On Stochastic Security of Pseudorandom Sequences

Yongge Wang

UNC Charlotte

Abstract. Cryptographic primitives such as secure hash functions (e.g., SHA1, SHA2, and SHA3) and symmetric key block ciphers (e.g., AES and TDES) have been commonly used to design pseudorandom generators with counter modes (e.g., in NIST SP800-90A standards). It is assumed that if these primitives are secure then the sequences generated by pseudorandom generators based on these primitives are indistinguishable from sequences from true random sources. However, no systematic research and analysis have been done to support this assumption. Based on complexity theoretic results for pseudorandom sequences, this paper analyzes stochastic properties of long sequences produced by pseudorandom generators DRBG-SHA from NIST SP800-90A. A collection of 6TB random sequences are generated and it is observed that the statistical distance between the collection of generated sequences and uniform distribution is around 0.07 (with 0 for statistically indistinguishable and 1 for completely distinguishable). We then try to see whether other seeding approaches to pseudorandom generators could be used to reduce the statistical distance. For example, we carried out experiment with dynamic seeding approach. The experimental results show that the collection of sequences generated by the revised seeding approach is also at least 0.07 statistically away from the collection of uniform chosen sequences. Though the statistical distance 0.07 is acceptable in practice for most applications, the preferred distance of a cryptographic "random oracle" from a true random source should be smaller than 0.03 with the sample size of 1000 sequences (2TB bits).

1 Introduction

Pseudorandom generators and pseudorandom sequences play important roles in modern cryptography. For example, the weakness in pseudorandom generators was employed to attack the SSL protocol [10] and the recent reports from New York Times [19] and The Guardian [1] show that NSA has put back doors in NIST SP800-90A pseudorandom bit generators (on which our experiments are based on) to get online cryptanalytic capabilities. A string is said to be cryptographically pseudorandom if no efficient observer can distinguish it from a uniformly chosen string of the same length. Secure cryptographic hash functions such as SHA1, SHA2, and SHA3 [20, 5] are often used to generate pseudorandom sequences with fixed length outputs (e.g., 160 bits or 256 bits). In practice it is also important to generate long pseudorandom sequences. For example, in the security proof of cryptographic protocols using the "random oracle model" paradigm (see, e.g., [3]), the participants and the adversaries may make polynomial number of queries to the random oracle and the total output from the random oracle could be several gigabytes (GBs) long. In order for the security proof to work, it is assumed that the random

oracle output is computationally indistinguishable from a string that is chosen with the uniform probability.

Though security of hash functions such as SHA1, SHA2, and SHA3 has been extensively studied from the one-wayness and collision resistant aspects, there has been limited research on the quality of long pseudorandom sequences generated by cryptographic hash functions. Recently, the authors of [4] used the indifferentiability concept to analyze the security of sponge function based pseudorandom generators [4, 2] by assuming that the underlying primitives are random permutations (or random functions). However, there is no existing feasible approach to verify whether a given primitive is a random permutation (or a random function). Even if a hash function (e.g., SHA1) performs like a random function based on existing statistical tests (e.g., NIST SP800-22 Revision 1A [21]), when it is called many times for a long sequence generation, it is not clear whether the resulting long sequence still satisfy the properties of pseudorandomness.

In complexity theoretic research, random sequences are considered as Brownian motions that are described by the Wiener process. One of the important laws for the Wiener process is the law of the iterated logarithm (LIL) which says that, for a pseudorandom sequence ξ, the value $S_{lil}(\xi[0..n-1])$ (this value is defined in Theorem 3) should stay in $[-1, 1]$ and reach both ends infinitely often when n increases. Our experimental results show that, if the DRBG in NIST SP800-90A is used to generate one thousand of 2GB-long sequences (i.e., a total of 2TB random bits), the distribution induced by $S_{lil}(\xi[0..n-1])$ has a statistical distance 0.07 from Brownian motion distributions. Though the statistical distance 0.07 is acceptable for most practical applications, the preferred distance of a cryptographic "random oracle" from a true random source should be smaller than 0.03 for a randomly chosen 1000 of 2GB-long sequences.

For hash function based DRBG in NIST SP800-90A, the seeding information to the pseudorandom generator is converted to a seedlen-bit counter, where seedlen is 440 for SHA1/SHA256 and is 888 for SHA384/SHA512. Thus the input to each hash function call in DRBG contains one message block after the hash function internal padding. We wonder whether an improved seeding approach could be used to reduce the observed statistical distance 0.07 by dynamically changing the hash function internal state for each primitive hash function call of the pseudorandom generators. In order to verify our conjecture, we revised the pseudorandom generators with dynamic seeding approach such that the last message block of the input has significant changes in the 0-1 distribution and the second from the last message block of the input (which contributes to the state of the hash function operation) changes for each hash function call also. In the experiment, the dynamic inputs are generated using linear feedback shift registers (LFSRs). The revised generators are used to generate 1000 sequences and each sequence is 2GB long. The analysis shows that this collection of sequences has a slightly larger statistical distance (but still around the range of 0.07 and 0.08) from the uniform distribution.

The paper is organized as follows. Section 2 gives a brief introduction and introduces notations. Section 3 reformulates pseudorandom generator concepts in terms of martingales. Section 4 compares cryptographic pseudorandom sequences and complexity theoretic pseudorandom sequences. Section 5 discusses the law of iterated loga-

rithms (LIL) and the distribution induced by the LIL formula S_{lil}. Section 6 reports experimental results on NIST DRBGs and Section 7 reports experimental results on revised pseudorandom generators with dynamic seeds.

2 Introduction

Classical random sequences were first introduced as a type of disordered sequences, called "Kollektivs", by von Mises [24] as a foundation for probability theory. The two features characterizing a Kollektiv are: the existence of limiting relative frequencies within the sequence and the invariance of these limits under the operation of an "admissible place selection". Here an admissible place selection is a procedure for selecting a subsequence of a given sequence ξ in such a way that the decision to select a term $\xi[n]$ does not depend on the value of $\xi[n]$. Ville [23] showed that von Mises' approach is not satisfactory by proving that: for each countable set of "admissible place selection" rules, there exists a "Kollektiv" which does not satisfy the law of the iterated logarithm (LIL). Later, Martin-Löf [17] developed the notion of random sequences based on the notion of typicalness. A sequence is typical if it is not in any *constructive* null sets. Schnorr [22] introduced p-randomness concepts by defining the *constructive* null sets as polynomial time computable measure 0 sets. The law of the iterated logarithm (LIL) plays a central role in the study of the Wiener process and Wang [25] showed that LIL holds for p-random sequences.

In this paper, N and R^+ denotes the set of natural numbers (starting from 0) and the set of non-negative real numbers, respectively. $\Sigma = \{0, 1\}$ is the binary alphabet, Σ^* is the set of (finite) binary strings, Σ^n is the set of binary strings of length n, and Σ^∞ is the set of infinite binary sequences. The length of a string x is denoted by $|x|$. λ is the empty string. For strings $x, y \in \Sigma^*$, xy is the concatenation of x and y, $x \sqsubseteq y$ denotes that x is an initial segment of y. For a sequence $x \in \Sigma^* \cup \Sigma^\infty$ and a natural number $n \geq 0$, $x[0..n]$ denotes the initial segment of length $n+1$ of x ($x[0..n] = x$ if $|x| \leq n+1$) while $x[n]$ denotes the nth bit of x, i.e., $x[0..n] = x[0] \dots x[n]$. For each string w, $\mathbf{C}_w = \{w\xi \ : \xi \in \Sigma^\infty\}$ is called the basic open set defined by w. For a set $\mathbf{C}$ of infinite sequences, $Prob[\mathbf{C}]$ denotes the probability that $\xi \in \mathbf{C}$ when ξ is chosen by a uniform random experiment. Martingales are used to describe betting strategies in probability theory.

Definition 1. *(Ville [23]) A* martingale *is a function $F : \Sigma^* \to R^+$ such that, for all $x \in \Sigma^*$,*

$$F(x) = \frac{F(x1) + F(x0)}{2}.$$

We say that a martingale F succeeds *on a sequence $\xi \in \Sigma^\infty$ if $\limsup_n F(\xi[0..n-1]) = \infty$.*

Lemma 1. *(Ville [23]) Let F be a martingale and $F_k = \{x \in \Sigma^* \ : \ F(x) > k\}$. Then $Prob[F_k \cdot \Sigma^\infty] \leq F(\lambda)k^{-1}$.*

Based on Lemma 1, Ville [23] showed that a set of infinite sequences has probability 0 (or Lebesgue measure 0) if and only if there is a martingale which succeeds on all

sequences in the set. For each basic open set $\mathbf{C}_{x_0}$, define a martingale F^{x_0} by

$$F^{x_0}(x) = \begin{cases} 2^{|x|-|x_0|} & x \sqsubseteq x_0 \\ 1 & x_0 \sqsubseteq x \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Then $F^{x_0}(\lambda) = 1/2^{|x_0|} = Prob[\mathbf{C}_{x_0}]$ and, for all $x \in x_0 \cdot \Sigma^*$, $F^{x_0}(x) = 1$. Throughout the paper, we will use the martingale F^U for the uniform distribution by letting $F^U(x) = 1$ for all $x \in \Sigma^*$. That is, $F^U(x) == F^\lambda(x)$ or all $x \in \Sigma^*$.

A martingale ensemble $\{F_n\}_{n\in N}$ is a sequence of martingales with the following properties:

1. For each $n \in N$, F_n is a martingale with $F_n(\lambda) = 1$.
2. For $|x| > n$, $F_n(x) = F_n(x[0..n-1])$.

In other words, for a martingale ensemble $\{F_n\}_{n\in N}$, each F_n defines a probability distribution over Σ^n.

3 Pseudorandom generators

The concept of "effective similarity" by Goldwasser and Micali [11] and Yao [26] is defined as follows: Let $X = \{X_n\}_{n\in N}$ and $Y = \{Y_n\}_{n\in N}$ be two probability ensembles such that each of X_n and Y_n is a distribution over Σ^n. We say that X and Y are computationally (or statistically) indistinguishable if for every feasible algorithm A (or every algorithm A), the difference $d_A(n) = |Prob[A(X_n) = 1] - Prob[A(Y_n) = 1]|$ is a negligible function in n. This concept can be rephrased in terms of martingales. First, we note that each probability ensemble $X = \{X_n\}_{n\in N}$ can be represented as a martingale ensemble $\{F_n\}_{n\in N}$ with the following properties:

1. $F_n(\lambda) = 1$ and $F_n(x) = 2^n \cdot Prob[X_n = x]$ for all $x \in \Sigma^n$. In other words, $F_n(x) = Prob[X_n|x]$ for all $x \in \Sigma^n$.
2. For $|x| > n$, $F_n(x) = F_n(x[0..n-1])$.

Definition 2. *Let $\{F_n\}_{n\in N}$ and $\{\bar{F}_n\}_{n\in N}$ be two martingale ensembles. $\{F_n\}_{n\in N}$ and $\{\bar{F}_n\}_{n\in N}$ are computationally (respectively, statistically) indistinguishable if for any polynomial time computable set $D \in \Sigma^*$ (respectively, any set $D \in \Sigma^*$) and any polynomial p, the inequality (2) holds for almost all n.*

$$\frac{1}{2^n} \cdot \left| \sum_{x\in D\cap\Sigma^n} F_n(x) - \sum_{x\in D\cap\Sigma^n} \bar{F}_n(x) \right| \leq \frac{1}{p(n)} \tag{2}$$

Let $l : N \to N$ with $l(n) \geq n$ for all $n \in N$ and G be a polynomial-time computable algorithm such that $|G(x)| = l(|x|)$ for all $x \in \Sigma^*$. The martingale ensemble $\left\{F^G_{l(n)}\right\}_{n\in N}$ is defined by letting $F^G_{l(n)}(x) = \sum_{x_0\in\Sigma^n} F^{n,G(x_0)}(x)$ where

$$F^{n,G(x_0)}(x) = \begin{cases} 2^{|x|-n} & x \sqsubseteq G(x_0) \\ 0 & \text{otherwise} \end{cases}$$

for $|x| \leq l(n)$, and $F^{n,G(x_0)}(x) = F^{n,G(x_0)}(x[0..l(n)-1])$ for $|x| > l(n)$. It should be noted that $F^{n,G(x_0)}$ is different from the martingale $F^{G(x_0)}$ for the basic open set $\mathbf{C}_{G(x_0)}$.

Let $\{F_n^U\}_{n\in N}$ be the martingale ensemble for the uniform distribution with $F_n^U = F^U$ for all n. Then the pseudorandom generator concept [6, 26] could be rephrased in terms of martingales as follows.

Definition 3. *Let $l : N \to N$ with $l(n) > n$ for all $n \in N$. A pseudorandom generator is a polynomial-time algorithm G with the following properties:*

1. *$|G(x)| = l(|x|)$ for all $x \in \Sigma^*$.*
2. *The martingale ensembles $\left\{F_{l(n)}^G\right\}_{n\in N}$ and $\left\{F_{l(n)}^U\right\}_{n\in N}$ are computationally indistinguishable.*

4 Long Pseudorandom sequences

In cryptography, long pseudorandom sequences are often generated using a cryptographic hash function or a block cipher. For example, in SecureRandom class of Java Cryptography Architecture [13, 14], the API SHA1PRNG generates a long pseudorandom sequence using the SHA1 hash function. NIST SP 800-90A [2] recommends three categories of random bit generators: hash function based, block cipher based, and elliptic curve based.

To evaluate the quality of pseudorandom sequences, NIST SP800-22 Revision 1A [21] (software package available at [18]) proposed a statistical test suite for pseudorandom number generators that includes 15 tests: frequency (monobit), number of 1-runs and 0-runs, longest-1-runs, binary matrix rank, discrete Fourier transform, template matching, Maurer's "universal statistical" test, linear complexity, serial test, the approximate entropy, the cumulative sums (cusums), the random excursions, and the random excursions variants. In a statistical test of [21], a significance level $\alpha \in [0.001, 0.01]$ is chosen for each test. For each input sequence, a P-value is calculated and the input string is accepted as pseudorandom if P-value $\geq \alpha$. A pseudorandom generator is considered as a good generator if approximately 100α percent sequences produced by the generator will fail the test. In other words, around 100α percent generated sequences has P-value less than α.

It is important to have 100α percent sequences to fail the test in [21]. We illustrate this with the following example. Let $\text{RAND}_{c,n}$ be the sets of Kolmogorov c-random binary strings, where $c \geq 1$. That is, for a universal Turing machine M, let

$$\text{RAND}_{c,n} = \{x : \text{ if } M(y) = x \text{ then } |y| \geq |x| - c\}. \tag{3}$$

Let $l(n) \geq n+3$ and $f_n : \{0,1\}^n \to \text{RAND}_{2,l(n)}$ be a given ensemble of functions (not necessarily computable). Then for each n-bit string x, $f_n(x)$ is a Kolmogorov 2-random string of length $l(n)$. Since $f_n(x)$ is Kolmogorov 2-random, it is guaranteed to pass all statistical tests specified in [21]. However, it is straightforward to show that the output of $\{f_n\}_{n\in N}$ could be distinguished from the uniform distribution with a non-negligible

probability. In other words, $\{f_n\}_{n\in N}$ is not a cryptographically secure pseudorandom generators.

Computational complexity based pseudorandom sequences have been studied extensively in the literature. For example, p-random sequences are defined by taking each polynomial time computable martingale as a statistical test.

Definition 4. *(Schnorr [22]) An infinite sequence $\xi \in \Sigma^\infty$ is p-random (polynomial time random) if for any polynomial time computable martingale F, F does not succeed on ξ.*

A sequence $\xi \in \Sigma^\infty$ is Turing machine computable if there exists a Turing machine M to calculate the bits $\xi[0]$, $\xi[1]$, $\cdots$. In the following, we prove a theorem which says that, for each Turing machine computable non p-random sequence ξ, there exists a martingale F such that the process of F succeeding on ξ can be efficiently observed in time $O(n^2)$. The theorem is useful in the characterizations of p-random sequences and in the characterization of LIL-test waiting period.

Theorem 1. *For a sequence $\xi \in \Sigma^\infty$ and a polynomial time computable martingale F, F succeeds on ξ if and only if there exists a martingale F' and a non-decreasing $O(n^2)$-time computable (with respect to the unary representation of numbers) function from $h : N \to N$ such that $F'(\xi[0..n-1]) \geq h(n)$ for all n.*

Proof. See Appendix. □

In the following, we establish the relationship between computational indistinguishability and p-randomness. Fix a standard polynomial time computable and invertible pairing function $\langle\cdot,\cdot\rangle : N \times N \to N$ such that, for each $i \in N$, there is a real $\alpha(i) > 0$ satisfying

$$|\{m : \langle i, m\rangle \leq 2^n\}| \geq \alpha(i) \cdot 2^n \text{ for almost all } n.$$

For each $i \in N$, use $\langle i, \cdot\rangle$ as a selection function to obtain a subsequence from ξ:

$$\xi_i = \xi[\langle i, 0\rangle]\xi[\langle i, 1\rangle]\xi[\langle i, 2\rangle]\xi[\langle i, 3\rangle]\cdots$$

In order to establish the relationship between complexity theoretic pseudorandom concepts and cryptographic indistinguishability concepts, we first define a martingale ensemble based on the series of sequences $\xi_0, \xi_1, \xi_2, \cdots$ derived from ξ. Let $r : N \to N^+$ be a non-decreasing function and $\{F^r_{n,\xi}\}_{n\in N}$ be a martingale ensemble defined by

$$F^r_{n,\xi}(x) = 2^n \cdot \frac{|\{\xi_i : x \sqsubseteq \xi_i, i < r(n)\}|}{r(n)}$$

for $x \in \Sigma^n$ and $F^r_{n,\xi}(x)$ for other $x \in \Sigma^*$ is defined correspondingly according to martingale ensemble requirements.

For the sake of convenience and completeness of the description, we present the following theorem in two parts: Martin-Löf randomness based result and p-randomness based result. For those who are not familiar with Martin-Löf randomness concepts, they may skip part one of the theorem or check [16, 22, 25] for details.

Theorem 2. *1. For each Martin-Löf random sequence $\xi \in \Sigma^\infty$, there exists a non-decreasing function $r(n)$ such that martingale ensembles $\{F^r_{n,\xi}\}_{n\in N}$ and $\{F^U_n\}_{n\in N}$ are computationally indistinguishable.*

2. For each p-random sequence $\xi \in \Sigma^\infty$, there exist a real number $\varepsilon > 0$, a polynomial $p(n)$, and a non-decreasing function $r(n)$ with $2^{\varepsilon n} \leq r(n) \leq 2^{p(n)}$ such that martingale ensembles $\{F^r_{n,\xi}\}_{n\in N}$ and $\{F^U_n\}_{n\in N}$ are computationally indistinguishable.

Sketch of Proof. We describe the proof for part one of the theorem. The proof arguments for part two are similar to that of part one with resource constraints and the details could be found in the full version of this paper. We first note that if a sequence $\xi \in \Sigma^\infty$ is Martin-Löf random, then all of the sequences $\xi_0, \xi_1, \xi_2, \cdots$ are Martin-Löf random. For a contradiction, assume that there is a Turing machine approximable martingale F that succeeds on ξ_i for some $i \in N$. Then we can easily convert the martingale F to another Turing machine approximable martingale F' that succeeds on ξ, which is a contradiction. Similarly, we can show that for any $n > 0$, the following sequence β is Martin-Löf random:

$$\beta = \xi_0[0..n-1]\xi_1[0..n-1]\xi_2[0..n-1]\cdots$$

By the fact that Martin-Löf random sequences are normal with respect to any given $n > 0$, we have $\lim_{n\to\infty} F^r_{n,\xi}(x) = 1 = F_U(x)$ for all $x \in \Sigma^*$. In other words, appropriate non-decreasing function r could be chosen such that the two given ensembles are computationally indistinguishable. □

The readers may wonder whether the other directions of Theorem 2 hold also. By Ville's construction of the counter example for a von Mises' "Kollektiv" that does not satisfy the law of the iterated logarithm, the other directions of Theorem 2 do not hold. Indeed, if we choose independent Martin-Löf random sequences $\xi_0, \xi_2, \xi_4, \cdots$ and let $\xi_{2i+1} = \xi_{2i}$ for all $i \in N$, then it can be shown that the martingale ensembles $\{F^r_{n,\xi}\}_{n\in N}$ and $\{F^U_n\}_{n\in N}$ are computationally indistinguishable for an appropriately chosen non-decreasing function r though the resulting sequence ξ is not random in any sense. It is an *open* question whether it is feasible to build some kind of equivalence between the cryptographic indistinguishability concepts and the complexity theoretic randomness concepts.

5 Stochastic Properties of Pseudorandom Sequences

It is shown in [25] that p-random sequences are stochastic in the sense of von Mises and satisfy common statistical laws such as the law of the iterated logarithm. It is not difficult to show that all p-random sequences pass the NIST SP800-22 [21] tests for $\alpha = 0.01$ since each test in [21] could be converted to a polynomial time computable martingale which succeeds on all sequences that do not pass this test. However, none of the sequences generated by pseudorandom generators are p-random since from the generator algorithm itself, a martingale can be constructed to succeed on sequences that it generates.

Since there is no efficient mechanism to generate p-random sequences, pseudorandom generators are commonly used to produce long sequences for cryptographic applications. While the required uniformity property (see NIST SP800-22 [21]) for pseudorandom sequences is equivalent to the law of large numbers, the scalability property (see [21]) is equivalent to the invariance property under the operation of "admissible place selection" rules. Since p-random sequences satisfy common statistical laws, it is reasonable to expect that pseudorandom sequences produced by pseudorandom generators satisfy these laws also (see, e.g., [21]).

The law of the iterated logarithm (LIL) describes the fluctuation scales of a random walk. For a nonempty string $x \in \Sigma^*$, let

$$S(x) = \sum_{i=0}^{|x|-1} x[i] \quad \text{and} \quad S^*(x) = \frac{2 \cdot S(x) - |x|}{\sqrt{|x|}}$$

where $S(x)$ denotes the *number* of 1s in x and $S^*(x)$ denotes the *reduced number* of 1s in x. $S^*(x)$ amounts to measuring the deviations of $S(x)$ from $\frac{|x|}{2}$ in units of $\frac{1}{2}\sqrt{|x|}$.

The law of large numbers says that, for a pseudo random sequence ξ, the limit of $\frac{S(\xi[0..n-1])}{n}$ is $\frac{1}{2}$, which corresponds to the frequency (Monobit) test in NIST SP800-22 [21]. But it says nothing about the reduced deviation $S^*(\xi[0..n-1])$. It is intuitively clear that, for a pseudorandom sequence ξ, $S^*(\xi[0..n-1])$ will sooner or later take on arbitrary large values (though slowly). The law of the iterated logarithm (LIL), which was first discovered by Khintchine [15], gives an optimal upper bound $\sqrt{2 \ln \ln n}$ for the fluctuations of $S^*(\xi[0..n-1])$. It was showed in Wang [25] that this law holds for p-random sequences also.

Theorem 3. *(LIL for p-random sequences [25]) For a sequence $\xi \in \Sigma^\infty$, let*

$$S_{lil}(\xi[0..n-1]) = \frac{2\sum_{i=0}^{n-1} \xi[i] - n}{\sqrt{2n \ln \ln n}}.$$

Then for each p-random sequence $\xi \in \Sigma^\infty$ we have both

$$\limsup_{n\to\infty} S_{lil}(\xi[0..n-1]) = 1 \textit{ and } \liminf_{n\to\infty} S_{lil}(\xi[0..n-1]) = -1.$$

When ξ is chosen according to a given probability, $S_{lil}(\xi[0..n-1])$ defines a induced probability measure on the real line R. Let $\mathcal{R} \subset \Sigma^n$ be a set of m sequences with a probability definition $Prob[x = x_0] = \frac{1}{m}$ for each $x_0 \in \mathcal{R}$. Then each set $\mathcal{R} \subset \Sigma^n$ induces a probability measure $\mu_n^{\mathcal{R}}$ on R by letting

$$\mu_n^{\mathcal{R}}(I) = Prob\left[S_{lil}(x) \in I, x \in \mathcal{R}\right]$$

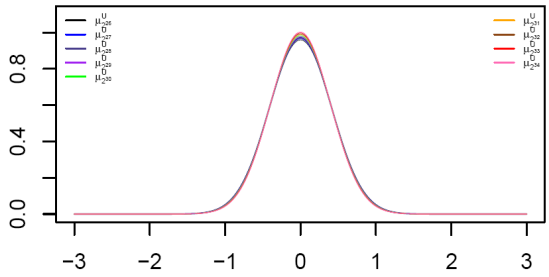
for each Lebesgue measurable set I on R. For $U = \Sigma^n$, we use μ_n^U to denote the corresponding probability measure induced by the uniform distribution. By Definition 2, if $\mathcal{R}_n$ is the collection of all length n sequences generated by a pseudorandom generator, then the difference between μ_n^U and $\mu_n^{\mathcal{R}_n}$ is negligible.

By DeMoivre-Laplace theorem, for a uniformly chosen ξ, the distribution of $S^*(\xi[0,,n-1])$ could be approximated by a normal distribution of mean 0 and variance 1, with error bounded by $\frac{1}{n}$ (see [8]). In other words, the measure μ_n^U can be calculated as

$$\mu_n^U((-\infty,x]) \simeq \sqrt{2\ln\ln n}\int_{-\infty}^{x} e^{-y^2\sqrt{\ln\ln n}}dy. \tag{4}$$

Figure 1 shows the distributions of μ_n^U for $n = 2^{26},\cdots,2^{34}$.

Fig. 1. Density functions for distributions μ_n^U with $n = 2^{26},\cdots,2^{34}$



For each pseudorandom generator G and any given integer n, it is natural to compare the probability induced by $S^*(\xi[0,,n-1])$ for uniformly chosen ξ and for ξ that are generated by G. Specifically, we can use G to generate a set $\mathcal{R} \subseteq \Sigma^n$ of m sequences and compare the distance between probability measures $\mu_n^{\mathcal{R}}$ and μ_n^U.

A generator G is considered "good" if, for large $m \geq 100$ and $n \geq 2^{10}$, the distance between $\mu_n^{\mathcal{R}}$ and μ_n^U are smaller than a given threshold. There are various definitions for statistical distances between probability measures. In our analysis, we use the total variation distance of probability measures [7]

$$d(\mu_n^{\mathcal{R}},\mu_n^U) = \sup_{A\subseteq\mathcal{B}} \left|\mu_n^{\mathcal{R}}(A) - \mu_n^U(A)\right| \tag{5}$$

and Hellinger distance [12]

$$H(\mu_n^{\mathcal{R}}||\mu_n^U) = \frac{1}{\sqrt{2}}\sqrt{\sum_{A\in\mathcal{B}}\left(\sqrt{\mu_n^{\mathcal{R}}(A)} - \sqrt{\mu_n^U(A)}\right)^2} \tag{6}$$

where $\mathcal{B} = \{(\infty,1),[1,\infty)\} \cup \{[0.05x-1, 0.05x-0.95) : 0 \leq x \leq 39\}$.

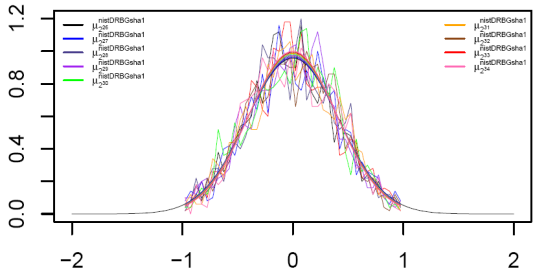
6 Stochastic properties of NIST DRBG pseudorandom generators

NIST SP800.90A [2] specifies three types of DRBG generators: hash function based, block cipher based, and ECC based. In our experiment, we used hash function based

DRBG, where a hash function G is used to generate sequences $G(V)G(V+1)G(V+2)\dots$ with V being seedlen-bit counter that is derived from the secret seeds. The seedlen is 440 for SHA1/SHA-256 and the value of V is revised after at most 2^{19} bits are output. For both SHA1 and SHA256 based generators, we generated 1000 sequences nistSHADRBG0, $\cdots$, nistSHADRBG999, each of which is 2GB long. For each sequence nistSHADRBGi, the seed "ith secret seed for NIST DRBG" is used to derive the initial DRBG state V_0 and C_0. Each sequence is of the format $G(V_0)G(V_0)\cdots G(V_0+2^{12}-1)G(V_1)G(V_1+1)\cdots G(V_{2^{25}}+2^{12}-1)$, where V_{i+1} is derived from the value of V_i and C_i. In other words, each V is used 2^{12} times before it is revised.

By using the partition $\mathcal{B}$ (unit intervals of size $0,05$) defined in Section 5, we get the discrete version of the probability distributions $\mu_n^{nistDRBGsha1}$ with $n=2^{26},\cdots,2^{34}$ in the Appendix Table 4. Figure 2 compares discrete versions of these distributions. Based on the testing results at points $2^{26},\cdots,2^{34}$, we can compute the corresponding

Fig. 2. The distributions μ_n^U and $\mu_n^{nistDRBGsha1}$ with $n=2^{26},\cdots,2^{34}$



total variation distance and Hellinger distance as shown in Table 1. These statistical distances are larger than what one may expect for a true random source of this sample size (1000 sequences).

Table 1. Total variation and Hellinger distances between μ_n^U and $\mu_n^{nistDRBGsha1}$

n	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}	2^{33}	2^{34}
d	.066	.072	.079	.067	.084	.073	.065	.078	.083
H	.060	.070	.073	.062	.077	.066	.067	.070	.087

The testing results for 1000 sequences (of 2GB long each) generated by NIST DRBG-SHA256 are similar and the statistical distance is at least 0.06 also. Since μ_n^U is a standard normal distribution with mean 0, for a sample size of 1000 points, it is expected to have a statistical distance smaller than 0.03. (see, e.g., [9]).

7 Dynamicly seeding a pseudorandom generator

Results in Section 6 show that NIST DRBG-SHA1 generated sequences have a large statistical distance from the uniform distribution. In order to improve the quality of pseudorandom generators, we wonder whether a better seeding approach will help. In existing hash function designs (e.g., SHA1/SHA2/SHA3 [20, 5]), the input to the hash function is padded with a bit 1 followed by 0s (and the length of the message itself in case of SHA1 and SHA2) so that the size of the padded message is a multiple of the hash function message block size. For example, for SHA1 and SHA256, the block size is 512-bit. For an input message M, one first appends a bit "1" followed by at leasst one "0" bit until the last message block has $512 - 64 = 448$ bits. Then append 64 bits indicating the length (in bits) of the original message M. The message blocks are then processed one by one and the hash values are updated correspondingly. If the combined inputs (based on seeds and counters) to the generators are small (e.g., smaller than 440 bits for SHA1 and SHA256) such as in DRBG [21], then each hash function call needs to process only one message block and there is no chance for the initial hash values (or internal state of the sponge function in SHA3) to be dynamically changed. Furthermore, if counter mode is used and consecutive counters are not significantly changed, then inputs to consecutive primitive function calls are almost identical (only a few bits of difference). If the combined inputs (based on seeds and counters) to the generators are larger than 448 bits but smaller than 512 bits for SHA1/SHA256 based generators, then the padded inputs have the form M_1M_2 where M_2 consists mainly of the padded 0-bits. Thus for each hash function call, the hash function processes the same last message block M_2 with different first hash values (or internal hash function state). We wonder whether these "sparse" inputs to the generators reduce the randomness property of the underlying primitives. Thus it is reasonable to design a "better" seeding process for pseudorandom generators. In [21], the seeding information is used to derive a start counter V of seedlen bits which is 440 for SHA1/SHA2. The length of V is chosen in such a way that each hash function call will only have one message block to process. The value of V is revised after at most 2^{19}-bit output using $G(V + G(0\text{x}03||V) + C + reseed\_counter)$ where C contains the entropy of the original seeding. As we have observed in the experiments, the generated sequences show a large probability distance from a true random source from S_{lil}'s viewpoint. One potential improvement is to revise the seeding process in such a way that each hash function call has significantly different last message block and different internal hash algorithm state when the last message block is processed. Specifically, we may use the following seeding approach with two choices for the value of $vLen$ which is defined using $seedlen$ from [21] (note that $seedlen = 440$ for SHA1 and SHA256).

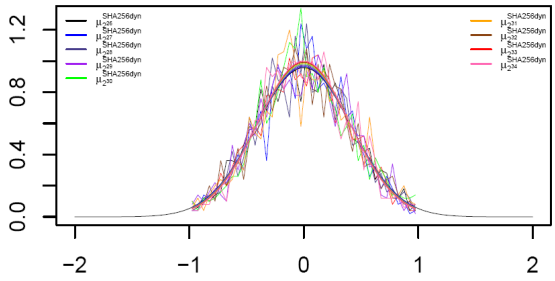
Approach: The seeding information is converted to a series of values $V_0, V_1, \cdots, V_T$ using a second independent pseudorandom generator such that each V_i is of $vLen$ bits and T is the maximal number of requests between re-seeding as defined in [21]. The generated pseudorandom sequence is $G(V_0) \cdots G(V_T)$ where G is a hash function or block cipher primitive. For the first choice of $vLen$, we set $vLen = seedlen$. The second choice is for hash function based generators only and we set $vLen = seedlen + u$ where u is the hash function G's message block size.

The values of $V_0, V_1, \cdots, V_T$ are generated from the seeding information using a second independent pseudorandom generator such as linear feedback shift registers (LFSR). For the first choice of $vLen$, we achieve the same efficiency of [21] by having one message block for each primitive function call. The advantage of the second choice for $vLen$ is that if G is a hash function, then the hash function internal states (or the first hash values) are dynamically changed for each hash function call and we may expect this will produce "better" randomness properties within the generated sequences.

We carried out experiments with the dynamic seeding approach by generating 1000 sequences SHA256dyn0, $\cdots$, SHA256dyn999. For each sequence SHA256dyn$i$, LFSR with the feedback polynomial $x^{32} + x^{22} + x^2 + x + 1$ is used to dynamically generate V_i. Specifically, $V_{0,0} = \text{LFSR}(s_{0,1}) \cdots \text{LFSR}(s_{0,29}) CTR_0$ where $s_{0,1}, \cdots, s_{0,29}$ are 4-bytes integers from the sequence nistSHA1DRBG0 that is generated by NIST DRBG-SHA1 with seed 0 and CTR_0 is a 3-byte integer with value 0. Since each LFSR outputs a 4-byte integer, $V_{0,0}$ contains 119 bytes (that is, 952=512+440 bits). Similarly, the value of $V_{0,1}$ is obtained by obtaining the next 4 bytes from each of the 29 LFSRs and append a 3-byte counter CTR_1 of value 1. In a summary, each sequence SHA256dyn$i$ is in the form of $\text{SHA256}(V_{i,0}) \cdots \text{SHA256}(V_{i,2^{26}})$ and $s_{0,0} \cdots s_{0,29} s_{1,0} \cdots s_{1,29} \cdots$ is a prefix of nistSHA1DRBG0.

The probability distributions $\mu_n^{SHA256dyn}$ with $n = 2^{26}, \cdots, 2^{33}$ is shown in the Appendix Table 5. Figure 3 compares these distributions. Similarly, the corresponding

Fig. 3. The distributions μ_n^U and $\mu_n^{SHA256dyn}$ with $n = 2^{26}, \cdots, 2^{34}$



total variation distance and Hellinger distance are shown in Table 2. The statistical

Table 2. Total variation and Hellinger distances between μ_n^U and $\mu_n^{SHA256dyn}$

n	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}	2^{33}	2^{34}
d	.058	.074	.078	.086	.082	.083	.074	.056	.074
H	.055	.074	.068	.083	.071	.081	.072	.054	.074

distances in Table 2 show no improvement from corresponding values in Table 1.

As a conclusion, we carried out stochastic property analysis of pseudorandom generators. Our analysis shows that distances between the sequences generated by commonly used pseudorandom generators and true random sources are generally larger than expected. It would be interesting to design pseudorandom generators with better (smaller) statistical distances.

References

1. J. Ball, J. Borger, and G. Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security. http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security, Sept. 13, 2013.
2. E. Barker and J. Kelsey. *NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. NIST, 2012.
3. M. Bellare and P. Rogaway. Random oracles are practical: a paradigms for designing efficient protocols. In *Proc. ACM CCS*, pages 62–73, 1993.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge-based pseudo-random number generators. *CHES 2010*, pages 33–47, 2010.
5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak reference. *NIST winning algorithm of SHA3*, 2012.
6. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM J. Comput.*, 13:850–864, 1984.
7. J.A. Clarkson and C.R. Adams. On definitions of bounded variation for functions of two variables. *Tran. AMS*, 35(4):824–854, 1933.
8. W. Feller. *Introduction to probability theory and its applicatons*, volume I. John Wiley & Sons, Inc., New York, 1968.
9. D. Freedman, R. Pisani, and R. Purves. *Statistics*. Norton & Company, 2007.
10. I. Goldberg and D. Wagner. Randomness and the netscape browser. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 21(1):66–71, 1996.
11. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Sys. Sci.*, 28(2):270–299, 1984.
12. E. Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *J. für die reine und angewandte Mathematik*, 136:210–271, 1909.
13. Oracle Inc. Class SecureRandom. http://docs.oracle.com/javase/1.4.2/docs/api/java/security/SecureRandom.html, 2004.
14. Oracle Inc. Java$^{\text{tm}}$ cryptography architecture – API specification & reference. http://docs.oracle.com/javase/1.5.0/docs/guide/security/CryptoSpec.html, 2004.
15. A. Khintchine. Über einen satz der wahrscheinlichkeitsrechnung. *Fund. Math*, 6:9–20, 1924.
16. J. H. Lutz. Almost everywhere high nonuniform complexity. *J. Comput. System Sci.*, 44:220–258, 1992.
17. P. Martin-Löf. The definition of random sequences. *Inform. and Control*, 9:602–619, 1966.
18. NIST. Test suite, http://csrc.nist.gov/groups/ST/toolkit/rng/, 2010.
19. N. Perlroth, J. Larson, and S. Shane. NSA able to foil basic safeguards of privacy on web. http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html, Sep. 5, 2013.
20. Federal Information Processing Standards Publication. Fips pub 180-4, secure hash standard (SHS). *US Department of Commerce*, 2011.
21. A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. NIST SP 800-22, 2010.

22. C. P. Schnorr. *Zufälligkeit und Wahrscheinlichkeit*. Lecture Notes in Math. 218. Springer Verlag, 1971.
23. J. Ville. *Étude Critique de la Notion de Collectif*. Gauthiers-Villars, Paris, 1939.
24. R. von Mises. Grundlagen der wahrscheinlichkeitsrechung. *Math. Z.*, 5:52–89, 1919.
25. Yongge Wang. Resource bounded randomness and computational complexity. *Theoret. Comput. Sci.*, 237:33–55, 2000.
26. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE FOCS*, pages 80–91, 1982.

8 Appendix I: Proof of Theorem 1

Proof. In order to construct the martingale F' and the non-decreasing function $h : N \to N$, we first construct the martingale F' and polynomial time computable function $d : \Sigma^* \to N$ such that,

1. For all $x \sqsubseteq y$, $d(x) \leq d(y)$ and $F'(x) \geq d(x)$.
2. For any sequence $\xi \in \Sigma^\infty$, if F succeeds on ξ then $\lim_n d(\xi[0..n-1]) = \infty$.

We construct d and F' by induction. Without loss of generality, we may assume that $F(\lambda) = 1$. First let $F'(\lambda) = F(\lambda) = 1$ and $d(\lambda) = F(\lambda) - 1 = 0$.

For induction, assume that $F'(x)$ and $d(x)$ have been defined for all $|x| \leq n$. Fix a string x of length n and $b \in \Sigma$, let $l(xb) = \frac{F(xb)}{F(x)}$ if $F(x) \neq 0$ and let $l(xb) = 0$ otherwise. For the definition of $F'(xb)$ and $d(xb)$, we distinguish the following two cases.

1. $d(x) + 1 \geq F'(x)$: Let $F'(xb) = d(x) + (F'(x) - d(x))l(xb)$ and $d(xb) = d(x)$.
2. $d(x) + 1 < F'(x)$: Let $F'(xb) = d(x) + 1 + (F'(x) - d(x) - 1)l(xb)$ and $d(xb) = d(x) + 1$.

It is straightforward that both F' and d are polynomial time computable, F' is a martingale, $d(x) \leq d(y)$ for $x \sqsubseteq y$, and $F'(x) > d(x)$ for all $x \in \Sigma^*$. Next we show that for strings $x, y \in \Sigma^*$, if $d(x) < F'(x) \leq d(x) + 1$ and $F'(xy') \leq d(x) + 1$ for all $y' \sqsubseteq y$, then we have

$$F'(xy) = \frac{F(xy)}{F(x)} \cdot (F'(x) - d(x)) + d(x). \tag{7}$$

We use induction on y to prove (7). If $y \in \Sigma$, then (7) follows from the construction. Assume that that (7) holds for $y \in \Sigma^*$ and $F'(xy) \leq d(x) + 1$. Then, by the construction, $d(xy) = d(x)$ and

$$\begin{aligned} F'(xyb) &= d(xy) + (F'(xy) - d(xy))l(xyb) \\ &= d(xy) + (F'(xy) - d(xy))\frac{F(xyb)}{F(xy)} \\ &= d(x) + \left(\frac{F(xy)}{F(x)} \cdot (F'(x) - d(x)) + d(x) - d(x)\right) \cdot \frac{F(xyb)}{F(xy)} \\ &= d(x) + \frac{F(xyb)}{F(x)} \cdot (F'(x) - d(x)). \end{aligned} \tag{8}$$

where $b = 0, 1$. Thus (7) holds for yb and the induction is complete.

Next we show that for a sequence $\xi \in \Sigma^\infty$, if F succeeds on ξ, then $\lim_n d(\xi[0..n-1]) = \infty$. We prove by induction that, for each $k \in N$, there exists $n \in N$ such that $d(\xi[0..n-1]) > k$. By the construction, $d(\lambda) \geq 0$.

Assume that $k+1 \geq F'(\xi[0..n_1-1]) > d(\xi[0..n_1-1]) = k$ for some $n_1 \in N$. Then, by (7),

$$F'(\xi[0..n-1]) = \frac{F(\xi[0..n-1])}{F(\xi[0..n_1-1])} \cdot (F'(\xi[0..n_1-1]) - d(\xi[0..n_1-1])) + d(\xi[0..n_1-1])$$

for $n \geq n_1$ until $F'(\xi[0..n-1]) > d(\xi[0..n_1-1]) + 1 = k+1$. Since F succeeds on ξ, there exists $n_2 > n_1$ such that

$$F(\xi[0..n_2-1]) > \frac{F(\xi[0..n_1-1])}{F'(\xi[0..n_1-1]) - d(\xi[0..n_1-1])}.$$

Hence there exists $n_3 \leq n_2$ such that $F'(\xi[0..n_3-1]) > d(\xi[0..n_1-1]) + 1 = k+1$ and $d(\xi[0..n_3]) \geq k+1$.

Now we are ready to construct the non-decreasing function h from d on induction. Let $h(0) = 0$ and assume that $h(n)$ is defined already. Using the Turing machine M to search for a string $x \sqsubseteq \xi[0..n]$ such that $d(x) \geq h(|x|) + 1 = h(n) + 1$. If such an x is found in n steps, then let $h(s+1) = h(s) + 1$. Otherwise let $h(s+1) = h(s)$.

It is straightforward that h is an n^2-time computable (with respect to the unary representation of numbers), unbounded, nondecreasing function and $F'(\xi[0..n-1]) \geq h(n)$ a.s. This completes the proof of the Theorem. □

9 Appendix II

Table 3 lists values $\mu_n^U(I)$ for 0.05-length intervals I with $n = 2^{26}, \cdots, 2^{34}$. Since $\mu_n^U(I)$ is symmetric, it is sufficient to list the distribution in the positive side of the real line. Table 4 lists values $\mu_n^{nistDRBsha1}(I)$ for 0.05-length intervals I with $n = 2^{26}, \cdots, 2^{34}$. Table 5 lists values $\mu_n^{SHA256dyn}(I)$ for 0.05-length intervals I with $n = 2^{26}, \cdots, 2^{34}$.

Table 3. Selected values of μ_n^U induced by S_{lil} for $n = 2^{26}, \cdots, 2^{34}$ (due to symmetry, only distribution on the positive part of real line R is given)

	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}	2^{33}	2^{34}
$[0.00, 0.05)$	.047854	.048164	.048460	.048745	.049018	.049281	.049534	.049778	.050013
$[0.05, 0.10)$	.047168	.047464	.047748	.048020	.048281	.048532	.048773	.049006	.049230
$[0.10, 0.15)$	.045825	.046096	.046354	0.04660	.046839	.047067	.047287	.047498	.047701
$[0.15, 0.20)$	.043882	.044116	.044340	.044553	.044758	.044953	.045141	.045322	.045496
$[0.20, 0.25)$	.041419	.041609	.041789	.041961	.042125	.042282	.042432	.042575	.042713
$[0.25, 0.30)$	.038534	.038674	.038807	.038932	.039051	.039164	.039272	.039375	.039473
$[0.30, 0.35)$	.035336	.035424	.035507	.035584	.035657	.035725	0.03579	.035850	.035907
$[0.35, 0.40)$	.031939	.031976	.032010	.032041	.032068	.032093	.032115	.032135	.032153
$[0.40, 0.45)$	.028454	.028445	.028434	.028421	.028407	.028392	.028375	.028358	.028340
$[0.45, 0.50)$	.024986	.024936	.024886	.024835	.024785	.024735	.024686	.024637	.024588
$[0.50, 0.55)$	.021627	.021542	.021460	.021379	.021300	.021222	.021146	.021072	.020999
$[0.55, 0.60)$	.018450	.018340	.018234	.018130	.018029	.017931	.017836	.017743	.017653
$[0.60, 0.65)$	.015515	.015388	.015265	.015146	.015032	.014921	.014813	.014709	.014608
$[0.65, 0.70)$	.012859	.012723	.012591	.012465	.012344	.012227	.012114	.012004	.011899
$[0.70, 0.75)$	.010506	.010367	.010234	.010106	.009984	.009867	.009754	.009645	.009541
$[0.75, 0.80)$	.008460	.008324	.008195	.008072	.007954	.007841	.007733	.007629	.007530
$[0.80, 0.85)$	.006714	.006587	.006466	.006351	.006241	.006137	.006037	.005941	.005850
$[0.85, 0.90)$	.005253	.005137	.005027	.004923	.004824	.004730	.004640	.004555	.004474
$[0.90, 0.95)$	.004050	.003948	.003851	.003759	.003672	.003590	.003512	.003438	.003368
$[0.95, 1.00)$	.003079	.002990	.002906	.002828	.002754	.002684	.002617	.002555	.002495
$[1.00, \infty)$	.008090	.007750	.007437	.007147	.006877	.006627	.006393	.006175	.005970

Table 4. Selected values of $\mu_n^{nistDRBGsha1}$ induced by S_{lil} for $n = 2^{26}, \cdots, 2^{34}$

	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}	2^{33}	2^{34}
$(-\infty, -1)$	.009	.008	.007	.008	.006	.007	.007	.006	.007
$[-0.1, -0.95)$	.002	.004	.001	.005	.002	.003	.003	.001	.005
$[-0.95, -0.90)$	.004	.007	.004	.005	.002	.006	.004	.002	.000
$[-0.90, -0.85)$	.009	.006	.011	.008	.005	.003	.006	.006	.009
$[-0.85, -0.80)$	.005	.010	.004	.010	.008	.003	.004	.010	.003
$[-0.80, -0.75)$	.007	.004	.010	.011	.006	.008	.011	.005	.002
$[-0.75, -0.70)$	.009	.005	.014	.008	.011	.017	.007	.013	.011
$[-0.70, -0.65)$	.019	.014	.014	.011	.026	.015	.012	.013	.009
$[-0.65, -0.60)$	.013	.020	.010	.012	.018	.011	.014	.012	.011
$[-0.60, -0.55)$	.016	.021	.019	.014	.019	.022	.021	.018	.017
$[-0.55, -0.50)$	.022	.018	.022	.027	.028	.022	.023	.023	.023
$[-0.50, -0.45)$	.027	.025	.020	.033	.021	.029	.025	.026	.034
$[-0.45, -0.40)$	.028	.030	.024	.027	.025	.033	.034	.028	.035
$[-0.40, -0.35)$	.030	.036	.031	.026	.027	.026	.037	.041	.036
$[-0.35, -0.30)$	.041	.032	.037	.035	.032	.026	.040	.039	.038
$[-0.30, -0.25)$	.034	.043	.052	.038	.039	.032	.034	.032	.048
$[-0.25, -0.20)$	.045	.031	.048	.038	.038	.046	.036	.030	.044
$[-0.20, -0.15)$	.055	.044	.048	.039	.039	.042	.046	.051	.050
$[-0.15, -0.10)$	.056	.058	.046	.046	.041	.050	.046	.050	.042
$[-0.10, -0.05)$	.046	.048	.048	.044	.044	.051	.046	.059	.039
$[-0.05, 0)$	.045	.050	.035	.051	.040	.053	.048	.059	.048
$[0, 0.05)$	.045	.040	.051	.052	.047	.041	.033	.044	.042
$[0.05, 0.10)$	.058	.038	.060	.047	.056	.044	.044	.056	.051
$[0.10, 0.15)$	.042	.044	.035	.041	.057	.047	.050	.040	.048
$[0.15, 0.20)$	.037	.040	.040	.051	.039	.049	.045	.038	.033
$[0.20, 0.25)$	.034	.050	.037	.056	.045	.039	.046	.039	.033
$[0.25, 0.30)$	.042	.041	.034	.046	.042	.032	.037	.039	.035
$[0.30, 0.35)$	.036	.036	.040	.035	.036	.031	.043	.037	.040
$[0.35, 0.40)$	.022	.038	.028	.033	.045	.029	.043	.032	.038
$[0.40, 0.45)$	.029	.020	.026	.023	.037	.036	.031	.018	.034
$[0.45, 0.50)$	.025	.026	.028	.023	.019	.029	.020	.019	.026
$[0.50, 0.55)$	.024	.025	.034	.019	.012	.031	.024	.023	.031
$[0.55, 0.60)$	.020	.012	.016	.015	.023	.020	.019	.022	.014
$[0.60, 0.65)$	.010	.016	.011	.014	.013	.019	.011	.011	.015
$[0.65, 0.70)$	.012	.013	.011	.008	.015	.012	.010	.013	.013
$[0.70, 0.75)$	.006	.012	.011	.008	.012	.011	.011	.014	.006
$[0.75, 0.80)$	.010	.011	.005	.012	.009	.006	.009	.006	.011
$[0.80, 0.85)$	.006	.005	.006	.005	.006	.005	.002	.008	.006
$[0.85, 0.90)$	.005	.003	.006	.003	.002	.005	.001	.007	.005
$[0.90, 0.95)$	.005	.007	.003	.002	.003	.004	.006	.004	.002
$[0.95, 1.00)$	.002	.004	.003	.004	.001	.001	.003	.001	.001
$[1.00, \infty)$	.008	.005	.010	.007	.004	.004	.008	.005	.005

Table 5. Selected values of $\mu_n^{SHA256dyn}$ induced by S_{lil} for $n = 2^{26}, \cdots, 2^{34}$

	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}	2^{33}	2^{34}
$(-\infty, -1)$	0.1	0.12	0.12	0.12	0.16	0.24	0.1	0.2	0.2
$[-0.1, -0.95)$	0.04	0.04	0.06	0.08	0.1	0.06	0.06	0.1	0.04
$[-0.95, -0.90)$	0.06	0.04	0.08	0.14	0.04	0.12	0.08	0.06	0.04
$[-0.90, -0.85)$	0.12	0.14	0.12	0.1	0.12	0.04	0.08	0.12	0.08
$[-0.85, -0.80)$	0.1	0.12	0.14	0.16	0.1	0.14	0.2	0.1	0.08
$[-0.80, -0.75)$	0.14	0.18	0.12	0.22	0.12	0.2	0.1	0.14	0.18
$[-0.75, -0.70)$	0.16	0.2	0.2	0.14	0.16	0.24	0.36	0.28	0.18
$[-0.70, -0.65)$	0.36	0.22	0.26	0.1	0.2	0.2	0.2	0.2	0.1
$[-0.65, -0.60)$	0.36	0.28	0.42	0.46	0.26	0.24	0.28	0.32	0.32
$[-0.60, -0.55)$	0.34	0.4	0.26	0.3	0.26	0.36	0.44	0.42	0.36
$[-0.55, -0.50)$	0.46	0.3	0.38	0.44	0.34	0.3	0.24	0.46	0.42
$[-0.50, -0.45)$	0.58	0.56	0.64	0.6	0.48	0.64	0.5	0.56	0.5
$[-0.45, -0.40)$	0.56	0.56	0.46	0.52	0.58	0.54	0.54	0.58	0.62
$[-0.40, -0.35)$	0.82	0.68	0.66	0.76	0.88	0.5	0.6	0.52	0.62
$[-0.35, -0.30)$	0.68	0.36	0.72	0.64	0.6	0.68	0.84	0.78	0.96
$[-0.30, -0.25)$	0.84	0.84	1.02	0.84	0.68	0.64	0.68	0.78	0.9
$[-0.25, -0.20)$	0.74	0.96	0.88	0.98	0.8	1.04	0.84	0.82	0.94
$[-0.20, -0.15)$	0.88	0.9	0.82	0.82	0.96	1.06	0.96	0.8	0.82
$[-0.15, -0.10)$	0.92	0.94	0.64	0.88	1.12	1.2	0.94	1.02	0.98
$[-0.10, -0.05)$	0.94	0.76	1.16	0.78	1.08	1	1.08	0.88	1
$[-0.05, 0)$	0.74	1.24	0.92	0.78	1.34	0.58	0.96	0.92	0.9
$[0, 0.05)$	1.06	1.08	1.24	0.94	0.9	0.96	1.08	0.84	0.86
$[0.05, 0.10)$	0.82	1.16	0.8	0.82	0.92	0.88	0.72	1.02	1
$[0.10, 0.15)$	0.8	0.82	0.82	1.04	0.8	0.88	0.98	1.04	0.88
$[0.15, 0.20)$	0.98	0.86	0.9	0.94	0.96	0.94	0.64	0.94	1.06
$[0.20, 0.25)$	0.78	0.78	0.8	0.74	0.76	0.74	0.98	0.88	0.88
$[0.25, 0.30)$	0.86	0.68	0.62	0.58	0.7	0.8	0.84	0.6	0.66
$[0.30, 0.35)$	0.74	0.9	0.82	0.76	0.52	0.62	0.82	0.72	0.5
$[0.35, 0.40)$	0.64	0.64	0.76	0.92	0.58	0.66	0.6	0.62	0.62
$[0.40, 0.45)$	0.5	0.68	0.46	0.58	0.66	0.5	0.42	0.5	0.48
$[0.45, 0.50)$	0.48	0.48	0.48	0.52	0.36	0.5	0.6	0.5	0.74
$[0.50, 0.55)$	0.42	0.4	0.34	0.52	0.46	0.32	0.32	0.52	0.3
$[0.55, 0.60)$	0.44	0.26	0.38	0.32	0.24	0.64	0.34	0.4	0.52
$[0.60, 0.65)$	0.24	0.32	0.22	0.34	0.38	0.34	0.32	0.24	0.36
$[0.65, 0.70)$	0.3	0.18	0.28	0.14	0.3	0.24	0.18	0.32	0.12
$[0.70, 0.75)$	0.28	0.26	0.16	0.12	0.24	0.22	0.24	0.22	0.12
$[0.75, 0.80)$	0.12	0.18	0.18	0.34	0.14	0.12	0.26	0.08	0.12
$[0.80, 0.85)$	0.12	0.08	0.12	0.12	0.14	0.14	0.14	0.14	0.14
$[0.85, 0.90)$	0.18	0.1	0.16	0.04	0.12	0.2	0.08	0.1	0.06
$[0.90, 0.95)$	0.04	0.02	0.12	0.1	0.12	0.1	0.1	0.04	0.08
$[0.95, 1.00)$	0.08	0.04	0.04	0.02	0.14	0.08	0.04	0.04	0.08
$[1.00, \infty)$	0.18	0.24	0.22	0.24	0.18	0.1	0.22	0.18	0.18