Journal of Korea Multimedia Society Vol. 14, No. 6, June 2011(pp. 759-765)

모바일 단말에 적합한 고속 스트림 암호 MS64

김윤도[†], 김길호[††], 조경연[†††], 서경룡[††††]

요 약

본 논문에서는 소프트웨어로 구현하기 쉽고 안전하면서 빠른 모바일 단말용 고속 스트림 암호 MS64를 제안한다. 제안한 알고리즘은 연산 속도가 빠른 213비트 산술 쉬프트 레지스터(ASR)을 이용하여 이진 수열을 생성하며, 비선형 변환에서는 워드별 간결한 논리연산으로 64비트 스트림 암호를 출력한다. MS64는 128비트 키를 지원하고 현대 암호 알고리즘이 필요로 하는 안전성을 만족한다. 시뮬레이션 결과 MS64는 32비트 암호인 SSC2에 비교하여 메모리 사용량도 적고 수행 속도도 빨라 고속의 암호처리가 필요한 모바일 단말에 적합하다.

MS64: A Fast Stream Cipher for Mobile Devices

Yoon-Do Kim[†], Gil-Ho Kim[††], Gyeong-Yeon Cho[†††], Kyung_ryong Seo[††††]

ABSTRACT

In this paper, we proposed fast stream cipher MS64 for use mobile that it is secure, fast, and easy to implement software. The proposed algorithm use the fast operating 213-bit arithmetic shift register(ASR) to generate a binary sequence and produce 64-bit stream cipher by using simple logical operation in non linear transform. MS64 supports 128-bit key in encryption algorithm and satisfy with the safety requirement in modern encryption algorithm. In simulation result shows that MS64 is faster than a 32-bit stream cipher SSC2 in the speed of operation with small usage of memory thus MS64 can be used for mobile devices with fast ciphering.

Key words: ASR(산술 쉬프트 래지스터), Mobile, Stream Cipher(스트림 암호)

1. 서 론

최근 들어 스마트 폰과 같은 모바일 단말의 사용자가 급증하여 PC를 급속히 대체해가는 추세이다. 그러나 모바일 단말은 몇 가지 점에서 PC와는 다른 문제점이 있다. 그 첫 번째로 다수가 수신할 수 있는 무선통신을 사용하게 됨으로 통신보안을 위한 암호가 필요하다는 점과 두 번째로 단말기의 자원이 제약되어 있다는 것이다. 때문에 모바일 단말에서는 제한된 자원에서 실시간 처리 가능한 고속 암호 알고리즘이 필요하다.

암호의 실시간 처리를 위해서 빠른 구현 속도가 요구되는데, 블록 암호에 비해 스트림 암호는 간결하기 때문에 실행 속도가 빠른 장점이 있다. 하지만 스트림 암호는 블록 암호에 비해 안전성이 떨어지고 만족할 수준으로 수행 속도가 빠르지 못하여 국제적인 스트림 암호 프로젝트에서 선정된 알고리즘이 없다. 반면에 128비트 국제 표준 블록암호인 AES는

※ 교신저자(Corresponding Author) : 서경룡, 주소 : 부산광역시 남구 대연3동 599-1(608-810), 전화 : 051)629-6254, FAX : 051)620-6210, E-mail : krseo@pknu.ac.kr
접수일 : 2010년 11월 10일, 수정일 : 2011년 3월 17일
완료일 : 2011년 5월 3일
[†] 정회원, 부경대학교 컴퓨터공학과 박사과정 (E-mail : kimyd@pknu.ac.kr)
[††] 정회원, 부경대학교 IT융합응용공학과 박사 (E-mail : vnlqpcdd@hanmail.net)
[†††] 정회원, 부경대학교 IT융합응용공학과 교수 (E-mail : gycho@pknu.ac.kr)
[††††] 종신회원, 부경대학교 컴퓨터공학과 교수

[1,2] 수행속도 향상을 위한 꾸준한 알고리즘 연구가 진행되어 현재는 스트림 암호와 거의 같은 속도로까지 구현되고 있다. 그러나 블록 암호는 하드웨어 적으로 제약이 많은 모바일 환경에서 그대로 구현하기는 어렵다. 블록 암호는 많은 메모리를 요구하고 복잡한 계산식을 갖는 알고리즘이 많아 그 실행 속도가 느리거나 구현이 불가능하기 때문이다. 이에 비해 스트림 암호는 간결한 알고리즘으로 인해 소프트웨어로 구현이 쉬운 만큼 하드웨어적으로 제약이 있는 모바일 환경에 구현하기 적합하다[3,4,5]. 그러나 우선적으로 스트림 암호의 속도와 안전성이 개선되어야 한다.

이와 같은 요구 사항을 바탕으로 본 논문에서 제안한 MS64(Mobile Stream cipher 64bit)는 소프트웨어 구현이 쉽고 간결한 연산 구조를 가진 64비트 출력의 스트림 암호이며, 구성은 213비트 ASR(Arithmetic Shift Rregister)을 이용하여 이진 수열을 발생하고, 비선형 변환을 위해서는 워드별 논리 연산을 적용하였다. 제안한 MS64는 소프트웨어 구현을 위한 고속 스트림 암호 SSC2[3] 보다 수행 속도 테스트 결과 320% 빠른 결과를 보여주고 메모리 사용도 적다. ASR은 기존의 LFSR(Linear Feedback Shift Regitser)보다 연산 횟수가 적으며 선형 복잡도가 높다[5]. MS64는 128비트 키를 지원하고 있으며, 현대 암호 알고리즘이 필요로 하는 안전성을 만족하고 있다.

본 논문의 구성은 2장의 관련연구에서 스트림 암호와 모바일 암호에 대해서 간략한 소개와 문제점을 중심으로 설명하고, 3장에서 제안한 MS64 알고리즘을 자세히 설명하고, 4장에서 구현결과 및 안전성에 대해 분석하고 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

스트림 암호는 의사 난수 발생기(PRNG, Pseudo_Random Number Generator)가 필요한데, 이진 수열을 생성하는 선형 궤한 시프트 레지스터(LFSR)와 생성된 이진 수열에 비선형을 부과하는 비선형 여과기(NFG, Non-linear Filter Generator)를 결합시킨 구조가 많다. LFSR의 구조가 간단하고 동작속도가 빠르며 수학적으로 잘 정의 되어있는 장점이 있기 때문이다. 스트림 암호는 난수성을 갖는 좋은 통계적 특성(Statistical Properties)과 최대 주기 수열을 갖도록 디자인하는 것이 중요하며 이를 위하여 선형 복잡도(Linear Complexity)가 큰 수열을 얻는 알고리즘을 많이 사용한다.

LFSR은 구성방식에 따라서 피보나치 구성과 갈로이 구성 방식이 있다. 피보나치 LFSR은 소프트웨어로 구현이 곤란하다는 단점이 있어 이를 변형한 갈로이 LFSR이 널리 사용되고 있다. 참고논문[5]에서 제안한 ASR은 갈로이 LFSR을 일반화한 것으로 소프트웨어 및 하드웨어 구현이 용이하며, 상수 D를 변형시켜서 갈로이 LFSR과 동등 이상의 선형 복잡도를 가지는 다양한 종류의 이진 수열을 얻을 수 있으며 갈로이, 피보나치 LFSR보다 연산횟수가 33%, 71% 적고 하드웨어 복잡도는 동일하다는 장점을 가지므로 이를 모바일용 암호 구현에 적용할 경우 상당한 속도 개선을 기대할 수 있다.

비선형 변환인 NFG(Non-linear Filter Generator)는 미리 계산 후 메모리에 적재한 다음 참조하는(Look-Up Table: S-box) 방식과 계산 복잡도가 높은 곱셈, 나눗셈, 지수연산 등의 비선형 산술 연산 및 유한체 연산을 일반적으로 많이 사용하고 있다. 그러나 모바일 단말에서는 제한된 자원으로 인하여 참조 테이블이 크거나, 계산 복잡도가 높은 연산들은 가급적 사용을 피하여 부족한 자원들이 고갈되지 않게 해야 하고 간결한 연산들과 워드별 연산이나 구조체 연산 등으로 연산 횟수를 줄여야 한다.

국제적인 표준으로 정해진 스트림 암호는 없지만 특별한 분야별 표준으로는 GSM[6]: A5/1, A5/2, A5/3, IEEE 802.11의 WEP: RC4[7], UMTS/3GPP: f8[8], 블루투스: E0[9] 등이다. 전 세계적으로 가장 많이 사용되고 있는 개인 휴대 통신 기술인 GSM은 A5 알고리즘을 사용한다. A5/1[10], A5/2는 현재 분석되어 안전성에 위협을 주고 있으며, A5의 대안으로 개발된 고속 소프트웨어 구현을 위한 SSC2[3]는 Hawkes, Rose 그리고 Quick의 논문[11]에 의하면 LFG(Lagged Fibonacci Generator)의 짧은 주기와 상관관계 분석(Correlation Analysis)을 통해 현대 암호에서 필요로 하는 안전성을 만족 시키지 못한다고 주장 했다. 그리고 무선 LAN 표준을 정의하는 IEEE 802.11 규약의 일부분으로 무선 LAN 운영 간 보안을 위해 사용되는 WEP는 RC4를 사용한다. RC4는 1987년 개발된 스트림 암호로 바이트 단위

치환을 기본 동작으로 내부 상태를 갱신하는 방식으로 1994년 역 어셈블리 방식으로 그 구조가 인터넷에 공개된 후 많은 암호학자들로부터 관심을 받았다. 현재까지 안전하다고 여겨지고 있지만 출력 키 수열과 랜덤함수를 구별하는 distinguishing 공격[12]과 WEP에 적용되는 경우 재 동기 과정[13]에서 문제점이 발견되었다. 그리고 RC4는 256바이트의 메모리를 사용하므로 초경량 하드웨어 환경에서는 적합하지 않다. 블루투스 무선 환경에서 사용되는 스트림 암호 E0는 합이 128비트인 4개의 LFSR로 이루어져 있으며 64비트 키를 사용한다. 현재까지는 안전하지만 많은 연구결과들이 발표되고 있어 안전성에 위협이 되고 있다.

3. 제안한 MS64

하드웨어적으로 제약이 있는 모바일 기기에서의 암호 설계를 위한 방법으로 소프트웨어로 구현하기 쉬워야 하며 복잡도 높은 연산이나 큰 메모리 사용은 지양되어야 하고 구현 속도가 빨라야 한다. 제안한 모바일용 스트림 암호 MS64(Mobile Stream cipher 64bit)는 속도가 빠른 ASR을 이용하여 이진 수열을 생성했으며, 비선형 변환에서는 비트나 바이트 단위의 연산을 지양하고 워드단위의 간결한 연산들로 속도를 높였으며, 32비트 출력을 병렬로 2번 출력케 함으로써 출력 효율을 높였다.

제안한 알고리즘은 사용자가 입력한 키와 IV (Initial Vector)를 가지고 MD5, SHA 등 해쉬 함수를 사용하여 초기화 값을 생성하지만 이러한 해쉬 함수는 이미 구현되어 있다. 본 논문에서는 초기화 과정은 생략하고 스트림 생성 부분을 중심으로 서술한다. 그림 1은 전체적인 흐름 구성도를 표현한 것이다.

그림 1에서 우측 ASR0~6의 7개 워드는 ASR이고 상단부분 SHA0~3 워드는 초기화된 스트림 값이다. 그 아래 MIXER0(), b(), b′(), <<<(회전), MIXER1() 등은 비선형 변환을 위한 연산들이며, 아래쪽 SHA0~3 워드는 진행 후의 새롭게 업데이트 된 값이다. 진행과정은 초기 값 SHA0~3워드가 MIXER0()을 통과하면서 혼합 치환되어 업데이트된다. 다음으로 b()와 b′()에서는 2워드가 입력되어 하나의 워드로 축소 연산되고 중간의 회전연산(<<<)에서는 입력 값에 의한 워드별 회전 연산 후, 다시 한 번 MIXER1()을 통과하며 치환된다. 그리고 처음 b()와 b′()의 위치를 반대로 하여 b′()와 b() 축소함수를 수행한다. 최종 갱신된 SHA는 다음 스트림을 만드는 입력으로 사용되며, 출력은 ASR 하나의 워드와 SHA 두 워드를 XOR하여 32비트 키 스트림을 두 번 출력한다. 그리고 우측 ASR은 진행 과정에서 라운드 키와 같은 역할을 한다. ASR의 다음 상태 진행은 ASR0~5의 워드들이 ASR6 비트열과 XOR 연산하며 21비트씩 상위로 회전 이동한다.

그림 1의 진행과정의 상세한 설명은 다음과 같다.

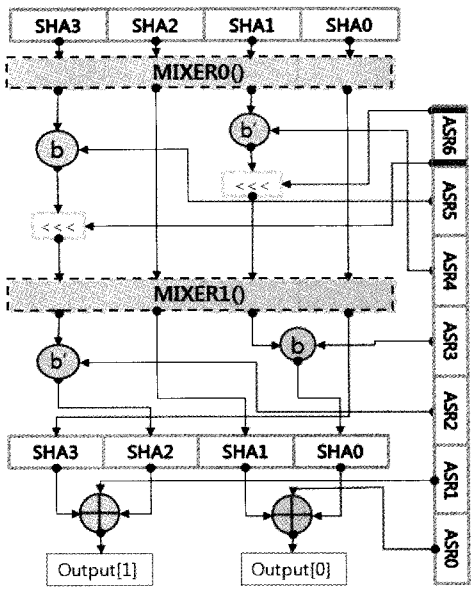


그림 1. MS64의 전체 흐름 구성도

(1) 초기화: 초기화 과정은 ASR0~6 213 비트와 SHA0~3 128 비트에 초기 값을 설정하는 과정으로 스트림 암호에서는 스트림 생성과정의 초기상태 값이 굉장히 중요하므로 AES, MD5, SHA 등의 알고리즘을 사용하여 초기화한다. 본 논문에서는 SHA256을 예로 초기화를 설명한다. 초기화 시작은 사용자가 입력한 128비트 키와 IV를 XOR 하여 SHA256의 초기 입력 값으로 하여 생성한 해쉬 값 256 비트를 ASR 213 비트에 대입하는데 하위 값부터 대입시키고 남는 상위 비트는 버린다. 다시 한번 1차 해쉬 값을 초기 값으로 SHA256을 돌려 생성된 해쉬 값으로 SHA 128비트를 하위 비트부터 초기화하고 남는 비트는 버린다.

```
w0 = ASR[6].w
ASR[6].w=ASR[5].w》11
ASR[5].w=((ASR[5].w《21)|(ASR[4].w》11))^w0
ASR[4].w=((ASR[4].w《21)|(ASR[3].w》11))^w0
ASR[3].w=((ASR[3].w《21)|(ASR[2].w》11))^(w0《1)
ASR[2].w=((ASR[2].w《21)|(ASR[1].w》11))^w0
ASR[1].w=((ASR[1].w《21)|(ASR[0].w》11))^(w0《3)
ASR[0].w=(ASR[0].w《21)^w0
```

그림 2. ASR의 다음 상태 진행

(2) ASR 진행: ASR0~5의 저장 공간은 각각 32비트씩이며, 최상위 워드를 저장하는 ASR6은 21비트만 유효한 값이 된다. 이진 수열을 생성하는 ASR은 $GF(2^N)$상에서 0이 아닌 초기 값에 0 또는 1이 아닌 임의의 수 w0를 곱하는 수열로 정의한다. 본 논문에서는 $GF(2^{213})$상에서 특성다항식은 '0x00200000 0x00000001 0x00000001 0x00000002 0x00000001 0x00000008 0x00000001'을 적용한다. ASR의 다음 상태 진행 알고리즘은 그림 2와 같다. 이와 같은 알고리즘 ASR 213비트는 최대 주기 수열을 생성한다[5].

```
MIX0(SHA[3].w, SHA[2].w, SHA[1].w, SHA[0].w)
SHA[2].w ^= SHA[0].w | SHA[1].w
SHA[3].w ^= ~(SHA[1].w & SHA[2].w)
SHA[0].w ^= SHA[2].w | (SHA[3].w^SHA[1].w
SHA[1].w ^= SHA[3].w & SHA[0].w
SHA[2].w ^= SHA[3].w
출력 SHA[3].w, SHA[2].w, SHA[1].w, SHA[0].w
```

그림 3. MIXER0() 진행 알고리즘

(3) MIXER0() 치환: MIXER0 함수는 4비트 입력, 4비트 출력, 32개 비트 슬라이스 S-박스로 4개의 워드를 입력받아 그림 3과 같이 비트별 연산으로 혼합 치환하는 과정이다.

(4) b(), b'() 연산: 함수 b(), b'()는 SHA와 ASR에서 각각 한 개 씩의 워드를 입력받아 바이트 단위로 AND, OR와 회전 연산을 수행한 후 결과를 하나의 워드로 만든다. 워드에서 바이트로 분할을 쉽게 구현하기 위해 union 구조체로 32비트 변수를 선언하였다. b(), b'()의 진행 알고리즘은 그림 4와 같다.

(5) <<<(회전) 연산: <<<(회전)은 ASR6의 21비트 중에서 하위 5비트와 상위 5비트 값에 의한 각각의 데이터 의존 회전연산(Data Dependent Rotation)이다.

```
b(A, B), b'(A, B) (입력 A, B는 32비트 값)
A = a3 ‖ a2 ‖ a1 ‖ a0,  B = b3 ‖ b2 ‖ b1 ‖ b0
b()
a3 = (a3 & b3) ≪ b3,  a2 = (a2 | b2) ≪ b2
a1 = (a1 & b1) ≪ b1,  a0 = (a0 | b0) ≪ b0
b'()
a3 = (a3 | b3) ≪ b3,  a2 = (a2 & b2) ≪ b2
a1 = (a1 | b1) ≪ b1,  a0 = (a0 & b0) ≪ b0
출력 A = a3 ‖ a2 ‖ a1 ‖ a0 한 32비트 값
```

그림 4. b(), b'()의 진행 알고리즘

(6) MIXER1()치환: MIXER1 함수는 MIXER0()와 같은 일종의 S-박스 역할을 하는 비선형 변환으로 4개의 워드를 입력받아 그림 5와 같이 비트별 연산으로 다시 한 번 치환하는 과정이다.

```
MIX_1(SHA[3].w, SHA[2].w, SHA[1].w, SHA[0].w)
SHA[2].w ^= SHA[3].w
SHA[1].w ^= SHA[3].w & SHA[0].w
SHA[0].w ^= SHA[2].w | (SHA[3].w ^ SHA[1].w)
SHA[3].w ^= ~(SHA[1].w & SHA[2].w)
SHA[2].w ^= SHA[0].w | SHA[1].w
출력 SHA[3].w, SHA[2].w, SHA[1].w, SHA[0].w
```

그림 5. MIXER1() 진행 알고리즘

(7) 표백(whitening) 출력: 출력을 위해서 업데이트 된 SHA 두 개의 워드와 ASR 하나의 워드씩을 XOR시켜 표백한다. 이는 SHA0~4 워드 중 하나의 워드라도 0값이 되거나 내부 값이 변화 없이 그대로 출력되는 것을 방지하여 출력 분석을 어렵게 만들기 위함이다. 표백 후 64비트 키 스트림을 출력하고 업데이트된 SHA0~4워드를 다음 키 스트림 생성을 위한 초기 값으로 해서 그림 1의 진행과정을 반복하며 연속적으로 64비트 키 스트림을 계속해서 생성한다.

4. 구현 및 평가

제안한 MS64의 소프트웨어 구현은 Visual Studio 2008 C 컴파일러를 사용하였고 실행환경은 Windows XP Professional Version 2002 Service Pack 3, Intel Core2 Duo CPU E7400 @ 2.80GHz, 2.79GHz. 3GB RAM의 환경에서 3Gb 키 스트림 생성 시간을 테스트했다. 다른 알고리즘들은 논문의 저자들이 공개한 각 알고리즘의 소스를 다운받아 같은 환경에서

실행 하였으며 결과는 표 1과 같다.

4.1 실험 결과 분석

표 1의 각 알고리즘 수행시간과 메모리 사용 분석을 위해 전체 알고리즘을 10번 수행 후 최대치와 최저치를 제외한 8번의 결과 값을 평균해서 반올림 했다. 본 논문의 경우 모바일 스트리밍 컨텐츠 암호화를 위한 시간 테스트로써, 하나의 키와 IV로 컨텐츠 길이만큼의 키스트림 생성이 필요하므로 3Gb의 키스트림 생성시간을 비교하였다. 스트림 암호는 생성된 키와 평문을 단순 XOR 연산 수행으로 암호문을 만들기 때문에 암호 복호 수행 시간보다는 키 생성 시간으로 스트림 암호의 수행 시간을 평가한다.

MS64는 64비트, RC4, SSC2는 32비트, Salsa20은 128비트 스트림 암호이지만 동일하게 3Gb 스트림 키 생성시간을 비교한다. 무선통신용 암호인 A5의 대안으로 개발된 SSC2보다 MS64가 약 320% 속도가 향상된 결과를 보여주고 있으며, 그 이유는 SSC2는 32비트 단위 LFSR이 4개, LFG(Lagged Fibonacci Generator) 17개를 사용하여 32비트 출력 결과를 생성한다. MS64는 NFG(SHA)에 4개, LFSR(ASR)에 7개를 사용하여 SSC2보다 10개의 워드를 적게 사용하고 64비트 출력을 생성한다. 이와 같은 적은 메모리 사용과 워드별 간결한 연산으로 실행 시간의 단축 결과를 보여주었다. 그리고 RC4는 MS64와 수행시간은 비슷하지만 RC4 알고리즘은 S-박스를 사용하고, 메모리 사용도 MS64보다 6배정도 많다. ecrypt Ⅱ[14]에서 주목받은 알고리즘인 Salsa20[15]과 비교는 MS64가 8배 빠른 실행시간을 보여주고 메모리 사용도 적다. 그리고 151비트 ASR을 이용하는 스트림암호 AA32[16]와 비교는 AA32는 블록암호 AES로 초기화함으로 메모리 부담이 일시적으로 높아질 수 있고, AA32의 2^{-119}안전성에 비해 MS64는 $2^{-191.5}$로 안전성이 개선되었고 수행 속도도 훨씬 빠르게 개선되었다.

표 1. 3Gb 스트림 암호 생성시간과 사용된 메모리 비교

알고리즘	수행시간	메모리 사용 공간
salsa20	41 sec	64Byte
SSC2	16 sec	84Byte
AA32	9 sec	35Byte
RC4	6 sec	256Byte
MS64	5 sec	43Byte

4.2 안전성 분석

MS64의 안전성 분석은 초기 값에서 최종적인 64비트 출력 과정을 단계별로 분석하여 확률을 합산해서 전체적인 MS64의 안전성을 입증한다. 첫 번째로 비선형 함수인 MIXER0()의 안전성을 분석한다. MIXER0()의 4개의 입력 워드에서 각 워드의 동일 위치의 한 비트씩을 묶어 4비트를 혼합(Mix) 입력한 후 대응하는 값으로 치환하여 출력 4비트를 각각의 워드에 동일 위치로 보낸다. 그림 3과 그림 5의 알고리즘을 수행하면 그림 6과 같은 결과로 치환된다.

예를 들어 MIXER0()의 4개의 입력 워드에서 각 워드의 첫 번째 비트들이 "0000"(0×0)이라면 Intput의 첫 번째 값이므로 이에 대응하는 Output의 첫 번째 값 "1111"(0×F)이 출력된다. 마찬가지로 "0011"(0×3)입력에는 "0110"(0×6)출력, "1101"(0×D)입력에는 "0001"(0×1)이 출력된다. 그림 3의 MIXER0() 알고리즘 진행으로 4비트 출력에 대한 최대 차분 및 선형 특성은 2^{-2}이고, 한 워드는 32비트이므로 최고 안전성은 $2^{-64} = (2^{-2})^{32}$가 된다. 그리고 MIXER0,1()은 같은 입출력 방법이며 치환되는 값만 다르므로 안전성은 동일하다.

(MIXER()입력: 4개의 워드에서 각 워드의 동일한 위치의 비트들 4비트를 입출력 값으로 한다.)
Input4bit = {0×0, 0×1, 0×2, 0×3, 0×4, 0×5, 0×6, 0×7, 0×8, 0×9, 0×A, 0×B, 0×C, 0×D, 0×E, 0×F}
(MIXER0() 출력 값)
Output = {0×F, 0×8, 0×7, 0×6, 0×B, 0×C, 0×E, 0×D, 0×0, 0×4, 0×9, 0×A, 0×5, 0×1, 0×3, 0×2}
(MIXER1() 출력 값)
Output = {0×2, 0×6, 0×B, 0×8, 0×7, 0×3, 0×1, 0×0, 0×D, 0×A, 0×4, 0×5, 0×9, 0×E, 0×F, 0×C}

그림 6. MIXER0(),1()의 4비트 입력 값에 대응하는 4비트 출력 값

두 번째로 MIXER()외의 함수 안전성을 분석하기 위해 b(), b'()와 회전연산(<<<)에 대해 분석해본다. b(), b'()의 안전성 분석확률은 AND, OR, 회전연산의 분석을 통해 안전성을 입증할 수 있다. 먼저 논리연산(AND, OR)[17]의 안전성은 바이트 단위에서 1의 개수(1의 위치는 아무런 영향이 없음)에 따라 다

음과 같은 분석확률을 구할 수 있다.

$$\frac{1}{2^N}\sum_{h=1}^{N}\binom{N}{h}\cdot 2^{-h} \quad (1)$$

수식 1에서 N은 바이트 단위이므로 8이고, h는 바이트 단위에서 출현한 1의 개수이다. 수식 1을 계산하면 약 0.1이며, $2^{-3.25}$이다. 다음은 데이터의존 회전연산의 안전성[18]은 가변적인 회전 연산 량일 경우 전수조사 이외의 특별한 방법이 없다. 그래서 바이트 단위의 회전연산은 하위 3비트만 회전연산에 적용되는 값이므로 분석 확률은 2^{-3}이 된다. 그래서 최종적인 바이트 단위 분석 확률은 AND, OR 연산과 데이터의존 회전연산이 서로 독립이므로 곱한 값 $2^{-6.25} = 2^{-3.25} * 2^{-3}$이 되고, 각각의 바이트 단위 또한 독립적으로 적용되므로 전체 워드의 분석확률은 $2^{-25} = (2^{-6.25})^4$가 된다. 그리고 그림 1에서 b()와 b'() 사이에 워드 단위 데이터의존 회전연산이 독립적으로 적용되며 분석 확률은 2^{-5}이다. 회전 연산 후 MIXER1() 함수를 수행한 다음 다시 b()와 b'()가 한 번 더 적용된다. 그러므로 MIXER0()과 MIXER1() 함수를 제외한 분석 확률은 $2^{-55} = (2^{-25})^2 * 2^{-5}$이 된다. 마지막으로 ASR의 최소 선형 복잡도는 213비트이다. 즉 426개 출력 값을 알면 ASR의 모든 상태를 분석할 수 있다. 1/426의 확률은 약 $2^{-8.5}$이다. 이로써 ASR의 안전성은 $2^{-8.5}$로 추정된다. 입력부터 출력까지의 모든 안전성은 각각이 독립적이므로 곱하면 $2^{-191.5} = (2^{-128} * 2^{-55} * 2^{-8.5})$이다. 그러므로 제안된 MS64의 최종 안전성은 $2^{-191.5}$이다.

최종 출력에서 두 개의 SHA 워드와 한 개의 ASR 워드를 XOR 연산을 수행하여 표백(whitening)하는 것은 MS64의 안전성에는 크게 영향을 미치지는 않지만 단순한 출력으로 인한 출력스트림 분석을 방지하기 위함이다. 제안한 MS64($2^{-191.5}$)의 안전성 분석을 통해 SSC2(2^{-52})보다 낮은 확률로 SSC2보다 높은 안전성을 보여주고 있다.

계산적인 안전성으로 위와 같이 안전성이 입증 되었지만 상관 공격과 대수적 공격에 대한 안전성은 입증하지 못하였다. 제시한 알고리즘 내부의 11개 워드들이 비선형으로 확장 축소 혼합 갱신되므로 LFSR과의 상관관계를 이용하여 키를 찾는 상관공격에 강하고, 갱신된 모든 비트가 출력에 영향을 미침으로 변수가 많아, 대수방정식을 이용하여 키를 복구하는 대수공격에도 강하다고 보여진다.

5. 결 론

본 논문에서는 모바일 기기 구현에 적합한 고속 스트림 암호 MS64를 제안하였다. 제안한 MS64는 213비트 ASR로 이진 수열을 생성했고, 워드별 확산 축소 연산과정을 거쳐 64비트를 출력하는 스트림 암호 알고리즘이다. ASR을 이용한 이진 수열 생성은 워드 단위 피드백을 수행하므로 기존의 비트단위 LFSR보다 수행 속도가 빠르며 선형 복잡도가 더 높다. 그리고 비선형 변환에서 S-박스를 사용하지 않아 메모리 사용량이 적으며 간결한 논리 연산들로 구성하여 제한된 하드웨어를 가진 모바일 단말에서 소프트웨어적으로 구현이 쉽게 디자인 되었다. MS64는 고속 스트림 암호 알고리즘인 SSC2와 수행 시간 테스트에서 320% 시간이 단축되었으며, 안전성 또한 SSC2보다 훨씬 높게 향상 되었다. MS64는 모바일 단말의 스트리밍 컨텐츠 암호화와 같은 실시간 암호 처리가 필요한 분야에 사용 가능한 고속 스트림 암호 알고리즘이다.

마지막으로 제안한 알고리즘을 실제로 모바일 단말에서 구현하고, 128비트 출력의 스트림 암호 알고리즘으로 확장하여 더욱 빠른 Gbps급 고속 스트림 암호 알고리즘으로 개발하는 것이 다음 연구의 목표이다.

참 고 문 헌

[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] C. Carroll, A. Chan, and M. Zhang "The software-oriented stream cipher SSC-II," FSE 2000, LNCS Vol.1978 pp. 39-56 2000.

[4] 이준석, 장화식, 이경현, "셀룰러오토마타를 이용한 스트림 암호," 한국멀티미디어 학회 논문지, Vol.5, No.2, 2002.

[5] 박창수, 조경연, "갈로이 선형 궤환 레지스터의 일반화.", 전자 공학회 논문지 제43권, C1편, 제1호, 2006.

[6] M. Walker and T. Wright, "GSM and UMTS:

The Creation of Global Mobile Communication," *John Wiley & Sons*, pp. 385-406, 2002.

[7] B. Schneier, "Appled Cryptography: Protocols, Algorithms and Source Code in C," *John Wiley & Sons*, 1996.

[8] ETSI. 3GPP TS 35.201. Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 1: f8 and f9 Specification, 2002.

[9] Bluetooth SIG. Specification of the Bluetooth System 2.0. http://www.bluetooth.com, 2005.

[10] A. Biryukov, A. Shamir and D. Wagner, "Real time Cryptanalysis of A5/1 on a PC," Proceedings of Fast Software Encryption-FES, 2000.

[11] P. Hawkes, F. Quick, and G. Roes, "A Practical Cryptanalysis of SSC2," *Selected Areas in Cryptography LNCS*, Vol.2259 pp. 27-37, 2001.

[12] P. Souradyuti and B. Preneel, "Analysis of Non-Fortuitous RC4 Key Stream Generator," *Progress in Crytology-INDOCRYPT*, 2003.

[13] L. of the IEEE CS, "Wireless LAN Medium Access Control(MAC) and Physical Layer (PHY) Specifications," *Technical Report, IEEE Standard 802.11*, 1999.

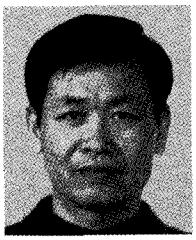
[14] http://www.ecrypt.eu.org/

[15] D. J. Bernstein, Synchronous Stream Cipher Salsa20, http://www.ecrypt.eu.org/stream/salsa20.html

[16] 김길호, 박창수, 김종남, 조경연, "소프트웨어 구현에 적합한 고속 스트림 암호 AA32", 한국통신학회논문지, 제35권 제6호, 2010. 6, pp. 954-961

[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.



김 윤 도

2003년 2월 한국방송통신대학교 컴퓨터과학과(학사)
2005년 8월 부산대학교 산업대학원 전산학과(석사)
2006년 3월~현재 부경대학교 컴퓨터공학과(박사수료)

관심분야 : 분산시스템, 컴퓨터 네트워크, 모바일 통신

김 길 호

2000년 한국방송통신대학교 전자계산학과(학사)
2002년 부경대학교 컴퓨터공학과(석사)
2010년 부경대학교컴퓨터공학과(공학박사)

관심분야 : 반도체회로설계, 암호 알고리즘, 컴퓨터 구조

조 경 연

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

서 경 룡

1983년 2월 부산대학교 전기기계공학과(학사)
1990년 2월 한국과학기술원 전기 및 전자공학과(석사)
1995년 8월 한국과학기술원 전기 및 전자공학과(공학박사)

1991년 10월~현재 부경대학교 컴퓨터공학과 교수
관심분야 : 분산시스템, 컴퓨터 네트워크