SDRAM을 사용한 난수 발생
(Random Number Generation using SDRAM)

표 창 우 †
(Changwoo Pyo)

요 약 보안을 위한 암호 키는 진난수 발생기를 사용하여 만들어야 한다. 진난수 발생기는 상태 예측이 거의 불가능한 혼란 진원지에서 초기값을 추출하여 비가역적 해시 알고리즘을 적용해 난수를 만들어 낸다. DRAM 접근 대기 시간(latency)에도 불규칙성이 존재하여 혼란 진원지 역할을 할 수 있음이 알려져 있는데, 요즘 널리 사용되는 동기식 DRAM (SDRAM)을 장착한 시스템에서는 접근 대기 시간의 불규칙성이 잘 노출되지 않으며, 난수 비트 패턴의 뭉침 현상이 심해진다. 이 문제를 xor 연산을 사용하여 해결하였다. 이 방법으로 만든 난수를 통계학적으로 평가하여 진난수에 필적하는 품질을 확인하였다. 이와 같은 난수 발생 방법의 성능은 100 Kbits/sec 수준이며, 별도의 장치나 회로를 요구하지 않아 DRAM을 장착하고 있는 여러 형태의 컴퓨터 장비에서 사용 가능하다.

키워드 : 암호 키, 난수 발생, DRAM, 접근 대기 시간, SDRAM

Abstract Cryptographic keys for security should be generated by true random number generators that apply irreversible hashing algorithms to initial values taken from a random source. As DRAM shows randomness in its access latency, it can be used as a random source. However, systems with synchronous DRAM (SDRAM) do not easily expose such randomness resulting in highly clustered random numbers. We resolved this problem by using the xor instruction. Statistical testing shows that the generated random bits have the quality comparable to true random bit sequences. The performance of bit generation is at the order of 100 Kbits/sec. Since the proposed random number generation requires neither external devices nor any special circuits, this method may be used in any computing device that employs DRAM.

Key words : Cryptographic key, random number generation, DRAM, access latency, SDRAM

1. 서 론

보안을 위한 암호 키는 난수 발생기를 사용하여 만들어 낸다. 난수 발생기는 초기값을 구성하는 비트열을 혼란도를 높이는 비가역적 해시 알고리즘으로 처리하여 난수 비트를 만들어 낸다. 난수 발생기는 초기값을 취하는 방법 또는 근원지에 따라 의사 난수 발생기(pseudo-random number generator, PRNG)와 진난수 발생기(true random number generator, TRNG)로 구분할 수 있다. PRNG는 프로그램 내에서 상수나 임의성이 크지 않은 출처로부터 초기값을 취하는 경우가 대부분이고, TRNG는 충분히 혼란스러운 진원지에서 초기값을 취한다. 원시 프로그램의 내용이나 시스템의 운영 상황을 알면 PRNG가 사용하는 초기값을 예측할 수 있는 가능성이 높아지고, 초기값을 알게 되면 난수열 전체를 예측할 수 있게 된다. 보안을 위한 암호 키는 TRNG를 사용하여 만들어야 한다.

TRNG가 사용할 수 있는 대표적 혼란 진원지는 자연계이다. 방사능 물질의 붕괴 신호[1], 번개와 같이 주변 공간에 떠도는 대기 잡음(atmospheric noise)[2], 웹캠과 같은 광학 장치에 포획되는 일광이 주는 배경 잡음[3]과 같은 예측 불가의 자연 현상이 좋은 예이다. 사람도 혼란 진원지 역할이 가능하여, 키보드나 음성 인식 장치 등을 통한 반응을 혼란 출처로 사용할 수 있다. 자연 현상으로부터 초기값을 취하는 일은 범용 컴퓨터 시

· 이 논문은 2007학년도 홍익대학교 학술연구진흥비에 의하여 지원되었음

† 종신회원 : 홍익대학교 컴퓨터공학과 교수
pyo@hongik.ac.kr

논문접수 : 2010년 1월 14일
심사완료 : 2010년 2월 17일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제4호(2010.4)

스템에서는 발견하기 힘든 특수한 신호 포획 장치를 사용해야 하기 때문에 추가적인 비용이 발생하고, 처리 속도에 있어 컴퓨터 시스템과 현저한 차이가 있어 난수 발생 성능이 제한적이고, 휴대성이 떨어져 이동 장치에는 부적합하다.

컴퓨터 시스템에 난수 발생만을 위한 별도의 회로로 도입될 수 있는 실물 TRNG는 소자의 잡음과 상태의 불안정성을 이용하여 만들어진다. 아날로그 신호인 열역학적 잡음을 증폭하여 디지털로 변환하거나[4], 디지털 소자로만 불안정한 순환 구조를 만들어, 각 소자의 상태가 끊임없이 바뀌게 하며, 임의의 시점에서 소자의 상태 값을 읽어내는 방식이다. 디지털 방식의 RNG는 회로 구동에 사용되는 시계들의 위상차 오류를 이용하거나[5], 상태 변화 중 겪을 수 있는 준안정상태를 유도한 후 임의의 안정상태로 들어가기까지의 예측 불가성을 이용하는 방식이 있다[6]. 경제성, 안정성, 성능 개선을 중심으로 많은 연구가 진행되고 있다.

컴퓨터 시스템 내부에서도 자연 현상에 가까운 무작위성을 보이는 구성원이 있어 난수 비트 발생에 사용될 수 있다. 디스크의 회전이 일으키는 공기 흔들림이 접근 시간에 변동을 일으킴을 이용하거나[7], 인터럽트 발생의 불규칙성과 이를 뒤따르는 컨텍스트 스위칭으로 인해 캐시 메모리에서 일어나는 대규모 데이터 교체 시간의 임의성을 이용하면 고성능의 비실물 TRNG를 구현할 수 있다[8].

DRAM 접근 대기 시간(latency)에도 활용할 수 있는 불규칙성이 존재하여 혼란 진원지 역할을 할 수 있다. 그러나 요즘 널리 사용되는 SDRAM을 장착한 시스템에서는 동기식 접근과 선반입 버퍼(prefetch buffer) 때문에 접근 대기 시간의 불규칙성이 잘 노출되지 않으며, 난수 비트 패턴의 뭉침 현상이 심해진다. SDRAM을 난수 발생에 활용하기 위해서는 이 문제점이 해결되어야 한다.

2절에서는 SDRAM 접근 시간이 불규칙함을 보이고, 3절에서는 SDRAM의 접근 대기 시간의 임의성을 활용하려고 할 때의 문제점인 뭉침 현상을 해소한 난수 비트 발생 방법을 펜티움 프로세서 기반의 환경에서 제시한다. 4절에서는 제안한 방식으로 생성되는 난수의 품질을 통계학적 검증 패키지를 사용하여 평가하고, 5절에서 마무리 한다.

2. 메모리 접근 시간의 임의성

DRAM의 접근 시간은 예상 보다 큰 임의성을 갖고 있는데, DRAM의 회복 연산(*refresh operation*)이 주요 요인이다. DRAM의 기억 소자는 일정 시간이 지나면 보유하고 있는 전하를 자연 방전으로 인해 상실하게 된다. 저장 값을 구분할 수 없을 정도로 전하를 잃어 버리기 전에 재충전이 이루어져야 한다. 대부분의 DRAM 소자는 64msec 이내에 회복되어야 한다 (RAMBUS DRAM의 경우 33msec 한계). 회복 연산이 시작되면 DRAM 접근은 끝날 때까지 일시 통제된다. 메모리 접근과 회복 연산이 충돌하면, 그 시점과 충돌 후 메모리 접근을 다시 허용할 때까지 남은 시간은 예측하기가 거의 불가능하며, 회복 연산 정책에 따라 50% 가까운 접근 시간의 증가가 있을 수 있다고 알려져 있다[9].

그림 1은 DDR2 SDRAM을 집중적으로 접근하는 일련의 명령어들을 일정 횟수 실행할 때 소요되는 실행 주기를 측정한 것이다. x-축은 측정 차수 이고, y-축은 경과 주기 수이다. 그림 1(a)의 아래 부분에 띠로 밀집되어 있는 부분을 확대하면 그림 1(b)와 같다. DRAM을 집중적으로 접근하는 코드는 그림 2와 같이 메모리 주소 L이 들어 있는 캐시 라인을 clflush 명령으로 쓸어낸 다음 같은 메모리 주소를 접근하는 것을 수백 차례 반복한다. 의도적으로 캐시 미스를 유발시켜 DRAM을 접근하게 하는 것이다. 핵심이 되는 clflush 명령과 mov 명령을 루프를 통해서 반복하지 않는 이유는 메모리 접근 효과가 없는 분기 연산을 피해 성능 과부하를 방지하기 위해서이다. 그림 1의 데이터는 clflush와 mov 명령 쌍을 256회 반복할 때 접근 대기 소요 주기의 변동성을 보이고 있다.

그림 2의 코드 실행 주기가 보이는 변동성은 DRAM의 회복 동작이 갖고 있는 변동성뿐만 아니라 프로세서

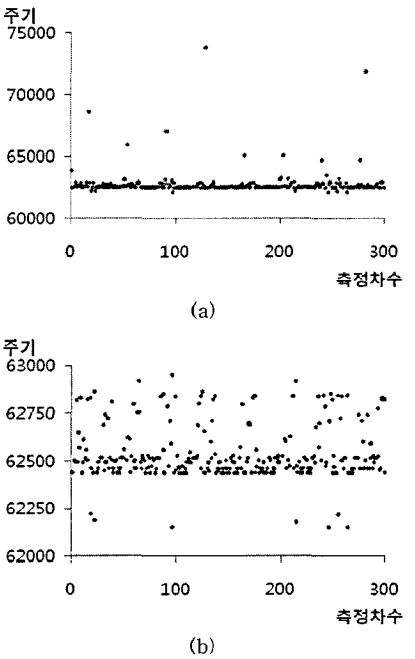

그림 1 SDRAM 접근 시간 변동성

```
clflush   L
mov       L, %eax   /* 메모리 주소 L의 데이터를 레지스터 eax로 이동 */
....
clflush   L
mov       L, %eax
```

그림 2 읽기 명령으로 구성된 DRAM 집중 접근 코드

의 비순차적 실행(out-of-order execution)도 원인이 된다. clflush 명령은 mov 명령에 대하여 전후 순서를 반드시 지키지는 않고 실행된다. 따라서 mov 명령이 실행될 때 캐시 미스가 발생할 수도 있고, 그렇지 않을 수도 있게 되는데, 이런 현상은 메모리 접근 시간의 또 다른 변동 요인이 된다.

메모리 접근 대기 시간의 변동성은 엔트로피[10]를 사용하여 정량적으로 나타낼 수 있다. 집중 접근 코드를 여러 차례 실행하여 n 종류의 실행 주기가 측정되었고, 각 실행 주기의 상대 빈도를 f_i라 하면 측정된 실행 주기의 분포가 갖는 엔트로피는 다음과 같다.

$$-\sum_{1}^{n} f_i \log_2 f_i$$

이 값의 의미는 측정된 값들의 분포가 갖고 있는 정보량이며, 측정된 값의 분포를 나타내는데 필요한 비트 수로도 볼 수 있다. 난수 발생에 대하여 적용하면 발생될 난수가 어떤 값을 가질지에 대한 불확실성의 정도를 나타낸다.

메모리 접근 시간의 불규칙성 중 대기 시간의 변동성과 비순차적 실행이 유발하는 변동성이 차지하는 비율은 접근 명령이 읽기인지 기록인지에 따라 다르게 나타난다. 그림 3은 메모리 집중 접근 코드를 읽기 연산으로만 구성했을 때와 기록 연산으로 구성했을 때, 그리고 동일 주소에 대하여 읽기와 기록 연산을 번갈아 할 때에 측정되는 엔트로피이다. 집중 접근 코드의 실행 주기가 명령어 배합에 따라 다르기 때문에 1,000 주기당 생산되는 엔트로피를 표시하였다. 각 히스토그램 기둥의 아래 부분은 순차적 접근 때의 엔트로피, 즉, 캐시 미스로 인해 DRAM이 반드시 접근될 때의 엔트로피 이다. 기둥의 윗부분은(회색) 비순차적 실행에 의해 증가되는 엔트로피 양을 나낸다. 각 기둥의 아래 부분의 숫자는 메모리 접근 반복 회수이다.

읽기 명령만으로 메모리 접근 코드를 구성하였을 때에는 메모리 접근과 비순차적 실행으로 인해 확률적으로 발생하는 캐시 메모리 미스가 만들어 내는 엔트로피가 비슷한 비율로 측정되었다. 대조적으로 기록 명령으로 메모리 집중 접근 코드를 만들었을 때에는 비순차적 실행에 의한 엔트로피 기여도가 거의 없거나, 그림 3의 write-128의 경우와 같이 순차적 접근의 경우 보다 엔트로피 감소 현상을 보이기도 했다(엔트로피 음의 영역으로 표시됨). 읽기 명령과 기록 명령을 번갈아 실행했

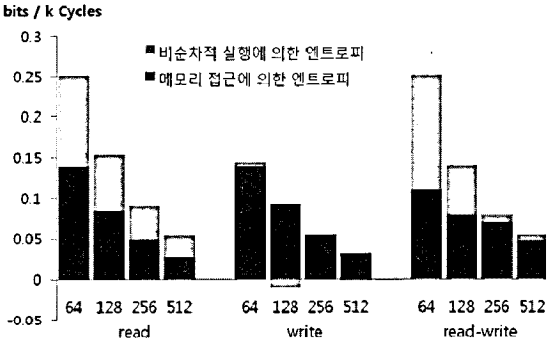

그림 3 SDRAM 접근 방식에 따른 1,000 주기 당 발생 엔트로피

을 때에도 비순차적 접근에 의한 엔트로피 기여 부분도 읽기 명령 만에 의한 것보다 현저히 적었으며 메모리 접근 회수가 증가할수록 임의로 발생하는 캐시 미스로 인한 엔트로피 기여 부분이 줄어들었다. 읽기 만으로 구성된 접근 코드와 비교할 때 엔트로피 발생 효율은 비슷한 수준이다.

읽기와 기록 연산에 따른 엔트로피 기여 부분에 차이가 나는 것은 SDRAM의 구조와 동작 방식에서 기인한다. SDRAM은 읽기 동작은 동기화 되어 진행되지만, 기록 동작은 비동기식으로 진행된다[11]. 읽기 연산은 프로세서 실행 주기에 맞추어 적시에 데이터를 공급하는 것이 목적이지만, 기록 연산은 작전에 기록한 데이터를 즉시 프로세서가 다시 사용할 때 외에는 프로세서 속도에 맞추어 진행할 필요가 없기 때문에 기록 요청은 별도의 대기열에 수집하여 메모리 시스템의 속도대로 처리하게 한다. DDR2 SDRAM의 경우에는 버스 속도를 DRAM 모듈 내부의 데이터 전송 속도 보다 2배 빠르게 하여도 이를 감당할 수 있게 지역성을 활용하는 선반입 버퍼를 가지고 있다. 선반입 버퍼에는 접근되는 기억 소자와 같은 행에 있는 인접 소자들의 값을 짧은 시간 안에 순차적으로 읽은 4 비트가 저장된다. 동기식 접근과 버퍼링은 DRAM의 읽기 접근이 보일 수 있는 접근 대기 시간의 변동성을 차폐하는 효과를 갖고 있어, 발생된 난수 값들의 뭉침을 심화시킨다.

3. 난수 발생기의 구성

발생된 난수에 뭉침 현상이 심할 경우 억지(抑止) 기법(brute force)에 의한 키 값 알아내기가 쉬워진다. 예를 들어 특정 비트 패턴의 값이 매우 높은 확률로 나타난다는 것을 알면, 이를 집중적으로 예측시도 할 수 있기 때문에 모든 가능한 비트 패턴을 사용하여 공격할 때 보다 시간을 줄일 수 있다. 뭉침 현상을 해소하는 데에는 xor 연산이 효과적이다[12]. 어떤 값들이 높은 빈

도로 나타난다는 것은 특정 비트 위치에 0과 1이 나타날 확률이 0.5에 근접하지 않고, 둘 중 하나가 더 높은 확률로 나타남을 뜻한다. 0이 나타날 확률 $p(0) = 0.5+s$, $0 < s < 0.5$, 라고 하면 xor 연산을 1회 적용했을 때 $p_1(0)$은 $(0.5+s)^2 + (0.5-s)^2 = 0.5 + 2s^2$이고, k 비트를 xor 연산으로 누적시키면, $p_k(0) = 0.5 + 2^{k-1} s^k$가 된다. $s < 0.5$ 이므로 k를 충분히 증가시키면 p(0)는 0.5에 수렴하게 된다. 즉, s 만큼의 쏠림을 제거할 수 있다.

x86 프로세서는 매 주기 마다 증가하는 64 비트의 타임 스탬프 카운터 레지스터(TSCR)를 갖고 있다. DRAM을 접근할 때마다 TSCR의 하위 32 비트를 정해진 메모리 위치에 xor 연산을 통해 비트들을 중첩시킨다. TSCR을 마지막으로 읽은 후 다음 읽을 때까지의 비트 패턴 변동에는 DRAM 접근으로 인한 일정 분량의 혼란도가 내재되게 되는데 이를 xor 연산을 통해 섞는 방식이다. TSCR은 단조 증가하기 때문에 전체 비트를 보면 가까운 미래에 가질 값이 특정 범위에 있다는 것을 알 수 있지만, 변화가 많은 하위 비트들은 작은 값의 모듈러스를 갖고 끊임 없이 회전하는 카운터와 같다. DRAM의 접근 대기시간이 보이는 정도의 변동성을 반영하여도 예측이 쉽지 않다. 2절에서는 프로그램의 2 지점에서 관측된 실행 주기의 차이의 변동성을 예시했는데, 단순히 TSCR을 읽기만 해도 같은 정도의 변동성이 누적된 값을 읽게 된다. 즉, t, $t+d_1$, $t+d_1+d_2$, $t+d_1+d_2+d_3$, … 값들을 xor 연산으로 중첩시키게 되는데, t의 상위 비트에 대응되는 부분을 피하여 비트들을 추출하면 변동성이 내재된 부분 d_i만 가지고 난수 비트를 만들 수 있다. 즉, 차를 내는 것과 다름없고, 계산 시간은 줄일 수 있다.

그림 4는 N 비트의 암호 키를 만드는 난수 발생기이다. 한번에 1 바이트씩 추출하는 과정을 N/8 회 반복한다. 메모리 접근 코드가 실행된 후에 변동성이 큰 하위 비트에서 연속적인 8 비트를 추출해 내는 방식으로 원하는 바이트 수의 난수 비트를 만들어 낸다.

```
for (i = 0; i < N/8; i++) {
    rdtsc
    clflush S
    xor %eax, S
    ...
    rdtsc
    clflush S
    xor %eax, S
    S의 비트 중 하위의 연속적인 위치의 8 비트를 추출
    추출된 비트를 버퍼에 저장
}
버퍼에 저장된 비트를 반환
```

그림 4 난수 N 비트 발생을 위한 코드의 골격. 핵심 부분은 x86 명령어로 표시되어 있다.

x86 프로세서의 rdtsc 명령은 TSCR을 읽는 연산인데, 읽은 값의 상위 32비트는 edx 레지스터에, 하위 32 비트는 eax 레지스터에 저장한다. 프로그램에서 xor 연산은 TSCR의 하위 32 비트를 지속적으로 메모리 주소 S에 섞는 일을 한다. 이와 같은 메모리 접근을 수백 차례 반복하면 S의 비트 중 일부분에 양질의 난수 비트가 발생하게 된다. 이 부분의 비트들을 필요한 만큼 축적하여 난수 비트로 공급한다.

4. 난수 품질 시험

시험은 2.33GHz 펜티움 코어2 쿼드 프로세서와 DDR2의 FB-DIMM을 장착한 서버 시스템에서 진행하였다. 난수의 품질은 15개의 개별 테스트로 구성된 NIST의 통계학적 난수 검증 패키지[13]를 사용하여 평가하였다(DFT 검증에 오류를 [14]에서 제안한 대로 수정하여 사용하였다). 각 테스트의 1회 검증은 1,000개의 표본에 대하여 반복적으로 진행하였다. 각 테스트는 표본 비트 열이 난수 비트라는 가설을 검정한다. 통계학적 검증을 통과하더라도 사용한 RNG가 완전한 TRNG라는 증거는 아니고 유의 수준에 따른 신뢰를 할 수 있음을 의미한다. 그러나 검증에 실패할 경우에는 대상 비트 열이 난수가 아님의 증거가 되며, 검증에 자주 실패할 경우 해당 RNG 난수화 능력을 다시 살펴보아야 한다. 각 테스트를 적용하면 0과 1 포함 사이 구간의 통계량 P-value를 준다. P-value의 의미는 완벽한 RNG가 검증하는 비트열 보다 난수화 정도가 낮은 비트열을 생성할 확률이다. P-value가 유의 수준 α 이상이면, $1-\alpha$ 신뢰도로 표본이 난수열임을 확신할 수 있다. NIST가 제안하는 α의 범위는 0.001이상, 0.01 이하 구간이고, 제시된 결과는 패키지 기정치인 0.01이다.

각 테스트가 주는 P-value들은 2가지 기준에서 확인하여 해당 테스트의 통과 여부를 최종적으로 결정한다. 첫째 기준은 통과율 이다. 통과하는 표본과 통과하지 않는 표본의 분포는 이항 분포를 이루는데, 표본수가 크면 정규 분포에 근접하게 된다. NIST는 통과율의 신뢰 구간을 표준 편차의 3배 이내로 제시하고 있다. 이를 $\alpha = 0.01$에 대하여 적용하면 $0.99 \pm 3\sqrt{0.99 \times 0.01/1000}$, 즉, 0.9805607 이상의 비율로 통과하여야 한다.

P-value를 검사하는 또 다른 기준은 P-value들의 분포의 균일성 이다. 각 표본이 독립적이라면 계산되는 P-value도 다양하게 나올 것이고, 균일한 분포를 보일 것이다. P-value 들을 0.1 차이의 10개의 등간격 구간에 나누어 배치한 다음 χ^2 검증을 수행한다. 균일 분포라면 구간 당 100에 가까운 P-value들을 가질 것이다.

표 1 NIST 통계학적 검증 결과. 통과하지 못한 경우가 있으면 F로 표시

검사 항목	반복회수 = 256		반복회수 = 512	
	분포	통과율	분포	통과율
Frequency				
Block Frequency (m = 128)	F	F		
Cumulative Sum Forward				
Cumulative Sum Backward				
Runs				
Long Runs of Ones	F			
Rank			F	
Spectral DFT				
Non-overlapping Templates (m = 9)	F			
Overlapping Templates (m = 9)				
Universal				
Approximate Entropy (m = 10)	F	F	F	
Random Excursions (x = +1)				F
Random Excursions Variant (x = -1)				
Serial (m = 16)	F	F		
Serial	F			
Linear Complexity (M = 500)				
전체 통과수 / 검증 회수	0/10	0/10	7/10	7/10
분포, 통과율 동시 통과 회수	0		5	
발생 속도 (K bits/sec)	176		88	

각 구간에 속한 P-value의 수를 P_i라 하면 $\chi^2 = \sum_1^{10} (P_i - 100)^2/100$ 값을 구해 불완전 감마 함수 (igamc)로 변환한 P-value가 0.0001 이상이면 균일 분포로 간주한다.

품질 평가 결과는 표 1에 요약되어 있다. 평가를 위한 난수 비트는 메모리 접근을 256회, 512회 반복하게 하고, 3번 비트부터 8 비트 단위로 추출하였다. 각 메모리 접근 회수의 첫째 열은 개별 테스트 P-value들의 분포를 보이는 P-value이며, 둘째 열은 테스트 통과 비율이다. 한번의 검증에 동일 테스트가 여러 차례 반복된 결과는 한번만 실패해도 실패로 표시했다. 표 1의 하단에는 전체 검증 회수 대비 통과 검증 회수와 비트 발생 성능을 표시하였다. 메모리 접근을 512 정도 반복하였을 때 10회 되풀이된 실험에서 5회 분포와 통과율 모두 통과하였다. 실험 결과는 xor 연산 회수를 증가시키면, 난수 품질이 좋아짐을 보이고 있다. 실험이 진행된 시스템에서 난수 발생 성능은 100 K bits/sec 정도인데, 프로세서의 속도와 DRAM 모듈의 구조에 따라 큰 변동을 보인다.

5. 결 론

DRAM 접근의 대기 시간이 회복 연산으로 인해 예측 불가능할 정도로 불규칙함을 이용하여 TRNG 수준의 난수 발생기를 제작할 수 있으나, 최근 고성능 컴퓨터에 널리 장착되는 DDR2 SDRAM은 시스템 시계와 동기화되어 작동되고 버퍼를 사용하기 때문에 DRAM 접근 대기시간의 변동성이 잘 노출되지 않는다. 이를 극복하기 위하여 xor 연산을 사용하여 포획된 비트들을 뒤섞는 방식으로 난수 발생기를 구성하였다. 제안된 난수 발생 방법은 진난수 수준에 근접하는 난수 비트열을 초당 수십에서 수백 킬로 비트 정도까지의 속도로 생성해 낼 수 있었다.

제안된 RNG는 난수화 품질이 좋은 PRNG에 초기값을 공급하는데 유용하게 사용될 수 있다. 메모리 접근 회수를 줄여 임의성에서는 손실을 입더라도 양질의 PRNG를 후처리기로 사용하면 성능과 품질의 적절한 균형을 이룰 수 있다. 제안된 RNG는 일반 컴퓨터 시스템으로부터 개인 휴대 단말기에 이르기까지 DRAM을 사용하는 모든 시스템에서 사용될 수 있는 장점이 있다.

참 고 문 헌

[1] A. Alkassar, T. Nicolay, and M. Rohe, "Obtaining true-random binary numbers from a weak radioactive source," *ICCSA (2), Lecture Notes in Computer Science*, vol.3481, pp.634-646, 2005.

[2] M. Haahr, RANDOM.ORG, http://www.random.org, Trinity College, Ireland, 1998.

[3] L. Noll, LAVARND, http://www.lavarnd.org, 2000.

[4] W. Killmann, W. Schindler, "A Design for a Physical RNG with Robust Entropy Estimators," *CHES*

2008, Lecture Notes in Computer Science, vol.5154, pp.146-163, 2008.

[5] M. Dichtl and J. Golic, "High-Speed True Random Number Generation with Logic Gates Only," *CHES 2007, Lecture Notes in Computer Science*, vol.4727, pp.45-62, 2007.

[6] I. Vasyltsov, E. Hambardzumyan, Y. Kim, B. Karpinskyy, "Fast Digital TRNG based on Metastable Ring Oscillator," *CHES 2008, Lecture Notes in Computer Science*, vol.5154, pp.164-180, 2008.

[7] D. Davis, R. Ihaka, and P. Fenstermacher, "Cryptographic randomness from air turbulence in disk drives," *CRYPTO '94: Proc. of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pp.114-120, Springer-Verlag, 1994.

[8] A. SEZNEC and N. SENDRIER, "HAVEGE: A User-Level Software Heuristic for Generating Empirically Strong Random Numbers," *ACM Transactions on Modeling and Computer Simulation*, vol.13, no.4, pp.334-346, 2003.

[9] V. Cuppu, B. Jacob, B. Davis, T. Mudge, "A Performance Comparison of Contemporary DRAM Architectures," *26th Annual International Symposium on Computer Architecture (ISCA'99)*, vol.27, no.2, pp.222-232, 1999.

[10] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed., pp.233-234, John Wiley & Sons, New York, 1996.

[11] B. Jacob, S. Ng, D. Wang, *Memory System: Cache, DRAM, Disk*, pp.465-480, Morgan-Kaufman Publishers Inc., Massachusetts, 2007.

[12] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed., p.425, John Wiley & Sons, New York, 1996.

[13] A. Rukhin, J. Soto, etal., *A statistical test suite for random and pseudorandom number generators for cryptographic applications, Revised*, L. Bassham III, Special Publication SP 800-22 rev.1, National Institute of Standards and Technology (NIST), Aug. 2008.

[14] S. Kim, K. Umeno, A. Hasegawa, Corrections of the NIST statistical test suite for randomness. *Cryptology ePrint Archive, Report 2004/018*, 2004.

표 창 우

1980년 서울대학교 전자공학과 학사 1982년 서울대학교 대학원 컴퓨터공학과 석사. 1989년 University of Illinois at Urbana-Champaign 대학원 컴퓨터과학과 박사. 1991년~현재 홍익대학교 컴퓨터공학과 교수. 관심분야는 프로그래밍 언어, 프로그램 최적화, 신뢰성 컴퓨팅