THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2014-01

Obtaining Statistically Random Information from Silicon Physical Unclonable Functions

Chi-En Yin and Gang Qu

Obtaining Statistically Random Information from Silicon Physical Unclonable Functions

Chi-En Yin and Gang Qu
Department of Electrical and Computer Engineering & Institute for Systems Research
University of Maryland, College Park, MD 20742, USA
Emails: *daniel.yin@gmail.com* and *gangqu@umd.edu*

***Abstract*—Silicon physical unclonable functions (PUF) utilize the variation during silicon fabrication process to extract information that will be unique for each chip. There have been many recent approaches to how PUF can be used to improve security related applications. However, it is well-known that the fabrication variation has very strong spatial correlation[1] and this has been pointed out as a security threat to silicon PUF. In fact, when we apply NIST's statistical test suite for randomness [1] against the random sequences generated from a population of 125 ring oscillator (RO) PUFs [2] using classic 1-out-of-8 Coding [3], [4] and Neighbor Coding [5], none of them can pass all the tests. In this paper, we propose to decouple the unwanted systematic variation from the desired random variation through a regression-based distiller, where the basic idea is to build a model for the systematic variation so we can generate the random sequences only from the true random variation. Applying Neighbor Coding to the same benchmark data [2], our experiment shows that 2^{nd} and 3^{rd} order polynomials distill random sequences that pass all the NIST randomness tests. So does 4^{th} order polynomial in the case of 1-out-of-8 Coding. Finally, we introduce two generic random sequence generation methods. The sequences they generate fail all the randomness tests, but with the help of our proposed polynomial distiller, all but one tests are passed. These results demonstrate that our method can provide statistically random PUF information and thus bolster the security characteristics of existing PUF schemes.**

***Index Terms*—ring oscillator (RO), physical unclonable functions (PUFs), linear regression, variation decomposition**

I. Introduction

A. Overview

One of the most renowned principles for the design of a cryptosystem is Kerckhoffs's law: "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge (1883)." In order to provide a secure storage for cryptographic keys, contemporary tamper-resistant devices such as smart cards arm themselves with a number of countermeasures to defeat various kinds of invasive, semi-invasive and non-invasive physical attacks. [6], [7], [8], [9], [6], [10], [11]. Nevertheless, it is still possible for attackers to read, and possibly write, the secret bits in the non-volatile memory through the electron beam of a Scanning Electron Microscope (SEM) once the surface of the chip is exposed by, for instance, Focused Ion Beam (FIB). [12], [13]. Physical unclonable functions (PUFs), in contrast, are 'inseparable' because the underlying nano-scale structural disorder will most likely be damaged during the course of physical tampering of the device, so will the keys [14]. Since the first introduction of PUFs in [15], many types of circuitry have been proposed to realize the notion. Most notable are Arbiter PUFs [16], [17] RO PUFs [3], [16] and SRAM PUFs [18], [19]. Many methodologies have been proposed to advance the art in terms of reliability, [20], [21], [5], [4] security, [22], [17], [23], [20], [24], [5], [4], [25], [26], [27], [28] and hardware efficiency. [20], [21], [5], [29], [30], [31].

Researchers further classify PUFs as 'Strong', 'Controlled', and 'Weak' mainly according to the number of challenge-response pairs (CRPs) a PUF can generate, where the words *weak* and *strong* are irrelevant to the strength of PUF security [27]. Considering that a large number of CRPs can be achieved through a keyed hash function seeded by a Weak PUF, we chooses Weak PUFs as the context of the discussion, though the proposed methodology is expected to work well with Strong PUFs in a similar fashion.

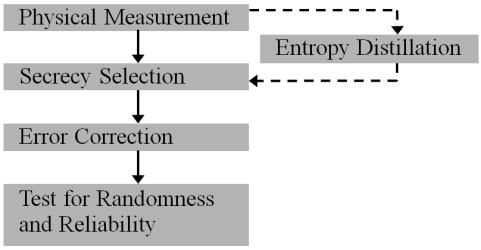

Fig. 1. The typical workflow of a Weak RO PUF

Figure 1 outlines the typical workflow of a Weak RO PUF that involves the following steps.

1) Fabrication Variation Extraction: The very first task of PUFs is to measure the unique characteristics endowed from the uncontrollable fabrication process. The analog-to-digital transformation is part of the physical entropy source subject to tests and in our case, this step corresponds to a full frequency characterization of a RO array [2].

2) Secret Selection: This step selects secure and reliable secrecy out of the variation profile measured in the previous step. Existing approaches include the classic 1-out-of-8 Coding [3] and its successor Index-Based Syndrome (IBS)

[1]Spatial correlations and systematic fabrication variations referred hereafter are different for each PUF.

Coding [4], Chain-like Neighbor Coding [5], [2], [32], Temperature-Aware Cooperative (TAC) Coding [29] and Group-Based Coding [30], [31].

3) Error Correction: In addition to the above error prevention methods, error correcting codes (ECC) are applied to further enhance reliability. Codes that have been used for RO PUFs include Hamming Code, BCH Code [3], Repetition Code [21], [4], Reed-Muller Code [20], [21], and Kendall Syndrome Code [31].

4) Tests for Randomness and Reliability: The security aspects of the PUF secrect can be judged by its statistical characteristics or randomness. Reliability, on the other hand, can be gauged by placing the device under extreme conditions for secret regeneration and failure rate below 1 part per million (ppm) has been reported under severe fluctuation of ambient temperature and supply voltage [4].

B. Motivation

Many cryptography applications such as key generation require random numbers. NIST has established several standards for cryptographically secure pseudo-random number generator (PRNG) as well as a statistical test suite for random and pseudo-random number generators [1]. If the numbers produced by a PRNG fail to pass the NIST test, it is considered vulnerable against cryptanalysis. Therefore, it is critical to verify that the secrecy generated by PUF is random and can pass the NIST test.

We consider the public available RO PUF data obtained from frequency characterization on 125 FPGA devices [2]. To our surprise, none of their random sequence can pass all the NIST tests that are applicable to their sequence length. Table I shows the detailed testing results. Column 1 lists the 9 statistical random tests we find applicable to the length of our test sequences.[2][3] Take Frequency Test for example, it examines whether the number of 1's and 0's in a sequence are approximately the same as would be expected for a truly random sequence, for which the number of 1's and 0's in a sequence should be about the same. If a sequence has a very disproportional 1's to 0's such that its *P-value*, the probability for events that at least as extreme as this instance to occur, is smaller than a significant level α, 1% in our case, the event is regarded significant. If it turns out that more than 4% of the total test sequences are significant, the 'PROPORTION' of a test fails; otherwise, it passes. Furthermore, the *P-values* of all the test sequences are expected to be uniformly distributed. To examine this, each of the 9 tests also calculates the *P-value* of the *P-values* using the χ^2 statistic. The 'P-VALUE (OF P-VALUES)' of a test fails if the *P-value* of χ^2 is smaller than 0.0001; otherwise, it passes.

[2]192 bits for the 1-out-of-8 coding. 480 bits for the chain-like neighbor coding. Choice of length may impact results.

[3]Tests like Rank Test, DFT Test, Overlapping Template Matching Test, Linear Complexity Test, Random Excursions Test and Random Excursion Variant Test require a longer random sequence over 1000 bits and thus not applicable to a 'true' random number generator like ours.

C. Contribution

Table I clearly indicates that none of the 125 PUF sequences can be deemed 'ideally random' and therefore cannot be used for critical cryptography applications. We will analyze why these PUF secrecy fails to pass the randomness tests in Section III. We argue that the systematic fabrication variation of the semiconductory process causes such failures. We study the architectures of the current RO PUFs and the aforementioned 4-step secret generation workflow. We propose to add one more step before the secret selection, which we refer to as *entropy distillation*, such that we can decouple the unwanted systematic variation from the desired stochastic variation.

There are three main contributions in this work. First, as we have pointed out in [33], this is the first work that evaluates the randomness of the random sequences generated by different PUF schemes, before being randomized by means such as hashing, against the NIST standard test. As researchers have suspected, none of them can pass all the randomness tests (results are detailed in Section V). Second, we propose to decouple the unwanted systematic variation from the desired random variation through a regression-based distiller, and then generate the random sequences based on the desired random variation. Third, we describe how to design such regression-based distillers and show their effectiveness in enhancing the randomness of the random sequences. Indeed, with the help of our distillers, previously proposed PUF schemes are able to generate random sequences that can pass all the NIST tests.

We demonstrate our approach by the example of RO PUF, but the proposed method can be applied to other PUFs to enhance their security as well (of course, for those silicon PUFs that are more resistant to systematic variation, it will be less effective). As for the implementation of the proposed method, one can either implement the distiller with hardware or rely on a secure ALU for the data process.

In the remainder of this paper, we will first introduce the basics on RO PUFs in Section II. We analyze the possible causes for the above randomness test failures in Section III. Then in Section IV, we elaborate our regression-based distiller which eliminates the systematic variation and thus fix the randomness test failures. Finally, we report the detailed experimental results and conclude.

II. Preliminaries

A. Basics on RO PUF

A RO PUF extracts fabrication variations through comparing the frequencies of ring oscillators that are identically designed. As depicted in Figure 2, the basic RO PUF consists of two ring oscillators, followed by two counters and one comparator at the end. When the start/stop control signal is asserted, the two ROs start to oscillate until the control signal is negated. The result of the race between the two ROs is determined by fabrication variations. During the course of the race, the two counters count the number of

STATISTICAL TEST	**1-out-of-8** P-VALUE	PROPORTION	**chain-like neighbor** P-VALUE	PROPORTION	**decoupled neighbor** P-VALUE	PROPORTION
Frequency	0.013689	122/125	0.000072 *	125/125	0.000003 *	115/125 *
BlockFrequency	0.166594	125/125	0.000000 *	125/125	0.050764	120/125
CumulativeSums (m-2)	0.231636	121/125	0.000000 *	125/125	0.000000 *	119/125 *
CumulativeSums (m-3)	0.059743	122/125	0.000000 *	125/125	0.000000 *	118/125 *
Runs	0.002320	117/125 *	0.000000 *	0/125 *	0.302788	120/125
LongestRun	0.000603	123/125	0.000000 *	62/125 *	0.000062 *	124/125
ApproximateEntropy	0.000001 *	117/125 *	0.000000 *	0/125 *	0.000001 *	119/125 *
Serial (forward)	0.004904	124/125	0.000000 *	1/125 *	0.070160	116/125 *
Serial (backward)	0.552185	125/125	0.000000 *	117/125 *	0.192277	123/125

TABLE I

NIST TEST RESULTS WITH RESPECT TO THE RANDOM SEQUENCES GENERATED BY **1-OUT-OF-8 CODING**, **CHAIN-LIKE NEIGHBOR CODING** AND **DECOUPLED NEIGHBOR CODING**, WHERE '*' MARKS A FAILURE.

logic cycles run by the respective RO. At the end of race, the comparator outputs a binary result x based on the two counter values, say,

$$x = \begin{cases} 1 & \text{if Counter 1} > \text{Counter 2} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

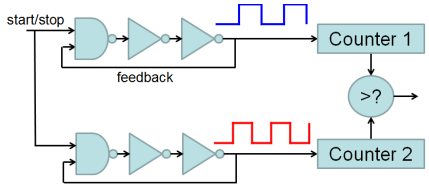

Fig. 2. The physical structure of a RO PUF [3]

To generate a secret in greater length, a RO PUF typically implements hundreds of ROs arranged in a 2-dimensional array. As illustrated in Figure 3, the dataset we use implements ROs in 16 (columns) by 32 (rows) on each their 125 FPGAs [2].

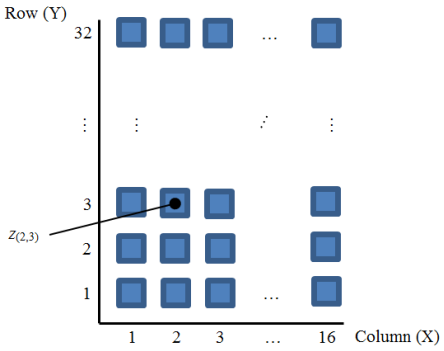

Fig. 3. The placement of 512 ROs as a 16 (columns) by 32 (rows) array; for site $RO_{x,y}$, its running frequency is denoted as $z_{x,y}$

B. 1-out-of-8 Coding

The result of the race between the same two ROs may differ when the environmental conditions change. For example, a RO running faster than its peer at low temperature can actually be slower than the same peer at high temperature [3]. To prevent this, the 1-out-of-8 coding scheme uses a multiplexer to select the pair with the largest frequency difference out of 8 RO pairs as depicted in Figure 4. For the two dimensional RO array illustrated in Figure 3, we may generate one random bit for each row j from its 16 ROs $RO_{1,j} \ldots RO_{16,j}$. However, in order to generate more random bits to better serve the statistical test purpose, we made a variation by forming two blocks $(RO_{1,j} \ldots RO_{8,j})$ and $(RO_{9,j} \ldots RO_{16,j})$ for each row. The 8 ROs in the same block are referenced by a 3-bit index: 000,001,010 $\ldots$111, and the index to the fastest RO is output as the random bits. This way we generate 6 random bits for each row.

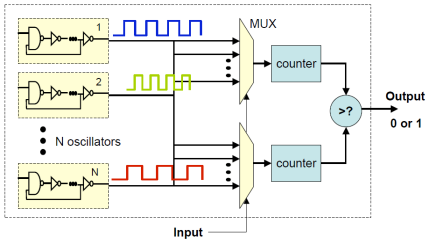

Fig. 4. The hardware structure of the 1-out-of-8 RO PUF [3]

C. Chain-like neighbor coding

Another well-known pairing strategy is the chain-like neighbor coding, which consists of two design principles: 1) place ROs as close as possible and 2) pair ROs located adjacent to each other. In the two dimensional setting of Figure 3, we may derive 15 random bits for each row j by pairing $(RO_{1,j}, RO_{2,j})$, $(RO_{2,j}, RO_{3,j})$, $(RO_{3,j}, RO_{4,j}) \ldots (RO_{15,j}, RO_{16,j})$.

III. SECURITY ANALYSIS

A. Failure Cause 1: Chain Dependency

The high failure rate of the chain-like neighbor coding can be attributed to the non-independent comparison chain. Take 3 ROs RO_A, RO_B and RO_C for example, two random bits are generated by comparing RO_A with RO_B and RO_B with RO_C. As we know, to pass NIST test for randomness, the random sequence is expected to demonstrate no significant deviation from the probability mass function (p.m.f) of tossing a fair coin twice, i.e., the 4 possible outcomes '00', '01', '10' and '11' are expected to occur equally with probability 1/4 is not the case for the two bits we generate from the 3 ROs. Let RO_i also denote

the running frequency of RO_i. For three ROs, their running frequencies can have six different orders: $RO_A < RO_B < RO_C$, $RO_A < RO_C < RO_B$, $RO_B < RO_A < RO_C$, $RO_B < RO_C < RO_A$, $RO_C < RO_A < RO_B$, $RO_C < RO_B < RO_A$, where each order happens with probability 1/6 when the running frequencies are random and identical and independent distributed (i.i.d.). According to Eqn. (1), both bits x_{AB} and x_{BC} will be equally likely to be '0' or '1'. However, the 2-bit data $x_{AB}x_{BC}$ will be '00', '01', '10', and '11' with probabilities 1/6, 1/3, 1/3, and 1/6 respectively. This means that '01' or '10' occur twice as frequent as '00' or '11', clearly not the p.m.f. of the ideal random sequences.

A simple solution to fix this problem caused by the chain dependency is to break the chain such that each RO will only be paired with its neighbor once as follows: $(RO_1, RO_2), (RO_3, RO_4) \ldots (RO_{2i-1}, RO_{2i}) \ldots$. We refer to this as *decoupled neighbor coding*. Apparently this is less efficient than the original chain-like neighbor coding. For example, when there are n ROs, the chain-like neighbor coding will generate $n-1$ bits, but the decoupled neighbor coding can only generate $\frac{n}{2}$ bits. However, even with this hardware cost, we are unable to produce true random sequence. As the last two columns in Table I show, the decoupled neighbor coding scheme helps the original chain-like neighbor coding in passing four out of the nine 'P-VALUE (OF P-VALUES)' tests, and improves the 'PROPORTION' tests too. But it still fails half of the NIST statistical randomness tests.

B. Failure Cause 2: Spatial Correlation

Chain dependency does not exist in the 1-out-of-8 coding so to investigate the cause of the failures of the 1-out-of-8 coding, we investigate the raw data from the physical measurement of fabrication variation. Figure 5 shows how the fabrication variation of the semiconductor process portrays: the roughness of the surface (random variation) is superimposed upon a spatial trend (systematic variation). With the existence of systematic variation, the 'random' bits generated by RO PUF arrays may have very low min-entropy[4], which means that they may not be secure for cryptographic purpose. For example, as we can see in Figure 5, along the row (Y), the RO's frequency tends to increase as the Y index increases. Therefore, for the two bits x_{A1A2} and x_{B1B2} generated by two pairs of ROs (A1, A2) and (B1, B2), they are more likely to be '0' at the same time. Similarly, on a different die where the frequency tends to increase along the row (Y), these two bits are more likely to be '1' at the same time. This means that the systematic variation (the spatial correlation for two ROs on the same row (Y) in this case) will render the probability of $x_{A1A2} = x_{B1B2}$ much higher than 0.5, making them not as random nor independent.

[4]the min-entropy of a discrete random event x with possible outcomes $1 \ldots n$ and corresponding probabilities $p_1 \ldots p_n$ is $H_\infty(X) = \min_{i=1}^{n}(-\log p_i)$.

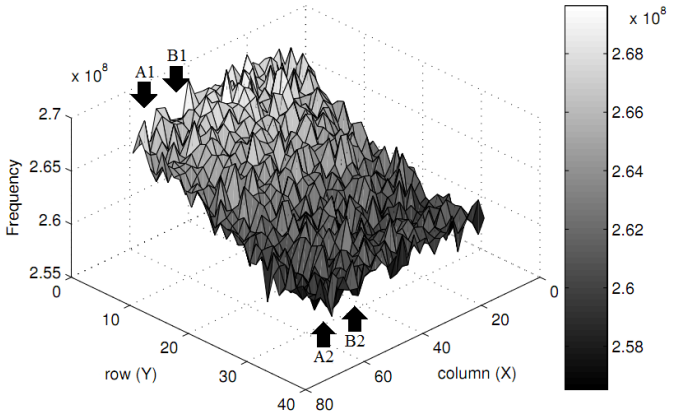

Fig. 5. The across-die frequency topology of a RO array. The roughness of the surface represents the random variation while the slope represents the systematic variation [34]

While spatial correlation may explain the reason why none of the coding strategies in Table I passes all tests, it is interesting to note that in fact this threat has been reported in the chain-like neighbor coding, where they attempt to let the systematic effect cancel out with each other, extracting secrecy out of the random effect [5]. Similar principles have been used in [32], [25]. However, the results we have in Table I indicate that such treatment is not sufficient to pass the NIST randomness tests. We postulate that a small remnant of the systematic variation can still be captured by the tests, causing the above failures. To illustrate this, we consider a hypothetical frequency characterization of 16 ROs as shown in Figure 6. Based on the chain-like neighbor coding, these 16 ROs will generate the following 15-bit sequence: 1101,1110,1001,000. The first bit is a '1' because $RO_1 < RO_2$ and the third bit is a '0' because $RO_3 > RO_4$, and so on. If our proposed decoupled neighbor coding is used, we will only have 8-bit data: 1011,1000. Although in both cases we have about the same number of 0's as the number of 1's, there is a clear trend that 1's are more likely to be in the first half of the sequence and the 0's in the second half. When we fit the frequencies into the curve in Figure 6, we see clearly the systematic trend of 'going up slope' first and then 'going down slope', which causes 0's and 1's not distributed uniformly in the sequences. Finally, we mention that this systematic trend can stay undiscovered when one tallies the total number of 0's and 1's or calculates the inter-die uniqueness via Hamming distance, e.g., 46.15-48.51% for RO PUFs [3], [32] and 49.97% for SRAM PUFs [35].

IV. Systematic Variation Elimination

We believe that one of the main causes of the failures in randomness tests for the RO PUF generated sequences is the systematic process variation. We propose to model such variation and thus remove them to build RO PUF sequences based on the true random part of the process variation. This section shows how the proposed distillation process can strengthen the randomness of the PUF output.

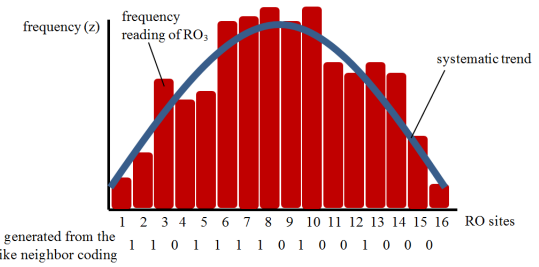

Fig. 6. Illustration of the impact from systematic variation even after decoupling.

Due to its simplicity, we apply polynomial regression to model the systematic trend. Our simulation results show that this simple model is sufficiently good as it can fix all the failures of the 1-out-of-8 coding and the decoupled neighbor coding in Table I.

A. The Causes of Process Variation

The semiconductor process variation has been modeled as the sum of a systematic component and a random component. The systematic component attempts to capture a deterministic trend and other identifiable patterns through one or a collection of estimators. The main causes of the systematic variation can be attributed to equipment and process non-uniformity such as the focus shift of photolithography, the gradient of thermal annealing, dissimilar interactions between circuit layout and the chemical mechanical polishing process [36], [37].

The random component, on the other hand, accounts for the difference between the model estimates and the observed data; its constituents include atomic-level stochastic phenomena such as random dopant profiles, measurement errors and any unidentified patterns [38], [37]. More discussion and related work on process variation modeling can be found in the Appendix.

It is important to clarify that our goal is not to build a new variation model. Indeed, the proposed distiller does not require the accuracy of the variation model to be as high as those for power or performance driven applications. In the rest of this section, we will illustrate how the distiller can improve PUF data's randomness using the simple polynomial regression model.

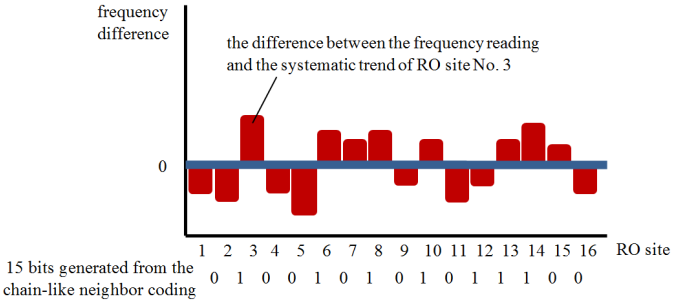

Fig. 7. The distilled random fabrication variation after the systematic trend is removed.

Let us first consider an example in Figure 7. It reports the frequency information of the same 16 RO PUF as in Figure 6 except that the Y-axis now shows the difference between each RO's frequency and the systematic trend (the bell-shape curve in Figure 6). When we use the same chain-like neighbor coding, the 15-bit sequence becomes 0100,1010,1011,100. Compared to the original sequence 1101,1110,1001,000 in Figure 6, we don't see the 'predictability' that there are more 1's in the first half and more 0's in the second half of the sequence.

B. Polynomial Regression

A k^{th}-order polynomial regression is a form of linear regression in which the relationship between independent variables and a dependent variable by a polynomial of order k, where k is a non-negative integer. For a RO PUF with its m ROs arranged in r rows by c columns, the Cartesian coordinates $(x,\ y)$ of ROs are regarded as two independent variables and the oscillating frequency z is the single variable dependent on x and y. In such a two dimensional setting, a polynomial regression model of order k takes the following general form

$$z_{x,y} = \sum_{i=0}^{k}\sum_{j=0}^{i} \beta_{k,i,j} {v_x}^{i-j} {h_y}^{j} + \epsilon_{k,x,y}, \tag{2}$$

where $1 \le x \le c, 1 \le y \le r; z, \beta, \epsilon \in \mathbb{R}$. On the right hand side of the equation, the summation term models the systematic variation at the physical location (v_x, h_y) on a chip and the residual term $\epsilon_{k,x,y}$ models the random variation. In the k^{th}-order polynomial model, there will be $m = c \times r$ equations in the form of Eqn. (2) as $1 \le x \le c$ and $1 \le y \le r$. The number of unknowns $\beta_{k,i,j}$ is $n = \frac{(k+1)(k+2)}{2}$ as $0 \le j \le i \le k$. This results in an overdetermined system (i.e. $m > n$), which can be solved by the ordinary least squares (OLS) method.

Equivalently, we can rewrite this in the matrix form

$$\boldsymbol{Z} = \boldsymbol{\Omega_k}\boldsymbol{\beta_k} + \boldsymbol{\epsilon_k} \tag{3}$$

where

$$\boldsymbol{\Omega_k} = \begin{pmatrix} \omega_{k,1,1} & \omega_{k,1,2} & \cdots & \omega_{k,1,n} \\ \omega_{k,2,1} & \omega_{k,2,2} & \cdots & \omega_{k,2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{k,m,1} & \omega_{k,m,2} & \cdots & \omega_{k,m,n} \end{pmatrix},$$

$$\boldsymbol{Z} = \begin{pmatrix} z_{1,1} \\ \vdots \\ z_{c,1} \\ z_{1,2} \\ \vdots \\ z_{c,2} \\ \vdots \\ z_{1,r} \\ \vdots \\ z_{c,r} \end{pmatrix}, \boldsymbol{\beta_k} = \begin{pmatrix} \beta_{k,0,0} \\ \beta_{k,1,0} \\ \beta_{k,1,1} \\ \beta_{k,2,0} \\ \beta_{k,2,1} \\ \beta_{k,2,2} \\ \vdots \\ \beta_{k,k,0} \\ \vdots \\ \beta_{k,k,k} \end{pmatrix}, \boldsymbol{\epsilon_k} = \begin{pmatrix} \epsilon_{k,1,1} \\ \vdots \\ \epsilon_{k,c,1} \\ \epsilon_{k,1,2} \\ \vdots \\ \epsilon_{k,c,2} \\ \vdots \\ \epsilon_{k,1,r} \\ \vdots \\ \epsilon_{k,c,r} \end{pmatrix}.$$

And $\omega_{k,p,q}$'s are in the format of ${v_x}^{i-j}{h_y}^{j}$, where $x = ((p-1) \mod r)+1$, $y = \lfloor \frac{p-1}{r} \rfloor + 1$, $i = \lfloor \frac{-1+\sqrt{1+8(q-1)}}{2} \rfloor$, $j = (q-1) - \frac{i^2+i}{2}$, $1 \le p \le m$ and $1 \le q \le n$. For example, when $k = 2$, $c = 2$, $r = 3$, we have

$$\begin{pmatrix} z_{1,1} \\ z_{2,1} \\ z_{1,2} \\ z_{2,2} \\ z_{1,3} \\ z_{2,3} \end{pmatrix} = \begin{pmatrix} 1 & v_1 & h_1 & {v_1}^2 & v_1h_1 & {h_1}^2 \\ 1 & v_2 & h_1 & {v_2}^2 & v_2h_1 & {h_1}^2 \\ 1 & v_1 & h_2 & {v_1}^2 & v_1h_2 & {h_2}^2 \\ 1 & v_2 & h_2 & {v_2}^2 & v_2h_2 & {h_2}^2 \\ 1 & v_1 & h_3 & {v_1}^2 & v_1h_3 & {h_3}^2 \\ 1 & v_2 & h_3 & {v_2}^2 & v_2h_3 & {h_3}^2 \end{pmatrix} \begin{pmatrix} \beta_{2,0,0} \\ \beta_{2,1,0} \\ \beta_{2,1,1} \\ \beta_{2,2,0} \\ \beta_{2,2,1} \\ \beta_{2,2,2} \end{pmatrix} + \begin{pmatrix} \epsilon_{2,1,1} \\ \epsilon_{2,2,1} \\ \epsilon_{2,1,2} \\ \epsilon_{2,2,2} \\ \epsilon_{2,1,3} \\ \epsilon_{2,2,3} \end{pmatrix}$$

The OLS method will find the 'best' estimate $\hat{\boldsymbol{\beta}}$ in terms of the minimum sum of squared errors as Eqn. (4) indicates. By taking partial directives of Eqn. (5) with respect to each $\beta_{k,i,j}$ and letting each gradient to zero, the solution of OLS can be expressed in the matrix form as in Eqn. (6).

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{k}} = \arg\min_{\boldsymbol{\beta}_{\boldsymbol{k}}} \left\{ \sum_{x=1,y=1}^{c,r} \epsilon_{k,x,y}^2 \right\} \tag{4}$$

$$= \arg\min_{\boldsymbol{\beta}_{\boldsymbol{k}}} \left\{ \sum_{x=1,y=1}^{c,r} (z_{x,y} - \sum_{i=0}^{k} \sum_{j=0}^{i} \beta_{k,i,j} x^{i-j} y^j)^2 \right\} \tag{5}$$

$$= (\boldsymbol{\Omega_k^T \Omega_k})^{-1} \boldsymbol{\Omega_k^T Z} \tag{6}$$

C. Regression-based Distiller

When we apply polynomial regression models to capture the systematic variation trend, higher order models have better accuracy and generate smaller residual terms ($\epsilon_{k,x,y}$'s in Eqn. (2)). While they may lead to sequences that are more random and secure, they incur more computational cost and more importantly, the small magnitude of the residual terms can cause difficulties in error correction phase and damage the efficiency of RO PUF. For example, Figure 8 shows the histograms of the random variation of a data set (see the result section for detailed description of the data) after regression models with polynomials of degrees 0 to 6 are applied. Clearly we see as the order increases, the number of ROs whose frequencies are far from the center decreases quickly, yielding a smaller variance. Nevertheless, they all appear normal distribution and it is difficult to judge which model is the best choice without running the standard randomness tests. Therefore, our goal is to find the polynomial regression model in minimal order that can successfully distill the ideal random variation. We propose to conduct this distillation procedure after the 'fabrication variation extraction' phase. The remaining question is in the next 'secret selection' phase that how to build the RO PUF sequence based on the residual terms, or the true random variations.

Suppose we have a 2-dimensional array of ROs placed in r rows and c columns (see Figure 3), there are many ways to define RO PUF bits from the distilled RO frequency information. For example, in our implementation of the 1-out-of-8 coding, each row generates $\frac{c}{8} \times 3$ bits; in the chain-like neighbor coding, we have $c-1$ bits from each row; in the decoupled neighbor coding, this number reduces to $\lfloor \frac{c}{2} \rfloor$. Of course, instead of focusing on each row, we can define RO PUF bits by comparing the ROs in the same column. In addition to these three coding schemes, we study the following two generic sequences, S and T, to gauge if there is still any trace of spatial correlation in the distilled random component:

.

$$S = X_1, \ldots, X_{l_X}, \ldots, X_{L_X} \text{ where } X_{l_X} = \begin{cases} 0 & \text{if } z_{u_X,v_X} \le z_{u_X+\lfloor \frac{c}{2} \rfloor, v_X} \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

$$T = Y_1, \ldots, Y_{l_Y}, \ldots, Y_{L_Y} \text{ where } Y_{l_Y} = \begin{cases} 0 & \text{if } z_{u_Y,v_Y} \le z_{u_Y,v_Y+\lfloor \frac{r}{2} \rfloor} \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

where in (7), $u_X = ((l_X - 1) \mod \lfloor \frac{c}{2} \rfloor) + 1$, $v_X = \lfloor (l_X - 1)/\lfloor \frac{c}{2} \rfloor \rfloor + 1$, $1 \le l_X \le L_X = r \times \lfloor \frac{c}{2} \rfloor$; similarly in (8), $u_Y = \lfloor (l_Y - 1)/\lfloor \frac{r}{2} \rfloor \rfloor + 1$, $v_Y = ((l_Y - 1) \mod \lfloor \frac{r}{2} \rfloor) + 1$, $1 \le l_Y \le L_Y = c \times \lfloor \frac{r}{2} \rfloor$.

Intuitively, S and T are formulated by cutting each row (or column in T) in the RO array into two equal halves, pairing up ROs in the two halves, and comparing their residual variation terms. Recall that the principle in neighbor coding is to pair up ROs that are next to each other in order to reduce the systematic variation. In S and T, we have purposely done the opposite to pair up ROs that are far from each other to amplify the effect of systematic variation in order to test the effectiveness of the proposed regression-based entropy distiller. In the next section, we will report our detailed findings on such randomness tests.

V. Results and Analysis on Randomness Tests

In this section, we conduct standard NIST randomness tests to validate that the proposed regression-based entropy distiller will improve the randomness of the RO PUF sequences. We use the test bench in the public domain which consists of the frequency characterization of 125 RO PUFs implemented on 125 Xilinx Spartan-3 FPGAs, where 512 ROs were placed on each FPGA as shown Figure 3 [2].

A. The Polynomial Regression Models

We first report the systematic variation distillation procedure and then results. For each chip, we apply regression models of different orders to its 512 averages of frequency readings. Figure 9 shows the modeled systematic variation for each RO on the first chip. In the 0^{th} order, the systematic variation is the average of the 512 averages. In the 1^{st} order linear model, we see that the ROs have higher frequency as their Y coordinates decrease. As we use higher order polynomials, it starts to show trend similar to Figure 5.

Figure 10 shows the random variation after distillation. We see the radial pattern close to the center for the 0^{th} and 1^{st} models, which is known as the 'bull's eye' and a clear indication of 'non-randomness'. However, it vanishes

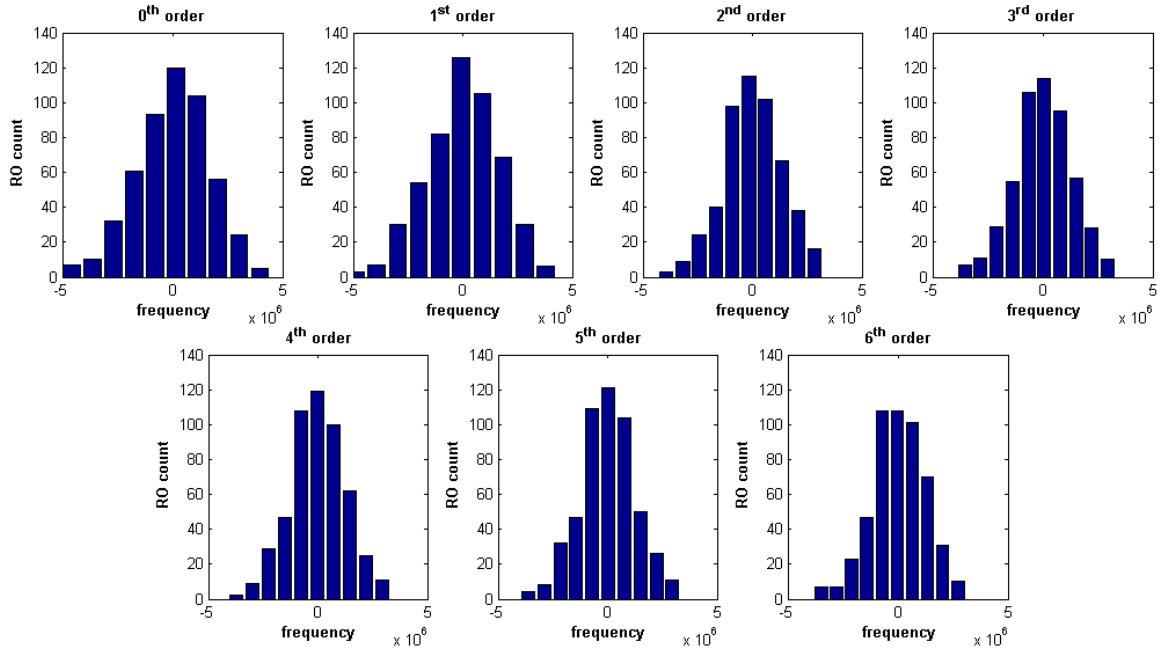

Fig. 8. The histogram of the distilled random variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1.

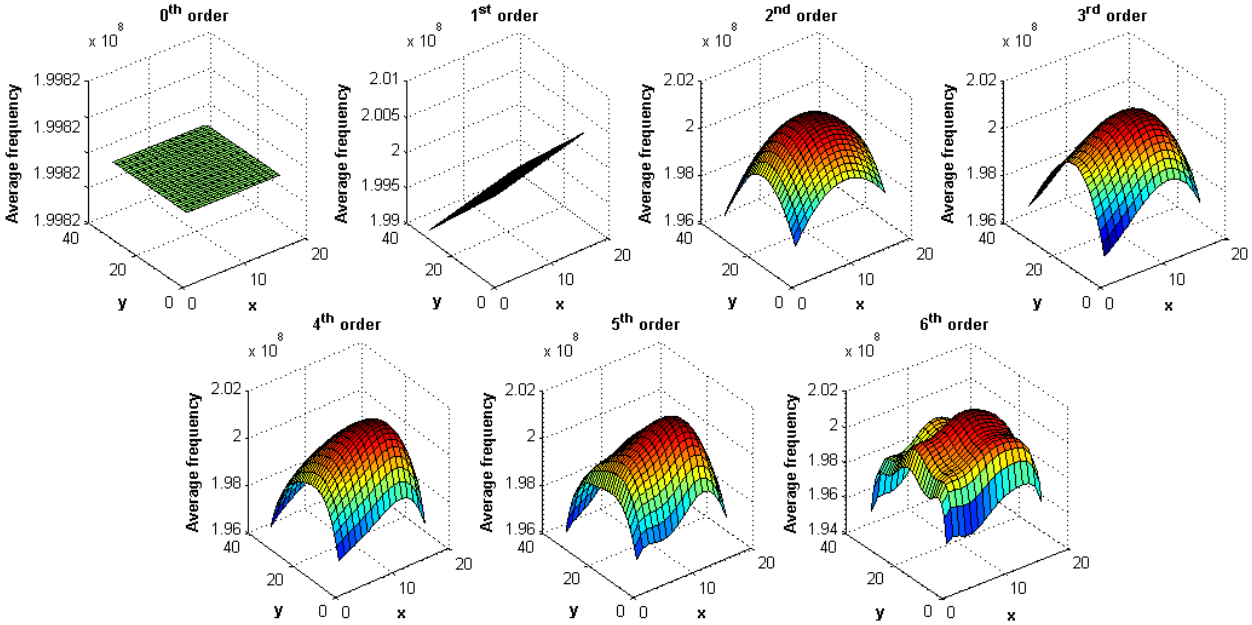

Fig. 9. The modeled systematic variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1.

in the cases of 2^{nd} model and beyond. This suggests us that polynomials of 2^{nd} degree or higher should be used.

B. NIST Randomness Tests

There are nine randomness tests in the NIST statistical test suite applicable to the length of our test sequences: Frequency Test, Block Frequency Test, Cumulative Sums Test (with block size $m = 2$ and $m = 3$), Runs Test, Longest Run Test, Serial Test (both forward and backward) and Approximate Entropy Test. According to [1], empirical results have to be interpreted in two forms of analysis: First, the proportion of sequences passing a test shall be above a minimum rate, 0.96 in our case, i.e., to pass 120 sequences out of a sample size of 125 sequences at significance level $\alpha = 0.01$. Secondly, the *P-values* of all the random sequences shall be uniformly distributed. Based on χ^2 Goodness-of-Fit Test, the underlying distribution is deemed uniform if the *P-value* of the *P-values* is equal or greater than 0.0001 for a population of 125 sequences. Whenever either of these two approaches fails, further tests based on a different sample space will help clarify whether the failure is a statistical anomaly or a clear non-randomness.

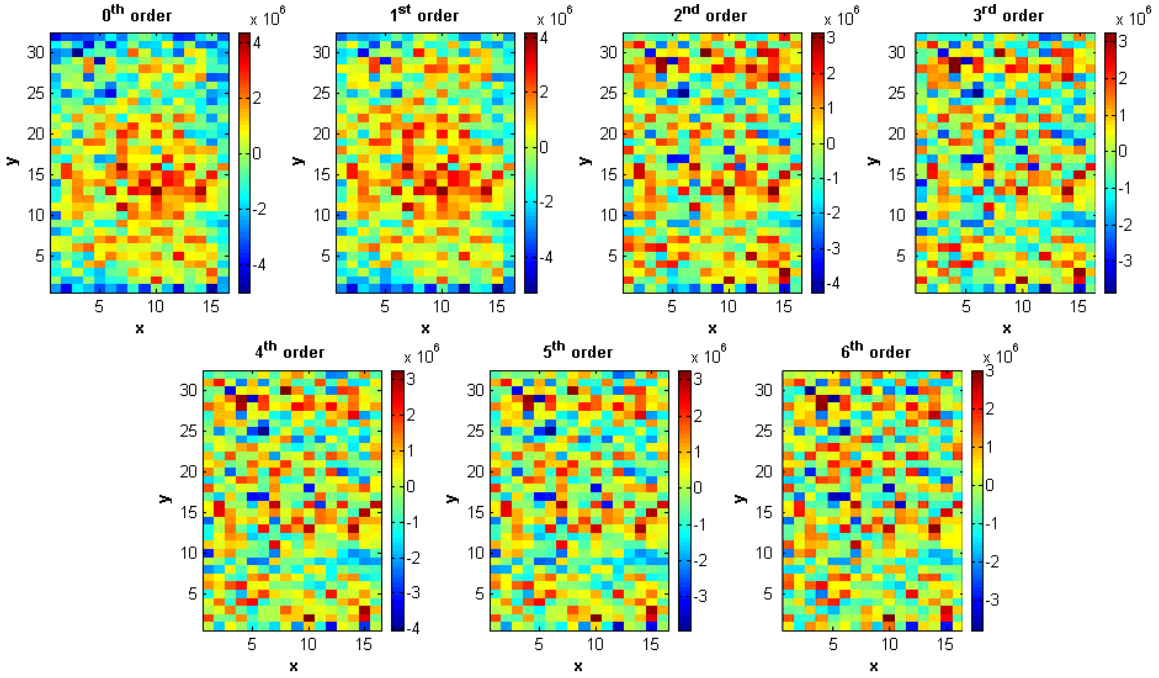

Fig. 10. The distilled random variation after applying 0^{th} through 6^{th}-order polynomial regression to the dataset of Chip No. 1. Notably, we see the 'bull's eye', i.e., the radial pattern close to the center, vanishing in the cases of 2^{nd} order model and beyond.

Table II reports the detailed test results on the generic S-sequence, T-sequence, and the sequences generated by the coding schemes of 1-out-of-8, chain-like neighbor, and decoupled neighbor.

1) S-sequence and T-sequence: The 512 ROs will generate a 256-bit S-sequence and a 256-bit T-sequence. The S-sequence and T-sequence for NIST randomness test are 32000 bits long obtained by concatenating the 125 such 256-bit sequence from the 125 chips.

As the 0^{th}-order section shows, random sequences generated without entropy distillation fail miserably for both forms of analysis 'PROP. (PROPORTION)' and 'P-VAL. (P-VALUE OF P-VALUES)', where '*' marks a failure. This strongly suggests the existence of systematic variation in the raw data. The failure rate decreases sharply when applied with 1^{st}-, 2^{nd}- or 3^{rd}-order distiller in the case of S and with 2^{nd}- or 3^{rd}-order distiller in the case of T.

Unfortunately, there is at least one failure with respect to S, though the failure is only slightly below the cutting value. In such a boarder case where a weak existence of systematic variation is inferred, further investigation with different dataset is necessary to conclude the entropy source, i.e., RO PUF plus the distillation model, 'good' or 'bad'. If simply taking the sum of failure rates with respect to S and T, either 2^{nd}- or 3^{rd}-order distillers can be considered optimal.

Finally, we mention that the pass rate of the 'P-VALUE OF P-VALUES' analysis drops when applied with a model of 4^{th} order or higher. A further investigation reveals that this is caused by model over-fitting. When we use high order models, there will be more coefficients $\beta_{k,i,j}$ (as in Eqn(2)) to describe the underlying systematic variation. In general, this will give us better model. However, considering the limited number of data samples we have (a 256-bit S-sequence or a 256-bit T-sequence), if we use a 6^{th} order polynomial model where 28 unknown $\beta_{k,i,j}$'s need to determined, the model will capture the random variation instead of the systematic variation and causes over-fitting. When that happens, there is little 'true' random variation left and thus randomness test will fail. We can see this from Table II. This result also indicates that the 2^{nd} or 3^{rd} order model should suffice with the number of data samples we have.

2) 1-out-of-8 Coding: For our implementation of the 1-out-of-8 coding, a 3-bit index '000', '001'... or '111' is generated by pointing to the fastest RO out of 8 consecutive ROs on the same row, i.e., 192 bits per chip or $125 \times 192 = 24000$ bits for the test sequence.

From Table II, we see that the 1-out-of-8 coding does very well even without the entropy distillation with only one clear 'P-VALUE' failure and two marginal 'PROPORTION' failures. The best linear model can fix all these three failures, but introduced a different 'P-VALUE' failure which is marginal. Also, distillers of 4^{th} order and higher pass all the tests and can be deemed 'good'. However, the 2^{nd} and 3^{rd} order models fail about half of the tests. We suspect that this is caused by the fact that we are collecting three bits at a time from the 3-bit index of the eight ROs. Models of low order may not be able to captuer certain intrinsic correlation behind such selection.

3) Neighbor Coding: In the case of the chain-like neighbor coding, 15 bits are generated per row by pairing up with row neighbors, which yields 480 bits per chip. Thus, the length of the test sequence is $125 \times 480 = 60000$ bits.

As shown in Table II, none of the polynomial distiller makes meaningful improvement. This phenomenon aligns with our expectation that the failures are caused by the intrinsic chain dependencies of the pairing strategy rather than spatial correlation. Moreover, consider our treatment to this problem, the decoupled neighbor coding, the 1^{st} order linear model is capable of helping it to pass all the randomness tests.

The over-fitting problem for high order polynomial models does not seem to be a concern in this case. The only exception is the 'LongestRun' test which is also the test that the 2^{nd} and 3^{rd} order models fail in the case of S-sequence. Considering the severe over-fitting problem in the case of S-sequence and T-sequence, we believe this is due to the structural difference between the decoupled neighbor coding and the S- or T-sequence. As we have discussed when defining the S-sequence and T-sequence, they are designed to amplify the systematic variation, so over-fitting is more likely to occur. In decoupled neighbor coding, we pair two ROs that are physically close to each other, hence they will have similar systematic variation that can be easily and accurately captured by the distillers.

More detailed results can be found in Appendix B.

VI. Conclusion

The systematic component of fabrication variation has long posted a security threat to RO PUFs. This work provides experimental data to demonstrate that none of the current coding schemes can pass all the NIST randomness tests. To address the issue, we propose a family of entropy distillers based on polynomial regression. We affirm their effectiveness in improving the randomness of the PUF output.

Acknowledgment

This work is supported in part by the National Science Foundation of China under grant 61228204, the Air Force Office of Scientific Research under grant FA95501010140, and University Partnership with the Laboratory of Telecommunications Sciences, contract number H9823013D00560002. Chi-En Yin is supported by Taiwan Merit Scholarships from the National Science Council of Taiwan (NSC-095-SAF-I-564-056-TMS).

References

[1] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, and L. E. B. III, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," *NIST Special Publication 800-22 Revision 1a*, Apr. 2010.

[2] A. Maiti and P. Schaumont, "A large scale characterization of ro-puf," *Proceedings of 3rd IEEE International Workshop on Hardware Oriented Security and Trust (HOST)*, Jun. 2010.

[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," *Proceedings of 44th ACM/IEEE Design Automation Conference (DAC) pp. 9–14*, Jun. 2007.

[4] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Journal of Design & Test Computers, Vol. 27, Issue 1*, Jan. 2010.

[5] A. Maiti and P. Schaumont, "Improving the quality of a physical unclonable function using configurable ring oscillators," *Proceedings of 19th IEEE International Conference on Field Programmable Logic and Applications (FPLA)*, Sep. 2009.

[6] R. Anderson, "Security engineering: A guide to building dependable distributed systems," *Wiley Computer Publishing*, Jan. 2001.

[7] R. Anderson and M. Kuhn, "Tamper resistance — a cautionary note," *Proceedings of 2nd USENIX Workshop on Electronic Commerce, pp. 1–11*, Nov. 1996.

[8] ——, "Low cost attacks on tamper resistant devices," *Proceedings of 5th International Workshop on Security Protocols, LNCS Vol. 1361, pp. 125–136, Springer*, Apr. 1997.

[9] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Proceedings of Crypto 99, LNCS Vol. 1666, Springer*, Aug. 1999.

[10] S. korobogatov and R. Anderson, "Optical fault induction attacks," *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 2523, pp. 2–12 Springer*, Aug. 2002.

[11] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor, "Improving smart card security using self-timed circuits," *Proceedings of 8th International Symposium on IEEE Asynchronous Circuits and Systems, pp. 211–218*, Apr. 2002.

[12] S. Weingart, "Physical security devices for computer subsystems: A survey of attacks and defenses," *Proceedings of 2th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 1965, pp. 302–317 Springer*, Aug. 2000.

[13] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings," *Proceedings of 8th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 4249, Springer*, Oct. 2006.

[14] P. Tuyls and B. Škorić, "Secret key generation from classical physics," *Hardware Technology Drivers of Ambient Intelligence, Philips Research Book Series, Kluwer, pp. 421–447*, 2005.

[15] R. Pappu, "Physical one-way functions," *PhD thesis*, Mar. 2001.

[16] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," *Proceedings of 9th ACM Computer and Communications Security Conference (CCS)*, Nov. 2002.

[17] D. Lim, J.-W. Lee, B. Gassend, M. van Dijk, E. Suh, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on VLSI Systems, Vol. 13, Issue 10*, Oct. 2005.

[18] D. Holcomb, W. Burleson, and K. Fu, "Initial sram state as a fingerprint and source of true random numbers for rfid tags," *Proceedings of the Conference on RFID Security 07*, Jul. 2007.

[19] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "Fpga intrinsic pufs and their use for ip protection," *Proceedings of 9th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS 4727, Springer*, Sep. 2007.

[20] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on fpgas," *Proceedings of 10th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 5154, pp. 181–197, Springer*, Aug. 2008.

[21] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for sram pufs," *Proceedings of 11th IACR International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS Vol. 5747, pp. 332–347, Springer*, Sep. 2009.

[22] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," *Proceedings of the 18th Annual Computer Security Applications Conference*, Dec. 2002.

[23] P. Tuyls, B. Skoric, and T. Kevenaar, "Security with noisy data on private biometrics, secure key storage and anti-counterfeiting," *Springer ISBN: 978-1-84628-983-5 (Print) 978-1-84628-984-2 (Online)*, 2007.

[24] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure pufs," *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2008.

[25] X. Wang and M. Tehranipoor, "Novel physical unclonable function with process and environmental variations," *Design, Automation & Test in Europe (DATE)*, Mar. 2010.

[26] M. Yu and S. Devadas, "Recombination of physical unclonable functions," *GOMACTech-10 Conference*, Mar. 2010.

	S		T		**1-out-of-8**		**chain-like neighbor**		**decoupled neighbor**		
	P-VAL.	PROP.	P-VAL.	PROP.	P-VAL.	PROP.	P-VAL.	PROP.	P-VAL.	PROP.	STATISTICAL TEST
0^{th}-order	0.000000 *	45 *	0.000000 *	38 *	0.013689	122	0.000072 *	125	0.000003 *	115 *	Frequency
	0.000000 *	59 *	0.000000 *	49 *	0.166594	125	0.000000 *	125	0.050764	120	BlockFrequency
	0.000000 *	46 *	0.000000 *	39 *	0.231636	121	0.000000 *	125	0.000000 *	119 *	CumulativeSums (m-2)
	0.000000 *	46 *	0.000000 *	38 *	0.059743	122	0.000000 *	125	0.000000 *	118 *	CumulativeSums (m-3)
	0.000000 *	65 *	0.000000 *	31 *	0.002320	117 *	0.000000 *	0 *	0.302788	120	Runs
	0.000000 *	66 *	0.000000 *	44 *	0.000603	123	0.000000 *	62 *	0.000062 *	124	LongestRun
	0.000000 *	53 *	0.000000 *	23 *	0.000001 *	117 *	0.000000 *	0 *	0.000001 *	119 *	ApproximateEntropy
	0.000000 *	65 *	0.000000 *	25 *	0.004904	124	0.000000 *	1 *	0.070160	116 *	Serial (forward)
	0.000000 *	103 *	0.000000 *	74 *	0.552185	125	0.000000 *	117 *	0.192277	123	Serial (backward)
1^{st}-order	0.166594	124	0.003598	122	0.000949	120	0.000072 *	125	0.130323	124	Frequency
	0.000002 *	120	0.889414	121	0.529142	125	0.000000 *	125	0.056599	122	BlockFrequency
	0.000100	120	0.136969	122	0.063046	120	0.000000 *	125	0.082208	124	CumulativeSums (m-2)
	0.405918	122	0.020616	119 *	0.043046	120	0.000000 *	125	0.034444	123	CumulativeSums (m-3)
	0.082208	124	0.000000 *	68 *	0.192277	124	0.000000 *	0 *	0.096097	122	Runs
	0.048059	120	0.000000 *	90 *	0.000067 *	123	0.000000 *	62 *	0.000274	124	LongestRun
	0.025948	120	0.000000 *	75 *	0.002471	120	0.000000 *	0 *	0.130323	122	ApproximateEntropy
	0.112055	122	0.000000 *	80 *	0.262219	124	0.000000 *	1 *	0.956806	122	Serial (forward)
	0.474938	121	0.000000 *	117 *	0.551044	125	0.000000 *	117 *	0.620686	123	Serial (backward)
2^{nd}-order	0.369588	122	0.012159	121	0.000000 *	115 *	0.001228	125	0.086622	121	Frequency
	0.000782	122	0.422488	122	0.059743	124	0.000000 *	125	0.262219	123	BlockFrequency
	0.020616	120	0.086622	120	0.000000 *	118 *	0.000000 *	125	0.073984	123	CumulativeSums (m-2)
	0.575157	122	0.066516	122	0.000000 *	116 *	0.000000 *	125	0.389809	120	CumulativeSums (m-3)
	0.316158	125	0.915772	122	0.552185	123	0.000000 *	0 *	0.493319	124	Runs
	0.000062 *	122	0.000782	120	0.000000 *	123	0.000000 *	58 *	0.000115	124	LongestRun
	0.750075	124	0.474938	120	0.000000 *	120	0.000000 *	0 *	0.316158	122	ApproximateEntropy
	0.874833	124	0.077998	122	0.000123	123	0.000000 *	0 *	0.643139	121	Serial (forward)
	0.231636	123	0.302788	125	0.457002	124	0.000000 *	110 *	0.262219	123	Serial (backward)
3^{rd}-order	0.011457	125	0.136969	124	0.000000 *	111 *	0.000000 *	125	0.389809	123	Frequency
	0.262219	124	0.551044	125	0.003829	123	0.000000 *	125	0.457002	122	BlockFrequency
	0.002320	125	0.000131	125	0.000000 *	112 *	0.000000 *	125	0.551044	124	CumulativeSums (m-2)
	0.002984	125	0.017315	125	0.000000 *	111 *	0.000000 *	125	0.192277	123	CumulativeSums (m-3)
	0.643139	124	0.529142	121	0.889414	124	0.000000 *	0 *	0.529142	124	Runs
	0.000058 *	123	0.012903	124	0.000000 *	120	0.000000 *	48 *	0.000000 *	123	LongestRun
	0.020616	125	0.422488	123	0.000000 *	114 *	0.000000 *	0 *	0.889414	125	ApproximateEntropy
	0.369588	124	0.915772	123	0.000017 *	121	0.000000 *	0 *	0.493319	123	Serial (forward)
	0.439517	124	0.506075	122	0.575157	124	0.000000 *	114 *	0.439517	125	Serial (backward)
4^{th}-order	0.000001 *	125	0.000000 *	125	0.000407	121	0.000000 *	125	0.192277	124	Frequency
	0.166594	125	0.000051 *	125	0.344248	123	0.000000 *	125	0.807956	124	BlockFrequency
	0.000000 *	125	0.000000 *	125	0.143910	121	0.000000 *	125	0.070160	124	CumulativeSums (m-2)
	0.000011 *	125	0.000000 *	124	0.130323	120	0.000000 *	125	0.117876	124	CumulativeSums (m-3)
	0.316158	125	0.903069	122	0.437182	125	0.000000 *	0 *	0.414457	125	Runs
	0.004904	123	0.045489	125	0.007522	120	0.000000 *	54 *	0.000000 *	123	LongestRun
	0.289860	125	0.265309	122	0.708591	122	0.000000 *	0 *	0.143910	122	ApproximateEntropy
	0.571108	125	0.665311	123	0.571108	125	0.000000 *	1 *	0.457002	123	Serial (forward)
	0.405918	123	0.283039	124	0.551044	125	0.000000 *	110 *	0.825875	124	Serial (backward)
5^{th}-order	0.000029 *	125	0.211194	125	0.316158	124	0.000000 *	125	0.096097	125	Frequency
	0.004074	125	0.000000 *	125	0.493319	124	0.000000 *	125	0.552185	124	BlockFrequency
	0.000000 *	125	0.000000 *	125	0.665311	124	0.000000 *	125	0.043046	124	CumulativeSums (m-2)
	0.000000 *	125	0.000000 *	125	0.166594	123	0.000000 *	125	0.036430	125	CumulativeSums (m-3)
	0.493319	124	0.687147	124	0.036430	125	0.000000 *	0 *	0.192277	123	Runs
	0.006661	124	0.001801	125	0.036430	124	0.000000 *	42 *	0.000000 *	124	LongestRun
	0.059743	125	0.302788	124	0.729586	125	0.000000 *	0 *	0.665311	122	ApproximateEntropy
	0.687147	125	0.304210	121	0.096097	125	0.000000 *	0 *	0.512137	124	Serial (forward)
	0.262219	123	0.789315	123	0.457002	125	0.000000 *	114 *	0.474938	125	Serial (backward)
6^{th}-order	0.000009 *	125	0.001586	125	0.001080	125	0.000000 *	125	0.231636	122	Frequency
	0.000000 *	125	0.000000 *	125	0.086622	125	0.000000 *	125	0.437182	123	BlockFrequency
	0.000000 *	125	0.000000 *	125	0.231636	125	0.000000 *	125	0.091249	123	CumulativeSums (m-2)
	0.000000 *	125	0.000000 *	125	0.050764	124	0.000000 *	125	0.211194	123	CumulativeSums (m-3)
	0.101175	125	0.130323	123	0.422488	124	0.000000 *	0 *	0.529142	122	Runs
	0.000643	124	0.007992	124	0.211194	125	0.000000 *	44 *	0.000017 *	122	LongestRun
	0.000006 *	125	0.643139	124	0.130323	124	0.000000 *	0 *	0.598008	124	ApproximateEntropy
	0.552185	124	0.289860	123	0.329976	124	0.000000 *	0 *	0.277369	123	Serial (forward)
	0.874833	124	0.529142	124	0.405918	124	0.000000 *	113 *	0.843024	121	Serial (backward)

TABLE II

THE RESULTS OF NIST 'P-VAL. (P-VALUE OF P-VALUES)' AND 'PROP. (PROPORTION)' ANALYSES WITH RESPECT TO RANDOM SEQUENCES GENERATED BY S, T, THE 1-OUT-OF-8 CODING, THE CHAIN-LIKE NEIGHBOR CODING AND THE DECOUPLED NEIGHBOR CODING ACCOMPANIED BY 0^{th}- TO 6^{th}-ORDER DISTILLERS, WHERE '*' MARKS A FAILURE.

[27] U. Ruhrmair, F. Sehnke, J. Soelter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," *Proceedings of 17th ACM Computer and Communication Security Conference (CCS)*, Oct. 2010.
[28] A. Sadeghi and D. Naccache, "Towards hardware-intrinsic security," *Springer ISBN 978-3-642-14452-3*, 2010.
[29] C.-E. Yin and G. Qu, "Temperature-aware cooperative ring oscillator puf," *Proceedings of 2nd IEEE International Workshop on Hardware Oriented Security and Trust (HOST)*, Jul. 2009.
[30] C.-E. D. Yin and G. Qu, "Lisa: Maximizing ro puf's secret extraction," *Proceedings of 3rd IEEE International Workshop on Hardware Oriented Security and Trust (HOST)*, Jun. 2010.
[31] C.-E. Yin, G. Qu, and Q. Zhou, "Design and implementation of a group-based ro puf," *Design, Automation and Test in Europe (DATE13), pp. 416-421*, March 2013.
[32] D. Merli, F. Stumpf, and C. Eckert, "Improving the quality of ring oscillator pufs on fpgas," *Proceedings of the 5th Workshop on Embedded Systems Security*, Oct. 2010.
[33] C.-E. Yin, G. Qu, and Q. Zhou, "Improving puf security with regression-based distiller," *50th ACM/IEEE Design Automation Conference (DAC13), pp. 1-6*, June 2013.
[34] P. Sedcole and P. Y. K. Cheung, "Within-die delay variability in 90nm fpgas and beyond," *Proceedings of 16th IEEE International Conference on Field Programmable Technology (FPT) pp. 97–104*, Dec. 2006.
[35] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "Physical unclonable functions and public-key crypto for fpga ip protection," *Proceedings of 17th IEEE International Conference on Field Programmable Logic and Applications (FPL*, Aug. 2007.
[36] B. E. Stine, D. S. Boning, and J. E. Chung, "Analysis and decomposition of spatial variation in integrated circuit processes and devices," *IEEE Transactions on Semiconductor Manufacturing, Vol. 10, Issue 1, pp. 24–91*, Feb. 1997.
[37] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance cmos variability in the 65-nm regime and beyond," *IBM Journal of Research and Development*, vol. 50, no. 4.5, pp. 433–449, july 2006.
[38] E. Chang, B. Stine, T. Maung, R. Divecha, D. Boning, J. Chung, G. R. K. Chang, D. Bradbury, O. Nakagawa, S. Oh, and D. Bartelink, "Using a statistical metrology framework to identify systematic and random sources of die- and wafer-level ild thickness variation in cmp processes," *International Electron Devices Meeting*, pp. 499–502, Dec. 1995.
[39] L. Cheng, P. Gupta, C. Spanos, K. Qian, and L. He, "Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability," *Proceedings of 46th ACM/IEEE International Annual Design Automation Conference (DAC)*, Jul. 2009.
[40] F. Liu, "A general framework for spatial correlation modeling in vlsi design," *Proceedings of 44th ACM/IEEE Design Automation Conference (DAC) pp. 817–822*, Jun. 2007.
[41] J. Xiong, V. Zolotov, , and L. He, "Robust extraction of spatial correlation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 4, pp. 619–631, april 2007.
[42] S. Ohkawa, M. Aoki, and H. Masuda, "Analysis and characterization of device variations in an lsi chip using an integrated device matrix array," *IEEE Transactions on Semiconductor Manufacturing, Vol. 17, Issue 2, pp. 155–165*, May 2004.
[43] T. Sato, H. Ueyama, N. Nakayama, and K. Masu, "Determination of optimal polynomial regression function to decompose on-die systematic and random variations," *Proceedings of 13th IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 518–523*, Jan. 2008.

Chi-En 'Daniel' Yin received B.S. and M.S. degrees in electrical engineering from National Taiwan University in 1999 and 2001, and Ph.D. degree in electrical and computer engineering from University of Maryland, College Park in 2012. He is currently a senior security engineer in a virtual currency startup in Silicon Valley, California. His research interests include physical unclonable function, random number generation, smart card design and security, virtual currency, and all aspects of hardware security.

Gang Qu received his Ph.D. degree in Computer Science from the University of California, Los Angeles in 2000. He is currently an Associate Professor in the Department of Electrical and Computer Engineering and the Institute for Systems Research at the University of Maryland, College Park. His current research interests are on VLSI design automation and wireless sensor networks, with special focus on security and energy efficiency. He has published more than 100 journal articles and conference papers in these fields with best paper awards in MobiCom (2001) and ASAP (2006). Dr. Qu is currently on the editorial boards of IEEE Transactions on Computers, IEEE Embedded Systems Letters, and Integration, the VLSI Journal.

Appendix A: Survey on Modeling Fabrication Variations

It is well-known that the fabrication variation across the die is not identical and independent distributed (i.i.d.) as shown in Figure 5 [34] as well as other literature [18]. Table I shows that PUF data based on the i.i.d. assumption will cause failures to the randomness tests. However, it is not uncommon to see that in many reports PUF outputs are assumed to be i.i.d. and has been affirmed by affirmed by statistical results [4]. The key difference appears to be our direct test on raw output as opposed to those obfuscated by a linear feedback shift register (LFSR) and/or an output hash function that de-correlate the challenge and the response [22], [18], [4]. In fact, the issue of spatial correlation can be more serious than one thought in that the across-die spatial variation results mostly from the deterministic across-wafer variation [39], that is, the systematic trend of a chip can be interpolated or extrapolated by the systematic trends of the surrounding chips on the same wafer.

Stine et al. [36] first proposed a comprehensive framework to identify, model and decompose fabrication variations. By analyzing the measurements collected from each chip on the same wafer, their methods can model variations at wafer level as well as at die level. However, the framework is not suitable for PUF applications because it requires revealing on-die characterization as well as trusting external information for model building, either of which would seriously compromise the security of PUFs. Also surveyed was Liu's framework for spatial correlation modeling [40], where the author used Generalized Least Square (GLS) fitting and structured correlation function to improve the accuracy of prediction for unobservable sites. In our case, nevertheless, we can always take measurement at any RO site of interest as wish. Also partly related to ours are works devoted to extracting the correlation matrix of the entire fabrication process to facilitate, for example, statistical static timing analysis (SSTA) [41], [39]. It is worth noting that the authors in [39] concluded that once across-wafer systematic variation is captured and removed by a quadratic polynomial, within-die variation no longer contains useful spatial correlation.

Since relying on external information for model building would result in a security loophole, our design principle is for each PUF device to build its own across-die systematic model. Works in line with this include [42], [34], [43], where Ohkawa et al. [42] first employed a 4^{th}-order polynomial model and demonstrated its effectiveness by contrasting the correlation coefficients of the systematic component and its random counterpart; later Sedcole et al. [34] employed a quadratic model to gauge the expected magnitude of the random variation versus the maximum magnitude of the systematic variation; more recently Sato et al. [43] argued that high order polynomials could result in an overfit and that corrected Akaike information criterion (AICc) was accurate in selecting the optimal model. Notably, 1^{st} order polynomial turned out to be the best fit in most cases with respect to the variability of NMOS threshold voltages [43]. Our specific application to RO PUFs distinguishes us from our predecessors; in particular, we impose stringent criteria on randomness. In this regard, instead of running D'Agostino-Pearson (D-P) and Kolmogorov-Smirnov (K-S) statistical tests for normality against the variation profile as in [43], we apply comprehensive tests for randomness designed by NIST. In our adaption, the optimal model is determined by the test results and thus, AICc is optional.

Appendix B: Additional Results on NIST Test Reports with C1–C10 Distribution

Tables III-VII give the complete C1-C10 distribution for each of the 5 sequences: S sequence, T sequence, and the sequences created by the 1-out-of-8 coding, neighbor coding, and decoupled neighbor coding. For example, in Table III, we can clearly see that the 0^{th} order, which corresponds to the raw data without using any regression modle is not random at all. However, when we use the proposed regression modle, the distribution becomes random.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	93	7	3	4	2	6	1	2	4	3	0.000000 *	45/125 *	Frequency
	95	8	8	3	3	1	1	4	0	2	0.000000 *	59/125 *	BlockFrequency
	95	6	3	5	4	2	3	0	3	4	0.000000 *	46/125 *	CumulativeSums (m-2)
	96	7	3	3	7	1	4	1	0	3	0.000000 *	46/125 *	CumulativeSums (m-3)
	67	5	5	13	9	6	6	4	7	3	0.000000 *	65/125 *	Runs
	82	6	7	11	6	3	1	3	3	3	0.000000 *	66/125 *	LongestRun
	88	10	4	7	3	6	1	2	2	2	0.000000 *	53/125 *	ApproximateEntropy
	81	8	7	5	10	3	1	6	1	3	0.000000 *	65/125 *	Serial (forward)
	41	12	16	6	16	10	8	5	5	6	0.000000 *	103/125 *	Serial (backward)
1^{st}-order	13	19	12	12	13	7	11	7	11	20	0.166594	124/125	Frequency
	29	20	16	16	8	9	8	6	5	8	0.000002 *	120/125	BlockFrequency
	18	26	15	16	4	8	9	9	15	5	0.000100	120/125	CumulativeSums (m-2)
	18	14	17	14	16	10	10	8	9	9	0.405918	122/125	CumulativeSums (m-3)
	12	15	19	14	11	20	9	5	9	11	0.082208	124/125	Runs
	16	17	15	21	13	7	8	13	7	8	0.048059	120/125	LongestRun
	24	14	10	15	9	10	10	15	13	5	0.025948	120/125	ApproximateEntropy
	18	16	13	15	6	11	16	4	14	12	0.112055	122/125	Serial (forward)
	15	15	17	11	10	11	7	12	9	18	0.474938	121/125	Serial (backward)
2^{nd}-order	17	12	16	13	8	11	12	9	10	17	0.369588	122/125	Frequency
	27	14	12	19	7	11	7	10	9	9	0.000782	122/125	BlockFrequency
	13	22	12	12	5	13	14	4	13	17	0.020616	120/125	CumulativeSums (m-2)
	15	16	14	10	9	11	17	9	11	13	0.575157	122/125	CumulativeSums (m-3)
	15	6	10	18	11	16	16	14	8	11	0.316158	125/125	Runs
	9	13	13	30	14	14	9	7	5	11	0.000062 *	122/125	LongestRun
	18	9	12	9	13	13	13	14	13	11	0.750075	124/125	ApproximateEntropy
	13	14	11	15	10	14	16	9	11	12	0.874833	124/125	Serial (forward)
	12	9	22	13	14	11	12	10	15	7	0.231636	123/125	Serial (backward)
3^{rd}-order	4	9	11	15	8	16	22	8	15	17	0.011457	125/125	Frequency
	8	14	14	14	6	16	11	12	17	13	0.262219	124/125	BlockFrequency
	6	5	11	10	11	14	18	8	23	19	0.002320	125/125	CumulativeSums (m-2)
	4	10	10	8	6	18	20	12	18	19	0.002984	125/125	CumulativeSums (m-3)
	9	10	14	14	11	18	11	15	13	10	0.643139	124/125	Runs
	9	5	14	24	24	14	7	5	11	12	0.000058 *	123/125	LongestRun
	5	12	12	7	15	14	20	10	9	21	0.020616	125/125	ApproximateEntropy
	7	17	15	12	9	12	16	12	15	10	0.369588	124/125	Serial (forward)
	12	19	17	12	13	12	14	10	6	10	0.439517	124/125	Serial (backward)
4^{th}-order	2	2	8	26	13	16	21	10	17	10	0.000001 *	125/125	Frequency
	11	7	8	7	11	15	16	16	19	15	0.166594	125/125	BlockFrequency
	2	3	5	11	10	24	19	8	20	23	0.000000 *	125/125	CumulativeSums (m-2)
	2	4	5	15	13	16	16	17	11	26	0.000011 *	125/125	CumulativeSums (m-3)
	11	11	13	5	16	13	10	19	16	11	0.316158	125/125	Runs
	9	6	14	22	20	10	17	5	11	11	0.004904	123/125	LongestRun
	4	11	9	15	13	12	12	17	15	17	0.289860	125/125	ApproximateEntropy
	14	11	6	13	10	15	11	12	14	19	0.571108	125/125	Serial (forward)
	8	16	11	21	11	11	9	13	13	12	0.405918	123/125	Serial (backward)
5^{th}-order	0	5	12	20	13	10	11	20	11	23	0.000029 *	125/125	Frequency
	6	10	8	8	11	16	15	14	11	26	0.004074	125/125	BlockFrequency
	0	4	8	9	14	14	16	10	16	34	0.000000 *	125/125	CumulativeSums (m-2)
	0	8	4	13	10	16	10	12	24	28	0.000000 *	125/125	CumulativeSums (m-3)
	9	13	16	10	10	8	13	11	19	16	0.493319	124/125	Runs
	9	11	18	23	17	7	13	4	12	11	0.006661	124/125	LongestRun
	5	9	11	11	14	15	22	15	15	8	0.059743	125/125	ApproximateEntropy
	15	12	15	13	10	11	11	10	18	10	0.687147	125/125	Serial (forward)
	19	11	17	13	8	11	14	12	11	9	0.262219	123/125	Serial (backward)
6^{th}-order	4	8	2	19	10	8	14	15	22	23	0.000009 *	125/125	Frequency
	2	8	12	9	6	9	10	18	20	31	0.000000 *	125/125	BlockFrequency
	2	5	4	11	7	11	20	7	19	39	0.000000 *	125/125	CumulativeSums (m-2)
	3	9	5	5	5	14	13	9	18	44	0.000000 *	125/125	CumulativeSums (m-3)
	8	18	8	6	19	14	12	17	13	10	0.101175	125/125	Runs
	7	6	16	22	23	9	7	12	8	15	0.000643	124/125	LongestRun
	1	5	16	14	13	14	9	11	13	29	0.000006 *	125/125	ApproximateEntropy
	9	6	12	14	14	13	13	13	16	15	0.552185	124/125	Serial (forward)
	8	14	11	13	10	16	13	13	14	13	0.874833	124/125	Serial (backward)

TABLE III

NIST TEST RESULTS WITH RESPECT TO RANDOM SEQUENCE S, WHERE $M = 32$ FOR BLOCK FREQUENCY TEST, $m = 2$ FOR APPROXIMATE ENTROPY TEST AND $m = 5$ FOR SERIAL TEST AND '*' MARKS A FAILURE.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	100	6	3	4	2	1	2	2	0	5	0.000000 *	38/125 *	Frequency
	88	9	3	5	4	4	5	1	6	0	0.000000 *	49/125 *	BlockFrequency
	100	5	4	5	1	3	2	2	0	3	0.000000 *	39/125 *	CumulativeSums (m-2)
	100	10	0	6	1	3	0	2	1	2	0.000000 *	38/125 *	CumulativeSums (m-3)
	108	7	4	1	2	0	1	1	1	0	0.000000 *	31/125 *	Runs
	100	4	7	7	3	2	0	1	1	0	0.000000 *	44/125 *	LongestRun
	114	3	2	1	1	1	1	2	0	0	0.000000 *	23/125 *	ApproximateEntropy
	112	5	4	1	1	0	1	1	0	0	0.000000 *	25/125 *	Serial (forward)
	73	11	8	7	10	3	4	2	1	6	0.000000 *	74/125 *	Serial (backward)
1^{st}-order	25	8	10	13	6	11	7	13	13	19	0.003598	122/125	Frequency
	14	8	13	16	12	14	10	13	12	13	0.889414	121/125	BlockFrequency
	22	11	13	7	13	13	15	12	6	13	0.136969	122/125	CumulativeSums (m-2)
	25	14	11	13	12	9	8	6	15	12	0.020616	119/125 *	CumulativeSums (m-3)
	86	14	4	6	4	4	1	2	2	2	0.000000 *	68/125 *	Runs
	66	15	13	16	5	3	3	3	0	1	0.000000 *	90/125 *	LongestRun
	78	15	7	9	2	5	2	2	2	3	0.000000 *	75/125 *	ApproximateEntropy
	80	6	9	6	4	6	6	0	5	3	0.000000 *	80/125 *	Serial (forward)
	40	15	13	14	6	8	6	6	7	10	0.000000 *	117/125 *	Serial (backward)
2^{nd}-order	20	15	14	22	6	9	8	12	7	12	0.012159	121/125	Frequency
	13	11	15	14	10	18	9	9	8	18	0.422488	122/125	BlockFrequency
	20	12	6	18	13	10	11	7	11	17	0.086622	120/125	CumulativeSums (m-2)
	18	15	9	15	9	17	13	2	13	14	0.066516	122/125	CumulativeSums (m-3)
	16	14	14	11	9	10	12	14	13	12	0.915772	122/125	Runs
	18	8	15	18	24	11	6	13	6	6	0.000782	120/125	LongestRun
	19	14	17	12	13	7	9	10	11	13	0.474938	120/125	ApproximateEntropy
	19	6	11	13	11	18	16	8	16	7	0.077998	122/125	Serial (forward)
	16	16	9	10	11	9	11	21	12	10	0.302788	125/125	Serial (backward)
3^{rd}-order	10	7	10	18	10	14	15	10	10	21	0.136969	124/125	Frequency
	10	10	12	16	9	8	13	12	17	18	0.551044	125/125	BlockFrequency
	8	9	8	15	9	10	27	5	14	20	0.000131	125/125	CumulativeSums (m-2)
	8	8	9	10	14	8	17	11	16	24	0.017315	125/125	CumulativeSums (m-3)
	14	13	13	19	11	7	12	12	11	13	0.529142	121/125	Runs
	9	11	11	24	20	10	11	11	12	6	0.012903	124/125	LongestRun
	15	10	11	6	14	11	9	18	15	16	0.422488	123/125	ApproximateEntropy
	11	9	11	15	12	12	15	11	14	15	0.915772	123/125	Serial (forward)
	11	18	15	9	13	12	9	10	12	16	0.506075	122/125	Serial (backward)
4^{th}-order	3	6	18	26	12	10	4	9	12	25	0.000000 *	125/125	Frequency
	6	10	5	5	12	15	8	19	22	23	0.000051 *	125/125	BlockFrequency
	2	5	8	18	15	9	18	8	11	31	0.000000 *	125/125	CumulativeSums (m-2)
	2	6	11	15	12	20	14	4	12	29	0.000000 *	124/125	CumulativeSums (m-3)
	14	13	12	13	8	11	11	13	14	16	0.903069	122/125	Runs
	12	6	23	15	16	8	13	8	13	11	0.045489	125/125	LongestRun
	18	7	11	7	13	19	12	11	12	15	0.265309	122/125	ApproximateEntropy
	13	16	8	13	13	15	13	15	8	11	0.665311	123/125	Serial (forward)
	14	13	13	12	12	13	18	15	10	5	0.283039	124/125	Serial (backward)
5^{th}-order	6	10	11	17	9	14	10	20	15	13	0.211194	125/125	Frequency
	6	5	4	6	7	13	18	13	22	31	0.000000 *	125/125	BlockFrequency
	2	8	8	19	5	9	16	10	19	29	0.000000 *	125/125	CumulativeSums (m-2)
	3	9	5	12	8	14	18	6	14	36	0.000000 *	125/125	CumulativeSums (m-3)
	13	17	9	17	11	10	13	11	11	13	0.687147	124/125	Runs
	11	9	17	25	18	9	13	5	7	11	0.001801	125/125	LongestRun
	9	15	15	7	14	14	15	18	12	6	0.302788	124/125	ApproximateEntropy
	13	11	11	10	10	11	17	11	20	11	0.304210	121/125	Serial (forward)
	12	13	13	16	10	16	11	12	14	8	0.789315	123/125	Serial (backward)
6^{th}-order	2	13	10	14	11	11	20	9	11	24	0.001586	125/125	Frequency
	5	2	4	8	8	8	16	22	19	33	0.000000 *	125/125	BlockFrequency
	1	4	11	10	10	12	12	15	19	31	0.000000 *	125/125	CumulativeSums (m-2)
	2	6	7	10	6	13	17	7	20	37	0.000000 *	125/125	CumulativeSums (m-3)
	23	14	12	14	7	10	13	8	13	11	0.130323	123/125	Runs
	9	12	13	24	19	6	15	8	9	10	0.007992	124/125	LongestRun
	13	17	11	18	12	10	11	12	11	10	0.643139	124/125	ApproximateEntropy
	11	14	11	18	11	13	11	4	17	15	0.289860	123/125	Serial (forward)
	15	9	14	13	11	13	15	14	6	15	0.529142	124/125	Serial (backward)

TABLE IV

NIST TEST RESULTS WITH RESPECT TO RANDOM SEQUENCE T, WHERE $M = 32$ FOR BLOCK FREQUENCY TEST, $m = 2$ FOR APPROXIMATE ENTROPY TEST AND $m = 5$ FOR SERIAL TEST AND '*' MARKS A FAILURE.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	21	15	7	20	11	13	9	14	11	4	0.013689	122/125	Frequency
	7	19	19	10	10	12	13	7	13	15	0.166594	125/125	BlockFrequency
	21	9	9	14	13	10	18	10	11	10	0.231636	121/125	CumulativeSums (m-2)
	22	10	9	17	13	7	11	8	11	17	0.059743	122/125	CumulativeSums (m-3)
	26	20	13	11	10	11	8	8	9	9	0.002320	117/125 *	Runs
	22	13	16	13	22	11	3	11	9	5	0.000603	123/125	LongestRun
	30	21	15	8	13	7	10	3	9	9	0.000001 *	117/125 *	ApproximateEntropy
	27	16	14	11	11	8	10	11	8	9	0.004904	124/125	Serial (forward)
	10	16	18	15	9	11	10	12	11	13	0.552185	125/125	Serial (backward)
1^{st}-order	24	21	13	14	9	10	11	13	4	6	0.000949	120/125	Frequency
	11	18	16	10	10	11	14	15	10	10	0.529142	125/125	BlockFrequency
	18	22	13	11	11	8	13	6	14	9	0.063046	120/125	CumulativeSums (m-2)
	21	17	17	6	14	6	13	11	9	11	0.043046	120/125	CumulativeSums (m-3)
	17	19	13	15	6	16	11	10	10	8	0.192277	124/125	Runs
	13	13	13	13	27	20	7	9	5	5	0.000067 *	123/125	LongestRun
	22	20	16	14	12	15	7	6	7	6	0.002471	120/125	ApproximateEntropy
	12	14	16	16	13	16	14	7	8	9	0.262219	124/125	Serial (forward)
	8	14	13	15	14	16	14	8	7	16	0.551044	125/125	Serial (backward)
2^{nd}-order	46	21	6	16	6	12	7	7	3	1	0.000000 *	115/125 *	Frequency
	20	18	10	15	9	7	18	10	10	8	0.059743	124/125	BlockFrequency
	41	17	15	10	5	9	9	6	6	7	0.000000 *	118/125 *	CumulativeSums (m-2)
	39	16	14	9	11	6	10	2	12	6	0.000000 *	116/125 *	CumulativeSums (m-3)
	19	10	13	12	9	13	14	9	14	12	0.552185	123/125	Runs
	21	11	27	12	23	14	2	6	4	5	0.000000 *	123/125	LongestRun
	41	18	10	12	9	10	5	9	8	3	0.000000 *	120/125	ApproximateEntropy
	15	28	16	10	8	12	16	8	7	5	0.000123	123/125	Serial (forward)
	14	10	9	14	14	15	19	13	11	6	0.457002	124/125	Serial (backward)
3^{rd}-order	55	14	4	15	7	8	6	3	8	5	0.000000 *	111/125 *	Frequency
	22	19	20	11	13	8	9	6	9	8	0.003829	123/125	BlockFrequency
	47	21	9	9	5	5	10	3	8	8	0.000000 *	112/125 *	CumulativeSums (m-2)
	47	19	7	4	13	4	12	4	12	3	0.000000 *	111/125 *	CumulativeSums (m-3)
	15	10	13	13	14	13	15	8	13	11	0.889414	124/125	Runs
	21	16	33	12	17	10	3	7	4	2	0.000000 *	120/125	LongestRun
	42	17	13	7	10	11	8	5	8	4	0.000000 *	114/125 *	ApproximateEntropy
	26	25	12	9	7	7	12	12	7	8	0.000017 *	121/125	Serial (forward)
	16	12	13	11	9	16	11	17	11	9	0.575157	124/125	Serial (backward)
4^{th}-order	19	23	7	16	8	7	15	17	10	3	0.000407	121/125	Frequency
	19	14	9	15	11	16	15	7	9	10	0.344248	123/125	BlockFrequency
	20	20	13	11	13	7	10	10	9	12	0.143910	121/125	CumulativeSums (m-2)
	19	17	14	12	10	4	15	9	15	10	0.130323	120/125	CumulativeSums (m-3)
	10	13	8	14	14	12	16	9	18	11	0.437182	125/125	Runs
	15	12	23	18	17	8	8	6	10	8	0.007522	120/125	LongestRun
	19	11	15	10	13	11	12	11	12	11	0.708591	122/125	ApproximateEntropy
	12	15	6	10	11	13	19	14	14	11	0.571108	125/125	Serial (forward)
	9	7	9	15	14	17	10	13	16	15	0.551044	125/125	Serial (backward)
5^{th}-order	12	18	8	18	11	12	9	17	12	8	0.316158	124/125	Frequency
	12	16	13	9	15	19	10	9	14	8	0.493319	124/125	BlockFrequency
	14	14	14	11	12	10	17	7	12	14	0.665311	124/125	CumulativeSums (m-2)
	10	15	18	8	10	11	18	6	17	12	0.166594	123/125	CumulativeSums (m-3)
	9	10	12	23	6	11	10	12	19	13	0.036430	125/125	Runs
	11	18	18	15	20	11	9	8	10	5	0.036430	124/125	LongestRun
	12	12	13	9	9	18	14	11	14	13	0.729586	125/125	ApproximateEntropy
	7	14	8	6	14	20	10	16	14	16	0.096097	125/125	Serial (forward)
	9	9	11	10	12	10	17	13	14	20	0.457002	125/125	Serial (backward)
6^{th}-order	7	20	5	18	15	15	12	12	19	2	0.001080	125/125	Frequency
	7	18	5	15	14	10	17	11	10	18	0.086622	125/125	BlockFrequency
	9	18	11	13	12	9	12	8	21	12	0.231636	125/125	CumulativeSums (m-2)
	6	15	18	10	10	6	20	10	16	14	0.050764	124/125	CumulativeSums (m-3)
	5	9	9	13	16	16	13	14	14	16	0.422488	124/125	Runs
	10	14	18	12	21	11	7	10	11	11	0.211194	125/125	LongestRun
	4	12	7	16	15	9	14	15	17	16	0.130323	124/125	ApproximateEntropy
	11	7	9	15	15	11	9	19	17	12	0.329976	124/125	Serial (forward)
	15	15	15	14	7	6	13	9	16	15	0.405918	124/125	Serial (backward)

TABLE V

NIST TEST RESULTS WITH RESPECT TO THE RANDOM SEQUENCE GENERATED BY **1-OUT-OF-8 CODING**, WHERE $M = 32$ FOR BLOCK FREQUENCY TEST, $m = 1$ FOR APPROXIMATE ENTROPY TEST AND $m = 4$ FOR SERIAL TEST AND '*' MARKS A FAILURE.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	0	9	6	12	12	20	15	23	10	18	0.000072 *	125/125	Frequency
	0	0	0	1	0	1	4	4	15	100	0.000000 *	125/125	BlockFrequency
	0	0	8	5	5	9	13	16	9	60	0.000000 *	125/125	CumulativeSums (m-2)
	0	3	8	2	5	10	13	19	12	53	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	113	6	4	0	2	0	0	0	0	0	0.000000 *	62/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	1/125 *	Serial (forward)
	36	21	12	11	8	8	9	8	7	5	0.000000 *	117/125 *	Serial (backward)
1^{st}-order	0	9	6	12	12	20	15	23	10	18	0.000072 *	125/125	Frequency
	0	0	0	1	0	1	4	4	15	100	0.000000 *	125/125	BlockFrequency
	0	0	8	5	5	9	13	16	9	60	0.000000 *	125/125	CumulativeSums (m-2)
	0	3	8	2	5	10	13	19	12	53	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	113	6	4	0	2	0	0	0	0	0	0.000000 *	62/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	1/125 *	Serial (forward)
	36	21	12	11	8	8	9	8	7	5	0.000000 *	117/125 *	Serial (backward)
2^{nd}-order	1	6	10	9	14	17	15	22	17	14	0.001228	125/125	Frequency
	0	0	0	0	2	2	6	2	20	93	0.000000 *	125/125	BlockFrequency
	0	2	6	6	6	11	12	14	7	61	0.000000 *	125/125	CumulativeSums (m-2)
	1	1	4	11	7	7	13	10	14	57	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	111	8	3	2	0	0	1	0	0	0	0.000000 *	58/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	38	24	14	11	6	6	12	6	3	5	0.000000 *	110/125 *	Serial (backward)
3^{rd}-order	1	2	6	6	15	24	8	30	17	16	0.000000 *	125/125	Frequency
	0	0	0	0	2	1	3	7	11	101	0.000000 *	125/125	BlockFrequency
	1	0	3	4	3	7	7	23	16	61	0.000000 *	125/125	CumulativeSums (m-2)
	0	1	4	3	9	5	9	19	16	59	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	115	6	2	0	1	1	0	0	0	0	0.000000 *	48/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	38	27	10	12	11	9	6	6	4	2	0.000000 *	114/125 *	Serial (backward)
4^{th}-order	0	2	6	11	8	23	11	36	10	18	0.000000 *	125/125	Frequency
	0	0	0	0	1	2	1	6	14	101	0.000000 *	125/125	BlockFrequency
	0	2	1	3	5	7	9	17	12	69	0.000000 *	125/125	CumulativeSums (m-2)
	0	1	2	3	5	9	6	22	10	67	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	113	8	2	0	0	0	2	0	0	0	0.000000 *	54/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	1/125 *	Serial (forward)
	40	22	15	14	3	10	7	6	4	4	0.000000 *	110/125 *	Serial (backward)
5^{th}-order	2	3	6	6	10	16	15	22	17	28	0.000000 *	125/125	Frequency
	0	0	0	0	0	1	3	5	14	102	0.000000 *	125/125	BlockFrequency
	1	1	2	7	3	5	6	11	17	72	0.000000 *	125/125	CumulativeSums (m-2)
	1	1	5	2	4	5	6	21	11	69	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	110	9	2	1	2	1	0	0	0	0	0.000000 *	42/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	40	28	13	12	8	10	5	3	3	3	0.000000 *	114/125 *	Serial (backward)
6^{th}-order	2	3	4	7	13	19	13	26	18	20	0.000000 *	125/125	Frequency
	0	0	0	0	0	0	3	8	12	102	0.000000 *	125/125	BlockFrequency
	1	1	4	2	8	5	8	12	10	74	0.000000 *	125/125	CumulativeSums (m-2)
	1	3	1	1	2	10	6	20	14	67	0.000000 *	125/125	CumulativeSums (m-3)
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Runs
	115	6	2	0	1	1	0	0	0	0	0.000000 *	44/125 *	LongestRun
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	ApproximateEntropy
	125	0	0	0	0	0	0	0	0	0	0.000000 *	0/125 *	Serial (forward)
	41	23	20	13	7	6	8	0	2	5	0.000000 *	113/125 *	Serial (backward)

TABLE VI

NIST TEST RESULTS WITH RESPECT TO THE RANDOM SEQUENCE GENERATED BY THE **ORIGINAL NEIGHBOR CODING**, WHERE $M = 32$ FOR BLOCK FREQUENCY TEST, $m = 2$ FOR APPROXIMATE ENTROPY TEST AND $m = 5$ FOR SERIAL TEST AND '*' MARKS A FAILURE.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	30	21	12	10	8	9	5	7	10	13	0.000003 *	115/125 *	Frequency
	20	9	11	16	11	12	9	21	8	8	0.050764	120/125	BlockFrequency
	31	14	9	9	9	14	7	3	9	20	0.000000 *	119/125 *	CumulativeSums (m-2)
	32	15	11	6	10	14	5	3	11	18	0.000000 *	118/125 *	CumulativeSums (m-3)
	22	12	11	13	16	11	12	9	9	10	0.302788	120/125	Runs
	11	16	15	26	16	17	9	7	5	3	0.000062 *	124/125	LongestRun
	29	17	7	20	10	6	9	3	11	13	0.000001 *	119/125 *	ApproximateEntropy
	21	16	12	11	11	15	4	16	9	10	0.070160	116/125 *	Serial (forward)
	17	7	10	14	18	17	13	13	10	6	0.192277	123/125	Serial (backward)
1^{st}-order	21	17	13	15	11	7	13	7	9	12	0.130323	124/125	Frequency
	16	7	9	21	12	7	15	18	10	10	0.056599	122/125	BlockFrequency
	23	10	12	13	12	9	9	7	13	17	0.082208	124/125	CumulativeSums (m-2)
	24	9	12	12	9	15	16	6	12	10	0.034444	123/125	CumulativeSums (m-3)
	22	11	10	11	8	18	13	7	11	14	0.096097	122/125	Runs
	7	13	16	27	17	9	13	11	4	8	0.000274	124/125	LongestRun
	21	16	12	14	9	10	5	10	13	15	0.130323	122/125	ApproximateEntropy
	14	12	16	11	12	15	12	10	12	11	0.956806	122/125	Serial (forward)
	10	12	13	15	12	17	11	13	15	7	0.620686	123/125	Serial (backward)
2^{nd}-order	20	16	15	17	5	14	10	10	9	9	0.086622	121/125	Frequency
	17	6	15	12	8	14	16	12	13	12	0.262219	123/125	BlockFrequency
	21	16	11	13	12	7	12	5	11	17	0.073984	123/125	CumulativeSums (m-2)
	20	10	13	10	15	13	15	6	11	12	0.389809	120/125	CumulativeSums (m-3)
	20	14	11	13	13	8	7	13	14	12	0.493319	124/125	Runs
	7	11	20	23	22	10	9	12	6	5	0.000115	124/125	LongestRun
	20	15	15	12	11	5	10	13	11	13	0.316158	122/125	ApproximateEntropy
	15	15	11	8	16	11	15	14	10	10	0.643139	121/125	Serial (forward)
	16	7	7	11	15	12	13	17	14	13	0.262219	123/125	Serial (backward)
3^{rd}-order	18	17	12	10	17	8	14	11	9	9	0.389809	123/125	Frequency
	13	9	13	10	9	12	18	15	18	8	0.457002	122/125	BlockFrequency
	17	15	14	9	13	8	14	9	9	17	0.551044	124/125	CumulativeSums (m-2)
	16	11	12	13	8	16	19	5	15	10	0.192277	123/125	CumulativeSums (m-3)
	18	15	13	15	8	9	11	12	13	11	0.529142	124/125	Runs
	5	13	20	29	16	16	8	8	2	8	0.000000 *	123/125	LongestRun
	17	13	13	11	12	11	14	9	14	11	0.889414	125/125	ApproximateEntropy
	15	11	8	13	16	5	16	14	13	14	0.493319	123/125	Serial (forward)
	16	10	8	10	10	10	19	16	15	11	0.439517	125/125	Serial (backward)
4^{th}-order	18	17	15	17	10	10	10	13	5	10	0.192277	124/125	Frequency
	15	8	14	13	12	12	11	11	17	12	0.807956	124/125	BlockFrequency
	20	17	12	8	11	14	5	13	8	17	0.070160	124/125	CumulativeSums (m-2)
	16	18	7	16	15	9	13	5	10	16	0.117876	124/125	CumulativeSums (m-3)
	18	14	12	9	16	13	9	13	8	13	0.414457	125/125	Runs
	7	8	19	30	21	7	13	5	5	10	0.000000 *	123/125	LongestRun
	15	17	17	7	11	5	17	11	10	15	0.143910	122/125	ApproximateEntropy
	12	12	15	8	11	16	11	14	19	7	0.457002	123/125	Serial (forward)
	13	6	12	13	13	15	13	14	13	13	0.825875	124/125	Serial (backward)
5^{th}-order	20	17	14	8	15	11	9	6	9	16	0.096097	125/125	Frequency
	10	9	12	12	13	13	16	15	17	8	0.552185	124/125	BlockFrequency
	21	13	10	14	10	9	13	3	15	17	0.043046	124/125	CumulativeSums (m-2)
	16	18	8	12	12	13	7	5	21	13	0.036430	125/125	CumulativeSums (m-3)
	16	10	13	10	7	13	12	19	7	18	0.192277	123/125	Runs
	4	13	17	30	20	11	10	5	3	12	0.000000 *	124/125	LongestRun
	15	14	14	12	9	10	18	12	10	11	0.665311	122/125	ApproximateEntropy
	13	15	13	12	14	13	19	7	12	7	0.512137	124/125	Serial (forward)
	11	10	20	10	13	16	12	10	15	8	0.474938	125/125	Serial (backward)
6^{th}-order	21	12	10	13	10	12	17	7	9	14	0.231636	122/125	Frequency
	16	10	11	9	9	11	17	17	13	12	0.437182	123/125	BlockFrequency
	21	12	11	11	7	15	11	6	13	18	0.091249	123/125	CumulativeSums (m-2)
	18	12	7	15	12	13	9	10	9	20	0.211194	123/125	CumulativeSums (m-3)
	14	10	15	15	15	15	11	6	13	11	0.529142	122/125	Runs
	7	7	22	25	20	10	7	13	6	8	0.000017 *	122/125	LongestRun
	17	12	11	13	11	14	17	8	12	10	0.598008	124/125	ApproximateEntropy
	19	14	6	11	9	14	16	16	9	11	0.277369	123/125	Serial (forward)
	15	10	10	13	10	13	11	17	14	12	0.843024	121/125	Serial (backward)

TABLE VII

NIST TEST RESULTS WITH RESPECT TO THE RANDOM SEQUENCE GENERATED BY THE **DECOUPLED NEIGHBOR CODING**, WHERE $M = 32$ FOR BLOCK FREQUENCY TEST, $m = 2$ FOR APPROXIMATE ENTROPY TEST AND $m = 5$ FOR SERIAL TEST AND '*' MARKS A FAILURE.