A Privacy-Preserving, Mutual PUF-Based Authentication Protocol

Wenjie Che, Mitchell Martin, Goutham Pocklassery,
Venkata K. Kajuluri and and Jim Plusquellic
ECE, Univ. of New Mexico
Albuquerque, NM
wjche@unm.edu, jimp@ece.unm.edu

Fareena Saqib
ECE, Florida Institute of Technology
Melbourne, FL
(fsaqib@fit.edu)

Abstract -- **This paper describes an authentication protocol using a Hardware-embedded Delay PUF called HELP. HELP derives randomness from within-die path delay variations that occur along the paths within a hardware implementation of a cryptographic primitive, such as AES or SHA-3. The digitized timing values which represent the path delays are stored in a database on a secure server (verifier) as an alternative to storing PUF response bitstrings. This enables the development of an efficient authentication protocol that provides both privacy and mutual authentication. The security properties of the protocol are analyzed using data collected from a set of Xilinx Zynq FPGAs.**

Keywords -- Physical Unclonable Function, Authentication Protocol, FPGA Implementation

1. INTRODUCTION

Authentication is the process between a prover, e.g., a hardware token or smart card, and a verifier, a secure server or bank, that confirms the identities, using corroborative evidence, of one or both parties [1]. With the Internet-of-things (IoT), there are a growing number of applications that require low cost authentication [2]. Physical unclonable functions (PUFs) are hardware security and trust primitives that can address issues related to low cost because they can potentially eliminate the need for NVM. Moreover, the special class of so-called 'strong PUFs' can also reduce area and energy overheads by reducing the number and type of cryptographic primitives and operations [3].

A PUF extracts randomness 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 bitstrings are generated on-the-fly, thereby eliminating the need to store digital copies of them in NVM, and are (ideally) reproducible under a range of environmental variations. The ability to control the precise generation time of the secret bitstring and the sensitivity of the PUF entropy source to invasive probing attacks (which act to invalidate it) are additional attributes that make them attractive for authentication in embedded hardware.

Most proposed PUF architectures require the insertion of a dedicated array of identically-designed test structures and are classified as 'weak PUFs'. Although weak PUFs can be used for authentication, they require cryptographic functions, e.g., secure hash and encryption, to exponentially expand the input/output space of challenge-response-based authentication protocols. A strong PUF, on the other hand, can generate, ideally, an exponential number of challenge-response-pairs (CRPs), and can potentially be configured to allow direct, unprotected access from outside the chip. This is true because it is infeasible for an adversary to apply all 2^n CRPs in an attempt to read-out and store all of the response bitstrings. The arbiter PUF is traditionally regarded as the first strong PUF because it can be configured to produce 2^n responses [4].

Strong PUFs with unprotected interfaces, however, must be able to withstand model-building attacks which attempt to machine learn (ML) the relationship among the much smaller number of random circuit elements, from which the 2^n response bits are generated. The arbiter PUF, for example, is typically configured with as few as 256 logic gates, making it susceptible to ML attacks [5].

In this paper, we propose a hardware-embedded Delay PUF (HELP) [6] as the basis for a novel authentication protocol. The entropy source of HELP is based on path delay variations that occur in the structural paths of an on-chip macro. In particular, we use data path components from a hardware implementation of the AES algorithm as the source of delay variations.

HELP accepts 2-vector sequences as challenges and supports an exponential input challenge space, i.e., with *n* inputs, the number of challenges is upper bounded at 2^{2n}, which indicates that any of the 2^n input vectors can be followed by any of the other 2^n input vectors. In order to improve the reliability of HELP, we constrain the 2-vector sequences to generate either rising transitions or falling transitions along the paths, but not both. This reduces the challenge space from 2^{2n} to $2*(3^n-2^n)$, which is still an exponential as required of a strong PUF. However, the number of unique paths is typically a smaller exponential 2^m, which indicates that the 2-vector sequences re-test these paths approx. $2*(3^n-2^n)/2^m$ number of times on average. If the response space is defined as 2^m, then *m* needs to be on order of 64 or larger to meet the conditions of a strong PUF. Although combinational logic circuits can be constructed to meet this condition, the resulting size is too large for resource-constrained devices.

To address this issue, we expand the response space of

HELP by defining a set of configuration parameters. The combination of the 2-vector sequences and these parameters increases the CRP space to a large exponential. For example, one of the configuration parameters is called the *Path-Select-Mask*. It allows the verifier to select a specific subset of the paths, from those tested by the applied 2-vector sequences, to be used in the bitstring generation process. By itself, the *Path-Select-Mask* adds an *n*-choose-*k* number of possibilities to the size of the response space. The values of *n* and *k* are typically in the range of 5000 and 2048, resp, which corresponds to a value larger than $3e^{1467}$.

HELP possesses a second distinguishing characteristic beyond those found in conventional PUF definitions. The paths defined by the functional unit have a complex interconnection structure requiring long runtimes of automatic test pattern generation (ATPG) software to determine the 2-vector sequences required to test them. The difficulty of generating challenges for HELP adds a new dimension to the difficulty of carrying out model-building attacks because the adversary must first expend a great deal of effort to determine the challenges that enable an effective model-building strategy. It can be argued that this effort only needs to be expended once for a given implementation but depending on the test generation strategy and the netlist characteristics, it may be infeasible to compute the required tests in a reasonable amount of time. Note that this characteristic is only a disadvantage for the adversary. The trusted authority can pick-and-choose which paths to target for test generation (only a limited number of CRPs are stored in the secure database), and therefore, test generation time can be kept small.

1.1 Characteristics of PUF-based Authentication Protocols

The simplest form of a PUF-based authentication protocol is carried out in two phases; enrollment and authentication. During enrollment (which occurs in a secure facility), the verifier randomly selects a small subset of the possible challenges and applies them to the PUF to generate a corresponding set of responses. The CRPs for each token are recorded by the verifier in a secure database. The CRPs are later used for authenticating the fielded token. The number of stored CRPs for each token can be relatively small because the large CRPs space of a strong PUF along with the secrecy of the selected subset make it very difficult for adversaries to build a clone to impersonate the token.

However, this simple model has several drawbacks. First, it does not provide privacy for the authenticating token, and therefore, adversaries will be able to track a fielded token across successive authentications. This is true because the token must first identify itself to the verifier using some type of token-ID to enable the verifier to select the proper CRP set. The token-ID is required because only a small, undisclosed, subset of the CRPs are recorded on the verifier for each token during enrollment. The token-ID must also be stored permanently on the token, e.g., 'burned in' using fuses, and must be sent in the clear. CRP chaining and encryption schemes have been proposed to avoid this, but incur additionally overhead because they require a read-writable NVM to implement the chaining component [7].

Second, the scheme is susceptible to denial-of-service (DOS) attacks, whereby an adversary depletes the verifier's CRPs for a token by repeatedly attempting to authenticate. Third, even when DOS attacks are not attempted, the stored CRPs can be exhausted in the course of a sequence of valid authentications because the verifier must delete a CRP once it is used (to avoid replay attacks), and the verifier stores only a fixed number of CRPs for each token.

In this paper, we propose a novel PUF-based, privacy-preserving, mutual authentication protocol that overcomes these limitations. Instead of storing response bitstrings on the verifier, the protocol stores path timing information, e.g., 15-bit digitized representations of measured path delays. In combination with a set of configuration parameters, the storage of path delays provide distinct advantages over response bitstrings by enabling a very large, exponential set, of response bitstrings to be generated using a fixed set of stored path delays on the verifier.

This paper builds on the work described in [6] and [8]. The novel contributions of this paper over previous work are:

- A complete end-to-end privacy-preserving, mutual PUF-based authentication protocol.
- A novel Dual-Helper-Data reliability-enhancing method.
- A hardware data analysis and demonstration of the authentication protocol on a set of Xilinx Zynq FPGAs.
- Analysis of the proposed protocol's bitstring and hardware implementation characteristics.

This paper is organized as follows. Related work is presented in Section 2. HELP is reviewed in Section 3 and the proposed PUF-based authentication protocol is presented in Section 4. Experimental results are presented in Section 5, a Security Analysis is Section 6 and Conclusions in Section 7.

2. Related Work

The authors of [9] propose the use of delay variations in functional units for authentication. However, the scheme makes use of the timing values directly, and does not account for path length bias effects. Moreover, the proposed authentication scheme is incomplete.

An improved ownership transfer and mutual authentication RFID protocol is proposed in [10]. The authors in [11] introduce a conditional privacy-preserving authentication scheme for Ad hoc Networks. A mutual authentication scheme is proposed in [12] for the Fog-Cloud network architectures.

A excellent recent survey has been published which summarizes the state-of-the-art in PUF-based authentication protocols [14-29] for resource-constraint devices [13]. The authentication protocols covered by the survey are evaluated according to: (1) resilience to environmental noise, (2) resilience to machine learning attacks, (3) the need to expand the response space of the strong PUF and (4) resilience to protocol attacks. The authors of [13] conclude that the main weakness in existing protocols relates to weaknesses in the PUF's entropy

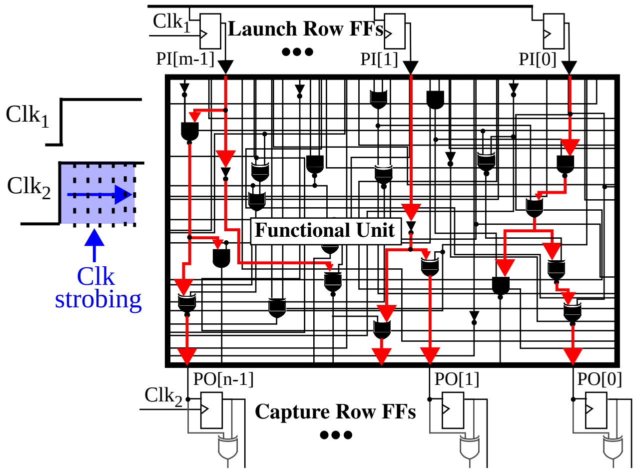

Fig. 1. Configuration of the functional unit and clock strobing method for measuring path delays.

source and that future research should focus on developing 'a truly strong PUF with great cryptographic properties'.

A prototype of a provably secure protocol is recently proposed in [7] that supports privacy-preserving and mutual authentication. The protocol makes use of a weak SRAM PUF, and requires NVM and several cryptographic functions to be implemented on the token. Their follow-up work in [30] makes use of an ASIP processor architecture for implementing compact and low-power authentication protocols on FPGAs. Resource utilization of the ASIP implementation is very small, approx. 250 LUTs and FFs, but excludes the PUF core, so it is difficult to carry out a direct comparison with the resources reported in this paper for HELP. We will investigate the proposed ASIP architecture for implementing the HELP PUF and protocol operations in a future work.

3. HELP Overview

The source of entropy for HELP is the manufacturing variations that occur in the delays of paths that define an on-chip functional unit, as shown in Fig. 1. In this paper, the functional unit 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**) [31]. This combinational data path component is implemented in a WDDL logic style [32], which doubles the number of primary inputs (PIs) and primary outputs (POs) 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*, we have also recently demonstrated the protocol using a lighter-weight functional unit consisting of single AES SBOX component that possesses approx. 600 LUTs, reducing the overall implementation size (HELP + functional unit) from approx. 6000 LUTs to less than 3000 LUTs. The details of area and time overheads associated with HELP are provided in Section 5.2.

As indicated above, a challenge for HELP consists of a 2-vector sequence and a *Path-Select-Mask*. The 'Launch Row FFs' in Fig. 1 are used to apply the 2-vector sequences to the primary inputs of the functional unit, labeled *PI[]*, while the 'Capture Row FFs' are used to measure the path delays at the *PO[]*. The path delays are measured by applying a series of launch-capture clocking events (called **clock strobing**) using Clk_1 and Clk_2 as shown on the left side of Fig. 1. The first vector of the sequence represents the initialization vector. The application of the second vector generates a set of transitions which are timed by the clock strobing technique. The clock strobing technique requires the repeated application of the 2-vector sequence. For each repeated application, the phase shift between Clk_1 and Clk_2 is increased by a small fixed Δt.

The phase shift value between the two clocks is digitally controlled, and is referred to as the **launch-capture interval** (LCI)[1]. The smallest LCI that allows the propagating edge along a path starting from a Launch FF to be captured in a Capture FF (occurs when an XOR gate on the output becomes 0) is used as the digitized timing value for the path. In the following description, we refer to the LCI path timing value as a **PUFNum** or **PN**.

The authentication protocol described in Section 4 requires HELP to generate nonces in addition to the PNs. The VHDL module responsible for implementing the PN timing engine generates nonces in parallel with PN generation by leveraging the meta-stability characteristics that exist in a subset of the tested paths. Meta-stability is determined for a path by repeatedly measuring it and then analyzing the variations in the fractional component of the computed average. Those paths that produce two consecutive PN values nearly of equal frequencies are used as a source of true random numbers (**TRNG**). Although not presented in this paper, the random statistical properties associated with the nonces generated in this fashion pass all of the NIST statistical tests [33].

We generate test data in this paper by applying a set of approx. 1200 challenges to test 2048 paths with rising transitions and 2048 paths with falling transitions. HELP constructs 2048 *signed differences* from the 4096 PNs by pairing each of the rising PNs with a falling PN using two linear-feedback shift register (LFSRs). The LFSRs are initialized with a pair of configuration parameters, called *LFSR seeds*. The set of 2048 signed differences are referred to as **PND** in the following.

3.1 TV Compensation (TVCOMP)

The reliability of a PUF refers to the number of bit flip errors that occur when the bitstring is regenerated. Ideally, the bitstrings are precisely reproduced during regeneration but this is rarely possible with PUFs. The largest source of 'noise' that causes bit flip errors for PUFs is a change in temperature and/or supply voltage (**TV noise**). Although sample-averaging of path delays is effective at reducing measurement noise, this strategy is not effective for TV noise, and instead a TV compensation (TVCOMP) method is required. The TVCOMP pro-

1. The ability to dynamically control the fine phase shift of a Clk signal is a common feature of on-chip digital clock managers (DCMs) in FPGAs.

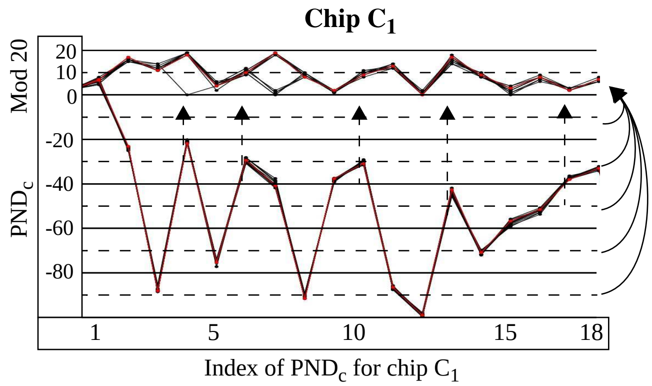

Fig. 2. Sample of (a) PNDc (bottom) and corresponding modPND$_c$ (top).

cess that we propose is described by Equations (1) and (2).

$$\text{zval}_i = \frac{(\text{PND}_i - \mu_{\text{token}})}{\text{Rng}_{\text{token}}} \quad \textbf{(1)}$$

$$\text{PNDc} = \text{zval}_i \text{Rng}_{ref} + \mu_{ref} \quad \textbf{(2)}$$

Here, $zval_i$ represents a standardized PND after subtracting a mean μ_{token} and dividing by a range Rng_{token}, with $\mu_{token}$and Rng_{token} derived from the distribution of all PND obtained during regeneration under potentially adverse environmental conditions, referred to as TV corners. The individual $zval_i$ are then transformed to a set of **PNDc** (with ‘c’ for compensated) using two additional configuration parameters, μ_{ref} and Rng_{ref} (*ref* for reference). This linear transformation is very effective at reducing TV noise. The noise from environmental variations that remain in the PNDc is called *uncompensated TV noise* or **UC-TVNoise**.

3.2 BitString Generation Algorithm

The bitstring generation process uses the signed PNDc as a means of both hardening the algorithm against model building and increasing the diversity in the PUF responses. A **modPNDc** is defined by applying a *Modulus* to the PNDc. The Modulus is a fifth configuration parameter to the HELP algorithm (adding to the μ_{ref}, Rng_{ref} and *LFSR seeds* parameters). The modulus is necessary because the paths in the functional unit vary in length and this path length bias is captured in the PNDc. The modulus reduces the bias while fully preserving the within-die delay variations, i.e., the most important source of randomness.

Fig. 2 shows a sample set of 18 PNDc computed from pseudo-random pairings of PN measured from chip C_1. Each PNDc is measured 16 times under different TV conditions. The red curve line-connects the data points obtained under **enrollment** conditions (25^{o}C, 1.00V) while 15 black curves line-connects data points under a set of **regeneration** TV corners, which in our current experiments is all combinations of temperatures -40^{o}C, 0^{o}C, 25^{o}C, 85^{o}C, 100^{o}C with supply voltages 0.95V, 1.00V and 1.05V. The curves plotted along the top of Fig. 2 show the modPND$_c$ values after a modulus of 20 is applied. The modPND$_c$ are used in HELP’s bitstring generation procedure described below.

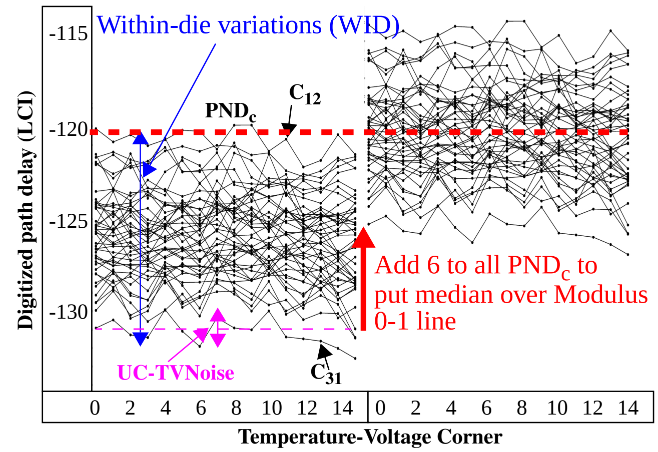

Fig. 3. TVCOMP’ed PND$_c$ for 45 chips (individual curves) and 16 TV corners (points in curves) illustrating goal of the offset technique.

3.3 A Simple Entropy Enhancing Technique

We recently developed an ‘offset’ technique that can be used to further reduce bias effects, particularly when the Modulus is greater than the magnitude of the within-die variations. Fig. 3 provides a plot of a PND$_c$ obtained from a set of 45 chips to illustrate the concept. The line connected points in each curve are generated by the same chip and represent the value of the PND$_c$ measured in the 16 TV corner experiments after they has been TVCOMP’ed. The UC-TVNoise referred to earlier that remains after TVCOMP is annotated on the bottom-most curve. In contrast, within-die variations (WID) are represented by the vertical extension of the individual curves, which is also annotated in the figure. The magnitude of WID for this PND$_c$ is approx. 11 LCIs.

If a Modulus of 20 is used, then the position of this group of curves, shown between -131 and -120, represents a worst-case scenario because the bit generated in the bitstrings (discussed below) would be the same for nearly all chips. The bias that creates this problem can be eliminated by adding a constant of 6 to the points in the all curves (see right side of Fig. 3). This ‘centers’ the PND$_c$ distribution over -120 and maximizes the entropy contained in this PND$_c$ by making the number of chips which produce a ‘1’ in the generated bitstrings nearly equal to the number that produce a ‘0’. The appropriate offset is computed by the verifier using the stored enrollment data and is encoded in the set of *Path-Select-Mask* sent to the token.

3.4 BitString Generation with Margining and Dual Helper Data

We propose a **Margin technique** as a method to improve reliability. The Margin technique identifies modPNDc that have the highest probability of introducing bit flip errors. The modPNDc data shown along the top of Fig. 2 is replicated and

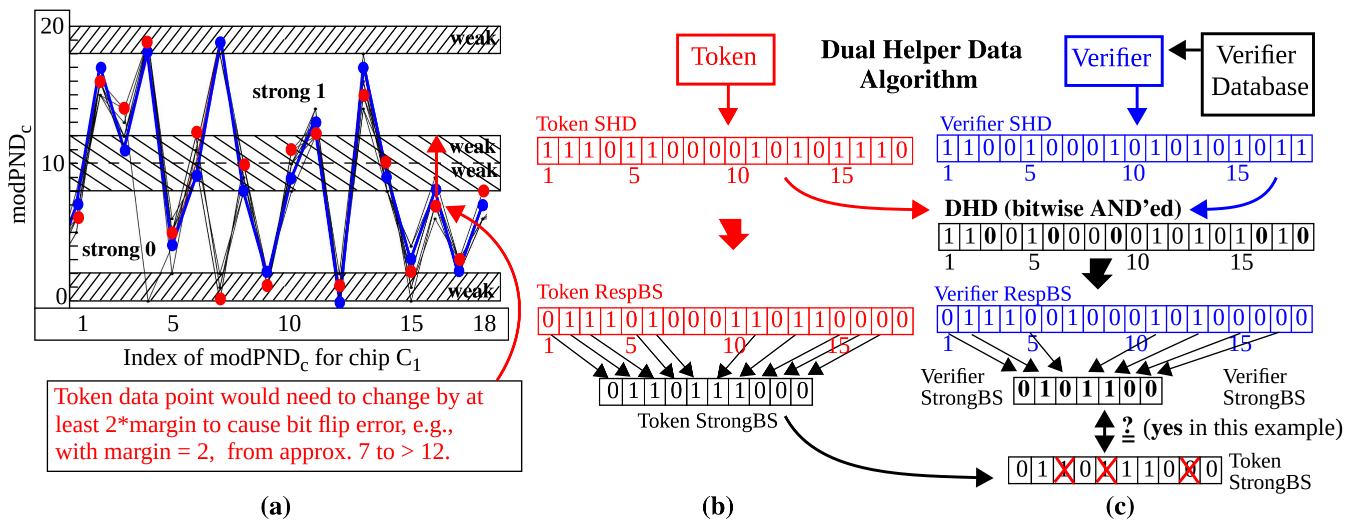

Fig. 4. Margin and Dual Helper Data Algorithm Illustration.

enlarged in Fig. 4(a) to serve as an illustration. The region defined by the Modulus is split into two halves, with the lower half used as the '0' region (between 0 and 9 in the figure) and the upper half as the '1' region.

Without Margining, bit flips would occur at modPND$_c$ indexes 4, 6, 7, 8, 10 and 14 because some of the values in the groups of PND$_c$ data points from the 16 TV corner experiments cross over the 0-1 lines at 9-10 and 19-0. The Margin technique avoids these bit flip errors by creating **weak** and **strong** classes for the bits associated with the modPNDc. The bit associated with a modPND$_c$ is classified as ***weak*** if the modPND$_c$ falls within a margin around the 0-1 boundaries, and is classified as a ***strong bit*** otherwise. The margin is set ideally to the worst case UC-TVNoise level for the best results, but can be tuned to attain a specific probability of failure in the authentication protocol as we will show.

A novel *Dual Helper Data* (DHD) scheme is proposed as a means of further reducing bit flip errors. The DHD technique is described in the context of our proposed authentication protocol in advance of its full description in Section 4. Fig. 4(b) shows the helper data (**HelpD**) and response bitstrings (**RespBS**) for the hardware token while Fig. 4(c) shows them for the verifier. The values are derived using the red (token) and blue (verifier) highlighted data points from the modPNDc graph in Fig. 4(a). Authentication in the field makes use of data stored earlier during enrollment in the Verifier Database. The following operations are carried out to generate the Token and Verifier **StrongBS**:

- The token generates helper data (Token HelpD) using the Margining technique to produce the Token StrongBS, which are both transmitted to the verifier.
- For each token stored in the Verifier Database, the verifier computes helper data (Verifier HelpD), and then bitwise AND's it with the received Token HelpD.
- The verifier constructs the Verifier StrongBS using the AND'ed HelpD while simultaneously eliminating strong bits from the Token's StrongBS that correspond to Token HelpD bits that were changed from '1' to '0' during the AND operation (3 bits are eliminated in this example as shown along the bottom of Fig. 4(c)).
- The two StrongBS are compared. A successful authentication requires either an exact match between the Token and Verifier StrongSB, or a 'fuzzy match' where a match is successful if most, but not all, of the bits match.

The AND'ing of the token and verifier's HelpD bitstrings allows the margin to be reduced to approx. one-half of that required if the individual HelpD bitstrings were used by themselves. This is true because a bit flip error can only occur if UC-TVNoise causes a modPNDc to move across both margins, and into the opposite **strong bit region**, as shown by the caption and illustration in Fig. 4(a). If the modPNDc moves but remains in either the '1' or '0' **weak bit regions**, then the AND operation will eliminate it. For example, a bit-flip error associated with the 6^{th} modPND$_c$ in Fig. 4(a) that would have occurred under the SHD scheme is avoided in the DHD scheme. As we will show, the smaller margins used with the DHD scheme allow the Modulus to be reduced, which in turn, allows better access to within-die variations.

4. Authentication Protocol

A privacy-preserving, mutual authentication protocol is presented in this section. As indicated above, we propose to store path delay information, the PNs, on the verifier instead of response bitstrings. The PNs can each be represented as a 15-bit values (which provides a range of +/- 1024 with 4 bits of fixed-point precision). The protocol employs several parameters, including a *Modulus*, a μ_{ref} and Rng_{ref} from Equations (1) and (2), a pair of *LFSR Seeds*, a *Margin* and a *Path-Selection-Mask*, to allow multiple response bitstrings to be generated from a fixed set of PNs. The verifier specifies a set of paths in the *Path-Select-Mask* and encodes offsets in the unused bits to improve entropy as discussed in Section 3.3.

A challenge is defined as a 2-vector sequence + a *Path-Select-Mask*. A *one-time interface* (implemented on the FPGA as a special programming bitstring) is used during enrollment

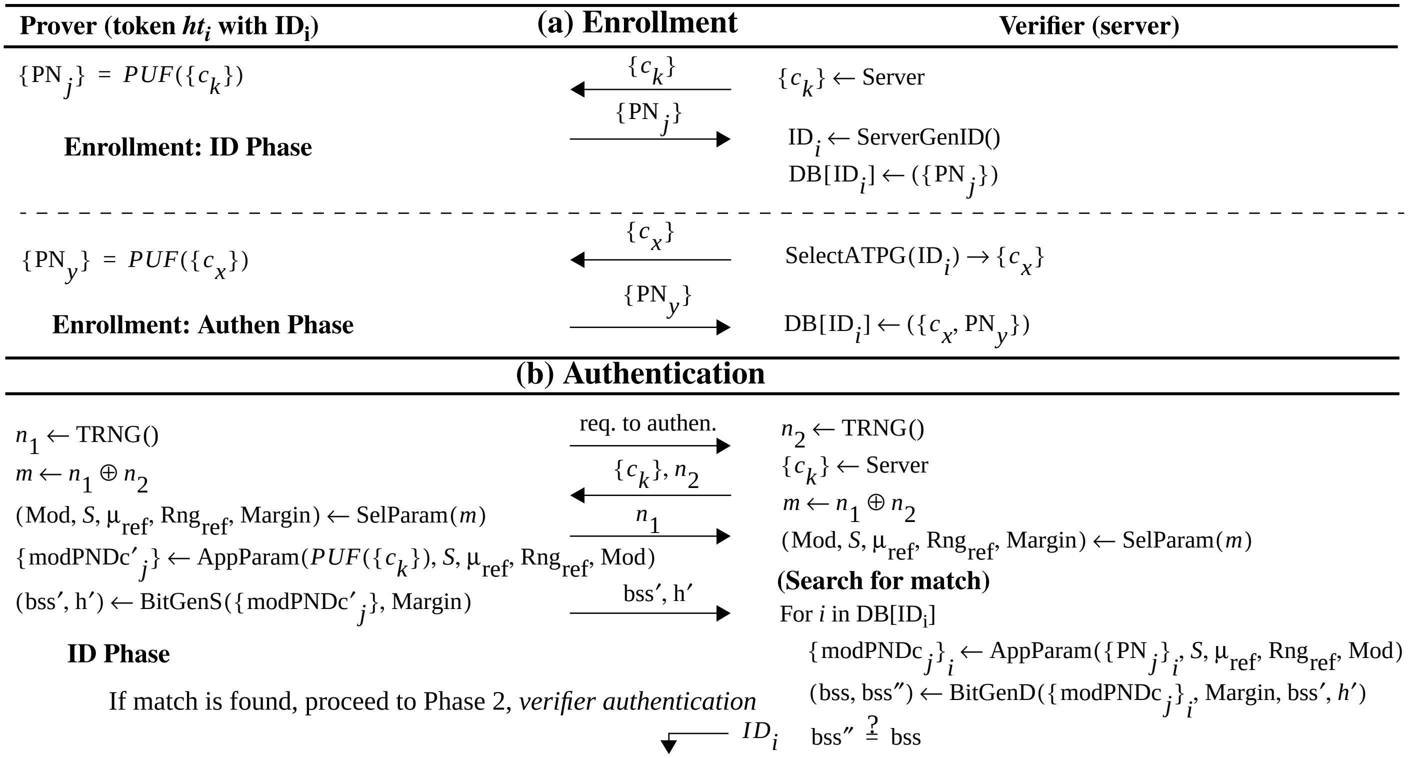

Fig. 5. Enrollment Operations (a) and Authentication Protocol for ID Phase (b).

to allow the token to transfer PNs to the verifier. The protocol separates token identification (*ID phase*) from authentication (*Authen phase*) to support the privacy preserving component. The protocol does not require any cryptographic primitives nor non-volatile memory (NVM) on the token.

The enrollment operation is graphically illustrated in Fig. 5(a). Prior to manufacture, automatic test pattern generation (ATPG) is used to select a set of test vector sequences, $\{c_k\}$, that will be used as a **common set** of challenges for all tokens in the *ID phase*. The number of vectors depends on the security requirements regarding privacy. The *sbox-mixedcol* functional unit produces 40 PNs on average per 2-vector sequence. Therefore, a set of 1000 vectors would produce approx. 40K timing values.

The common challenges are transmitted to the token in a secure environment during enrollment and applied to the functional unit's PIs. The token generated PN are transmitted to the verifier, annotated as $\{PN_j\}$ in Fig. 5(a). The verifier generates an internal identifier ID_i for each token using *VerifierGenID()* and stores the set $\{PN_j\}$ under ID_i in the secure database.

A similar process is carried out during the *Authen Phase* of enrollment except that a distinct set of ATPG-generated challenges are selected (using *SelectATPG*(ID_i)) for each token. The number of hazard-free testable paths in typical functional units can be very large (*sbox-mixedcol* has approx. 8 million paths), making it possible to create minimally overlapping sets for each token (some overlap is desirable for privacy reasons as discussed below). Note that the task of generating 2-vector sequences for **all** paths is likely to be computationally infeasible for even moderately sized functional units. However, it is feasible and practical to use ATPG to target random subsets of paths for the enrollment requirements. The set of PNs, $\{PN_y\}$, generated in the *Authen Phase* are also stored, along with the challenge vectors that are used, in the secure database under ID_i.

The fielded token authenticates using a 3-phase process, Phase 1 is *token identification* (**ID**), Phase 2 is *verifier authentication* (**Mutual**) and Phase 3 is *token authentication* (**Authen**). The operations carried out in the ID Phase are shown graphically in Fig. 5(b). The other two phases are nearly identical, with only the differences noted below.

The token initiates the process by transmitting a 'req. to authen.' signal to the verifier. The verifier generates nonce n_2 and transmits it to the token, along with a selected set of challenges $\{c_k\}$ to the token. Note that the transmitted challenges are typically a subset of those used during enrollment. The token generates a nonce n_1 and transmits it to the verifier. This strategy, first proposed in [25] for challenge selection, prevents the adversary from constructing n_2 as a means of carrying out a systematic attack.

The token and verifier compute $m = (n_1 \text{ XOR } n_2)$ and use the m as an input parameter to the *SelParam* function. *SelParam* constructs the parameters *Mod*, S, μ_{ref}, Rng_{ref} and *Margin* using bit-fields from m. The two *LFSR Seed* parameters S can be derived directly from a bit-field in m. The remaining parameters are derived using a table lookup operation as a means of constraining them to specific ranges. For example, *Mod* is lower bounded by the *Margin* and is constrained to be an even number less than 30. Similarly, μ_{ref} and Rng_{ref} parameters are constrained to a range of fixed-point values. Section 5 provides recommendations on the ranges and presents statistical results using a subset of the possible parameter combinations. *SelParam* is carried out on the verifier in the same fashion.

Once the parameters are selected, the bitstring generation process is carried out as follows:

- The challenges $\{c_k\}$ are applied to generate a set $\{PN'_j\}$, referenced as *PUF(*$\{c_k\}$*)* in Fig. 5(b).
- The *PNDiff*, *TVCOMP* and *Modulus* operations described in Sections 3, 3.1 and 3.2 are then applied to the set of PNs using the *AppParam* procedure with parameters *S*, μ_{ref}, Rng_{ref} and *Mod* parameters to generate the set $\{modPNDc'_j\}$.
- Bitstring generation (*BitGenS*) is then performed on the token using the *Margining* process described in Section 3.4, and shown graphically in Fig. 4(b). *BitGenS* returns both a bitstring *bss'* that is composed of only strong bits under the constraints of the *Margin* and a helper data string *h'*. Both *bss'* and *h'* are transmitted to the verifier.
- The verifier carries out a search process by processing each of its stored token *i* data sets $\{PN_j\}_i$ using the same parameters. However, the DHD scheme, denoted *BitGenD* in Fig. 5(b), is used instead. *BitGenD* bitwise-ANDs the token's helper data *h'* with the helper data derived for each data set (not shown), and uses it to modify the token's bitstring *bss'* to *bss"* eliminating bits as needed (see bottom of Fig. *4*(c)) and to produce the verifier's StrongBS *bss*. The verifier then compares *bss* with *bss"*, and completes the *ID Phase* successfully if a match is found.

Note that this is a compute-intensive operation for large databases because *AppParam* and *BitGenD* must be applied to each stored $\{PN_j\}_i$ in the database. However, the search operation can be carried out in parallel on multiple CPUs given the independence of the operations if needed. The runtime of the search algorithm is reported on in Section 5.

As indicated, the search terminates when a match is found or the database is exhausted. In the latter case, authentication terminates with failure at the end of the *ID Phase*. Therefore, the *ID Phase* also serves as a gateway that prevents an adversary from depleting a token's authentication information on the verifier in a denial-of-service attack.

In the former case, the ID_i of the matching verifier data set is passed to Phase 2, *verifier authentication* and Phase 3, *token authentication*. In Phase 2, the same process is carried out except the token and verifier roles are reversed and the search process is omitted. Also, the challenges used in the *ID Phase* can be re-used and only *SelParam* run using two new nonces (n_3 XOR n_4). Phase 3 is similar to Phase 1 in that the token is again authenticating to the verifier, but uses a 'token specific' set of challenges $\{c_x\}$. Similar to Phase 2, the search process is omitted (note, Phase 3 can be omitted in applications that have lower security requirements, e.g., RFID and home automation applications).

Note that token privacy is preserved in the *ID Phase* because, with high probability, the transmitted information *bss'* and *h'* will be different from one run of the protocol to the next, given the diversity of the parameter space provided by the *Mod*, *S*, μ_{ref}, Rng_{ref}, *Margin*. This diversity is exponentially increased as discussed in the Introduction through the use of the *Path-Select-Mask*. Moreover, by creating overlap in the challenges used by different tokens in the *token authentication* phase, tracking is prevented in this phase as well.

We note that the process of generating helper data on the token was proposed previously in [3], but for the purpose of addressing *error correction* issues. HELP uses an *error avoidance* scheme and therefore, the motivating factor for previously proposed *reverse fuzzy extraction* schemes, i.e., for reducing the computing burden associated with error correction on the token, does not exist for HELP. As a consequence, it is possible in HELP to implement an efficient helper data scheme in either direction, as proposed in the multiple phases of our authentication scheme.

5. Statistical Evaluation of Hardware Data

The *Mod*, *S*, μ_{ref}, Rng_{ref} and *Margin* collectively represent parameters that can be varied within limits to create distinct bitstrings from a set of measured PNs. This feature of the proposed authentication scheme offsets the increased overhead associated with storing multi-bit PNs on the verifier as an alternative to response bitstrings. However, this scheme depends heavily on high statistical quality among the generated StrongBS. This section investigates StrongBS statistical quality using the standard metrics, including Intra-chip hamming distance (HD_{intra}), Inter-chip hamming distance (HD_{inter}) and the NIST statistical test tools, as measures of bitstring reproducibility, uniqueness and randomness, resp.

5.1 Bitstring Statistical Analysis

The analysis in this section is carried out using data collected from Xilinx Zynq 7020 SoC FPGAs [34]. A set of 4096 PNs are collected from 45 chips at each of 16 TV corners. The enrollment data stored in the verifier database is collected at 25^oC, 1.00V (nominal conditions), while regeneration data is collected at all combinations of the extended industrial-grade temperature-voltage specification limits for the parts, -40^oC, 0^oC, 25^o, 85^oC and 100^oC and voltages 0.95V, 1.00V and 1.05V. A set of low-noise, high within-die variations paths are selected using *Path-Selection-Masks* from approx. 600 rising and 600 falling 2-vector test sequences.

PNDs are created using LFSR-selected pairings of the 2048 rising and 2048 falling edge PNs. Although not analyzed here, this rise-fall pairing strategy reduces TV noise while increasing the randomness among the PNDs. Each of the 2048 rising edge PNs can be paired with any of the 2048 falling edge PNs, yielding 4,194,304 possible combinations. We report

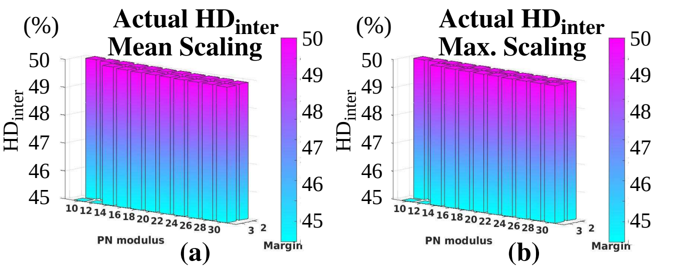

Fig. 6. Actual HD_{inter} using the StrongBS from 45 copies of the chips under Enrollment conditions using Mean and Maximum scaling factors for μ_{ref} and Rng_{ref}.

results on a subset of 256 of these pairing combinations.

A 2-bit offset scheme is applied to the PND_c to improve entropy, as discussed in 3.3. The verifier computes the offsets using stored enrollment data and uses it to shift the individual PND_c upwards by 0, 1/8, 1/4, or 3/8s the range given by the applied *Modulus* to better center the distribution over the 0-1 lines.

A set of *Moduli* between 10 and 30, in steps of size 2, and *Margins* of size 2 and 3, are also investigated, as shown along the x- and y-axes in Figs. 6 and 8 (to be discussed). Note that the bars of size 0 in the figures indicate that the analysis is not valid for these combinations of *Margin* and *Moduli*. The minimum value of the *Modulus* is given by 4**Margin* + 2 because four weak regions are required as shown by the example in Fig. 4(a) and the two strong bit regions must be at least of size 1. For example, the smallest *Modulus* for a *Margin* of size 3 is 14, so elements in the histogram for *Moduli* of 10 and 12 are 0.

Our analysis reveals that of the 20 combinations of these parameters, 17 are useful. The only combinations that cannot be used are *Modulus* of 10 for *Margin* 2 and *Moduli* of 14 and 16 for *Margin* 3. As we show, the bitstring sizes are too small for these combinations of *Margin* and *Moduli*.

Our analysis also investigates two of the scaling factor combinations given by the μ_{ref} and Rng_{ref} parameters (see Eqs. (1) and (2)), in particular, the Mean and Maximum recommended values, which are derived from the individual distributions of the 45 chips. We conservatively estimate that μ_{ref} and Rng_{ref} can be independently set to 10 different values between these Mean and Maximum values.

Given these bounds on the configuration parameters, it is possible to generate a total of 4,194,304 * 17* 10 * 10 ~= 7 billion different bitstrings using the same set of paths (PNs). As discussed earlier, the verifier also applies a *Path-Selection-Mask* to each of the 2-vector sequences, which increases the number of possible bitstrings exponentially.

5.1.1 Actual Inter-chip Hamming Distance (HD_{interA})

Inter-chip hamming distance is reported in two ways, Actual and True. In this section, we compute HD_{inter} using the StrongBS produced after the application of the DHD method described in Section 3.4.

A set of StrongBS are created by AND'ing pairs of Helper Data bitstrings as follows. First, the enrollment $modPND_c$ is

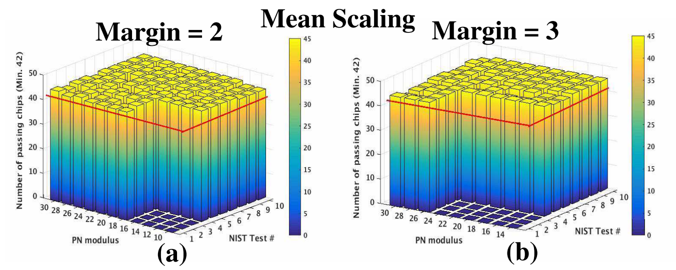

Fig. 7. NIST statistical test results [33].

used to create a set of 45 Helper Data bitstrings for each of the 45 chips. Second, Helper Data is computed using the $modPND_c$ collected under each regeneration corner for these 45 chips. For each chip, the enrollment Helper Data bitstring is AND'ed with the corresponding regeneration Helper Data bitstrings. The 45*15 AND'ed Dual Helper Data bitstrings are used to create a corresponding set of StrongBS using the method shown in Fig. 4(b) and (c). Note that the DHD method creates variable-sized bitstrings. We use the smallest bitstring that is produced by one of the chips in the HD_{interA} analysis. The smallest bitstring sizes are analyzed and reported on in Section 5.1.3.

HD_{InterA} is computed using Equation 3. The symbols *C, T,*

$$HD_{interA} = \frac{1}{CC \times B \times T} \sum_{t=1}^{T} \sum_{i=1}^{C} \sum_{j=i}^{C} \sum_{k=1}^{B} (SBS_{i,t,k} \oplus SBS_{j,t,k}) \times 100$$

Eq. 3.

B and *NC* represent 'number of chips' (45), 'number of regeneration TV corners' 'number of bits' (smallest bitstring size) and 'number of chip combinations' (45*44/2 = 990), resp. This equation simply sums all the bitwise differences between each of the possible pairing of chip StrongBS, and then converts the sum into a percentage by dividing by the total number of bits that were examined. HD_{interA} is computed in this fashion for each of the 256 seeds and averaged.

The HD_{interA} are shown in Fig. 6(a) and (b) for each of the *Moduli* and *Margin* combinations using Mean and Max. scaling factors for μ_{ref} and Rng_{ref}. The height of the bars are all very close to the ideal of 50%. Although an excellent result, this approach to computing Interchip-HD differs from the traditional approach because corresponding positions in the bitstrings are generated from different $modPND_c$. The results using the traditional approach, i.e., where the positions of the $modPND_c$ are preserved in the bitstrings, are reported on in Section 5.1.3.

5.1.2 NIST Statistical Test Results

The StrongBS referenced in Section 5.1.1 are used as input to the NIST statistical test suite [33]. The results using Mean Scaling and only 1 of the 256 *LFSR seed* pairs are presented in Fig. 7(a) and (b), for *Margins* of 2 and 3, resp. (the results for other configuration parameters are very similar). NIST test criteria classifies a test category as *passed* if at least 42 of the 45 chips pass the test. The figure shows all bars are above the red

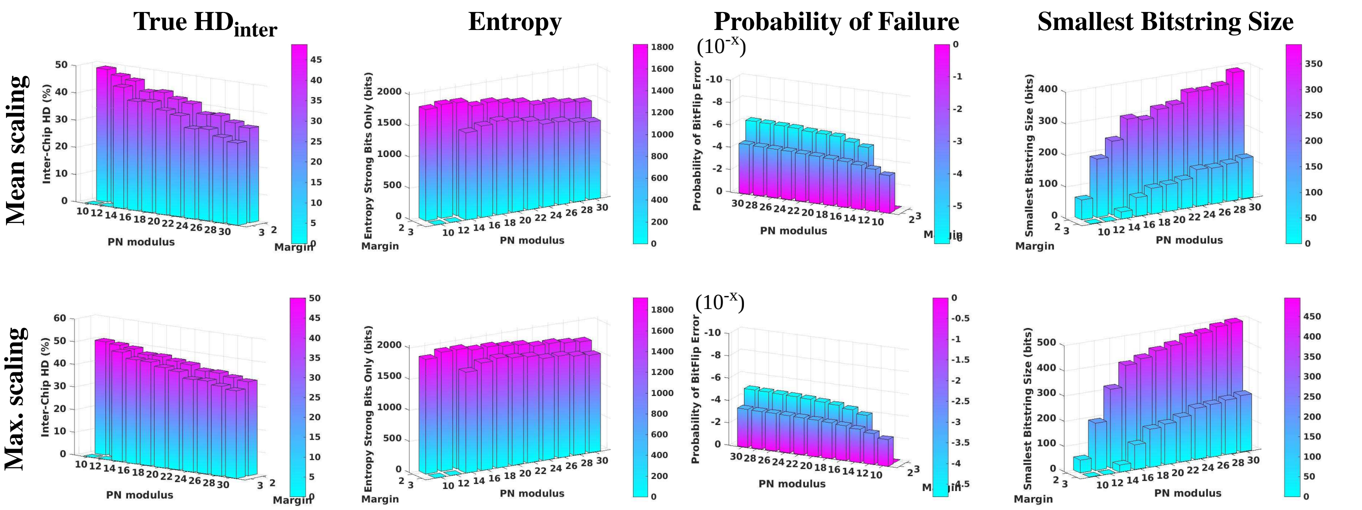

Fig. 8. Bitstring statistics using 4096 PNs collected from 45 copies of the FPGAs under 16 temperature/voltage corners using Mean and Maximum scaling factors for μ_{ref} and Rng_{ref}.

threshold line at 42, and therefore all test categories are passed. Bars of height 0 for NIST Tests 1, 2 and 3 identify *Moduli* that produced bitstrings with sizes less than the NIST requirement for those tests. The pass percentage when the NIST tests are applied to the bitstrings produced from **all** combinations of the investigated parameters is approx. 98.8%.

5.1.3 True Inter-chip HD (HD_{interT}), Entropy, Probability of Failure and Smallest Bitstring Size

Fig. 8 shows the results for *true* Inter-chip HD (HD_{interT}), Entropy, Probability of Failure and Smallest Bitstring Size (columns) using Mean and Max. scaling factors for μ_{ref} and Rng_{ref} (rows). Similar to HD_{interA}, HD_{interT} is computed as the average percentage across 990 pairings of bitstrings and 256 different pairs of *LFSR seeds*. However, the full length bitstrings of length 2048 are used and for each pairing of bitstrings, the hamming distance is computed using only bits classified as strong in both bitstrings. Under the Mean scaling factor, the HD_{interT} vary from 30% to 50% with the smallest value of 30.2% for *Margin* 3 and *Modulus* 30. For the Max scaling, most of the HD_{interT} values are between 40% and 50% with the smallest value of 38.7%. These results are also very good and indicate that a 2-bit offset can be used effectively with this range of *Moduli*.

Similarly, entropy is computed using the strong bits from each enrollment-generated bitstring of length 2048 and Eq. 4. The frequency p_i of '1's is computed as the fraction of '1's at each bit position using only those chips of the 45 which identify the bit as strong. The entropy values vary over a range from

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i) + (1-p_i)\log_2(1-p_i) \qquad \textbf{Eq. 4.}$$

approx. 1240 to over 1900. The ideal value is 2048 in this analysis so these results indicate that each bit contributes between 0.60 and 0.93 bits of entropy.

The Probability of Failure is reported as an exponent x from 10^{-x} with a value of -6 indicating 1 chance in 1 million. The HD_{intra} is computed by pairing the enrollment StrongBS for each chip against each of the 15 regeneration StrongBS under the DHD scheme and then counting the differences (bit flips) across all combinations of the 15 DHD-generated bitstrings. The number of bit flips for all chips are summed and divided by the total number of bits inspected. An average HD_{intra} is then computed using this process across a set of 256 *LFSR seed* pairs, which is then converted into an exponent representing the Probability of Failure. The results show that the Probability of Failure varies between 10^{-2} and 10^{-4}, with the largest (worst case) value at $10^{-2.4}$. Therefore, fewer than 1% of the bits for any authentication differ between the token and verifier under worst case environmental conditions.

The smallest StrongBS sizes are shown in the last column of Fig. 8. Using the condition that at least 80 bits are needed to meet the de facto lightweight security standard [30], the only parameter combinations that fail to meet this criteria are those noted earlier, i.e, *Modulus* of 10 for a *Margin* of 2 and *Moduli* of 14 and 16 for a *Margin* of 3.

5.2 Resource Utilization and Runtime Performances of FPGA Implementation

We implemented the proposed authentication protocol on the Xilinx Zynq 7020 SoC using the *sbox-mixedcol* data path component. Table 1 gives the resource utilization and runtime overhead associated with the *ID Phase* and *Mutual Phase* of the protocol. The table lists the resources in the order in which they are used by the authentication protocol, with '-' indicating repeated use of resources previously listed. The totals at the bottom indicate that area overhead is 6038 LUTs and 1724 FFs while the runtime is approx. 1.25 seconds. An alternative, lighter-weight implementation which uses only a single AES *sbox* component yields an area overhead of 2909 LUTs and

952 FFs and a runtime of approx. 2.2 seconds.

The implementation of HELP also requires an 18-bit multiplier and an on-chip BRAM memory of size 7.5 KBytes. The Xilinx IP blocks used in the implementation include a MMCM and a dual-channel (64-bits) AXI-GPIO for implementing communication between the processor and programmable logic components of the Zynq 7020 FPGA. The AXI-GPIO uses an additional 128 LUTs and 397 FFs.

Table 1: HELP authentication protocol area and runtime overhead.

Activity/Component	LUTs	FFs	Time (us)
ID Phase			
Network delay	-	-	44347
PN generation using *sbox-mixedcol*	3170	128	577834
Token timing engine	721	828	
Token bitstring gen. engine	1104	385	2359
Token controller and I/O	705	297	-
Verifier authentication	-	-	80
Mutual Phase			
Network + verifier delays	-	-	50830
Verifier bitstring gen.	-	-	54
Token timing engine + bitgen engine	-	-	577037
Token authentication	338	86	571
TOTALS	**6038**	**1724**	**1.25 sec**

The runtime is measured using an 8-core 3.4 GHz Intel i7 desktop computer as the verifier. The authentication time of 1.25 seconds includes network transmissions between the token and verifier. The exhaustive search carried out on the verifier takes approx. 300 microseconds per entry in the database. The runtime reported uses a database with only a single entry. Therefore, applications that incorporate a relatively small number of tokens (10K or less) require a search time of approx. 1.5 seconds on average, and a total authentication time of approx. 2.75 seconds.

6. Security Analysis

In this section, we investigate several important security properties of HELP that relate to its resistance to model building and to the size of its CRP space. The response space refers to the number of bitstrings that each token can generate using the six user-defined parameters described earlier. Our security analysis assumes the verifier securely stores the token's timing information that is collected during enrollment, encrypting it if necessary.

Earlier, we reported the size of the challenge space to be $2*(3^n-2^n)$ 2-vector sequences, and the number of response bitstrings to be approx. 7 billion excluding the diversity introduced by the *Path-Select-Mask*. The (n_1 XOR n_2) operation used in the protocol does not allow direct control over these configuration parameters. The *Path-Selection-Mask* increases the number of possible response bitstrings exponentially by changing the set of PNs used in the bitstring generation process. These characteristics of HELP and the protocol collectively add significant resilience to model-building attacks.

Two additional factors further increase HELP's model-building resistance. The first is referred to as the 'distribution effect'. The PNs selected by the *Path-Selection-Mask* change 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_{token} and Rng_{token} of the measured PND distribution to standardize the PNDs before applying the reverse transformation given by Eq. 2. The first transformation makes the final PND_c values dependent on the other components of the PND distribution. Therefore, machine learning techniques designed to learn the relative path delays as a mechanism to 'break the PUF' need to account for this 'distribution effect'.

We have also determined that the physical model for HELP is more complex than the models developed for the arbiter PUF. Therefore, it is likely that machine learning (ML) algorithms will require much larger training sets to achieve good prediction capability, if it is possible at all. This is true for several reasons. First, the adversary is required to run automatic test pattern generation (ATPG) to generate the vector pairs used in the training phase of the ML attack. Although this is a one-time cost, ATPG requires long runtimes and commonly fails to find vector pairs that test paths in a hazard-free robust manner, which is required to eliminate uncertainly about which path is actually being tested during the training phase. Second, a level of uncertainty will always remain because not all paths are hazard-free robust testable. In particular, the path that dominates the timing for cases where paths reconverge and have nearly equal nominal delays will be different from chip-to-chip. Third, ML algorithms such as *Probably Approximately Correct* (***PAC***) that have been effective against arbiter PUFs, guarantee success only when the model is polynomial in size [5][35-36]. Our preliminary work on the physical model indicate that the model has components that appear to be exponential in size, eliminating the possibility of a 'guaranteed' success. A full analysis of ML resistance will be provided in a future work.

7. Conclusions

A PUF-based, mutual, privacy preserving authentication protocol is described using a hardware-embedded delay PUF called HELP. The protocol uses an AES data path component referred to as *sbox-mixedcol* as the source of entropy. The proposed protocol does not require non-volatile memory or cryptographic primitives on the token. Path delay information is stored on the verifier during enrollment instead of response bitstrings. A set of configuration parameters are defined that create an exponentially large CRP space using a small set of measured path delays. A dual helper data scheme is proposed as a means of improving reliability. Data collected from the *sbox-mixedcol* functional unit on 45 copies of the Zynq 7020

FPGA shows HELP is capable of generating bitstrings of high statistical quality for use in PUF-based authentication protocols.

8. 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] M. Areno and J. Plusquellic, “Secure Mobile Association and Data Protection with Enhanced Cryptographic Engines”, *International Conference on Privacy and Security in Mobile Systems (PRISMS)*, 2013.

[3] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, “Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-enabled RFIDs”, *Vol. 7397 of Lecture Notes in Computer Science*, 2012, pp. 374-389.

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

[5] F. Ganji, S. Tajik, J.-P. Seifert, “Why Attackers Win: On the Learnability of XOR Arbiter PUFs”, International Conference on Trust & Trustworthy Computing, Aug. 2015.

[6] J. Aarestad, P. Ortiz, D. Acharyya and J. Plusquellic, “HELP: A Hardware-Embedded Delay-Based PUF”, *Design and Test of Computers*, Mar., 2013, pp. 17-25.

[7] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont, and M. Yung, "End-to-end Design of a PUF-based Privacy Preserving Authentication Protocol", *CHES*, 2015.

[8] W. Che, F. Saqib, J. Plusquellic, "PUF-Based Authentication", *International Conference on Computer-Aided Design*, Nov, 2015.

[9] J. Li, J. Lach; "At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection", *HOST*, 2008, pp. 8-14.

[10] P. Cui, “An Improved Ownership Transfer and Mutual Authentication for Lightweight RFID Protocols”, *International Journal of Network Security*, Vol. 18, No. 6, 2016, pp. 1173-1179.

[11] Y. Wang, H. Zhong, Y. Xu and J. Cui, “ECPB: Efficient Conditional Privacy-Preserving Authentication Scheme Supporting Batch Verification for VANETs”, *International Journal of Network Security*, Vol. 18, No. 2, 2016, pp. 374-382.

[12] M. H. Ibrahim, “Octopus: An Edge-fog Mutual Authentication Scheme”, *International Journal of Network Security*, Vol. 18, No. 6, 2016, pp. 1089-1101.

[13] J. Delvaux, D. Gu, R. Peeters, and I. Verbauwhede, "A Survey on Lightweight Entity Authentication with Strong PUFs", *Journal of ACM Computing Surveys*, vol. 48, no. 2, 2015.

[14] R. S. Pappu. Physical One-Way Functions. *PhD thesis*, MIT, 2001.

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

[16] L. Bolotny and G. Robins, “Physically Unclonable Function-based Security and Privacy in RFID Systems”, *PerCom*, 2007, pp. 211-220.

[17] E. Ozturk, G. Hammouri, and B. Sunar, “Towards Robust Low Cost Authentication for Pervasive Devices”, *PerCom*, 2008, pp. 170-178.

[18] G. Hammouri, E. Ozturk, and B. Sunar, “A Tamper-Proof and Lightweight Authentication Scheme, *Pervasive and Mobile Computing*, 2008, 807-818.

[19] L. Kulseng, Z. Yu, Y. Wei, and Y. Guan, “Lightweight Mutual Authentication and Ownership Transfer for RFID Systems”, *INFOCOM*, 2010, pp. 251-255.

[20] A.-R. Sadeghi, I. Visconti, and C. Wachsmann, “Enhancing RFID Security and Privacy by Physically Unclonable Functions”, *Information Security and Cryptography*, 2010, pp. 281-305.

[21] S. Katzenbeisser, Unal Kocabas, V. Van Der Leest, A. Sadeghi, G. J. Schrijen, H. Schroder, and C. Wachsmann, “Recyclable PUFs: Logically Recongurable PUFs”, *CHES*, 2011, pp. 374-389.

[22] U. Kocabas, A. Peter, S. Katzenbeisser, and A. Sadeghi, “Converse PUF-Based Authentication” *TRUST*, 2012, pp. 142-158.

[23] Y. S. Lee, T. Y. Kim, and H. J. Lee, “Mutual Authentication Protocol for Enhanced RFID Security and Anticounterfeiting”, *WAINA*, 2012, pp. 558-563.

[24] Y. Jin, W. Xin, H. Sun, and Z. Chen, “PUF-Based RFID Authentication Protocol against Secret Key Leakage”, *Vol. 7235 of Lecture Notes in Computer Science*, 2012, pp. 318-329.

[25] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, “Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching”, *Symposium on Security and Privacy Workshop*, 2012, pp. 33-44.

[26] Y. Xu and Z. He, “Design of a Security Protocol for Low-Cost RFID”, *WiCOM*, 2012, pp. 1-3.

[27] Y. S. Lee, H. J. Lee, and E. Alasaarela, “Mutual Authentication in Wireless Body Sensor Networks Based on Physical Unclonable Function”, *IWCMC*, 2013, pp. 1314-1318.

[28] M.-D. M. Yu, D. M’Rahi, I. Verbauwhede, and S. Devadas, “A Noise Bifurcation Architecture for Linear Additive Physical Functions, *HOST*, pp. 124-129.

[29] S. T. C. Konigsmark, L. K. Hwang, D. Chen, and M. D. F. Wong, “System-of-PUFs: Multilevel Security for Embedded Systems”, *CODES*, pp. 27:1-27:10, 2014.

[30] A. Aysu, E. Gulcan, D. Moryama and P. Schaumont, “Compact and Low-power ASIP Design for Lightweight PUF-based Authentication Protocols”, *IET Information Security*, Vol. 10, Issue 5, 2016, pp. 232-241.

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

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

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

[34] http://zedboard.org/product/zedboard

[35] F. Ganji, S. Tajik, J.-P. Seifert, “PAC Learning of Arbiter PUFs”, *Journal of Cryptographic Engineering*, Spec. Sect. Proofs 2014, 2016, pp. 1-10.

[36] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber, “Modeling Attacks on Physical Unclonable Functions”, *Cryptology ePrint Archive*, Report 2010/251, 2010.