클라이언트-서버 환경에서 암호계를 위한 의사 난수 발생에 대한 연구

김도완*, 정태충 (haetnim@iislab.kyunghee.ac.kr, tcchung@nms.kyunghee.ac.kr)
경희대학교 전자정보학부

Research of Pseudo-Random Number Generator for Cryptography in Client-Server Environment

Kim DoWan, Chung TaeChoong
Department of Computer Science, Kyunghee University

요 약

본 논문에서는 각종 암호계에 중요하게 이용되는 난수를 클라이언트-서버 환경에서 생성하는 방법에 대해 연구하였다. 완벽하게 랜덤으로 생성되는 난수를 만든다는 것은 불가능하므로, 난수를 발생시키는 알고리즘의 목표는, 입수할 수 있는 정보만으로는 예측 불가능한 랜덤성을 가지는 것이다. 여기서는 클라이언트-서버 환경의 특징을 이용해 새로운 돌연변이를 만들어 좀 더 강한 랜덤성을 지니는 난수의 생성을 조합 시프트 레지스터를 이용해 연구하였다

1. 서론

암복호화를 위해서 많은 난수를 생성시키고 있다. 메시지 다이제스트의 경우 한 번 쓰고 버리는 키로서 난수를 발생시키며, ECC[3]의 경우에는 난수가 바로 암복호화 키가 된다. 또한 RSA과 같은 고전적 방법에서도 소수를 찾기 위해 난수를 발생시킨다. 하지만 완전한 랜덤성을 가지는 난수를 생성해 낸다는 것은 사실 불가능하므로, 외부에 유출될 수 있는 정보를 이용해서는 예측이 불가능한 정도의 랜덤성을 가지는 난수를 만든다.

보통 난수를 발생시키는 데 있어서는 선형 합동 발생기[1][2](Linear Congruential Generator)와 다중 재귀 발생기(Multiple Recursive Generator)[2], 비선형 방법, 그리고 선형 피드백 시프트 레지스터[2](Linear Feedback Shift Register)의 3가지 방법을 많이 사용한다. 본 논문에서는 이 중에서 피드백 시프트 레지스터 방식을 기본으로, 비선형적으로 레지스터를 피드백하는 방식에 더불어 인터넷에서 사용하는 TCP-IP상의 Ping을 이용해 여기에 Genetic Algorithm에서 사용하는 돌연변이 생성 연산을 적용해 좀 더 강한 랜덤성을 가지는 난수 발생에 관해 연구하였다.

2. 난수 발생 방법의 종류

(1) 선형 합동 발생기(LCG)와 다중 재귀 발생기(MRG)[1][2]

1차원적인 선형 합동 발생기는 다음과 같은 수식으로 이루어진다.

$$X_n = (aX_{n-1} + b) \bmod m$$

또한, 다중 재귀 발생기는 다음과 같은 수식을 사용한다.

$$X_n = (a_1X_{n-1} + \cdots + a_kX_{n-k}) \bmod m$$

불행히도 이 발생기들은 예측이 가능하기 때문에 이대로는 암호계에 적용할 수 없다. 따라서 이것을 조합한 조합 선형 발생기를 사용하는데, 여기에는 X_n으로서

행렬을 이용하는 방법과 MRG와 LCG를 조합한 방법 등이 있다. Quadratic LCG[2]같은 경우, LCG를 4개를 조합해서 난수를 생성하고 있다.

(2) 비선형 발생기[2]

X_{n-1}에서 X_n으로 전이하는 전이 함수 T를 비선형 함수를 이용한다는 전략이다. 대표적인 것으로 역합동 발생기(Inversive Congruential Generator)가 있다. 이것은 LCG에 비선형적인 왜곡을 추가한 것으로 예를 들면 다음과 같다.

$$U_n = ((X_{n-1} \times X_n^{-1}) \bmod m) / m$$

(3) 피드백 시프트 레지스터(Feedback Shift Register)

컴퓨터 내에서 산술 연산이 모두 2진으로 이루어진다는 것을 이용해, 선형 혹은 비선형적으로 각 비트들을 선발해 시프트와 비트 연산을 거쳐 새로운 난수를 발생시키는 방법이다. 가장 기본적인 선형 피드백 시프트 레지스터(Linear feedback Shift Register)[1]를 살펴보면 다음과 같은 구조를 지니고 있다.

기본 다항식이 존재하는데, 그것은 (32,7,5,3,1,0)과 같이 표시하며 이 의미는 $x^{32} + x^7 + x^5 + x^3 + x + 1$ 을 말하는 것이다. 이 다항식은 사용자가 임의의 것을 선택할 수 있다. 그리고 다항식이 준비되고, 어떤 한 비트 스트림이 준비되면, 다항식의 차수에 해당하는 비트를 모두 XOR한다. 위의 다항식을 예로 들면, 32번째 비트와 7번째 비트, 그리고 5번째 비트와 3번째 비트, 그리고 첫 번째 비트이다. 그리고 이것을 모두 XOR한 값을 가장 최상위 비트에 넣고 오른쪽으로 1비트를 시프트시킨다. 이것이 원래의 비트 스트림으로부터 새롭게 유도된 비트 스트림이 된다.

3. 본 논문에서 제시한 방법

본 논문에서는 FSR을 기본으로 하여 작성된 ISSAC[4]을 기본 베이스로 하고 있다. 기본적인 구조는 그림 1에 표시한 바와 같다. 그림에서 보는 바와 같이 기본적으로 비트를 더하는 것이 아니라 생성되고 있는 i번째의 난수를 생성시키기 위해서 비트를 더하는 것이 아니라 난수들의 chunk를 만들어 이들을 더한 후 이 난수의 비트 스트림을 로테이션시켜 새로운 난수를 구하

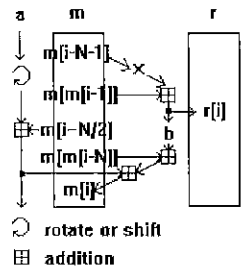

그림 1. ISSAC의 구조

고 있다. 이 난수 발생기의 C언어로 된 소스는 http://ourworld.compuserve.com/homepages/bob_jenkins/isaac.htm에서 찾아볼 수 있다.

본 논문에서는 이 ISSAC의 성능을 좀 더 향상시키기 위해 다음과 같은 방법을 적용하고 있다.

①먼저 현재의 방법으로는 그 컴퓨터가 지원하는 최대 정수형의 길이까지만 난수를 발생시킬 수 있으므로(인텔계열의 CPU에서는 2^{64}-1까지) 이것을 확장하기 위해 정수인 난수를 비트스트림으로 바꾸어 다음과 같은 공식에 의해 조합한다.

Output =

if X_1 : *even,* X_2 . *even* $\Rightarrow X_1$ * (*maximum mumber of available* X_1 + 1) + X_2

if X_1 : *even,* X_2 . *odd* $\Rightarrow (X_2$ * (*maximum mumber of available* X_2 + 1))$^{-1}$ + $X1$

if X_1 : *odd,* X_2 : *even* $\Rightarrow X_1$ * (*maximum mumber of available* X_1 + 1) + X_2^{-1}

if X_1 : *odd,* X_2 . *odd* $\Rightarrow X_2$ * (*maximum mumber of available* X_2 + 1) + X_1

②클라이언트-서버 환경에서 작동하기 때문에 좀 더 강한 랜덤성을 보장할 수 있게 새로운 연산자를 도입하였다. 이것은 다음과 같다.

1. 먼저 클라이언트가 서버에 접속을 요구하면서 그 서버에 TCP-IP서비스 중 하나인 ping을 보낸다.
2. 돌아온 ping의 소요 시간을 측정한다.

first ping time = P_1

second ping time = P_2

third ping time = P_3 라고 둔다.

3. $Q = P_1 \bmod 3$을 구한다.

4. 만약 $Q = 0$이면 난수를 그대로 둔다.
5. 만약 $Q = 1$이면 $R = P2 \times P3 \bmod L$(*L:binary length of random number*)을 구해 1번째 비트부터 R번째 비트와 $R+1$번째 비트부터 최상위 비트까지를 서로 바꾼다. (역위 연산)
6. 만약 $Q=2$이면 5와 같이 R을 구해 R의 비트를 역으로 바꾼다. (돌연변이)

4. 실험 결과

위에 제시한 알고리즘을 통해 다음과 같은 실험을 하였다.

① $0 < x < 2^{32}-1$의 난수인 정수를 512개 발생시킨다.
② 위의 구간을 각각 10^8을 구간 거리로 하여 43개로 나눈 후 그 구간에 존재하는 난수의 개수를 세었다.
③ 이상적으로 골고루 퍼진 분포인 구간마다 존재하는 난수의 개수가 12개인 경우와 비교 실험하였다.

위 실험을 10번 반복하여 가장 높은 값과 가장 낮은 값을 제거한 후 평균치를 내었다.

먼저, 발생된 난수의 구간 도수(구간에 존재하는 난수의 개수를 의미함)의 표준편차는 약 19.7이었다. 보다 정확한 척도를 위하여 본 논문에서는 이상적으로 골고루 퍼진 분포의 엔트로피와 실험 결과의 엔트로피를 구해 보았다. 엔트로피는 다음과 같이 구해진다.

$$Ent(p) = -\sum_{i=1}^{k} p_i \log p_i$$

엔트로피는 정보의 불확실성을 나타내며 일어날 수 있는 사실의 발생확률의 대수값을 나타내며, 대수함수는 단조증가이므로 엔트로피가 클수록 대상은 불확실하다는 것을 의미한다. 실험에서는 이상적인 분포의 엔트로피가 약 3.59로 나왔고 기본적인 ISSAC의 엔트로피는 3.70인 반면 본 논문의 난수 발생기에 의해 분포된 도수의 엔트로피는 약 3.73이 나왔다. 참고로 LCG같은 발생기의 엔트로피는 이상적 분포와 거의 일치하게 나오지만 이것은 LCG의 성능이 좋은 것이 아니라 예측 가능한 패턴으로 난수가 발생하기 때문이다. 실험 결과 돌연변이 연산은 해의 불확실성을 높이는데는 별로 영향을 주지 않음을 알 수 있었다. 반면에 TCP-IP상의 Ping에 의한 "real"난수를 돌연변이 연산에 적용함으로서 좀 더 강한 랜덤성을 지니게 하고 있다.

5. 향후 연구 과제

DIEHARD[5]테스트란 어떤 난수의 수열이 임의성을 강하게 지니고 있는지를 테스트하는 것으로서 많은 학자들 사이에서 신뢰를 얻고 있는 테스트이다. 이 테스트를 본 논문에서 제시한 발생기에 적용시켜 좀 더 신뢰할 수 있는 결과를 얻게 하는 것이 향후 연구 계획의 주 목적이다. 아울러 비선형 방법 중 하나인 inversive congruential generator(ICG)방법을 적용해 한층 더 강화된 랜덤성을 부가할 계획이다. 아울러 ICG의 단점 중 하나인 느린 속도를 알고리즘의 개선을 통하여 이룰 것이다.

6. 참고 문헌

[1] Bruce Schneier, Applied Cryptography 2nd Edition, P.369, 1996
[2] Pierre L'ecuyer, Handbok of Simulation, Chapter 4. P.15 ~ P.51, Wiley, 1997
[3] 타원곡선 암호시스템에 관한 고찰, 이인수, 박성준, 정보보호뉴스 1998년 6월호 (통권 12호)
[4] http://ourworld.compuserve.com/homepages/bob_jenkins/isaacafa.htm
[5] George Marsaglia's Homepage, http://stat.fsu.edu/~geo