Leveraging Distributions in Physical Unclonable Functions

W. Che, V. K. Kajuluri, F. Saqib* and J. Plusquellic, UNM and *UNCC

Abstract

A special class of Physical Unclonable Functions (PUFs) referred to as strong PUFs can be used in novel hardware-based authentication protocols. Strong PUFs are required for authentication because the bitstrings and helper data are transmitted openly by the token to the verifier and therefore, are revealed to the adversary. This enables the adversary to carry out attacks against the token by systematically applying challenges and obtaining responses in an attempt to machine-learn and later predict the token's response to an arbitrary challenge. Therefore, strong PUFs must both provide an exponentially large challenge space and be resistant to machine learning attacks in order to be considered secure. We investigate a transformation called TVCOMP used within the Hardware-Embedded deLay PUF (HELP) bitstring generation algorithm that increases the diversity and unpredictability of the challenge-response space and therefore increases resistance to model-building attacks. HELP leverages within-die variations in path delays as a source of random information. TVCOMP is a linear transformation designed specifically for dealing with changes in delay introduced by adverse temperature-voltage (environmental) variations. In this paper, we show that TVCOMP also increases entropy and expands the challenge-response space dramatically.

1 Introduction

A physical unclonable function (PUF) is a next-generation hardware security primitive. Security protocols such as authentication and encryption can leverage the random bitstring and key generation capabilities of PUFs as a means of hardening vulnerable mobile and embedded devices against adversarial attacks. Authentication is a process that is carried out between a hardware token (smart card) and a verifier (a secure server at a bank) that is designed to confirm the identities of one or both parties [1]. With IoT, there are a growing number of authentication applications in which the hardware token is resource-constrained. Conventional methods of authentication which use area-heavy cryptographic primitives and non-volatile memory (NVM) are less attractive for these types of evolving embedded applications [2]. PUFs, on the other hand, can address issues related to low cost because they can potentially eliminate the need for NVM. Moreover, the special class of *strong PUFs* can further reduce area and energy overheads by eliminating cryptographic primitives that would otherwise be required.

A PUF measures parameters that are random and unique on each IC, as a means of generating digital secrets (bitstrings). The bitstrings are generated in real time, and are reproducible under a range of environmental variations. The elimination of NVM for key storage and the tamper evident property of PUFs to invasive probing attacks represent significant benefits for authentication applications in resource-constrained environments.

Many existing PUF architectures utilize a dedicated on-chip array of identically-designed elements. The parameters measured from the individual elements of the array are compared to produce a finite number of challenge-response-pairs (CRPs). When the number of challenges is polynomial in size, the PUF is classified as *weak*. Weak PUFs require secure hash and/or other types of cryptographic functions to obfuscate the challenges, the responses or both when used in authentication applications. In contrast, the number of challenges is exponential for a *strong PUFs*, making exhaustive readout of the CRP space impractical. However, in order to be secure, a truly strong PUF must also be resilient to machine learning algorithms, which attempt to use a subset of the CRP space to build a predictive model.

The hardware-embedded Delay PUF (HELP) analyzed in this paper generates bitstrings from delay variations that occur along paths in an on-chip macro, such as the datapath component of the Advanced Encryption Standard (AES) algorithm. The HELP processing engine defines a set of *configuration* parameters which are used to transform the measured path delays into bitstring responses. One of these parameters, called the *Path-Select-Mask* provides a mechanism to choose k paths from n that are produced, which enables an exponential number of possibilities. However, resource-constrained versions of HELP typically restrict the number of paths to the range of 2^{20}. Therefore, the CRP space of HELP is not large enough to satisfy the conditions of a truly strong PUF unless mechanisms are provided by the HELP algorithm to securely and significantly expand the number of path delays that can be compared to produce bitstrings.

A key contribution of this work is an experimentally-derived proof of a claim that a component of the HELP algorithm called *Temperature-Voltage-Compensation* (**TVCOMP**) is capable of providing this expansion. TVCOMP is an operation carried out within the HELP bitstring generation process that is designed to calibrate for variations in path delays introduced by changes in environmental conditions. Therefore, the primary purpose of TVCOMP is unrelated to entropy, but rather is a method designed to improve reliability.

The HELP bitstring generation process begins by selecting a set of k paths, typically 4096, from a larger set of n paths that exist within the on-chip macro. A series of simple mathematical operations are then performed on the path delays. The TVCOMP operation is applied to the entire distribution of k path delays. It first computes the mean and range of the distribution and then applies a linear transformation that *standardizes* the path delays, i.e., subtracts the mean and divides each by the range, as a mechanism to eliminate any changes that occur in the delays because of adverse environmental conditions.

The standardized values therefore depend on the mean and range of the original k-path distribution. For example, a fixed path delay that is a member of two different distributions, with different mean and range values, will have different standardized values. This difference is preserved in the remaining steps of the bitstring generation process. Therefore, the bit generated for a fixed path delay can change from

0-to-1 or 1-to-0 depending on the mean and range of the distribution. We refer to this dependency between the bit value and the parameters of the distribution as the ***Distribution Effect***. Distribution Effect adds uncertainty for algorithms attempting to learn and predict unseen CRPs.

It is important to note that this type of diversity-enhancing CRP method is not applicable to PUFs built from identically-designed test structures, e.g., RO and Arbiter PUFs [3], because it is not possible to construct distributions with widely varying means and ranges. In other words, the distributions defined by sets of *k* RO frequencies measured from a larger set of *n* RO frequencies are nearly indistinguishable. The HELP PUF, on the other hand, measures paths which have significant differences in path delays and therefore, crafting a set of CRPs which generate distributions with distinct parameters is trivial to accomplish, as we demonstrate in this paper.

Although there are *n*-choose-*k* ways of creating a set of *k*-path distributions (an exponential), there are only a polynomial number of different integer-based means and ranges that characterize these distributions, and of these, an even smaller portion actually introduce changes in the bit value derived from a fixed path delay. Unfortunately, deriving a closed form expression for the level of CRP expansion is difficult at best, and in fact, may not be possible. Instead, an alternative empirical-based approach is taken in this paper to derive an estimate. We first demonstrate the existence of the Distribution Effect, and then evaluate the bitstring diversity introduced by Distribution Effect using Interchip Hamming distance.

Note that even though the increase in the CRP space is polynomial (we estimate conservatively that each path delay can produce approx. 100 different bit values), the real strength of the Distribution Effect is related to the real time processing requirements of attacks carried out using machine learning algorithms. With Distribution Effect, the machine learning algorithm needs to be able to construct an estimate of the actual *k*-path distribution. This in turn requires detailed information about the layout of the on-chip macro, and an algorithm that quickly decides which paths are being tested for the specific set of server-selected challenges used during an authentication operation. Moreover, the machine learning algorithm must produce a prediction in real time and only after the server transmits the entire set of challenges to the authenticating token. We believe these additional tasks will add significant difficulty to a successful impersonation attack.

The implications of the Distribution Effect are two-fold. First, HELP can leverage smaller functional units and still achieve an exponential number of challenge-response-pairs (CRPs) as required of a strong PUF. Second, the difficulty of model-building HELP using machine learning algorithms will be more difficult because the path delays from the physical model are no longer constant.

2 Related Work

Although references [4-7] describe previous research on HELP, no prior work exists that describes the Distribution Effect presented in this paper. We have found no related work that leverages the membership characteristics of a group of physical elements as a mechanism to increase bitstring diversity. Moreover, we have found no related work that demonstrates that the same fixed path delays for a chip can generate a different (stable) response simply by changing the set of challenges. The linear (analog) transformation applied to a selected group of elements in combination with a subsequent modulus operation has, so far, proven to be unlearnable by machine-learning algorithms including Deep Learning within Neural Network frameworks and AdaBoost. Unfortunately, the scope of our machine-learning evaluation is too large and complex to include as supporting evidence in this paper.

We also point out that the mathematical operations performed by the HELP algorithm have linear time and space complexity. Our failure to successfully machine-learn the bitstring responses produced by HELP indicate that complex challenge and/or response obfuscation methods, e.g., those proposed for other weak and strong PUFs which are based on secure hashes, are not needed. Secure-hash-based obfuscation techniques introduce considerable cost in time, area, energy and reliability, and are more expensive than the HELP module operations applied to a small set of path delays. Moreover, the bit-flip avoidance schemes proposed for HELP also have linear time complexity, in contrast with most, if not all, of the error correction schemes that have been proposed for other PUFs. Time and resource utilization of a typical implementation of HELP are reported in [6].

A method to estimate the “extractable” entropy in PUF-generated bitstrings is proposed in [8] by calculating the mutual information between the bias measurements done at enrollment and regeneration. The authors in [9] evaluate the robustness and unpredictability of five different PUFs (including Arbiter, RO, SRAM, flip-flop and latch PUFs) by estimating the entropy from the available responses. The authors in [10] proposed an S-ArbRO PUF where only a subset of *k* RO pairs (out of *N*) contributes to the final delay difference. The technique proposed in this paper is unique and novel among published work related to this topic.

3 HELP Overview

A combinational logic circuit is used as the source of Entropy for HELP. The left side of Fig. 1 shows sequences of logic gates that define several paths within a typical logic circuit (which is also referred to as the functional unit). Unlike other proposed PUF structures, the functional unit used by HELP is an arbitrary, tool-synthesized netlist of gates and wires, as opposed to a carefully structured physical layout of identically-designed test structures such as ring oscillators. In this paper, the combinational logic that defines a 32-bit column from the Advanced Encryption Standard (AES) algorithm, subsequently referred to as **sbox-mixedcol**, is synthesized using Xilinx Vivado to a bitstream for programming an FPGA [11]. *sbox-mixedcol* is implemented using a hazard-free logic style called WDDL [12]. WDDL transforms the netlist from the original 32-bit design into true and complementary netlists. A complementary set of 32-bit primary inputs (PIs) and primary outputs (POs) are added to the design, doubling the input/output width to 64-bits. Structural analysis reveals that approx. 8 million paths exist within the 2,900 LUTs and 30K wires that define the final form of the synthesized netlist.

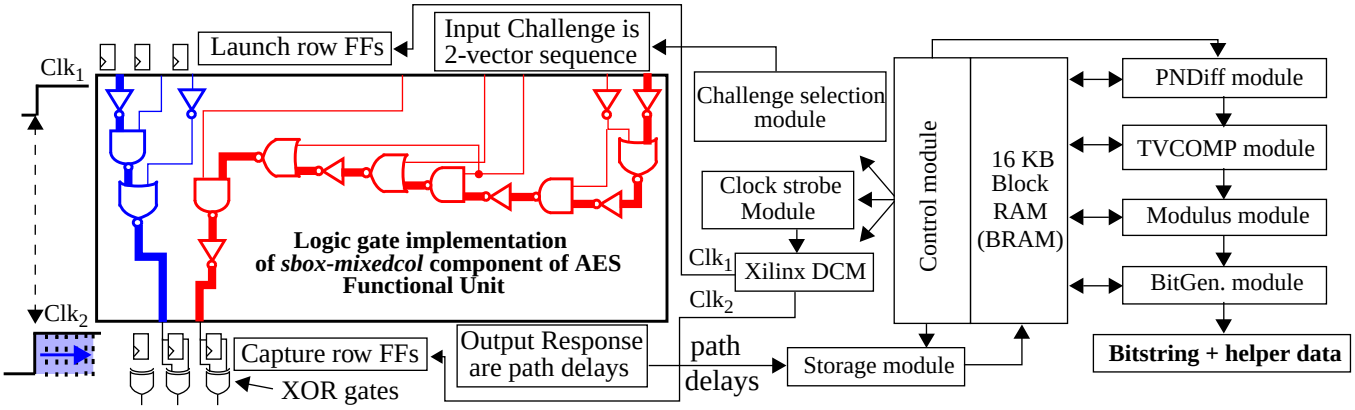

Fig. 1. Instantiation of the HELP entropy source (left) and HELP processing engine (right).

HELP defines challenges as 2-vector sequences. The sequences are applied to the PIs of the functional unit and the delays of the sensitized paths are measured at the POs. The delay of a path is the amount of time (Δt) it takes for a rising or falling signal to propagate along the path from PI to PO. High precision measurements of path delay are obtained using a clock-strobing technique which is graphically depicted on the left side of Fig. 1. The challenge is repeatedly applied to the PIs of the functional unit using the Launch row flip-flops (FFs), which are driven by Clk_1. The Capture row FFs are driven by a second clock, Clk_2, whose phase is incrementally increased by small Δt's (approx. 18 ps) across the sequence of repeated applications of the 2-vector challenge. The digital clock manager (MMCM) on a Xilinx FPGA is used to generate and tune the phase offsets between the two clocks. The process terminates when all of the emerging signal transitions on the POs are successfully captured in the Capture row FFs. The status of each PO is monitored by an XOR gate, which is connected between the input and output of each Capture row FF. A successful capture of an emerging signal transition occurs when the XOR outputs a 0, which occurs when the input and output of the FF are the same. At the beginning of the test sequence, the phase shift between Clk_1 and Clk_2 is too small to allow a successful capture. Therefore, the XOR gates output a 1 (except on outputs that do not have transitions). The first test in the clock-strobing sequence which causes the XOR gate to output a 0 identifies the phase shift value that best represents the delay of the path. The term *launch-capture-interval* (**LCI**) is used to refer to the current phase shift value. The finite state machine that implements the clock strobing technique is labeled as *Clock strobe* module in the center portion of Fig. 1.

The phase shift values used to represent the path delays are 12-bit integers, which typically vary between 100 (1.8 ns) to 600 (10.8 ns). These integer-based path delays are collected and stored by the *storage* module in an on-chip block RAM (BRAM) (see Fig. 1). A *Path-Select-Mask* is also sent by the verifier (not shown), along with the challenges, to specify which path outputs, from those that have transitions, are actually stored. The BRAM stores the digitized path delays as 16-bit values, with an additional 4 bits added as a fixed point fraction to enable averaging of up to 16 samples. The bitstring generation algorithm requires a set of challenges and masks to be applied that test a total of 2048 paths with rising transitions and 2048 paths with falling transitions. The term **PN** is used to refer to the 16-bit averaged path delays in the following.

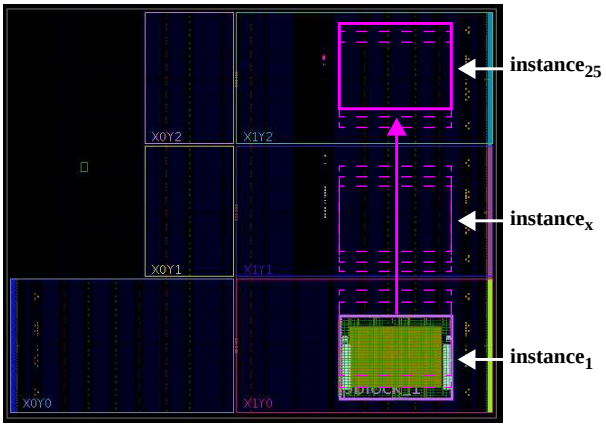

Fig. 2. sbox-mixedcol functional unit instance placement in Xilinx Zynq 7020 using Vivado implementation view.

3.1 Experimental Setup

The data analyzed in this paper is collected from a set of 20 FPGAs (chips). For each chip, we created 25 identical, but shifted, instances of *sbox-mixedcol* for a total of 500 **chip-instances**. The shifted versions are shown in Fig. 2 as *instances*, highlighted as magenta rectangles in a screen snapshot of Implementation View created by Xilinx Vivado. In order to keep the contents within the magenta rectangles identical, a Xilinx construct called a *pblock* is used as a container for the *sbox-mixedcol*. Vivado synthesis is performed only once for the *sbox-mixedcol* design, and tcl commands are used to save a set of constraints which fix the locations of the wires and LUTs in a file called a *check-point*. A set of 25 programming bitstreams are generated one-at-a-time by shifting the fixed contents within the *pblock* vertically as shown by sequence of magenta rectangles in Fig. 2. For each instance, the base *y* coordinate of the *pblock* is incremented by 3 as a means of implementing the vertical shift. The shifted versions of the design significantly increase the size of our data set (from 20 to 500), which in turn, increases the statistical significance of the analysis.

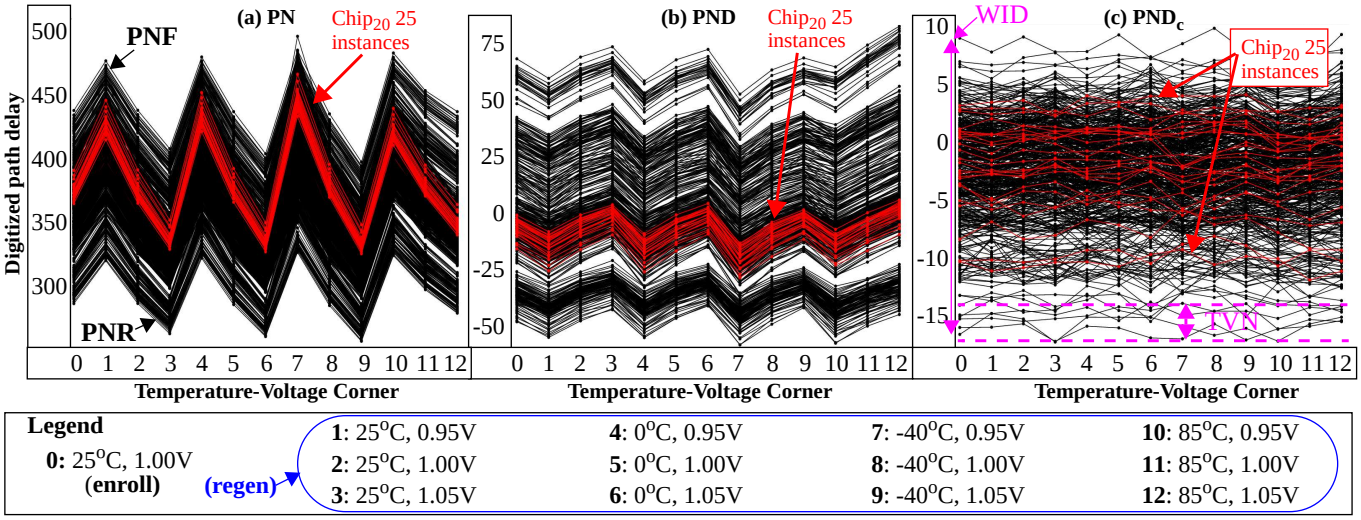

Fig. 3. (a) Example rising (PNR) and falling (PNF) path delays, (b) (PNR-PNF) path delay differences (PND) and (c) TV Compensated PND_c for 500 chips (individual curves) and 16 TV corners (points in curves).

3.2 PN Processing

The bitstring generation process is carried out using the stored PN as input. The right side of Fig. 1 lists the operations performed by a set of state machines during bitstring generation. The operations are simple and therefore, they can be applied in time linear to the size of the stored PN (4096 in total). The first operation is performed by the *PNDiff* module. *PNDiff* creates PN differences by subtracting the 2048 falling PN from the 2048 rising PN. Pairings between rising and falling PN are determined by two seeded 11-bit linear feedback shift registers (LFSR). The LFSRs each require an 11-bit *LFSR seed* to be provided as input during the first iteration of the algorithm. The two LFSR seeds can be varied from one run of the HELP algorithm to the next. We refer to the LFSR seeds as user-specified *configuration* parameters. The term **PND** is used subsequently to refer to the PN differences. The *PNDiff* module stores the 2048 PND in a separate portion of the BRAM.

The waveforms shown in Fig. 3(a) illustrate this process using data obtained from a set of FPGA experiments in which exactly two paths are tested, one with a rising transition (PNR) and one with a falling transition (PNF). Each waveform plots the PNR and PNF measured from one of the 500 chip-instances. The 13 line-connected points in each waveform represent delays from the same path but measured under different environmental conditions, called temperature-voltage (TV) corners. The left-most points in the waveforms (assigned 0 along the x-axis) represent the values measured with the conditions set to 25°C, 1.00V. The term **enrollment** refers to data collected under this (nominal) TV corner. The x-axis positions 1, 2 and 3 identify PN measured at 25°C but at supply voltages of 0.95, 1.00 and 1.05 V. The legend below the figure gives the correspondence for other x-axis values. The term **regeneration** refers to data collected under TV corners 1 through 12. Fig. 3(b) shows the corresponding PND waveforms that are computed by subtracting the fall PN from the rise PN shown in (a).

From Fig. 3(a), it is clear that changes in temperature-voltage conditions change delay (otherwise the waveforms would be straight horizontal lines). Variations in delay introduced by changes in TV conditions are undesirable because such changes reduce the ability of the HELP algorithm to reproduce the generated bitstrings, a function that is required when the bitstrings are used as security keys. Moreover, from Fig. 3(b), the PND also portray TV-related variations, despite the fact that the difference operation reduces their magnitude over that shown in (a). **TV compensation** or **TVCOMP** is a process designed to further reduce TV-related variations, such as those that remain in (b).

The TVCOMP process measures the mean and range of the PND **distribution** and applies a linear transformation to the original PND as a means of removing TV-related variations. A histogram distribution of the 2048 PND is created in a separate portion of the BRAM shown in Fig. 1, which is then parsed to obtain its mean and range parameters. Changes in the mean and range of the PND distribution capture the shifting and scaling that occurs to the delays when temperature and/or supply voltage vary above or below the nominal values. The mean and range parameters, μ_{chip} and Rng_{chip}, are used to create standardized values, $zval_i$, from the original PND according to Eq. (1). The fractional $zval_i$

$$\mathrm{zval}_i = \frac{(\mathrm{PND}_i - \mu_{\mathrm{chip}})}{\mathrm{Rng}_{\mathrm{chip}}} \qquad \textbf{Eq. 1.}$$

$$\mathrm{PND}_c = \mathrm{zval}_i \mathrm{Rng}_{\mathrm{ref}} + \mu_{\mathrm{ref}} \qquad \textbf{Eq. 2.}$$

are transformed back into fixed point values using Eq. (2). The **reference** distribution parameters, μ_{ref} and Rng_{ref}, given in Eq. (2) are also user-specified configuration parameters, adding to the *LFSR seeds* described earlier.

Fig 3(c) illustrates the impact of TVCOMP using the PND from Fig. 3(b). The same μ_{ref} and Rng_{ref} is used in all TVCOMP transformations of the data obtained from the 500 chip-instances at each of the 13 TV corners (Note: 500 x 13

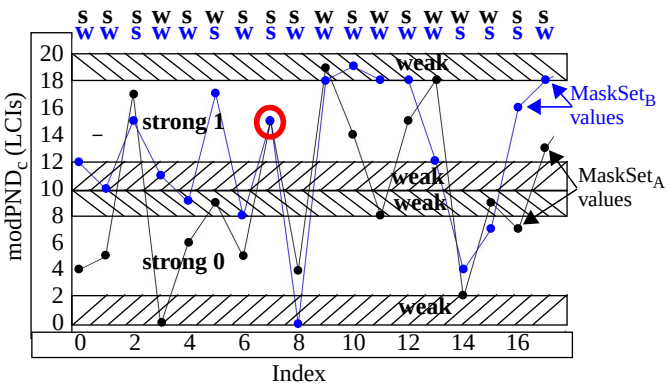

Fig. 4. Illustration of the Modulus-Margin process carried out by HELP for bitstring generation.

= 6500 applications of TVCOMP are applied). The TV compensated PND are referred to as **PND_c**. The zig-zag trends evident in (b) are eliminated in (c) and the shape of the waveforms are closer to the ideal 'horizontal line'. Also, in addition to TV-related variations, TVCOMP also eliminates global (chip-wide) performance differences that occur between chips, leaving only within-die variations (**WDV**). WDV are widely recognized as the best source of Entropy for PUFs. As an illustration, the highlighted red waveforms in Figs. 3(a), (b) and (c) are associated with the 25 instances created on $chip_{20}$. The close grouping of the waveforms in Fig. 3(a) and (b) illustrates that the performance characteristics of all instances are similar. This is the expected result because the path delays for these 25 instances are measured from the same chip. In contrast, Fig. 3(c) shows the red waveforms are in fact distributed across most of the range, and are inter-mingled with the 450 waveforms from the remaining 19 chips. Therefore, the distinction in the PND attributable to global performance variations is eliminated in the PND_c. WDV, on the other hand, are preserved and are the primary source of variations that remain in the PND_c.

A second important component of the variations that remain in Fig. 3(c) is referred to as *uncompensated* TV noise (**TVN**). TVN is portrayed by the variations in each waveform that occur across TV corners. TVN is illustrated in the bottom-most curve of Fig. 3(c), with the dotted lines delineating its worst case behavior at approx. 3 LCIs (which translates to approx. 90 ps). The probability of a bit-flip error during bitstring regeneration is directly related to the magnitude of TVN. The primary purpose of TVCOMP is to minimize TVN and therefore, to improve the reliability of bitstring regeneration. However, TVCOMP can also be used to improve randomness and uniqueness in the enrollment-generated bitstrings, and is at the heart of the contributions described in this paper.

The *Modulus* module shown on the right side of Fig. 1 applies a final transformation to the PND_c. Modulus is a standard mathematical operation that computes the positive remainder after dividing by the modulus. The bias introduced by testing paths of arbitrary length reduces randomness and uniqueness in the generated bitstrings. The Modulus operation significantly reduces, and in some cases eliminates, large differences in the lengths of the tested paths. The value of the Modulus is also a user-specified configuration parameter, similar to the *LFSR seeds*, μ_{ref} and Rng_{ref} parameters, and is discussed further below. The term **$modPND_c$** is used to refer to the values used in the bitstring generation process.

3.3 Bitstring Generation

The bitstring generation process uses a fifth user-specified configuration parameter, called the *Margin*, as a means of further improving the reliability of the bitstring regeneration process (beyond that provided by the TVCOMP process). Fig. 4 illustrates the bitstring generation process using two sets of 18 $modPND_c$ from $Chip_1$ labeled $MaskSet_A$ and $MaskSet_B$[1]. A modulus of 20 is used in combination with a set of margins of size 2 surrounding two strong bit regions of size 6. HELP classifies the $modPND_c$ as **strong** (s) and **weak** (w) based on their position within the range defined by the Modulus. Designators along the top given as 's' and 'w' indicate the classification status of the enrollment $modPND_c$. Data points that fall on or within the hatched areas are classified as weak.

The Margin method improves bitstring reproducibility by eliminating data points classified as 'weak' in the bitstring generation process because they are too close to the bit-flip lines of 10 and 0 (or 20). A helper data bitstring is generated to record the status of the bits using 0 for weak and 1 for strong. A strong bitstring is constructed using only those data points classified as strong. When HELP is used in authentication protocols, both the helper data bitstring and strong bitstring are sent to the verifier in the clear and therefore, an adversary can leverage this information to model-build the PUF.

4 Distribution Effect

As indicated above, the *Path-Select-Masks* are configured by the server to select different sets of *k* PN among the larger set *n* generated by the applied challenges (2-vector sequences). In other words, the 4096 PN are not fixed, but vary from one authentication to the next. For example, assume that a sequence of challenges produces a set of 5000 rising PN and a set of 5000 falling PN, from which the server selects a subset of 2048 from each set. The number of ways of choosing 2048 from 5000 is given by Eq. 3.

$$\text{Path-select-combos} = \binom{5000}{2048} = 3.3e^{1467} \qquad \textbf{Eq. 3.}$$

From this equation, it is clear that the *Path-Select-Masks* enables the PN to be selected by the server in an exponential *n*-choose-*k* fashion. However, there are only 5000^2 possible PND that can be created from these rising and falling PN. Therefore, the exponential *n*-select-*k* ways of selecting the PN would be limited to choosing among the n^2 number of bits (one bit for each PND) unless it is possible to vary the bit value associated with each PND. This is precisely what the Distribution Effect is able to accomplish.

Previous work has shown that an exponential number of response bits is a necessary condition for a truly strong PUF

1. The reason we include two sets of $modPND_c$ will be explained later.

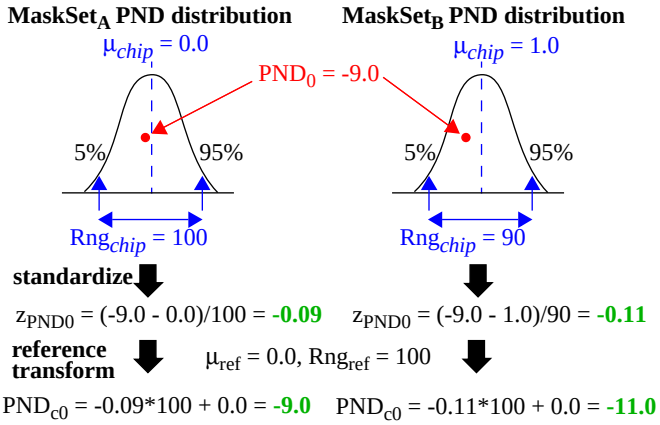

Fig. 5. Impact of the TVCOMP process on PND$_0$ when members of the PND distribution change for different mask sets *A* and *B*.

but not a sufficient condition. The responses must also be largely *uncorrelated* as a means of making it difficult or impossible to apply machine learning algorithms to model-build the PUF. The analysis provided in this section shows that the *Path-Select-Masks* in combination with the TVCOMP process add significant complexity to the machine-learning model.

The set of PN selected by the *Path-Select-Mask*s changes the characteristics of the PND distribution, which in turn impacts how each PND is transformed through the TVCOMP process. The TVCOMP process was described earlier in reference to Eqs. 1 and 2. In particular, Eq. 1 uses the μ_{chip} and Rng_{chip} of the measured PND distribution to standardize the set of PND before applying the second transformation given by Eq. 2.

Fig. 5 provides an illustration of the TVCOMP process. The two distributions are constructed using data from the same chip but selected using two different sets of *Path-Select-Masks*, MaskSet$_A$ and MaskSet$_B$. The point labeled PND$_0$ is present in both distributions, with value -9.0 as labeled, but the remaining components are purposely chosen to be different. Given the two distributions are defined using distinct PND (except for one member), it is possible that the μ_{chip} and Rng_{chip} parameters for the two distributions will also be different (a simple algorithm is described below that ensures this). The example shows that the μ_{chip} and Rng_{chip} measured for the MaskSet$_A$ distribution are 0.0 and 100, resp., while the values measured for the MaskSet$_B$ distribution are 1.0 and 90.

The TVCOMP process builds these distributions, measures their μ_{chip} and Rng_{chip} parameters and then applies Eq. 1 to *standardize* the PND of both distributions. The standardized values for PND$_0$ in each distribution are shown as -0.09 and -0.11, resp. **This first transformation is at the heart of the Distribution Effect, which shows that the original value of -9.0 is translated to two different standardized values.** TVCOMP then applies Eq. 2 to translate the standardized values back into an integer range using μ_{ref} and Rng_{ref}, given as 0.0 and 100, resp. for both distributions. The final PND$_{c0}$ from the two distributions are -9.0 and -11.0, resp. **This shows that the TVCOMP process creates a dependency between the PND and corresponding PND$_c$ that is based on the parameters of the entire distribution**.

The *Modulus-Margin* graph of Fig. 4 described earlier illustrates this concept using data from chip-instance C$_1$. The 18 vertically-positioned pairs of modPND$_c$ values included in the curves labeled MaskSet$_A$ and MaskSet$_B$ are derived from the same PND. However, the remaining PND, i.e., (2048-18) = 2030 PND, (not shown) in the two distributions are different. These differences change the distribution parameters, μ_{chip} and Rng_{chip}, of the two distributions, which in turn, introduces vertical shifts in the PND$_c$ and wraps in the modPND$_c$. The Distribution Effect affects all of the 18 pairings of modPND$_c$ in the two curves except for the point circled in red.

The Distribution Effect can be leveraged by the verifier as a means of increasing the unpredictability in the generated response bitstrings. One possible strategy is to intentionally introduce skew into the μ_{chip} and Rng_{chip} parameters when configuring the *Path-Select-Masks* as a mechanism to force diversity in bit values derived from the same PN, i.e., those PN that have been used in previous authentications. The sorting-based technique described in the next section represents one such technique that can be used by the server for this purpose.

5 Experimental Results

In this section, we construct a set of PN distributions using a specialized process that enables a systematic evaluation of the Distribution Effect. As indicated earlier, the number of possible PN distributions is exponential (*n*-choose-*k*), making it impossible to enumerate and analyze all possibilities. The fixed number of data sets constructed by our process therefore represents only a small sample from this exponential space. However, the specialized construction process described below illustrates two important concepts, namely, the ease in which bitstring diversity can be introduced through the Distribution Effect, and the near ideal results that can be achieved, i.e., the ability to create bitstrings using the same PN that possess a 50% Interchip Hamming distance. Our evaluation methodology ensures the only parameters that can change are those related to the distribution, namely, μ_{chip} and Rng_{chip}, so the differences in the bitstrings reported are due entirely to the Distribution Effect.

The distributions that we construct in this analysis include a fixed set of 300 rising and 300 falling PN drawn randomly from 'Master' rise and fall PN data sets of size 7271. The bitstrings subjected to evaluation use only these PN, which are subsequently processed into PND, PND$_c$ and modPND$_c$ in exactly the same way except for the μ_{chip} and Rng_{chip} used within TVCOMP process. The μ_{chip} and Rng_{chip} of each distribution are determined using a larger set of 2048 rise and fall PN, which includes the fixed sets of size 300 plus two sets of size 1748 (2048 - 300) drawn randomly each time from the Master rise and fall PN data sets. Therefore, the μ_{chip} and Rng_{chip} parameters of these constructed distributions are largely determined by the 1748 randomly selected rise and fall PN.

A windowing technique is used to constrain the randomly selected 1748 rise and fall PN as a means of carrying

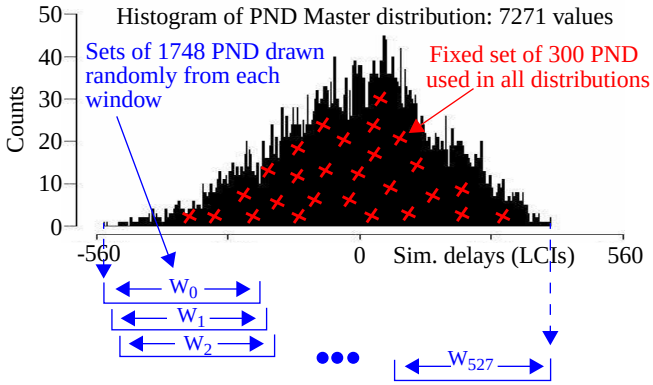

Fig. 6. Illustration of the distribution creation process using a Master distribution of 7271 PND. The 'x's represent the set of randomly selected 300 fixed PND that are included in the EVERY distribution. A set of windows W_x are used to confine the selection of the 1748 remaining PND to specific regions within the sorted Master distribution. This process is used to generate a set of 528 PND distributions of size 2048.

out a systematic evaluation which ensures that the μ_{chip} and Rng_{chip} parameters increase (or decrease) by small deltas. Since TVCOMP derives the μ_{chip} and Rng_{chip} parameters from the PND distribution, our random selection process is applied to a Master PND distribution as a means of enabling better control over the μ_{chip} and Rng_{chip} parameters.

The Master PND distribution is constructed from the Master PNR and PNF distributions in the following fashion. The 7271 elements from the PNR and PNF Master distributions are first sorted according to their worst-case simulation delays. The rising PN distribution is sorted from largest to smallest while the falling PN distribution is sorted from smallest to largest. The Master PND distribution is then created by subtracting consecutive pairings of PNR and PNF from these sorted lists, i.e., $PND_i = PNR_i - PNF_i$ for i = 0 to 7271. This construction process creates a Master PND distribution that possesses the largest possible range among all possible PNR/PNF pairing strategies.

A histogram portraying the PND Master distribution is shown in Fig. 6. The PNR and PNF Master distributions (not shown) from which this distribution is created were created from simulations of the *sbox-mixedcol* functional unit described in Section 3 using approx. 1000 challenges (2-vector sequences). The range of the PND is given by the width of the histogram as approx. 1000 LCIs (~18 ns).

The 2048 rise and fall PN used in the set of distributions evaluated below are selected from this Master PND distribution. The PND Master distribution (unlike the PNR and PNF Master distributions) permits distributions to be created such that the change in the μ_{chip} and Rng_{chip} parameters from one distribution to the next is controlled to a small delta. The red 'x's in Fig. 6 illustratively portray that the set of 300 fixed PND (and corresponding PNR and PNF) are randomly selected across the entire distribution. These 300 PND are then removed from Master PND distribution. The remaining 1748 PND for each distribution are selected from specific regions of the Master PND distribution as a means of constraining the μ_{chip} and Rng_{chip} parameters. The regions are called ***windows*** in the Master PND distribution and are labeled W_x along the bottom of Fig. 6.

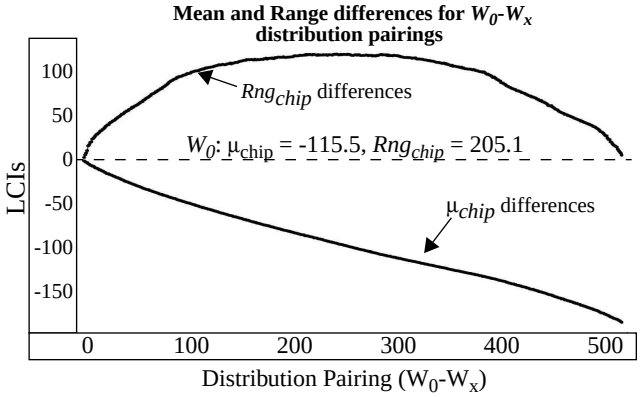

Fig. 7. Change in μ_{chip} and Rng_{chip} as the window W_x is moved from left to right over the Master distribution.

The windows W_x are sized to contain 2000 PND and therefore, the width of each W_x varies according to the density of the distribution. Each consecutive window is skewed to the right by 10 elements in the Master PND distribution. Given the Master contains 7271 total elements, this allows 528 windows (and distributions) to be created. The 2048 PND for each of these 528 distributions, referred to as W_x distributions, are then used as input to the TVCOMP process. The 300 fixed PND are present in all distributions and therefore, prior to TVCOMP, they are identical in value.

The objective of this analysis is to determine how much the bitstrings change as the μ_{chip} and Rng_{chip} parameters of the W_x distributions vary. As noted earlier, the bitstrings are constructed using only the 300 fixed PND, and are therefore of size 300 bits. We measure changes to the bitstrings using a reference bitstring, i.e., the bitstring generated using the W_0 distribution. Interchip Hamming distance (**InterchipHD**) counts the number of bits that are different between the W_0 bitstring and each of the bitstrings generated by the W_x distributions, for x = 1 to 527. The expression used for computing InterchipHD is discussed further below.

The construction process used to create the W_0-W_x distribution pairings ensures that a difference exists in the μ_{chip} and Rng_{chip} parameters. Fig. 7 plots the average difference in the μ_{chip} and Rng_{chip} of each W_0-W_x pairing, using FPGA data measured from the 500 chip-instances. The differences are created by subtracting the W_x parameter values, e.g., μ_{chipWx} and Rng_{chipWx}, from the reference W_0 parameter values, e.g., μ_{chipW0} and Rng_{chipW0}. The W_0 distribution parameters are given as μ_{chip} = -115.5 and Rng_{chip} = 205.1 in the figure. As the window is shifted to the right, the mean increases towards 0, and the corresponding (W_0 - W_x) difference becomes more negative in nearly a linear fashion as shown by the curve labeled 'μ_{chip} differences'. Using the W_0 values, μ_{chip} varies over the range from -115 to approx. +55.

The range, on the other hand, decreases as window shifts to the right because the width of the window contracts (due to the increased density in the histogram), until the mid-point of the distribution is reached. Once the mid-point is reached, the range begins to increase again. Using the W_0 values, Rng_{chip} varies from 205 down to approx. 105 at the

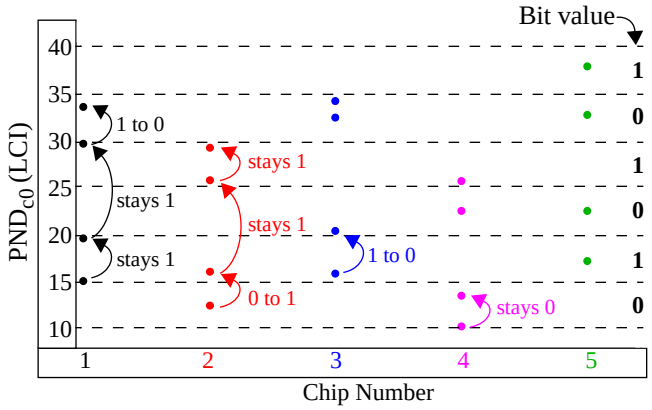

Fig. 8. Illustration showing 'shifting' (y-axis) introduced by Distribution Effect on a single PND$_{c0}$ for 5 different chips (x-axis) as window W_x from Fig. 6 is shifted from W_0 (lowest points) through W_{25}, W_{50} and W_{75} (top points).

mid-point. Note that the window construction method creates nearly all possible μ_{chip} values but only a portion of the possible Rng_{chip} values, e.g., distributions with ranges up to nearly 1000 can be constructed from this Master PND distribution. Therefore, the results reported below represent a conservative subset of all possible distributions.

Also note that Rng_{chip} continues to change throughout the set of W_x distributions. This occurs because the range is measured between the 6.25% and 93.75% points in the histogram representation of the 2048 element PND distributions. If the extreme points were used instead, the Rng_{chip} values from Fig. 7 would become constant once the window moved inside the points defined by the fixed set of 300 PND.

Fig. 8 provides an illustration of the Distribution Effect using data from several chip-instances. The effect on PND$_{c0}$ is shown for 5 chips given along the x-axis for four windows given as W_0, W_{25}, W_{50} and W_{75}. The bottom-most points are the PND$_{c0}$ for the distribution associated with W_0. As the index of the window increases, the PND$_{c0}$ from those distributions is skewed upwards. A modulus grid of 20 is shown superimposed to illustrate how the corresponding bit values change as the parameters of the distributions change.

We use InterchipHD to measure the number of bits that change value across the 527 W_0-W_x distributions. It is important to note that **we apply InterchipHD to only those portions of the bitstring that correspond to the fixed set of 300 PN**. InterchipHD counts the number of bits that differ between pairs of bitstrings. Unfortunately, InterchipHD cannot be applied directly to the HELP algorithm-generated bitstrings because of the Margining technique described in Section 3.3. Margining eliminates weak bits to create the strong bitstring (SBS), but the bits that are eliminated are different from one chip-instance to another. In order to provide a fair evaluation, i.e., one that does not artificially enhance the InterchipHD towards its ideal value of 50%, the bits compared in the InterchipHD calculation must be generated from the same modPND$_c$.

Fig. 9 provides an illustration of the process used for ensuring a fair evaluation of two HELP-generated bitstrings. The helper data bitstrings *HelpD* and raw bitstrings *BitStr*

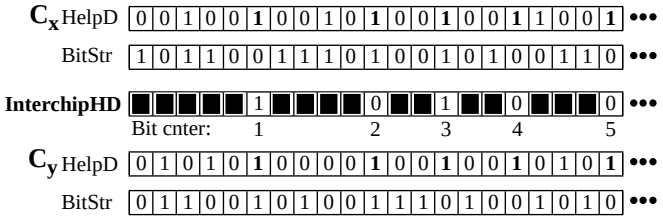

Fig. 9. Illustration showing InterchipHD process under HELP's Margin scheme.

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 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 modPND$_c$ from C_x and C_y are being used in the InterchipHD calculation. Note that the number of bits considered in each InterchipHD is less than 300 using this method, and in fact, will be different for each pairing.

Eq. 4 provides the expression for InterChipHD, HD_{Inter}, that takes into consideration the varying lengths of the individual InterchipHDs. The symbols *NC*, NB_x and *NCC* repre-

$$HD_{inter} = \left(\frac{1}{NCC} \sum_{i=1}^{NC} \sum_{j=i+1}^{NC} \frac{\left(\sum_{k=1}^{NB_x} (BS_{i,k} \oplus BS_{j,k}) \right)}{NB_x} \right) \times 100 \quad \textbf{Eq. 4.}$$

sent 'number of chips', 'number of bits' and 'number of chip combinations', resp. We used 500 chip-instances for the 'number of chips', which yields 500*499/2 = 124,750 for *NCC*. 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. The final value of *Bit cnter* from the center of Fig. 9 counts the number of bits that are used for NB_x in Eq. 4, which varies for each pairing as indicated above.

The InterchipHD results shown in Fig. 10 are computed using enrollment data collected from 500 chip-instances of a Xilinx Zynq 7020 chip as described earlier. The x-axis plots the W_0-W_x pairing, which corresponds one-to-one with the graph shown in Fig. 7. The HELP algorithm is configured with a Modulus of 20 and a Margin of 3 in this analysis (the results for other combinations of these parameters are similar). The HDs are nearly zero for cases in which in windows W_0 and W_x have significant overlap (left-most points) as expected because the μ_{chip} and Rng_{chip} of the two distributions are nearly identical under these conditions (see left side of Fig. 7). As the windows separate, the InterchipHDs rise quickly to the ideal value of 50% (annotated at W_0-W_x pairing = 4), demonstrating that the Distribution Effect provides significant benefit for relatively small shifts in the distribution parameters.

The overshoot and undershoot on the left and right sides of the graph in Fig 10 reflect correlations that occur in the

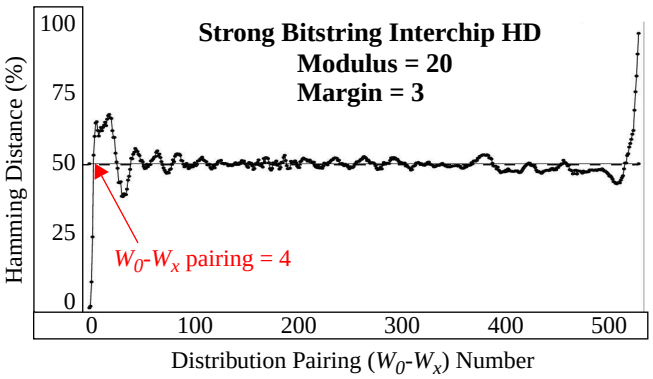

Fig. 10. Interchip HD of strong bitstrings derived from distributions in which 300 of the modPND$_c$ values are fixed (common) in each pair of distributions of size 2048.

movement of the modPND$_c$ for special case pairs of the μ_{chip} and Rng_{chip} parameters. For example, for pairings in which the Rng_{chip} of the two distributions are identical, shifting μ_{chip} causes all modPND$_c$ to rotate through the range of the Modulus (with wrap). For μ_{chip} shifts equal to the Modulus, the exact same bitstring is generated by both distributions. This case does not occur in our analysis otherwise the curve would show instances where the InterchipHD is 0 at places other than when x = 0. For μ_{chip} shifts equal to 1/2 Modulus (and with equal Rng_{chip}), the InterchipHD becomes 100%. The upward excursion of the right-most portion of the curve in Fig 10 show results where this boundary case is approached, i.e., for x > 517. Here, the Rng_{chip} of both distributions (from Fig. 7) are nearly the same and only the μ_{chip} are different.

A key take-away here is that the InterchipHDs remain near the ideal value of 50% even when simple, distribution construction techniques are used. As we noted earlier, these types of construction techniques can be easily implemented by the server during authentication.

5.1 Security Implications

The results of this analysis provide strong evidence that the Distribution Effect increases bitstring diversity. As indicated earlier, the number of PND that can be created using 7271 rising and falling PN is limited to $(7271)^2$ before considering the Distribution Effect. Based on the analysis presented, the number of times a particular bit can change from 0 to 1 and vise versa is proportional to the number of μ_{chip} and Rng_{chip} values that yield different bit values. In general, this is a small fixed value on order of 100 so the Distribution Effect provides only a polynomial increase in the number of PND over the n^2 provided in the original set.

However, determining which bit value is generated from a set of 100 possibilities for each modPND$_c$ independently requires an analysis of the distribution, and there are an exponential *n*-choose-*k* ways of building the distribution using the *Path-Select-Masks*. Therefore, model-building needs to incorporate inputs that track the form of the distribution, which is likely to increase the amount of effort and the number of training CRPs significantly. Furthermore, for authentication applications, the adversary may need to compute the predicted response in real-time after the verifier has sent the challenges and *Path-Select-Masks*. This adds considerable time and complexity to an impersonation attack, beyond that required to build an accurate model. Unfortunately, a closed-form quantitative analysis of the benefit provided by the Distribution Effect is non-trivial to construct. Our on-going work is focused on determining the difficulty of model-building the HELP PUF as an alternative.

6 Conclusions

A novel entropy-enhancing technique called the Distribution Effect is proposed for the HELP PUF that is based on purposely introducing biases in the mean and range parameters of path delay distributions. The biased distributions are then used in the bitstring construction process to introduce differences in the bit values associated with path delays that would normally remain fixed. **The Distribution Effect changes the bit value associated with a PUF's fixed and limited underlying source of Entropy, expanding the CRP space of the PUF**. The technique uses *Path-Select-Masks* and a TVCOMP process to vary the path delay distributions over an exponential set of possibilities. The Distribution Effect is likely to make the task of model-building the HELP PUF significantly more difficult, which is supported by our on-going work in this area.

7 References

[1] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, "Handbook of Applied Cryptography," CRC Press, ISBN: 0-8493-8523-7, Oct. 1996, http://cacr.uwaterloo.ca/hac/

[2] S. P. Skorobogatov, "Semi-Invasive Attacks - A New Approach to Hardware Security Analysis," *Technical Report UCAM-CL-TR-630*, 2005.

[3] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Silicon Physical Random Functions", *Computer and Communication Security Conference*, Nov. 2002.

[4] J. Aarestad, J. Plusquellic, D. Acharyya, "Error-Tolerant Bit Generation Techniques for Use with a Hardware-Embedded Path Delay PUF," *HOST*, 2013, pp. 151-158.

[5] W. Che, F. Saqib, J. Plusquellic, "PUF-Based Authentication", *ICCAD*, 2015, pp. 337-344.

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

[7] W. Che, V. K. Kajuluri, M. Martin, F. Saqib and J. Plusquellic, "Analysis of Entropy in a Hardware-Embedded Delay PUF", *Cryptography*, 2017.

[8] R. van den Berg, B. Skoric, and V. van der Leest, "Bias-based modeling and entropy analysis of PUFs", *TrustED'13*, 2013.

[9] S. Katzenbeisser, U. Kocabas, V. Rozic, A. Sadeghi, I. Verbauwhede and C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon", *CHES* 2012, pp. 283-301.

[10] D. Ganta and L. Nazhandali, "Easy-to-Build Arbiter Physical Unclonable Function with Enhanced Challenge/Response Set," in International Symposium on Quality Electronic Design, *ISQED* 2013, March 2013, pp. 733–738.

[11] https://en.wikipedia.org/wiki/AES

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