타임 스템프 카운터 레지스터를 사용한 난수 발생기

이정희° 표창우
홍익대학교 컴퓨터 공학과
jhl@cs.hongik.ac.kr°, pyo@hongik.ac.kr

Random Number Generator using Time Stamp Counter Register

Junghee Lee° Changwoo Pyo
Department of Computer Engineering, Hongik University

요 약

보안 시스템은 암호화 기능을 필요로 하고 암호화를 위한 비밀 키로 난수를 사용한다. 난수 발생기에는 순수 난수 발생기와 의사 난수 발생기가 있다. 본 논문에서는 펜티엄부터 인텔 프로세서들이 가지고 있는 타임 스탬프 카운터 레지스터(TSC MSR)에서 시드를 가져와 비트 가공을 통해 난수를 발생하는 난수 발생기를 구현하였다. 구현된 난수 발생기의 난수 품질을 평가하기 위해 순수 난수 발생기, 의사 난수 발생기의 난수 시퀀스와 비교하였다. 구현된 난수 발생기가 생성한 난수 시퀀스는 순수 난수 발생기의 난수 시퀀스와 큰 차이가 없고 특정 디바이스 없이 응용이 간단하다는 점에서 보안 시스템의 암호화 키로 사용하기에 적합하다.

1. 서 론

보안 시스템은 암호화 기능을 필요로 하고, 암호화를 위한 비밀 키로 난수를 사용한다. 전자 서명에서의 비밀 파라미터나 각종 인증 메커니즘에서 세션 키, 암호화 알고리즘과 네트워크 프로토콜의 초기 벡터에 난수가 사용된다[1]. 암호화 과정이 비밀 키에 의존하므로 공격자는 비용이 많이 드는 보안 시스템 자체에 대한 공격이나 암호화 알고리즘의 분석보다는 난수 발생기를 분석하여 비밀 키를 알아 내려고 한다.

난수 발생기에는 순수 난수 발생기(True Random Number Generator)와 의사 난수 발생기(Pseudo Random Number Generator)가 있다. 순수 난수 발생기는 사람이 예측 할 수 없는 물리적 소스로부터 시드를 취하여 난수를 생성한다. 예를 들면 이미지를 캡쳐하여 디지털 자료로 변환한 후 얻어지는 각 픽셀의 광도를 이용하거나[2] 공중전파를 이용하는 것들이 있다[3]. 이러한 순수 난수 발생기는 물리적 소스로부터 시드를 수집하기 위해 특정 디바이스가 필요하다는 단점이 있다.

의사 난수 발생기는 수학적 알고리즘이나 연산을 통해 시드를 가공하여 난수를 발생시킨다. 시스템에 적용하기 용이하나 이미 잘 알려진 암호화 알고리즘을 이용하기 때문에 난수가 노출될 수 있으므로 예측 불가능한 시드를 선택해야 한다[2]. C 라이브러리에 구현되어 있는 rand() 함수의 경우 의사 난수 발생기를 이용하며, 인자를 주지 않으면 시드를 1로 세팅하여 0과 RAND_MAX 사이의 숫자를 난수로 발생한다. 시드를 바꾸어 주지 않으면 반복적으로 같은 난수 시퀀스를 생성하는 문제점이 있다.

본 논문에서는 난수 발생을 위한 무질서 소스(chaotic source)로 마이크로프로세서의 타임 카운터 레지스터가 사용될 수 있음을 보이고 있다. 이를 위해 펜티엄 계열의 프로세서들이 가지고 있는 TSC MSR(Time Stamp Counter Model-Specific Register)[4]에서 시드를 가져와 난수를 발생하는 TSC RNG(TSC Random Number Generator)를 구현하고 발생 난수의 품질을 평가하였다. 평가는 ENT[5] 프로그램을 사용하여 다른 난수 발생기의 난수와 비교 평가하였다.

2. TSC RNG 구현

X86계열의 프로세서들은 펜티엄부터 64비트의 TSC MSR을 가지고 있다. TSC MSR 은 프로세서가 초기화되면 0부터 시작하여 매 프로세서 사이클 마다 단조 증가하며 RDTSC[6] 명령어로 EDX:EAX 레지스터 쌍에 읽어 들일 수 있다. RDTSC 명령은 항상 순차적으로 실행되는 명령이 아니고 TSC MSR 은 프로세서 사이클마다 증가되므로 RDTSC 명령이 실행될 때의 TSC MSR 의 비트 패턴을 예측하는 것은 거의 불가능하다.

그림 1은 시드를 생성하고 비트를 가공하여 난수를 생

```
long rand_gen()
{
        long reg_l, reg_r;
        asm
        (
                "rdtsc;"
                "movl %%eax, %0;"
                "movl %%edx, %1;"

                "movl %%eax, %%edx;"
                "roll $7, %%eax;"
                "rorl $7, %%edx;"
                "xorl %%eax, %%edx;"

                : "=r"(reg_l), "=r"(reg_r)

        );
        return reg_r;
}
```

그림 1 TSC MSR을 사용한 난수 생성

성하는 과정이다. 난수를 발생시킬 때마다 TSC MSR 을 읽어서 EDX 레지스터에 상위 비트들을, EAX 레지스터에 하위 비트들을 저장하고 그 중, EAX에 저장한 하위 32비트를 시드로 사용한다. 비트의 단조성을 제거하기 위해 비트 가공을 하는데 시드를 좌측으로 7비트 회전 시킨 결과와 우측으로 7비트 회전 시킨 결과를 xor 연산하여 난수를 생성한다.

3. TSC RNG 평가 실험

ENT 프로그램으로 TSC RNG 의 난수 품질을 평가하기 위해 TSC RNG 가 발생하는 난수 시퀀스를 random.org, 리눅스 C 라이브러리에 있는 rand() 함수의 난수 시퀀스와 비교하였다. 실험을 위해 각 난수 발생기로부터 10000개의 난수를 생성하였다. TSC MSR 의 단조 증가로 인해 TSC RNG 가 생성한 연속된 난수들 사이에 연관성이 발생할 수도 있으므로 난수를 하나 발생하고 다시 TSC MSR을 읽어 들이기 전, rand() 함수를 호출하여 하위 11비트만큼 루프를 돌며 시간차를 두었다. random.org 사이트에서 10000바이트의 난수 시퀀스를 구하고 rand() 함수는 시드를 주지 않은 상태에서 호출하여 10000개의 난수를 발생시켰다.

난수 시퀀스를 평가하는 여러 가지 방법이 있는데 ENT 프로그램은 난수 시퀀스를 8비트씩 잘라서 난수를 평가하며 파일 압축률, 엔트로피, 카이 제곱 분포, 산술 평균, 몬테 카를로 파이, 난수 간의 연속 상관 관계를 이용하여 난수가 가지는 단일 분포와 난수간의 독립성을 보여준다[7, 8].

일정한 패턴을 가지고 있는 파일은 압축이 가능하며, 난수를 생성한 파일에서 일정한 패턴은 곧 난수의 재현성을 의미한다. 압축률이 0%에 가까울수록 일정한 패턴을 가지고 있지 않다는 것을 보여준다.

엔트로피의 값은 압축률과도 연관이 있다. 압축률이 0%라면 일정한 패턴이 없는 난수 시퀀스이고 8비트를 최대한 난수 시퀀스로 사용한 것이다. 압축률이 0%일 경우, 엔트로피 값은 8에 가까우며 난수로 평가한다.

카이 제곱 분포는 발생된 난수의 표본 분포가 기대값과 얼마나 일치하는가를 평가한다. 기대값은 순수 난수 시퀀스의 분포를 말한다. 평가 결과는 정확한 숫자나 백분율로 나타내는데 난수 시퀀스를 백분율 범위로 표현했을 때 5%~95%사이의 값이 나오면 난수이다.

산술 평균값은 발생한 값이 집중되어 있는 정도를 나타낸다. 발생한 난수 시퀀스를 모두 합하여 파일 길이로 나누어 주며 127.5에 가까워질수록 난수로 평가한다.

몬테 카를로 파이는 정사각형 안에 원을 내접하여 그리고 좌표를 무작위로 찍었을 때, 원 안에 들어오는 값의 백분율을 계산하여 파이 값을 구한다. 원의 반지름은 1로 계산하고 원의 넓이를 파이로 본다. 난수 시퀀스를 6바이트씩 잘라 각 24비트를 X 와 Y 좌표값으로 변환하여 좌표를 구하며, 비트 시퀀스가 난수에 가까울수록 파이 값인 3.14에 근접한다.

연속 상관 관계는 발생한 값이 이전에 발생한 값에 어느 정도 의존하는지 난수간의 연관성을 평가하는 것이다. 음수든 양수든 0에 가까울수록 난수로 평가된다.

그림 2는 각 난수 시퀀스의 카이 제곱 분포를 보여준다. random.org 와 TSC RNG 가 발생한 난수 시퀀스의 카이 제곱 분포는 5%~95% 사이에 분포되어 있는 것을 볼 수 있다. 실험 횟수와 무관하게 시드가 바뀌지 않는 한 rand() 함수는 같은 난수 시퀀스를 생성하므로 rand() 함수의 분포는 하나의 결과 값만을 가지는데 그 값은 95%였다. 그림 3은 각 난수 시퀀스의 산술 평균이다. 세 난수 발생기 모두 127.5에 근접하지만 random.org의 산술 평균보다 TSC RNG의 평균이 127.5에 더 많이 분포한다. rand() 함수의 산술 평균은 127이다. 그림 4는 이전 값과의 연관성 여부를 평가하는 연속 상관 관계를 보여준다. TSC RNG와 random.org의 값 모두 0에 근사한 값을 보임으로써 난수를 생성하였다. 역시 rand() 함수의 결과 값은 - 0.00375 하나의 값만을 가진다. 압축률은 세 난수 발생기 모두 0%였고 엔트로피 값도 역시 난수로 평가 될 수 있는 7.97~8.0비트 사이의 결과를 보여주었다. 몬테 카를로 파이는 rand() 함수와 TSC RNG 의 결과값 모두 순수 난수 발생기의 평가 결과인 3.04~3.2사이의 값에 포함되는 것을 보여준다.

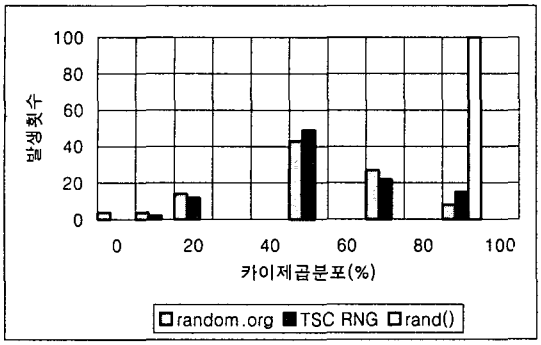

그림 2 카이 제곱 분포

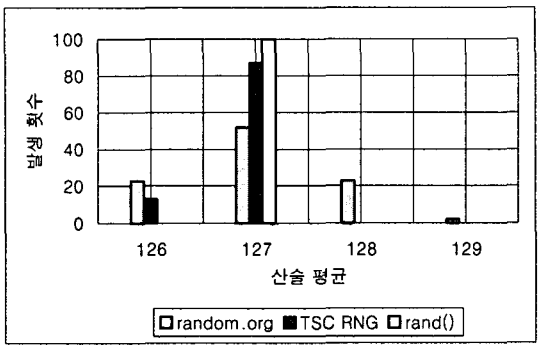

그림 3 산술 평균

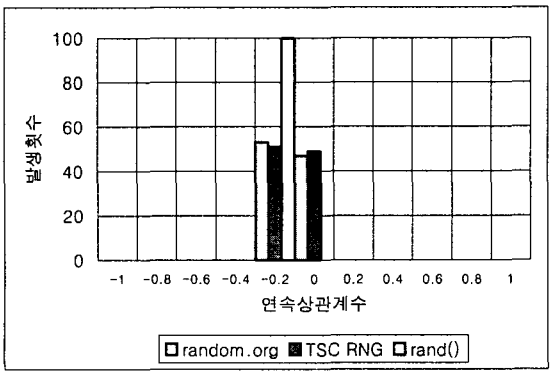

그림 4 연속 상관 관계

실험 결과, TSC RNG는 카이 제곱 분포의 경우 95%~100% 사이에 발생한 난수가 random.org 보다 많았지만 전체적으로 순수 난수 시퀀스와 매우 유사하며 rand() 함수보다 좋은 결과를 보였다. 산술 평균은 TSC RNG 가 127.5에 더 많이 분포함을 보여주었고 연속 상관 관계도 순수 난수 시퀀스 와 차이가 없었다.

4.결론

TSC RNG 는 실험에서 순수 난수 시퀀스와 매우 유사한 평가 결과를 보여주었다. TSC RNG의 난수는 TSC MSR의 특성상 값의 예측이나 재현이 매우 어렵다. 또한 엔트로피 소스로서의 특별한 외부 장치를 필요로 하지 않고 프로세서 내부에 있는 TSC MSR을 사용하여 난수를 발생하기 때문에 타임 카운터를 읽을 수 있는 시스템에서는 프로그램 설치만으로도 경제적이며 효율적인 난수 발생기를 얻을 수 있다. 이러한 특성은 TSC RNG가 안전성, 효율성, 이식성이 높은 암호화 키 발생기로 사용될 수 있음을 보여 준다. TSC RNG가 사용한 비트 난수화 알고리즘 보다 우수한 암호학적 안전 해쉬 알고리즘[9]을 사용할 경우 발생 비용이 증가할 것이나 더 나은 품질의 암호화 키를 얻을 수 있을 것으로 예상된다.

참고 문헌

[1] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, *Cryptanalytic Attacks on Pseudorandom Number Generators,* In Proceeding of the Fast Software Encryption, Fifth International Workshop, Springer-Verlag, Mar 1998.

[2] LavaRnd Random Number generator, http://www.lavarand.com.

[3] Random.org generator http://www.random.org.

[4] IA-32 Intel® Architecture Software Developer' s Manual Volume 3: System Programming Guide .

[5] John Walker, Forumilab, ENT: A Pseudorandom Number Sequence Test Program.

[6] IA-32 Intel® Architecture Software Developer' s Manual Volume 2B : Instruction Set Reference, N-Z.

[7] Pierre L' ecuyer, *Software for uniform random number generation : distinguishing the good and the bad.* In Proceedings of the 2001 Winter Simulation Conference.

[8] Louise Foley, *Analysis of an On-Line Random Number Generator.* The Distributed Systems Group, Computer Science Department, Trinity College Dublin, Apr 2001.

[9] National Institute for Standards and Technology, *Secure Hash Standard.* NIST FIPS PUB 180,U.S. Department of Commerce, 1993.