 암호 알고리즘

이경현(Kyung Hyune Rhee)

1982. 2 경북대학교 수학교육과
(이학사)
1985. 2 KAIST 응용수학과
(이학석사)
1992. 8 KAIST 수학과 (이학박사)
1982. 2～1993. 3 한국전자통신연구원, 선임연구원
1993. 3～현재 부경대학교 IT융합응용공학과 교수
※관심분야 : 암호이론, 암호프로토콜, 네트워크 보안, 멀티미디어 보안, 클라우드 컴퓨팅 보안

신상욱(Sang Uk Shin)

1995.2 : 부경대학교
전자계산학과(이학사)
1997.2 : 부경대학교
전자계산학과(이학석사)
2000.2 : 부경대학교 전자계산학과(이학박사)
2000.4～2003.8 : 한국전자통신연구원 선임연구원
2003.9～현재 : 부경대학교 IT융합응용공학과 부교수
※관심분야 : 암호 프로토콜, E-Discovery, 디지털 포렌식, 모바일 네트워크 보안