THE INTEL® RANDOM NUMBER GENERATOR

CRYPTOGRAPHY RESEARCH, INC. WHITE PAPER PREPARED FOR INTEL CORPORATION
Benjamin Jun and Paul Kocher
April 22, 1999

1. Introduction

Good cryptography requires good random numbers. This paper evaluates the hardware-based Intel Random Number Generator (RNG) for use in cryptographic applications.

Almost all cryptographic protocols require the generation and use of secret values that must be unknown to attackers. For example, random number generators are required to generate public/private keypairs for asymmetric (public key) algorithms including RSA, DSA, and Diffie-Hellman. Keys for symmetric and hybrid cryptosystems are also generated randomly. RNGs are also used to create challenges, nonces (salts), padding bytes, and blinding values. The one time pad – the only provably-secure encryption system – uses as much key material as ciphertext and requires that the keystream be generated from a truly random process.

Because security protocols rely on the unpredictability of the keys they use, random number generators for cryptographic applications must meet stringent requirements. The most important is that attackers, including those who know the RNG design, must not be able to make any useful predictions about the RNG outputs. In particular, the apparent entropy of the RNG output should be as close as possible to the bit length.

According to Shannon[1], the entropy H of any message or state is:

$$H = -K\sum_{i=1}^{n} p_i \log p_i ,$$

where p_i is the probability of state i out of n possible states and K is an optional constant to provide units (e.g., $1/\log(2)$ bit). In the case of a random number generator that produces a k-bit binary result, p_i is the probability that an output will equal i, where $0 \le i < 2^k$. Thus, for a *perfect* random number generator, $p_i = 2^{-n}$ and the entropy of the output is equal to k bits. This means that all possible outcomes are equally (un)likely, and on average the information present in the output cannot be represented in a sequence shorter than k bits. In contrast, the entropy of typical English alphabetic text is 1.5 bits per character.[2]

An RNG for cryptographic applications should appear to computationally-bounded adversaries to be close as possible to a perfect RNG. For this review, we analyze whether there is any feasible way to distinguish the Intel RNG from a perfect RNG.

2. Pseudorandomness

Most "random" number sources actually utilize a pseudorandom generator (PRNG). PRNGs use deterministic processes to generate a series of outputs from an initial seed state. Because the output is purely a function of the seed data, the actual entropy of the output can never exceed the entropy of the seed. It can,

[1] Shannon, C.E., "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, p. 379-423, July 1948.

[2] Menezes, Oorschot, and Vanstone, *Handbook of Applied Cryptography*, Ch.7, CRC Press, 1997.

however, be computationally infeasible to distinguish a good PRNG from a perfect RNG.

For example, a PRNG seeded with 256-bits of entropy (or one with a 256-bit state) cannot produce more than 256 bits of true randomness. An attacker who can guess the seed data can predict the entire PRNG output. Guessing a 256-bit seed value is computationally infeasible, however, so it is possible that such a PRNG could be used in cryptographic applications. Examples of PRNGs designed for cryptographic applications include MD5Random and SHA1Random (which respectively hold 128 bits and 160 bits of state) in the BSAFE™,[3] cryptographic toolkit.

Although properly-implemented and seeded PRNGs are suitable for most cryptographic applications, great care must be taken in the development, testing, and selection of PRNG algorithms. It is critical that a PRNG be properly seeded from a reliable source. For example, most PRNGs included in standard software libraries use predictable seed or state values or produce output that can be distinguished from random data.

3. The Need for TRNGs

A true random number generator (TRNG) uses a non-deterministic source to produce randomness. Most operate by measuring unpredictable natural processes, such as thermal (resistance or shot) noise, atmospheric noise, or nuclear decay. The entropy, trustworthiness, and performance all depend on the TRNG design.

A PRNG by itself will be insecure without a TRNG for seeding. Seeding requires a source of true randomness, since it is impossible to create true randomness from within a deterministic system.

On computers without a hardware RNG, programmers typically try to obtain entropy for seed data using existing peripherals. The most common techniques involve timing user processes, but these methods are awkward and slow. For example, PGP[4] version 6.02 requires that users spend about 15 seconds entering random keystrokes or mouse movements to produce a new key. Methods involving user timing require inelegant user interfaces and cannot be used (or become insecure) when controlled by automated scripts. Some applications use hard drive seek times, but factors such as the drive technology, disk caches, and system timer resolution limits require careful consideration. Measurements of system activity, delays, configuration data, etc. are also sometimes used.

Overall, true random number generators implemented using conventional hardware tend to be slow, difficult to implement, require user involvement, and often provide unknown amounts of true entropy. These methods also make assumptions about the hardware that are not guaranteed. For example, operation timing measurements may not contain the expected amount of randomness under all system configurations. Techniques that rely on user events may not be reliable in unattended systems such as servers.

> *Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.*
>
> *– John Von Neumann (1951)*

Even though it is possible for applications to produce their own secure random data, many do not. Reviews by Cryptography Research frequently identify weaknesses in random number generation. Similarly, Christopher Allen and Tim Dierks of Consensus Development report that "most common problems [found in software security reviews] were related to random number seeding."[5] Bruce Schneier writes, "Good random-number generators are hard to design, because their security often depends on the

[3] BSAFE is a software toolkit available from RSA Data Security, Inc.

[4] PGP is available from Network Associates, Inc.

[5] Dierks, T., and Allen, C., "Lessons from Doing Source Code Reviews of Commercial Products," Consensus Development, 1997.

particulars of the hardware and software. Many products we examine use bad ones."[6]

Ian Goldberg and David Wagner found that Netscape's random number generator seed was derived from "just three quantities: the time of day, the process ID, and the parent process ID. Thus, an adversary who can predict these three values can apply the well-known MD5 algorithm to compute the exact seed generated."[7] Although most RNG flaws are not reported, problems are common, partly because cryptographic software libraries often leave it to programmers to find reliable sources of seed material.

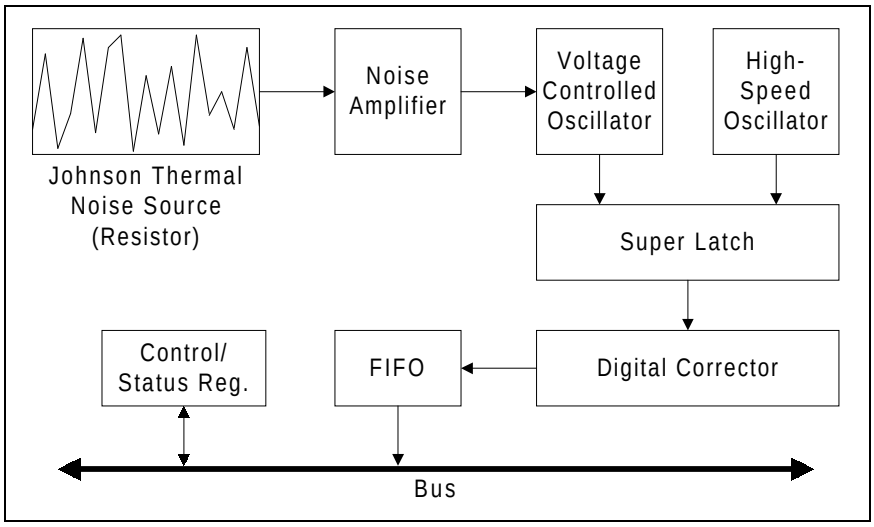

Figure 1: Block diagram of the Intel RNG

In many cases, system designers are faced with a trade-off between security and convenience. For example, to avoid having to collect fresh seed data each time the program loads, many software applications store their seed material on the hard drive where there can be a risk of compromise. The best sources for unpredictable data involve timing user processes, which add undesirable complexity to user interfaces.

4. Architecture Analysis

Measuring randomness is as much a philosophical debate as it is a mathematical issue. An infinite sequence of random numbers will carry a known statistical distribution. It is impossible, however, to prove whether any finite set of numbers is actually random. For example, the 128-bit value "`0`" is just as likely to occur as hexadecimal "`7c26b1b7f931eedb1f7e-e1b84764ae93`". Although it is impossible to prove randomness, we have analyzed Intel's design assumptions, design, and testing procedures, as well as performed statistical tests on RNG output data.

For this review, Cryptography Research performed a series of tests and evaluated the results of experiments performed by Intel. Raw data and design specifications for the analysis were provided by Intel.

4.1. Noise Source

Johnson noise (commonly referred to as thermal noise), shot noise, and flicker noise are present in all resistors. They have electrically measurable characteristics and are the result of random electron and material behavior.

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[8], such as electromagentic radiation, temperature, and power supply fluctuations. The Intel RNG significantly reduces the coupled component by subtracting

[6] Schneier, B., "Security Pitfalls in Cryptography," Counterpane Systems, 1998.

[7] Goldberg, I. and Wagner, D., "Randomness in the Netscape Browser," Dr. Dobb's Journal, January 1996.

[8] For an introduction to physical noise sources and externally coupled noise sources, see section 7.11 of Horowitz, P. and Hill, W., *The Art of Electronics*; Cambridge, MA: Cambridge University Press, 1980.

the signals sampled from two adjacent resistors.

Circuitry used for amplification or signal handling should preserve as many of the components of randomness as possible. Intel's design was found to have a high degree of linearity and bandwidth for passing high frequency signals to later stages.

4.2. 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. Similar RNG designs using independent oscillators are well known.[9,10]

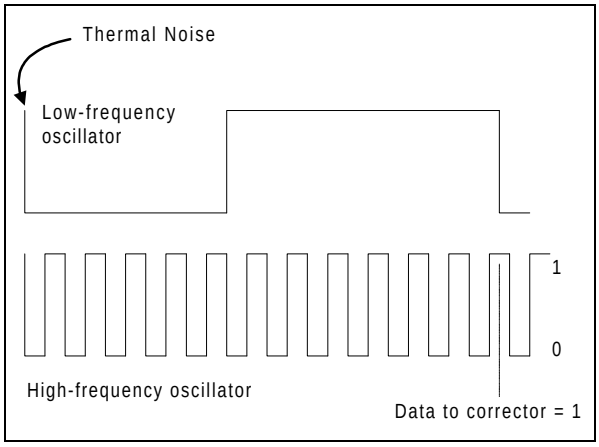

Figure 2: Overview of dual-oscillator design (frequency ratio not to scale)

The slow oscillator frequency must be significantly perturbed by the noise source, in addition to any pseudorandom environmental, electrical, or manufacturing conditions. Recorded histograms of modulated frequency resemble a normal distribution. The modulated frequency has standard deviation that spans approximately 10-20 high frequency clock periods; indicating that the sampling process is significantly varied by the random source. Provided that a significant random component is present in the low-speed oscillator, additional nonrandom effects should not reduce the quality of the RNG output. For example, environmental interference should only add additional unpredictability to the results. Intel reports that the modulation bandwidth is approximately 2 times the variable oscillator's center frequency. This fast response preserves useful components of high frequency noise from the random source.

The oscillators used in the Intel RNG have center frequency ratios on the order of 1:100. If both run without drift, the sampled bits could be "colored" with beats periodicity at multiples of the ratio. Statistical tests were used to try to locate such beats in the sampled bitstream, but could not be detected in over 10^8 output bits.

4.3. 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]`.

Input bits	Output
0,0	none
0,1	1
1,0	0
1,1	none

Figure 3: Von Neumann corrector

[9] Velichko, S. "Random-number Generator Prefers Imperfect Clocks." *EDN Access*, 1996. (http://ednmag.com/reg/1996/112196/23_di04.cfm).

[10] Hoffman, Eric. *Random Number Generator*, 1996, U.S. Patent 5,706,208.

[11] The hardware corrector design is patent pending by Intel Corporation.

The corrector is a simple way to generate statistically balanced outputs from data with residual bias. In particular, the corrector prevents imbalances in the fast clock's duty cycle from biasing the output. Intel has enhanced the von Neumann corrector to reduce the effect of any bit to bit correlations that might exist in the dual oscillator source.

One consequence of the corrector is that the RNG has a variable bitrate. The corrector generates an average of one bit for every 6 raw binary samples. The RNG's exceptional performance (over 75 Kbit/sec after the corrector) exceeds the TRNG requirements for all standard cryptographic applications, but the variable rate could complicate some esoteric usage scenarios. While it is not possible to prove that the RNG will produce an output bit in any finite time period, the probability of a long delay is negligibly small.

Corrector output is queued in a 32-bit register and provided to the software layer. The interface assures that random outputs cannot be read twice and that each read operation returns fresh random data.

> *Is there any hope for strong portable randomness in the future? There might be. All that's needed is a physical source of unpredictable numbers. A thermal noise or radioactive decay source and a fast, free-running oscillator would do the trick directly. This is a trivial amount of hardware, and could easily be included as a standard part of a computer system's architecture… All that's needed is the common perception among computer vendors that this small additional hardware and the software to access it is necessary and useful.*
>
> – Eastlake, Crocker, and Schiller , "RFC 1750: Randomness Recommendations for Security," *IETF Network Working Group*, December 1994.

4.4. Statistical Evaluation

Because subtle output correlations are always a possibility, the verification process has included a wide array of statistical tests by Cryptography Research and by Intel. These tests are designed to detect nonrandom characteristics by comparing statistical distributions in large samples of actual RNG outputs against distributions expected from a perfect random source.

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 library's 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 DIEHARD[12] specification, FIPS 140-1[13], and Knuth's[14] tests. The test suite for the Intel RNG included the following:

- *Block Means Spectral analyses*
- *Random walk test*
- *Block Mean correlations, 1-129*
- *Block means*
- *Periodogram*
- *Spectral analyses; hi, med, lo smoothing*
- *Spectral analyses, 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*

Tests were performed on at least 80 megabits of continuous RNG output. Major tests such as the autocorrelation and bit frequency tests were run at a variety of extreme

[12] Marsaglia, George. DIEHARD Statistical Tests, Florida State University.

[13] Federal Information Processing Standards Publication 140-1, "Security Requirements for Cryptographic Modules," U.S. Department of Commerce/NIST, Springfield, VA: NTIS, 1994.

[14] Knuth, Donald E. *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2, ch. 3, Addison Wesley Longman, 1998.

environmental conditions and on devices produced at extremes of manufacturing process parameters. All of these tests were performed at confidence level of 1% and 5%. The only 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. Cryptography Research performed tests to detect biases in pre- corrector data.

As expected, significant biases are present before the corrector. By far the largest nonrandom characteristic detected 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 $.5 \pm 10^{-2}$. Data with a 1% bias has 0.9997 bits of entropy per bit. Higher biases are possible under other conditions or manufacturing parameters, but the von Neumann corrector will improve the output quality.

Applications directly accessing the RNG should make more conservative assumptions about the output quality. Although our estimates indicate that the hardware provides over 0.999 bits of entropy per output bit, a conservative assumption of ½ bit of entropy per output bit can generally be used without any significant performance impact.

4.5. Software Architecture

The Intel software library uses a mixing function based on the Secure Hash Algorithm (SHA-1).[15] SHA-1 constructions are widely used, are recommended in a number of literature sources[16,17,18], and are believed to be very strong. SHA-1 is an effective mixer because it combines variable size inputs to generate independent output bits with excellent statistical distributions. The cryptographic properties of SHA destroy any remaining statistical structure and make it computationally infeasible to recover the seed state.

The Intel Security Driver uses a SHA-1 mixer with 512 bits of state. The SHA-1 construction cryptographically combines all RNG output since SDK initialization. To produce each 32-bit output, 32 bits of fresh data from the RNG are supplied to the mixing function, as diagrammed below. The addition of new random output reflects solid conservative approach, but is not necessary as the cycle length of the mixing function should exceed 2^{200}.

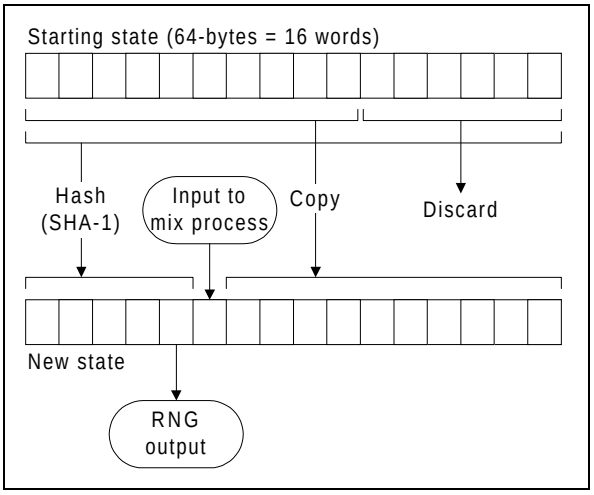

Figure 4: SHA-1 mixer design

Cryptography Research has performed a review of the SHA-1 code, and has tested the implementation of the SHA-1 and mixer.

[15] Federal Information Processing Standards Publication 180-1, "Secure Hash Standard," U.S. Department of Commerce/NIST, Springfield, VA: NTIS, 1995.

[16] Ellison, C., "Cryptographic Random Numbers", http://www.clark.net/pub/cme/P1363/ranno.html.

[17] Eastlake, Crocker, and Schiller, "RFC 1750: Randomness Recommendations for Security," *IETF Network Working Group*, December 1994.

[18] Federal Information Processing Standards Publication 186, "Digital Signature Standard," U.S. Department of Commerce/NIST, Springfield, VA: NTIS, 1994.

4.6. Operational Testing

Intel's manufacturing and test process is designed to detect most component failures. As such, chances of independent RNG field failure should be very low. In situations where system security requires good randomness, careful testing and analysis of failure modes is advisable. For example, particularly sensitive applications can independently test the integrity of the random source.

In the specific case of the Intel RNG, we believe the most likely failure modes cause the output to be "stuck" in one state (e.g., stuck on), causing no output from the von Neumann corrector. Failures could also cause the output from the corrector to stick in a fixed state. An oscillating state (which would produce an output of 101010101…) is theoretically possible but unlikely. Partial failures are more difficult to detect[19] and could reduce the entropy of the output, but many such faults would be fixed by the corrector.

Intel specifies that all hardware RNG units must pass the FIPS 140-1 randomness tests at time of manufacture. For added assurance, the Intel Security Driver also performs the RNG self-tests defined in the FIPS 140-1 cryptographic specification (monobit, runs, and poker) when the RNG is initialized. If desired, additional tests could be performed, but the existing test suite should detect most failures.

Many aspects of a system's operational security are, of course, beyond the scope of this review. While a good RNG is needed for good cryptographic security, other parts of a system can also fail. For example, properly seeding a defective PRNG does not provide a secure solution. It is also not possible to guarantee that manufacturing flaws, intentional sabotage, and other unexpected failures will never occur.

5. Conclusions

In producing the RNG, Intel applied conservative design, implementation, and testing approaches. Design assumptions about the random source, sampling method, system consistency, and algorithm appear appropriate. Careful attention was paid to analyze and avoid likely failure modes.

We believe that the Intel RNG is well-suited for use in cryptographic applications. Direct use of Intel's software libraries should simplify the design and evaluation process for security products. Alternatively, developers can combine data from the Intel RNG with data from other sources. For example, data from the Intel RNG can be safely exclusive-ORed with output from any independent RNG. The Intel RNG will help designers avoid relying on proprietary entropy gathering techniques in critical security routines. We believe the Intel RNG will prevent many RNG failures and improve the integrity and security of cryptographic applications.

Cryptographically, we believe that the Intel RNG is strong and that it is unlikely that any computationally feasible test will be found to distinguish data produced by Intel's RNG library from output from a perfect RNG. As a result, we believe that the RNG is by far the most reliable source of secure random data available in the PC.

[19] Knuth, D.E. *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2, ch. 3, Addison Wesley Longman, 1998.

About Cryptography Research

Cryptography Research provides technology and services to companies that build and use cryptography products. The company has a strong technical focus and is active in many areas of cryptosystem research. In 1998 alone, systems designed by Cryptography Research engineers protected more than a billion dollars of commerce and secured communications for financial, wireless, telecommunications, digital television, and Internet industries.

Benjamin Jun

Benjamin Jun (ben@cryptography.com) is a member of the research and engineering staff at Cryptography Research. As a cryptosystem architect, he has performed system architecture and hardware development on a number of secure and fault-resistant systems. Some of his recent efforts include the development and evaluation of secure content distribution systems.

Ben has also held positions at IDEO Product Development, Bain and Company, the National Institute of Standards and Technology, and the Institute for Defense Analysis. He holds BS and MS degrees in electrical engineering from Stanford University and is an NSF Graduate Fellow and a Mayfield Entrepreneurship Fellow.

Paul Kocher

Cryptography Research President and Chief Scientist Paul Kocher (paul@cryptography.com) has gained an international reputation for consulting work and academic research in cryptography. He has provided applied cryptographic solutions to clients ranging from start-ups to Fortune 50 companies. As an active contributor to major conferences and standards bodies, he has helped design many cryptographic applications and protocols including SSL v3.0.

His development of timing attacks to break RSA and other algorithms was widely received by the academic community. More recently, Kocher led research efforts that discovered Differential Power Analysis, developed new methods for securing smartcards against external monitoring attacks, and designed the record-breaking DES Key Search machine. His work has been reported in forums ranging from technical journals to CNN and the front page of the New York Times. Paul is also a co-founder and the Chief Scientist of ValiCert, Inc.

Contact Information

Cryptography Research, Inc.
870 Market Street, Suite 1088
San Francisco, California 94102-3002

http://www.cryptography.com
E-mail: info@cryptography.com
Telephone: 415.397.0123
Fax: 415.397.0127