Analysis of Entropy in a Hardware-Embedded Delay PUF

Wenjie Che, Venkata K. Kajuluri, Mitchell Martin, Fareena Saqib* and Jim Plusquellic
University of New Mexico, *Florida Institute of Technology

***Abstract** -- The magnitude of the information content associated with a particular implementation of a Physical Unclonable Function (PUF) is critically important for security and trust in emerging IoT applications. Authentication, in particular, requires the PUF to produce a very large number of challenge-response-pairs (CRPs) and of even greater importance, requires the PUF to be resistant to adversarial attacks that attempt to model and clone the PUF (model-building attacks). Entropy is critically important to the model-building resistance of the PUF. A variety of metrics have been proposed for reporting Entropy, each measuring the randomness of information embedded within PUF-generated bitstrings. In this paper, we report the Entropy, minEntropy, conditional minEntropy, Interchip hamming distance and NIST statistical test results using bitstrings generated by a Hardware-Embedded Delay PUF called HELP. The bitstrings are generated from data collected in hardware experiments on 500 copies of HELP implemented on a set of Xilinx Zynq 7020 SoC Field Programmable Gate Arrays (FPGAs) subjected to industrial-level temperature and voltage conditions. Special test cases are constructed which purposely create worst case correlations for bitstring generation. Our results show that the processes proposed within HELP to generate bitstrings add significantly to their Entropy, and show that classical re-use of PUF components, e.g., path delays, does not result in large Entropy losses commonly reported for other PUF architectures.*

Index terms -- Entropy Analysis, Physical Unclonable Function, Authentication.

I. Introduction

The number of independent sources of information used to distinguish a system is a measure of its complexity, and relates to the amount of effort required to copy or clone it. The relationship between complexity and effort can be exponential, particularly for systems designed to conceal or mask the information and only provide controlled access to it. A physical unclonable function (PUF) is an information system that can meet these criteria under certain conditions. The information embedded in a PUF is random, enabling it to serve hardware security and trust roles related to key generation, key management, tamper detection and authentication [1]. PUFs represent an alternative to storing keys in non-volatile-memory (NVM), thereby reducing cost and hardening the embedding system against key-extraction-based attacks. PUFs are widely recognized as next-generation security and trust primitives that are ideally suited for authentication in industrial, automotive, consumer and military IoT-based systems, and for dealing with many of the challenges related to counterfeits in the supply chain.

PUFs enable access to their stored random information using a challenge-response-pair (CRP) mechanism, whereby a server or adversary 'asks a question' usually in the form of a digital bitstring and the PUF produces a digital response after measuring a set of circuit parameters within the chip. The nanometer size of the integrated circuit (IC) features and the analog nature of stored information makes it extremely difficult to read out the information using alterative access mechanisms. The circuit parameters that are measured vary from one copy of the chip to another, and can only be controlled to a small, but non-zero, level of tolerance by the chip manufacturer. This feature of the PUF makes it unclonable and provides each copy of the chip with a distinct 'personality', in the spirit of fingerprints or DNA for biological systems.

Strong PUFs are a special class of PUFs that are distinguished from *weak PUFs* by the amount of information content they possess. The traditional definition for distinguishing between weak and strong PUFs is to consider only the number of CRPs that can be applied. For weak PUFs, the number of CRPs is polynomial while strong PUFs have an exponential number, e.g., the number of challenges for an n-binary-input weak PUF can be n^2 while a strong PUF typically has 2^n. Unfortunately, this traditional definition leads to a misnomer as to the true strength of the PUF to adversary attacks. For example, the original Arbiter PUF [2-3] is classified as strong even through machine-learning-based model-building attacks have shown that only a small, polynomial, number of CRPs are needed to predict its complete behavior.

Therefore, a truly strong PUF must have both an exponential number of CRPs and an exponential number of unique, uncorrelated responses, i.e., a large input challenge space is necessary but is not a sufficient condition. This requires the PUF to have access to a large source of Entropy, either in the form of IC features from which random information is extracted, or in an artificial form using a cryptographic primitive, such as a secure hash function. Either mechanism makes the PUF resilient to machine learning attacks. However, using a secure hash for expanding the CRP space of the PUF and for obfuscating its responses consumes additional area and increases the required reliability of the PUF. Therefore, the former scenario, i.e., a large source of Entropy, is more attractive but more difficult to achieve.

In this paper, we present results that support this more attractive alternative using a hardware-embedded delay PUF called HELP. HELP generates bitstrings from delay variations that occur along paths in an on-chip macro, i.e., the source of Entropy for HELP is within-die manufacturing process variations that cause path delays to be slightly different in each copy of the chip. Macros or functional units that implement cryptographic algorithms and common data path operators such as multipliers typically possess at least 32 inputs and therefore, HELP meets the large input space requirement of a strong PUF.

Moreover, the wire interconnectivity within the macro used by HELP provides a large number of testable paths, on order of 2^n for n inputs, satisfying the large output space requirement of a strong PUF. Unlike other PUFs that meet these conditions, the task of generating input test sequences (challenges) that test all of the testable paths is an NP-complete problem. Although this may appear to be a drawback, it, in fact, makes the task of model-building HELP much more difficult. For example, the adversary not only must devise a machine learning strategy that is able to predict output responses, but he/she must also expend a large effort on generating the challenges, which is typically accomplished using automatic test pattern generation (ATPG) algorithms. Note that these characteristics of HELP, namely, the use of a functional unit as a source of Entropy, paths of arbitrary length and the ATPG requirement, distinguish HELP from other delay-based PUFs such as the Arbiter and RO PUFs.

This paper investigates Entropy within and across HELP-generated bitstrings using 500 instances of a functional unit (the Entropy source) embedded on a set of 20 Xilinx Zynq 7020 FPGAs. The specific contributions of this paper include the following:

- Strong experimental evidence that HELP leverages within-die variations (WDV) almost exclusively as its source of Entropy.
- A statistical evaluation of Entropy, minEntropy, Conditional minEntropy, Interchip Hamming Distance and NIST statistical test results on hardware generated bitstrings.
- A special worst-case analysis that maximizes correlations and dependencies introduced by 1) **full path reuse** and 2) **partial path reuse** where the ***same*** paths in different combinations, or paths with *many common segments*, are used to generate distinct bits.

The rest of this paper is organized as follows. Related work is presented in Section II and an overview of HELP is given in Section III. Statistical results are described in Section IV using FPGA-based path delay data and bitstrings. A worst-case correlation analysis is presented in Section V and conclusions in Section VI.

II. Related Work

The source of random information varies widely among proposed PUF architectures, and includes transistor threshold voltages [4], delay chains and ring oscillators (RO) [2-6], FPGAs [7-8], SRAMs [9], leakage current [10], metal resistance [11], transistor transconductance [12], the path delays of core logic macros [13-15], memristors [16], scan chains [17], phase change memory [18], plus many others.

One of the earliest delay-based PUFs, called the Arbiter PUF, uses n-bit differential delay lines and a latch to generate a 1-bit PUF response [19-20]. Because of the limited amount of Entropy, model-building attacks are effective against the Arbiter PUFs [21]. Ring Oscillator (RO) PUF [22-23] measure the frequency difference between two identical ring oscillators by counting the transitions on the output of each RO and then comparing counter values to generate a PUF bit. The number of challenges is limited to the number of pairings (n^2) and therefore the RO PUF is a weak PUF. The authors of [24] analyze RO frequency differences, selecting those pairings where the frequency difference is large enough to avoid any bit flip errors caused by environmental variations. The authors of [25] propose a scheme to produce (n-1) reliable bits, and [26] proposes a longest increasing subsequence-based grouping algorithm (LISA) for FPGAs that sequentially pairs RO-PUF bits and can generate n/2 reliable bits out of n ring oscillators. In [27], the authors proposes a regression based distiller to remove systematic variations.

PUF responses are affected by the environmental variations such as temperature and voltage variations, thus processing is required to extract the Entropy from the noise. Several schemes including helper data and fuzzy extractor schemes are proposed to improve the reliability of bitstring regeneration and improve randomness [28]. Helper data is generated during the enrollment phase which is carried out in a secure environment and is later used with the noisy responses generated during regeneration to reconstruct the key. Bosch et al. [29] demonstrated a hardware implementation of concatenated codes based fuzzy extractors that have been used to produce bitstrings with high reliability. Reference [30] discusses a fuzzy extractor scheme based on repetition codes that can limit the usable entropy and show that such a scheme is not applicable to PUFs with small levels of Entropy. Dodis et al [31] provided a formal definition and analysis of Entropy loss in fuzzy extractors. The authors of [32] evaluated the reliability and unpredictability properties of five different types of PUFs (Arbiter, RO, SRAM, flip-flop and latch PUFs) from an ASIC implementation.

III. HELP Overview

HELP attaches to an on-chip functional unit, such as a portion of the Advanced Encryption Standard (AES) labeled *sbox-mixedcol* on the left side of Fig. 1. The logic gate structure of the functional unit defines a complex interconnected network of wires and transistors. This combinational data path component includes 64 primary inputs (PIs) and 64 primary outputs (POs) and is implemented in WDDL logic-style [33] on a Xilinx Zynq FPGA using approx. 2,900 LUTs

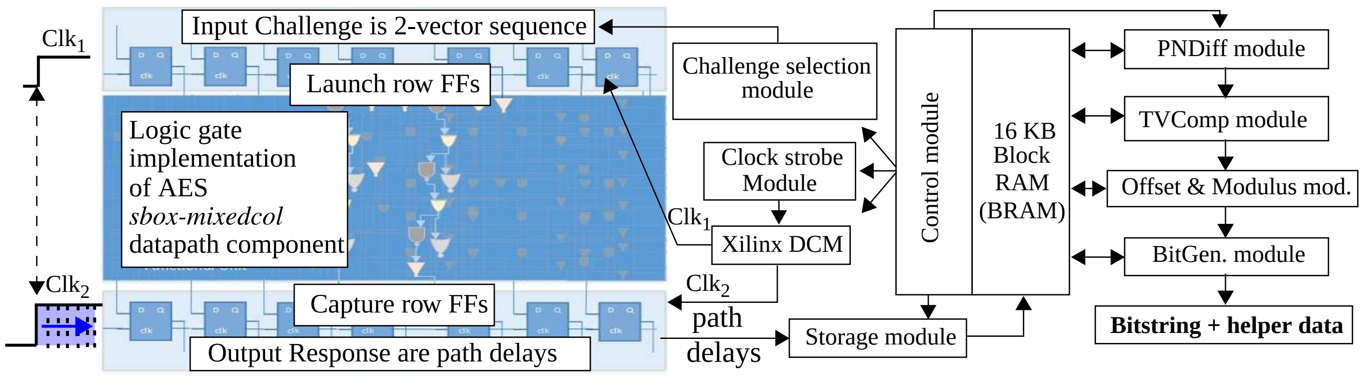


Fig. 1. Instantiation of the HELP entropy source (left) and HELP processing engine (right).

and 30K wire segments.

Path delay is defined as the amount of time (Δt) it takes for a set of 0-to-1 and 1-to-0 bit transitions introduced on the PIs of the functional unit (input challenge) to propagate through the logic gate network and emerge on a PO. HELP uses a clock-strobing technique to obtain high resolution measurements of path delays as shown on the left side of Fig. 1. A series of launch-capture operations are applied in which the vector sequence that defines the input challenge is applied repeatedly to the PIs using the Launch row flip-flops (FFs) and the output responses are measured on the POs using the Capture row FFs. On each application, the phase of the capture clock, Clk_2, is incremented forward with respect to Clk_1, by small Δts (approx. 18 ps), until the emerging signal transition on a PO is successfully captured in the Capture row FFs. A set of XOR gates connected to the Capture row FF inputs and outputs (not shown) provide a simple means of determining when this occurs. When an XOR gate value becomes 0, then the input and output of the FF are the same (indicating a successful capture). The first occurrence in which this occurs during the clock strobe sweep causes the current phase shift value to be recorded as the digitized delay value for this path. The current phase shift value is referred to as the launch-capture-interval (**LCI**). The *Clock strobe* module is shown in the center portion of Fig. 1, which utilizes features on Xilinx Digital Clock Manager (DCM).

The digitized path delays are collected by a *Storage* module and stored in an on-chip block RAM (BRAM) as shown in the center of Fig. 1. Each digitized timing value is stored as a 16-bit value, with 12 binary digits serving to cover a signed range between +/- 2048 and 4 binary digits of fixed point precision to enable up to 16 samples of each path delay to be measured and averaged. The digitized path delays are stored in the upper half of the 16 KByte BRAM. We configure the applied challenges to test 2048 paths with rising transitions and 2048 paths with falling transitions. The digitized path delays are referred to as PUFNums, or **PN**, with **PNR** used to refer to rising path delays and **PNF** for falling. Once a set of 4096 PN are collected, a sequence of operations implemented in VHDL are started to produce the

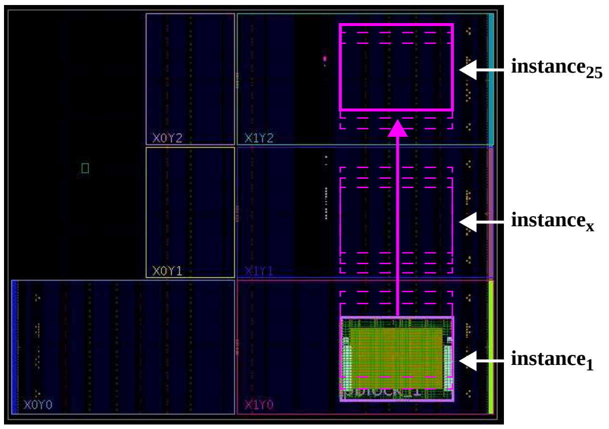


Fig. 2. *sbox-mixedcol* functional unit instance placement in Xilinx Zynq 7020 using Vivado implementation view.

bitstring and helper data, as shown on the far right of Fig. 1. These operations are described below.

A. Implementation Details

We created 25 instances of *sbox-mixedcol* on each of 20 chips, for a total of 500 implementations (25 separate programming bitstreams are generated). Fig. 2 shows a screen snapshot of Xilinx Vivado Implementation view which depicts a completed instance of the functional unit in the lower right corner (labeled as $instance_1$). The VHDL code for *sbox-mixedcol* is synthesized and implemented into a *pblock*, which is shown as a magenta rectangle surrounding $instance_1$. Once completed, *tcl* commands are issued that save a set of constraints for the wire and LUT components of the functional unit to a file called a *check-point*. The base *y* coordinate of the *pblock* is then incremented by 3 to create a sequence of *pblock* implementations, each of which is synthesized into a separate bitstream. In this fashion, a sequence of identical and overlapping *pblock* instances of the functional unit are created and tested, one at a time. The rationale for doing this is two-fold. First, it increases the statistical significance of the analysis without requiring a corresponding increase in the number of FPGAs. Second, data from over-

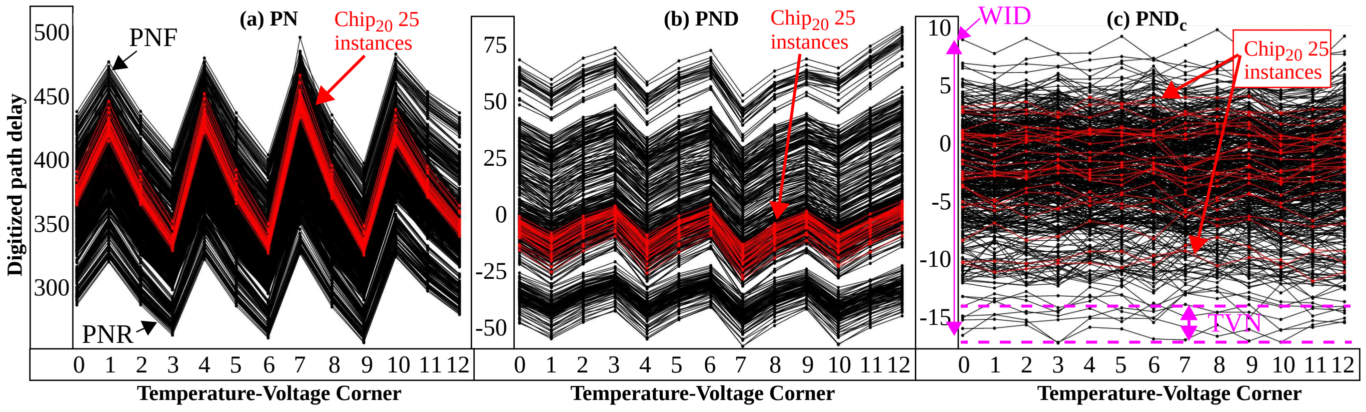


Fig. 3. (a) Example rising and falling path delays (PN), (b) Rise-fall delay (PND) and (c) TV Compensated PND (PND_c).

lapping instances on the same FPGA implicitly eliminate chip-to-chip process variations, and provides a basis on which we can prove experimentally that HELP leverages within-die variations almost exclusively.

B. PN, PND and PND_c Processing Steps

The PN processing operations shown on the far right in Fig. 1 are designed to eliminate both chip-to-chip performance differences and environmental variations, while leaving only within-die variations as a source of entropy for HELP. In order to accomplish this, the following modules and operations are defined. The *PNDiff* module creates unique, pseudo-random pairings between elements of the PNR and PNF groups using two seeded linear feedback shift registers (LFSR). The LFSRs are used to generate 11-bit addresses to access any of the 2048 PNR and PNF values. The two 11-bit *LFSR seeds* are configuration parameters. The PN differences are referred to as **PND**. The primary reason for creating PND is to increase the magnitude of within-die variations, i.e., path delay variations are doubled (in the best case) over those available in the PNR and PNF.

Fig. 3(a) shows an example of this process using a pairing of paths from the PNR and PNF sets. The graph contains curves for 500 PNR and 500 PNF, one for each of the 500 chip-instances. Although it is difficult to distinguish between the two groups in the figure, the PNF have a larger delay and are displayed above the PNR. The 13 line-connected points in each curve represent the PN measured under a range of environmental conditions, called temperature-voltage (TV) corners. The PN at the x-axis position given by 0 are those measured under nominal conditions (referred to as **enrollment** values below), i.e., at 25^oC, 1.00V. The PN at positions 1, 2 and 3 are also measured at 25^oC but at supply voltages of 0.95, 1.00 and 1.05 V. Similarly, the other groups of 3 consecutive points along the x-axis are measured at these supply voltages but at temperatures 0^oC, -40^oC and 85^oC. The PN measured under TV corners numbered 1 to 12 are referred to as **regeneration** PN. Fig. 3(b) plots the corresponding PND defined by subtracting pointwise, each PNF from a PNR for each chip-instance.

TV-related effects on delay negatively impact bitstring reproducibility. It is clear that subtraction alone which is used to create the PND is not effective at removing all of the variations introduced by different environmental conditions (if it was, the curves would be horizontal lines). We propose a *TV compensation* (TVComp) process that is applied to the PND as a mechanism to eliminate most of the remaining temperature-voltage variations (called TV-noise).

TVComp is applied to the entire set of 2048 PND measured for each chip-instance at each of the 13 TV corners separately (note, Fig. 3(b) shows only one of the PND from the larger set of 2048 that exist for each chip-instance and TV corner). The TVComp procedure first converts the PND to 'standardized' values. Eq. (1) represents the first transformation which makes use of two constants, i.e., μ_{chip} (mean) and Rng_{chip} (range), obtained by measuring the mean and range of the distribution defined by the PND. The second

$$zval_i = \frac{(PND_i - \mu_{chip})}{Rng_{chip}} \qquad \textbf{Eq. 1.}$$

$$PND_c = zval_i Rng_{ref} + \mu_{ref} \qquad \textbf{Eq. 2.}$$

transformation is represented by Eq. (2), which translates the standardized *zvals* to a new distribution with mean μ_{ref} and range Rng_{ref}. The **reference** mean and range values are also configuration parameters. In our experiments, we fix μ_{ref} and Rng_{ref} in the TVComp operation for all chip-instances as a means of eliminating chip-to-chip performance differences.

Fig. 3(c) illustrates the effect of TVComp under these conditions. The $\mathbf{PND_c}$ ('c' for compensated) plotted in the graph are obtained by applying the TVComp procedure to the 2048 PND measured under each of the 13 TV corners for each chip, i.e., 13 TV corners * 500 chip-instances = 6500 separate applications. Several featured of TVComp are evident. First, the transformation significantly reduces TV-noise which is reflected by the flatter curves (note the scale used on the y-axis is amplified over that shown in Fig. 3(b)). Second,

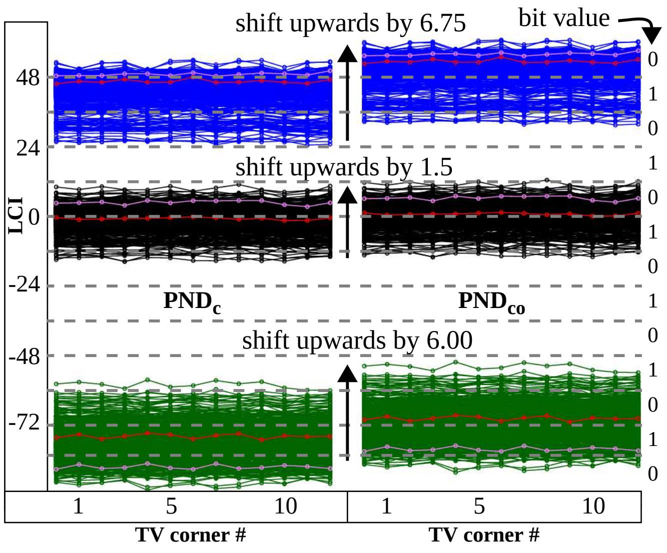


Fig. 4. Three example PND_c from 500 chip-instances (y-axis) at each of the 13 TV corners. PND_c before 4-bit offset is added (left) and afterwards (right), PND_{co}, using a Modulus of 24. Dashed lines identify 0-1 lines, with corresponding bit values associated with each region shown on the far right. Two chip-instances are highlighted as red and magenta to illustrate their random occurrence among different sets of PND_c, which is caused by within-die variation effects.

global (chip-wide) performance differences are also nearly eliminated between the chip-instances, leaving only within-die variations. This is illustrated nicely by the highlighted red curves (25 instances) for $chip_{20}$. The curves shown in Fig. 3(a) and (b) for the 25 instances on $chip_{20}$ are grouped together, illustrating these instances have similar performance characteristics as expected since they are obtained from the same chip. However, the corresponding curves in 3(c) are distributed across most the y range, and are indistinguishable from the 450 curves from the other 19 chip-instances. The dispersion of the $chip_{20}$ curves across the entire range illustrates that the random information leveraged by HELP is based on within-die variations (**WDV**), and not on global performance differences that occur from chip-to-chip.

The differences that remain in the PND_c are those introduced by WDV and *uncompensated* TV noise (**TVN**). The range of TVN for the bottom-most curve in Fig. 3(c) is labeled and is approx. 3, which translates to approx. 90 ps. In general, PND_c with larger amounts of TVN are more likely to introduce bit flip errors. Therefore, it is desirable to make TVN as small as possible, and is the main driver for using the TVComp process.

The last operation applied to the PN is represented by the *Modulus* operation shown on the right side of Fig. 1. Modulus is a standard mathematical operation that computes the positive remainder after dividing by the modulus. The Modulus operation is required by HELP to address the path length bias that exists in the PND_c, which acts to reduce randomness and uniqueness in the generated bitstrings. The value of the Modulus is also a configuration parameter, similar to the LFSR seeds, μ_{ref} and Rng_{ref} parameters, and is discussed further in the following. The term $\mathbf{modPND_{co}}$ is used to refer to the values used in the bitstring generation process.

C. Offset Method

An optional *offset* can also be applied to PND_c values prior to the application of the Modulus to further improve the statistical quality of the bitstrings. An offset is computed for each PND_c separately in a characterization process. The offset is simply the *median value* of the PND_c, derived using PN from a sample of chips or from a nominal simulation. The offsets are transmitted to the token and are therefore a second component of the challenges. The token adds the individual offsets to each of the PND_c as they are generated. The offset shifts the PND_c upwards and *centers the population* over the 0-1 line associated with the Modulus. We use the term $\mathbf{PND_{co}}$ to refer to the PND_c with offsets applied. Since the offset is a population-based value, it leaks no information regarding the bit values generated from the $modPND_{co}$ (to be discussed).

As an example, three randomly selected PND_c are shown in Fig. 4. The PND_c from the 500 chip-instances are given on the left in the same format as that used in Fig. 3(c), while the corresponding ‘shifted’ PND_{co} are shown to their immediate right. The 0-1 lines associated with a Modulus of 24 are superimposed as dashed horizontal lines. The Modulus creates vertical partitions of size 24, with 0-1 lines at Modulus/2 and Modulus. The corresponding bit assignments for each region are shown on the far right.

The shift amounts are shown between the two sets of waveforms. The centering of the population over the 0-1 lines ensures that nearly equal numbers of chips produce 0’s and 1’s for each of the corresponding PND_{co}. We restrict the offset encoding to 4 bits, making it possible to shift the population by Modulus/(2*16). The additional factor of 2 in the denominator accounts for the fact that the maximum shift required to reach one of the 0-1 lines is half the Modulus.

D. Margining

A *Margin* technique is used to improve reliability by identifying and excluding bits that have the highest probability of ‘flipping’ from 0 to 1 or 1 to 0. As an illustration, Fig. 5 plots 18 of the 2048 $modPND_{co}$ from chip C_1 along the x-axis. The red curve line-connects the data points corresponding to enrollment conditions while the black curves line-connects data points under the 12 regeneration TV corners. A set of margins are shown of size 2 surrounding two strong bit regions of size 8. Designators along the top given as ‘s0’, ‘s1’, ‘w0’ and ‘w1’ classify each of the enrollment data points as either a strong 0 or 1, or a weak 0 or 1, resp. Data points that fall on or within the hatched areas are classified as

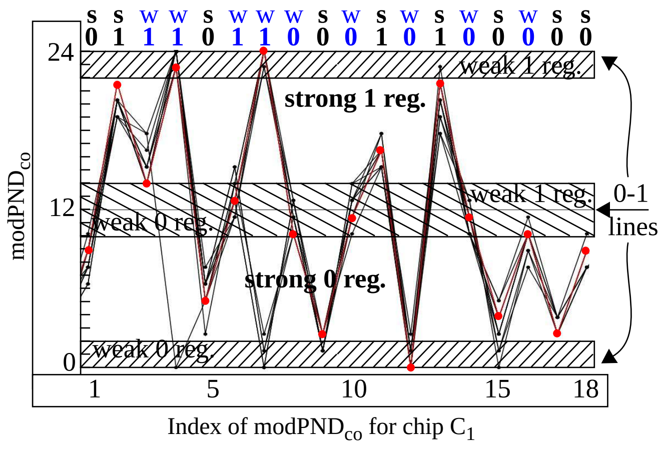


Fig. 5. Strong and weak bits associated with modPND$_{co}$ from chip C$_1$ using margining.

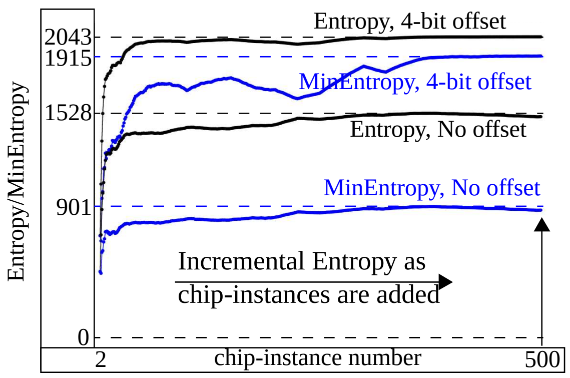


Fig. 6. Entropy (black) and MinEntropy (blue) change as chips are added to the analysis along the x-axis. Maximum value is 2048 bits. Top curves show result using 4-bit offset while lower curves show analysis with no offset using a Modulus of 24.

weak as a mechanism to avoid bit flip errors introduced by uncompensated TV noise (TVN) that occurs during regeneration.

The Margin method improves bitstring reproducibility by eliminating data points classified as 'weak' in the bitstring generation process. For example, the data points at indexes 4, 6, 7, 8, 10 and 14 would introduce bit flip errors at one or more of the TV corners during regeneration because at least one of the regeneration data points is in the opposite bit value region, i.e., they cross one of the annotated 0-1 lines, from the corresponding enrollment value. A *helper data* string is constructed during enrollment that records the strong/weak status of each modPND$_{co}$ which is used during regeneration to identify which modPND$_{co}$ generate bits (strong) and which are skipped (weak).

IV. Statistical Results

A. Entropy Analysis

The statistical analysis is carried out using the bitstrings generated from the 500 chip-instances. Entropy is defined by Eq. 3 and MinEntropy by Eq. 4. The frequency p_{ij} of '0's and '1's is computed at each bit position i across the 500 chip-instance bitstrings of size 2048 bits, i.e, no Margin is used in this analysis.

$$H(X) = \sum_{i=0}^{2047}\sum_{j=0}^{1} p_{ij}\log_2(p_{ij}) \qquad \textbf{Eq. 3.}$$

$$H_{\infty}(X) = \sum_{i=0}^{2047} -\log_2(max(p_{ij})) \qquad \textbf{Eq. 4.}$$

Fig. 6 plots incremental Entropy and MinEntropy for both the original modPND$_{co}$ and the 4-bit offset technique using black and blue curves, resp, as chip-instances are added, one at a time, to the analysis (this method is inspired from the technique described in [34]). The x-axis gives the index of the chip-instance starting with 2 chip-instances on the left and ending with 500 chip-instances on the right. The 4-bit offset technique shifts and centers the population of chip-instances associated with each modPND$_c$ over a 0-1 line as discussed in Section III.C. The centering has a significant impact on Entropy and MinEntropy which is reflected in the larger values and the gradual approach of the curves to the ideal value of 2048 as chip-instances are added.

Fig. 7(a) and (b) depict bar graphs of Entropy and MinEntropy for Moduli 10 through 30 (x-axis). The height of the bars represent the average values computed using the 2048-bit bitstrings from 500 chip-instances, averaged across 10 separate LFSR seed pairs. Entropy varies from 2037 to 2043, and is close to the ideal value of 2048 independent of the Moduli. MinEntropy varies between 1862 at Moduli 12 up to 1919, which indicates that, in the worst case, each bit contributes between 91% and 93.7% bits of Entropy.

B. Uniqueness

The InterChip hamming distance (InterChipHD) results are shown in Fig. 7(c), again computed using the bitstrings from 500 chip-instances, averaged across 10 separate LFSR seed pairs. Hamming distance is computed between all possible pairings of bitstrings, i.e., 500*499/2 = 124,750 pairings for each seed and then averaged.

The values for a set of Margins of size 2 through 4 (y-axis) are shown for each of the Moduli. Fig. 8 provides an illustration of the process used for dealing with weak and strong bits under HELP's Margin scheme in the InterchipHD calculation. The helper data bitstrings HelpD and raw bitstrings BitStr for two chips C$_x$ and C$_y$ are shown along the top and bottom of the figure, resp. The HelpD bitstrings classify the corresponding raw bit as weak using a '0' and as strong using a '1'. The InterchipHD is computed by XOR'ing only those BitStr bits from the C$_x$ and C$_y$ that have

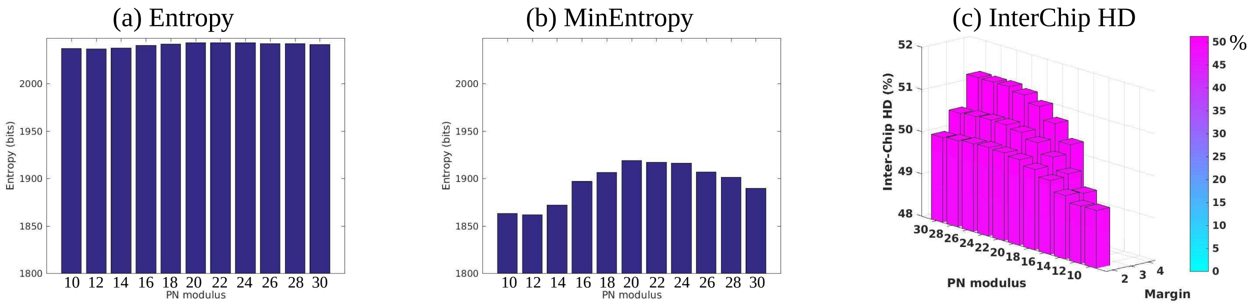


Fig. 7. (a) Entropy, (b) MinEntropy and (c) InterChip hamming distance (HD) computed as average values across 10 seeds of 2048 bits each using 500 chip-instances with TVCOMP reference set to 'Mean' scaling. Bars of zero height for InterChip HD are invalid combinations of Margin and Modulus. Entropy varies over the range 2037 to 2043, and MinEntropy from 1862 to 1919, with 2048 as the ideal value. InterChip HD varies from 49.4% to 51.2% with ideal at 50%.

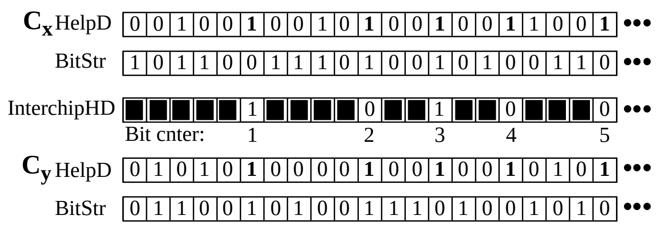


Fig. 8. Hamming distance illustration for results shown in Fig. 7.

BOTH HelpD bits set to '1', i.e., both raw bits are classified as strong. This process maintains alignment in the two bitstrings and ensures the same modPNDc from C_x and C_y are being used in the InterchipHD calculation.

InterChip HD, HD_{Inter}, is computed using Eq. 5. The

$$HD_{inter} = \left(\frac{1}{NCC} \sum_{i=1}^{NC} \sum_{j=i+1}^{NC} \frac{\left(\sum_{k=1}^{NB_a} (BS_{i,k} \oplus BS_{j,k}) \right)}{NB_a} \right) \times 100 \qquad \textbf{Eq. 5.}$$

symbols *NC*, NB_a and *NCC* represent 'number of chips', 'number of bits' and 'number of chip combinations', resp. (NCC is 124,750 as indicated above) This equation simply sums all the bitwise differences between each of the possible pairing of chip-instance bitstrings *BS* as described above and then converts the sum into a percentage by dividing by the total number of bits that were examined. *Bit cnter* from the center of Fig. 8 counts the number of bits that are used for NB_a in Eq. 5, which varies for each pairing of chip-instances *a*. The HD_{Inter} is computed separately for each of the 10 seed pairs and the average value is given in Fig. 7(c). The HD_{inter} vary from 49.4% to 51.2% and therefore are close to the ideal value of 50%.

C. NIST Test Evaluation

The NIST statistical test suite is used to evaluate randomness of the bitstrings [36]. The bitstrings are constructed as described above for Interchip HD. All tests are passed with at least 488 bitstrings passing of the 500 bitstrings as required by NIST except for CummulativeSums (NIST test #4) under two Moduli. The two failing cases failed with 487 and 482 bitstrings passing, resp., so the failures were only by at most 6 chips in the worst case.

V. Correlation Analysis

Correlation analysis measures whether a relationship exists between $modPND_{co}$ in which the bit response from one allows the response from a second to be predicted with probability greater than 50%. All strong PUF architectures to date have potential to exhibit correlation because the 2^n response bits are generated from a much smaller set of *m* components, with the *m* components representing the underlying random variables. For the case of a 64-stage Arbiter PUF, the 256 path segments are all reused in every challenge, and therefore, the potential for correlation introduced by *path segment reuse* is very high. HELP also reuses path segments but, unlike the Arbiter, the probability of two paths sharing a large number of path segments is very small. The following analysis focuses on the reuse of path segments within HELP despite the fact that, in practice, it is statistically rare.

Our correlation analysis of path segment reuse (called ***Partial Reuse***) is carried out using a set of 'unique' paths, and therefore, it ensures that at least one path segment is different in any pairing of PN used to create PND, PND_c, PND_{co} and $modPND_c$. (Note: we refer to PND_c in the following because the analysis focuses on how the Offset and Modulus operations affect the results). An example of partial reuse is shown in Fig. 9. The highlighted red wire on the left indicates that the two paths, labeled 'path #1' and 'path #2', share all of the initial path segments, and are only different at the fanout point where they diverge into LUT_a and LUT_b. The two paths then reconverge at the next gate and form a 'bubble' structure.

It is also possible to pair the **same** PN in different combinations to produce a much larger set of PND_c (on order of n^2 with *n* PN). We refer to this as ***Full Reuse***. Full path reuse

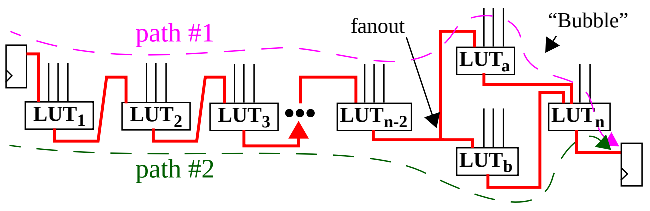


Fig. 9. Partial reuse worst-case example of two paths forming a 'bubble'. The path segments which define the bubble are unique to each path while the remaining components are common to both paths.

can result in *dependent bits*, i.e, bits that are completely determined by other bits. Reference [26] investigates these dependencies in the RO PUF and proposes schemes designed to eliminate and/or reduce the number of dependent bits.

We show in the following that the **Offset and Modulus operations break the correlations found in classic dependency analysis** typically exemplified using RO frequencies as f(RO$_A$) > f(RO$_B$) and f(RO$_B$) > f(RO$_C$) implies f(RO$_A$) > f(RO$_C$). Therefore, partial reuse and full reuse of paths have a smaller penalty in terms of Entropy and minEntropy when they occur within HELP.

A. *Preliminaries*

As indicated earlier, the HELP algorithm creates differences (PND) between PNR and PNF using a pair of LFSR seeds, which are then compensated using TVComp to produce PND$_c$. A key objective of our analysis is to purposely create worst case conditions for correlations by crafting the PND such that *partial reuse* and *full reuse* test cases are created. The analysis of correlations requires the set of PND that are constructed to be adjacent to each other in the arrays on which the analysis is performed. Therefore, the LFSRs used in the HELP algorithm are not used to create the PND and instead a linear, sequential pairing strategy is used.

The Offset and Modulus operations in the HELP algorithm are the key components to improving Entropy. As an aid to help with the discussion that follows, Fig. 10 illustrates how these two operators modify the PND$_c$. The figure shows 4 groups of 10 vertical line graphs, with each line graph containing 500 PND$_c$ data points corresponding to the 500 chip-instances. The line graph on the left and bottom illustrates that the vertical spread in the line-connected points is caused by within-die delay variations.

The *Reference PND$_c$* shown on the left are the compensated differences before the Offset and Modulus operations are applied. The DC bias introduced by differences in the lengths of the paths changes the vertical positions of the line graphs, which spans a range from -72 to +40 launch-capture intervals (LCIs)[1]. The Offset and Modulus operations are

1. Recall that 1 LCI = 18 ps, and represents the phase adjustment resolution of the Xilinx DCM.

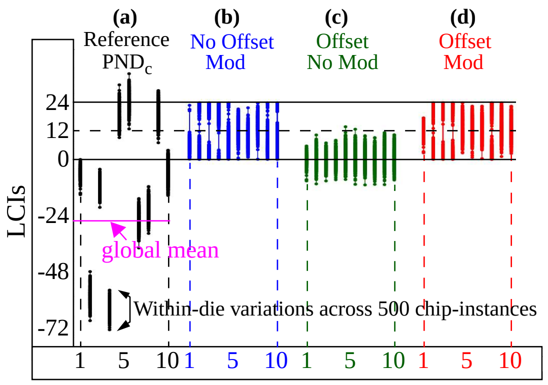


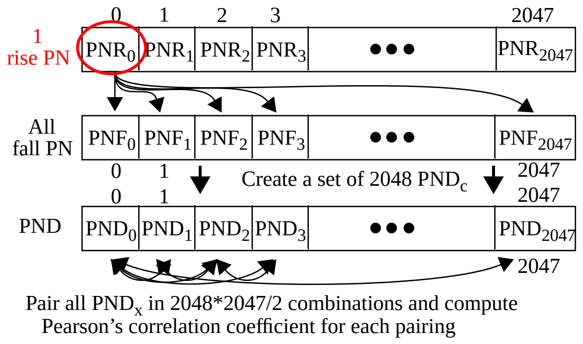
Fig. 10. A sample of 10 PND$_c$ from 500 chip-instances illustrating four experimental scenarios. (a) *Reference* represents PND$_c$ that have no Offset or Modulus applied. Scenarios (b) *No Offset, Mod* (c) *Offset, No Mod* and (d) *Offset, Mod* show how the PND$_c$ change as each of these operatons are applied individually or in combination.

designed to increase the Entropy in the PND$_c$ by eliminating this bias. For example, the *No Offset, Mod* group show the PND$_c$ from the *Reference PND$_c$* group after a Modulus of 24 is applied. Similarly, the *Offset, No Mod* group show the *Reference PND$_c$* after subtracting the median value from each line graph, which effectively centers the populations of 500 PND$_{co}$ over the 0 horizontal line. Finally, the *Offset, Mod* group shows the PND$_c$ with both operations applied, and represents the values used in the HELP algorithm. Here, an Offset is first applied to center the populations over the closest multiple of 12 and then a modulus of 24 is applied (the boundaries used to separate the '0' and '1' bit values are 12 and 24 for a Modulus of 24, see Fig. 5). We analyze the change in Entropy and minEntropy as each of the operations are applied. Note that HELP processes 2048 PND$_c$ at a time during bitstring generation, of which only 10 are shown in Fig. 10.

B. *Partial Reuse*

Although we defined *path segment reuse* above as a pair of paths with at least one path segment that is different for a given PND$_c$, we do not want to restrict our analysis to these types of specific physical characteristics but instead want to analyze the actual worst case. Xilinx Vivado implementation view does not provide information that directly reflects the chip layout, and therefore, a broader approach to correlation analysis is required to ensure the worst case correlations are identified.

We use Pearson's correlation coefficient (PCC) [35] to measure the degree of correlation that exist among PND$_c$ and then select a subset of the most highly correlated for Entropy and minEntropy analyses. Fig. 11 depicts the construction process used to create an exhaustive set of PND$_c$, from which the most highly correlated are identified. In order to simplify the construction process, the TVComp operation is applied to a set of 2048 PNR and 2048 PNF separately for



0 1 2 3 2047
1 rise PN
PNR0 PNR1 PNR2 PNR3 ••• PNR2047
All fall PN
PNF0 PNF1 PNF2 PNF3 ••• PNF2047
0 1 2047
0 1 2047
Create a set of 2048 PNDc
PND
PND0 PND1 PND2 PND3 ••• PND2047
2047
Pair all PNDx in 2048*2047/2 combinations and compute Pearson's correlation coefficient for each pairing

Fig. 11. PND pairing creation process for *partial reuse* analysis using Pearson's correlation coefficient. Note: all PNR and PNF are TVComp'ed but subscript 'c' is removed for clarity.

each of the 500 chip-instances[1]. Note the 'c' subscript is not used in the PNR/PNF designation for clarity. TVComp eliminates chip-to-chip delay variations and makes it possible to compare data from all chips directly in the following analysis.

Only one of the PNR, PNR_0, is used to create a set of 2048 PND_c by pairing it as shown with each of the PNF. Correlations that occur in the generated bitstring are rooted in correlations among the PND_c. Therefore, the 2048 PND_c are themselves paired, this time with each other under all combinations for 2048*2047/2 = 2,096,128 pairing combinations. The same process is carried out using the first PNF, PNF_0, with all of the PNR (not shown) to create a second set of PND_c, which are again paired under all combinations. We use only one rising reference PN, PNR_0, and one falling reference PN, PNF_0, because the value of the PCC is identical for other choices of these references.

For each of the 2 million+ PND_c pairings, the Pearson correlation coefficient (PCC) given by Eq. 6 is computed using enrollment data from the 500 chip-instances. PCC can vary from highly correlated (-1.0 and 1.0) to no correlation (0.0). The absolute value of the PCC in each group of 2 million+ rising and falling PND_c are then sorted from high to low. Scatterplots of the most highly and least correlated PND_c pairings are shown in Fig. 12 from the larger set of more than 4 million pairings. The most highly correlated 1024 PND_c pairings (for a total of 2048 PND_c since each pairing contains two PND_c) are used in the bitstring genera-

$$PCC = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\left[\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2\right]^{1/2}} \quad \text{where } -1 \le PCC \le 1 \qquad \textbf{Eq. 6.}$$

1. HELP normally applies TVComp only once, and to the PND as discussed in Section III.B, for processing efficiency reasons but the results using either method are nearly identical.

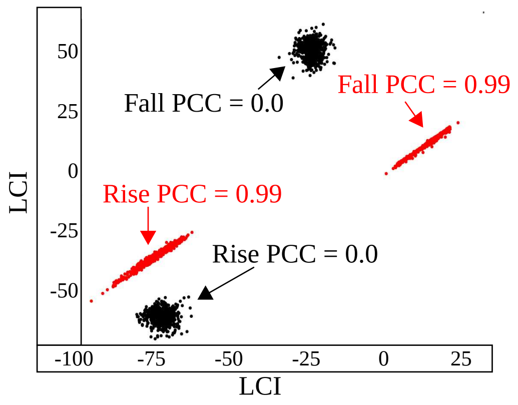


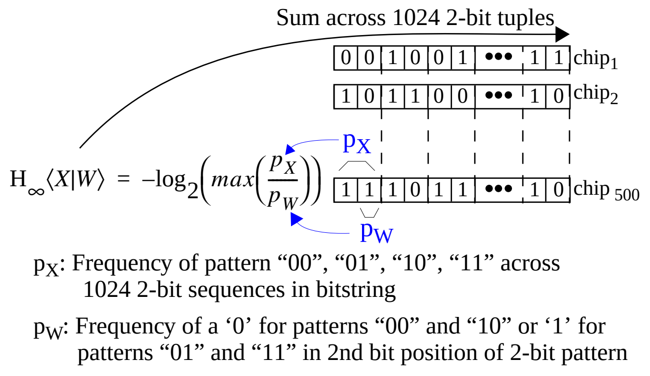
Fig. 12. Scatterplot showing the most highly correlated and least correlated rising and falling PND_c under the partial reuse analysis.

tion process for the Entropy and Conditional minEntropy (**CmE**) evaluation below. Highly correlated PND_c are stored as adjacent values to facilitate analysis of the corresponding 2-bit sequences.

The 2048 PND_c are processed into bitstrings under four different scenarios as shown in Fig. 10. For example, the PND_c are compared to a *global mean* under the ***Reference*** scenario (see annotation in figure). The global mean is the average PND_c across all chip-instances and all 2048 PND_c (500 * 2048). A '0' is assigned to the bitstring for cases in which the PND_c for a chip-instance falls below the *global mean* and a '1' otherwise. Given the large DC bias associated with the PND_c under the *Reference* scenario, the Entropy and CmE statistics are expected to be very poor.

The ***No Offset, Mod*** and ***Offset, Mod*** bitstring generation scenarios use the value 12 as the boundary between '0' and '1' (for Modulus 24 as shown in the figure), i.e., PND_c >= 0 and < 12 produce a '0' and those >= 12 and < 24 produce a '1'. The '0'-'1' boundary for the ***Offset, No Mod*** scenario is 0 and the sign bit is used to assign '0' (for negative PND_c) and '1' (for positive PND_c). The ***Offset, Mod*** scenario represents the operations performed by the HELP algorithm. The analysis is extended for this scenario by evaluating Entropy and CmE over Moduli between 14 and 30 to fully illustrate the impact of the Modulus operation.

The PND_c from a *normal* use case are also analyzed using these four bitstring generation scenarios to determine how much Entropy/CmE is lost when compared to the *highly correlated* case analysis. For the normal use case, no attempt is made to correlate PND_c and instead random pairings of PNR and PNF are used to construct the PND_c. Table 1 pro-



Sum across 1024 2-bit tuples
chip1
chip2
chip 500
pX
pW
H∞⟨X|W⟩ = −log2(max(pX/pW))
pX: Frequency of pattern "00", "01", "10", "11" across 1024 2-bit sequences in bitstring
pW: Frequency of a '0' for patterns "00" and "10" or '1' for patterns "01" and "11" in 2nd bit position of 2-bit pattern

Fig. 13. Conditional minEntropy (CmE) expression and illustration of its application.

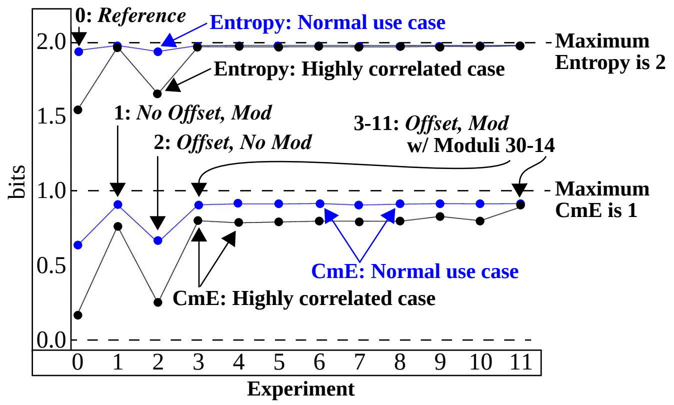


Fig. 14. Entropy and Conditional minEntropy results under various scenarios (Fig. 10) and cases as given in Table 1.

vides a summary of the 8 scenarios investigated.

Table 1: Summary of Scenarios for Partial Reuse Analysis.

Cases	Scenarios			
Highly Correlated	No Offset No Modulus (Reference)	No Offset Modulus	Offset No Modulus	Offset Modulus (HELP)
Normal use				

Fig. 13 provides a graphic that depicts the process used to compute Entropy and Conditional minEntropy (**CmE**), (modeled after the technique proposed in [32]). As indicated earlier, highly correlated PND_c and the corresponding bits that they generate are kept in adjacent positions in the array. The bitstrings are of length 2048. Therefore, each chip-instance provides 1024 sets of 2-bit sequences.

Eq. 7 is used to compute the Entropy of the 1024 2-bit sequences for each chip-instance, which is then divided by 1024 to convert into Entropy per bit. The p_i represent the frequencies of the four 2-bit patterns as given in Fig. 13. The Entropy per bit value reported below is the average of 500 chip-instance values. CmE is computed using Eq. 8 (also

$$H(X) = -\sum_{i=0}^{3} p_i \log_2(p_i) \qquad \textbf{Eq. 7.}$$

$$H_\infty\langle X|W\rangle = -\log_2\left(max\left(\frac{p_X}{p_W}\right)\right) \qquad \textbf{Eq. 8.}$$

from [32]). The expression $max(p_X/p_W)$ represents the maximum conditional probability among the four values computed for each 2-bit sequence. Again, the sum over the 1024 2-bit sequences is converted to CmE per bit for each chip-instance and the average across all 500 chip-instances is reported.

The Entropy and CmE results are plotted in Fig. 14 for both the highly correlated and normal use cases. The x-axis represents the experiment, with 0 plotting the results using the ***Reference*** bitstring generation scenario (from Fig. 10), 1 representing the ***No Offset, Mod***, 2 representing ***Offset, No Mod*** and 3 through 11 representing the ***Offset, Mod*** scenario for Moduli between 30 and 14, resp. The maximum Entropy

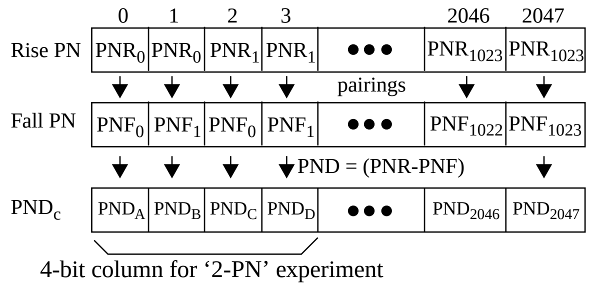


Fig. 15. PND_c construction process for full reuse analysis, called 2-PN. Every column of 4-bits, with first one labeled PND_A through PND_D, are correlated because the same two PNR and PNF are subtracted under all combinations to create PND_c.

per bit is 2 while the maximum CmE is 1. From the trends, it is clear that both Offset and Modulus improve the statistical quality of the bitstrings over the Reference. However, Modulus appears to provide the biggest benefit, which is captured by the drops in Entropy and CmE for experiment 2 in which the Modulus is not applied. Moreover, the loss in Entropy is almost zero between the normal use and highly correlated cases and CmE drops on average by only 0.2 bits for experiments 3 through 11 for the *Offset, Mod* scenario. Therefore, partial reuse under highly correlated conditions introduces only a small penalty on the quality of the bitstrings generated by the HELP algorithm.

C. *Full Reuse*

Full reuse refers to the **repeated use of the PN** in multiple PND_c as shown for the 2-PN reuse example in Fig. 15. Here, two rise PN, PNR_0 and PNR_1 are paired in all combinations with two fall PN, PNF_0 and PNF_1. A traditional analysis predicts that because of correlation, only a subset of the 16 possible bit patterns can be generated when using PND_A through PND_D to produce a 4-bit response. In particular, patterns "0110" and "1001" are not possible. However, as indicated earlier, the Modulus and Offset operations break the classical dependencies and allow all patterns to be generated,

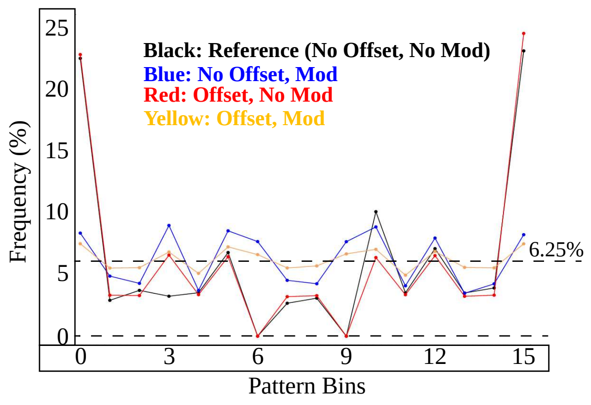


Fig. 16. Frequency of 4-bit patterns for bin 0 with pattern "0000" through bin 15 with pattern "1111" under 2-PN reuse case and four scenarios from Fig. 10. Ideal frequency value is 1/16 = 6.25%. *Reference* PND_c exhibits the worst case behavior with frequencies of 0% for patterns "0110" and "1001", while *Offset, Mod* exhibits the best behavior.

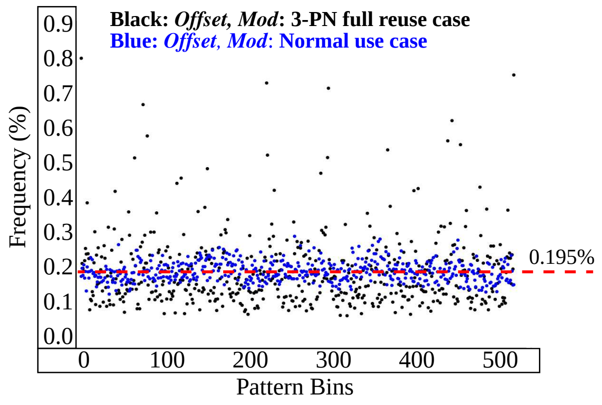


Fig. 17. Frequency of 9-bit patterns for bin 0 with pattern "000000000" through bin 512 with pattern "111111111" under 3-PN reuse case (black) and normal use case (blue). The distribution should be uniform with each bin percentage at 1/512 = 0.195% as shown by the dotted line.

as we show below.

The frequency of the 16 patterns for the 2-PN experiment are shown in Fig. 16. Here, PND_c are created for each of the 500 chip-instances according to the illustration in Fig. 15. With 2048 bits per chip-instance, there are 512 4-bit columns each with 500 instances. The graph simply plots the percentage of each pattern across this set of 500*512 = 256,000 samples for each of the PND_c scenarios described earlier with reference to Fig. 10. The ideal distribution is uniform with the percentage equal to 1/16*100 = 6.25% for each 'Pattern Bin' along the x-axis.

The distributions associated with the ***Reference*** (black) and ***Offset, No Mod*** (red) experiments are clearly not uniform. Pattern bins 6 and 9 are zero for *Reference*, as predicted by the classical dependency analysis. Although the differences are small, the ***Offset, No Mod*** distribution is slightly better with non-zero values in pattern bins 6 and 9 and most of the other pattern bins closer to the ideal value of 6.25%. The Modulus operation, particularly in combination with the Offset operation, produce much better results. The percentages for the ***Offset, Mod*** experiment (yellow curve) vary by at most 1.2% from the ideal value of 6.25%.

The positive impact of the Offset and Modulus operations on Entropy is further supported by an analysis carried out in a 3-PN experiment, where 3 rise and 3 fall PN are combined under all combinations to produce a 9-bit column (analogous to 2-PN illustration in Fig. 15). With 9-bit columns, there are 512 possible pattern bins. Using the 2048 bitstrings from 500 chip-instances, we were able to construct 227 full 9-bit columns (left over columns were discarded), for a total sample size of 113,500. A scatterplot showing the results for the 3-PN experiment is given in Fig. 17 using ***Offset, Mod*** PND_c bitstring data (black dots). The ideal percentage is 1/512*100 = 0.195%. As a reference, the results using PND_c constructed without reusing any rising or falling PN (referred to as the normal use case above) are superimposed in blue. The smaller variation of the frequencies under the normal use case, when compared with the 3-PN full reuse case, clearly shows that there is a penalty associated with reuse, but none of pattern bins are empty and most of the frequency values are within 0.1% of the ideal value at 0.195%.

Table 2 presents the minEntropy computed using Eq. 4 for each of the PND_c scenarios (rows) for the 2-PN and 3-PN experiments described above, and an additional 4-PN experiment. For the 4-PN experiments, all combinations of 4 PN are used and the frequency of the 65536 possible patterns in the set of 128 16-bit columns are analyzed. The corresponding minEntropy values under the normal use case (with column labeled 'Normal') are also given for reference.

In all cases except for row 3, column 2, the minEntropy values in the last row are larger than those in the first 3 rows. Moreover, the drop in minEntropy over the normal use case in the last row is 0.19, 1.45 and 1.9 bits, resp., illustrating the penalty associated with reuse is very modest.

Table 2: minEntropy for PND_c and x-PN experiments.

	2-PN	Normal	3-PN	Normal	4-PN	Normal
No Offset, No Mod	2.11 of 4	3.05 of 4	3.17 of 9	6.15 of 9	4.3 of 16	7.0 of 16
No Offset, Mod	3.92 of 4	3.81 of 4	6.10 of 9	7.89 of 9	8.3 of 16	10.2 of 16
Offset, No Mod	2.02 of 4	2.82 of 4	3.05 of 9	5.34 of 9	4.1 of 16	8.5 of 16
Offset, Mod	**3.73 of 4**	**3.92 of 4**	**6.95 of 9**	**8.40 of 9**	**9.2 of 16**	**11.1 of 16**

VI. Conclusions

An analysis of the statistical characteristics of a Hardware-Embedded Delay PUF (HELP) are presented in this paper, with emphasis on Interchip Hamming Distance, Entropy, minEntropy, conditional minEntropy and NIST statistical test results. The bitstrings generated by the HELP algorithm are shown to exhibit excellent statistical quality.

An experiment focused on purposely constructing worst case correlations among path delays is also described as a means of demonstrating the Entropy-enhancing benefit of the Offset and Modulus operations carried out by the HELP algorithm. Special data sets are constructed which maximize physical correlations and dependencies introduced by reusing components of the underlying Entropy. Although statistical quality is reduced under these worst case conditions, the reduction is modest. Therefore, the Modulus and Offset operations harden the HELP algorithm against model-building attacks.

A quantitative analysis of the relationship between Entropy as presented in this paper and the level of effort required to carry out model-building attacks on HELP is the subject of a future work. Developing a formal quantitative framework that expresses the relationship between Entropy and model-building effort is inherently difficult because of the vastly different mathematical domains on which each is based. Best practice relating Entropy to security properties that predict attack resilience is focused on correlating results from separate analyses of Entropy and model-building resistance. A thorough treatment of model-building resistance requires a wide range of machine-learning experiments. Work on this topic is on-going and will be reported in a separate paper in the near future.

VII. References

[1] W. Che, M. Martin, G. Pocklassery, V. K. Kajuluri, F. Saqib and J. Plusquellic, “A Privacy-Preserving, Mutual PUF-Based Authentication Protocol”, *Cryptography*, Nov. 2017.

[2] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, “Controlled Physical Random Functions”, *Conference on Computer Security Applications*, 2002.

[3] B. Gassend and D. E. Clarke and M. van Dijk, S. Devadas, “Silicon Physical Unknown Functions”, *Conference on Computer and Communications Security*, 2002, 148-160.

[4] K. Lofstrom, W. R. Daasch, D. Taylor, “Identification Circuits using Device Mismatch”, *International Solid State Circuits Conference*, 2000, pp. 372-373.

[5] A. Maiti, P. Schaumont, "Improving the quality of a Physical Unclonable Function using Configurable Ring Oscillators", *FPLA*, 2009.

[6] Y. Meng-Day, R. Sowell, A. Singh, D. M’Raihi, S. Devadas, “Performance Metrics and Empirical Results of a PUF Cryptographic Key Generation ASIC”, *Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 108-115.

[7] E. Simpson and P. Schaumont, “Offline Hardware/Software Authentication for Reconfigurable Platforms”, *Cryptographic Hardware and Embedded Systems*, Volume 4249, Oct., 2006, pp. 10-13.

[8] B. Habib, K. Gaj and J.-P. Kaps, “FPGA PUF Based on Programmable LUT Delays”, *Euromicro Conference on Digital System Design*, 2013, pp. 697 -704.

[9] J. Guajardo, S. S. Kumar and G. Schrijen and P. Tuyls, “Brand and IP Protection with Physical Unclonable Functions”, *Symposium on Circuits and Systems*, 2008, pp. 3186-3189.

[10] Y. Alkabani and F. Koushanfar and N. Kiyavash and M. Potkonjak, “Trusted Integrated Circuits: A Nondestructive Hidden Characteristics Extraction Approach”, *Information Hiding*, 2008.

[11] R. Helinski, D. Acharyya, J. Plusquellic, “Physical Unclonable Function Defined Using Power Distribution System Equivalent Resistance Variations”, *Design Automation Conference*, 2009, pp. 676-681.

[12] R. Chakraborty, C. Lamech, D. Acharyya and J. Plusquellic, “A Transmission Gate Physical Unclonable Function and On-Chip Voltage-to-Digital Conversion Technique”, *Design Automation Conference*, 2013, pp. 1-10.

[13] J. Aarestad, J. Plusquellic, D. Acharyya, “Error-Tolerant Bit Generation Techniques for Use with a Hardware-Embedded Path Delay PUF,” Symposium on Hardware-Oriented Security and Trust (HOST), 2013, pp. 151-158.

[14] F. Saqib, M. Areno, J. Aarestad and J. Plusquellic, “An ASIC Implementation of a Hardware-Embedded Physical Unclonable Function”, *IET Computers & Digital Techniques*, Vol. 8, Issue 6, Nov. 2014, pp. 288-299.

[15] W. Che, F. Saqib, J. Plusquellic, "PUF-Based Authentication," ICCAD, Nov, 2015.

[16] G. S. Rose, N. McDonald, Y. Lok-Kwong, B. Wysocki and K. Xu, “Foundations of Memristor Based PUF Architectures”, *International Symposium on Nanoscale Architectures*, 2013, pp. 52-57.

[17] Z. Yu, A. R. Krishna and S. Bhunia, “ScanPUF: Robust Ultralow-Overhead PUF using Scan Chain”, *Asia and South Pacific Design Automation Conference*, 2013, pp. 626-631.

[18] S. T. C. Konigsmark, L. K. Hwang, C. Deming, M. D. F. Wong, “CNPUF: A Carbon Nanotubebased Physically Unclonable Function for Secure Low-Energy Hardware Design”, *Asia and South Pacific Design Automation Conference*, 2014, pp. 73-78.

[19] M. Majzoobi, F. Koushanfar, S. Devadas, “FPGA PUF using Programmable Delay Lines”, *Workshop on Information Forensics and Security*, 2010, pp.1-6

[20] Y. Hori, T. Yoshida, T. Katashita, A. Satoh, “Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs”, *Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 298-303.

[21] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas, “Identification and Authentication of Integrated Circuits”, *Concurrency Comput., Practice Exp.*, Vol. 16, No. 11, pp. 1077-1098, 2004.

[22] M. Majzoobi, F. Koushanfar, S. Devadas, “FPGA PUF using Programmable Delay Lines”, *Workshop on Information Forensics and Security*, 2010, pp. 1 -6.

[23] X. Xin, J. Kaps, K. Gaj, “A Configurable Ring-Oscillator-Based PUF for Xilinx FPGAs”, Conference on Digital System Design, 2011, pp. 651-657.

[24] E. Suh, S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation”, *Design Automation Conference*, 2007, June 4-8, 2007, pp. 9-14.

[25] A. Maiti, K. Inyoung, P. Schaumont, “A Robust Physical Unclonable Function With Enhanced Challenge-Response Set”, *Trans. on Information Forensics and Security*, Volume: 7, Issue: 1, Part: 2, 2012, pp. 333-345.

[26] Chi-En, Daniel Yin, Gang Qu, “LISA: Maximizing RO PUF’s Secret Extraction”, *Sym. Hardware-Oriented Security and Trust*, 2010, PP 100-105.

[27] Chi-En Yin and Gang Qu, “Improving PUF Security with Regression-based Distiller”, *Design Automation Conference*, 2013.

[28] J. Delvaux, D. Gu, D. Schellekens, I. Verbauwhede, “Helper Data Algorithms for PUF-based key generation: Overview and analysis”, *Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34, No. 6, p 889-902, 2015.

[29] C. Bosch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls. “Efficient Helper Data Key Extractor on FPGAs”, *Workshop on Cryptographic Hardware and Embedded Systems*, Vol. 5154 of LNCS, pp. 181-197, 2008.

[30] P. Koeberl, J. Li, A. Rajan and W. Wu, “Entropy Loss in PUF-based Key Generation Schemes: The Repetition Code Pitfall”, *Sym. on Hardware-Oriented Security and Trust*, 2014, PP 44-49.

[31] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and

Other Noisy Data", *SIAM Journal on Computing*, 38(1): 97-139, 2008.

[32] S. Katzenbeisser1, Ü. Kocabas, V. Rozic, A.-R. Sadeghi, I. Verbauwhede, C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon", CHES, 2012.

[33] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," *DATE*, 2004, pp. 246-251.

[34] M. Claes, V. van der Leest and A. Braeken, "Comparison of SRAM and FF PUF in 65nm Technology", *Nordic Conference on Secure IT Systems*, 2011, pp. 47-64.

[35] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[36] http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html