Journal of Signal and Information Processing, 2011, 2, 1-10
doi:10.4236/jsip.2011.21001 Published Online February 2011 (http://www.SciRP.org/journal/jsip)

An Efficient Noise Generator for Validation of Channels Equalizers

Nihar Panda[1], Siba P. Panigrahi[1], Sasmita Kumari Padhy[2]

[1]Electrical Engineering, Konark Institute of Science & Technology, Bhubaneswar, Orissa, India; [2]ITER, SOA University, Bhubaneswar, India
Email: siba_panigrahy15@rediffmail.com

Received December 10th, 2010; revised January 11th, 2011; accepted February 18th, 2011

ABSTRACT

This paper develops an efficient pseudo-random number generator for validation of digital communication channels and secure disc. Drives. Simulation results validates the effectiveness of the random number generator.

Keywords*: *Digital Communication*, *Channel Equalization

1. Introduction

Digital data while transmission over communication channels becomes corrupted because of Inter Symbol Interference (ISI), Co-Channel Interference (CCI), multi-path-fading etc. All the parameters that make a data corrupted are known as noise. Extraction of transmitted data mitigating this noise is known as equalization. Details about the algorithms used in the channel equalization filters can be found in [1,2]. However in order to validate an equalizer, the same need to be tested in presence of a noise or random number generator.

The noise sources and levels have been extensively studied in [3,4], their effects on the signal in the read channel have also been investigated in [5-8]. The resulting inherent randomness in the channel filter coefficients has been proposed for use for random number generators in [9], but the included randomness extraction algorithm is very inefficient.

Generators for Cryptographic random number are employed in many systems, like in self-encrypting disk drives, such as the Seagate Momentus Full Disk Encryption (FDE) drives. The random numbers so generated can be used for encryption keys, facilitating secure communication (via nonces), performing self-tests, and so forth. Previous data of the random number generator are difficult to store securely, 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 may cause repeated nonces, of which recurring use of the same encryption keys, and so forth, that is, loss of security.

Physical entropy sources are used to initialize generators for cryptographic random number at every power up, and at special requests, like at reinitializing the firmware, or before generating long used cryptographic keys. Seeding with physical values that can not be predicted makes a cryptographic random number generator to supply pseudorandom sequences, with negligible probability of repetition. Correspondingly generated 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.

In next sections, we discuss 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 available physical entropy estimation is discussed, resulting in an efficient seeding process. These will 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. System Overview

2.1. The Architecture

The read and write transducers are embeded 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 [10]. The head is embedded on an *arm*, which is connected to an *actuator*. In a usual 3.5" disk drives this arm of 5 cm long and prone to mechanical vibrations, affected by air damping while

the drive is in operation. The vibration in vertical direction affects the amplitude of the read signal, while the radial vibration affects the noise pattern from the granular structure of the magnetic particles and cross talk from neighbor tracks, because of the small spacing between tracks (in the range of 10-100 nm).

To have the head to be on track, *servo pattern*s are written on the disk. These servo patterns are arranged in radial spokes, which are traversed by the head about 200 times per revolution (at a rotational speed of 5400 rpm, 18000 times per second). After the head covers these servo patterns, a controller used to evaluate the read signal and corrects the radial position accordingly. It also makes the channel *equalizer filter* for optimum signal shaping. The tracking correction is based on the present radial position, velocity, and acceleration of the head. These values are random, strongly affected by turbulent air damping and mechanical vibrations. This is still to be explored with a useful model of the disk drive physics. In [3] some mathematical formulations are presented, but still lacks a reasonably accurate picture of disk drive internals.

2.2. Entropy Requirements

In this paper, we show that disk drives can provide physical randomness for seeding generators for cryptographic random number, but they are targets to specific attacks, exploiting their use and special characteristics, leading to specific entropy requirements of the disk. The generalized "birthday bound" speaks 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 virus 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 a network shares these results, 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 rise to a requirement of at least 64-bit entropy of the seed. Of course, a 50% chance of a successful attack is 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), 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 serve 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 random 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 generator for cryptographic random number).

3. Entropy Sources in Rotating Disc Drives

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

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

3.1. Timing Variations

In the disk drive literature there are internal high-speed *counters* available. Least significant bits of these disk drives 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, and consumed on requirement. The entropy of the timing data can be found in [11].

Such random number generators have been presented, the slow [3], implemented externally in the host computer, where synchronous communication masks off most of the original timing variations.

3.2. Tracking Error

Another type of randomness source was investigated in [12], 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 entropy that can be used. Results in this paper in the newest generation of disk drives showed much less achievable speed or entropy/s than claimed in [12], but the position error of the read/write head certainly represents another alternative source of randomness

3.3. Channel Filter-Coefficients

The drive firmware can access the *coefficients of an adaptive channel-equalizer*, via a diagnostic interface between the main control ASIC and the channel signal processor, which also does the coding/decoding of the head signal [13]. Resistor values of an analog filter represented by the coefficients, continuously tuned by the control mechanism of the read/write channel chip to make the

peaks of the output signal close to equally high. The filter coefficients affected by 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.

FDE drives there are 12 such coefficients accessible, each 8 bit long in the Momentus. Asymmetry compensation tap is fixed coefficient of 11, set for each head and zone in the manufacturing process. The other coefficients are constantly adapted to the distorted noisy signal of the servo patterns.

When the generator for random number 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 with involvement of significant mechanical arm movements. These translate to hundreds of changes in the adaptive channel filter, strongly influenced by affecting noise; therefore, there will be very *little correlation* between consecutive acquired values of the same coefficients. A very high noise is experienced in the read-back signal in modern disk drives. In a disk drive under investigation the read-back signal was captured with a digital storage oscilloscope and shown in **Figure 1**.

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

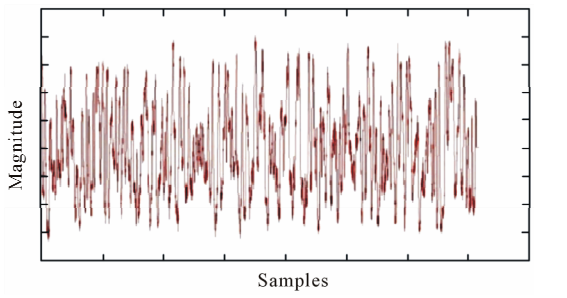

Figure 1. Noisy read-back signal.

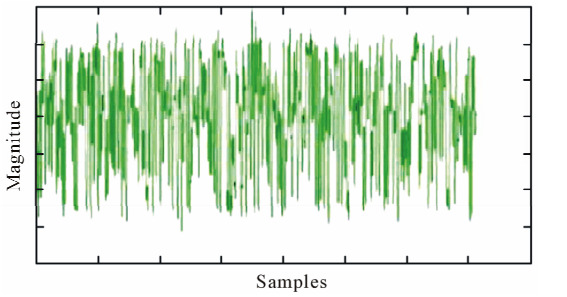

Figure 2. Signal after equalization.

4. Entropy Esimation

We experimented on 22 data sets, 100 M coefficient bytes in each. Those data 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 sets were captured over a diag nostic port and recorded in another PC, not to influence the data collection.

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

We could not found any 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 affecting its working environment.

4.1. Data Dependencies

The plot of the 12 filter coefficients is of relatively stable shape in time. **Figure 3** gives the curves of 10 consecutive captured sets of filter coefficients from the same drive, plotted on top of each other. The abscissa is the index of the filter coefficients (1-12), the ordinate is the value of the corresponding coefficient byte (P1-P12). A curve plotted in one color shows the 12 filter coefficient values of one example set, connected by straight lines.

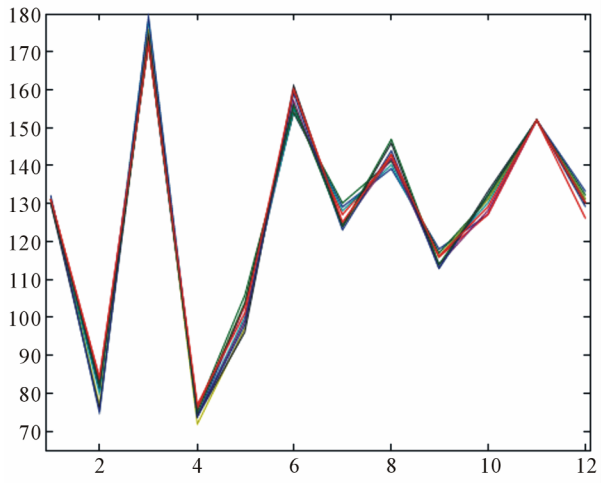

Figure 3. Coefficient changes in time.

One can observe that at some points (*i.e*., between x = 4 and 5) these segments are almost parallel. This means if P4 increases, P5 does, too; therefore, they are positively correlated. At other segments, like the ones between x = 7 and x = 8, cross each other at roughly the same point half way in between. This means that if P7 decreases, P8 increases by roughly the same amount. This 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. This indicates to a *potential* issue: the available entropy could be less than the estimates the coefficient samples provide in isolation.

4.2. Coefficient Distribution

By plotting the histograms of each filter coefficient from contiguous measurement sequences of a disk drive (**Figure 4**) we will observe 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 [1-4]).

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

4.2.1. Autocorrelation of Sequences of Individual Coefficients

This paper uses the discrete Fourier transform of the same individual coefficient sequences described above to compute many autocorrelation values at once: $F^{-1}\left(F(x)\cdot F^{T}(x)\right)$, where $F(x)$ denotes the discrete Fourier transform of the sequence x. $F^{T}(x)$, its transposed complex conjugate and $F^{-1}(X)$ is its inverse. The autocorrelation values are plotted for each of the 12 coefficient sequences in **Figure 5**, lags = 1-50. No value of the autocorrelation exceeds 21%, with an average around 13%. The small residual (large lag) autocorrelation values are the artifacts of the very non-uniform distributions. The uneven distributions and short-term autocorrelation makes overall entropy loss (which only causes a loss of a handful bits entropy). The process described in next subsection "hashing" will eliminate both problems.

4.3. Entropy of Coefficients

Usually, the filter coefficients carry about 3 bits of Shannon entropy:

$$H = -\sum p_i \log(p_i)$$

There are some exceptions as: coefficient 1 carries 1.5 bit, coefficient 2 does 3.5 bits, and coefficient 4 does 2.4 bits. If all of these coefficients were independent, the overall entropy of the 12 channel filter coefficient bytes could be 32 bits. Statistical tests next subsection shows less actual randomness (16-24 bit), because of the correlation among them, and because of their internal autocorrelation.

4.3.1. Min Entropy

The min-entropy is better for estimating the security: $M = -\log_2\left(\max(p_i)\right)$.

A distribution of a min-entropy of at least b bits if no state has a probability greater than 2^{-b}. It determines the complexity of such attack strategies, when the attacker seeds his generator for cryptographic random number (identical to the one in the disk drive) with the most likely coefficient values. If it finds a match, he guessed the seed right. If it 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. The 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 a virus feeds various possible values of the filter coefficients to a copy of the generator for cryptographic random number, he can try millions of seed values in the time of one reboot. This means, 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: for $p_i < 10^{9}$).

4.3.2. Mix-Truncate (Hash) Entropy Estimation

The entropy estimation process is as follows: hash the bits of each channel filter coefficient dataset (12 × 8 = 96 bits) to k bit output. k Decreases from 32 (the upper bound of the entropy from **Figure 6**) until the concatenated output blocks pass all commonly used randomness tests. Perfect hashing results in a more uniform distribution and reduces the autocorrelations in the costs of decreasing the number of random bits. (We make use of 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 virus with reasonable resources cannot distinguish it from a perfect hash.)

There are some other methods in use to shrink data to improve randomness. The first of such method was the Neumann corrector to remove bias [14], but more recent entropy amplification techniques are all related to hashing [15-17]. (A hash function maps arbitrary data to a fix

range of integers, not preserving those simple structures of the input sequences.) The used randomness tests are very sensitive to non-uniform distribution of k-bit blocks, but many other nonrandom properties are checked, too. If 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 virus has a search space of at least 2^{kn} elements when trying different

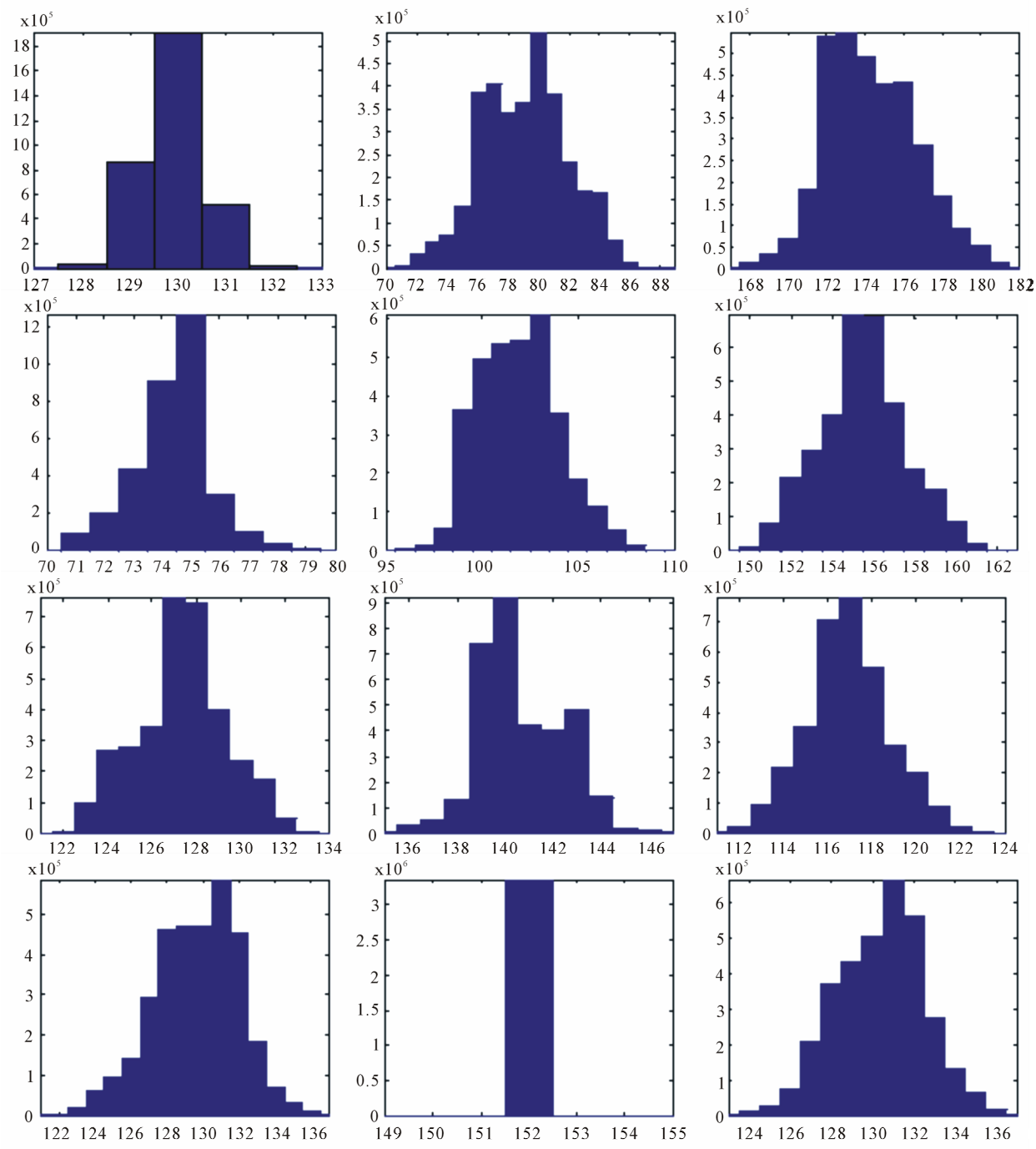

Figure 4. Histograms of the filter coefficients.

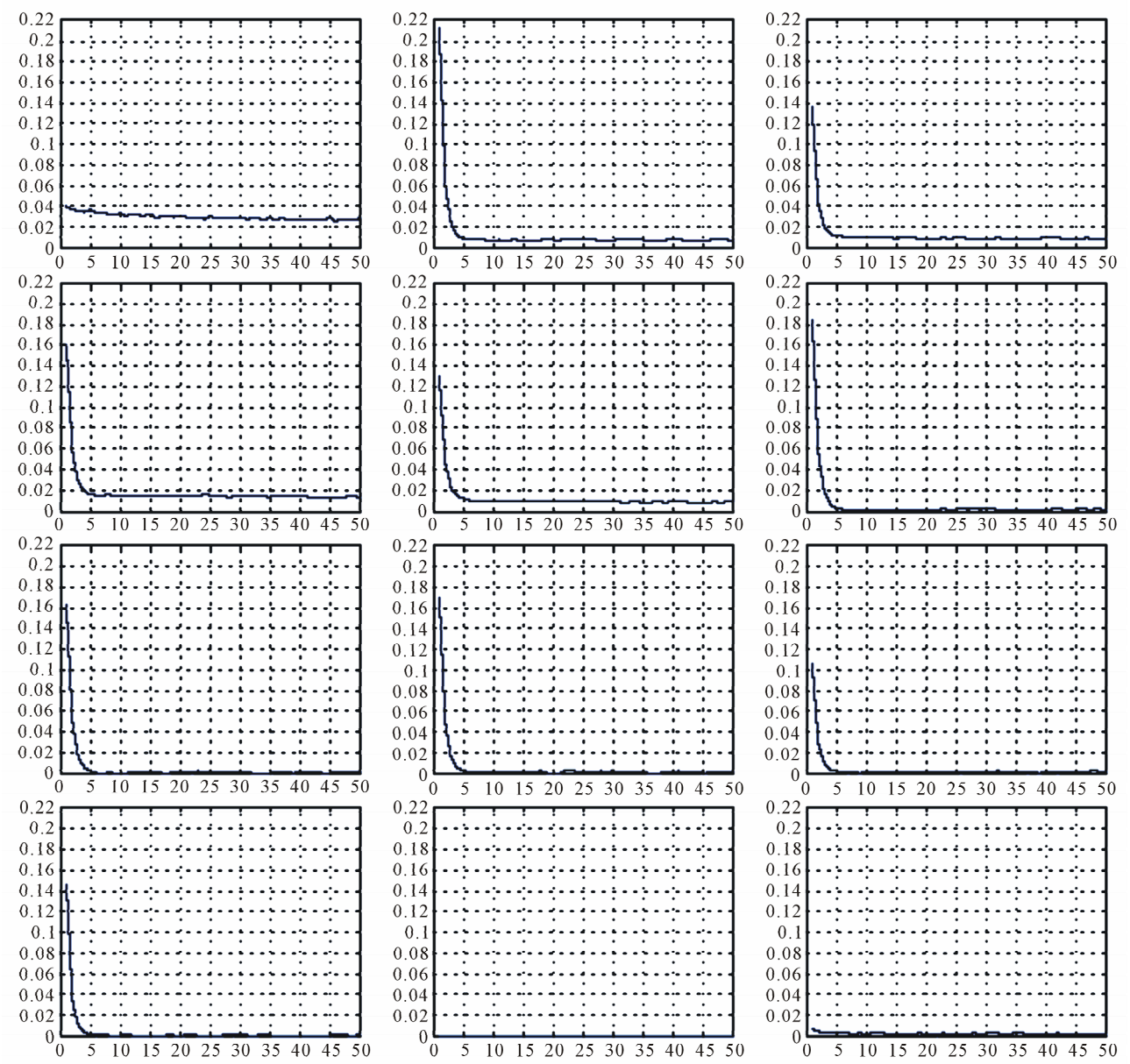

Figure 5. Autocorrelation of the filter coefficients.

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).

4.3.3. Justification of the Mix-Truncate Entropy Estimation

The use of physical randomness in this paper 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 are not be ruled out by the observed data. One can only state that no evidence for non-randomness was found.

Common statistical tests accept many cryptographically hashed non-random sequences as perfectly random, termed pseudo-random, if the size of the hash output is large enough (greater than the binary logarithm of the length of the sequence). E.g., if we hash the members of the sequence $0,1,2,\cdots,10^9$ 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. Arbitrarilymany similar pseudorandom sequences can easily be constructed, which fool the statistical randomness tests, even if we make certain assumptions

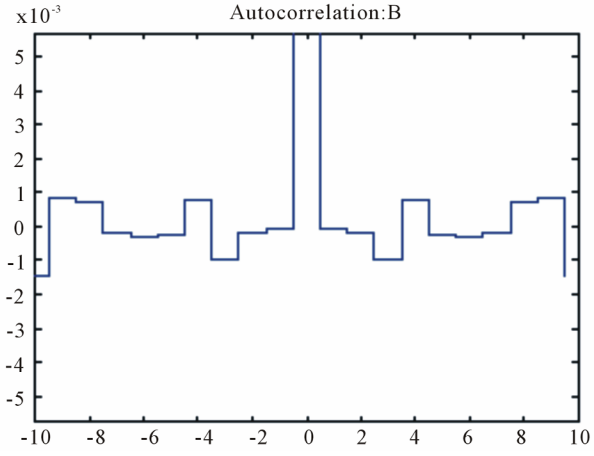

Figure 6. Autocorrelation of a 96-16 bit hashed coefficient sequence.

about the data, like lack of autocorrelation. But, *physical* considerations established that our sample blocks are *independent* to a great degree (which invalidates the pseudorandom counter examples above). Autocorrelation tests did not argue this claim. To be remembered that the independence has physical reasons, not mathematically proven.

The proposed hashing process of this paper 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 changes individual distributions and dependencies within data blocks. Even correlations between groups of coefficients are avoided.

Statistical randomness tests verifies long-term non-randomness, 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 enough for the security of seeding generators for cryptographic pseudorandom number with the hashed data blocks, originated from sets of channel filter coefficients, separated by largely unpredictable mechanical events.

4.3.4. Security of Hashed Seeding of Pseudorandom Number Generator

When the analyzed sequence is used for seeding generators for (cryptographic) pseudorandom number, one 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 second implies the former: if a block repeated often, autocorrelation would be large. This 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. Here, in this paper, 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.

Lower value of autocorrelation assures that no seed blocks occur frequently nor are some blocks correlated. Else, 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 will increase the chance of a successful guess of a seed, revealing all newly generated random numbers until a fresh seed is applied.

4.3.5. 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 affected by environmental conditions. Hence, even for non-cryptographic applications the physical randomness source is usually hashed (corresponding to seeding generators for pseudorandom number), although for lower security requirements there are much faster hash algorithms (e.g., the ones in [18]) than the secure hash functions used in cryptography (e.g., SHA1/2).

5. Randomness Tests

There are many randomness tests presented in [19-23]. A survey can be found in [24].

Diehard Test Suite. 15 different groups of statistical randomness tests can be found in [20,21]. This set of tests is probably the most widely used in literature. Many different properties are tested and the list of the results is 17 pages long. The randomness measures are of 250 *P*-values. The usual 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 convention 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 common procedure used in [25,26] for testing the random number generator implemented in the Intel Pentium III chip works as follows. To come down 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 was considered to pass if all 250 *P*-values are in the corresponding interval [0.0001, 0.9999]. This paper adopted this acceptance criterion, with an additional check described in [12]: count the number of near-fails among the 250 *P*-values returned by the Diehard tests (those *P*- values which are not in [0.025, 0.975]). Because asymptotically the relative number of fails for the given interval is 5%, there must be about 12 near-fails among the 250

values. This 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 accurate enough to autocorrelations, which occur at other than integer multiples of 8 bit offsets. (Some of data sets, which pass the Diehard tests at k = 28, but failed with k = 24 reduction.) Hence, only the tests of hashed filter coefficient set to k = 24, 16, and 8 bits can be fully trusted. Some of 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*: While the Diehard tests and Maurer's test passed on our hashed data, the NIST tests also accepted the input as random [23]. Advantages 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 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 certain given level. For the tests to clear 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. Presented in [22], and further investigated in [27], analyzes the statistics of gaps between the closest occurrences of the same bit blocks. A test for each block size 1-16 is to be performed. Larger test blocks require huge datasets for high confidence in the test results. e.g., the necessary size of the data sets for 16-bit test blocks is $1000 \cdot 2^{16} \cdot 12 \approx 800$ MB. All of the Maurer tests with block sizes b = 1-16 passed, when the data was hashed to k = 16. In this case virtually no memory is present, because of the many seek-induced filter coefficient updates between data acquisitions.)

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

Transform-Tests. An *FFT*-test is may be included among the NIST tests. After computation 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

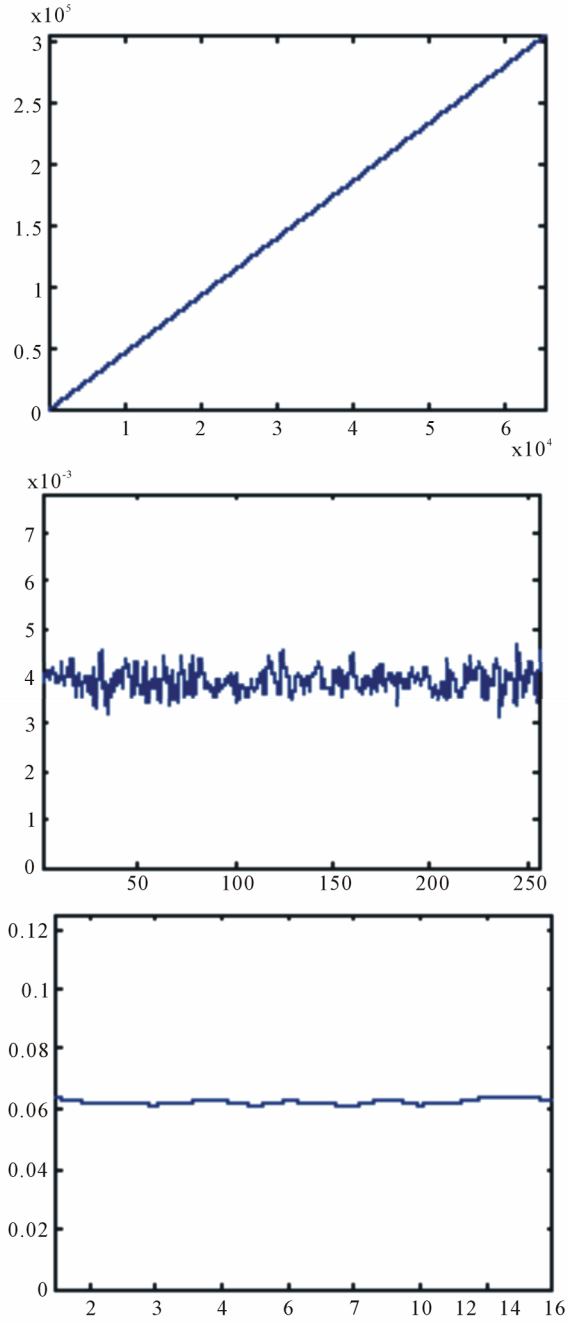

Figure 7. More uniformdistribution via hashing of 1, 256 and 4096 samples together, respectively.

expect 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. Here, we find other type of structured (pseudo-periodic) components in the data. The actual physical model and the observed level of autocorrelation in the individual coefficient sequences predict no significant signal 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).

The Cryptographic Pseudorandom Number Generator. With the techniques described above we found that one channel filter coefficient datasets provides at least 16bit entropy, therefore eight datasets are enough for our desired 128-bit entropy. Here, in this section the algorithm is described, how the available physical randomness is converted to cryptographically secure random numbers.

Channel filter coefficients are collected to start with. 8 datasets need all together about 80 ms (1.6 Kb/s), allowing 12 reseedings a second, which would only rarely be needed. By hybridizing 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. 4 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 generator for cryptographic random number specified in the FIPS-186-2 document [28] was used with SHA1 as hash function and 24-byte (192 bit) internal state. While 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 key value for the seed key, $0 < XKEY < 2^{192}$.

Step 2. In hexadecimal notation let

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

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

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

$$XSEED_j = \text{Optional user input}$$

$$XVAL = \left(XKEY + XSEED_j\right) \bmod 2^{192}$$

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

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

5.1. Accumulated Entropy

The initial entropy of $XKEY$ (the internal state of the cryptographic pseudorandom number generator) is 0 while booting up. After the 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 $XKEY$. During the next iterations only this LS 160 bits are further modified (disregarding a possible carry bit), there fore the accumulated entropy stored in $XKEY$ increases very slowly beyond 160 bits. During initialization (Step 1) one can choose a new secret value for $XKEY$. This can be anything (cannot be 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. Next round calls of the RNG affect the LS bits of $XKEY$, keeping the initial entropy stored in the MS bits intact. Hence, the seeding process can be performed in *two phases*. The 1st phase starts with an all 0 $XKEY$ and uses half of the total number of seeding rounds to mix in the HW entropy. During 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. While in 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) we can execute more phases like the above. SHA1 limits the number of usable bits to 512, but if required, it can be replaced by hash functions operating on larger (or on multiple) blocks.

5.2. 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 followed with, and desirable. Having 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. Then, the LS bits of free running counters are attached. Several of 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.

REFERENCES

[1] B. Vasic, *et al.*, "Coding and Signal Processing for Magnetic Recording Systems," CRC Press, Boca Raton, 2005.

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

[3] D. Davis, R. Ihaka and P. R. Fernstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives," *Proceedings of the* 14*th Annual International Crytology Conference*, Santa Barbara, 21-25 August 1994, pp. 114-120.

[4] R. S. Indeck, *et al*., "Effect of Trackwidth and Linear Spacing on Stability and Noise in Longitudinal and Perpendicular Recording," *Journal of the Magnetics Society of Japan*, Vol. 21, No. 3, 1997, pp. 213-219.

[5] R. Behrens and A. Armstrong, "An Advanced Read/Write Channel for Magnetic Disk Storage," *Proceedings of the* 26*th IEEE Asilomar Conference on Signals, Systems* &

Computers, 26-28 October 1992, pp. 956-960.

[6] 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, 1987, pp. 3666-3668. doi:10.1109/TMAG.1987.1065230

[7] W. L. Abbott, J. M. Cioffi and H. K. Thapar, "Channel Equalization Methods for Magnetic Storage," *Proceedings of the IEEE International Conference on Communications*, Helsinki, 11-14 June 1989, pp. 1618-1622.

[8] 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, 1991, pp. 705-716. doi:10.1109/20.101120

[9] 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.

[10] Hard Disk Drives, 2002, http://www.storagereview.com/guide 2000/ref/hdd/index.html

[11] E. Schreck and W. Ertel, "Disk Drive Generates High Speed Real Random Numbers," *MicrosystemTechnologies*, Vol. 11, No. 8-10, 2005, pp. 616-622. doi:10.1007/s00542-005-0532-6

[12] 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, 1992, pp. 38-56. doi:10.1109/49.124468

[13] J. von Neumann, "Various Techniques Used in Connection with Random Digits," *von Neumann's Collected Works*, Vol. 5, Pergamon Press, Oxford, 1963.

[14] L. Hars, "Randomness of Timing Variations in Disk Drives," Manuscript, 2007.

[15] M. Blum, "Independent Unbiased Coin Flips from a Correlated Biased Source: A Finite StateMarkov Chain," *Proceedings of the* 25*th Annual Symposium on Foundations of Computer Science*, Singer Island, 26 October 1984, pp. 425-433.

[16] M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits," *SIAM Journal on Computing*, Vol. 13, No. 4, 1984, pp. 850-864. doi:10.1137/0213053

[17] B. Chor and O. Goldreich, "Unbiased Bits Fromsource of Weak Randomness and Probabilistic Communication ComPlexity," *Proceedings of the* 26*th Annual Symposium on Foundations of Computer Science*, Washing DC, 21-23 October 1985, pp. 429-442.

[18] L. Hars and G. Petruska, "Pseudorandom Recursions: Small and Fast Pseudorandom Number Generators for Embedded Applications," *EURASIP Journal of Embedded Systems*, No. 1, 2007.

[19] D. E. Knuth, "Seminumerical Algorithms, the Art of Computer Programming," Addison-Wessley, Boston, 1997.

[20] G. Marsaglia, "A Current View of Random Number Generators," *Computer Science and Statistics: The Interface*, Elsevier Science, Amsterdam, 1985, pp. 3-10.

[21] G. Marsaglia and A. Zaman, "Monkey Tests for Random number Generators," *Computers and Mathematics with Applications*, Vol. 26, No. 9, 1993, pp. 1-10. doi:10.1016/0898-1221(93)90001-C

[22] U. M. Maurer, "A Universal Statistical Test for Random Bit Generators," *Journal of Cryptology*, Vol. 5, No. 2, 1992, pp. 89-105. doi:10.1007/BF00193563

[23] 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

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

[25] Intel Platform Security Division, the Intel Random Number Generator, 1999.

[26] B. Jun and P. Kocher, The Intel Random Number Generator (White Paper), 1999. http://www.securitytechnet.com/rsource/crypto/algorithm/random/criwp.pdf

[27] J. S. Coron and D. Naccache, "An Accurate Evaluation of Maurer's Universal Test," *Proceedings of the ACM Symposium on Applied Computing* (*SAC*'98), Atlanta, 27 February-1 March 1998.

[28] 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.

