Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 598246, 10 pages
doi:10.1155/2009/598246

Research Article

Random Number Generators in Secure Disk Drives

Laszlo Hars

Seagate Technology, 389 Disc Drive, Longmont, CO 80503, USA

Correspondence should be addressed to Laszlo Hars, laszlo@hars.us

Received 15 October 2008; Revised 19 March 2009; Accepted 9 June 2009

Recommended by Sandro Bartolini

Cryptographic random number generators seeded by physical entropy sources are employed in many embedded security systems, including self-encrypting disk drives, being manufactured by the millions every year. Random numbers are used for generating encryption keys and for facilitating secure communication, and they are also provided to users for their applications. We discuss common randomness requirements, techniques for estimating the entropy of physical sources, investigate specific nonrandom physical properties, estimate the autocorrelation, then mix reduce the data until all common randomness tests pass. This method is applied to a randomness source in disk drives: the always changing coefficients of an adaptive filter for the read channel equalization. These coefficients, affected by many kinds of physical noise, are used in the reseeding process of a cryptographic pseudorandom number generator in a family of self encrypting disk drives currently in the market.

1. Introduction

Cryptographic random number generators are employed in many embedded systems, like in self encrypting disk drives, such as the Seagate Momentus Full Disk Encryption (FDE) drives. The generated random numbers can be used for encryption keys, facilitating secure communication (via nonces), performing self-tests, and so forth. Previous states of the random number generator are difficult to securely store, because an attacker could read, and in some point in the future restore earlier states (together with any possible local authentication tags) with the help of specialized tools (spin stand), and so force the generation of the same random sequence as earlier. This causes repeated nonces, recurring use of the same encryption keys, and so forth, that is, loss of security.

Physical entropy sources are used to initialize cryptographic random number generators at every power up, and at special requests, like at reinitializing the firmware, or before generating long used cryptographic keys. Seeding with unpredictable physical values makes a cryptographic random number generator to supply pseudorandom sequences, with negligible probability of repetition. Generating secure random sequences this way needs no secure protected storage for keys or for the internal state of the generator, therefore it reduces costs and improves security.

Below we describe how an available digital signal with random components, the coefficients of the adaptive channel filter, is used in seeding a cryptographic random number generator in self encrypting disk drives. The estimation of the available physical entropy is discussed, resulting in an efficient seeding process. These should provide *confidence in the generated random numbers* for their users, and *tools for developers* of embedded random number generators in testing and evaluation of designs.

2. Disk Drive Architecture Overview

[1] The write and read transducers are mounted on the *head* which is separated from the rotating disk by an air bearing that keeps the read/write transducers at a distance of about 10 nm from the disk surface. The head is mounted on an *arm*, which is connected to an *actuator*. In 3.5″ disk drives this arm is about 5 cm long and prone to mechanical vibrations, affected by air turbulence while the drive is operating. The vibration in vertical direction influences the amplitude of the read signal, while the radial vibration affects the noise pattern from the granular structure of the magnetic particles and crosstalk from neighbor tracks, because of the small spacing between tracks (in the range of 10–100 nanometers).

To guide the head to remain on track, *servo patterns* are written on the disk. These servo patterns are organized in

radial spokes which are traversed by the head about 200 times per revolution (at 5400 rpm rotational speed 18 000 times per second). After the head crosses these servo patterns a controller evaluates the read signal and corrects the radial position accordingly. It also tunes the channel *equalizer filter* for optimum signal shape. The tracking correction is based on the current radial position, velocity, and acceleration of the head. These values are nondeterministic, strongly affected by turbulent airflow and mechanical vibrations. No one succeeded so far with a useful model of the disk drive physics. In [2] some equations are presented, but they do not give a reasonably accurate picture of disk drive internals.

3. Entropy Requirements

In this paper we show that disk drives can provide physical randomness for seeding cryptographic random number generators, but they are targets to specific attacks, exploiting their use and special characteristics, leading to disk specific entropy requirements. The generalized "birthday bound" tells that after taking $2^{n/2}$ samples there is a 50% chance of a uniformly distributed n-bit random variable to attain the same value more than once. In a data center an attacker could observe thousands of disk drives rebooting thousands of times, so $10^7 \approx 2^{23}$ samples from different random number sequence are easily taken. When these results are shared over a network, one could build a database from over 2^{32} initial sets of values of the random number generator, to search for a collision. It gives a requirement of at least 64 bit entropy of the seed. Of course, 50% chance of a successful attack is far too high. A commonly accepted allowable collision probability is 10^{-8} (half of the chance of hitting the jackpot in a 5-out-of-90 lottery), which adds 27 bits to the entropy requirements for the seed, so for unlikely repeated sequences the entropy of the seed has to be more than 90 bits. To account for HW differences, environment changes, and so forth, at least *128 bit entropy* is desired for the seed of a cryptographic random number generator.

The smallest AES cipher needs 128-bit fully unpredictable encryption keys, also posing the requirement of at least 128 bit seed entropy. (High entropy public keys and longer symmetric keys must be generated with several calls to a reseeded cryptographic random number generator.)

4. Entropy Sources in Rotating Disk Drives

There are many unpredictable physical processes, noise sources in disk drives. Economic constraints compel using electronic signals, which are available in digital form in standard unmodified disk drives, and which contain strong random components. At boot time, or at a special request they provide the entropy sources to seed an SW-based cryptographic random number generator of self encrypting disk drives, ensuring the uniqueness of the generated (pseudo)random sequences with very high probability.

In disk drives currently in the market several such sources are used. Combinations of their data improve the quality, the speed of the random number generation, and the safety against potential attacks influencing the entropy sources.

4.1. Timing Variations. In the disk drive electronics there are internal high speed *counters* available. Their least significant bits are sufficiently random when sampled during the disk boot up process, or in general, after actions involving a lot of mechanical activities of timing uncertainties, such as at spin-up and rotation of the motor and platters, and at arm movements in seek operations. These random bits can be collected into an entropy pool, consumed when needed. The entropy of the timing data is analyzed in [3].

Such random number generators have been published, for example, the slow [2], implemented externally in the host computer, where synchronous communication masks off most of the original timing variations.

4.2. Tracking Error. In [4] another randomness source was investigated, with the tracking error of the magnetic read head trying to remain in the middle of the path of the recorded data. Consecutive samples are strongly correlated, which limits the useable entropy. Our experiments in the newest generation of disk drives showed much less achievable speed or entropy/s than claimed in [4], but the position error of the read/write head certainly represents another alternative source of randomness.

4.3. Channel Filter Coefficients. The drive firmware can access the *coefficients of an adaptive channel-filter*, via a diagnostic interface between the main control ASIC and the channel signal processor, which also does the coding/decoding of the head signal [5]. The coefficients represent resistor values of an analog filter, continuously tuned by the control mechanism of the read/write channel chip to make the peaks of the output signal close to equally high. For details about the algorithms used in the channel equalization filters see [6] or [7]. The filter coefficients depend on the amplified head signal, containing many random components, including head noise; electronic noise; the effects of motor speed variations; internal air turbulence; the vibration of the head arm; the amplitude uncertainty due to the flight height variations of the read head; the actual path of the head over the track, influenced by the tracking errors and their corrections.

In the Momentus FDE drives there are 12 such coefficients accessible, each 8 bit long. Coefficient 11 is fixed as asymmetry compensation tap, set for each head and zone in the manufacturing process (it is included in the analysis below as sanity check for the algorithms, it does not provide randomness). The other coefficients are constantly adapted to the noisy distorted signal of the servo patterns.

When the random number generator is reseeded, seek operations are executed followed by a read from a fixed location. At least a full track worth of data affect the adaptive filter, and significant mechanical arm movements are involved. These translate to hundreds of changes in the adaptive channel filter, strongly influenced by physical noise; therefore, there will be very *little correlation* between consecutive acquired values of the same coefficients.

The noise in the read-back signal in modern disk drives is very high. In a disk drive under investigation the read-back

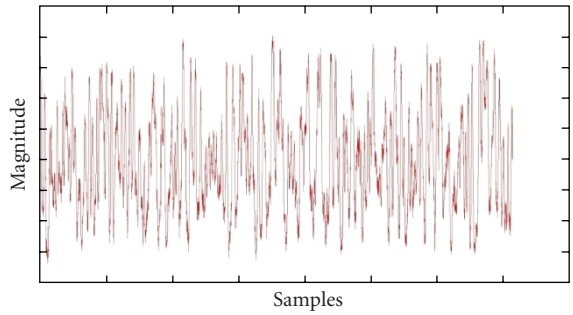

Figure 1: Noisy read-back signal.

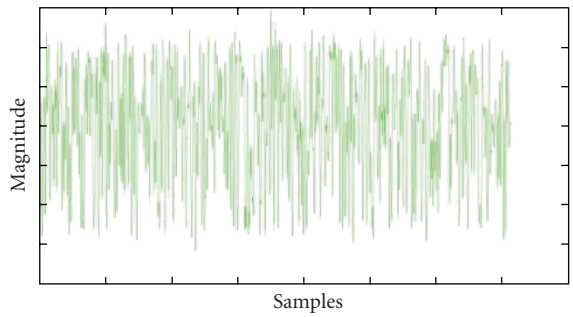

Figure 2: Signal after equalization.

signal was captured with a digital storage oscilloscope and depicted in Figure 1.

One can see wildly varying signal peaks. The adaptive equalization filter makes the height of these peaks more uniform, as shown in Figure 2.

History. The noise sources and levels have been extensively studied; see [2, 8–11]. Their effects on the signal in the read channel have also been investigated; see [12–15]. The resulting inherent randomness in the channel filter coefficients has been proposed to be used for random number generators in [16], but the included randomness extraction algorithm is very inefficient.

Below the nonrandom physical properties of the channel filter coefficients and their entropy estimation technique is discussed, then we describe a secure and efficient RNG implementation, taking into consideration the randomness requirements and the entropy of the randomness source under varying environmental conditions.

5. Entropy Estimation

We analyzed 22 data sets, 100 M coefficient bytes in each. They were collected in continuous sessions (performing two seek operations and reading the full track before data acquisition), from Seagate Momentus FDE disk drives of different capacities from different manufacturing sites, under varying environmental conditions (temperature 0°C, 20°C, 60°C; supply voltage 4.75 V, 5 V, 5.25 V). The samples were captured over a diagnostic port and recorded in another PC, not to influence the data collection.

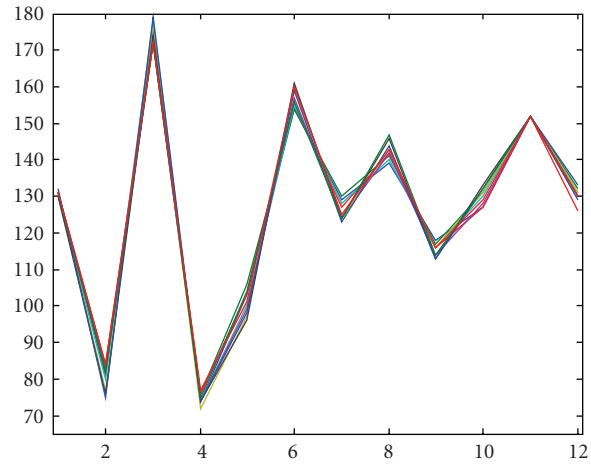

Figure 3: Coefficient changes in time.

There have been some non-random properties identified in the channel filter coefficient data, which have to be considered when the available entropy is estimated. In the sequel we will estimate the entropy as 16 bits in each block of coefficients (96 raw bits), which can be acquired in every 10 milliseconds. The yield is 1.6 K very high quality random bits per second.

We found no significant differences in the randomness between datasets, that is, the manufacturing process and environmental conditions do not considerably influence the available entropy. An attacker gains no exploitable information by examining a disk drive, over generally available data (collected from other drives), or influencing its working environment.

5.1. Data Dependencies. The graph of the 12 filter coefficients is of relatively stable shape in time. Figure 3 shows the curves of 10 consecutive captured sets of filter coefficents from the same drive, plotted on top of each other. The x-axis is the index of the filter coefficients (1–12), the y-axis is the value of the corresponding coefficient byte (P1–P12). A curve plotted in one color shows the 12 filter coefficient values of one sample set, connected by straight lines.

One can observe that at some places (i.e., between $x = 4$ and 5) these segments are almost parallel. It means that if P4 increases, P5 does, too, therefore, they are positively correlated. Other segments, like the ones between $x = 7$ and $x = 8$, cross each other at roughly the same point half way in between. It means that if P7 decreases, P8 increases by roughly the same amount. It is an indication of negative correlation between P7 and P8, therefore, the entropy of coefficient P7 and P8 together is not much larger than that of P7 alone, or the entropy of P4 and P5 together is close to the entropy of P5 alone. These point to a *potential* issue: the available entropy could be less than the estimates the coefficient samples provide in isolation. This autocorrelation is investigated in subsections 5.2 and 5.3 by statistical methods.

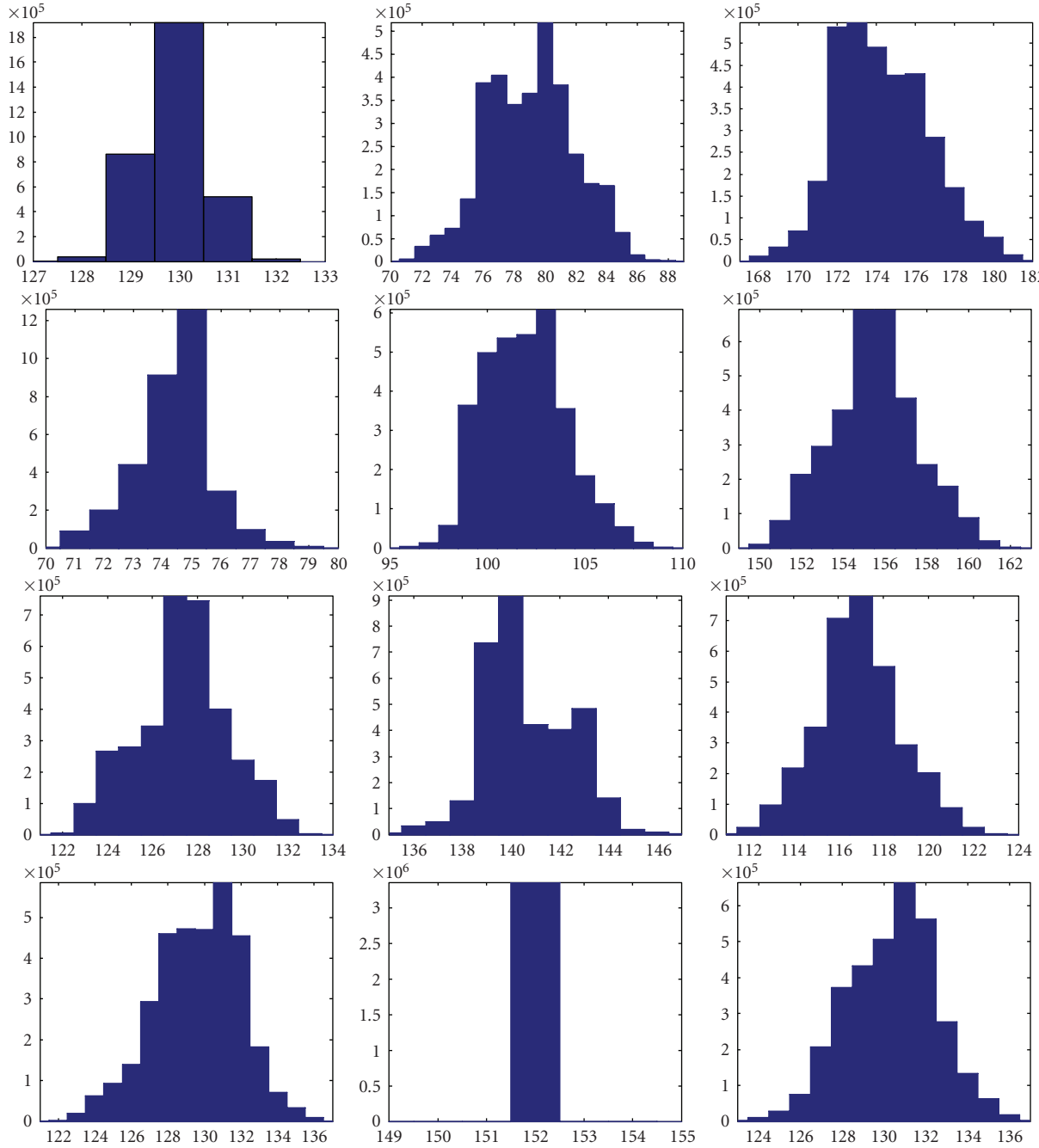

Figure 4: Histograms of the filter coefficients.

5.2. Coefficient Distribution. By plotting the histograms of each filter coefficient from contiguous measurement sequences of a disk drive (Figure 4) we can see that each individual coefficient attains only a few distinct values, and almost all their variability is preserved in their few least significant bits (bits [1, 2]—bits [1, 2, 3, 4]).

The widths of the bins (bars) help visually comparing the histograms. Curiously, the coefficients are not uniformly or normally distributed, but can be well approximated by the supperposition of two normal distribution (bell) curves, but it is irrelevant to our discussions.

5.3. Autocorrelation of Sequences of Individual Coefficients. We used the discrete Fourier transform of the same individual coefficient sequences described above to compute many autocorrelation values at once: $\mathcal{F}^{-1}(\mathcal{F}(x) \cdot \mathcal{F}'(x))$, where $\mathcal{F}(x)$ denotes the discrete Fourier transform of the sequence x, $\mathcal{F}'(x)$ is its transposed complex conjugate and $\mathcal{F}^{-1}(X)$ is its inverse. (It gives the same results as the direct method used by the MATLAB tstool/autocorrelation, but faster.) In Figure 5 the autocorrelation values are plotted for each of the 12 coefficient sequences, lags = 1–50.

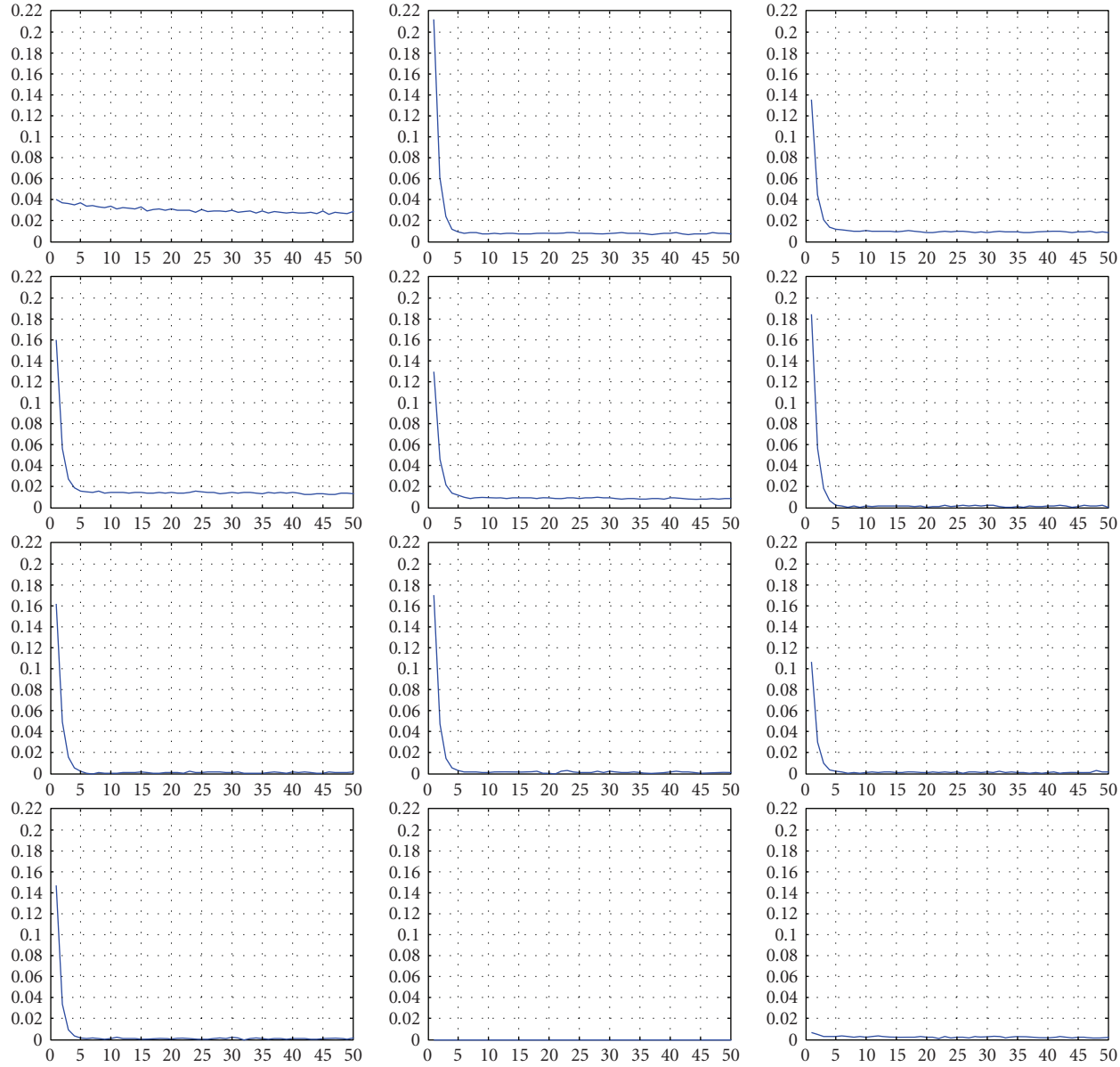

Figure 5: Autocorrelation of the filter coefficients.

None of the autocorrelation values exceeds 21%, with an average around 13%. The little residual (large lag) autocorrelation values are the artifacts of the very nonuniform distributions. The overall entropy loss is due to uneven distributions and short-term autocorrelation (which only causes a loss of a handful bits entropy). The hashing process described below will eliminate both problems.

5.4. Entropy of the Coefficients. Most of the filter coefficients carry about 3 bits of Shannon entropy: $\mathbf{H} = -\boldsymbol{\Sigma}\ \mathbf{p}_i\ \mathbf{log}_2(\mathbf{p}_i)$. The exceptions are coefficient 1 carries 1.5 bit, coefficient 2 does 3.5 bits , and coefficient 4 does 2.4 bits. If all of them were independent, the overall entropy of the 12 channel filter coefficient bytes could be 32 bits. Statistical tests below showed less actual randomness (16–24 bit), because of the correlation among them, and because of their internal autocorrelation.

5.5. Min-Entropy. Often the so-called min-entropy is better for estimating the security: $\mathbf{M} = -\mathbf{log}_2(\mathbf{max}(\mathbf{p}_i))$.

A distribution has a min-entropy of at least b bits if no state has a probability greater than 2^{-b}. It estimates the complexity of such attack strategies, when the attacker seeds his cryptographic random number generator (identical to the one in the disk drive) with the most likely coefficient values. If he finds a match, he guessed the seed right. If he does not, he reboots and checks the random numbers generated by the disk drive again, until the most likely filter coefficients appear to be the actual seed.

Seed byte	Entropy
1	1.491
2	3.536
3	3.266
4	2.378
5	3.082
6	3.104
7	3.018
8	2.765
9	2.967
10	3.268
11	0
12	3.144
Sum:	32.019

Figure 6: Individual entropies.

This attack is slow; it needs tens of seconds for each reboot. (Working on many identical disk drives, costing \$50–100 each, could speed up the process proportionally, but with a very large investment.) If instead an attacker feeds various possible values of the filter coefficients to a copy of the cryptographic random number generator, he can try millions of seed values in the time of one reboot. In this sense, the Shannon entropy measures better the security of physical randomness sources seeding a cryptographic random number generator in disk drives, but we have to make sure that the min-entropy is also reasonable, that is, no seed occurs at exploitable frequency (1 second trial/30 year: $\forall \mathbf{p}_i < 10^{-9}$).

5.6. Mix-Truncate (Hash) Entropy Estimation. The entropy estimation process is the following: hash the bits of each channel filter coefficient dataset ($12 * 8 = 96$ bits) to k bit output. Decrease k from 32 (the upper bound of the entropy from Figure 6) until the concatenated output blocks pass all commonly used randomness tests. Perfect hashing makes the distribution of the results more uniform and reduces the autocorrelations (see the appendix), in the costs of decreasing the number of random bits. (We used the SHA1 hash on zero-padded input and keeping the least significant k bits of its 160 digest bits. SHA1 has no known exploitable weakness in this mode: an attacker with reasonable resources cannot distinguish it from a perfect hash.)

There are other methods in use to shrink data to improve randomness. The first such published method was the Neumann corrector to remove bias [17], but more recent entropy amplification techniques are all related to hashing [18–20]. (A hash function maps arbitrary data to a fix range of integers, in such a way that simple structures of the input sequences are not preserved.)

The employed randomness tests are very sensitive to non-uniform distribution of k-bit blocks, but many other non-random properties are checked, too. When all the tests pass with a particular choice of k, we know that each possible k-bit block in the sequence of the hashed coefficient sets occurs at roughly the same number of times: *each hashed filter coefficient set appears independently, at about* 2^{-k} *frequency.* Consequently, no filter coefficient set occurs with probability much larger than 2^{-k}, that is the min-entropy of one coefficient set is about k. When n such independent blocks are used to seed the random number generator, an attacker has a search space of at least $2^{k \cdot n}$ elements when trying different seeds in a copy of the RNG to guess the seed of the disk drive (e.g., $n = k = 16$ gives about $2^{256} > 10^{77}$ seeds to try).

5.6.1. Justification of the Mix-Truncate Entropy Estimation. Our use of physical randomness justifies this hashing-then-statistical testing process, although *proving* true randomness is impossible from any finite number of input bits, for example, the bit sequence could be periodic with a period longer than the observed data, or all unseen bits could be 0. These cannot be ruled out by the observed data. We can only state that no evidence for nonrandomness was found.

Common statistical tests accept many cryptographically hashed non-random sequences as perfectly random, if the size of the hash output is large enough (greater than the binary logarithm of the length of the sequence). For example, if we hash the members of the sequence $0, 1, 2, \ldots, 10^9$ to more than 30 bits each, the result will pass all the standard statistical randomness tests, although the original sequence is clearly not random, and this non-randomness is apparent in the finite input data. Arbitrarily many similar pseudorandom sequences can easily be constructed, which fool the statistical randomness tests, even if we make certain assumptions about the data, like lack of autocorrelation.

However, *physical* considerations established that our sample blocks are *independent* to a great degree (which invalidates the pseudorandom counter examples above). Autocorrelation tests did not refute this claim. Note that the independence has physical reasons, not mathematically proven.

The proposed hashing process changes data blocks independently from each other, and so it does not introduce pseudorandomness, which would make the statistical test suites to accept hashed regular sequences. Hashing affects individual distributions and dependencies within data blocks. Even correlations between groups of coefficients are removed (see the appendix).

Statistical randomness tests check long-term nonrandomness, like that the hashed blocks do not repeat more often than true random blocks would, and there are no exploitable ways to guess the next block, having observed an arbitrary number of hashed blocks. These are sufficient for the security of seeding cryptographic pseudorandom number generators with the hashed data blocks, originated from sets of channel filter coefficients, separated by largely unpredictable mechanical events.

5.6.2. Security of Hashed Seeding of Pseudorandom Number Generators. When the analyzed sequence is used for seeding (cryptographic) pseudorandom number generators, we don't need uniform randomness of the seed blocks: but *large variability* (no one should occur with large probability), and

independence (seed blocks at any distance vary a lot). The later implies the former: if a block repeated often, autocorrelation would be large. Independence provides protection against an attacker, who records several generated random numbers and tries to derive seeds for an identical random number generator, to find a match. Our sets of seed blocks take a huge number of different values, and so an actual one cannot be guessed with a significant chance of success; identical sequences occur very rarely.

Low autocorrelation assures that no seed blocks occur frequently nor are some blocks correlated. Otherwise an attacker could find frequent blocks in another drive, or could modify spied out earlier seed blocks according to the property, which caused large autocorrelation. This would increase the chance of a successful guess of a seed, revealing all newly generated random numbers until a fresh seed is applied.

5.6.3. Hash Functions for Data Whitening. Physical random numbers almost always have to be whitened, because their distribution could be non-uniform and changing in time and with environmental conditions. Therefore, even for non-cryptographic applications the physical randomness source is usually hashed (corresponding to seeding pseudorandom number generators), although for lower security requirements there are much faster hash algorithms (e.g., the ones in [21]) than the secure hash functions used in cryptography (e.g., SHA1/2).

5.7. Randomness Tests. There are many randomness tests published, for example, [22–26]. A survey is in [27].

Diehard Test Suite. 15 different groups of statistical randomness tests were published by Marsaglia [23, 24]. This set of tests is probably the most widely used. Many different properties are tested and the protocol of the results is 17 pages long. The randomness measures are 250 P-values. The standard way for accepting a single p-value is to check if it is in a certain interval, like [0.01,0.99]. The difficulty with the interpretation of the Diehard test is to establish an overall acceptance criterion, because related tests are applied to the same set of data and so the results of the individual tests are correlated.

A procedure used in [28, 29] for testing the random number generator implemented in the Intel Pentium III chip works as follows. To come from a 95% confidence interval for each of the 250 test results the 5% confidence level is divided by 250, resulting in 0.02%. The Diehard test is considered to pass if all 250 P-values are in the corresponding interval [0.0001, 0.9999]. We adopted this acceptance criterion, with an additional check described in [4]: count the number of near-fails among the 250 Pvalues returned by the Diehard tests (those P-values which are not in [0.025, 0.975]). Since asymptotically the relative number of fails for the given interval is 5%, there must be about 12 near-fails among the 250 values. These near fails are expected, as the Diehard test suite states in the test protocol: "*Such p's happen among the hundreds that DIEHARD produces, even with good RNG's. So keep in mind that "p happens"*."

The Diehard (or the NIST) tests are not sensitive enough to autocorrelations, which occur at other than integer multiples of 8 bit offsets. (There are data sets, which pass the Diehard tests at $k = 28$, but failed with $k = 24$ reduction.) Therefore, only the tests of hashed filter coefficient sets to $k = 24$, 16, and 8 bits can be fully trusted. Some data sets proved to be sufficiently random with $k = 24$, but a few did not, while all of the Diehard tests passed on our every hashed channel filter coefficient sets at $k = 16$ or less.

NIST 800-22 Randomness Tests [26]. When the Diehard tests and Maurer's test passed on our hashed data, the NIST tests also accepted the input as random. The main advantage of the NIST test suite is that it works on data of size other than 10 MB, needed for Diehard, but our hashed files were large enough for Diehard. Each one of the NIST tests provides a P-value, and depending on the length of the sequence an acceptance threshold is provided. The ratio of accepted P-values for each test must be above a given level. For the tests to pass the collected P-values are assessed in the end, to verify their uniform distribution between 0 and 1, which is similar to the overall acceptance of Diehard.

Maurer's Universal Randomness Test. It was published in [25], and further investigated in [30]. The test analyzes the statistics of gaps between the closest occurrences of the same bit blocks. A test for each block size 1–16 was performed. Larger test blocks required huge datasets for high confidence in the test results. For example, the necessary size of the data sets for 16-bit test blocks is $1000 \cdot 2^{16} \cdot 12 \approx 800$ MB. All the Maurer tests with block sizes b=1–16 passed, when the data was hashed to $k = 16$. (Maurer's test was developed for stationary ergodic entropy sources with finite memory. In our case virtually no memory is present, because of the many seek-induced filter coefficient updates between data acquisitions.)

Autocorrelation. the MATLAB tstool/autocorrelation tool was used, and the results (one depicted in Figure 7) were compared to high quality pseudorandom data. All hashed channel filter coefficient dataset with $k = 24$ or less provided autocorrelation curves indistinguishable from that of uniform, true random data (we found roughly the same maximum, average, and standard deviation).

Transform-Tests. An *FFT*-test is included among the NIST tests. By computing the correlation of the hashed coefficient sequences to periodic signals (sine waves) the FFT test finds periodic components in the hashed data. The physical model and the observed level of autocorrelation in the individual coefficient sequences predict no periodic signal components, which was confirmed by these tests on every hashed channel filter coefficient dataset with $k = 24$ and 16.

Walsh Transform-Test. It finds other type of structured (pseudoperiodic) components in the data. The physical model and the observed level of autocorrelation in the individual coefficient sequences predict no significant signal

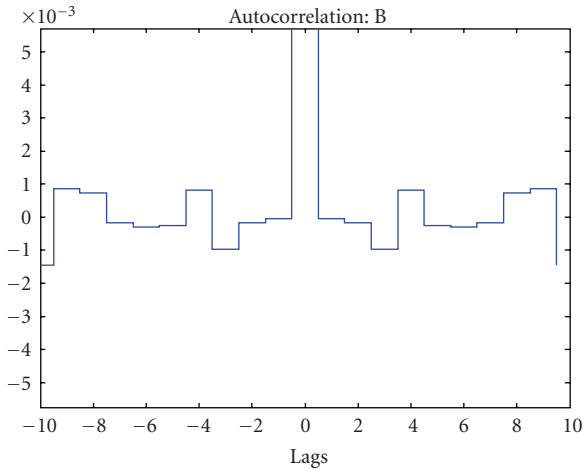

Figure 7: Autocorrelation of a 96 → 16 bit hashed coefficient sequence.

components of this type, either, which was confirmed by the Walsh transform tests on every hashed channel filter coefficient dataset with $k = 24$ and 16 (showing little deviation from the expected values).

6. The Cryptographic Pseudorandom Number Generator

With the techniques described in Section 5.6 we found that one channel filter coefficient datasets provides at least 16 bit entropy, therefore eight datasets are enough for our desired 128 bit entropy. In this section the algorithm is described, how the available physical randomness is converted to cryptographically secure random numbers. Incidentally, it also uses the SHA1 hash function.

Channel filter coefficients are collected as a background task. Eight datasets need all together about 80 ms (1.6 Kb/s), allowing 12 reseedings a second, which would only rarely be needed. By mixing in samples of a free running counter, additional randomness is gained and the safety improves against HW-based attacks trying to influence the channel filter coefficients. Four LS bits of each 8 sets of 11 channel filter coefficients, together with the counters, give 384 raw seed bits, used in two halves as XSEED values, in two iterations of the FIPS-186-2 generator.

The cryptographic random number generator specified in the FIPS-186-2 document [31] was used with SHA1 as hash function and 24-byte (192 bit) internal state. When x is a desired (160-bit) pseudorandom number (may be cut and the pieces combined for the requested number of bits), the following FIPS-186 algorithm generates m random values of x.

Step 1. Choose a new secret value for the seed key, $0 < \text{XKEY} < 2^{192}$.

Step 2. In hexadecimal notation let

$$t = 67452301\ \text{EFCDAB89}\ 98\text{BADCFE}\ 10325476\ \text{C3D2E1F0}. \quad (1)$$

This is the initial value for H0‖H1‖H2‖H3‖H4 in the SHA1 hash function. ("‖" is concatenation.)

Step 3. For $j = 0$ to $m - 1$ do

(a) XSEED_j = optional user input

(b) $\text{XVAL} = (\text{XKEY} + \text{XSEED}_j) \bmod 2^{192}$

(c) $x_j = \text{SHA1}(t, \text{XVAL})$

(d) $\text{XKEY} = (1 + \text{XKEY} + x_j) \bmod 2^{192}$.

6.1. Accumulated Entropy. The initial entropy of XKEY (the internal state of the cryptographic pseudorandom number generator) is 0 at boot up. After Step 3(d), regardless of the entropy of XSEED, the entropy in XKEY cannot increase to more than 160 bits (the length of the added x), stored in the LS (*least significant*) 160 bits of XKEY. In the next iterations only this LS 160 bits are further modified (disregarding a possible carry bit), therefore the accumulated entropy stored in XKEY increases very slowly beyond 160 bits.

During initialization (Step 1) we can choose a new secret value for XKEY. It can be anything (not specified), so we can use the current XKEY value after a few iterations of the random number generation, *shifted up* to fill its *most significant* (MS) bits. Subsequent calls of the RNG affect the LS bits of XKEY, keeping the initial entropy stored in the MS bits intact.

Accordingly, the seeding process can be performed in *two phases*. The first phase starts with an all 0 XKEY and uses half of the total number of seeding rounds to mix in the HW entropy. In the second phase we shift the LS 160 bits of the current XKEY to its MS bits and then perform the remaining rounds to mix in the rest of the HW entropy. During these steps the generated random numbers (x_j) are discarded, only the internal state (XKEY) is kept updated.

For accumulating more than 320 bit internal entropy (when XKEY is chosen longer than 40 bytes) one can execute more phases like the above. SHA1 limits the number of usable bits to 512, but if needed, it can be replaced by hash functions operating on larger (or on multiple) blocks.

6.2. Compression of the HW Seed. The format and content of the seeding data is not specified in the original FIPS-186-2 document, therefore preprocessing is allowed, and desirable. Keeping fewer LS bits of the filter coefficients (as many as necessary to preserve the entropy) each channel filter coefficient data set can be compressed to 40 bits, without significant computational work. The LS bits of free running counters are then attached. Several compressed blocks like these can be used concatenated in Step 3(a), speeding up the seeding process proportionally, by trading slow SHA1 hash operations for fast data compression steps.

7. Future Improvements

The FIPS-186-2 cryptographic pseudorandom number generator [31] could be replaced by one compliant to the NIST Special Publication 800-90: Recommendation for Random

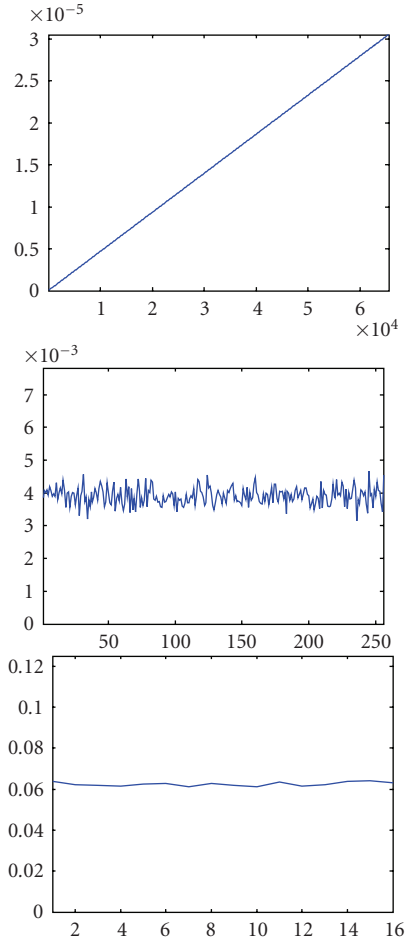

Figure 8: More uniform distribution via hashing of 1, 256 and 4096 samples together, respectively.

Number Generation Using Deterministic Random Bit Generators [32]. SHA1 can be replaced by SHA-256, seeded with more HW data, providing 256 bit physical entropy in each call.

Appendix

More Uniform Distribution, Less Autocorrelation via Hashing

Informally. A hash function f returns the same value h for any input value from a set H(h). These H sets are all of "similar" size and not related by "simple" transformations. We assume that the input is "independent" from the internal structure of the hash function, for example, not only elements of H(h) are fed to the function (which would always return h). If f is a cryptographic hash function, and the input ξ is physical random, this is very likely the case; that is, we may assume that $h = f(\xi)$ is random.

In our application the length of the input values of the hash function is fixed (96 bits), thus the size of the domain of possible inputs is finite (2^{96}). The output is truncated to a given length (e.g., to 24 bits, making the size of the range of the output 2^{24}). In this example $2^{96}/2^{24} = 2^{72}$ is the *reduction factor*, these many possible input values yield the same output, in the average.

Claim 1. The *distribution* of the hashed values $h = f(\xi)$ of a random variable ξ is *closer to uniform* than the distribution of ξ. The expected improvement is proportional to the square root of the reduction factor.

Justification. The probability of the union of k different atomic events is the sum of the individual probabilities. We show smoothness improvement in the distribution of the sum of k copies of a new random variable η, which takes values the probabilities of the individual original samples. (Because the sum of the copies of the same random variable includes the case, when some of them are equal, there is a small error. In a large domain, like 2^{96} values, with significantly smaller k, like 2^{24}, this collision has negligible probability.)

The hash function lumps together certain input values of the possible n and produces a single output. It behaves like adding m copies of a random variable taking these probabilities as values. When the input is unrelated to the structure of the hash, each selection is of equal probability, $p = 1/n$.

The unevenness (deviation from the uniform distribution) is measured by the standard deviation of the individual probability values (relative to the expected value). The expected value of the sum of m samples increases m-fold from the original, the same as the increase of the variance, so the standard deviation increases $\sqrt{m}$ -fold. Therefore, hashing m samples together improve the evenness $\sqrt{m}$-fold.

It is true for large m, in the average. Around a factor of two deviations from this value can be observed in the praxis for a given hash function and sufficiently non-uniform initial distribution of the samples.

The original expected value was $\mu_0 = 1/n$, having n probability values summing up to 1. The estimated standard deviation is $\sigma_0 = \sqrt{(1/(n-1))\sum (p_i - 1/n)^2}$. When blocks of samples are hashed together to obtain $k = n/m$ new sample values, the expected value is $\mu = 1/k$, having k probability values summing up to 1. The standard deviation is $\sigma = \sqrt{(1/(k-1))\sum (\sum_k p_{k_i} - 1/k)^2}$, where $p'_i = \sum_k p_{k_i}$ is the probability of the occurrence of ith hashed event.

As an *example* (Figure 8), take a very non-uniform distribution: the probability of a sample in $[0,2^{16})$ is proportional to its value (the distribution is represented by a slanted line instead of a horizontal one of the uniform distribution). If we hash 256 samples together (reducing 16-bit samples to 8 bits), the relative variation is decreased by a factor around 16, making the resulting distribution quite close to uniform. If we hash 4 K samples together (mix and drop 12 bits), the resulting 16 different sample values occur practically with the same probability (around a 64-fold improvement). Here DES encryption was used for hashing, on 0-padded input data, and truncated result.

Hashing similarly improves the short-term *autocorrelation*, even between groups of entries close by. If the input blocks of the hash get filled up with correlated samples,

certain blocks occur more, others occur less frequently, so a correlation between close-by bits leads to non-uniform distribution of the blocks, that is, there are differences in the frequencies of occurrences of certain block contents. As discussed above, the hashing process smoothes out the distribution of blocks of bits, thereby removing any kind of autocorrelation among (groups of) samples, in the input blocks of the hash.

References

[1] Hard Disk Drives, http://www.storagereview.com/guide2000/ref/hdd/index.html.

[2] D. Davis, R. Ihaka, and P. R. Fernstermacher, "Cryptographic randomness from air turbulence in disk drives," in *Proceedings of the 14th Annual International Crytology Conference (Crypto '94)*, 1994.

[3] L. Hars, "Randomness of timing variations in disk drives," Manuscript, 2007.

[4] E. Schreck and W. Ertel, "Disk drive generates high speed real random numbers," *Microsystem Technologies*, vol. 11, no. 8–10, pp. 616–622, 2005.

[5] R. D. Cideciyan, F. Dolivo, R. Hermann, W. Hirt, and W. Schott, "A PRML system for digital magnetic recording," *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 1, pp. 38–56, 1992.

[6] B. Vasic and E. M. Kurtas, Eds., *Coding and Signal Processing for Magnetic Recording Systems*, CRC Press, Boca Raton, Fla, USA, 2005.

[7] C.-H. Wei and A. Chung, "Adaptive Signal Processing," http://cwww.ee.nctu.edu.tw/course/asp.

[8] Y.-S. Tang, "Noise autocorrelation in magnetic recording systems," *IEEE Transactions on Magnetics*, vol. 21, no. 5, pp. 1389–1391, 1985.

[9] J. R. Hoinville, R. S. Indeck, and M. W. Muller, "Spatial noise phenomena of longitudinal magnetic recording media," *IEEE Transactions on Magnetics*, vol. 28, no. 6, pp. 3398–3406, 1992.

[10] R. S. Indeck, M. N. Johnson, G. Mian, J. R. Hoinville, and M. W. Muller, "Noise characterization of perpendicular media," *Journal of the Magnetics Society of Japan*, 1991.

[11] R. S. Indeck, P. Dhagat, A. Jander, and M. W. Muller, "Effect of trackwidth and linear spacing on stability and noise in longitudinal and perpendicular recording," *Journal of the Magnetics Society of Japan*, 1997.

[12] R. Behrens and A. Armstrong, "An advanced read/write channel for magnetic disk storage," in *Proceedings of the 26th IEEE Asilomar Conference on Signals, Systems & Computers*, vol. 2, pp. 956–960, October 1992.

[13] H. K. Thapar and A. M. Patel, "A class of partial response systems for increasing storage density in magnetic recording," *IEEE Transactions on Magnetics*, vol. 23, no. 5, pp. 3666–3668, 1987.

[14] W. L. Abbott, J. M. Cioffi, and H. K. Thapar, "Channel equalization methods for magnetic storage," in *Proceedings of the IEEE International Conference on Communications*, vol. 3, pp. 1618–1622, 1989.

[15] W. L. Abbott, J. M. Cioffi, and H. K. Thapar, "Performance of digital magnetic recording with equalization and offtrack interference," *IEEE Transactions on Magnetics*, vol. 27, no. 1, pp. 705–716, 1991.

[16] W. W. L. Ng, E. H. Lim, and W. Xie, "Method and apparatus for generating random numbers based on filter coefficients of an adaptive filter," US patent no. 6931425.

[17] J. von Neumann, "Various techniques used in connection with random digits," in *von Neumann's Collected Works*, vol. 5, Pergamon Press, Oxford, UK, 1963.

[18] M. Blum, "Independent unbiased coin flips from a correlated biased source: a finite state Markov chain," in *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pp. 425–433, 1984.

[19] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984.

[20] B. Chor and O. Goldreich, "Unbiased bits from source of weak randomness and probabilistic communication complexity," in *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pp. 429–442, 1985.

[21] L. Hars and G. Petruska, "Pseudorandom recursions: small and fast pseudorandom number generators for embedded applications," *EURASIP Journal of Embedded Systems*, vol. 2007, Article ID 98417, 13 pages, 2007.

[22] D. E. Knuth, *Seminumerical Algorithms*, vol. 1 of *The Art of Computer Programming*, Addison-Wessley, Reading, Mass, USA, 1997.

[23] G. Marsaglia, "A current view of random number generators," in *Computer Science and Statistics: The Interface*, pp. 3–10, Elsevier Science, Amsterdam, The Netherlands, 1985.

[24] G. Marsaglia and A. Zaman, "Monkey tests for random number generators," *Computers and Mathematics with Applications*, vol. 26, no. 9, pp. 1–10, 1993.

[25] U. M. Maurer, "A universal statistical test for random bit generators," *Journal of Cryptology*, vol. 5, no. 2, pp. 89–105, 1992.

[26] NIST Special Publication 800-22, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," August 2008, http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf, Source code, http://csrc.nist.gov/publications/nistpubs/800-22-rev1/sp800-22rev1.zip.

[27] T. Ritter, "Randomness Tests: A Literature Survey," 1996, http://www.ciphersbyritter.com/RES/RANDTEST.HTM.

[28] Intel Platform Security Division, The Intel random number generator, 1999.

[29] B. Jun and P. Kocher, The Intel random number generator (white paper), 1999, http://www.securitytechnet.com/resource/crypto/algorithm/random/criwp.pdf.

[30] J. S. Coron and D. Naccache, "An accurate evaluation of Maurer's universal test," in *Proceedings of the ACM Symposium on Applied Computing (SAC '98)*, Lecture Notes in Computer Science, Springer, 1998.

[31] Digital Signature Standard (DSS), FIPS PUB 186-2, Federal Information Processing Standards Publication, U.S. Department of Commerce/National Institute of Standards and Technology, January 2000.

[32] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," NIST Special Publication 800-90, June 2006.