Automated Design, Implementation, and Evaluation of Arbiter-based PUF on FPGA using Programmable Delay Lines

Mehrdad Majzoobi, Department of Electrical and Computer Engineering, Rice University, Houston, TX, 77005 USA
Akshat Kharaya, Department of Electrical and Computer Engineering, Indian Institute of Technology, Powai, Mumbai, MH 400076
Farinaz Koushanfar, Department of Electrical and Computer Engineering, Rice University, Houston, TX, 77005 USA
Srinivas Devadas, Department of Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02142

This paper proposes a novel approach for automated implementation of an arbiter-based physical unclonable function (PUF) on field programmable gate arrays (FPGAs). We introduce a high resolution programmable delay logic (PDL) that is implemented by harnessing the FPGA lookup-table (LUT) internal structure. PDL allows automatic fine tuning of delays that can mitigate the timing skews caused by asymmetries in interconnect routing and systematic variations. To thwart the arbiter metastability problem, we present and analyze methods for majority voting of responses. A method to classify and group challenges into different robustness sets is introduced that enhances the corresponding responses' stability in the face of operational variations. The trade-off between response stability and response entropy (uniqueness) is investigated through comprehensive measurements. We exploit the correlation between the impact of temperature and power supply on responses and perform less costly power measurements to predict the temperature impact on PUF. The measurements are performed on 12 identical Virtex 5 FPGAs across 9 different accurately controlled operating temperature and voltage supply points. A database of challenge response pairs (CRPs) are collected and made openly available for the research community.

Additional Key Words and Phrases: Reconfigurable systems, physically unclonable functions, hardware security, process variation.

1. INTRODUCTION

FPGAs provide a generic substrate of interconnected blocks that can be (re)programmed to achieve the desired functionality. The inherent flexibility of FPGAs compared to Application Specific Integrated Circuits (ASICs) together with their lower time-to-market as well as availability of third party IPs, have made them the platform of choice for many applications. Like other systems, FPGAs demand security and resilience to attacks. In addition, techniques for ensuring IP security are necessary for prevention against piracy and unauthorized access.

A common denominator for many security protocols is the concept of a *secret*. For example, in public- and private- key cryptography, there is a secret key shared among a limited number of parties. However, permanent storage of keys on FPGA is not straightforward, as FPGAs often do not include nonvolatile on-chip memory. Even when the keys are externally powered or hidden in the bitstream, side channel attacks for extracting the keys have been reported [Moradi et al. 2011].

Physical unclonable functions (PUFs) aim at addressing the shortcomings of the digital key storage by relying on the secrets generated by the inherent and unclonable unique mesoscopic characteristics (signatures) of the physical phenomena [Pappu et al. 2002; Gassend et al. 2002]. The physical properties of each device determine a specific mapping between a set of *challenges* (inputs) to a set of *responses* (outputs). Security protocols take advantage of the unique mappings provided by the CRPs to authenticate the device and/or its components [Majzoobi et al. 2012].

To date, a number of possible implementations of PUFs on FPGAs based on the unique silicon device-specific variations has been reported [Guajardo et al. 2007; Kumar et al. 2008; Suh and Devadas 2007]. New methods based on the reconfigurability of FPGA inherent delay variations of the PUF that is present even when the device is not configured, is used to configure blank FPGA every time an authentication takes place.

Applications of FPGA PUFs include securing programs and data, IP protection, RFIDs, secure key generation, remote activation, and IC enablement [Suh et al. 2005; Guajardo et al. 2007; Kumar et al. 2008; Suh and Devadas 2007; Majzoobi et al. 2009; Alkabani et al. 2007; Alkabani and

Koushanfar 2007]. However, limitations of the existing PUF implementations on FPGA include the polynomial number of CRPs, high power consumption, response errors, arbiter metastability, and/or the delay imbalances dictated by the routing constraints [Morozov et al. 2010; Majzoobi et al. 2009].

This paper introduces new methodologies that enable automated and stable implementation of an arbiter-based PUF on FPGA. An arbiter-based PUF works by comparing path timings for two routes with the same nominal delay (by design) but with slightly different actual delays (caused by manufacturing variations). To achieve equal nominal delays and to avoid biases, the two routes must be symmetric in shape. We introduce a low-overhead and high-resolution programmable delay line (PDL) implemented by a single lookup table (LUT) on the FPGA. The new PDL is used to tune and calibrate the delay bias caused by asymmetries in signal routing. Furthermore, a symmetric PDL-based switch structure is introduced that is implementable on FPGA. Also, the PDL mechanism can remove biases due to systematic variation effects.

To mitigate arbiter metastability and to achieve a higher robustness, we introduce redundancy and majority voting of the responses. We further present a new method to classify and group challenges into different robustness sets. The challenge classification increases the corresponding responses' resilience to environmental variations. Using the measurement data collected from PUFs on 12 FPGA across 9 different temperature and power supply conditions, we quantify the response robustness of each group and investigate the trade-off between response robustness and response entropy. Finally, using the measurement data, correlations between the effect of temperature and power supply variations on the PUF responses is quantified.

Our contributions in this paper are as follows:

— We introduce the first finely tunable PDL mechanism on FPGAs. This PDL can be implemented using single LUT and can achieve a resolution of $\frac{1}{32}$ pico-second and a dynamic range of 1 pico-second.
— We demonstrate the first working implementation of arbiter-based PUF on FPGA that can be fully automated[1]. Our implementation uses the PDLs to adjust for undesired asymmetries that complicate FPGA realization of delay-based PUFs.
— We demonstrate the evaluation of our arbiter-based PUF across a population of 12 identical FPGAs. A comprehensive open-source database of 64,000 CRPs per PUF is collected in controlled temperature and power supply settings and tuning conditions.
— The CRP database is thoroughly analyzed to derive optimal tuning levels, quantify response stability, train the PUF model, and reverse engineer the component delays.
— We suggest a new hypothesis that a larger delay difference at the arbiter input leads to a more robust (stable) response. We utilize PUF model building and delay parameter training to classify the challenges by the resulting delay difference at the arbiter input and use this classification to confirm our hypothesis.
— We investigate and quantify the trade-off between response robustness and response entropy (uniqueness). We hypothesize that highly robust responses are more likely to be similar (non-unique) across different PUFs.
— We present a new method, based on the temperature and power supply variations, to partially predict response errors in presence of temperature variations without costly temperature tests. We show that the less costly (controlled) power supply tests can help the response error prediction under temperature variations.

The rest of the paper is organized as follows. In Section 2, we provide a short background on arbiter-based PUF construct as well as methods to implement a PDL. In Section 3, we study the related literature on PDL and PUF implementations on FPGA. In Section 4, we introduce our LUT-based programmable delay line mechanism and show how this construct can help in automated implementation of arbiter-based PUF on FPGA. In Section 5, the method for improving the arbiter

[1]An earlier brief conference version of this work appeared in [Majzoobi et al. 2010c]

results accuracy by majority voting is discussed. Section 6 introduces robust challenge/response classification methodology, whereas in the next Section 7, we investigate the trade-off between response robustness and response uniqueness. In Section 8, measurement and evaluation results taken across various PUFs in different operating conditions are analyzed and presented. Section 9 concludes the paper.

2. BACKGROUND

A PUF utilizes the inherent specific properties of a physical device to define a unique mapping from a set of challenges (inputs) to a set of responses (outputs). The delay variations of CMOS logic components can be exploited to produce unique responses. In the PUF structure introduced in [Gassend et al. 2002], the analog delay difference between two parallel timing paths is compared by an arbiter. The paths are built identically and their delays must be equal by construction, but the physical device imperfections make them different. The architecture of the *arbiter-based PUF* racing parallel paths is demonstrated in Figure 1. A step input simultaneously triggers the two paths. At the end of the two parallel (racing) paths, an arbiter is used to convert the analog delay difference between the paths to a digital value. The two paths can be divided into several smaller subpaths by inserting path swapping switches. Each set of inputs to the switches act as a challenge set (denoted by C_i), defining a new pair of racing paths whose delays can be compared by the arbiter to generate a one-bit response.

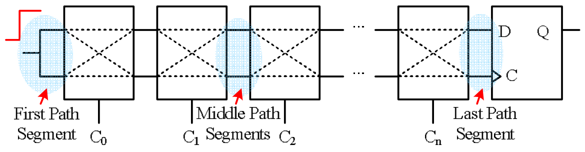


Fig. 1. Arbiter-based PUF with path swapping switches.

2.1. Programmable delay lines

Programmable delay lines (PDLs) alter the signal propagation delay in a controlled fashion. The common mechanisms used to change the delay includes (i) varying the effective load capacitance, (ii) modifying the device current drive (by increasing/decreasing the effective threshold voltage by body biasing), or (iii) incrementally altering the length of the signal propagation path. The first two methods are often employed in either analog fashion and/or in application specific integrated circuits (ASICs) and are not amenable to FPGA implementation.

On reconfigurable digital platforms such as FPGAs, PDLs can be implemented by only changing the signal propagation path length or by altering the circuit fanout that modifies the effective load capacitance. The latter is only feasible if dynamic reconfiguration is available. In other words, changing circuit fanout requires topological changes to the circuit which in turn needs a new configuration.

3. RELATED WORK

The authors in [Gassend et al. 2002] were the first to exploit the unique and unclonable silicon process variations in nanometer scales for PUF formation. Their PUF used the analog differences between the delays of two parallel paths that are equal in design, but the physical device imperfections make the delays different. An arbiter inserted at the end of the paths generates binary responses indicating a comparison between the delays. To generate many CRPs, the paths are divided into multiple subpaths and multiplexed by challenges. It is shown in [Majzoobi et al. 2009; Morozov et al. 2010] that implementation of delay-based PUFs on FPGAs is problematic because of the routing constraints and arbiter inaccuracy. Ring oscillator (RO) PUFs rely on the specific and unique delay of an oscillating path on each device [Suh and Devadas 2007]. The presently known PUFs of this type contain many RO's so there are many possible pairs to compare. One can only have a quadratic number of challenges with respect to the number of RO's on FPGAs. Another disadvantage of RO PUF is the continuous dynamic power dissipation due to oscillation.

Another class of candidate FPGA PUFs are SRAM-PUFs and butterfly PUFs [Guajardo et al. 2007; Kumar et al. 2008]. Each FPGA SRAM cell would naturally tend to one logic state (either zero or one) upon startup. There are only a polynomial number of challenges with respect to the number of SRAM cells. An FPGA-based PUF along with a suite of time-bounded authentication protocols is introduced in [Majzoobi et al. 2010b]. The PUF produces binary responses based on the difference between the clock speed and some combinational circuit delay. Some instances of analog and digital PUFs that attempt to implement public cryptography are presented in [Ruhrmair 2009; Beckmann and Potkonjak 2009; Csaba et al. 2009; Jaeger et al. 2010]. A comprehensive survey of PUFs can be found in [U. Ruhrmair 2011].

The scope of previous work on implementation of programmable delay line on FPGA is very limited. Altering the delays is usually performed using hard coded blocks that come ready inside FPGAs or phase locked loops (PLLs). These mechanisms are usually limited to the system clock or subsystem clocks rather than any arbitrary signals. In addition, such blocks provide limited resolutions in the order of micro seconds. To the best of our knowledge, the only work that attempts to use FPGA generic components to build programmable delay lines are the work presented in this paper and the one in [Bergeron et al. 2008].

In [Bergeron et al. 2008], a technique is proposed to alter the propagation path length by letting the signal bounce a few times inside the switch matrices of FPGA instead of a direct and straight connection. The concept is illustrated in Figure 2. In the switch matrix on the left side, the signal bounces three times off the switch edges before it exits the switch. In the right switch, the signal only bounces once; as a result a shorter propagation path length and a smaller delay is achieved. However, changing the switch connections points and routings require a new configuration, and doing so during the circuit operation is only possible by dynamic reconfigurability. The experiments on Virtex-II Pro devices show that any differential delay in a range of 947ps can be reached with a precision of +/- 18ps.

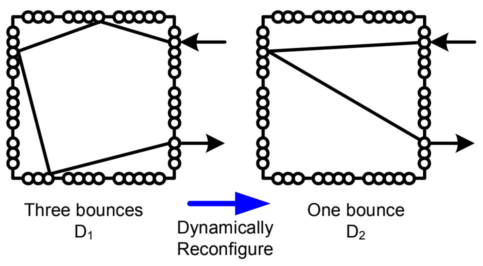


Fig. 2. A PDL implemented by altering the signal routing inside FPGA switch matrix.

This paper is an extension to the work presented in [Majzoobi et al. 2010c]. It presents a low overhead implementation of a programmable delay line (PDL) mechanism that uses only one look-up table. The proposed PDL can provide a differential delay in range of 10ps with a resolution of a fraction of pico-second without the need for dynamic reconfiguration.

4. ARBITER PUF ON FPGA

One of the major problems in implementation of PUFs on FPGAs, particulary the arbiter-based PUFs, is in signal routing. Unlike ASICs where hand-drawn custom layout is possible, routing on FPGA is constrained by its rigid fabric and interconnect structure. As a result, performing completely symmetric routing is physically infeasible in most cases. The PUF designer may do his/her best to constrain and guide the placement and routing software to achieve the highest degree of symmetry in the PUF layout. However, due to physical constraints of the FPGA fabric, the designer may still not be able to achieve complete symmetry on some routes. Asymmetries in routing when implementing PUFs can lead to bias in delay differences leading to predictable responses, lack of randomness, and decreased response entropy [Majzoobi et al. 2009; **?**].

The PUF routing can be divided into four different sections; the routing (1) before the first switch, (2) inside the switches, (3) between switches, and (4) after the last switch or before the arbiter (see Figure 1). As we will show later, by placing the logic components on symmetric sites and locations on the FPGA, the routing between switches will automatically follow a symmetric route. However, maintaining a *complete* symmetry between the top and bottom path routes before the first switch and after the last switch is structurally infeasible. To alleviate this problem, we introduce and exploit accurate PDLs to tune and remove the bias delay differences caused by asymmetries in net routing. We further introduce a new switch structure that has a symmetric implementation by construction.

4.1. Automated tuning with programmable delay lines

In this section, we introduce a low overhead and high precision PDL with *pico*-second resolution. The introduced PDL is implemented by a single LUT. Figure 3 shows the internal structure of an example 3-input LUT. An *n*-input LUT can be configured to implement any *n*-input logic function. The LUT in Figure 3 is configured so that the inputs A_2 and A_3 act as *don't-care* bits. The LUT output is inverted A_1 and is not a function of A_2 and A_3. However, looking more closely, the inputs A_2 and A_3 determine the signal propagation path inside LUT. For instance, if $A_2A_3 = 00$, the signal propagates through the solid path (red), whereas if $A_2A_3 = 11$, the signal propagates through the path marked with the dashed-lines (blue). The lower dashed path is slightly longer than the upper solid path which results in a larger propagation delay. The Xilinx Virtex 5 FPGA has 6-input LUTs

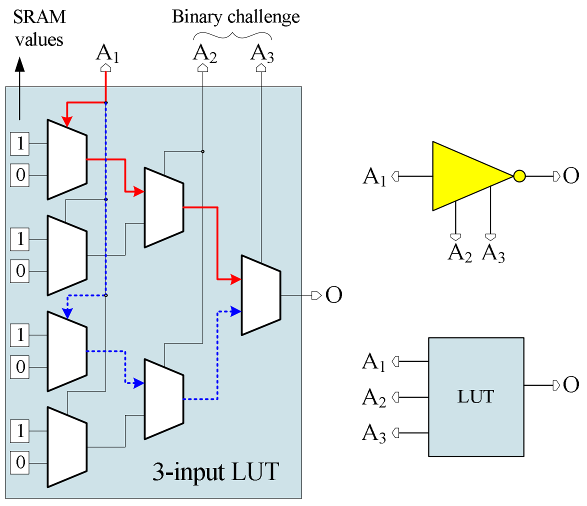


Fig. 3. The internal structure of a 3-input LUT.

which can implement a PDL with 5 control bits - there are 4 LUTs in each Slice and two Slices per CLB. Similar to the above example, the first LUT input, A_1, is the inverter input and the rest of the LUT inputs control the delay of the inverter. For, $A_2A_3A_4A_5A_6$=$A_{[2:6]}$=00000, the inverter has the smallest delay (shortest internal propagation path) and for $A_2A_3A_4A_5A_6$=$A_{[2:6]}$=11111, the inverter has the maximum delay. In general if $A_{[2:6]} > A'_{[2:6]}$ then $D_{LUT}(A) > D_{LUT}(A')$, where $D_{LUT}(A)$ and $D_{LUT}(A')$ are the delay of the inverter with A and A' as the control inputs respectively.

We measured the changes in LUTs' propagation delays under different inputs. For delay measurements, we used the timing characterization circuit shown in Figure 8.1. The characterization circuit consists of a *launch* flip-flop, *sample* flip-flop, and *capture* flip-flop, an XOR gate, and the *Circuit Under Test (CUT)* whose delay is to be measured.

At the rising edge of the clock a signal is sent through the CUT by the launch flip-flop. At the falling edge of the clock, the output of the CUT is sampled by the sample flip-flop. If the signal arrives at the sample flip-flop well before sampling takes place, the correct value is sampled. The XOR compares the sampled value with steady state output of the CUT and produces a zero if they are the same. Otherwise, the XOR output rises to '1', indicating a timing violation. If the signal arrival and the sampling time (almost) simultaneously occur, the sample flip-flop would enter into a metastable condition and produce a non-deterministic output. By sweeping the clock frequency and monitoring the rate at which timing errors happen, the CUT delay can be measured with a very high accuracy and in an automated way. For further details on the delay characterization method the reader is referred to [Majzoobi et al. 2010b].

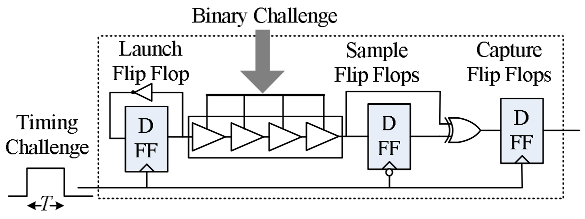


Fig. 4. Delay characterization circuit.

The measurements performed on Xilinx Virtex 5 FPGAs suggest that the maximum delay difference (i.e., A=00000, and A'=11111) achieved by each inverter is $9ps$ on average.

4.2. PDL-based symmetric switch

The first arbiter-based PUF introduced in [Gassend et al. 2002] (see Figure 1) uses path swapping switches as shown in Figure 5 (a). The switch, based on its selector bit, provides a straight or cross connection. Figure 5 (b) shows the equivalent circuit implementation and delays. The path swapping switch structure does not lend itself to FPGA implementation, since it is extremely difficult to equalize the nominal delays of the top and bottom paths due to routing constraints, i.e., ***a*** and ***d*** (or the diagonal paths ***b*** and ***c***). To alleviate the issue, we propose a new non-swapping switch structure as shown in Figure 5 (c). The yellow triangles in the figure represent two PDLs. Figure 5 (d) shows its equivalent circuit where the nominal delay values of ***a*** and ***d*** (or the diagonal paths ***b*** and ***c***) must be the same.

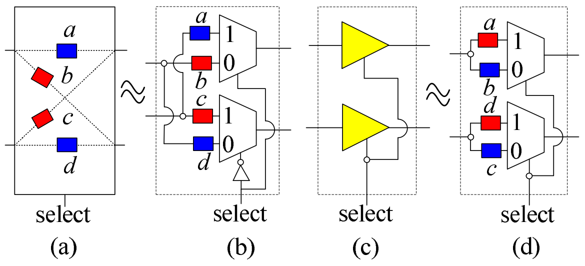


Fig. 5. (a),(b) path swapping switch and its delay abstraction (c),(d) PDL-based switch and its delay abstraction.

The complete PUF circuit that uses the new switch structure and the tuning blocks is shown in Figure 6. The presented system consists of N switches and K tuning blocks. The tuning blocks insert extra delays into either the top or bottom path based on their selector inputs to cancel out the delay bias caused by routing asymmetry. The only difference between a tuning block and a switch block is that in the former, the selectors to the top and bottom PDLs are controlled independently but in the latter, the same selector bit drives both PDLs. Also note that the tuning blocks do not necessarily have to be placed at the end of the PUF. As a matter of fact, they can be placed anywhere on the PUF in between the switches.

The design of this new PUF structure can be readily automated. Similar to the arbiter-based PUF with path swapping switches, the new PUF structure is a linear system. The PUF response will be '1' if the sum of the delay switch differences along the path is greater than zero, and '0' otherwise:

$$\sum_{i=1}^{N} C_i \times (a_i - d_i) + (1 - C_i) \times (b_i - c_i) + \Delta \underset{R=1}{\overset{R=0}{\lessgtr}} 0, \tag{1}$$

where a_i, b_i, c_i, d_i are the i-th switch delays as shown in Figure 5 (d), $C_i \in \{0, 1\}$ is the i-th challenge bit, and R is the response. Also, Δ is a constant delay difference from the first and last path segments and tuning blocks lumped together. The security aspects of the linear PUF structures against machine learning attacks can be boosted by insertion of feed forward arbiter and attaching input/output XOR logic networks to multiple rows of PUFs [Majzoobi et al. 2008; Daihyun et al. 2005]. The work in analyzing the complexity of machine learning and model attacks against different classes of PUFs is given in[Rhrmair et al. 2010].

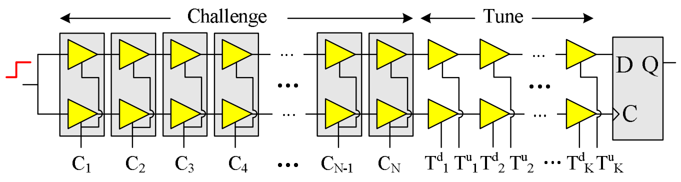


Fig. 6. The new arbiter-based PUF structure.

5. PRECISION ARBITER

Arbiters in practice are implemented by D flip-flops. As a result, an arbiter has a limited resolution meaning that if the absolute delay difference of the arriving signals is smaller than its setup and/or hold time, it enters a metastable state where its output becomes highly sensitive to circuit noise and will be unreliable. The probability of flip-flop output being equal to '1' is a monotonically decreasing function of the input signal timing difference (Δ_T). Such probability in fact follows a Gaussian CDF curve as shown in [Majzoobi et al. 2009; Majzoobi et al. 2010a]:

$$P_{O=1}(\Delta_T) = Q(\frac{\Delta_T}{\sigma}) \tag{2}$$

where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right)$ is the Q function. For an infinitely precise arbiter, σ is infinitesimal i.e., $\sigma \to 1/\infty$, and $P_{O=1}(\Delta_T) \to 1 - U(\Delta_T)$ where U is the step function.

To increase the arbiter accuracy, we propose multiple evaluations of the same challenge to the PUF and running a majority vote on the output responses as shown in Figure 17. The repetitive challenge evaluation combined with majority voting is equivalent to having an arbiter with effectively smaller σ. We will quantify the reduction in σ as a function of the number of repetitions in the experimental results section.

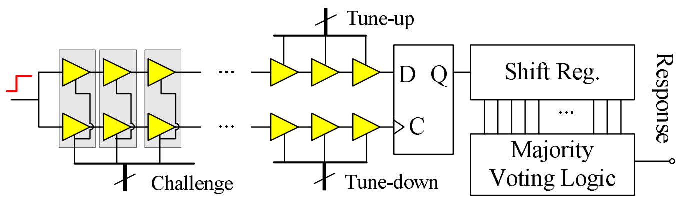


Fig. 7. Reducing the response instability due to arbiter metastability by using majority voting.

6. ROBUST RESPONSES

Fluctuations in operational conditions such as temperature and supply voltage can cause variations in device delays. The impact on delays may not be equal on all devices. As an example, the signal propagation delay on the PUF top and bottom paths is represented in Figure 8 by solid and dashed lines respectively. In this example, the path delays increase with temperature at different rates. In the diagram in Figure 8 (a), the delay difference Δ_d at the end of the PUF for a given applied challenge at nominal temperature is small, whereas Δ_d in Figure 8 (b) is larger for another challenge. The response to the challenge in Figure 8 (a) changes as temperature varies because the delays change their order (cross). However, in Figure 8 (b) the PUF response remains the same. As demonstrated by this example, the responses to those challenges that cause large delay differences are unlikely to be affected by temperature or supply voltage variations [Suh and Devadas 2007].

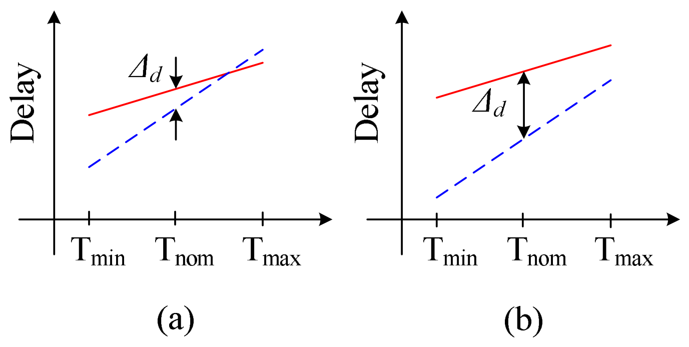


Fig. 8. Signal propagation delay as a function of temperature.

In this paper, we estimate the delay difference at the input of the arbiter. To estimate the cumulative delay difference (Δ_d), we ought to first train the delay parameters of the linear model of the PUF expressed in Equation 1 on the available set of challenge and responses. After estimating the delay parameters, the left hand sum in Equation 1 is evaluated for every new challenge. The distribution of the resulting sum (Δ_d) to the set of available CRPs is next calculated. Now based on the distribution, if the delay difference caused by a given challenge falls in the tails of the distribution, we expect (and will later verify and quantify it through experiments) that the response to this challenge is less likely to be affected by variations in operating conditions. Figure 9 shows the distribution of the delay differences at arbiter input to a diverse set of challenges. The challenge set is partitioned into equal sized partitions (bin) based on the delay difference each challenge produces. Next, the stability of response to the challenges in each set is measured. We argue that the responses to challenges that fall into the center partitions exhibit lower robustness compared to those in corner partitions.

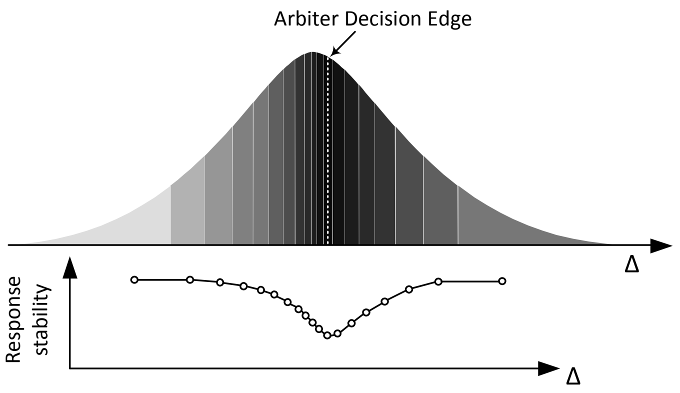


Fig. 9. The distribution of Δ_d and stability of responses in the corresponding partitions.

7. ROBUSTNESS VERSUS ENTROPY

The next question that arises from classifying robust challenges from non-robust ones is: ”Are robust challenges that good?”. In other words, are we trading something off to gain stability and robustness? From information theoretical point of view, it is likely that the responses from more robust challenges bear lower entropy. For example, consider the extreme case where responses are absolutely biased towards either zero or one. In this case we have ultimate robustness whereas the entropy is zero and the responses are not distinct enough for identification. This trade-off (if exists) can only be quantified through measurements. We show this is in fact the case and quantify the loss in entropy in exchange for robustness in the experimental results section.

8. EXPERIMENTAL EVALUATION

8.1. Programmable delay line

Before moving onto the PUF system performance evaluation, we shall first discuss the results of our investigation on the maximum achievable resolution of the programmable delay lines. We set up a highly accurate delay measurement system similar to the delay characterization systems presented in [Majzoobi et al. 2010b; Majzoobi et al. 2010a; Majzoobi and Koushanfar 2011].

The circuit under test consists of four PDLs each implemented by a single 6-input LUT. The delay measurement circuit as shown in Figure 8.1 consists of three flip-flops: launch, sample, and capture flip-flops. At each rising edge of the clock, the launch flip-flop successively sends a low-to-high and high-to-low signal through the PDLs. At the falling edge of the clock, the output from the last PDL is sampled by the sample flip-flop. At the last PDL’s output, the sampled signal is compared with the steady state signal. If the signal has already arrived at the sample flip-flop when the sampling takes place, then these two values will be the same; otherwise they take on different values. In case of inconsistencies in sampled and actual values, XOR output becomes high, which indicates a timing error. The capture flip-flop holds the XOR output for one clock cycle.

To measure the absolute delays, the clock frequency is swept from a low frequency to a high target frequency and the rate at which timing errors occur are monitored and recorded. Timing errors start to emerge when the clock half period (T/2) approaches the delay of the circuit under test. Around this point, the timing error rate begins to increase from 0% and reaches 100%. The center of this transition curve marks the point where the clock half period (T/2) is equal to the effective delay of the circuit under test.

To measure the delay difference incurred by the LUT-based PDL, the measurement is performed twice using different (complementary) inputs. In the first round of measurement, the inputs to the four PDLs are fixed to $A_{2-6} = 11111$. In the second measurement, the inputs to the last PDL are changed to $A_{2-6} = 00000$. In our setup, a 32×32 array of the circuit shown on Figure 8.1 is implemented on a Xilinx Virtex 5 LX110 FPGA, and the delay from our setup is measured under

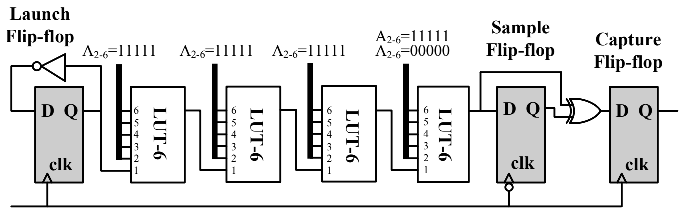
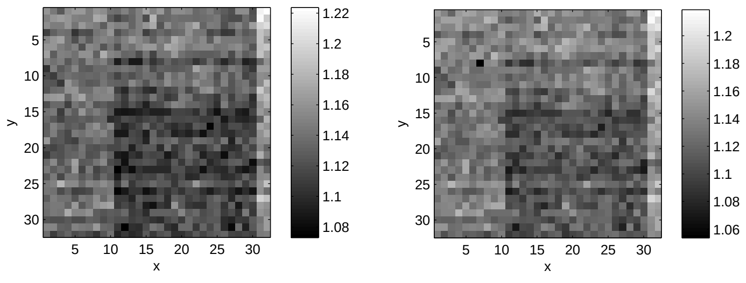


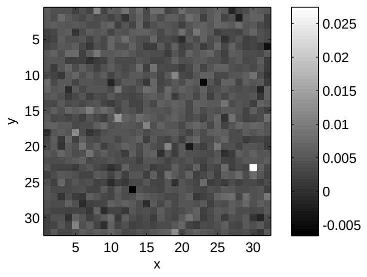
Fig. 10. The delay measurement circuit. The circuit under test consists of four LUTs each implementing a PDL.

the two input settings. The clock frequency is swept linearly from 8MHz to 20MHz using a desktop function generator and this frequency is shifted up by 34 times inside the FPGA using the built-in PLL.

The results of the measurement are shown on Figure 11. Each pixel in the image corresponds to one measured delay value across the array. The scale next to the color-map is in nano-seconds. Figure 11 (a) and (b) show the path delay when the last LUT in Figure is driven by $A_{2-6} = 00000$ and $A_{2-6} = 11111$ respectively. Figure 11 (c) depicts the difference between the measured delays in (a) and (b). As can be seen, the delay values in (b) are on average about 10 pico-seconds larger than the corresponding pixel values in (a).



(a) Delay for $A_{2-6} = 00000$ (b) Delay for $A_{2-6} = 11111$



(c) Delay difference

Fig. 11. The measured delay of 32×32 circuit under tests containing a PDL with PDL control inputs being set to (a) $A_{2-6} = 00000$ and (b) $A_{2-6} = 11111$ respectively. The difference between the delays in these two cases is shown in (c).

8.2. Arbiter-based PUF evaluation

Next, we use the programmable delay lines to implement the arbiter-based PUF on FPGA. The implemented PUF has 16 rows whose challenge input bits are connected together and placed in parallel on the FPGA to produce 16 bits of responses per challenge. Each PUF consists of 64 stages of PDLs, where the PDL is implemented by 2 LUTs each acting as an inverter. Figure 12 shows the placement and routing of one of the PUF rows. As it can be seen, except for the routing at the beginning and end of the PUF, the rest follows a completely symmetric pattern.

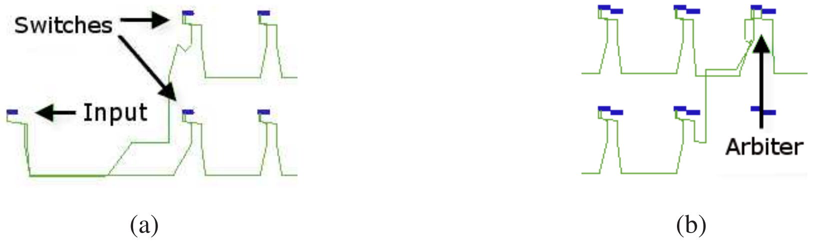


Fig. 12. Routing and placement of the PUF (a) first segment (b) last segment.

8.3. Measurement setup

We have a population of 12 Xilinx Virtex 5 (LX110) FPGAs at our disposal. The FPGAs are mounted on a ball-grid array socket available on Xilinx FF676 Prototype board only. Since the prototype board is stripped of any communication interface, we create a synchronous serial communication protocol to send/receive the data to/from XUP-V5 development board. From the XUP-V5 board, the data is sent to the PC through the Ethernet communication interface at a very high speed by using SIRC API. SIRC (Simple Interface for Reconfigurable Computing) is an open sourced software/hardware API developed at Microsoft Research that enables data transfer at full Ethernet speed of 1GB/s between the FPGA and PC [Eguro 2010]. Additionally, to perform measurements under various temperature points, we use PTC10 temperature controller from Stanford Research Systems. The temperature controller drives a TEC (Thermo-electric coupler) Peltier device. TEC is attached on the top of the FPGA and beneath a heat-sink. A closed-loop feedback system is established to control the FPGA temperature accurately. The temperature feedback is provided by an on-die diode junction voltage on the Virtex 5 device. This way the stable temperature would be that of the die temperature rather than the package temperature. The temperature controller is further calibrated to reliably map the junction voltage of the diode to die temperature using the temperature readings obtained through ChipScope Pro on-die temperature sensor. The measurement system connections and setup is depicted in Figure 13. Figure 14 shows the measurement system setup in the lab. The raw data, scripts and software is made available online at http://aceslab.org/node/1012.

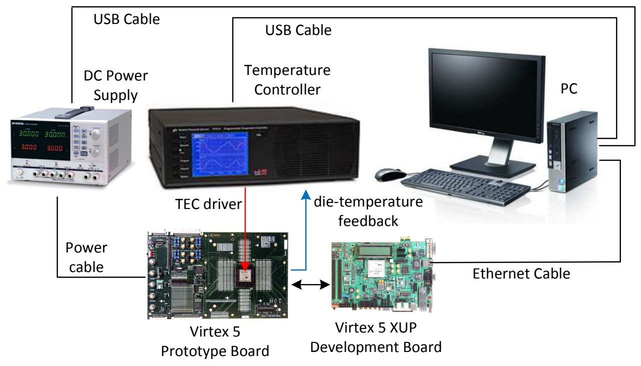


Fig. 13. Measurement system setup diagram.

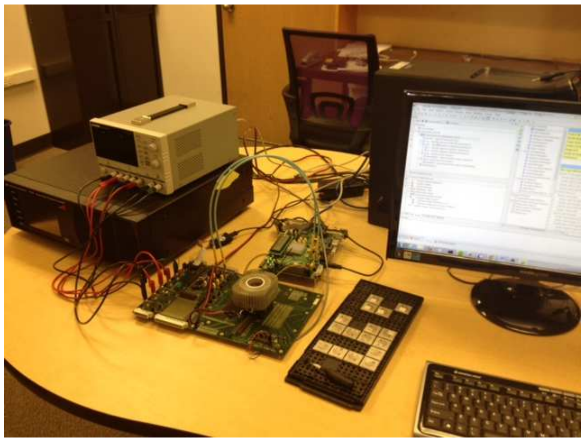


Fig. 14. Lab setup.

8.4. Tuning the PUF

Before using the PUF, in order to see any changes in the responses it must be tuned to remove the delay bias resulting from routing asymmetry. In the first experiment, we look at all 16 responses to find out at what tuning level their responses to a set of random challenges are 50% zeros and 50% ones. To be able to find the best tuning level, we feed the PUF with a set of 64,000 random challenges while for each challenge, we sweep the tuning level from -10 to 40. In each sweep point (each tuning level), we collect 64,000 responses from each PUF row (64,000$\times$16 total for each FPGA). Then, we look at the percentage of ones and zeros in each response set across different tuning levels and find the set that is properly balanced.

We refer to *tuning level* as the difference in the number of '1's in the top and bottom PDL selector bits. The tuning level can be either positive or negative indicating insertion of delays to the top and bottom path respectively. Note that when the tuning level is set to 40, for example, then it means that 40 of the PDL blocks out of 64 blocks are dedicated to tuning and only 24 bits of the inputs serve as the input challenge.

The response to a given challenge at each tuning level is repeated 128 times, and a majority vote on the responses is performed to resolve the repeated readings to a single response value. Figure 15 shows the ratios of ones in each response set (y-axis) as a function of tuning level (x-axis) for FPGA number 6. Since each PUF on each FPGA produces 16 response bits, there are 16 lines on each subplot. There are 9 subplots in each plot. Each subplot corresponds to the measurement taken under a different operating condition. The center subplot refers to the normal operating condition (i.e. supply voltage V_{DD}= 1 V and room temperature of 30^{o}C). Note that the plot is only for one FPGA (FPGA number 6). We have repeated the same experiment on all 12 FPGAs in the lab and the results are available online at http://aceslab.org/node/1012.

Figure 16 shows the distribution of the center of the transition points across all PUFs on all FPGAs.

8.5. Majority voting

As discussed in the paper, repeating the challenges to the PUF and running majority voting of the obtained responses can help improve the precision of the arbiter. In this section, we quantify this effect. Figure 17 shows the probability of observing a '1' output from a flip-flop as a function of the input signals delay difference. This characteristic has been measured on Xilinx Virtex 5 FPGAs [Majzoobi et al. 2009; Majzoobi et al. 2010a]. The width of the transition region (3σ) gets narrower as evaluation is repeated and more statistics is gathered.

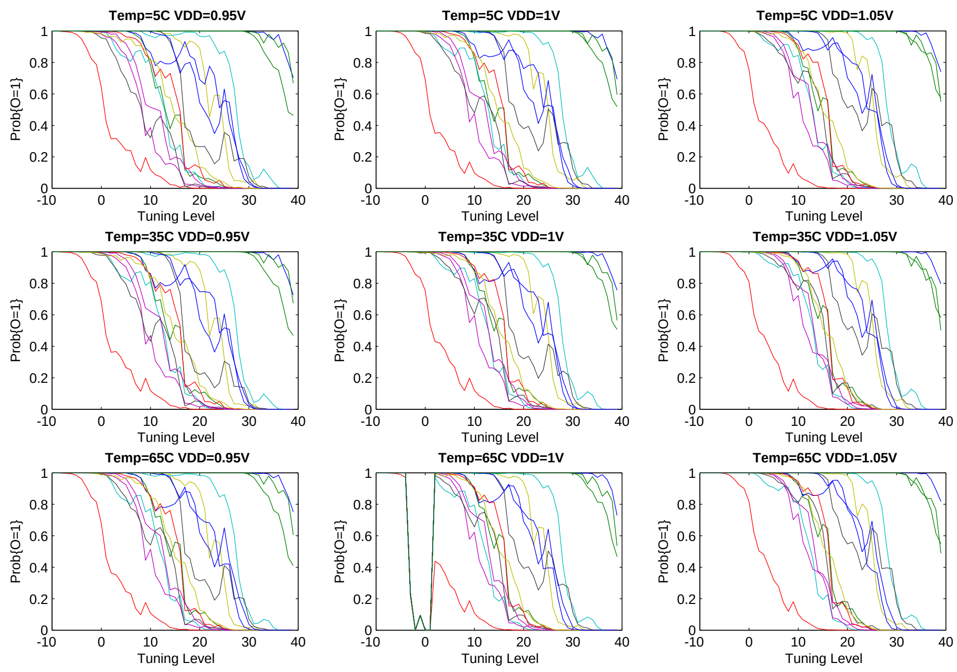


Fig. 15. Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 6.

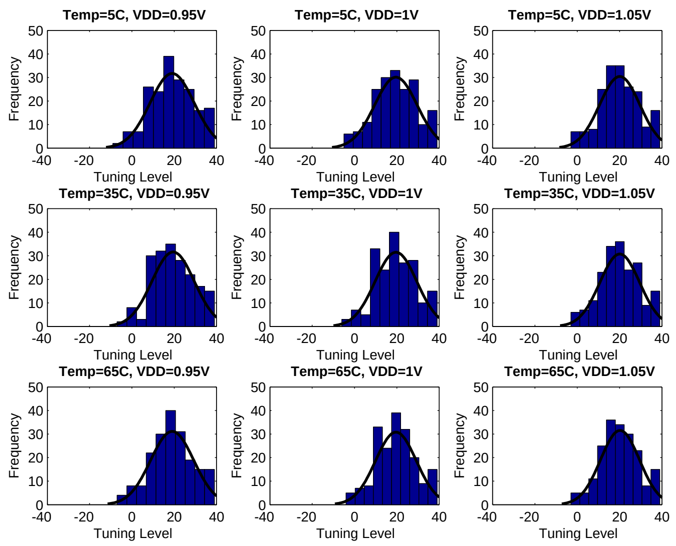


Fig. 16. Distribution of the tuning levels across all PUF rows on all FPGAs for different operating conditions.

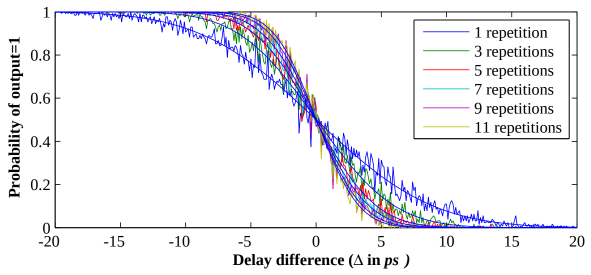


Fig. 17. The probability of majority voting system output being equal to 1 as a function of the delay difference.

The equivalent σ which represents the width of the metastable window (i.e., 3σ) is calculated for different number of repetitions as shown in Figure 18. The reduction in the metastable window width is logarithmic with respect to the number of repetitions. For 10 repetitions, σ = 2.5 ps.

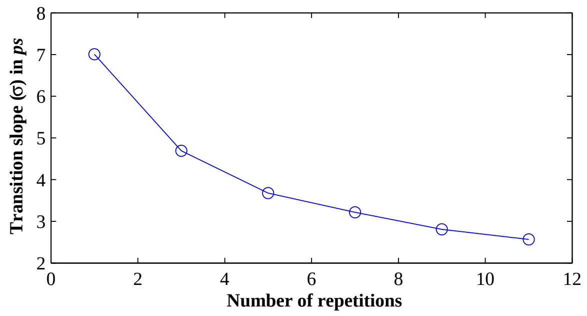


Fig. 18. The sharpness (σ) of the transition slope versus the number of repetitions for majority voting.

8.6. Robust response classification

Next, we measure the effect of robust challenge classification on PUF error rate in presence of temperature and supply voltage variations as discussed in Section 6. Each challenge to the arbiter PUF creates a delay difference (Δ) at the input of the arbiter (flip-flop). The Δs produced by all challenges in the challenge space form a Gaussian distribution. If half of the responses are one and half are zero, then this distribution has a mean of zero. The distribution is split by the arbiter decision edge. Those challenges that create a Δ that is larger that e, result in a '1' response and a zero response otherwise, where e is basically the arbiter bias that has remained after tuning. We partition the Δ distribution and the corresponding challenge space into 20 sets of equal size. The Δs close to the decision border and their corresponding responses are more sensitive to environmental condition fluctuations, and those farther apart from the decision border (i.e. $|\Delta$ - $e| \gg 0$) are less affected by such variations. The Figure 19 shows the robustness of the responses to different subset of challenges. The x-axis in each subplot refers to the challenge partition (bin) number. Each partition contains 64000/20 = 3200 challenges. The y-axis shows the stability of the corresponding responses, where '1' means no errors in the responses and '0' means completely erroneous responses. The error is measured by comparing the responses from eight corner cases to the response at the normal operating condition (room temperature and nominal supply voltage). Therefore, each subplot contains eight lines for each corner case. As it can be observed, the challenges in bins that are closer to the decision border produce responses with larger error rates. There are 16 subplots in each figure where each correspond to a PUF output response bit.

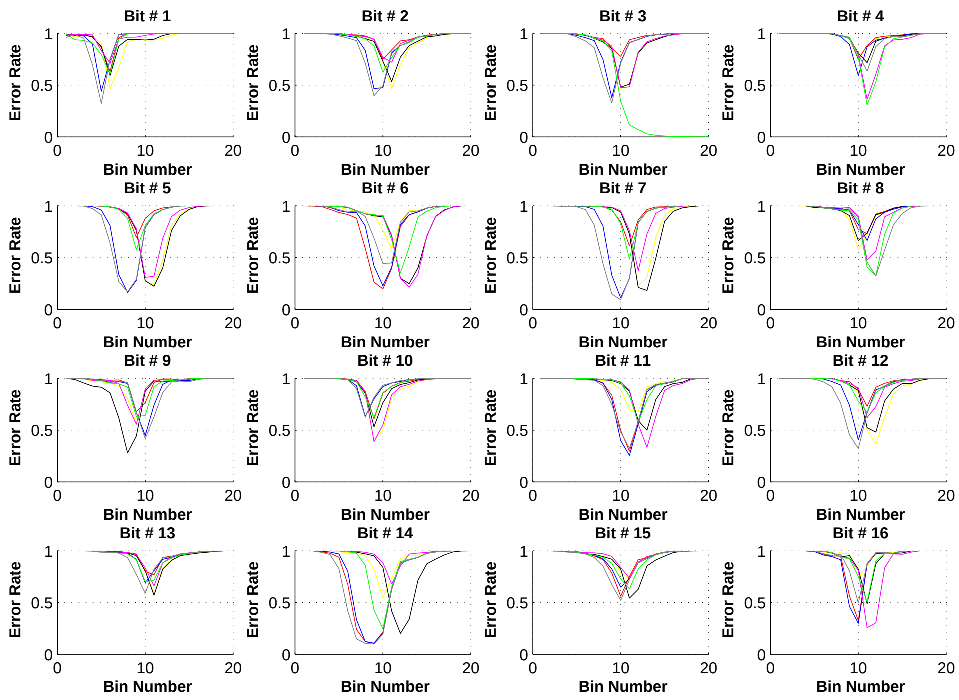


Fig. 19. Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 6.

Case	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T_1	L	M	L	L	M	L	L	M	L	M	H	H	M	H	H	M	H	H
T_2	M	H	H	M	H	H	M	H	H	L	M	L	L	M	L	L	M	L
V_1	L	L	L	M	M	M	L	L	L	M	M	M	H	H	H	H	H	H
V_2	M	M	M	H	H	H	H	H	H	L	L	L	M	M	M	L	L	L

Table 1: 18 correlation cases studies for various increments/decrements on temperature and power supply

Figure 20 shows the distribution of the error rates for each challenge partition using boxplots. Each subplot corresponds to an operating condition corner. As it can be seen, the average error rates is considerably lower at corner (lower and higher) partitions.

8.7. Robustness versus entropy

Now that we have quantified the stability of responses to different challenges, it is time to investigate the entropy of the responses to such challenges. In order to quantify the entropy, we look at the inter-chip Hamming distance of PUF responses to challenges in different partitions. For the 12 available FPGAs, 66 distinct pairing of FPGAs can be selected. However, since the tuning level of each PUF on FPGA is different, the challenge set is selected based on the target FPGA. For example, the challenge set for the pair FPGA A and FPGA B is different from FPGA B and FPGA A. This asymmetric challenge selection requirement also means that the inter-chip Hamming distance between FPGA A and FPGA B might be different from FPGA B and FPGA A. Therefore, we

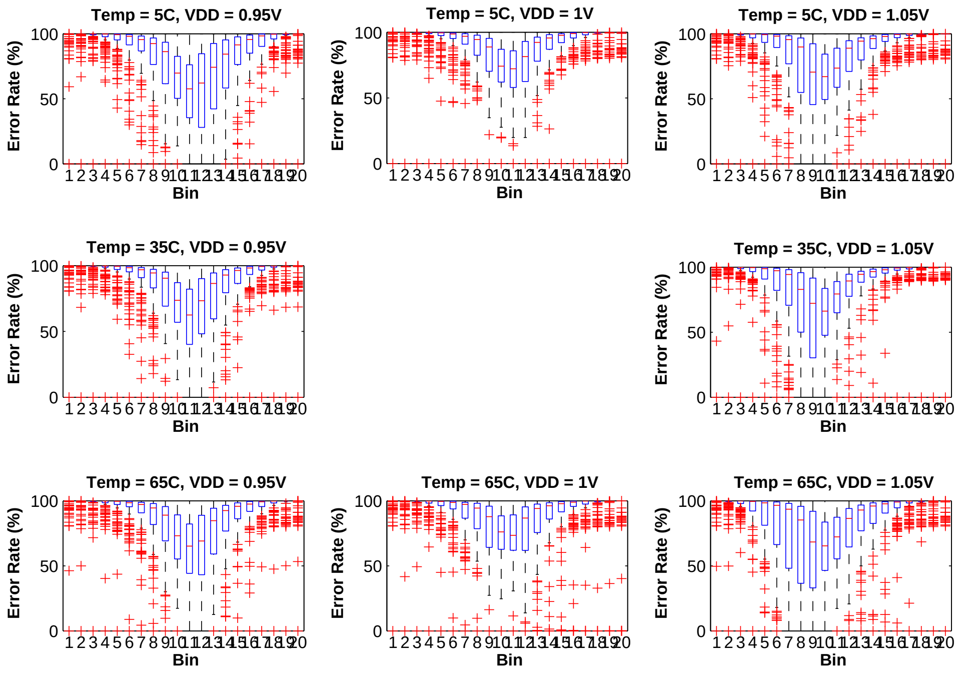


Fig. 20. Boxplot showing the distribution of error rates for a given operating condition corner and challenge partition.

investigate the Hamming distance for all 12×11 possible pairing (of course excluding similar chip parings). At each partition, a set of 3200 response vectors of size 16 bits are compared to another set. The result is 3200 integer hamming distances between 0 and 16. We take the average value as the inter-chip hamming distance and normalize it with 16. Next we need to link entropy with Hamming distance. Entropy is maximum if the average normalized inter-chip hamming distance is at 0.5. Any deviation from 0.5 lowers the entropy. In other words, both Hamming distance of 0 and 1 indicate entropy of zero. Figure shows the entropy as measured by Hamming distance for response to challenges in each partition. Each line on this figure corresponds to one paring of FPGAs.

8.8. Correlation between effects of temperature and power supply variations

Variation of temperature and/or core voltage from nominal values changes the response to challenges, especially the non-robust challenges. We argue that response flips due to change in temperature is related to response flips due to change in core voltage. Temperature testing is expensive; if a correlation between variation due to temperature and variation due to core voltage can be established even partially, it will help predict temperature effects from core voltage effects and thus lead to a huge cost saving.

The 64000x16 responses for each of the 12 FPGA under various experimental conditions (different temperature and voltage) are used to quantify this argument. The response set obtained in a reference condition is compared to the response set obtained in condition N_1 and the challenges for which the response flips are noted, where N_1 condition being an increment (or decrement) in core voltage from the reference value.

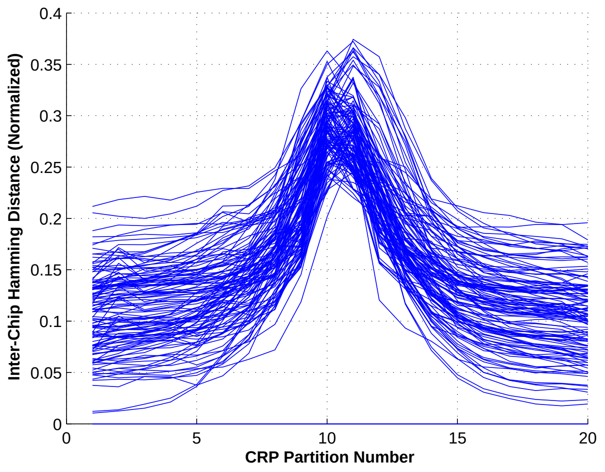


Fig. 21. Entropy of the response to the challenges at each robustness partition.

Then the response set obtained in reference condition are compared to the response set obtained in N_2 condition only for the challenges noted in N_1, where N_2 condition being an increment (or decrement) in temperature from the reference value. In other words, if the response to challenge "C", flips (changes from 0/1 to 1/0) as the power supply goes from V_1 to V_2, how likely is it that the response to the same challenge "C", flips as the temperature goes from T_1 to T_2 (while the core voltage stays at V_1). Each PUF is set at a characteristic tuning level for which it has an equal probability of 0 or 1 as an output and the response set is analyzed at that characteristic tuning level to obtain a response error correlation value. (T_1, V_1) and (T_2, V_2) comprise the condition N_1 and N_2 respectively. Figure 22 shows the results as boxplot for 18 different experimental conditions tabulated in Table 8.7. The low/high values for core voltage are set assuming a practical tolerance level of 5% in power supply. Low (L), medium (M) and high (H) values for core voltage are 0.95V, 1.00V and 1.05V respectively and for temperature are 5^o C, 35^o C and 65^o C respectively."

Each box in Figure 22 represents the result of the corresponding case and is drawn for the set response error correlation values obtained from 12×16 PUF response sets. The lower and upper edges represent the 25th and 75th percentile respectively while the edge partitioning the box at the centre is the median correlation value from the set of 192 correlation values which is used to quantify this response error correlation. Correlation between voltage and temperature is maximized in case 16 (0.68355), while the correlation in case 7 is also comparable (0.66355). It is interesting to note that case 16 and case 7 are complementary, i.e. (T1, V1) are interchanged with (T2, V2).

9. CONCLUSION

We introduced a new high-precision low-overhead programmable delay lines (PDL) implementation using a single look-up table. We designed and implemented an arbiter-based PUF structure that exploits programmable delay lines (PDL) to tune and cancel out the delay skews caused by asymmetries in routing on FPGAs. A PDL-based symmetric switch structure was further introduced to resolve the routing issues. To mitigate the arbiter metastability problem, we presented and analyzed majority voting of responses. Furthermore, a method to classify challenges into different robustness groups was introduced, to increase the response stability in the presence of environmental variations. Using the measurement data collected from 16 PUF on 12 FPGAs across 9 different temperature and power supply operating conditions, we measured the response robustness and quantified its trade-off with response uniqueness (entropy). Finally, the correlation between effects of power supply and

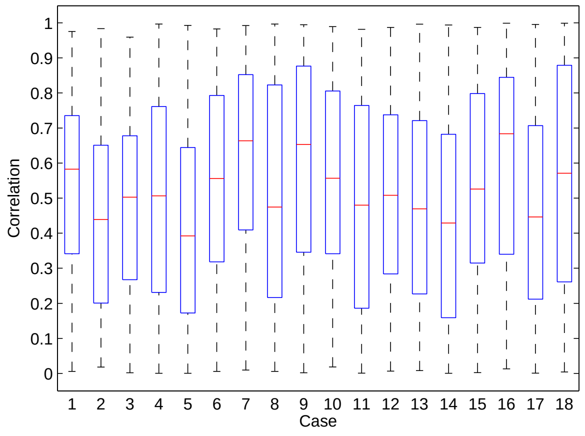


Fig. 22. The correlation between effect of temperature and power supply variations on responses for 18 different scenarios. Each box plot is made of response correlation values across 12x16 PUFs.

temperature variations on PUF responses were analyzed and quantified for in-field response error prediction.

REFERENCES

Y. Alkabani, F. Koushanfar, and M. Potkonjak. 2007. Remote activation of ICs for piracy prevention and digital right management. In *International Conference on Computer Aided Design (*ICCAD*)*. 674–677.

Y. M. Alkabani and F. Koushanfar. 2007. Active hardware metering for intellectual property protection and security. In *USENIX Security Symposium*. 1–16.

N. Beckmann and M. Potkonjak. 2009. Hardware-based public-key cryptography with public physically unclonable functions. In *IH*. 206–220.

E. Bergeron, M. Feeley, M.-A. Daigneault, and J.P. David. 2008. Using dynamic reconfiguration to implement high-resolution programmable delays on an FPGA. In *Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA*. 265 –268.

György Csaba, Xueming Ju, Qingqing Chen, Wolfgang Porod, Jürgen Schmidhuber, Ulf Schlichtmann, Paolo Lugli, and Ulrich Rührmair. 2009. On-Chip Electric Waves: An Analog Circuit Approach to Physical Uncloneable Functions. *Cryptology ePrint Archive* (2009).

L. Daihyun, J.W. Lee, B. Gassend, G.E. Suh, M. van Dijk, and S. Devadas. 2005. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, 10 (2005), 1200 – 1205.

Ken Eguro. 2010. SIRC: An Extensible Reconfigurable Computing Communication API. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 135–138.

B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. 2002. Silicon physical random functions. In *CCS*. 148–160.

J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls. 2007. FPGA intrinsic PUFs and their use for IP protection. In *CHES*. 63–80.

C. Jaeger, M. Algasinger, U. Ruhrmair, G. Csaba, and M. Stutzmann. 2010. Random pn-junctions for physical cryptography. *Applied Physics Letters* 96, 17 (2010), 172103 –172103–3.

S.S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls. 2008. The butterfly PUF protecting IP on every FPGA. In *HOST*. 67–70.

M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar. 2010a. Rapid FPGA Characterzation using Clock Synthesis and Signal Sparsity. In *ITC*.

M. Majzoobi, A. Elnably, and F. Koushanfar. 2010b. FPGA Time-bounded Unclonable Authentication. In *IH*.

M. Majzoobi and F. Koushanfar. 2011. Time-Bounded Authentication of FPGAs. *Information Forensics and Security, IEEE Transactions on* 6, 3 (sept. 2011), 1123 –1135.

M. Majzoobi, F Koushanfar, and S Devadas. 2010c. FPGA PUF using programmable delay lines. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*. 1–6.

M. Majzoobi, F. Koushanfar, and M. Potkonjak. 2008. Testing Techniques for Hardware Security. In *ITC*. 1–10.

M. Majzoobi, F. Koushanfar, and M. Potkonjak. 2009. Techniques for Design and Implementation of Secure Reconfigurable PUFs. *TRETS* 2, 1 (2009), 1–33.

M. Majzoobi, M. Rostami, F. Koushanfar, D.S. Wallach, and S. Devadas. 2012. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In *IEEE Symposium on Security and Privacy Workshops (SPW)*. 33 – 44.

Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. 2011. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs. In *Proceedings of the 18th ACM conference on Computer and communications security (CCS '11)*. ACM, New York, NY, USA, 111–124.

S. Morozov, A. Maiti, and P. Schaumont. 2010. *An Analysis of Delay Based* PUF *Implementations on* FPGA. Springer, 382–387.

R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. 2002. Physical one-way functions. *Science* 297 (2002), 2026–2030.

U. Rhrmair, F. Sehnke, J. Slter, G. Dror, S. Devadas, and J. Schmidhuber. 2010. Modeling Attacks on Physical Unclonable Functions. In *Conference on Computer and Communications Security*.

U. Ruhrmair. 2009. SIMPL system: on a public key variant of physical unclonable function. *Cryptology ePrint Archive* (2009).

G. Suh and S. Devadas. 2007. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *DAC*. 9–14.

G.E. Suh, C.W. O'Donnell, I. Sachdev, and S. Devadas. 2005. Design and implementation of the AEGIS single-chip secure processor using physical random functions. In *ISCA*. 25–36.

F. Koushanfar U. Ruhrmair, S. Devadas. 2011. *Security based on Physical Unclonability and Disorder*. Springer.