Novel Offset Techniques for Improving Bitstring Quality of a Hardware-Embedded Delay PUF

W. Che, F. Saqib* and J. Plusquellic, Unversity of New Mexico and *University of North Carolina Charlotte

Abstract

Statistical properties including uniqueness, randomness and reproducibility are commonly used as metrics for Physical Unclonable Functions (PUFs). When PUFs are used in authentication protocols, the first two metrics are critically important to the overall security of the system. In this paper, we investigate the statistical qualities of bitstrings generated by a Hardware-Embedded Delay PUF called HELP using hardware data collected from a set of 500 Xilinx Zynq FPGAs. HELP measures and analyzes variations in path delays that occur within a hardware-implemented macro. Two novel techniques are proposed in which the verifier computes a set of offsets that are used to fine tune the token's digitized path delays as a means of maximizing Entropy and reproducibility in the generated bitstrings. The offsets are derived from the enrollment data stored by the server in a secure database. A population-based offset method is proposed which computes median values using data from multiple tokens (the population). A second chip-specific technique is proposed which fine tunes path delays using the stored enrollment data associated with the authenticating token. The analysis of FPGA data shows that the population-based offset method significantly improves Entropy while the chip-specific technique, used alone or in combination with the population-based method, significantly improves reproducibility.

1 Introduction

Security and trust have become critically important for a wide range of existing and emerging microelectronic systems including those embedded in aerospace and defense, industrial ICS and SCADA environments, automotive and autonomous vehicles, data centers, communications and medical healthcare devices. The vulnerability of these systems is increasing with the proliferation of internet-enabled connectivity and unsupervised in-field deployment. Authentication and encryption are heavily used for ensuring data integrity and privacy of communications between communicating devices. These protocols require keys and bitstrings (secrets) to be stored in non-volatile memory (NVM). Current methods utilizing a NVM-based key represent a vulnerability, particularly in fielded systems where adversaries can access the hardware and carry out probing and other invasive attacks uninhibited. Physical Unclonable Functions or PUFs on the other hand provide an alternative to NVM-based key-storage, and for the generation of unique and unpredictable authentication information.

PUFs extracts random information (Entropy) from variations in the physical and electrical properties of ICs, that are unique to each IC, as a means of generating digital secrets (bitstrings). The type and amount of information available to a PUF through these physical-layer variations in the chip are critically important security properties, and are the most often cited benefits of PUFs over conventional NVM-based alternatives.

A PUF is defined by a source of on-chip electrical variations. The hardware-embedded Delay PUF (HELP) investigated in this paper generates bitstrings from delay variations that occur along paths in an on-chip macro (**functional unit**), such as a cryptographic primitive. Therefore, the circuit structure that HELP utilizes as a source of random information differs from traditional PUF architectures which use precisely placed and routed arrays of identically designed components. In contrast, HELP imposes no restrictions on the physical layout characteristics of the Entropy source.

This departure from a traditional definition of a PUF architecture provides both advantages and disadvantages. An important advantage is related to the effort involved in constructing the functional unit and the diversity in the number of possible implementations. For example, commercial logic synthesis tools such as Xilinx Vivado can be used to quickly build multiple implementations of the functional unit. Each implementation is identical in function but has a unique circuit architecture and therefore, a unique set of path delays.

The widely varying nature of path delays in an arbitrarily synthesized functional unit represents the disadvantage. Unlike PUFs with identically designed components, comparing the delays of paths within the functional unit during bitstring generation introduces bias. Bias reduces the statistical quality of the bitstrings by skewing the distribution of '0's and '1's away from the ideal of 50%. Although it is possible to implement a bitstring generation algorithm that identifies and compares paths that are nearly equal in delay to address this problem, the effectiveness of the procedure would depend on the quality of the delay model and/or characterization data. Furthermore, implementing constraints of this nature would increase the complexity of the bitstring generation algorithm.

Instead, HELP addresses the path length bias problem by first subtracting pairs of path delays and then applying a modulus operation to the difference. The modulus operator divides the path delay differences by a constant, which we refer to as the **modulus**, and returns the remainders. The modulus is chosen to minimize the bias that remains in the differences for pairings of paths that are originally of different lengths, while simultaneously preserving the smaller delay variations that occur because of random processes, e.g., within-die process variations.

In order to ensure that bias is minimized for all path pairing combinations used to generate bits, the modulus needs to be as small as possible. This is true because the range of the smaller and randomly varying component of path delays that contribute to the randomness and uniqueness properties of HELP-generated bitstrings is upper bounded. The modulus is also lower bounded by temperature and power supply noise sources which adversely affect bitstring reliability. Therefore, the range of *suitable* moduli that achieve the PUF's primary goals of producing unique, random and reproducible bitstrings is upper and lower bounded

to a specific range.

In this paper, we propose **population-based** and **chip-specific offset** methods for improving Entropy, reliability and the length of the HELP generated bitstrings. The methods build on a concept first presented in [1]. The population-based offset method can also widen the range of suitable moduli that can be used while maintaining zero-information leakage in the helper data. Information leakage associated with the chip-specific offset method can be kept near zero with constraints imposed on the parameters used by the HELP engine.

We describe the techniques in reference to a PUF-based authentication scenario [1][2], which occurs between a hardware token and a verifier. In our proposed authentication protocol, a set of path delays are measured and stored by the verifier in a secure database during the enrollment process, i.e., before the token is released for field use. The proposed population-based offset method also requires the verifier to compute and store a set of *median* values of each path delay using the enrollment data of all tokens (the population). During authentication, the verifier selects a modulus and then computes the difference between the mean path delays and the modulus, and encodes the differences (called **offsets**) in the challenge data sent to the token. The token and verifier add the offsets to the path delays before computing the corresponding bits. The offsets effectively shift the distributions of the path delays such that approx. half of the chips generate a '0' and half generate a '1', maximizing the Entropy of each generated bit.

We evaluate the effectiveness of the offset technique using inter-chip Hamming distance and Entropy metrics from data collected from a set of Xilinx Zynq FPGAs. The results are compared with those obtained using the original 'unshifted' data. The following summarizes the specific contributions of this paper.

- We discuss the implementation details of the proposed population-based and chip-specific offset methods, as well as their benefits and limitations.
- Statistical tests including Entropy, Min-Entropy, Inter-chip hamming distance (HD), Intra-chip HD and the NIST statistical test suite are used to show the impact of the offset methods on Entropy, uniqueness and reproducibility.
- The overhead associated with the offset methods is compared with previously proposed ECC methods.

The remainder of this paper is organized as follows. Section 2 presents related work and Section 3 provides an overview of HELP. Sections 4 and 5 provide details on the population-based offset method and a second chip-specific offset method, resp, and corresponding experimental results. Section 6 compares the overhead associated with PUF bit and Helper data generation with previous proposed ECC methods and summarizes FPGA resource utilization. Conclusions are provided in Section 7.

2 Related Work

Implementation-related bias that is associated with identically-designed PUF architectures, in particular, the Arbiter PUF, have been addressed in [5] with a Programmable Delay Line (PDL) and in [6][7] using a Double Arbiter PUF. The authors in [8] propose to select adjacent RO pairs as a means of reducing systematic across-chip bias effects. A regression-based distiller is proposed to decouple systematic variations of the RO PUF in [9]. The authors in [10] proposed a framework that enables Optical Proximity Correction (OPC) to increase the randomness and uniqueness for PUFs. A statistical metric which evaluates architectural bias is proposed in [11] and applied to variants of the Arbiter PUF. A recent work [12] summarizes and compares the efficiency and overhead of several Error Correction Codes (ECC) methods proposed previously in [13-18]. They also propose a lightweight key reconciliation based ECC method.

HELP and the population-based method are described in previous work [1], [3] and [4]. However, a thorough and comparative analysis of the population-based offset method, alone and in combination with a new chip-specific method, is presented for the first time in this paper. The data from the overhead analysis presented for the ECC methods in [12] is leveraged in our comparative analysis.

3 HELP Overview

HELP attaches to an on-chip module (functional unit), such as a hardware implementation of the cryptographic primitive, as shown in the block diagram of Fig. 1. The logic gate structure of the functional unit defines a complex interconnection network of wires and logic gates. The functional unit in the block diagram is a 32-bit column from Advanced Encryption Standard (AES) which includes 4 copies of the SBOX and 1 copy of the MIXEDCOL (called **sbox-mixedcol**) [19][20]. This combinational data path component is implemented in a Wave Dynamic Differential Logic (WDDL) logic style [21], which doubles the number of primary inputs and primary outputs to 64. The implementation of *sbox-mixedcol* requires approx. 3000 LUTs on a Xilinx Zynq FPGA and provides approx. 8 million paths. Although the analysis carried out in this paper uses *sbox-mixedcol*, alternative lighter-weight functional units can also be used [22].

HELP accepts challenges as 2-vector sequences. The vector sequences are applied to the primary inputs of the functional unit and the delays of the sensitized paths are measured at the primary outputs. Path delay is defined as the amount of time (Δt) it takes for a set of 0-to-1 and 1-to-0 transitions introduced on the primary inputs to propagate through the logic gate network and emerge on a primary output.

A clock-strobing technique is used 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 using the Launch row flip-flops (FFs) and the output responses are measured 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 is successfully captured in the Capture row FFs. A set of XOR gates connected between the inputs and outputs of the Capture row FFs 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 strobing operation causes the current phase shift value to be recorded as the digitized delay value for this

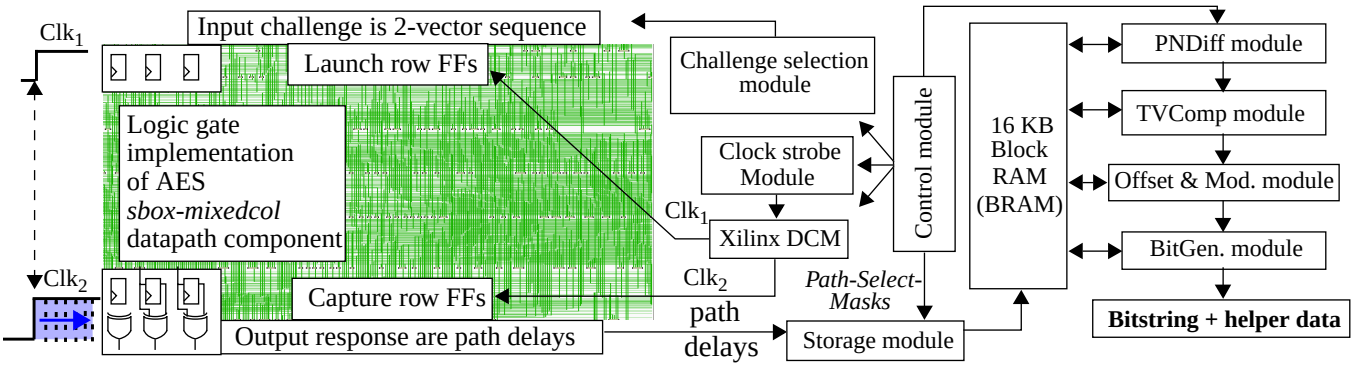

Fig. 1. Block diagram of the HELP Entropy source (left) and HELP processing engine (right).

path. The current phase shift value is referred to as the launch-capture-interval (**LCI**)[1]. The *Clock strobe* module is shown in the center portion of Fig. 1.

The digitized path delays are transferred by the *Storage* module into an on-chip block RAM (BRAM). Each digitized timing value is stored as a 16-bit value, with 12 binary digits serving to cover a signed integer range of [-2048, 2047] and 4 binary digits of fixed point precision to enable up to 16 samples of each path delay to be averaged. The upper half of the 16 KB BRAM shown in Fig. 1 allows 4096 path delays to be stored.

The challenges for HELP are defined as a set of 2-vector sequences, *Path-Select-Masks* and offsets. The ***Path-Select-Masks*** and offsets are also constructed by the verifier, along with the 2-vector sequences. A *Path-Select-Mask* is associated with each 2-vector sequence, and is used to select a subset of the delays produced for paths tested by the 2-vector sequence. The selected delays are stored in the BRAM for subsequent processing. The offsets are described below and are the focus of this paper. HELP configures the 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**.

3.1 PN Processing

Once the PN are collected, a sequence of mathematical operations are applied as shown on the right side of the Fig. 1 to produce the bitstring and helper data. The *PNDiff* module creates unique, pseudo-random pairings between the rising and falling PN using two 11-bit linear feedback shift registers (LFSRs). Two 11-bit *LFSR seeds* are used to initialize the LFSRs, and are referred to as *configuration parameters*. The two 11-bit LFSR seeds (as well as several other configuration parameters described below) are derived from an XOR'ed nonce using two nonces generated separately by the token and verifier during authentication [1]. The PN differences (**PND**) are defined as (rising $PN_{LFSR1(i)}$ - falling $PN_{LFSR2(i)}$), where i = 0 to 2047. The PND are stored in the lower portion of the BRAM.

Changes in the temperature and supply voltage (TV) negatively impact bitstring reproducibility. HELP uses a *TV compensation* (***TVComp***) process to reduce variations in the PND introduced by changes in TV conditions (called TV noise). *TVComp* is applied to the set of 2048 PND stored in BRAM. The *TVComp* procedure first converts the PND to 'standardized' values. Eq. (1) represents the first transformation which makes use of two constants, μ_{chip} and Rng_{chip}, obtained by measuring the mean and range of the histogram distribution defined by the PND. The second transformation

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

$$PND_{ci} = zval_i Rng_{ref} + \mu_{ref} \quad \textbf{Eq. 2.}$$

is represented by Eq. (2), which translates the standardized *zvals* to a new distribution with mean μ_{ref} and range Rng_{ref}. The μ_{ref} and Rng_{ref} constants are also *configuration* parameters of the HELP algorithm. The *TVComp*'ed PND are referred to as $\mathbf{PND_c}$.

The variations that remain in the PND_c are those introduced by within-die variations (**WDV**) and *uncompensated* TV noise (**UC-TVNoise**). UC-TVNoise sets the low bound on the range of suitable moduli as discussed earlier, while WDV defines the upper bound. The offset method described below is designed to extend the range of suitable moduli upwards while maintaining or improving the randomness, uniqueness and reproducibility statistical quality metrics of the generated bitstrings.

The offset and modulus operations are applied as the 3rd and 4th operations by the *Offset & Mod.* module shown on the right side of Fig. 1. The offsets are computed by the server and transmitted to the token as a component of the challenge. Offsets are added to the PND_c to produce $\mathbf{PND_{co}}$. The modulus operator computes the positive remainder after dividing the PND_{co} by the modulus value. The final values, referred to as $\mathbf{modPND_{co}}$, are used by the *BitGen* module to generate the bitstring.

3.2 Bitstring Generation

The *BitGen* module uses a fifth *configuration* parameter, called the *margin*, as a means of improving reliability. As an example, the curves labeled PND_{co} in Fig. 2(a) plot 18 of the 2048 PND_{co} from $Chip_1$ along the x-axis. The red curve line-connects the data points obtained under **enrollment** conditions (25°C, 1.00V) while the black curves line-connects data points under a set of 12 **regeneration** TV corners, which in our experiments, is all combinations of temperatures -40°C, 0°C, 25°C, 85°C with supply voltages 0.95V,

1. Table 3 at the end of this paper defines all of the acronyms used in this document.

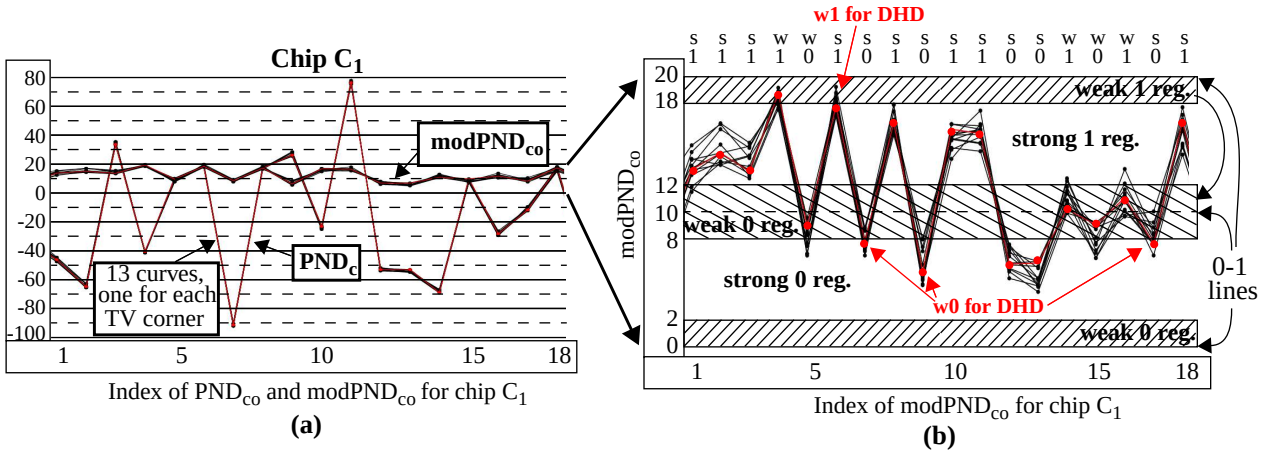

Fig. 2. (a) Conversion from PND_c to $modPND_c$ and (b) Strong/Weak PND_c classification using margining.

1.00V and 1.05V.

The curves plotted in the center of Fig. 2(a) show the $modPND_{co}$ values after a modulus of 20 is applied. Fig. 2(b) enlarges this region and includes a set of margins of size 2 surrounding two strong bit regions of size 6. HELP classifies the $modPND_{co}$ as **strong** (s) and **weak** (w) based on the region in which it is located. Designators along the top given as 's0', 's1', 'w0' and 'w1' indicate the classification status and bit value of the enrollment $modPND_{co}$ (red points). 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. For example, the data points at indexes 14 and 16 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 region than its corresponding enrollment value. The term **Single Helper Data** (SHD) refers to the bitstring generated by this bit-flip avoidance scheme because the classification of the $modPND_{co}$ as strong or weak is determined solely by the enrollment data.

A second technique, referred to as the **Dual Helper Data** (DHD) scheme, requires that both the enrollment and regeneration $modPND_{co}$ be in strong bit regions before allowing the bit to be used in the bitstring during regeneration by either the token or verifier. In the DHD scheme, SHD is first generated by both the token and verifier and the SHD bitstrings are exchanged. A DHD bitstring is created by bitwise 'AND'ing the two SHD bitstrings. The DHD scheme doubles the protection provided by the margin against bit-flip errors because the $modPND_c$ produced during regeneration must now move (because of UC-TVNoise) across both a '0' and '1' weak region before it can introduce a bit-flip error. This is true because both the enrollment and regeneration $modPND_{co}$ must be classified as strong to be included in the bitstring and the strong bit regions are separated by 2**margin*.

Fig. 2(b) highlights four cases where an enrollment-classified strong bit would be reclassified as weak in the DHD scheme. This occurs for only those regeneration $modPND_{co}$ that fall within a weak region. Therefore, the DHD scheme also enables different bitstrings to be produced each time the token authenticates even when using the same challenges and *configuration* parameters. The bitstrings constructed using only strong bits are referred to as **StrongBS**.

4 Population-Based Offset Method

The modulus operation described earlier removes most, but not all, of the bias associated with the paths of different lengths. The offset method is designed to remove the remaining component of this bias. It accomplishes this by shifting the individual PND_c upwards. The shift amount, which is always less than 1/2 the modulus, is computed by the server using the enrollment data associated with a subset of the tokens stored in its database. **The objective of the population-based offset method is to increase Entropy** by adjusting the population associated with each PND_c such that the number of tokens which generate 0 is nearly equal to the number that generate a 1. The best results are obtained when data from the entire database is used. However, significant improvements in Entropy can be obtained using smaller, randomly selected subsets of tokens in cases where the database is very large. Note that the offset method adds a third component to the challenge (beyond the 2-vector sequences and *Path-Select-Masks*).

In a typical authentication round, a token (fielded chip) and verifier (secure server) exchange nonces to decide on the set of HELP *configuration* parameters to be used. The verifier selects a set of 2-vector sequences and then randomly chooses 2048 rising PN (PNR) and 2048 falling PN (PNF) from those generated by the 2-vector sequences. The server constructs a set of *Path-Select-Masks* as a means of conveying this information to the token. For each PNR and PNF, the server also computes an offset (discussed below). The challenges, which include the 2-vector sequences, *Path-Select-Masks* and offsets, are then transmitted to the token.

The server computes the population-based offsets using the PNR and PNF stored in its enrollment database. Fig. 3 shows an example Enrollment database with rows corresponding to tokens, T_x, and columns corresponding to PNR/

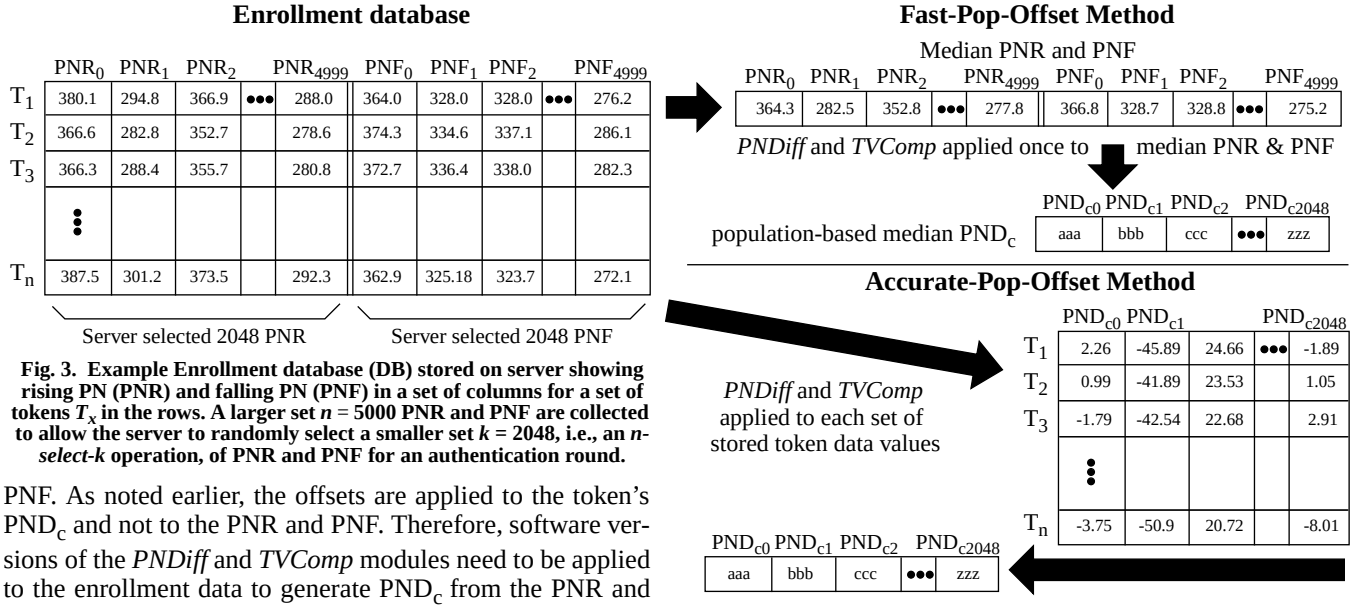

Fig. 3. Example Enrollment database (DB) stored on server showing rising PN (PNR) and falling PN (PNF) in a set of columns for a set of tokens T_x in the rows. A larger set n = 5000 PNR and PNF are collected to allow the server to randomly select a smaller set k = 2048, i.e., an *n-select-k* operation, of PNR and PNF for an authentication round.

PNF. As noted earlier, the offsets are applied to the token's PND_c and not to the PNR and PNF. Therefore, software versions of the *PNDiff* and *TVComp* modules need to be applied to the enrollment data to generate PND_c from the PNR and PNF stored for each of the tokens in the database. Once PND_c are available, the population-based offset method computes the 'median' value of each PND_c. The medians split the token population into two halves and enable the offset method to skew each PND_c individually to meet the goal of maximizing Entropy.

The availability of the enrollment data makes it possible for the server to pre-compute the median PND_c in advance of authentication requests, thereby reducing server-side delays. Unfortunately, the HELP authentication protocol makes this intractable. As we noted earlier, the LFSR seeds, and μ_{ref} and Rng_{ref} configuration parameters used to create the PND_c are selected from an XOR combination of token and server generated nonces, and are not known in advance. Moreover, the *Path-Select-Masks* generated by the server are also derived on-the-fly using a random process. Therefore, the number of median PND_c that the server would need to compute in advance is exponential to the number of stored PNR and PNF.

We developed two alternative techniques that address this issue. The first approach, called the Fast-Pop-Offset Method, computes the medians of the PNR and PNF from the database in advance (addressing the compute time burden on the server) and then applies the *PNDiff* operation to these precomputed median PNR and PNF after the LFSR seeds become available during authentication. *TVComp* is then applied to the set of median PND to obtain the median PND_c. An illustration of this method, labeled Fast-Pop-Offset Method, is provided along the top of Fig. 4. The benefit of this approach is that it is fast because the precomputed median PNR and PNF are leveraged and only one *PNDiff* and *TVComp* operation is required to obtain the median PND_c. The drawback is that it is only able to approximate the true median values of the PND_c. This is true because this technique is defined mathematically as shown by the left-hand side of Eq. 3 while the HELP algorithm implements the operation on the right-hand side. Here, *med* is an operation

Fig. 4. Two methods for computing population-based median PND_c values. Method #1 leverages precomputed PNR and PNF medians from Enrollment database while Method #2 computes population-based median PND_c from individual token stored data values.

that computes the median using data from all tokens.

$$\mathrm{TVComp}(\mathrm{med}(\mathrm{PNR}) - \mathrm{med}(\mathrm{PNF})) \neq \mathrm{med}(\mathrm{TVComp}(\mathrm{PNR\text{-}PNF}))$$

Eq. 3.

The second approach waits for the authentication parameters to become available and then applies the standard HELP algorithm processes to the enrollment data using software versions of the *PNDiff* and *TVComp* processes (as given by the expression on the right-hand side of Eq. 3). As indicated earlier, only the enrollment data from a subset *s* of the tokens is required to obtain good estimates of the population medians. This requires *s* applications of the *PNDiff* and *TVComp* operations, once for each of the *s* tokens. The median PND_c can then be computed from these sets of PND_c. An illustration of this second method, labeled Accurate-Pop-Offset Method, is provided along the bottom of Fig. 4. This approach requires more compute-time by the server but provides higher levels of Entropy in the resulting bit-strings. The Accurate-Pop-Offset Method is used to generate the results presented in subsequent sections of this paper.

Once the 2048 median PND_c are available, the population-based offsets are computed for each of the token's 2048 PND_c used in the authentication round. The offsets are integers that discretize the vertical distance from each median PND_c to the nearest 0-1 line located above the median PND_c. The integer range for each offset is 0 to 2^{OB}, with *OB* representing the number of offset bits used for each offset (*OB* is a server-defined parameter). The token and server multiply these integers by the Offset Resolution (*OR*) defined below to obtain the actual (floating point) offsets added to the PND_c.

Larger *OB* provide higher resolution but also have larger overhead. Eq. 4 expresses the offset resolution as a function

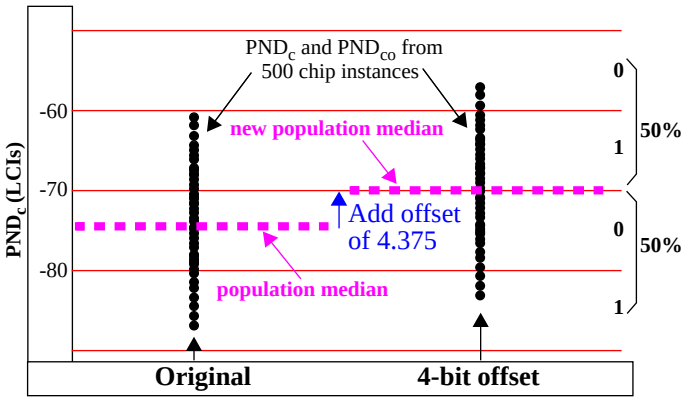

Fig. 5. Illustration of an offset calculation using one set of PND_c from 500 chip-instances and a 4-bit offset. Original distribution on left shows population median is not coincident with a 0-1 line. The offset method shifts all curves upwards to the nearest 0-1 line at -70. A modulus of 20 is used in this example.

of the number of Offset Bits (*OB*). For example, using a 4-bit offset indicates that the offset data transmitted to the token is 4*2048 = 8192 bits in length. If a modulus of 20 is used, then the offset resolution is $20/2^{4+1} = 20/32 = 0.6250$. Therefore, offsets specified using 4 bits vary from 0 to 15 and allow upward floating point skews of 0.0000, 0.6250, 1.2500 ... 9.3750 to be added to each of the 2048 PND_c. The PND_c with offset applied are referred to as PND_{co}.

$$OR = \frac{\text{Modulus}}{2^{OB+1}} \qquad \textbf{Eq. 4.}$$

Fig. 5 provides an example illustration of the process using one PND_c. The points in the vertical line graph represent a PND_c derived from 500 PNR and PNF token data sets stored in the Enrollment database. The median PND_c value is shown as a horizontal dotted line. The server computes the distance between the median and the center-located 0-1 line as 4.375, which is encoded as an offset of 7 using a 4-bit offset scheme (7*0.625 = 4.375). The token adds this offset to the corresponding PND_c it computes.

Under the condition that this same offset is used by all tokens for this particular PND_c, the shift ensures that half of the tokens place this PND_c above the 0-1 line and half place it below. Note that this does not guarantee an equal number of 0's and 1's because it is possible the spread of the distribution exceeds the width of the modulus (Fig. 5 illustrates this case). The distribution of points would need to be uniform and/or symmetric over the width of the distribution to guarantee equality. Although such 'ideal' distributions are rare in practice, most PND_c distributions possess only minor deviations from this ideal case, and therefore, nearly a full bit of Entropy is obtained as we show in the results section. Fig. 6 shows a set of 10 PND_c (black) and PND_{co} (red) superimposed, each obtained from our FPGA experiments, to better illustrate the small magnitudes associated with the actual floating point offsets.

Note that the offsets leak no information about the corresponding bit that is assigned to the PND_{co} (bits are assigned after the modulus is applied as shown in Fig. 2(b)). This is true because the offsets are computed using the PND_c

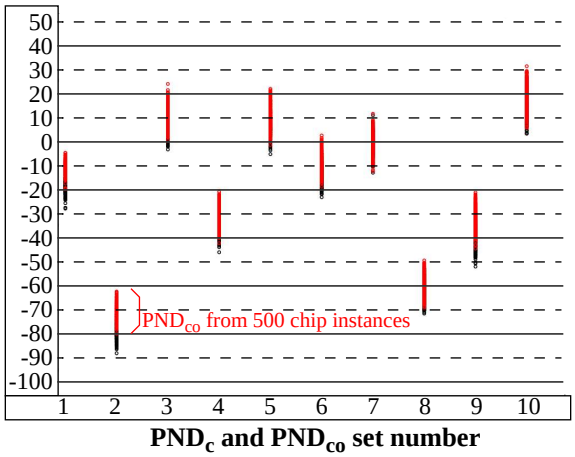

Fig. 6. Line plots of 10 PND_c from 500 chip-instances tested under enrollment conditions illustrating the small magnitudes of the vertical offsets that are actually applied. The original PND_c are shown in black while the PND_c with offset applied (PND_{co}) are shown in red.

from the token population and therefore, no chip-specific information is present in the offsets computed and transmitted by the server to the token.

Also note that it is possible to insert the offsets into unused bits of the *Path-Select-Mask*s, reducing the transmission overhead associated with the offset method. Unused bits in the *Path-Select-Masks* correspond to functional unit outputs that do not produce transitions under the applied 2-vector sequence. These bit positions in the *Path-Select-Masks* can be quickly and easily identified by both the server and token, allowing the offsets to be transparently inserted and removed in these masks.

4.1 Population-based Offset Statistical Results

We applied the offset method to the data collected from a set of 500 Xilinx Zynq FPGAs. The results are shown in two rows of bar graphs in Fig. 7 to make it easy to visualize the improvements provided by the offset method. The first row gives results without offsets while the second row gives the results when using a 4-bit offset. *Mean scaling* refers to values that are assigned to μ_{ref} and Rng_{ref} in the TVComp processing. Results using other scaling factors are similar.

The first two columns of Fig. 7 present bar graphs of Entropy and minEntropy for moduli 10 through 30 (x-axis). Entropy is defined by Eq. 5 and minEntropy by Eq. 6. The frequency p_{ij} of '0's and '1's is computed at each bit position i across the 500 bitstrings of size 2048 bits, i.e, no margin is used in this analysis. The height of the bars represent the

$$H(X) = \sum_{i=0}^{2047} \sum_{j=0}^{1} p_{ij} \log_2(p_{ij}) \qquad \textbf{Eq. 5.}$$

$$H_\infty(X) = \sum_{i=0}^{2047} -\log_2(max(p_{ij})) \qquad \textbf{Eq. 6.}$$

average values computed using the 2048-bit bitstrings from 500 chips, averaged across 10 separate LFSR seed pairs. Entropy varies from 1200 to 2040 for the 'No Offset' case

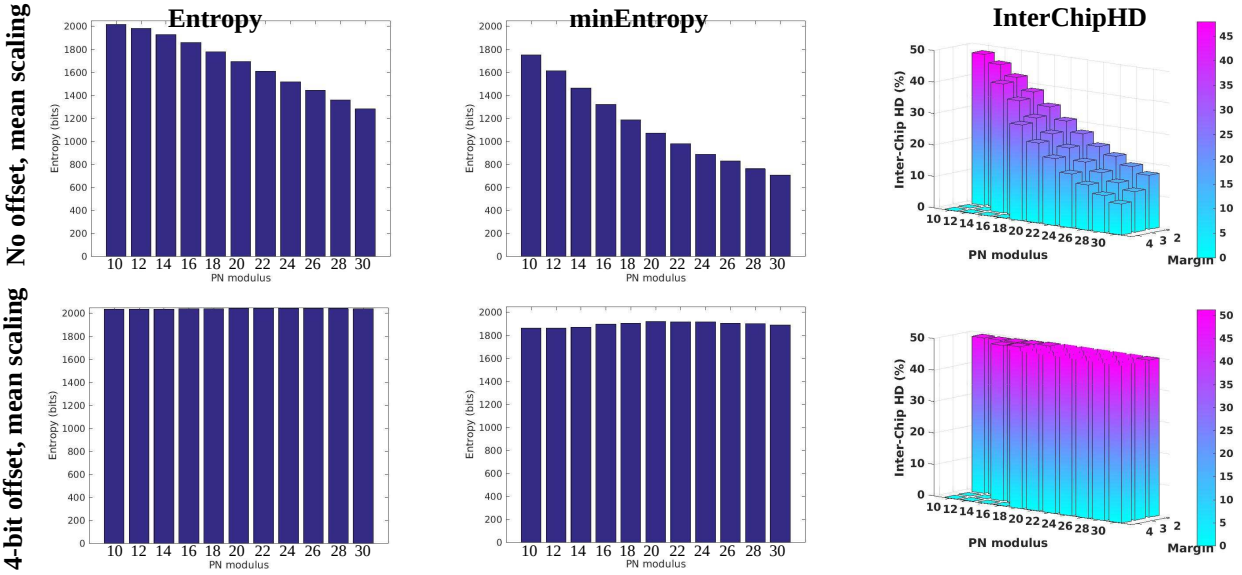

Fig. 7. Entropy, minEntropy and InterChipHD bar graphs for moduli 10 through 30. Top row shows results using the original data while 2nd row shows results using a 4-bit offset. Mean scaling refers to values that are assigned to μ_{ref} and Rng_{ref} in the *TVComp* processing. Results using other scaling factors yield similar results. For moduli 10 and 12, the offset method provides only a small benefit. As the value of the modulus increases, the Entropy-enhancing characteristic of the offset method becomes increasingly significant.

shown in the first row and between 2037 to 2043 with the 4-bit offset. The results using offsets are close to the ideal value of 2048 and are nearly independent of the modulus. Similarly, for minEntropy, the 'No Offset' results vary from approx. 700 for large moduli up to approx. 1750 for a modulus of 10. On the other hand, minEntropy using the 4-bit offset method vary from 1862 at moduli 12 up to 1919, which indicates that each generated bit has between 91% and 93.7% of a full bit of Entropy in the worst case.

The third column gives the results for inter-chip Hamming distance (InterChipHD), again computed using the bitstrings from 500 chips, 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 is 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 *BS* 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 *BS* bits from 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 $modPND_{co}$ from C_x and C_y are being used in the InterChipHD calculation.

InterChipHD is computed using Eq. 7. The symbols *NC*, NB_a and *NCC* represent 'number of chips', 'number of bits' and 'number of chip combinations', resp. (NCC is 124,750

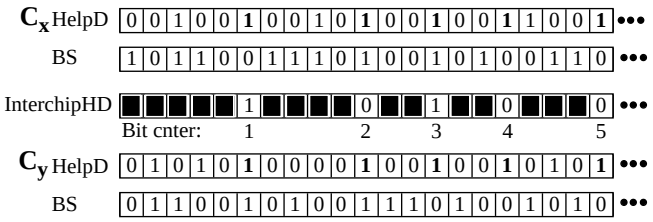

Fig. 8. Hamming distance illustration for results shown in Fig. 7.

$$\text{InterChipHD} = \left(\frac{1}{NCC} \sum_{i=1}^{NC} \sum_{j=i+1}^{NC} \frac{\left(\sum_{k=1}^{NB_a} (\text{BS}_{i,k} \oplus \text{BS}_{j,k}) \right)}{NB_a} \right) \times 100 \qquad \textbf{Eq. 7.}$$

as indicated above) This equation simply sums all the bitwise differences between each of the possible pairing of 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. 7, which varies for each pairing of chips *a*. The InterChipHD is computed separately for each of the 10 seed pairs and the average value is given in Fig. 7(c). The InterChipHD vary from approx. 10% to 48% without the offset (first row) and between 49.4% to 51.2% with the offset (second row), again showing the significant improvement provided by the offset method, particularly for larger moduli.

The bitstrings used as input for the InterChipHD analysis are also subjected to the NIST statistical tests [23]. The size of the bitstrings allowed 10 of the 15 tests to be run.

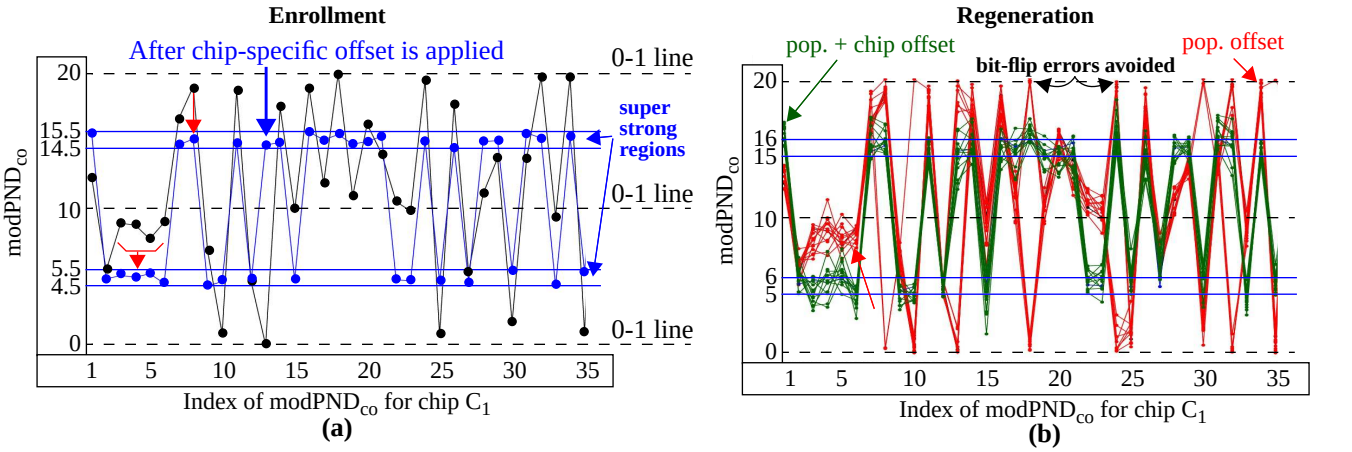

Fig. 9. Illustration of modifications made to $modPND_{co}$ before and after the chip-specific offset is applied, showing improved resilience to bit flip errors. (a) Black curve is derived from chip C_1 after population-based offset is applied while the blue curve adds chip-specific offsets. (b) Regeneration $modPND_{co}$ which use population-based and chip-specific offsets. Red curves show $modPND_{co}$ from chip C_1 at 12 regeneration TV corners using only population-based offsets while green curves show $modPND_{co}$ for 12 TV corners after applying both population-based and chip-specific offsets and several examples of bit-flip errors that are avoided.

NIST requires that at least 488 bitstrings of the 500 (one for each chip) pass each of the tests in order to qualify as an overall pass. The bitstrings pass all but two of the tests, but the fails occur by a narrow margin with 487 and 482 bit-strings passing.

5 Chip-specific Offset Method

As discussed earlier, the population-based offset method described above leaks no information regarding the bit value encoded by the $modPND_{co}$, and adds significantly to the Entropy of the bitstring. Although not presented, the probability of a bit-flip error varies from $10^{-3.4}$ to 10^{-7} and is nearly the same with or without the offsets. However, the size of the strong bitstrings decreases by approx. 2x to 5x when compared to the 'no offset' case because the center of PND_c populations are moved over the 0-1 line, and the density of the PND_c is largest at the center of the distributions. Therefore, the bits generated by a larger fraction of the PND_c are classified as weak.

The chip-specific offset method presented in this section addresses these issues. **The objective of the chip-specific offset method is to reduce bit-flip errors and increase the length of the strong bitstrings.** Unlike the population-based offset method, the chip-specific offset method is applied to the PND_c for each chip, and therefore has the potential to leak information regarding the value of the corresponding bits. The amount of leakage is related to the size of the modulus, with moduli smaller than the range of within-die variations eliminating leakage completely. Therefore, larger moduli need to be avoided. In particular, the average range of within-die variations in our 500 chip sample is 23. The analysis presented in the following restricts the modulus to be <= 20.

The chip-specific offset method is orthogonal to the population-based method and can be used in combination with it to produce the best results as we show in this section. Note that the combined method requires only one set of offsets to be transmitted to the token and is therefore similar in overhead to either of the individual methods.

The objective of the chip-specific method is illustrated in Fig. 9(a) using a small subset of the $modPND_{co}$ produced by chip C_1 under Enrollment conditions. The black curve represents the $modPND_{co}$ after population-based offsets are applied (pop. offset). The chip-specific method applies a second offset to these $modPND_{co}$ to shift them to one of two regions labeled *super strong regions*. The blue curve depicts the $modPND_{co}$ with both offsets applied. The super strong regions are located in the center of the 0 and 1 regions, i.e., at vertical positions furthest from the 0-1 lines, and therefore they represent the positions that maximally protect against bit-flip errors. The server adds the population-based offsets to the chip-specific offsets to produce a final set of offsets. Although not apparent from this analysis, the Offset Resolution (OR) given by Eq. 4 is doubled in magnitude to allow the server to specify offsets that traverse the entire modulus range. For example, the offset resolution for a 4-bit offset and a modulus of 20 is increased to $20/2^4 = 20/16 = 1.25$.

During regeneration, the server transmits the offsets to the token as a component of the challenge and the token applies them to the regenerated PND_c. The red curves in Fig. 9(b) show the $modPND_{co}$ using population-based offsets only while the green curves show the $modPND_{co}$ under the combined scheme. There are 12 curves in each group, one for each of the 12 temperature-voltage corners used in the hardware experiments. The combined offset scheme provides additional resilience against bit-flip errors (several bit-flip error cases under a zero margin scheme are identified in the figure).

Note that the enrollment helper data under the chip-specific offset (and combined) methods is all 1's, i.e., all enrollment $modPND_{co}$ are in strong bit regions, and therefore it does not need to be transferred to the token in the DualHelperData (DHD) scheme. However, the helper data generated by the token commonly has both 0's and 1's because the regenerated $modPND_{co}$ can fall within a weak region. Therefore, the DHD scheme can be layered on top of the offset methods to further improve reliability.

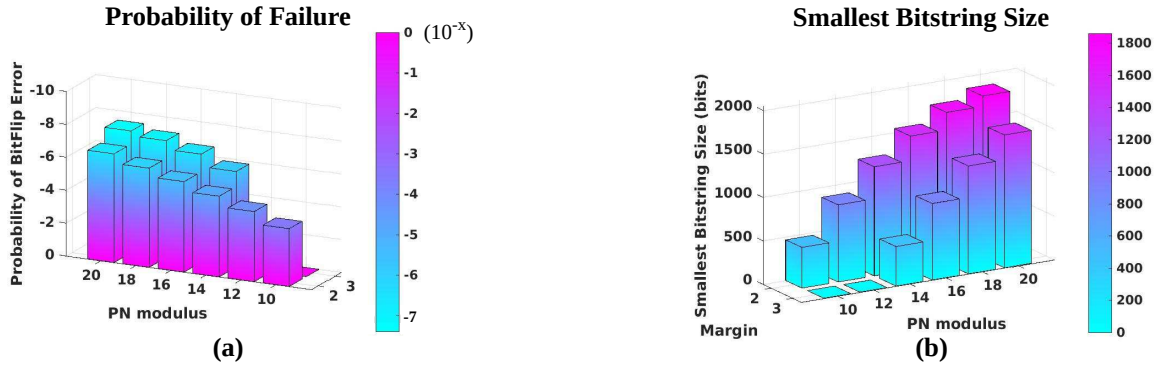

Fig. 10. (a) Probability of Failure (left) and Smallest Bitstring Size (right) results using the combined population-based and chip-specific offset methods with the DualHelperData scheme. Probability of failure is less than $10^{-5.9}$ for moduli 18 and 20 using a margin of 2 and for all moduli using a margin of 3. Results obtained from 500 copies of the Xilinx Zynq 7020 tested under Enrollment and 12 temperature-voltage corners from -40°C to 85°C and +/- 5% supply voltage. (b) The range of smallest bitstring sizes is between 450 and 1850 bits.

5.1 Population-based and Chip-specific Offset Methods

The statistical results presented in Fig. 10 are derived using data collected from 500 copies of the Xilinx Zynq 7020 tested under Enrollment and 12 temperature-voltage corners from -40°C to 85°C and +/- 5% supply voltage. The Probability of Failure (POF) is reported as an exponent *x* from 10^{-x} with a value of -6 indicating 1 chance of a bit-flip error in 1 million bits generated. The smallest bitstring size is the length of the smallest bitstring produced by one of the chips under the DHD scheme.

A probability of a bit-flip error is less than $10^{-5.9}$ for moduli 18 and 20 and a margin of 2 and for all moduli using a margin of 3. The probability of failure without offsets (not shown) is nearly the same at small moduli but decreases by more than 10x as the moduli increases. The benefit of the offset scheme are universally better than the results without offsets with regard to the smallest bitstring size. For example, the smallest bitstring size is lower bounded by 500 bits in Fig. 10. The smallest bitstring sizes for the analysis that does not use offsets are 10 to 20 times smaller.

6 Comparative Overhead Analysis

6.1 PUF bits required per full bit of Entropy

The overhead associated with ECC methods is typically reported as the number of PUF bits required to generate a fixed size bitstring. Colombier et al. collate the results reported in previous work for a variety of ECC techniques [12]. A subset of their analysis is replicated in the first 4 columns of Table 1 for comparison with the HELP results. Only those techniques with failure rates similar to those obtained for HELP are included (ECC techniques from [12] that report failure rates of 10^{-9} are excluded). We *normalized* the values shown in column 4 by dividing the corresponding values from [12] by 128 and instead report the number of PUF bits required to generate a full bit of Entropy. The HELP algorithm described in previous sections always processes 2048 PND and produces final bitstrings that are always larger than 128 bits (see Fig. 10(b)). Therefore, the normalization to 'PUF bits required per full bit of Entropy' allows the overhead of all techniques to be directly compared.

The right side of Table 1 gives the results for 7 margin (Mar.) and modulus (Mod.) combinations for HELP. The Probability of Failure (POF) data shown in column 7 is obtained from Fig. 10(a). Column 8 reports the corresponding 'minEntropy per bit' results for each of the margin/modulus combinations (note that the minEntropy results shown in Fig. 7 are computed across chips and not across bitstrings, and therefore cannot be used). The 'PUF bits required per full bit of Entropy' values in column 9 are computed by multiplying the smallest bitstring sizes shown in Fig. 10(b) by the minEntropy values from column 8, inverting these products and then multiplying by 2048. The values range from approximately 1.1 to 2.4 bits, which are smaller than all the values shown in column 4 for the ECC methods.

6.2 Helper data bits required per full bit of Entropy

The 10th column of Table 1 reports the number of bits of helper data that are required to generate a full PUF bit of Entropy. The *bit-flip-avoidance* scheme implemented within HELP creates 2048 bits of Helper data for each group of 2048 PND that are processed. The population-based and/or chip-specific offset techniques require 2048*4 = 8192 offset bits (using an offset resolution of 4 bits). Therefore, the Helper data overhead for the 2048 PUF bits is 10,240 bits, or 5 Helper data bits per PUF bit. The values in column 10 can be obtained directly by multiplying the values from column 9 by 5.

6.3 Overhead Associated with Hardware Implementation of HELP

The hardware resources for the individual modules of the HELP engine implemented on Xilinx Zynq 70x0 FPGAs are reported in Table 2. The offset methods are implemented within the Modulus module, which is labeled "Offset and Modulus' in column 5. The technique is trivially implemented on the token, adding less than 20 LUTs to original Modulus-only module. The resources are reported as the number of LUTs, FFs and nets. Many papers report slices instead of LUTs, as in [12]. The number of slices can be obtained by dividing the number of LUTs from Table 2 by 4.

7 Conclusions

Population-based and chip-specific offset methods are described in this paper. The population-based offset method improves Entropy significantly by shifting path delay distributions such that the generated bitstrings have nearly equal numbers of 0's and 1's. The tuning is designed to center the populations over the 0-1 lines used during the bitstring generation process, as a means of increasing the Entropy per bit

Table 1: Efficiency and Overhead Comparison of Proposed Method with Several ECC Methods

ECC methods [12]				HELP method					
Ref.	**Construction and codes**	**Failure Rate (POF)**	**PUF bits required per full bit of Entropy**	**Mar.**	**Mod.**	**Failure Rate (POF)**	**min Entropy per bit**	**PUF bits required per full bit of Entropy**	**Helper data bits required per full bit of Entropy**
[14]	Complementary IBS with Reed-Muller(2, 6)	10^{-6}	> 12	3	16	$10^{-6.5}$	0.974	2.37	11.85
[15]	BCH(255, 21, 55)	$10^{-6.97}$	13.94	3	18	$10^{-7.0}$	0.977	1.65	8.25
[16]	Reed-Muller(2, 6)	10^{-6}	12	3	20	$10^{-7.4}$	0.972	1.38	6.90
[17]	Concatenated: Repetition(5, 1, 5) and Reed-Muller(1, 6)	$1.49*10^{-6}$	36.25	2	14	$10^{-4.9}$	0.977	1.67	8.35
[17]	Concatenated: Repetition(11, 1, 11) Golay G_{24}(24, 13, 7)	$5.41*10^{-7}$	28.875	2	16	$10^{-5.5}$	0.974	1.36	6.80
[18]	DSC + Viterbi decoder	10^{-6}	7.6	2	18	$10^{-6.0}$	0.977	1.20	6.00
[12]	Key Reconciliation protocol	10^{-6}	8	2	20	$10^{-6.6}$	0.972	1.12	5.60

Table 2: HELP Hardware Resources.

Module	LUTs	Flip Flops (FFs)	Nets	Module	LUTs	Flip Flops (FFs)	Nets
Clock strobe	475	831	460	TVComp	421	155	91
PN storage	288	79	182	Offset & Modulus	194	67	98
Control	414	104	589	BitGen	346	117	213
PNDiff	212	101	104	**Total**	**2350**	**1454**	**1737**

Table 3: Acronyms and Definitions.

Acronym	Full Name	Acronym	Full Name
LCI	Launch-capture-interval	TVComp	Temperature and voltage compensation
PN	PUFNums	WDV	Within-die variations
PND	PN differences	UC-TVNoise	Uncompensated TV noise
PNF	Rising PN	PNDc	PND after compensation
PNR	Falling PN	PNDco	PND after compensation and offset operation
TV	Temperature and supply voltage	modPNDco	Final value after modulus operation on PNDco
SHD	Single helper data	DHD	Dual helper data
StrongBS	Strong bitstring	OB	Offset bits
OR	Offset resolution	HelpD	Helper data

toward the ideal value of 50%. The chip-specific offset method, on the other hand, is designed to reduce bit-flip errors and to increase the length of the strong bitstrings. Both offset methods are low in overhead and their effectiveness is demonstrated using hardware data collected from a set of FPGAs.

8 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. 2016.

[2] G. Suh and S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation”, *DAC*, 2007, pp. 9-14.

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

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

[5] M. Majzoobi, F. Koushanfar and S. Devadas, “FPGA PUF using Programmable Delay Lines”, *WIFS*, 2010, pp. 1–6.

[6] T. Machida, D. Yamamoto, M. Iwamoto and K. Sakiyama, “A

New Mode of Operation for Arbiter PUF to Improve Uniqueness on FPGA", *FedCSIS*, 2014, pp. 871–878.

[7] T. Machida, D. Yamamoto, M. Iwamoto, K. Sakiyama, "Implementation of Double Arbiter PUF and Its Performance Evaluation on FPGA", *ASPDAC*, 2015, pp. 6-7.

[8] A. Maiti and P. Schaumont, "Improving the Quality of a Physical Unclonable Function Using Configurable Ring Oscillators", *FPL'09*, 2009, pp. 703-707.

[9] C.-E. Yin and G. Qu, "Improving PUF Security with Regression-based Distiller", *DAC*, 2013.

[10] D. Forte and A. Srivastava, "On Improving the Uniqueness of Silicon-Based Physically Unclonable Functions Via Optical Proximity Correction", *DAC*, 2012, pp. 96-105.

[11] D. P. Sahoo, P. H. Nguyen, R. S. Chakraborty, and D. Mukhopadhyay, "Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants", *IACR Cryptology ePrint Archive*, Report 2016/057, 2016.

[12] B. Colombier, L. Bossuet, David HÈly, and V. Fischer, "Key Reconciliation Protocols for Error Correction of Silicon PUF Responses", *Trans. Information Forensics and Security*, Vol. 12, No. 8, 2017, pp. 1988-2002.

[13] J. Guajardo, S. Kumar, G.J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and their use for IP Protection", *CHES*, 2007, pp. 63-80.

[14] M. Hiller, D. Merli, F. Stumpf, and G. Sigl, "Complementary IBS: Application Specific Error Correction for PUFs", *HOST*, 2012, pp. 1-6.

[15] A. V. Herrewege et al., "Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs", *Conf. Financial Cryptography Data Secur.*, 2012, pp. 374-389.

[16] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs", *CHES*, 2009, pp. 332-347.

[17] C. Bˆsch, J. Guajardo, A. Sadeghi, J. Shokrollahi and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs", *CHES*, 2008, pp. 181-197.

[18] M. Hiller, M. Yu, and G. Sigl, "Cherry-Picking Reliable PUF Bits with Differential Sequence Coding", *Trans. Inf. Forensics Security*, Vol. 11, No. 9, 2016, pp. 2065-2076.

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

[20] J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard", Springer, Heidelberg, Berlin Germany, 2002.

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

[22] G. Pocklassery, V. K. Kajuruli and F. Saqib and J. Plusquellic, "Physical Unclonable Functions and Dynamic Partial Reconfiguration for Security in Resource-Constrained Embedded Systems", *HOST*, 2017.

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