Analysis of Random Number Generators

Parijat Naik
Department of Computer Science
Oregon State University
naikpa@cs.orst.edu

Abstract— **In this paper, we compare the different methods of Random number generation used in practice and a comparison of their efficiency. The paper focuses on the techniques used for RSA, SHA-1 and MD5. The results also provide a basis for a better Crypto Coprocessor.**

I. Introduction

Random number generators are used for generating an array of numbers that have a random distribution. The random number generators used in practice do not actually generate numbers, which are random. These programs use deterministic algorithms and are actually pseudo-random number generators, because the arrays of numbers produced are deterministic and we can only approximate random numbers.

Random number generators require the user to specify an initial value, or seed. Initializing the generator with the same seed will give the same sequence of random numbers. If we want a different sequence, we just initialize using a different seed. Some generators come up with different seeds by initializing using the clock time, but this is not recommended since it is then not possible to reproduce the results.

Since random number generators do not produce truly random sequences, the generator used may affect the results. Many widely used random number generators have been shown to have quite poor randomness properties that lead to incorrect results in certain applications. It is better to use a random number generator that has been thoroughly tested and recommended. For applications in which random numbers are only used occasionally, the quality of the generator will probably not matter, however in applications which use a lot of random numbers, such as Monte Carlo simulations, the quality of the generator is crucial and poor generators can produce incorrect results.

A. True Random Number Generators

Security protocols heavily depend on the randomness of keys; RNG for cryptographic applications must be such that the third party should not be able to make any predictions about outputs. Most of the existing programs use strong cryptographic algorithms, an anomaly in this process is that they all start from the random seeds that are not truly random weakening the overall system robustness in security. Some examples are using different timing of system hard disk drives, system status register, two different clocking speeds, movement of the mouse, number of current processes being executed, etc.

A random number generator should use non-deterministic finite automation to generate truly random numbers. This task is usually associated with an analog source. The usual method is to amplify noise generated by a semi-conductor diode and feed it to a Schmitt trigger, the output is sampled to get a series of statistically independent bits. These are then grouped as bytes giving us a random distribution.

White noise is a sound with an irregular, random waveform. When all audible sound frequencies are represented with the same strength it is called white noise. This designation is in analogy to vision: white light contains all visible frequencies of light. Such sources can generate true random numbers, but are difficult to embed in a chip.

B. Pseudo Random Number Generators

Pseudo Random Number Generators use an algorithm to generate numbers, which behave like genuine random numbers, used for simulations of random processes and statistical methods. Usually a good pseudo-random number generator should work, as one would expect a true random generator to work. Perfect randomness is a scarce resource, Currently pseudo random generators for all cryptographic applications it is sufficient to use pseudo random bits.

A PRBG can be considered to be cryptographically strong if it passes all polynomial time statistical tests. A PRGB is provably secure, if its security can be reduced to a well-established conjectured hard problem. Pseudo-random number generators are often based on cryptographic functions like block ciphers or stream ciphers. For instance, iterated DES encryption starting with a 56-bit seed produces a pseudo-random sequence.

A PRBG provides applications with a stream of numbers having properties for system security: It should be impossible for a third party to predict the output of the random number generator even with knowledge of previous output. The generated numbers should not have repeating patterns which means the PRNG should have a very long cycle length. A PRBG is normally just an algorithm where the same initial starting values will yield the same sequence of outputs.

C. Practical Constraints

One should understand that the numbers generated by current random number generators are not random, but are instead pseudo-random. Since PRBG's use deterministic algorithms, they cannot be completely uncorrelated, thus as the numbers they produce are reproducible, they cannot be truly random. However most of todays applications

that use random numbers require that the random number generator be of highest quality, such that they produce sequences that satisfy the standard statistical tests for randomness. [1][2] show that even if random number generators satisfy statistical tests for randomness, they have been sometimes found to be unreliable, especially for certain Monte Carlo applications. Such problems with PRBG's on sequential, as well as vector computers, get worse on parallel supercomputers where parallel PRBG's are required. Parallel PRBG's can probe other qualities of random number generation, such as inter-processor correlations, which do not appear on serial random number generators.

II. IBM 4758

The IBM 4758 Secure Crypto Coprocessor is a hardware card that plugs into PCI slots. The Coprocessor secure processing environment contains a 486-compatible micro coprocessor; it contains hardware to perform DES, random number generation, and modular math functions for RSA and similar public-key cryptographic algorithms. Secure code loading that enables updating of the functionality while installed in application systems. IBM Common Cryptographic Architecture (CCA) and PKCS ♯11 as well as custom software options. See Figure for a complete list of specifications. It also has protective shields, sensors, and control circuitry to protect against a wide variety of attacks against the system. More specifically the 4758 is protected against attacks involving probe penetration, power sequencing, radiation and temperature manipulation, consistent with the FIPS 140-1 Level 4 Certification. The basic element of the protective layer is a grid of conductors, which is monitored by circuitry that can detect changes in the properties of the conductors. The conductors themselves are non-metallic and closely resemble the material they are embedded in. This makes discovery, isolation and manipulation all the more difficult. These grids are arranged in several layers and the sensing circuitry can detect accidental connections between layers as well as changes in an individual layer. The sensing grids are made of flexible material and are wrapped around and attached to the secure processor package.[3]

DES	Hardware Support
RSA,DSS	Software with hardware Support
Hashing	SHA - 1 in hardware
Random Numbers	Noise based hardware RNG

TABLE I
Cryptographic Features of the IBM 4758

The Coprocessor generates a unique public key pair in the last production step, which is stored in the device. The tamper detection circuitry is activated at this time and remains active throughout the useful life of the Coprocessor, protecting this private key, as well as all other keys and sensitive data. The Coprocessor public key is certified at the factory by a global IBM private key and the certificate is retained in the Coprocessor. Subsequently, the Coprocessor private key is used to sign the Coprocessor status responses which in conjunction with the public key certificate, demonstrate that the Coprocessor remains intact and is genuine. From the time of manufacture, if the tamper sensors are ever triggered, the Coprocessor zeros its critical keys, destroys its certification, and is rendered inoperable.

A. The Algorithm

$$RG_{n,c} : Z_{p-1} \rightarrow Z*_p \tag{1}$$

$$RG_{n,c}(s) = \hat{g}^{(sdiv2^{n-c})} g^{s1} mod p \tag{2}$$

We do a modular exponentiation in $Z*_p$ with base g, after zeroing the bits from 2, ...,n-c of the input s. The function RG induces a distribution over $Z*_p$ in the usual way. We denote it with RG_n, c, the following probability distribution over $Z*_p$

$$ProbRG_{nc}[y] = Prob[y = RG_{n,c}(s); s \leftarrow Z_{p-1}] \tag{3}$$

Thus the algorithm receives the random element s in Z_{p-l} as a seed and then it iterates the function RG on it. The pseudo-random bits outputted by the generator are the bits ignored by the function RG. The output of the function RG serves as the new input for the next iteration.

B. Efficiency Analysis

According to the analysis conducted by [4], taking n = 1024 and c = 160, We obtain 863 bits making roughly 240 multiplications, yielding a rate of about 3.5 bits per modular multiplication. The modular exponentiations are all computed over the same basis $\hat{g}$. This feature allows us to precompute powers of $\hat{g}$ and store them in a table, and then use these Values to compute $\hat{g}$'s for any s.[3]

Thus using the choice of parameters for 160-bit exponents, we can get roughly 40 multiplications with a table of only 12 Kbytes. Yielding a rate of more than 21 pseudo-random bits per multiplication. A large memory implementation (300 Kbytes) yields a rate of roughly 43 pseudo random bits per multiplication.

III. Intel 82802 Firmware Hub

The Intel Firmware Hub integrates a Random Number Generator that utilizes thermal noise generated as a result of the inherently random quantum mechanical properties of silicon, in order to modulate a proven hardware RNG design. Internal circuitry is included to enhance the entropy of the output. Since the output of the RNG is non-deterministic, it is an excellent choice for cryptography applications, but it also is a convenient source of random numbers for mathematics, modelling, graphics algorithms, artificial intelligence, entertainment, and many other applications. The fact that it is a component of the platform and may be utilized remotely on a locked-away server makes it

an ideal (and much more reliable) source of entropy for applications that, in the past, have relied exclusively on a key press or other environmental input.[5]

This Random Number Generator (RNG) is designed to fill an 8-bit register, a bit at a time, with hardware-generated random numbers. When this register is full, a flag bit in the RNG data status register transitions to a 1, indicating that a valid random number is ready to be read. This bit will immediately reset to 0 upon reading the RNG data register.

A. Random Number Generation

The Intel RNG is based on the four tests whose results were evaluated by Intel and used in the 8202[3]

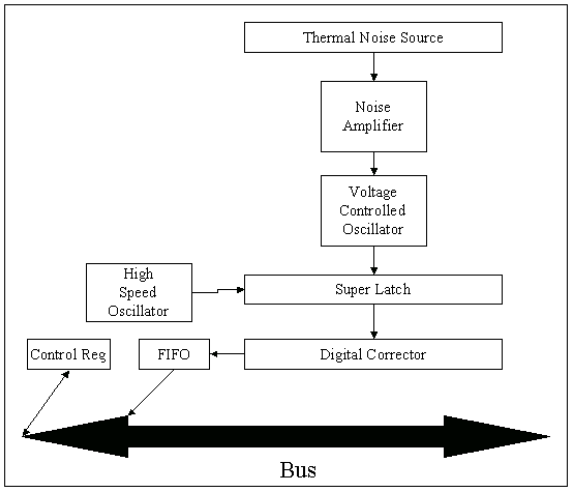

Fig. 1. Intel RNG Block Diagram

A.1 Noise Source

The Intel RNG primarily samples thermal noise by amplifying the voltage measured across undriven resistors. In addition to a large random component, these measurements are correlated to local pseudorandom environmental characteristics, such as electromagnetic radiation, temperature, and power supply fluctuations.

A.2 Dual Oscillator Architecture

The Intel RNG uses a random source that is derived from two free-running oscillators, one fast and one much slower. The thermal noise source is used to modulate the frequency of the slower clock. The variable, noise-modulated slower clock is used to trigger measurements of the fast clock. Drift between the two clocks thus provides the source of random binary digits.

A.3 Digital Post Processing

The initial random measurements are processed by a hardware corrector based on a concept proposed by John von Neumann to produce a balanced distribution of 0 and 1 bits.11 A von Neumann corrector converts pairs of bits into output bits by converting the bit pair [0,1] into an output 1, converting [1,0] into an output 0, and outputting nothing for [0,0] or [1,1].

A.4 Statistical Evaluation

Wide arrays of statistical tests were performed both before and after the digital post-processing. Tests on pre-corrected data help to identify characteristics that might be difficult to detect after the correction process. All statistical tests were performed on data prior to the software librarys SHA-1 mixing, as the SHA operation would mask nonrandom characteristics.

A large number of generalized statistical tests for randomness have been proposed, such as the DIEHARD12 specification, FIPS 140-113,and Knuths14 tests. The test suite for the Intel RNG included the following: Block Means Spectral analyzes Random walk test Block Mean correlations, 1-129 Block means Periodogram Spectral analyzes; hi, med, lo smoothing Spectral analyzes, adjusted for correlations autocorrelations, blocking and no blocking 8,16-bit Maurer test 4,8,16-bit Monkey test 4,8,16-bit Goodness of Fit Komolgorov-Smirnov test of trend CR/LF test Overall mean Column means Run length variances FIPS 140-1 test suite

B. Efficiency Analysis

Federal Information Processing Standards (FIPS) are issued by the National Institute of Standards and Technology (NIST), the body responsible for developing technical standards and guidelines for federal information resources. The Secretary of Commerce for use throughout the federal government to protect sensitive, unclassified information approves these standards and guidelines. FIPS 140-1 covers security requirements for cryptographic modules and specifies recommended statistical tests for random number generators. Each of the tests is to be performed on 20,000 consecutive bits of output from the random number generator. If any of the four tests fail, then the random number generator is deemed to be in an error condition.

IV. Efficiency Comparison

This section compares the two RNG's, we performed two theorotical tests; discrepancy and the spectral test.

A. Development of a Fast Discrepancy Code

The uniformity of a sequence of points is measured in terms of its discrepancy.
The L∞ discrepancy is defined by:

$$D_N = \frac{SUP}{JeE} \mid R_N(J) \mid \tag{4}$$

The L2 discrepancy is defined by

$$T_N = [\oint_{(x,y)eI^{2d}, x_1<y_1} R_N(J(x,y))^2 dxdy]^{\frac{1}{2}} \tag{5}$$

Where RN(J) is the Monte Carlo quadrature error in measuring the volume of subset J of Id. Similarly, the star

discrepancy is defined over rectangular sets with one vertex at 0. Usually the computation of the L2 discrepancy of N d-dimensional points is $O(N^2)$. The computation of the L∞ discrepancy of N d-dimensional points is $O(N^d)$. The project is to try to find an efficient way to reduce the computational cost of discrepancy to measure the uniformity of multi-dimensional random sequence.[6]

B. Development of A Fast Spectral Code

Spectral test is one of the theoretical tests that provides a way to check the quality of RNGs with a lattice structure such as linear congruent RNGs. Compared with theoretical test, empirical testing consumes more time and resources, also we will have to carry out these tests with samples of similar size and in similar dimensions as in our original simulation. For this reason, theoretical figures of merit have been designed that allow to assess RNGs. The spectral test embodies aspects of the both the empirical and theoretical tests because it deals with properties of the full period of the sequence also requires a computer program to determine the results. The spectral test quantity has a nice geometric interpretation which computes the maximal distance between successive parallel hyper planes covering all possible points Xn that the RNG produces. This interpretation leads to very efficient algorithms to compute the value of the spectral test.[6]

C. Overall Results

Tests were performed by Cryptography Research on at least 80 megabits of continuous RNG output. The tests identifying any statistically significant deviation from values expected from a perfect random source were minor deviations in tests involving spectral analysis. The analysis by Cryptography Research placed particular emphasis on areas that would identify statistical biases or failure modes in the RNG's. The largest nonrandom characteristic detected in both the generators was the bias in the ratio of 0 and 1 bits. In devices operating under extreme environmental conditions, bit frequencies were observed on the order of .510-2. Data with a 1 bias has 0.9997 bits of entropy per bit. Applications directly accessing the RNG's should make more conservative assumptions about the output quality. Although their estimates indicate that the hardware provides over 0.999 bits of entropy per output bit, a conservative assumption of bit of entropy per output bit can generally be used without any significant performance impact.

References

[1] A. De Matteis and S. Pagnutti, "Long-range correlations in linear and nonlinear random number generators," *Parallel Computing*, vol. 14, pp. 207–210, 1990.

[2] T. L. Jordan B. Smith P. Frederickson, R. Hiromoto and T. Warnock, "Pseudo-random trees in monte carlo," *Parallel Computing*, vol. 1, pp. 175–180, 1984.

[3] N. Koblitz, "The intel random number generator," *Cryptography Research, Inc. White Paper Prepared For Intel Corporation*, Apr. 1999.

[4] J. Dyer N. Howgrave-Graham and R. Gennaro, "Pseudo-random number generation on the ibm 4758 secure crypto coprocessor," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, Ç. K. Koç and C. Paar, Eds. 1999, Lecture Notes in Computer Science No. 2162, pp. 93–101, Springer, Berlin, Germany.

[5] Intel Corporation, "Intel 82802ab/82802ac firmware hub (fwh) datasheet," *Cryptography Research, Inc. White Paper Prepared For Intel Corporation*.

[6] Robert Davies, "True random number generators," http://webnz.com/robert.