Adaptive Characterization and Emulation of Delay-based Physical Unclonable Functions using Statistical Models

Teng Xu, Dongfang Li, and Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles
{xuteng, lidongfang, miodrag}@cs.ucla.edu

ABSTRACT

It is commonly known that physical unclonable functions (PUFs) are hard to predict and hard to emulate. However, in this paper, we propose to use statistical models to adaptively characterize the delay-based PUFs, and use this as a starting point to emulate a delay-based PUF. The essential idea is that for any challenge C_A of a delay-based PUF A, there is a high probability of finding a paired challenge C_B. When apply C_B to another delay-based PUF B, it can produce the same output as applying C_A on PUF A. Our simulation results indicate more than 99% correctness for the PUF response prediction using characterization and 96% correctness using emulation. Finally, we implement and test the feasibility of our approach on the Xilinx Spartan-6 Field Programmable Gate Array (FPGA).

1. INTRODUCTION

Process variation (PV) is an important side effect in circuit manufacturing. Due to the effect of PV, the physical attributes of transistors (channel length, delay, leakage) become unique when integrated circuits are fabricated. On the other hand, physical unclonable functions (PUFs) are physical devices that have a random but deterministic mapping of inputs to outputs. They take advantage of the effect of PV to build unique and unclonable devices. When given a challenge to a PUF, it produces a response based on both the given challenge and the intrinsic physical properties of the PUF itself.

As a type of security primitive, PUFs have the advantages of low-power, fast-speed, small-area, unpredictability, and most importantly, unclonablity. Among the family of PUFs, the delay-based PUFs are widely studied and analyzed. They take advantage of the effect of PV on delays to create two theoretically identical paths to rival each other. A challenge vector is used to modify and to decide the two signal paths. To generate a response, a impulse signal is sent to both paths simultaneously, and an arbiter is at the end of the two paths taking them as two inputs. Depending on the delay of the two paths, one impulse signal triggers the arbiter first to generate an output.

The goal of out paper is to characterize and to emulate the delay-based PUF. Although it has been a common wisdom that PUF is a system that is hard to predict, there have been many emulation attacks proposed to characterize the delay-based PUF and to emulate the PUF in software. Most of them collect and observe many challenge-response pairs and adopt statistical approaches or machine learning techniques to build a statistical model from it. Based on the statical model, the response of the PUF can be predicted given any random challenge.

Compared to the previous work, the contribution of our paper comes from two aspects. The first aspect is that we have proposed an adaptive characterization technique on the base of traditional statistical model. Instead of purely statistically predicting the PUF response, our approach looks into the PUF structure and builds a delay model to estimate the delays in all the segments of the PUF. Consequently, it not only helps to predict the PUF response, but also provides insight into the delays in PUF segments which can be used as the premise for our PUF emulation. The second aspect is that instead of purely using software to emulate the PUF, we use hardware based emulation. To be more specific, we produce another piece of PUF to emulate the original PUF. The new PUF theoretically has a high probability to generate the same outputs as the original PUF.

Our work flow is shown in Figure 1. We first propose our statistical model for the adaptive characterization of PUF. The adaptive characterization acts as the premise of later hardware emulation. We combine the traditional statistical approach with our adaptive test to improve the characterization accuracy. Then based on the characterization results, we propose our algorithm to emulate a PUF using another PUF. This part is the major novelty of our paper. Finally, we use both the simulation and the real implementation data on Xilinx Spartan-6 FPGA to evaluate our approaches. It is a known fact that PUFs, including the delay-based PUFs, are vulnerable against environmental and operational variations, e.g., temperature, and supply voltages. Thus, it is difficult to duplicate the simulation results in the real hardware implementation. To explain this, we discuss the influence of noise on PUF implementation at the end of our paper.

2. RELATED WORK

In this section, we briefly review the related literature on process variation, PUFs, and PUF emulation.

DAC '15, June 07-11, 2015, San Francisco, CA, USA

http://dx.doi.org/10.1145/2744769.2744791

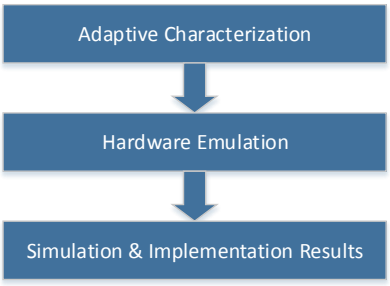

Figure 1: Work flow illustration.

2.1 Process Variation

Process variation is a widely recognized phenomenon in modern CMOS technologies [1]. PV exists among gates or transistors when the components are designed to be identical. But due to manufacturing limitations, the real produced components are different and unique in terms of structural and operational properties, such as propagation delay and leakage power. Many factors can cause PV, including wafer lattice structure imperfections, non-uniform dopant distribution, mask alignment, and chemical polishing [1].

2.2 PUFs

The concept of PUF is first proposed by Pappu et al using mesoscopic optical systems [2]. Devadas et al developed the first silicon PUFs through the use of intrinsic process variation in deep submicron integrated circuits [3]. A variety of PUFs are proposed after that, including arbiter-based (APUF) [3], ring oscillator-based (RO-PUF) [4], SRAM PUFs [5], and digital PUFs [6][7]. The delay-based FPGA PUF was first proposed in [8]. The core idea is take advantage of the intrinsic delay difference between LUTs to design the delay path.

2.3 PUF Emulation

Many technologies have been proposed to emulate PUFs. Lee first proposed to use statistical model to extract unique secret key information from PUFs [9]. Rührmair et al. developed numerical modeling attacks combing with machine learning techniques to break various types of PUFs [10]. Previous PUF emulation collects a number of challenge-response pairs of the PUFs and derives statistical models from there. But all of them employ software-based emulation. Our work is the first to use hardware, more specifically, another PUF to emulate the original characterized PUF.

3. PUF MODEL

The PUF model we are using is shown in Figure 2. The basic structure of the n-bit PUF consists n delay segments. Two identically designed paths are generated by connecting delay components from each segment, and an arbiter is at the end of the two paths.

The two paths of the delay-based PUF can be modified using the control bit of each segment. When the control bit is 0, the two paths will not shuffle. When the control bit is 1, the two paths swap. For example, in Figure 2, a connects to c, b connects to d when $c_1 = 0$, and a connects to d, b connects to c when $c_1 = 1$. If we denote the propagation delays from the input ports to the output ports as t_{ac}, t_{ad}, t_{bc}, t_{bd}, then t_{ac} needs to be designed notoriously equals to t_{bd}, and t_{ad} notoriously equals to t_{bc}. After manufacturing, the effect of PV causes unpredictable delay differences between them.

When a n bit challenge $(c_1c_2...c_{n-1}c_n)$ is given to the PUF, two identically designed paths are generated. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of PV, one path will reach the arbiter earlier, and an output bit R is generated as the PUF response. For a n bit PUF, there exists 2^n challenge-response pairs.

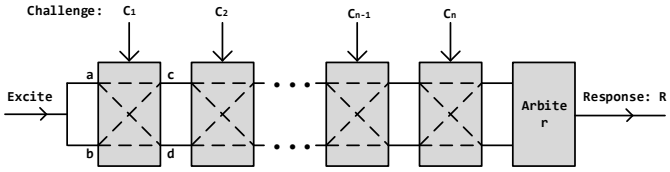

Figure 2: The model of delay-based PUF with a n-bit challenge.

4. PUF CHARACTERIZATION

The core idea of our PUF characterization is to first observe a number of challenge-response pairs to build a statistical model, then make predictions on the PUF response based on the model. Using this as a starting point, we further propose two algorithms for adaptive characterization: respectively "adaptive challenges", and "compensate challenges".

4.1 Statistical Model

Assume that it is a n-bit PUF that we are characterizing. For the ith segment, there exists 4 delays: t^i_{ac}, t^i_{ad}, t^i_{bc}, and t^i_{bd} (i∈{1,2,...n}). The positions of a, b, c, d can be found in Figure 2. After collecting some number of challenge-response pairs, we statistically calculate the following 4 probabilities for each segment as shown in Equation 1. Note that for each segment, $P^i_{ac} + P^i_{bd} = 1$ as well as $P^i_{ad} + P^i_{bc} = 1$ (i∈{1,2,...n}).

$$\begin{aligned} &P^i_{ac}(path\ p_1\ is\ longer \mid p_1\ contains\ t^i_{ac}) \\ &P^i_{ad}(path\ p_2\ is\ longer \mid p_2\ contains\ t^i_{ad}) \\ &P^i_{bc}(path\ p_3\ is\ longer \mid p_3\ contains\ t^i_{bc}) \\ &P^i_{bd}(path\ p_4\ is\ longer \mid p_4\ contains\ t^i_{bd}) \end{aligned} \quad (1)$$

We claim that when a delay in a segment is longer than its rival delay, e.g., t^i_{ac} and t^i_{bd} are rivals, so are t^i_{ad} and t^i_{bc} (i∈{1,2,...n}), the path that contains the delay will have larger probability to be longer the opposite path. The intuition is that each path is a sum of single delays from each segment, consequently, if the delay in one segment is longer than its rival, the path that contains this delay has better chance to be longer than the opposite path which contains the relatively shorter rival delay in that segment. Therefore, this assumes there exists correlations between the real segment delays and the probabilities listed in Equation 1.

According to Figure 2, assume that if the path $(path_0)$ that reaches the upper port of the arbiter is earlier, the output equals 0. Otherwise, if the other path $(path_1)$ that

reaches the lower port of the arbiter is earlier, the output equals 1. The delay of the two paths can be written in the format shown in Equation 2. The definition of $parity(i)$ can be found in Equation 3. It represents the parity of the times of switching after segment i for a given challenge. Because the output is decided by the relation between p_0 and p_1, we represent the output of the PUF using only the difference between the rival delays in each segment. The simplified representation can be expressed in Equation 4.

$$
\begin{aligned}
p_0 &= \sum_{i=1}^{n} t_0^i \\
p_1 &= \sum_{i=1}^{n} t_1^i \\
t_0^i = t_{ac}^i,\ t_1^i = t_{bd}^i,\ &when\ c_i = 0\ and\ parity(i) = 0; \\
t_0^i = t_{bc}^i,\ t_1^i = t_{ad}^i,\ &when\ c_i = 1\ and\ parity(i) = 0; \\
t_0^i = t_{bd}^i,\ t_1^i = t_{ac}^i,\ &when\ c_i = 0\ and\ parity(i) = 1; \\
t_0^i = t_{ad}^i,\ t_1^i = t_{bc}^i,\ &when\ c_i = 1\ and\ parity(i) = 1;
\end{aligned}
\tag{2}
$$

$$
Parity(i) = \begin{cases} 0,\ even\ 1s\ in\ \{c_{i+1}...c_n\},\ i \in \{1, 2, ...n-1\} \\ 1,\ odd\ 1s\ in\ \{c_{i+1}...c_n\},\ i \in \{1, 2, ...n-1\} \\ 0,\ i = n \end{cases}
\tag{3}
$$

$$
Output = \begin{cases} 0,\ \sum_{i=1}^{n} t_{diff}^i < 0, \\ 1,\ \sum_{i=1}^{n} t_{diff}^i > 0,\ t_{diff}^i = t_0^i - t_1^i \end{cases}
\tag{4}
$$

Equation 2, 3, and 4 have mathematically defined our PUF model. Based on the model, after collecting a number of challenge-response pairs, the probabilities in Equation 1 can be summarized statistically. Since Our goal is to characterize the PUF, therefore, the major question now is how to derive the real delays from the probabilities. To be more specific, since it is the delay difference that decides the outputs of the PUF, our target becomes to derive the t_{diff}^i of each segment in Equation 4 from the probabilities in Equation 1. To achieve this, we simulate to find a suitable regression model.

Assume that each delay follows a Gaussian distribution due to the effect of PV. After collecting 100,000 challenge-response pairs for a 64-bit PUF, we calculate the probabilities in Equation 1 for the delays in all the segments. By repeating the test on 100,000 PUFs, we plot all the calculated probabilities in x-axis and the corresponding delay difference in y-axis. Note that because $P_{ac}^i + P_{bd}^i = 1$ and $P_{ad}^i + P_{bc}^i = 1$ (i∈{1,2,...n}), in order to avoid repeating, we only plot the probabilities which are larger than 0.5. Due to the same reason, we only plot the absolute value of t_{diff}^i. The regression plot in Figure 3 indicates a perfect linear mapping between the delay difference and the probability. The results strong suggest that a statistical delay model can be derived from the collected probabilities using linear regression.

The derived delay model can be used to predict PUF responses. When given a random challenge, the delay of the two paths can be calculated and compared using the estimated delay differences in the segments. We will discuss the accuracy of the prediction in Section 6 and Section 7 using both simulation and implementation.

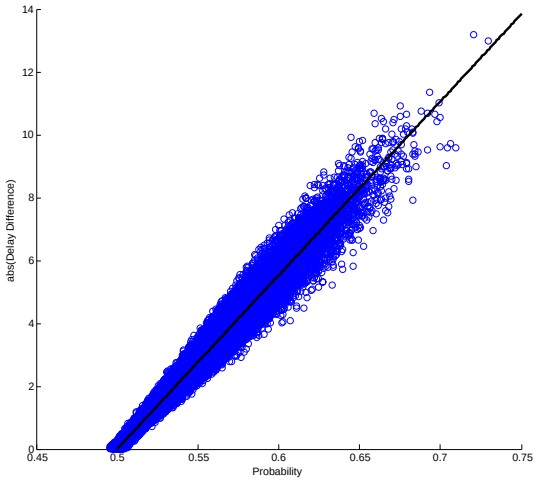

Figure 3: Regression plot between the delay difference and the probability.

4.2 Adaptive Challenges

On the base of the proposed statistical model, we propose two approaches to improve the model accuracy. The first approach is to use adaptive challenges.

According to Equation 4, there exists challenges that can produce two paths with ultra-small delay difference. Motivated by such challenges, there exists some sets of challenges that produce almost identical delay over only part of the segments. As a motivational example, we consider challenge $0c_2c_3...c_n$. Normally, the delay difference of the two paths can be represented as $(t_{ac}^0 + t_{rest}) - (t_{bd}^0 + t'_{rest})$. However, if $c_2c_3...c_n$ results $t_{rest} \approx t'_{rest}$, the delay difference can be simplified as $t_{ac}^0 - t_{bd}^0$. Consequently, from the output of the PUF, the relation between t_{ac}^0 and t_{bd}^0 can be concluded. We define such challenges as adaptive challenges, because they can be adaptively adjusted to test over only part of the segments.

In order to find such challenges, we need to use the statistical model. Given the delays of each segment from the model, to find the challenges with optimal delay difference (closest to 0) to cover m segments, the time complexity is exponential. As an alternative, we use greedy algorithm to find adaptive challenges. The basic idea is to first sort the delay difference of each segment. Then starting from the segment with largest delay difference to the segments with smallest delay difference, we put the challenge bit in such a way that the current segment compensates the sum of already considered segments' delay difference as close to 0 as possible.

An inequality can be derived from each adaptive challenge. For a n-bit PUF, if an adaptive challenge produces identical delay difference over m segments, then 2^{n-m} inequalities can be created, because the rest $n - m$ challenge bits can have any combination. Each generated inequality can be used to verify and to modify the statistical model. Once we find the current statistical model violates an inequality, we modify the model to the smallest extend to re-meet the inequalities.

4.3 Compensate Challenges

A drawback in the adaptive challenge is that the inequal-

ities it introduces only compare the delay difference with 0. For instance, if the real delay difference is 0.2, and in the statistical model the delay difference is calculated as 1.0. Despite the error is large, the use of adaptive challenge can not test it out. Because as long as the real delay difference has the same sign as the predicted one, the output will show the same. Due to this reason, adaptive challenges miss many cases to reveal the errors in the statistical model.

In order to better quantify the delay difference, we propose our second method: compensate challenges. The basic idea is similar to the adaptive challenge, we still start from a motivational example. We consider the challenge $0c_2c_3...c_n$. The delay difference of the two paths can be represented as $(t^0_{ac} + t_{rest}) - (t^0_{bd} + t'_{rest})$. In the case of compensate challenge, instead of designing $c_2c_3...c_n$ to make $t_{rest} \approx t'_{rest}$, we make $t_{rest} - t'_{rest} \approx \alpha$. Therefore, the delay difference can be simplified as $t^0_{ac} - t^0_{bd} + \alpha$. From the output of the PUF, the relation between t^0_{ac} and $t^0_{bd} - \alpha$ can be concluded. We define the challenges that are intentionally designed to use part of the segments to compensate the rest delay difference as compensate challenges.

When designing a compensate challenge, the primary goal is to compensate the original delay difference as accurate as possible, so that the real delay difference can be accurately tested. For example, in the example of $0c_2c_3...c_n$, we want to design α to be as close to $t^0_{bd} - t^0_{ac}$ as possible. The algorithm to find such compensate challenges can be the same as finding adaptive challenges.

Compared to the adaptive challenge, a most significant improvement of the compensate challenge is that it better quantifies the delay difference. This is especially helpful when dealing with segments with large delay difference. On one hand, such segments with large delay difference have big influence on the output. On the other hand, they are not likely to violate the inequalities generated by adaptive challenges because the delay difference of those segments are larger than the delay difference of the rest segments, thus has dominate effects on the inequality. However, with the use of compensate challenges, we can easily create α close to the large delay difference by combining a few segments.

Both the idea of adaptive challenges and compensate challenges are built on the top of basic statistical model, aiming at improving the model accuracy. Our characterization algorithm has the advantage of being fast, scalable, and can be used to derive delay models for each individual segments. These properties are especially important for our next step PUF emulation.

5. PUF EMULATION

The novelty of our PUF emulation is that we are the first to use one PUF to emulate another PUF using the delay model. In order to achieve this, we first set up a few preconditions. Assume that we are using a m-bit PUF A to emulate a n-bit PUF B. The prerequisites are shown below.

- PUF A and PUF B are characterized.
- $m \geq n$.

The theoretical base for our algorithm is delay scaling which will be explained in the next part. Starting from there, two steps are required in PUF emulation, challenge mapping and segment restructuring. Among which, the core idea is the challenge mapping, segment restructuring is a supplemental procedure to increase emulation accuracy.

5.1 Delay Scaling

Equation 5 shows the delays for PUF A and PUF B. The delay in each segment of PUF B is a scaling of the delay of the corresponding segment in PUF A. We claim that PUF A and PUF B generates a same response when given the same challenge. The prove is straightforward. According to Equation 4, the PUF response is decided by the sign of the sum of delay difference. When scaling all the delay segments by α at the same time, the path delay difference is also scaled by α. But this will not change the sign of the path delay difference, thus the response keeps the same.

$$\begin{aligned} PUF\ A &= \{t^i_{ac}, t^i_{ad}, t^i_{bc}, t^i_{bd}\} \\ PUF\ B &= \{t^i_{ac}/\alpha, t^i_{ad}/\alpha, t^i_{bc}/\alpha, t^i_{bd}/\alpha\},\ \alpha > 0 \end{aligned} \tag{5}$$

5.2 Challenge Mapping

The basic idea of challenge mapping is that given a challenge C_A for PUF A, there always exists a mapped challenge C_B for PUF B that has high likelihood to produce the same output. Now assume that both PUF A and B are n-bits. Derived from Equation 4, we use the following notions to represent the path delay difference as shown in Equation 6. In the function, the path delay difference is represented as a sum of plus and minus of the absolute value of each segment's delay difference. The sign before the absolute delay difference $|t^i_{diff}|$ is decided by s_i which is the sum of $parity(i)$ and the sign of t^i_{diff-A} as shown in Equation 7. $sign = 0$ when the t_{diff} is positive, and $sign = 1$ otherwise.

$$\begin{aligned} T_{diff-A} &= \sum_{i=1}^{n} (-1)^{s^A_i} |t^i_{diff-A}| \\ T_{diff-B} &= \sum_{i=1}^{n} (-1)^{s^B_i} |t^i_{diff-B}| \end{aligned} \tag{6}$$

$$\begin{aligned} s^A_i &= parity^A(i) + sign(t^i_{diff-A}) \\ s^B_i &= parity^B(i) + sign(t^i_{diff-B}) \end{aligned} \tag{7}$$

The motivation of challenge mapping starts from a question: for a t^i_{diff-A} in PUF A, is it possible to find a t^j_{diff-B} in PUF B to replace its position when emulating? It requires the contribution of t^i_{diff-A} to T_{diff-A} is similar to the contribution of t^j_{diff-B} to T_{diff-B}. Our research suggests that for every t^i_{diff-A} in PUF A, there exists such a t^j_{diff-B} in PUF B to match it. Additionally, in order to make the delay difference of each matched segment has the same s_i, for any challenge C_A for PUF A, we propose an algorithm to find a matched challenge C_B for PUF B.

The algorithm to find mapped challenge is shown in Algorithm 1. The main idea is that we want to make sure the segment with large(small) difference in PUF A is matched with the segment of large(small) delay difference in PUF B. To match means that they have the same sign when calculating the total path delay difference. To achieve this, we sort the absolute value of all the delay difference in a non-increasing order first, then start matching from the largest difference delay segment until the smallest difference delay segment by dynamically adjusting the challenges. It is not guaranteed that we can always find a mapped challenge that completely matches all the pairs of delay segments. But since the segments with large delay difference have more significant impact on the final path delay difference, the process starts

from the largest difference delay segment and continues in a non-increasing order.

Algorithm 1 Challenge Mapping

Input: A challenge for PUF A: $C_A = c_1^A ... c_n^A$,
t^i_{diff-A} for segment i in PUF A,
t^i_{diff-B} for segment i in PUF B,
Output: A mapped challenge for PUF B: $C_B = c_1^B ... c_n^B$,

1. Sort $|t^i_{diff-A}|$ in a non-increasing order to $|t^{a_i}_{diff-A}|$.
2. Sort $|t^i_{diff-B}|$ in a non-increasing order to $|t^{b_i}_{diff-A}|$.
3. **For** i from 1 to n:
4. Calculate $s^A_{a_i}$ for $|t^{a_i}_{diff-A}|$.
5. Find $c_1^B ... c_n^B$ to meet all $s^B_{b_j} == s^A_{a_j} (j \leq i)$.
6. **If** $c_1^B ... c_n^B$ exists:
7. **continue**.
8. **else**:
9. **Return** $c_1^B ... c_n^B$.
10. **Return** $c_1^B ... c_n^B$.

5.3 Segment Restructuring

On the base of challenge mapping, we propose the idea of segment restructuring to improve the emulation performance. One problem for challenge mapping is that the matched segments are not scaled by the same or similar ratio. For example, the first and second largest delay difference for PUF A are 100 and 90, while for PUF B, the difference are 50 and 30. Therefore, the scaling ratio for the largest difference segment is $100/50 = 2$ while for the second largest is $90/30 = 3$. This problem exists across all the matched segments. The proposal of segment restructuring can leverage it.

Instead of using one segment in PUF B to match one segment in PUF A, segment restructuring proposes to use the sum of a few segments in PUF B to match one segment in PUF A. Still using the same example, for the second largest difference segment with delay difference 30 in PUF B, assume that there exists two segments with delay difference 5 and 20 adjacent to that segment. In this case, by properly assigning challenge bit, we can achieve delay difference $30 - 5 + 20 = 45$ over the 3 segments together. If we regard the above three segments together as a single segment to match the second largest delay difference segment in PUF A, we can have delay ratio of $90/45 = 2$ which is the same ratio as the segment with largest delay difference.

Segment restructuring can significantly decrease the deviation between the scaling ratios, but the premise of this approach is to require a longer PUF B to emulate the PUF A. Theoretically, the longer PUF B is used, the smaller deviation can be achieved. However, one side-effect to use a longer PUF B is that there will be some segments left if not all the segments are used to match PUF A. Our solution is simple, we first make sure that we use all the segments in PUF B with large delay difference, thus the ones left will not have big influence in response. Then we put the challenge bits of the rest segments in such a way that the delay differences in those segments compensate to each other to make the total effect as close to 0 as possible. With the above steps, the side-effect of unused segments in PUF B is reduced to be negligible.

To summarize the process of PUF emulation, after the characterization of a n-bit PUF A and a m-bit PUF B, for any random challenge C_A given to PUF A with a response R_A, we use challenge mapping and segment restructuring to form a mapped challenge C_B to PUF B. This challenge has high likelihood to generate $R_B = R_A$ in PUF B.

6. SIMULATION RESULTS

In the following simulation, we assume that the delay of each PUF segment follows a gaussian distribution. For each result, we repeat test on 1,000 individual PUFs or paris of PUFs and take the average value to present. We illustrate the results for characterization and emulation respectively.

6.1 Characterization Results

Table 1 shows the prediction accuracy using our PUF characterization algorithm. For each instance, we vary the number of challenge-response pairs (CRPs) to test on PUFs of different bits, and compare the real response of the PUF and the predicted response based on the characterized PUF model. From there, we calculate the correct rate of our response prediction.

CRPs	1,000	10,000	100,000	1,000,000
32-bits	96.7%	99.0%	99.7%	99.9%
64-bits	95.0%	98.3%	99.4%	99.9%
128-bits	92.6%	97.2%	98.8%	99.7%

Table 1: Response prediction accuracy using PUF characterization model under various number of CRPs.

In real scenario, due to the effect of noise, e.g., temperature, supply voltage, a small portion of the response will be affected with error. To simulate this, we assume a percentage of responses used for PUF characterization were wrong, and their bit values were flipped. Afterwards, we evaluate the predication accuracy on the error-free test sets. For each test, we use 1,000,000 CRPs. The results are demonstrated in Tables 2.

Error Rate	0%	2%	5%	10%
32-bits	99.9%	99.4%	98.8%	97.7%
64-bits	99.9%	99.2%	98.5%	97.5%
128-bits	99.7%	99.2%	98.6%	97.3%

Table 2: Response prediction accuracy using PUF characterization model under various error rate.

6.2 Emulation Results

Table 3 shows the prediction accuracy using PUF emulation. Assume that PUF B is used to emulate PUF A. Each row is the number of bits of PUF A, and each column is the number of bits of PUF B. Table 3 shows the effect of increasing the size of PUF B. The characterization model we are using is based on the statistical model achieved using 1,000,000 CRPs.

We also consider the prediction accuracy when PUF responses are tested with errors, this will directly affect the characterization results, consequently influencing emulation results. The result in Table 4 is based on the emulation that PUF A and PUF B have the same number of bits.

# of bits	32	64	128	256
32-bits	89.6%	92.8%	95.6%	96.2%
64-bits	-	92.3%	95.1%	96.6%
128-bits	-	-	95.9%	97.1%

Table 3: Response prediction accuracy using PUF emulation: emulate PUF A using PUF B of various bits.

Error Rate	0%	2%	5%	10%
32-bits	89.6%	87.6%	85.8%	84.9%
64-bits	92.3%	91.0%	89.4%	87.5%
128-bits	95.9%	93.8%	92.2%	90.4%

Table 4: Response prediction accuracy using PUF emulation under various error rate.

7. IMPLEMENTATION RESULTS ON FPGA

In this section we implement the delay-based PUF on FPGA to test our characterization and emulation algorithms. The platform we are using is Xilinx Spartan-6 FPGA. Because Spartan-6 uses 6-input look-up tables (LUT6) as the basic logic element, we separate each LUT6 into two LUT5 units to act as the two rival delays in each segment. To accommodate the opposite input order, the memory contents of each LUT5 are adjusted accordingly. This architecture allows each segment to be efficiently mapped to a single LUT6.

We repeat the tests for simulation on the implementation data. The results shown in Table 5 shows the prediction accuracy using our PUF characterization algorithm. Compared to Table 1, the accuracy drops, and the best accuracy is at around 85%. Table 6 retests the emulation results. When compared to Table 3, similarly, the prediction accuracy decreases with the best we can get at around 82%.

We can conclude that the result of PUF implementation is highly affected by the noise, including various environmental variations. Although the results shown in Table 2 and Table 4 suggest some error rate will not significantly effect the PUF characterization and emulation model. However, in real PUFs, there is no such error-free test set. Thus, even though the model is relatively accurate, any errors in the real PUF's response will directly result in errors in the test set, thus dramatically decrease the prediction accuracy.

CRPs	1,000	10,000	100,000	1,000,000
32-bits	76.5%	81.1%	83.8%	85.6%
64-bits	73.6%	76.9%	79.7%	83.7%
128-bits	70.3%	74.0%	76.5%	82.3%

Table 5: Collected data with PUF implemented on FPGA. Response prediction accuracy using PUF characterization model under various number of CRPs.

8. CONCLUSION

We have two major contributions in this paper. The first is that we have proposed a new approach to characterize a delay-based PUF by combining the statistical model with the adaptive challenge tests. The second is that on the bases of our characterization results, we propose our approach to emulate a delay-based PUF using another delay-based PUF. Our simulation result suggests that the accuracy of response

# of bits	32	64	128	256
32-bits	72.1%	76.8%	77.8%	80.7%
64-bits	-	75.5%	78.2%	81.1%
128-bits	-	-	78.1%	82.5%

Table 6: Collected data with PUF implemented on FPGA. Response prediction accuracy using PUF emulation: emulate PUF A using PUF B of various bits.

prediction based on our characterization model can easily reach 99% without using an outrageous number of CRPs. Meanwhile, the prediction accuracy based on PUF emulation can reach around 96%-97%. Due to the reason of noise in PUF implementation, our implementation results on Xilinx Spartan-6 FPGA suggest a drop in prediction accuracy to 82%-86% in characterization and 80%-83% in emulation. Our work indicate that the traditional delay-based PUF is insecure in a sense that it can be accurately modeled, emulated, even functionally replicated by another delay-based PUF.

9. REFERENCES

[1] K. Bernstein, et al, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM journal of research and development* 50.4.5, pp. 433-449, 2006.

[2] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026-2030, 2002.

[3] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," *ACM Conference on Computer and Communications Security*, pp. 148-160, 2002.

[4] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," *Design Automation Conference (DAC)*, pp. 9-14, 2007.

[5] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 63-80, 2007.

[6] T. Xu, M. Potkonjak, "Robust and Flexible FPGA-based Digital PUF," *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1-6, 2014.

[7] T. Xu, J. B. Wendt and M. Potkonjak, "Secure Remote Sensing and Communication using Digital PUFs," *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 173-184, 2014.

[8] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, IEEE, 2010.

[9] J. W. Lee, et al, "A technique to build a secret key in integrated circuits for identification and authentication applications," *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on.* IEEE, 2004.

[10] U. Rührmair, et al, "Modeling attacks on physical unclonable functions," *ACM conference on Computer and communications security.* ACM, 2010.