THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2011-13

Kendall Syndrome Coding (KSC) for Group-Based Ring-Oscillator Physical Unclonable Functions

Chi-En Yin and Gang Qu

Kendall Syndrome Coding (KSC) for Group-Based Ring-Oscillator Physical Unclonable Functions

Chi-En Yin and Gang Qu
Department of Electrical and Computer Engineering & Institute for Systems Research
University of Maryland, College Park, USA
{chienyin,gangqu}@umd.edu

***Abstract*—Rank permutation has been proposed to maximize the secrecy extraction power of ring oscillator (RO) physical unclonable functions (PUFs) [1]. To encode the frequency rank of a group of ROs, the authors suggested to encode compactly. One problem is that the error weight distribution does not correlate well with the underlying error probability distribution such that many errors occurring frequently are encoded in greater (instead of smaller) Hamming distance than those less likely to happen. Consequently, the previous scheme would suffer greater min-entropy loss [2] when worked with a error correcting code (ECC). Instead, we propose a new encoding scheme such that the Hamming distance between two encoded codewords is the same as the Kendall tau distance [3] between the two corresponding rank permutations. We also improve manufacturing feasibility as well as the randomness of the secrecy such that our generated secret demonstrates the statistics of the uniform distribution [4]. Overall, the proposed PUF is about 9% more efficient than its predecessor and 50% or more efficient than an index-based (IBS) approach [5] in most error correcting scenarios we considered under an i.i.d. uniform output assumption.**

***Index Terms*—ring oscillator (RO), physically unclonable functions (PUFs), Compact Syndrome Coding (CSC), Kendall Syndrome Coding (KSC), error correcting code (ECC), Hamming distance d_h, Kendall tau distance d_τ, rank permutation codes**

I. Introduction

A Physical Unclonable Function (PUF) is a physical structure whose functional characteristic is hard to predict before fabrication but once fabricated the characteristic of each device is rather stable and unique. One key application of PUFs is cryptographic secret generation and storage that may not demand a large number of challenge-response pairs (CRPs) but requires a high level of randomness and stability. Nevertheless, these two properties are often at odds, making their co-existence without raising hardware budget a great challenge.

A. Group-based Coding

LISA [1], a heuristic group-based coding algorithm, has been proposed to extract the maximum comparison-based entropy $\log_2 M!$ out of M ROs by searching for longest increasing subsequences (LIS) to form independent groups containing the largest set of ROs while maintaining certain stability within the group. However, there are four issues that may reduce its usability.

- **Manufacturing Complexity** In order to ensure stability it demands frequency measurements at two temperature boundaries, e.g., 0°C and 100°C, when the secret is enrolled. This complicates the fabrication process significantly.
- **Error Tolerance** LISA is effective to address temperature variability that is linear to RO frequency but not so when dealing with non-linear variability such as supply voltage fluctuation.
- **Spatial Correlation** The underlying systematic trend may render each extracted bit no longer independent and identically distributed (i.i.d.) [6], which in turn weakens the security of the secret bitstring.
- **Coding Inefficiency** The previous work encoded the frequency rank permutation of M ROs in binary of integer $0 \dots M! - 1$, here we refer to as Compact Syndrome Coding (CSC); however, CSC does not work well with error correcting codes (ECC). To illustrate, let us consider a list of four ROs RO_A, RO_B, RO_C and RO_D; the CSC codebook in lexicographic order is listed in Table I. Suppose a rank permutation $\{RO_A < RO_D < RO_C < RO_B\}$ is first enrolled with its codeword 00101_2; later on, we would like to regenerate the codeword given certain errors. Let us say a flipover occurs between RO_A and RO_D, yielding us a new frequency measurement $\{RO_D < RO_A < RO_C < RO_B\}$ or 10011_2 in its encoded form. Since the Hamming distance d_h between the two codewords 00101_2 and 10011_2 is three, the error can be corrected by BCH($n = 15$, $k = 5$, $t = 3$) code using the Code-Offset technique in [2], assuming there is no more error in the subsequent 10 bit of the 15-bit code block. To estimate the effective min-entropy, which is defined later, let us assume the 10 bits are derived from other two independent lists of 4 ROs. If

00000_2	$\{ABCD\}$	01000_2	$\{BCAD\}$	10000_2	$\{CDAB\}$
00001_2	$\{ABDC\}$	01001_2	$\{BCDA\}$	10001_2	$\{CDBA\}$
00010_2	$\{ACBD\}$	01010_2	$\{BDAC\}$	10010_2	$\{DABC\}$
00011_2	$\{ACDB\}$	01011_2	$\{BDCA\}$	10011_2	$\{DACB\}$
00100_2	$\{ADBC\}$	01100_2	$\{CABD\}$	10100_2	$\{DBAC\}$
00101_2	$\{ADCB\}$	01101_2	$\{CADB\}$	10101_2	$\{DBCA\}$
00110_2	$\{BACD\}$	01110_2	$\{CBAD\}$	10110_2	$\{DCAB\}$
00111_2	$\{BADC\}$	01111_2	$\{CBDA\}$	10111_2	$\{DCBA\}$

TABLE I
THE CODEBOOK OF THE RANK PERMUTATIONS OF 4 ROS USING COMPACT SYNDROME CODING (CSC), WHERE $\{ABCD\}$ IS A SHORTHAND OF THE FREQUENCY RELATION $RO_A < RO_B < RO_C < RO_D$

each rank permutation is equally likely with probability $\frac{1}{4!}$, then the raw entropy of the block is $3\log_2 4!$ or 13.754 bits. The min-entropy, however, is merely 3.7543 bits because we have to deduct 10 bits of entropy loss due to public syndrome disclosure[1]. In fact, the loss can be reduced by better correlating the error weight distribution in Hamming space with the probability mass function (p.m.f.) of erroneous flipovers. In other words, if an error event happens in higher probability, it should be encoded closer in Hamming space. To see how this may not be true in CSC, consider three flipovers happening at the same time: the first between RO_A and RO_B, the second between RO_A and RO_C and the third between RO_A and RO_D, with presumably fairly low probability. But instead, the Hamming distance between the new codeword 10111_2 of $\{DCBA\}$ and the enrolled codeword 00101_2 of $\{ADCB\}$ is only two, even closer than the previous error pattern presumably in higher probability.

B. Contributions

This work offers a new group-based RO PUF with improved entropy efficiency, security and fabrication simplicity. Specifically, 1) we propose a new syndrome coding based on Kendall tau distance for better stability in the encoded Hamming space; 2) we incorporate two new processes for entropy distillation and entropy packing to strengthen security; 3) we redesign the grouping algorithm to simplify the manufacturing process. The improvements make the new design about 9% more efficient than its predecessor and 50% or more efficient than the index-based (IBS) approach [5] in most of our experimental scenarios. The NIST hypothesis testing affirms that our PUF output is indistinguishable from those drawn from an ideal uniform distribution. We emphasize that the results are *not* derived with help from universal hash function (UHF) nor linear feedback shift register (LFSR) typically employed for PUF secrecy amplification. When external seeding is spared, our PUF can generate its secrecy in a fully autonomous fashion.

C. Paper Organization

The rest of paper is organized as follows: Section II introduces Kendall Syndrome Coding (KSC); Section III describes the architecture of the proposed RO PUF; Section IV presents experimental results; lastly, Section V concludes.

II. Kendall Syndrome Coding (KSC)

Kendall's correlation statistic [3] can be used to address the aforementioned issue of coding inefficiency. To begin with, rank permutation of n elements is defined as permutation of integers (ranks) $1 \dots n$; Kendall tau distance $d_\tau(\sigma, \pi)$ is defined as the minimum number of transpositions of adjacent ranks required to change from one rank permutation σ into another π of the same size [7]. For instance, consider two rank permutations of three elements $\sigma = \{\sigma(1) = 1, \sigma(2) = 3, \sigma(3) = 2\}$ and $\pi = \{\pi(1) = 2, \pi(2) = 3, \pi(3) = 1\}$. The Kendall distance $d_\tau(\sigma, \pi)$ is one because σ would be equal to π after transposing of rank 1 and rank 2 of the first element and the third element[2]. d_τ can also defined as the number of pairwise disagreements between two rank permutations [8], namely,

$$d_\tau(\sigma, \pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} s(i, j) \tag{1}$$

where

$$s(i,j) = \begin{cases} 1 & \text{if } (\sigma(i) < \sigma(j) \cap \pi(i) > \pi(j)) \\ & \cup\ (\sigma(i) > \sigma(j) \cap \pi(i) < \pi(j)) \\ 0 & \text{otherwise.} \end{cases}$$

For a group-based RO PUF, d_τ is equal to the number of flipovers in a group during secret regeneration and presumably reversely proportional to the error probability. Now we want to encode the frequency rank permutation of a list of ROs from the Kendall space into the Hamming space such that the Kendall distance d_τ between any two rank permutations is equivalent to the Hamming distance d_h of the corresponding codewords. Indeed, such a class of codes exists and the conversion is efficient [9]. Given a group of ROs g, $\{RO_1 \dots RO_n\}$ in certain physical order, its rank permutation can be encoded pairwisely into a bitstring s_g of length $\frac{n(n-1)}{2}$,

$$s_g = (s_g(1,2) \dots s_g(1,n), s_g(2,3) \dots s_g(n-1,n)) \tag{2}$$

where $\forall s_g(i,j), i < j, 1 \le i, j \le n$,

$$s_g(i,j) = \begin{cases} 0 & \text{if } RO_i < RO_j \\ 1 & \text{otherwise.} \end{cases}$$

For instance, consider a group g composed of 3 ROs with frequency readings $\{RO_A = 38, RO_B = 97, RO_C = 54\}$ in the same rank permutation as the $\sigma = \{1, 3, 2\}$ defined earlier; from Eqn. (2), we can KSC encode g into $s_g = (0, 0, 1)$. Let us say later certain environmental variations cause a re-measurement of the frequencies to $g' = \{RO_A = 78, RO_B = 103, RO_C = 60\}$, which has the same rank permutation as the $\pi = \{2, 3, 1\}$ defined above. In turn, Eqn. (2) yields a KSC encoded bitstring $s_{g'} = (0, 1, 1)$. As we see, $d_h(s_g, s_{g'}) = d_\tau(\sigma, \pi) = 1$, where the pairwise disorder takes place between elements (1, 3) for $g(1) < g(3)$ but $g'(1) > g'(3)$. Conversely, [9] also shows how to decode s_g into $\sigma = (\sigma(1) \dots \sigma(n))$, where

$$\sigma(i) = 1 + \sum_{j=1}^{i-1} (1 - s(i,j)) + \sum_{j=i+1}^{n} s(i,j). \tag{3}$$

III. The Proposed Group-Based RO PUF

The architecture of the proposed PUF is depicted in Figure 1. Our public helper data comprises three parts (i) distiller coefficients, (ii) group information and (iii) syndrome bits; all of them are determined and stored publicly in non-volatile memory such as EEPROM or NAND/NOR flash when the secret is first enrolled. The seven-step secrecy extraction procedure is explained in the following.

[1] More generally, $(n - k)$-bit loss for linear codes [2].

[2] In the previous section, the elements are listed in the order of ranks, different from here the ranks are listed in the order of elements.

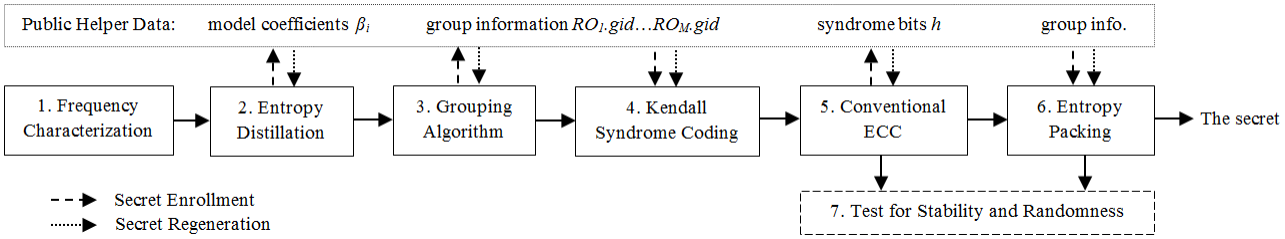

Fig. 1. The architecture of the proposed group-based RO PUF

1) Frequency Characterization: A RO PUF typically consists of a RO array as well as counters and multiplexers to help acquire frequency readings of the RO array. Given M ROs, the frequency readings are denoted as $RO_1 \ldots RO_M$ in certain physical order, say, scanning sequentially by rows when ROs are placed as a 2-D array. One may take an average over multiple measurements for each RO as the output of the step.

2) Randomness Distillation: The output from Step 1 contains both random and systematic variations. Since the systematic component may render the output lack of randomness, we apply polynomial regression to remove it. Each PUF calculates its own model coefficients and stores them as public helper data. As an example, let us model the systematic trend of a m-by-n 2-D RO array by means of 1^{st}-order polynomial $sys_{x,y} = \beta_1 x + \beta_2 y + \beta_3$, where β_i's are the model coefficients to be solved and $(1,1) \leq (x,y) \leq (n,m)$. The frequency measurement can then be written as $RO_{x,y} = sys_{x,y} + \epsilon_{x,y}$, where $sys_{x,y}$ denotes the systematic variability and $\epsilon_{x,y}$ denotes the random variation at location (x,y). Putting $m \times n$ model equations in matrix form, we have

$$\boldsymbol{z} = \boldsymbol{\Omega\beta} + \boldsymbol{\epsilon}, \tag{4}$$

where

$$\boldsymbol{z} = \begin{pmatrix} RO_{1,1} \\ \vdots \\ RO_{n,1} \\ \vdots \\ RO_{1,m} \\ \vdots \\ RO_{n,m} \end{pmatrix}, \boldsymbol{\Omega\beta} = \begin{pmatrix} 1 & 1 & 1 \\ \vdots & \vdots & \vdots \\ n & 1 & 1 \\ \vdots & \vdots & \vdots \\ 1 & m & 1 \\ \vdots & \vdots & \vdots \\ n & m & 1 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix}, \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_{1,1} \\ \vdots \\ \epsilon_{n,1} \\ \vdots \\ \epsilon_{1,m} \\ \vdots \\ \epsilon_{n,m} \end{pmatrix}.$$

To solve the model coefficients $\boldsymbol{\beta}$, one can use the least squares method that takes the first derivative on the sum of the squares of the residual terms $\boldsymbol{\epsilon}$ and set the derivative to zero, or formally,

$$\frac{\partial \boldsymbol{\epsilon}^2}{\partial \boldsymbol{\beta}} = \frac{\partial (\boldsymbol{z} - \boldsymbol{\Omega\beta})^{\boldsymbol{T}} (\boldsymbol{z} - \boldsymbol{\Omega\beta})}{\partial \boldsymbol{\beta}} \tag{5}$$

$$= \frac{\partial (\boldsymbol{z^T z} - \boldsymbol{z^T \Omega\beta} - \boldsymbol{\beta^T \Omega^T z} + \boldsymbol{\beta^T \Omega^T \Omega\beta})}{\partial \boldsymbol{\beta}} \tag{6}$$

$$= -2\boldsymbol{\Omega^T z} + 2\boldsymbol{\Omega^T \Omega\beta} = \boldsymbol{0} \tag{7}$$

$$\Longrightarrow \boldsymbol{\beta} = \boldsymbol{(\Omega^T \Omega)^{-1} \Omega^T z}, \tag{8}$$

where $\boldsymbol{(\Omega^T \Omega)^{-1} \Omega^T}$ is a constant solver and $\boldsymbol{z}$ is the frequency characterization of the $m \times n$ ROs at enrollment. Once $\boldsymbol{\beta}$ is determined, we can easily calculate the output of the step, that is, the distilled random variation $\boldsymbol{\epsilon'}$, which is equal to $\boldsymbol{z' - \Omega\beta}$ for any frequency re-characterization $\boldsymbol{z'}$.

3) Grouping Algorithm: The goal of the step is to form independent groups while meeting certain stability criterion. The key difference from its predecessor [1] is that only *one* environmental condition is needed to take the measurements in the enrollment phase. This resolves the manufacturing problem mentioned in the very beginning. Due to the change, the maximization problem in [1] is rephrased as: given M ROs whose frequency output are i.i.d., we want to find a partition $G = \{g_1 \ldots g_{|G|}\}$ that maximizes the total group entropy $\sum_{i=1}^{|G|} \log_2 |g_i|!$ while ensuring that no RO pair in the same group g_i have their frequency difference less than a stability threshold f_{th} at enrollment time, that is,

maximize $\sum_{i=1}^{|G|} \log_2 |g_i|!$ ***subject to***

a) $g_i \cap g_j = \emptyset$, where $1 \leq i, j \leq |G|, i \neq j$
b) $g_1 \cup g_2 \cup \ldots \cup g_{|G|} = RO_1, \ldots RO_M$
c) $\forall RO_i, RO_j \in g_k$, $|RO_i - RO_j| \geq f_{th}$, where $1 \leq i, j \leq |g_k|, i \neq j, 1 \leq k \leq |G|$.

Constraint a) ensures that no RO is used twice to maintain the i.i.d. PUF output assumption; b) leaves no RO unexplored[3]; c) parametrizes PUF stability with f_{th}, whose value can be determined empirically. To solve the new maximization problem, LISA can be simplified as LISA-lite as below. LISA-lite completes in $O(M^2)$ and the worst case occurs, for instance, when all ROs have the same frequency readings such that only one RO can be removed from the linked list for each run of the for loop. Lastly, each RO is assigned with a group ID in solution G and all $RO_i.gid$'s are stored in their physical order for future reference.

4) Kendall Syndrome Coding: Each group g_i in solution G is KSC encoded by Eqn. (2). The resulting bitstrings $s_{g_1} \ldots s_{g_{|G|}}$ are concatenated as the output of the step.

5) Conventional ECC: A linear block code (n, k, t) can be used to correct runtime errors, where n denotes block size, k the number of information bits, $n-k$ the number of parity bits and t maximum errors within the block that are correctable by the code. The Code-Offset technique [2] is assumed to bound the min-entropy loss due to public disclosure of syndrome bits.

[3]The constraint is redundant in the setting of a maximization problem but kept for clarity

Algorithm 1 LISA-lite

Input: (i) M ROs, $RO_1 \dots RO_M$, in their physical order with $RO_i.phy$ denoting i^{th} RO's physical position, $RO_i.frq$ its frequency reading at enrollment, and $RO_i.gid$ its group ID (with initial value -1); (ii) reliability threshold f_{th}

Output: (i) $RO_1 \dots RO_M$ with all $RO_i.gid \neq -1$

1: sort $RO_1 \dots RO_M$ in increasing order of $RO_i.frq$'s and keep the sorted objects $RO'_1 \dots RO'_M$ on a linked list L
2: $gid \leftarrow 1$
3: $frq_{pre} \leftarrow -\infty$ // previous frequency
4: **while** $frq_{pre} \neq -\infty$ **do**
5: $frq_{pre} \leftarrow -\infty$
6: **for** $i \leftarrow 1$ to $|L|$ **do**
7: **if** $(RO'_i.frq - frq_{pre}) \geq f_{th}$ **then**
8: $RO'_i.gid \leftarrow gid$
9: $frq_{pre} \leftarrow RO'_i.frq$
10: remove RO'_i from L
11: $i = i - 1$
12: **end if**
13: **end for**
14: $gid = gid + 1$
15: **end while**
16: **return** $G = \{RO_1 \dots RO_M\}$

At enrollment, the output from the previous step is divided into blocks of n-bit secret w. The first k bits of w is encoded with certain ECC to produce $n-k$ parity bits p. The parity bits p then exclusive-or with the last $n-k$ bits of w to produce the $n-k$ syndrome bits h, which are then saved as public helper data to assist secrecy recovery. To recover the enrolled secret block w given new input w' from Step 4, we retrieve the saved syndrome bits h and exclusive-or with the last $n-k$ bits of w' to produce $n-k$ parity bits p'. The first k-bit of w' is then appended with p' to form a n-bit block to decode. As long as $d_h(w, w') \leq t$, ECC decoder can correct all errors in w' and output the w enrolled at first. All restored secret blocks w's form the output of the step and input of Step 6–7.

6) Entropy Packing: Although KSC is designed to help reduce the complexity of ECC, it does not encode entropy efficiently. Many bitstrings are left unused; taking three ROs for example, (0, 1, 0) would never occur due to the contradiction ($RO_A < RO_B, RO_A \geq RO_C, RO_B < RO_C$). Therefore, we have to re-encode each group g_i compactly with CSC with help from the stored group information $RO_i.gid$'s. Since error correction has been done, the coding inefficiency issue mentioned in the very beginning is no longer a concern. Before encoding in CSC, we have to KSC decode each group s_{g_i} into its rank permutation via Eqn. 3. To encode a group g in the most compact form c_g, we can calculate the inversion vector (line 2–8,10) and interpret it in factorial number system (line 9) as below [10]. Note that the algorithm we are showing does not produce a codebook in lexicographic order like Table I. CSC decoding can also be done efficiently but is beyond the scope of the work. Lastly, in order to construct a secret in uniform distribution, we have to close the unused gap between $|g|!$ and $2^{\lceil \log_2 |g|! \rceil} - 1$ for each output c_g as we put together all c_{g_i}'s to form the final PUF secret. This can be done by simple addition, subtraction and shift operations; the pseudo code is omitted for brevity.

Algorithm 2 Compact Syndrome Coding (CSC) Encoding

Input: a group g containing ordered ROs $RO_1 \dots RO_{|g|}$

Output: a CSC encoded integer c_g in $\lceil \log_2 |g|! \rceil$ bits

1: $c_g = 0$
2: **for** $i \leftarrow |g|$ to $|2|$ **do**
3: $inv = 0$ // number of inversions
4: **for** $j \leftarrow 1$ to $i - 1$ **do**
5: **if** $RO_i < RO_j$ **then**
6: $inv = inv + 1$
7: **end if**
8: **end for**
9: $c_g = (c_g + inv) \times (i - 1)$
10: **end for**
11: **return** c_g

7) Test for Randomness and Stability: This verification process not only helps us ensure security and reliability of the final secret but also helps us choose the parameter f_{th} and the exact regression model. This leads our discussion to the next section.

IV. Experimental Results

This section extends Step 7 in greater detail. The prototype of the proposed PUF has been realized as an embedded system on Xilinx Virtex-5 FPGAs via Xilinx ISE and Xilinx Platform Studio (XPS) EDK/SDK 13.2 and 9.2 tool chains[4]. The PUF is implemented as an IP core connecting to the MicroBlaze™ soft processor core through Processor Local Bus (PLB). The bus exchanges control signals and RO frequency characterization between the processor and the IP via user defined soft registers. The frequency characterization is in turn passed through to a RS232 serial port logger on PC for the following analyses. Timing critical logic such as RO is instantiated as a hard macro in ISE. Polynomial regression employed in Step 2 can be solved quickly as the Floating Point Unit (FPU) of MicroBlaze™ is enabled. In addition to our own dataset, the public dataset [11] is also used to confirm the results.

A. Test for Randomness

The first goal is to test whether our PUF is secure. To this end, the output of Step 6 is subject to the NIST statistical test for randomness [4]. An 'ideal' random sequence is regarded as the outcome of consecutive flips of a fair coin. In other words, the random variable assigned to each toss is identical and independent distributed (i.i.d.) and uniformly distributed between 0 and 1 with equal probability 1/2. The null hypothesis of the test is that the random sequence under test is 'ideal'

[4]v13.2 for ML506 and ML510 development boards and v9.2 for ML501 boards.

unless the test statistic indicates a clear deviation. NIST test results are interpreted in two ways: a) the proportion of total bitstrings that passes a test shall be above a minimum value; b) the *P-values* of all bitstrings shall be uniformly distributed such that the *P-value* of the *P-values* is equal or greater than a minimum value; default settings were used in the test suite. Eleven out of fifteen tests in the suite are applicable to our output length; for each of the two datasets, we use the first half to select the order of the polynomial model and the second half to validate the randomness of the output.

1) Virginia Tech Dataset: This dataset comprises frequency characterization of 125 Xilinx Spartan-3 (90-nm) FPGAs [11]. For each FPGA, 512 ROs are implemented in 32 rows by 16 columns. Although there are 100 frequency measurements available for each RO in one operating condition, only the first measurement at 1.2V 25°C is used as the output of Step 1. The outputs of Step 6 from chip No.1 to No.125 are concatenated altogether to construct a long random sequence subject to test. Test parameters are set as follows: bitstring length 400 (except 3× for FFT Test in order to meet with the minimum length requirement), block length for Frequency Test 32, 2 for Approximate Entropy Test and 5 for Serial Test, all following NIST recommendations. The test results from the first half of the dataset suggest that we can select 1^{st}-order polynomial to remove the systematic component with least computation. Indeed, the results from the second half affirm the selection and more importantly the i.i.d. uniform assumption we have made on our PUF output; see Appendix Table V and Table III for detailed reports.

2) In-House Dataset: The second dataset is collected from 9 Xilinx Virtex-5 (65-nm) FPGAs in our own lab. The 9 FPGAs come with three different types of development boards: 3 ML501, 3 ML506 and 3 ML510. Because of different sizes, they are placed with different number of RO arrays: 3 for ML501, 6 for ML506 and 12 for ML510, 63 in total. Each array contains 32 × 16 ROs just as in the previous dataset and is regarded as an independent PUF for the test. Although multiple measurements are available for each array, only the first measurement at 20°C is used as the output of Step 1 to produce the final secret. All test parameters are the same as mentioned previously. Similar results are derived from this dataset: The first half suggests the choice of the first order model since it passes all tests with the lowest computing and storage cost. The i.i.d. uniform assumption along with the selected model is affirmed by the second half. Respective results are listed in Appendix Table VI and Table IV.

B. Test for Stability

The second goal is to determine the optimal value of f_{th} given a class of ECC in Step 5; moreover, in the criterion that all errors have to be corrected, we want to estimate the effective min-entropy we can extract given a fixed number of ROs. Three classes of BCH codes are considered, namely, BCH($n = 31$, k, t), BCH($n = 63$, k, t) and BCH($n = 127$, k, t), where n denotes code block size, k the number of information bits per block and t the maximum correctable errors within a block. As we know from [2], the larger the k, the smaller the min-entropy loss $n - k$ per block; both depend on t. For each class of code, we first pick a f_{th} and find out the largest t among *all* ECC blocks under test. As long as t is small enough such that all errors can be corrected by the given class of code, we have at least one solution k for the picked f_{th}, among which the largest k is used to calculate the effective min-entropy of PUF schemes. For the proposed PUF, the effective min-entropy H_∞^{KSC} is estimated as

$$\sum_{i=1}^{|G|} \log_2 |g_i|! - \lceil \frac{\sum_{i=1}^{|G|} |s_{g_i}|}{n} \rceil (n-k) \text{ if } \geq 0, \text{ else } 0, \quad (9)$$

where the first term represents the total entropy extracted from the grouping algorithm in Step 3 and the second term discounts the total min-entropy loss due to public disclosure of syndrome bits for ECC in Step 5. To compare, when CSC rather than KSC is used in Step 4 (as proposed in [1]), replacing the s_{g_i}'s in Eqn. (9) with the c_{g_i}'s defined in Step 6 forms the estimate of H_∞^{CSC}. In case there is no solution k for a code class to correct all errors with a given f_{th}, the effective min-entropy is then set to 0 as the case when the result of Eqn. (9) turns negative. For both datasets, f_{th} is swept from 0.1 to 3 standard deviation of the 512 random variations distilled in Step 2.

1) Virginia Tech Dataset: This dataset consists of frequency characterization of 5 Spartan-3 FPGAs provisioned with ±10% and ±20% core supply voltage fluctuation and separately temperature variation from 25°C to 65°C [11]. We rule out the cases with ±20% voltage swings for they would drive the results too conservative. Rather, our estimate is based on the criterion that *all* errors must be corrected in rest of the 7 cases: Case 1) the nominal condition at 1.2V 25°C; Case 2–3) provisioned in 25°C with core supply voltage at 1.08V and 1.32V; Case 4–7) provisioned at 1.2V 35°C, 45°C, 55°C and 65°C. The secret is enrolled in Case 1 and regenerated in Case 2–7). Also, we scan through 0^{th} to 6^{th} order model in Step 2 and use the average to report.

Figure 2 draws two test results when BCH(n=31, k, t) is considered: (i) the maximum number of errors among all ECC blocks, see vertical bars that correspond to the secondary y-axis; (ii) the averaged H_∞^{CSC} and H_∞^{KSC} given (i), see lines and the primary y-axis. As we see, both H_∞^{CSC} and H_∞^{KSC} drop to zero when f_{th} approaches zero, meaning that errors are so overwhelming that either there is no solution to make all blocks error-free or the entropy loss surpasses the group entropy in Eqn. 9. On the other hand, as f_{th} approaches 3 there is no error to be corrected such that H_∞^{CSC} and H_∞^{KSC} converge. When f_{th} is in between, H_∞^{CSC} and H_∞^{KSC} increase if the reduction in min-entropy loss is greater than the reduction in group entropy extraction; otherwise, they decrease. The tipping point, 1.9 for H_∞^{KSC}, suggests the optimal value we can choose for f_{th}. We can see similar results for BCH(n=63, k, t) and BCH(n=127, k, t) in Appendix Figure 3 and 4. On average H_∞^{KSC} outperforms H_∞^{CSC} by 8% in those optimal cases.

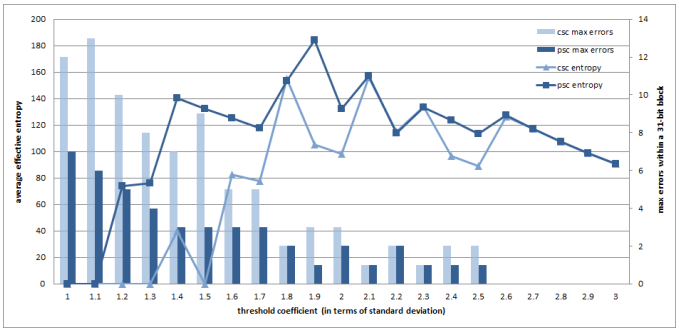

Fig. 2. The estimated H_∞^{CSC} and H_∞^{KSC} derived with BCH(31, k, t) based on the Virginia Tech dataset

2) In-House Dataset: The second dataset is derived from the 9 FPGAs mentioned earlier. Frequency measurements are provisioned at chip temperature 20°C, 50°C and 100°C respectively with no intended core supply voltage variation. Temperatures are manually controlled by monitoring the on-chip system sensor. Ten measurements are taken for each of the three scenarios; the first measurement at 20°C is used to generate the secret and then we try to recover it from the rest of 29 measurements with help from ECC. As in the previous dataset, polynomial orders 0^{th} to 6^{th} are all tested out. The test results are similar to those derived from the previous dataset only the tipping point shift lower to f_{th} around 0.4 to 0.9 standard deviation due to less errors are introduced with a stable voltage supply. Consequently, the estimates of H_∞^{CSC} and H_∞^{KSC} in those optimal cases almost double than in the previous dataset to 353 and 385 bits respectively, which translates to a 9% gain. Complete charts are reported in Appendix Figure 5, 6, 7.

C. In Comparison with IBS-Based RO PUF

To compare with the IBS-based scheme [5] in the context of the i.i.d. uniform PUF output assumption, we form IBS blocks each out of k consecutive ROs[5] and ensure no blocks share a common RO. Each IBS block generates one bit secret through comparing the ROs that yield the largest frequency difference at enrollment time; the selected pair is recalled by keeping a $\log_2 \binom{k}{2}$-bit index as public helper data. The k we consider ranges from 2 to 6. Applying a similar methodology we estimate the effective min-entropy H_∞^{IBS} using the the same datasets and BCH codes. For the Virginia Tech dataset no error takes place when $k = 6$, whereas no error is observed when $k \geq 4$ for the in-house dataset. The H_∞^{IBS} vs. k relation is drawn in Appendix Figure 8 and 9. Table II summarizes the best case H_∞^{IBS} and H_∞^{KSC} when different error correcting capability is assumed. As we see, in most cases the proposed group-based RO PUF is 50% or more efficient than the conjectured IBS-based RO PUF in terms of effective min-entropy given the same number of ROs. The only case KSC underperforms is due to one single error that cannot be cured until f_{th} reaches 2.7; the study of the root cause and preventing mechanisms is among our future work.

[5]Since there is no distillation process assumed, a block has to be formed out of ROs physically as close as possible to reduce systematic correlation [6].

	Virginia Tech			In-house		
	H_∞^{IBS}	H_∞^{KSC}	Gain	H_∞^{IBS}	H_∞^{KSC}	Gain
No ECC	85	128	50%	128	102	-20%
BCH(31)	96	184	92%	128	280	118%
BCH(63)	104	189	82%	134	366	173%
BCH(127)	121	197	63%	172	509	196%
Average	107	190	72%	140	314	123%

TABLE II
THE ESTIMATE OF H_∞^{IBS} AND H_∞^{KSC} IN VARIOUS ECC SCENARIOS

V. CONCLUSION

This work presents a new group-based RO PUF. First of all, we introduce Kendall Syndrome Coding (KSC) to address the coding inefficiency issue. Our experiment indicates a 9% gain in terms of effective min-entropy. Second, we incorporate entropy distillation and entropy packing processes to achieve stronger secrecy. NIST test results derived from two independent datasets both affirm the i.i.d. uniform assumption we have made on our PUF output. Third, we redesign the grouping algorithm LISA-lite to simplify the manufacturing process. Besides, we construct a IBS-based PUF based on the i.i.d. assumption and show that our design can be 50% or more efficient in various error correcting scenarios. The proposed PUF can be quickly realized as an embedded system on FPGA with modern tool chain.

REFERENCES

[1] 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.
[2] L. R. Yevgeniy Dodis and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *Proceedings of IACR Eurocrypt 2004 International Conference, LNCS Vol. 3027, pp. 523–540, Springer*, May 2004.
[3] M. Kendall, "Rank correlation methods," *London: Griffin*, 1958.
[4] J. N. M. S. E. B. S. L. M. L. M. V. D. B. A. H. J. D. S. V. Andrew Rukhin, Juan Soto 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.
[5] 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.
[6] 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.
[7] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *Proceedings of 2010 IEEE International Symposium on Information Theory (ISIT)*, June 2010.
[8] H. Chadwick and L. Kurz, "Rank permutation group codes based on kendall's correlation statistic," *IEEE Transactions on Information Theory, Vol. 15, No. 2, pp. 306–315*, 1969.
[9] H. Chadwick and I. Reed, "The equivalence of rank permutation codes to a new class of binary codes," *IEEE Transactions on Information Theory, Vol. 16, No. 5, pp. 640–641*, 1970.
[10] D. E. Knuth, "Volume 2: Seminumerical algorithms (3^{st} edition)," *the Art of Computer Programming, Addison-Wesley*, 1998.
[11] 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.

VI. APPENDIX

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
1^{st}-order	7	6	8	4	7	6	10	4	8	4	0.609372	63/64	Frequency
	12	9	7	4	6	6	5	4	7	4	0.465914	62/64	BlockFrequency
	8	6	11	8	5	5	5	7	5	4	0.517608	63/64	CumulativeSums (m-2)
	7	8	10	8	11	4	8	3	2	3	0.109242	63/64	CumulativeSums (m-3)
	7	2	14	10	6	2	5	5	8	5	0.023812	63/64	Runs
	6	3	12	7	4	9	7	7	5	4	0.366511	63/64	LongestRun
	7	9	6	6	5	4	8	7	8	4	0.817009	64/64	Rank
	1	2	3	0	4	4	0	6	0	1	0.028264	21/21	FFT
	10	8	3	9	5	5	7	6	5	6	0.517608	63/64	ApproximateEntropy
	10	5	6	7	6	5	6	6	6	7	0.933004	64/64	Serial (forward)
	4	9	9	6	6	8	3	6	4	9	0.380722	64/64	Serial (backward)

TABLE III

NIST TEST RESULTS DERIVED FROM THE SECOND HALF OF THE VIRGINIA TEACH DATASET [11] WITH 1^{st}-ORDER POLYNOMIAL REGRESSION MODEL APPLIED IN STEP 2. ALL TESTS ARE PASSED.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
1^{st}-order	5	4	3	2	7	0	2	5	3	2	0.259438	31/33	Frequency
	9	6	1	3	2	1	4	2	3	2	0.033490	31/33	BlockFrequency
	7	5	2	4	2	1	4	1	6	1	0.120885	31/33	CumulativeSums (m-2)
	6	4	3	3	4	3	5	1	4	0	0.502674	30/33	CumulativeSums (m-3)
	8	2	3	2	1	4	2	2	5	4	0.216159	32/33	Runs
	6	7	2	5	2	4	0	2	1	4	0.098607	33/33	LongestRun
	6	3	3	8	1	2	3	2	4	1	0.120885	31/33	Rank
	0	3	1	0	1	1	0	5	0	0	0.003950	11/11	FFT
	9	4	2	4	3	0	2	4	3	2	0.064760	33/33	ApproximateEntropy
	7	3	2	5	4	5	4	3	0	0	0.120885	32/33	Serial (forward)
	3	3	5	3	2	5	3	5	1	3	0.697921	32/33	Serial (backward)

TABLE IV

NIST TEST RESULTS DERIVED FROM THE SECOND HALF OF OUR OWN DATASET WITH 1^{st}-ORDER POLYNOMIAL REGRESSION MODEL APPLIED IN STEP 2. ALL TESTS ARE PASSED

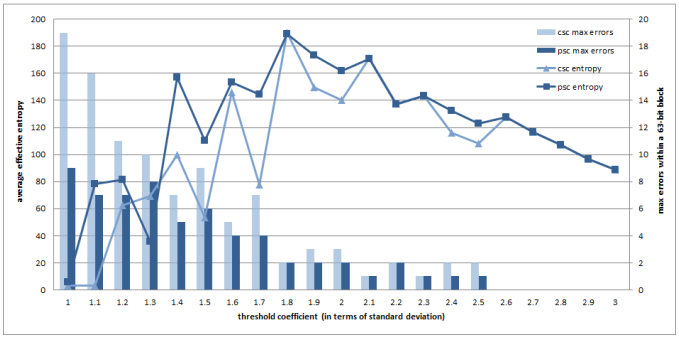

Fig. 3. The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(63, k, t) based on the Virginia Tech dataset

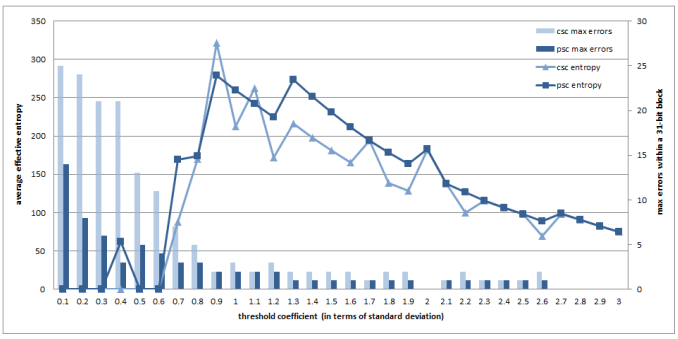

Fig. 5. The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(31, k, t) based on the in-house dataset

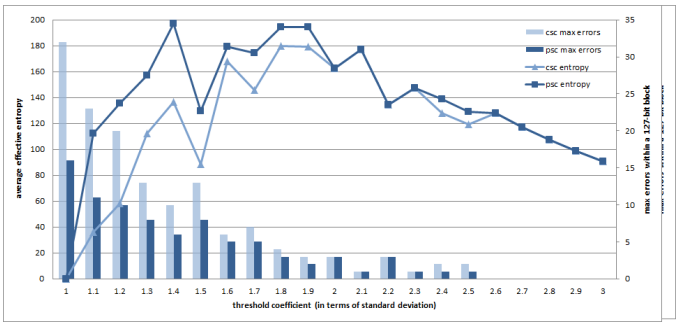

Fig. 4. The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(127, k, t) based on the Virginia Tech dataset

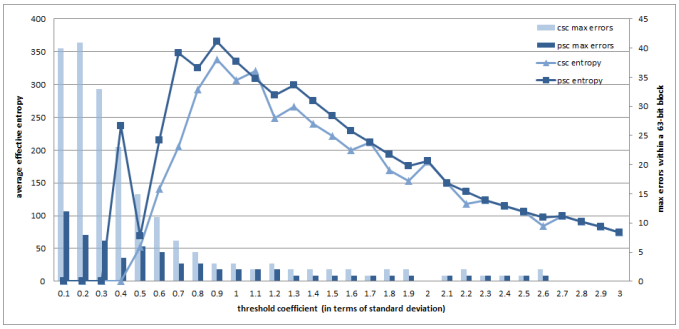

Fig. 6. The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(63, k, t) based on the in-house dataset

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	10	9	4	6	4	4	9	3	5	11	0.187777	64/65	Frequency
	11	10	12	6	5	3	6	4	5	3	0.068316	64/65	BlockFrequency
	12	8	5	11	6	3	9	3	5	3	0.061374	63/65	CumulativeSums (m-2)
	9	10	5	7	4	7	5	6	7	5	0.676274	63/65	CumulativeSums (m-3)
	19	9	5	8	4	6	5	4	2	3	0.000049 *	59/65 *	Runs
	7	6	7	14	14	4	5	1	3	4	0.000919	65/65	LongestRun
	11	4	8	6	7	6	4	5	10	4	0.484073	64/65	Rank
	4	2	2	0	3	3	0	3	0	4	0.323011	21/21	FFT
	18	9	5	6	6	5	8	5	2	1	0.000145	62/65	ApproximateEntropy
	15	9	5	9	6	5	1	7	4	4	0.009867	63/65	Serial (forward)
	6	3	4	7	11	11	7	6	9	1	0.075967	65/65	Serial (backward)
1^{st}-order	7	7	6	7	9	2	8	5	5	9	0.494547	63/65	Frequency
	9	7	11	6	4	4	6	5	4	9	0.521707	61/65	BlockFrequency
	8	8	6	10	6	2	5	6	9	5	0.314919	63/65	CumulativeSums (m-2)
	6	7	9	7	5	6	4	8	7	6	0.921761	63/65	CumulativeSums (m-3)
	6	9	5	6	8	7	7	8	3	6	0.798722	64/65	Runs
	8	3	6	8	14	4	4	5	8	5	0.093645	65/65	LongestRun
	7	8	7	4	7	6	9	7	6	4	0.867205	62/65	Rank
	1	3	4	0	2	3	0	4	0	4	0.187777	21/21	FFT
	9	6	5	7	6	5	6	7	9	5	0.896359	63/65	ApproximateEntropy
	13	4	6	11	3	4	10	6	7	1	0.011121	65/65	Serial (forward)
	7	5	6	7	13	7	2	5	10	3	0.093645	63/65	Serial (backward)
2^{nd}-order	8	7	9	6	10	5	4	5	5	6	0.631944	64/65	Frequency
	11	8	11	7	4	5	5	6	3	5	0.296269	64/65	BlockFrequency
	10	6	7	6	7	3	5	5	8	8	0.631944	64/65	CumulativeSums (m-2)
	8	10	6	7	5	6	4	5	6	8	0.760113	64/65	CumulativeSums (m-3)
	13	8	9	4	8	2	6	3	5	7	0.084389	62/65	Runs
	8	1	8	12	11	1	7	7	6	4	0.017828	64/65	LongestRun
	9	8	6	2	5	5	7	7	8	8	0.540669	64/65	Rank
	1	0	4	0	8	5	0	3	0	0	0.000065 *	21/21	FFT
	12	8	8	3	6	8	6	5	5	4	0.414146	63/65	ApproximateEntropy
	14	9	5	6	10	3	5	5	4	4	0.044252	64/65	Serial (forward)
	12	5	4	5	8	5	9	6	4	7	0.448203	64/65	Serial (backward)
3^{rd}-order	6	9	6	7	10	2	7	3	8	7	0.521707	65/65	Frequency
	7	8	9	9	8	2	6	9	1	6	0.226378	64/65	BlockFrequency
	3	13	5	10	10	10	3	0	7	4	0.002550	65/65	CumulativeSums (m-2)
	3	11	7	5	10	5	5	7	7	5	0.521707	65/65	CumulativeSums (m-3)
	10	10	6	8	8	4	10	4	3	2	0.127107	64/65	Runs
	4	5	7	8	14	2	3	9	4	9	0.022481	65/65	LongestRun
	6	10	6	3	5	3	2	8	14	8	0.020027	65/65	Rank
	2	6	1	0	2	8	0	2	0	0	0.000097 *	21/21	FFT
	11	6	6	11	7	3	7	3	5	6	0.296269	64/65	ApproximateEntropy
	9	11	8	3	6	5	7	4	5	7	0.561026	64/65	Serial (forward)
	8	8	5	8	2	7	7	5	6	9	0.540669	63/65	Serial (backward)
4^{th}-order	9	5	4	7	6	3	11	6	3	11	0.170659	63/65	Frequency
	8	13	8	4	3	10	4	5	4	6	0.093645	64/65	BlockFrequency
	7	5	9	5	2	4	10	9	8	6	0.448203	63/65	CumulativeSums (m-2)
	8	7	4	6	10	3	10	4	6	7	0.561026	62/65	CumulativeSums (m-3)
	7	9	6	6	6	1	6	9	8	7	0.358516	64/65	Runs
	4	10	5	8	11	9	4	5	3	6	0.271286	65/65	LongestRun
	9	5	5	3	6	11	10	5	6	5	0.414146	63/65	Rank
	3	4	2	0	3	3	0	2	0	4	0.323011	20/21	FFT
	8	10	4	7	8	5	2	7	4	10	0.351554	63/65	ApproximateEntropy
	9	5	9	5	5	7	6	4	7	8	0.760113	65/65	Serial (forward)
	3	6	6	7	10	5	8	9	3	8	0.272584	65/65	Serial (backward)
5^{th}-order	6	8	5	3	11	3	11	6	6	6	0.271286	64/65	Frequency
	13	5	8	5	10	7	5	7	0	5	0.039609	63/65	BlockFrequency
	7	8	6	3	9	3	4	8	9	8	0.272584	64/65	CumulativeSums (m-2)
	8	9	7	6	5	3	5	13	6	3	0.170659	64/65	CumulativeSums (m-3)
	13	9	5	9	6	3	6	7	3	4	0.114933	61/65	Runs
	3	9	12	10	13	4	3	1	4	6	0.001981	64/65	LongestRun
	5	3	13	7	3	7	3	9	7	8	0.103803	63/65	Rank
	0	2	4	0	4	7	0	1	0	3	0.003277	21/21	FFT
	18	8	6	1	9	5	5	1	5	7	0.000074 *	64/65	ApproximateEntropy
	12	7	3	11	7	8	7	4	4	2	0.068316	64/65	Serial (forward)
	6	8	8	6	5	8	6	10	7	1	0.561026	65/65	Serial (backward)
6^{th}-order	7	4	9	2	4	5	9	8	5	12	0.154893	64/65	Frequency
	14	13	13	8	2	2	4	4	2	3	0.000019 *	65/65	BlockFrequency
	7	10	8	8	5	6	3	3	11	4	0.271286	65/65	CumulativeSums (m-2)
	5	6	8	6	12	7	4	6	6	5	0.403161	65/65	CumulativeSums (m-3)
	15	14	3	6	5	2	6	7	2	5	0.000323	64/65	Runs
	7	3	13	9	10	7	5	2	4	5	0.049394	65/65	LongestRun
	8	8	5	5	7	5	6	6	7	8	0.960834	63/65	Rank
	1	4	3	0	1	5	0	4	0	3	0.075967	21/21	FFT
	15	7	5	9	7	5	3	4	7	3	0.028264	64/65	ApproximateEntropy
	13	7	8	10	5	6	4	0	8	4	0.025217	64/65	Serial (forward)
	10	8	5	6	9	4	4	6	4	9	0.314919	64/65	Serial (backward)

TABLE V

NIST TEST RESULTS DERIVED FROM THE FIRST HALF OF THE DATASET [11], WHERE THE LENGTH OF ONE BITSTRING IS 400 (EXCEPT 3X FOR FFT TEST), THE BLOCK LENGTH FOR FREQUENCY TEST 32, THE BLOCK LENGTH FOR APPROXIMATE ENTROPY TEST 2 AND THE BLOCK LENGTH FOR SERIAL TEST 5. '*' MARKS A FAILURE.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0^{th}-order	13	3	3	1	1	1	4	1	2	5	0.000008 *	30/34 *	Frequency
	16	3	4	3	0	2	2	1	2	1	0.000000 *	29/34 *	BlockFrequency
	14	4	3	1	2	2	2	1	3	2	0.000001 *	27/34 *	CumulativeSums (m-2)
	10	5	5	2	3	0	2	2	3	2	0.007297	29/34 *	CumulativeSums (m-3)
	15	6	4	2	0	3	1	1	1	1	0.000000 *	26/34 *	Runs
	12	1	4	3	3	4	4	1	2	0	0.000119	31/34	LongestRun
	5	3	6	7	4	1	2	2	3	1	0.196868	33/34	Rank
	1	2	1	0	2	1	0	1	0	3	0.581286	11/11	FFT
	17	4	3	4	1	1	1	2	1	0	0.000000 *	24/34 *	ApproximateEntropy
	16	4	2	6	1	1	2	1	0	1	0.000000 *	26/34 *	Serial (forward)
	6	5	3	2	4	2	3	5	2	2	0.471531	34/34	Serial (backward)
1^{st}-order	6	5	0	5	5	0	5	2	3	3	0.133610	33/34	Frequency
	6	6	4	2	4	1	2	2	4	3	0.541162	33/34	BlockFrequency
	6	3	5	3	4	1	1	3	4	4	0.380722	33/34	CumulativeSums (m-2)
	6	5	1	6	3	1	1	2	5	4	0.196868	33/34	CumulativeSums (m-3)
	7	5	4	2	4	0	4	2	0	6	0.058152	33/34	Runs
	3	1	4	7	6	3	2	2	5	1	0.196868	34/34	LongestRun
	4	3	4	5	0	4	4	3	6	1	0.465914	32/34	Rank
	2	4	0	0	0	1	0	4	0	0	0.003950	11/11	FFT
	6	1	6	6	3	3	3	1	3	2	0.283561	32/34	ApproximateEntropy
	6	5	6	4	3	1	3	2	0	4	0.236992	32/34	Serial (forward)
	3	5	7	7	4	2	0	1	1	4	0.037462	34/34	Serial (backward)
2^{nd}-order	9	3	3	1	4	2	4	1	2	5	0.058152	32/34	Frequency
	8	4	4	4	1	2	4	6	1	0	0.037462	32/34	BlockFrequency
	6	6	2	4	3	5	3	1	2	2	0.465914	33/34	CumulativeSums (m-2)
	11	1	4	4	5	0	2	3	3	1	0.000757	32/34	CumulativeSums (m-3)
	11	6	2	0	2	1	6	1	1	4	0.000069 *	31/34	Runs
	6	3	7	3	3	4	0	3	4	1	0.196868	33/34	LongestRun
	9	8	2	4	1	3	2	2	3	0	0.002716	32/34	Rank
	2	2	4	0	0	2	0	1	0	0	0.064760	10/11	FFT
	12	3	6	4	4	1	2	0	1	1	0.000023 *	32/34	ApproximateEntropy
	11	1	7	1	5	2	2	0	4	1	0.000053 *	32/34	Serial (forward)
	5	5	6	3	2	1	2	2	7	1	0.133610	34/34	Serial (backward)
3^{rd}-order	3	5	4	5	1	2	5	3	4	2	0.563683	34/34	Frequency
	5	6	5	2	5	2	2	3	3	1	0.541162	33/34	BlockFrequency
	5	2	5	6	2	2	5	3	2	2	0.293582	34/34	CumulativeSums (m-2)
	4	2	2	5	7	3	4	3	2	2	0.293582	34/34	CumulativeSums (m-3)
	7	5	7	1	3	3	4	1	1	2	0.072049	31/34	Runs
	1	1	5	4	6	5	4	5	1	2	0.283561	34/34	LongestRun
	6	4	4	3	4	4	2	3	1	3	0.654263	33/34	Rank
	1	4	2	0	1	1	0	1	0	1	0.216159	11/11	FFT
	4	5	9	3	3	2	1	2	4	1	0.058152	34/34	ApproximateEntropy
	7	3	5	1	4	2	4	2	3	3	0.541162	33/34	Serial (forward)
	4	4	8	2	1	3	3	1	4	4	0.236992	34/34	Serial (backward)
4^{th}-order	5	7	1	1	5	4	4	2	1	4	0.196868	33/34	Frequency
	7	2	6	1	7	0	4	3	2	2	0.029914	34/34	BlockFrequency
	5	5	3	2	5	2	2	3	4	3	0.739897	33/34	CumulativeSums (m-2)
	4	7	1	6	2	2	3	2	4	3	0.337055	34/34	CumulativeSums (m-3)
	5	8	4	2	2	0	4	7	1	1	0.011803	33/34	Runs
	4	2	4	4	7	2	2	4	4	1	0.541162	34/34	LongestRun
	6	1	2	3	5	3	4	4	2	4	0.471531	34/34	Rank
	1	4	2	0	0	3	0	1	0	0	0.033490	11/11	FFT
	7	6	1	4	5	1	5	1	3	1	0.072049	33/34	ApproximateEntropy
	4	4	5	3	4	5	3	4	2	0	0.471531	34/34	Serial (forward)
	2	5	4	5	6	4	0	2	1	5	0.236992	34/34	Serial (backward)
5^{th}-order	4	3	2	2	4	2	6	3	2	6	0.380722	34/34	Frequency
	5	1	2	2	3	1	2	6	5	7	0.133610	34/34	BlockFrequency
	2	5	1	4	3	1	4	7	5	2	0.283561	34/34	CumulativeSums (m-2)
	4	0	1	2	4	5	3	4	5	6	0.337055	34/34	CumulativeSums (m-3)
	5	7	3	5	1	2	3	2	1	5	0.236992	34/34	Runs
	2	1	5	6	9	4	3	3	1	0	0.009292	34/34	LongestRun
	3	5	2	3	6	1	4	1	4	5	0.541162	34/34	Rank
	1	2	1	0	2	1	0	3	0	1	0.581286	11/11	FFT
	7	5	1	5	2	1	3	2	4	4	0.283561	34/34	ApproximateEntropy
	5	6	5	3	1	4	3	4	1	2	0.541162	34/34	Serial (forward)
	5	5	2	1	4	8	2	2	4	1	0.109242	34/34	Serial (backward)
6^{th}-order	7	4	3	4	5	1	3	1	4	2	0.397806	34/34	Frequency
	6	2	4	5	4	3	5	0	0	5	0.162612	33/34	BlockFrequency
	6	4	5	3	2	3	2	4	3	2	0.654263	34/34	CumulativeSums (m-2)
	8	1	4	8	3	1	2	1	2	4	0.011803	34/34	CumulativeSums (m-3)
	11	3	3	0	4	3	3	3	2	2	0.003488	32/34	Runs
	3	4	9	7	4	1	0	3	2	1	0.005718	34/34	LongestRun
	8	3	6	3	3	0	3	3	3	2	0.133610	31/34	Rank
	2	2	2	0	0	2	0	2	0	1	0.581286	11/11	FFT
	9	5	5	1	2	3	4	1	2	2	0.037462	33/34	ApproximateEntropy
	8	6	2	3	6	2	4	1	2	0	0.023812	31/34	Serial (forward)
	4	4	6	3	3	2	2	4	4	2	0.739897	32/34	Serial (backward)

TABLE VI

NIST TEST RESULTS USING THE FIRST HALF OF THE IN-HOUSE DATASET, WHERE THE LENGTH OF ONE BITSTRING IS 400 (EXCEPT 3X FOR FFT TEST), THE BLOCK LENGTH FOR FREQUENCY TEST 32, THE BLOCK LENGTH FOR APPROXIMATE ENTROPY TEST 2 AND THE BLOCK LENGTH FOR SERIAL TEST 5. '*' MARKS A FAILURE.

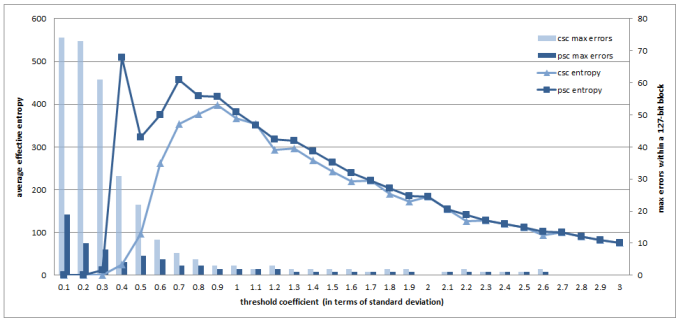

Fig. 7. The estimated H_{∞}^{CSC} and H_{∞}^{KSC} derived with BCH(127, k, t) based on the in-house dataset

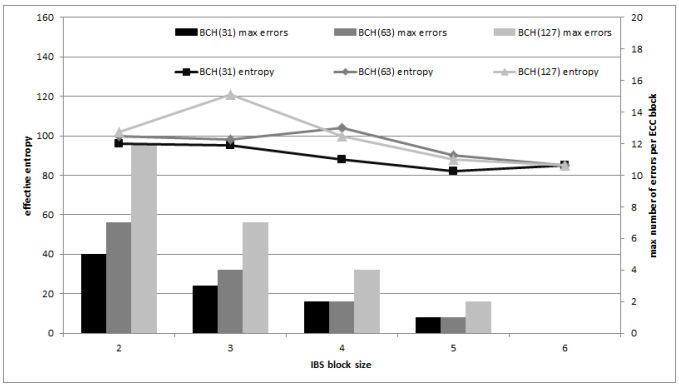

Fig. 8. The estimated H_{∞}^{IBS} derived from the Virginia Tech dataset

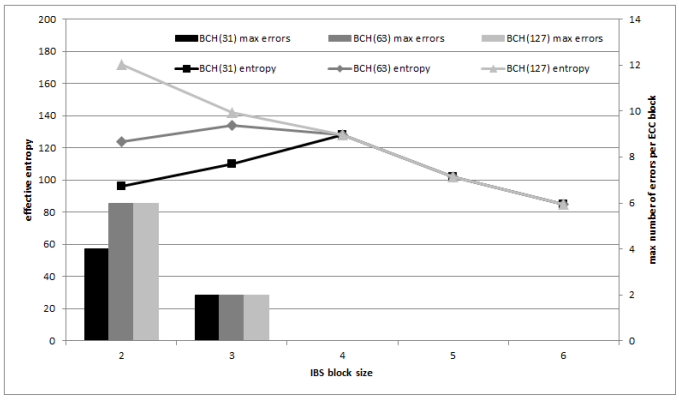

Fig. 9. The estimated H_{∞}^{IBS} derived from the in-house dataset