An Ultra-Low Energy PUF Matching Security Platform Using Programmable Delay Lines

Teng Xu, Hongxiang Gu, Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles
{xuteng, hxgu, miodrag}@cs.ucla.edu

***Abstract*—We have proposed a new security platform: physical unclonable function (PUF) matching using programmable delay lines (PDL). Our platform inherits good security properties of standard PUFs, such as low energy, low delay, and unclonability. However, standard PUF-based security protocols induce high computational resources of at least one involved party. To resolve this issue, we take advantage of PDL technology to match standard PUFs in such a way that two PUFs have the same challenge response mapping function. The matched pair of PUFs enables a majority of protocols to be executed in an ultra low energy, low latency manner for all the involved parties.**

I. Introduction

The growing number of mobile devices have imposed new security requirements. Most mobile devices are highly constrained by their battery life, thus they require embedded security platforms to be ultra low energy. On the other hand, current trend shows the state of the art mobile devices are much thinner and smaller in size to achieve higher portability. Thus, area overhead becomes especially important. Therefore, when protecting mobile electronics using security platforms, it is crucial to ensure that the design is ultra-low power and compact in size.

The physical unclonable function (PUF) as a hardware security primitive meets the above requirements. A PUF is a hardware implemented one-way function which takes advantage of the effect of process variation to guarantee the uniqueness of each individual piece. The input-output mapping function of an individual PUF is deterministic, but unpredictable due to the fact that process variation is not predictable. As a hardware security primitive, PUF employs much lower overhead comparing to traditional cryptographic approaches. More importantly, a PUF itself is unclonable which physically provides one more level of protection.

PUFs enable a variety of security protocols, such as message encryption/decryption, authentication, and public key communication. Unfortunately, a PUF has its own shortcomings when applied in protocols. It is because such protocols usually require at least two parties to share the same input-output mapping function. However, a PUF can not be copied due to its unclonability, consequently, one of the parties can only simulate the mapping function which is significantly slower and takes more energy comparing to directly using the PUF device.

Our technical goal is to demonstrate a new platform for PUF matching to resolve the above issues of standard PUFs. It preserves all advantages of traditional PUFs but allows all parties in the protocol to employ low-cost communication. The key idea is to match multiple standard PUFs in such a way that they have the same input-output mapping function. We achieve our goal by using the look-up table (LUT) based PDL. The PDL are applied to tune and modify the delays of each segment in the standard PUFs, and eventually in hope to create multiple PUFs with the same functionality. The generated matched PUFs preserve all properties of standard PUFs, and the probability of a third party creating a same copy of the PUF must be negligible. By creating a set of PUFs that are matched to each other, each party is able to possess one PUF copy from the set, thus facilitating low-cost communication.

In this paper, we first review the related literature of PUFs and PDL. Then we give the preliminaries of the basic PUF model we are using as well as the design of PDL on field-programmable gate array (FPGA). We also provide a motivational example of our PUF matching scheme. Later we demonstrate in detail on the architecture of our system, PUF matching algorithm, and the protocols that can be enabled by our security platform in the subsequent sections. Lastly, we depict our implementation detail and present the tested results.

II. Related Work

We review the previous literatures on process variation, PUFs, PDL, and lightweight security primitives in this section.

A. 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.

B. Physical Unclonebale Function (PUF)

PUF is first proposed by Pappu et al. using mesoscopic optical systems [2]. Devadas et al. from MIT developed the first silicon PUFs through the use of intrinsic process variation in deep submicron integrated circuits [3]. A variety of other types of PUFs are proposed afterwards, including arbiter PUFs [3], ring oscillator PUFs [4], SRAM PUFs [5], butterfly PUFs [6], and digital PUF [7][8]. All of the above PUFs are designed by utilizing the effect of process variation on different platforms.

Numerous traditional protocols can be interpreted using PUFs, ranging from the traditional security key communication and authentication [9] to more sophisticated public key communication [10]. The key idea is to employ the high unpredictability of PUF responses to secure the information. However, conventionally only one party has the actual hardware of PUF (e.g., decryption party). Thus, all the rest communication parties can only simulate the PUF (e.g., encryption party), which takes relatively high timing/energy overhead comparing to directly using the PUF hardware. Our new proposed PUF matching platform can leverage the above drawbacks.

C. Programmable Delay Lines (PDL)

Programmable delay lines are a series of digital delay lines with electrically programmable and trimmable delay times [11]. The design of PDL implemented on FPGA is proposed by Majzoobi et al [12]. They take advantage of the lookup table (LUT) internal structure on FPGA to create delay bias and use it to generate the controllable delay.

To measure the delay bias, high precision delay testing technique with pico-second resolution on FPGA is required. Some previous work includes: Raychowdhury et al. [13] proposed on-chip delay measurement hardware techniques to estimate the segment path delay. Tsai et al. [14] proposed a vernier delay lines based built-In delay measurement circuit with a small area overhead that can provide high-resolution delay measurement. Majzoobi et al. [15] designed a timing characterization circuit with clock synthesis that can measure pico-second resolution on FPGA.

D. Lightweight Security Primitives

There are a number of lightweight protocols as well as security primitives proposed. Meanwhile, many of the previous works also focus on proposing low-power architecture for already existed cryptographic cyphers, e.g., SHA-1, AES [16]. In terms of lightweight PUFs, Koushanfar et al. have developed a new methodology for low-power PUF design which enables multiple delay lines for response creation [17]. The matched public PUF (mPPUF), as a new type of low-power PUF, has been proposed by Meguerdichian et al. which takes advantage of both process variation and device aging to create pairs of identical PUFs [18]. More recently, Xu. et al proposed the digital bidirectional function which works completely in the digital domain with even lower power consumption compared to the traditional analog security primitives [19] [20].

III. Preliminaries

A. PUF Model

The PUFs we use for matching are standard arbiter PUFs. Figure 1 shows the schematic diagram of the PUF model. The basic structure of the n-bit PUF consists n delay segments. The two propagation delays in the ith segment are denoted as d_i^0 and d_i^1 respectively. The two delays are designed notoriously equal to each other, but after manufacturing, the effect of process variation will cause unpredictable delay difference between them. When built on FPGA, the upper delay and the lower delay in each segment are directly implemented using the LUTs with the same size. 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 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 1, if the control bit of the ith segment is 0, then d_i^0 stays with the upper path and d_i^1 stays with the lower path. However, if the control bit is 1, d_i^0 shuffles to the lower path and d_i^1 shuffles to the upper path. Note that when the shuffling happens, all the delays that connect prior to $d_i^0(d_i^1)$ will be shuffled at the same time.

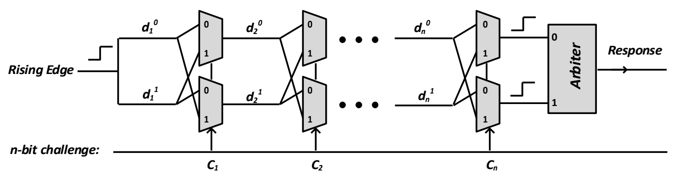

Fig. 1: The model of delay-based PUF with an n-bit challenge.

The vector consists of all control bits is denoted as the PUF challenge. When an n-bit challenge $(c_1c_2...c_{n-1}c_n)$ is provided 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 process variation, the signal travels along one of the two paths will reach the arbiter earlier, generating a corresponding arbiter output denoted as the PUF response. For an n bit PUF, there exists 2^n challenge-response pairs.

B. PDL on FPGA

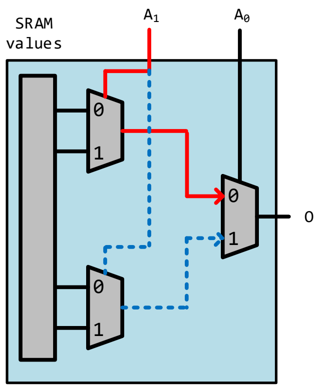

Fig. 2: The internal structure of a 2-input LUT.

The PDL design on FPGA is proposed by Majzoobi et al [12]. It is implemented by a single LUT. Figure 2 shows an example LUT with 2 selection bits A_1 and A_0. Now consider two scenarios respectively when $A_0 = 0$ and $A_0 = 1$. The propagation delay from A_1 to O when $A_0 = 0$ is depicted in the solid red line, and the propagation delay when $A_0 = 1$ is marked in the dashed blue line. From the figure, we can clearly see that due to the asymmetric structure in the LUT, the propagation delay in the blue line is slightly larger than the propagation delay in the red line. Many modern FPGAs have on board 6 selection-bit LUTs, the delay difference caused by asymmetricity is in pico-second resolution. We take advantage of such small delay difference in the arbiter PUF segments to tune and match the PUFs.

IV. A Motivational Example

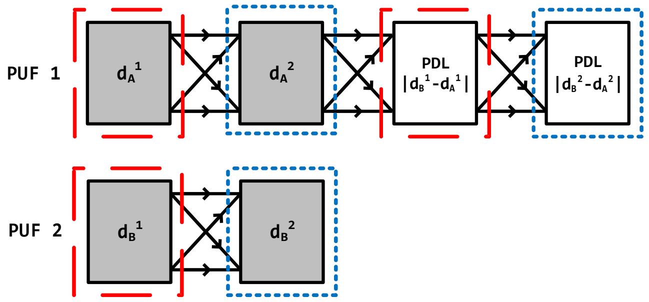

Fig. 3: An example of PUF matching using PDL.

We present a motivational example of our PUF matching scheme in Figure 3. Both PUF 1 and PUF 2 originally have two segments. In order to match their first segments, respectively with d_A^1 delay difference in PUF 1 and d_B^1 in PUF 2, we add an additional segment built by the PDL to PUF 1 with the delay of $|d_B^1 - d_A^1|$. By properly combining the existing first segment in PUF 1 (d_A^1) with the additional PDL segment ($|d_B^1 - d_A^1|$), they together can represent the first segment in PUF 2 which has the delay of d_B^1. The similar matching process can be repeated on the second segment of PUF 1 and 2. To summarize, the basic idea for matching two $n - bit$ PUFs is to create n additional segments using PDL and attach them by the end of a PUF. The delays of PDL are designed in such a way that each one of them can be combined with an existing ith segment in the current PUF so that they together realizes the same delay as the ith segment in the other PUF.

V. PUF Matching

The goal of PUF matching is to match at least 2 PUFs in such a way that they all have the same challenge-response mapping function. The matching process can be divided into four steps as shown in Figure 4, respectively PUF characterization, delay information exchange, add extra PDL segments, and challenge reassignment. We demonstrate each procedure separately in the following parts. For simplification, we start with PUF matching between two parties. We assume that Alice owns an n-bit arbiter PUF A and Bob owns an n-bit arbiter PUF B. The goal is to match PUF A with PUF B.

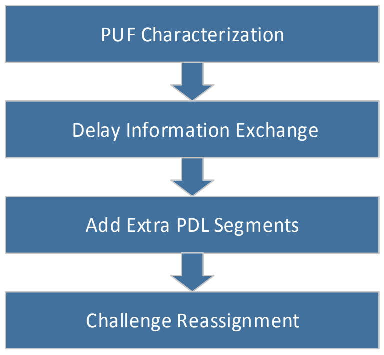

Fig. 4: The flow of PUF matching.

A. PUF Characterization

It has been well known that standard arbiter PUFs can be characterized using statistical methods. Many technologies have been proposed to characterize PUFs. Rührmair et al. developed numerical modeling attacks combining with machine learning techniques to break various types of PUFs [21]. Xu et al. proposed to use statistical models and regression techniques to retrieve the delay characterization of each segment in the arbiter PUF [22]. Both PUF characterization approaches collect a number of CRPs of the PUFs and derive statistical models from there. To match PUF A with PUF B, the first step is that Alice and Bob need to characterize PUF respectively.

For the PUF model shown in Figure 1, the ith segment has two delays, d_i^0 and d_i^1, we denote the delay difference between d_i^0 and d_i^1 as d_i^{diff} as shown in Equation 1.

$$d_i^{diff} = d_i^0 - d_i^1, i \in \{1, 2, ...n\} \tag{1}$$

Using the notation of d_i^{diff}, the total delay difference (T_{diff}) between two PUF paths can be represented in Equation 2. The parity function denotes the number of times that switching happens after the current segment, which is decided by the number of ones in the challenge. Equation 2 indicates that the total delay difference of the two PUF paths can be denoted as the plus or minus of the delay difference of each segment.

$$\begin{gathered} T_{diff} = \sum_{i=1}^{n} (-1)^{parity(i)} * d_i^{diff} \\ parity(i) = \begin{cases} 0, & even\ 1s\ in\ \{c_i...c_n\} \\ 1, & odd\ 1s\ in\ \{c_i...c_n\} \end{cases} \end{gathered} \tag{2}$$

We rewrite the Equation 2 to Equation 3 using vectors formats. Note that each challenge corresponds to a unique P vector which consists of only 1 and -1.

$$\begin{gathered} T_{diff} = P \cdot D \\ P = \{(-1)^{parity(1)}, (-1)^{parity(2)}, ...(-1)^{parity(n)}\} \\ D = \{d_1^{diff}, d_2^{diff}, ...d_n^{diff}\} \end{gathered} \tag{3}$$

Using the above arbiter PUF model, the process of PUF characterization is easily done by collecting a number of PUF challenge-response pairs, and building a statistical model based on them. In our characterization, we follow the notations in Equation 3 and measure a set of T given different P values, so that to create linear equations regarding d_i^{diff}. We solve the equations and retrieve the delay difference $(d_i^0 - d_i^1)$ for each PUF segment.

B. Delay Information Exchange

Bob and Alice exchange the retrieved delay difference information in the second step. This process is only one-time and can be done through standard cryptography. Based on the exchanged characterization information, Alice and Bob need to define a PUF matching template, which is used as the target PUF that PUF A and PUF B are matched to. Note that the template PUF has no physical entity; it is purely a conceptual function. To facilitate the demonstration, we denote the **delay difference of each segment** in PUF A, PUF B, and the template PUF in Equation 4. There are multiple ways to define the delay properties of the template PUF, we have defined it as shown in Equation 5 to reduce the number of segments need to be built using PDL.

$$\begin{aligned} PUF\ A &= \{d_A^1, d_A^2, ...d_A^n\} \\ PUF\ B &= \{d_B^1, d_B^2, ...d_B^n\} \\ PUF\ Template &= \{d_T^1, d_T^2, ...d_T^n\} \end{aligned} \tag{4}$$

$$d_T^i = max(d_A^i,\ d_B^i),\ i \in \{1, 2, ...n\} \tag{5}$$

C. Appending PDL Segments

This goal of this step is to modify PUF A and PUF B to the template PUF using PDL. Following the characterization of the template PUF in Equation 5, two situations may happen during the matching process. Take the matching of PUF A as an example:

(1)$d_A^i < d_T^i$. An extra segment with delay difference $|d_T^i - d_A^i|$ needs to be added to PUF A. This segment together with the ith segment in PUF A equivalent to the ith segment in the template PUF.

(2)$d_A^i = d_T^i$. In this situation, nothing needs to done to modify the ith segment in PUF A.

We build the extra delay segments using PDL. In each segment, one LUT with selection bit combination C_{upper} is used as the upper delay and the other LUT with selection bit combination C_{lower} is used as the lower delay. Due to the delay bias caused by the asymmetric internal structure of LUTs, different input combinations to LUTs will generate distinct delays. Therefore, by properly assigning C_{upper} and C_{lower}, target delay difference ($|d_T^i - d_A^i|$) can be created. In some cases, $|d_T^i - d_A^i|$ is larger than the maximum delay difference that can be generated using C_{upper} and C_{lower}, then multiple LUTs can be applied as the upper delay/lower delay to boost the delay difference proportionally.

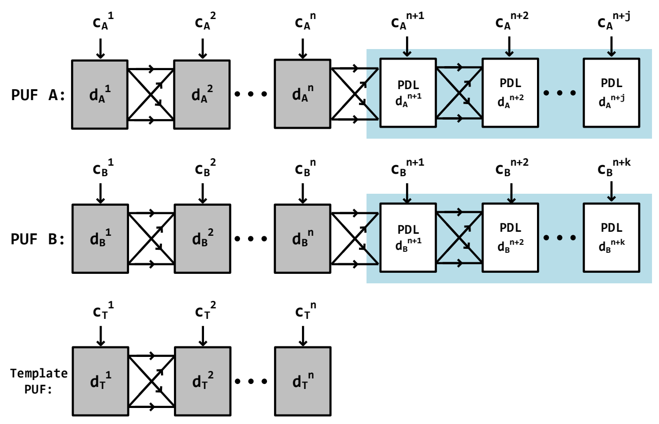

Fig. 5: Structures of PUF A and PUF B after matching using PDL.

D. Challenge Reassignment

After the previous steps, both PUF A and PUF B now are matched to the template PUF, thus are expected to have the same challenge-response mapping function. The structures of matched PUF A and PUF B as shown in Figure 5 have more than n segments. Assume that PUF A has j additional segments and PUF B has k additional segments implemented using PDL. Then $k + j$ should equal to n according to our template defined above. Now consider a random n-bit challenge C_T is fed to the template PUF to generate a response R. The key question is how to adjust the $(n+j)$-bit challenge C_A to be fed to PUF A and the $(n+k)$-bit challenge C_B to be fed to PUF B to generate the same response R.

Our challenge reassignment algorithm works in the following way. We take PUF A as an example. We start with the rightmost segmental delay difference d_A^{n+j}, and check the corresponding segmental delay difference d_T^i that d_A^{n+j} is matched to in the template PUF. The segmental delay difference d_T^i, which equals to d_B^i in PUF B should be the sum of d_A^{n+j} and d_A^i based on the matching policy. Then according to the challenge C_T, we check whether the segmental delay difference d_T^i contributes to the upper path or the lower path before the arbiter. Based on the observed path P in the template PUF, we assign the $(n+j)th$ challenge bit C_A^{n+j} in C_A in such a way that d_A^{n+j} also contributes to the same path P in PUF A. The above process is repeated with the $(n+j-1)$th segment, and continues all the way to the first segment in PUF A. Consequently, each time when we assign a challenge bit to PUF A, all the challenge bits after have been assigned already, making it deterministic which value to assign to that challenge bit in order to be consistent with the corresponding segment in the template PUF. With the above process, for any random challenge to the template PUF, we can find a matched challenge to PUF A and another challenge to PUF B to make them all generate the same response.

E. Discussion

The above PUF matching flow can be easily extended to the multi-party PUF matching. As long as all the parties are synchronized with the template PUF, they can always add new PDL segments and follow the same algorithm to reassign challenges for matching. This provides the potential for message encryption/decryption between multiple parties.

The above matching process maintains the properties of standard PUFs. On one hand, the extra area and delay overhead are reasonably small. In the case of matching two PUFs, the newly generated PUFs have more segments comparing to the original PUFs, but the total number of additional segments merely equals to n, which is the length of an original PUF. On the other hand, the generated matched PUFs remain unclonable. Although an attacker can reproduce the PDL segments by applying the same selection bits to the LUTs, the attacker is unable to duplicate the segments of the original PUF because of process variation. This explains the reason that we can not directly use PDL to build new PUFs, otherwise the unclonability of the function will be compromised.

VI. Security Protocols

Various security protocols are enabled using our PUF matching platform. We choose three typical protocols to demonstrate in this section, multi-party message communication, pairwise encryption/decryption, and authentication.

A. Multi-party Message Communication

The protocol of multi-party message communication assumes that N trusted parties want to exchange information mutually ($N \geq 2$). The message exchange should be acted in such a way that when a message is sent, all the other $N-1$ parties are able to retrieve the message, but untrusted parties are incapable of doing so. Protocol 1 shows the flow of the multi-party message communication. In the protocol, we assume that party i wants to transfer message m to all the other trusted parties. The protocol is designed based on the fact that after the PUF matching, each and only each trusted party owns a matched PUF with the same mapping function E. The whole process employs low timing overhead in the sense that it only takes one time PUF calculation (one clock cycle) for all the parties.

Protocol 1 Multi-party Message Communication

```
1: Party $1$ to party $N$ match their PUFs.
2: Party $i$ generates a random challenge $c$.
3: Party $i$ calculates $R = E(c) \oplus m$, where $E$ is the function
   of the matched PUF, and $m$ is the message to transfer.
4: Party $i$ broadcasts $R$ and $c$.
5: All the other parties with the match PUF calculate $m =$
   $E(c) \oplus R$.
```

B. Pairwise Encryption and Decryption

An alternative form of multi-party communication is to only allow pairwise communication which means that when party A encrypts a message and sends to party B, only party B can decrypt the message and all the rest parties can not. Our PUF matching platform can also be applied in this scenario. Traditionally, if party A needs to talk to N parties exclusively using PUFs, he/she needs N separate PUFs. However, using our PUF matching, only a single PUF is required for party A since PDL can be easily reconfigured to match to different parties. Therefore, by simply adding reconfigurable PDL, the required number of PUFs used for communication is reduced exponentially. The detailed encryption and decryption scheme is the same as the multi-party message communication with the only modification that the PUF matching in this protocol is merely between two parties.

C. Authentication

Authentication, as another commonly used protocol, can also be easily realized with the matched PUF platform. The key idea is to use matched PUFs to calculate the response of a random challenge and expect only the authorized parties to provide the correct responses. The whole process is also low overhead since each trusted party only needs one clock cycle computation.

Protocol 2 Authentication

```
1: Party $1$ to party $N$ match their PUFs.
2: Party $i$ generates a random challenge $c$.
3: Party $i$ calculates $R = E(c)$ and broadcasts $c$.
4: All the other parties with the match PUF calculate $R' =$
   $E(c)$ and send $R'$ to party $i$.
5: Party $i$ compares $R$ to $R'$. If and only if $R = R'$, party $i$
   authenticates the party.
```

VII. Implementation

We demonstrate our implementation and evaluation of PDL as well as PDL-based PUF matching mechanism in this section. All implementations and measurements are done on Spartan-6 XC6SLX45 FPGAs.

A. Delay Measurement Setup

In order to measure and verify the delay of PDL on the FPGA we use the circuit describe by Majzoobi et al [12]. The delay characterization circuit is shown in Figure 6. The entire measurement system is constructed by three parts: *circuit under test (CUT)*, flip-flops and external modules.

1) Circuit Under Test: Our goal is to measure the delay difference of an LUT-based PDL when providing different input vectors. However, such delay is too small to measure individually, thus a chain of LUTs is used as our target *CUT*. We use on-board LUT6 to build our *CUT* and each LUT in the chain implements an inverter. To be specific, only the most significant selection bit fed to the LUT is inverted and the remaining 5 bits are used as configuration bits to configure the

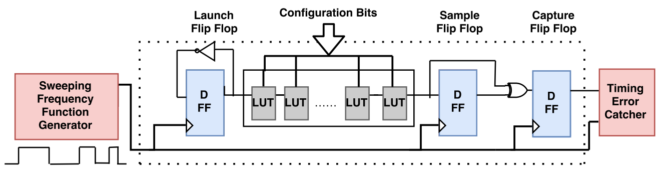

Fig. 6: Delay characterization circuit.

internal signal routing path inside the LUT. The configuration bits are identical for every LUT in the *CUT* so that we are able to estimate individual delay by calculating *CUT* delay over the number of LUTs. We measure the delays of *CUT* that consists of 10 LUTs when providing all possible configuration bits from “00000” to “11111”.

2) Flip-flops: There are three D flip-flops used in the delay characterization circuit: launch flip-flop, sample flip-flop and capture flip-flop. All three flip-flops share the same clock signal generated by a sweeping frequency function generator. The launch flip-flop generates a low-to-high signal through the *CUT* at the rising edge of the clock. Then the signal arrives at the sample flip-flop. If the signal arrives before the sampling action takes place (at the rising edge of the clock) then the correct signal value is successfully sampled, otherwise the sampled value would be different indicating a timing error. Such a mismatch can be detected by a simple XOR gate. If the sampled value and the correct signal value are the same, the XOR gate would produce a 0 indicating no timing violation occurred. Otherwise, the capture flip-flop will receive a 1 from the XOR gate indicating a timing error.

3) External Modules: There are two external modules used in the delay measurement process. A sweeping frequency function generator is used to generate a square wave from 2MHz to 100MHz. We then shift the frequency up by 32 times using the phased lock loop (PLL) on the FPGA. For each frequency, the generator produces a fixed number of 10,000 pulses which are used to drive the flip-flops. A timing error catcher module takes the output of the capture flip-flop and the clock signal as inputs and calculates the probability of a timing error occurrence. By monitoring the timing error probability, the *CUT* delay can be inferred at pico-second resolution.

B. Delay Measurement Results

We have measured the average delay for our *CUT* under $25\,^{\circ}\mathrm{C}$ operating temperature and then calculate the individual LUT delay. The results are shown in Figure 7. Figure 7(a) shows the delay difference between any pair of configuration bits. Figure 7(b) shows absolute value of delay difference between any pair of configuration bits and Figure 7(c) demonstrates hamming distance heatmap between each pair.

The largest difference is 11 ps, which occurs between 00000 and 11111. This case is found at location (x,y) = (0,31) and location (31,0) in Figure 7(a) and 7(b). The diagonal line from lower left corner to upper right corner is all 0s because we are comparing each configuration bits to themselves.

When we compare Figure 7(b) with Figure 7(c), we notice that some patterns shown in Figure 7(b) can be observed in Figure 7(c). In many cases, if two configuration vectors have large delay difference in PDL, these two vectors also have large hamming distance. We believe this is a valid observation because large hamming distance indicates that the corresponding internal signal paths share very few common routes, consequently it is more likely to generate higher delay difference. However, note that sharing few common routes does not always indicate large delay difference. Two very distinct signal paths might as well produce small delay difference. Thus, we also see many patterns in hamming difference heatmap are not observable in Figure 7(b).

C. Process Variation

Process variation is not avoidable when we measure delays at pico-second resolution. We have run experiments on 3 difference FPGA boards as well as different locations on each board to test the effect of process variation. We have measured the delays of PDL on 5 different locations on each board when providing 00000 and 11111 as configuration bits. The average delays of PDL on three boards are compared and presented in Table I. The results indicate that the delay difference is relatively stable when providing the same pair of configuration bits on different boards.

	00000 (ns)	11111 (ns)	Difference (ns)
FPGA 1	1.253	1.265	0.012
FPGA 2	1.248	1.259	0.011
FPGA 3	1.257	1.267	0.010

TABLE I: Delay measurement results on three different FPGAs.

Based on our measurement, we believe it is safe to assume that the time difference between different routes within a PDL is larger than the difference caused by the effect of process variation.

VIII. Results

We implement our PDL-based PUF matching platform on Spartan-6 XC6SLX45 FPGAs. We first test the matching accuracy of our matching scheme. We then examine the system overhead to prove that our design is lightweight.

A. Matching Accuracy

When applying the same challenge vector to a pair of matched PUFs, the probability that the two PUFs generate the same response is defined as matching accuracy.

We followed our proposed matching scheme to implement and match two 64-bit PUFs. We have generated 1,000,000 random challenges and applied them to the two PUFs. The test is repeated 10 times and the average matching accuracy reaches 98.64%. We believe the high matching accuracy proves that our design is not only efficient but also stable.

B. System Overhead

We measure the delay, area and energy consumption for PDL-based PUF matching platform and show the results in Table II. Note that we have measured the average overhead of

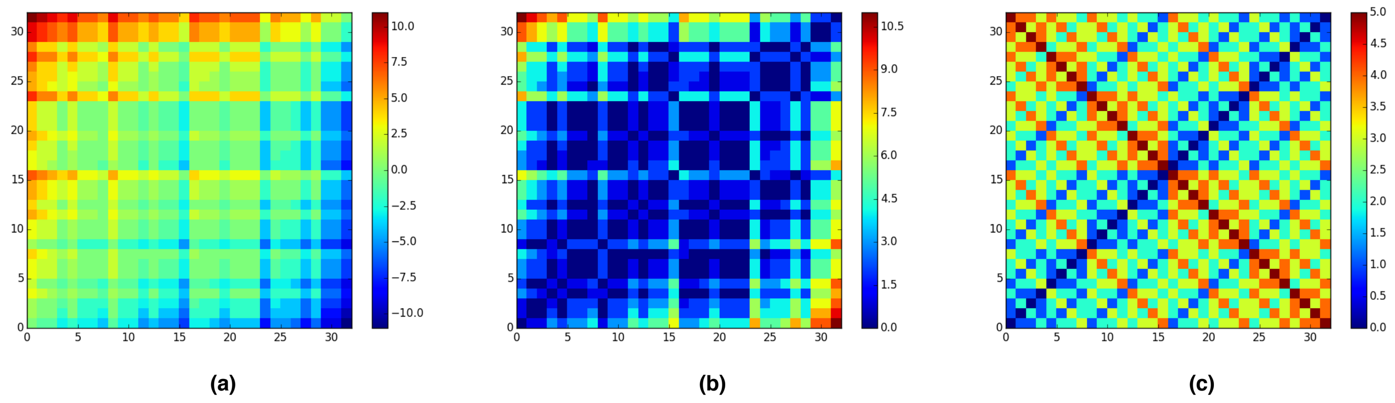

Fig. 7: (a) Delay difference between any pair of configuration bits. Delay difference unit: (ps). (b) Absolute value of delay difference between any pair of configuration bits. Delay difference unit: (ps). (c)Hamming distance between all pairs of configuration bits.

a single party in our matching scheme. We further compare our design with several popular low energy block ciphers as shown in Table III. The comparison indicates that our design is comparable regarding the size of the design while is the most competitive in terms of energy consumption (due to small delay overhead).

Type	Overhead
LUTs	196
Slices	145
Max Delay (ns)	116.712
Energy (μJ)	9.54×10^{-4}

TABLE II: Overhead of PDL-based matched PUF.

Type	Energy(μJ)	LUTs
TinyXTEA-3 [16]	5.45×10^{-3}	364
Present [16]	3.16×10^{-3}	159
HIGHTs [16]	1.07×10^{-3}	132
Matched PUF using PDL	9.54×10^{-4}	196

TABLE III: Energy consumption comparison.

C. Security Test

As a security primitive, the output randomness is an important criteria to evaluate the security property. We quantify the statistical randomness of the matched PUF outputs by applying the industry-standard statistical test suite of the National Institute of Standards and Technology (NIST) [23]. We generate a stream of outputs in the following way: a random seed is used as the primary inputs to the matched PUF after configuration and the corresponding outputs are generated. In each subsequent clock cycle, the outputs are first shuffled and then XORed with the previous inputs to generate the inputs for the next clock cycle. We repeat the process until we have collected enough outputs required by the benchmark suite. For each test, we use 1000 instances of matched PUFs, the results in Table IV show the average passing ratio of each subtest over all the instances.

Statistical Test	Success Ratio
Frequency	100%
Block Frequency (m=128)	97.6%
Cusum-Forward	98.1%
Cusum-Reverse	98.3%
Runs	98.5%
Longest Runs of Ones	96.5%
Rank	98.2%
Spectral DFT	95.6%
Non-overlapping Templates ($m = 9$)	95.9%
Universal	100%
Approximate Entropy ($m = 8$)	96.5%
Rand. Excursions ($x = 1$)	98.2%
Rand. Excursions Variant ($x = -1$)	97.6%
Serial ($m = 16$)	98.7%
Linear Complexity ($M = 500$)	97.8%

TABLE IV: The average success ratio for the NIST statistical test suite. 1000 bitstreams of 10000 bits are passed to each test. The test passes for p-value$\geq \sigma$, where σ is 0.01.

IX. Conclusion

In this paper, we have proposed an ultra-low energy PUF matching scheme by using PDL. Our core idea is to modify the delay difference of arbiter PUF segments in such a way that multiple PUFs have the same challenge-response mapping function. On the top of our PUF matching platform, a variety of low overhead security protocols between multiple parties are enabled. Furthermore, we have implemented our design on the Spartan-6 FPGA platform. The experiment results indicate that our design allows the PUFs to be matched with high accuracy while requiring ultra low overhead.

X. Acknowledgment

This work was supported in part by the NSF under Award CNS-0958369, and Award CNS-1059435.

References

[1] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM journal of research and development*, vol. 50, no. 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,” in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 148–160, ACM, 2002.

[4] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *Proceedings of the 44th annual Design Automation Conference*, pp. 9–14, ACM, 2007.

[5] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA intrinsic PUFs and their use for IP protection*. Springer, 2007.

[6] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, “The butterfly PUF protecting IP on every FPGA,” in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pp. 67–70, IEEE, 2008.

[7] T. Xu and M. Potkonjak, “Digital puf using intentional faults,” in *Quality Electronic Design (ISQED), 2015 16th International Symposium on*, pp. 448–451, IEEE, 2015.

[8] T. Xu, J. B. Wendt, and M. Potkonjak, “Secure remote sensing and communication using digital pufs,” in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pp. 173–184, ACM, 2014.

[9] L. Bolotnyy and G. Robins, “Physically unclonable function-based security and privacy in RFID systems,” in *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pp. 211–220, IEEE, 2007.

[10] U. Rührmair, “Simpl systems: On a public key variant of physical unclonable functions.,” *IACR Cryptology ePrint Archive*, vol. 2009, p. 255, 2009.

[11] T. Hui and R. W. Mounger, “Programmable delay line,” Aug. 3 1999. US Patent 5,933,039.

[12] M. Majzoobi, F. Koushanfar, and S. Devadas, “FPGA PUF using programmable delay lines,” in *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pp. 1–6, IEEE, 2010.

[13] A. Raychowdhury, S. Ghosh, and K. Roy, “A novel on-chip delay measurement hardware for efficient speed-binning,” in *On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International*, pp. 287–292, IEEE, 2005.

[14] M.-C. Tsai, C.-H. Cheng, and C.-M. Yang, “An all-digital high-precision built-in delay time measurement circuit,” in *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, pp. 249–254, IEEE, 2008.

[15] M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar, “Rapid FPGA characterization using clock synthesis and signal sparsity,” in *International Test Conference (ITC)*, pp. 1–10, 2010.

[16] P. Yalla and J.-P. Kaps, “Lightweight cryptography for FPGAs,” in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*, pp. 225–230, IEEE, 2009.

[17] M. Majzoobi, F. Koushanfar, and M. Potkonjak, “Lightweight secure pufs,” in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 670–673, IEEE Press, 2008.

[18] S. Meguerdichian and M. Potkonjak, “Matched public PUF: ultra low energy security platform,” in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pp. 45–50, IEEE Press, 2011.

[19] T. Xu and M. Potkonjak, “The digital bidirectional function as a hardware security primitive: Architecture and applications,” in *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pp. 335–340, IEEE, 2015.

[20] T. Xu, H. Gu, and M. Potkonjak, “Data protection using recursive inverse function,” in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pp. 1–4, IEEE, 2015.

[21] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, “Modeling attacks on physical unclonable functions,” in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 237–249, ACM, 2010.

[22] T. Xu, D. Li, and M. Potkonjak, “Adaptive characterization and emulation of delay-based physical unclonable functions using statistical models,” in *Proceedings of the 52nd Annual Design Automation Conference*, pp. 76–81, ACM, 2015.

[23] J. Soto, “Statistical testing of random number generators,” in *Proceedings of the 22nd National Information Systems Security Conference*, vol. 10, p. 12, NIST Gaithersburg, MD, 1999.