University of California

Los Angeles

Digital Physical Unclonable Functions: Architecture and Applications

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Computer Science

by

Teng Xu

2014

Abstract of the Thesis

Digital Physical Unclonable Functions: Architecture and Applications

by

Teng Xu

Master of Science in Computer Science

University of California, Los Angeles, 2014

Professor Miodrag Potkonjak, Chair

The rapid growth of small form, mobile, and remote sensor network systems require secure and ultralow power data collection and communication solutions due to their energy constraints. The physical unclonable functions (PUFs) have emerged as a popular modern security primitive. They have the property of low power/energy, small area, and high speed. Moreover, they have excellent security properties and are resilient against physical and side-channel attacks. However, traditional PUFs have two major problems. The first is that current designs are analog in nature and lack susceptibility in environmental and operational conditions, e.g., supply voltage and temperature. The second is that due to the analog nature, the analog PUFs are difficult to be integrated into existing digital circuitry.

Therefore, in this thesis, we propose the digital PUF, as a new type of security primitive. It preserves all the good properties of traditional analogy PUFs and is stable in the same sense that digital logic is stable. It has a small footprint, a small timing overhead, a low energy consumption, and can be easily integrated into existing designs. The key observation is that for any analog delay PUF, there is a subset of challenge inputs for which the PUF output is stable regardless of operation and environmental conditions. We use only such stable inputs to initialize the look-up tables (LUTs) in digital bimodal functions (DBFs) that are configured in such a way that the digital PUF is formed. We first demonstrate the concept and the FPGA-based architecture of the digital PUF. Then we present our security analysis on digital PUFs using standard randomness tests and confusion and diffusion analysis. Finally, we address security protocols of digital PUF: public key communication, and remote trust.

The thesis of Teng Xu is approved.

Mario Gerla

Milos D. Ercegovac

Miodrag Potkonjak, Committee Chair

University of California, Los Angeles

2014

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The rapid proliferation of mobile systems and devices that operate in potentially hostile environments has elevated security to one of the most important design metrics. For example, security is essential in smart phones, laptops, and wireless sensor networks. Classical software-based public-key cryptography provides a spectrum of elegant and powerful security protocols. However, it is also subject to several important limitations and drawbacks including susceptibility to physical and side channel attacks and high implementation and energy costs.

The physical unclonable function (PUF) is a cryptographic primitive that has been suggested for sensor network security due to its low power requirements. PUFs are physical devices that have a random but deterministic mapping of inputs to outputs. Their unclonability—and functionality—are often inextricably tied to the physical characteristics of the device components (e.g. gate delay, leakage energy). While PUFs receive and generate digital inputs and outputs, they are analog in nature due to their reliance and design based on their inherent physical characteristics. Thus, current PUFs have many limitations. The most limiting of which includes stability and susceptibility to environmental and operational conditions. Many PUFs, including the standard delay-based PUF require arbiters to operate. These memory components limits the PUF in terms of placement and coordination in circuitry since their outputs cannot be used directly in the current cycle like a combinational module, but require an additional clock cycle to be used.

These main limitations can be removed by creating a purely digital PUF. The digital PUF must be stable in the same sense that digital logic is stable against environmental and operational conditions and must produce deterministic outputs for all input vectors. The digital PUF must integrate with existing combinational logic without requiring additional clock cycles to use its outputs. And lastly, the digital PUF must be flexible in the sense that its structure can be altered for different tradeoffs between security, energy, and delay as required by the pertinent task.

In this thesis, we present a digital PUF design with such characteristics. Its underlying architecture consists of a series of lookup tables (LUTs) which are initialized using standard delay-based PUFs. The standard PUFs enable both unclonability and configurability in our design. Despite the inherent instabilities known to exist in them, we ensure stability through two means: (a) through a slight modification in the standard delay-based PUF design that enables stable output validation and input selection, and (b) through a reduction in use to only circuit initialization, thus tremendously reducing the impact of device aging on its gate delays.

We analyze the security of the digital PUF as it stands alone by applying the NIST randomness benchmark test suite [1] and demonstrating that it passes all tests. We also analyze the outputs of our digital PUF using the security principles of confusion and diffusion, as presented by Shannon [2], through demonstration of the avalanche criterion.

However, despite the theoretically sound and mathematically proven security properties of many digital cryptographic systems, there exist many potential side-channels which can effectively bypass these mathematical constructs altogether by reading internal memory or inferring internal procedures through power analysis and memory attacks. Since our digital PUF utilizes memory cells, such as arbiters, SRAM, and flip-flops in its LUTS, it is potentially susceptible to side-channel attacks [3]. We demonstrate that these attacks can be prevented through analysis of modern feature sizes, the use of 3D integrated circuitry, and the use of inspection resistant memory [4].

Lastly, we explore two important security protocols. Our first protocol is public key communication with DBF. It utilizes the unique property of DBF to enable low-energy, high-speed, small-area public key communication. The second protocol is remote trust, which utilize digital PUF to enable authentication between parties remotely.

CHAPTER 2

Related Work

2.1 Physical Unclonable Functions

Pappu et al. introduced the concept of the first PUF and demonstrated it using mesoscopic optical systems [5]. Devadas' research group at MIT developed the first family of silicon PUFs through the use of intrinsic process variation in deep submicron integrated circuits [6]. Guarardo and his coworkers at Philips Research in Eindhoven demonstrated how PUFs can create unique startup values in SRAM cells [7]. Consequently a great variety of technologies were used for PUF creation including IC interconnect networks, thyristors, memristors, and several nanotechnologies. Although a variety of PUF structures have been proposed, arbiter-based (APUF) [6], ring oscillator-based (RO-PUF) [8], and SRAM PUFs [7] are by far most popular.

PUFs were immediately applied to a number of applications including authentication, cryptographic key generation and secure storage [9], anti-counterfeiting [10], FPGA intellectual property (IP) protection [11], remote enabling and disabling of integrated circuits [12], and remote trusted sensing [13] [14]. PUFs are also used in conjunction with traditional creation and operation of remote secure processors [15]. The security role of the PUF has been greatly enhanced with several proposals for employing PUFs in public key security protocols in systems such as the public PUF (PPUF), SIMPL, and one time pads [16] [17]. Recently, the matched PPUF (mPPUF) [18] has been proposed as a new public key security primitives. mPPUF uses both process variation and device aging to create pairs of identical PPUFs that can be matched only with negligible small probability. It is a very energy efficient security primitive that can be used in a variety of cryptographic protocols. However, mPPUF poses high implementation requirements in terms of measurement accuracy and environmental stability [19].

There have been two efforts that aim to remove the limitations of analog PUFs. The first is the digital bimodal function (DBF) [20]. The DBF easily passes several security and randomness tests, but is not unclonable and cannot be integrated with regular digital logic without significant time overhead. In the second effort, Fyrbiak et al. proposed the creation of software security primitives using hardware random generators [21]. Hardware-software security primitives require relatively long execution times and depend on unspecified reproducible random generators.

2.2 Attacks on PUFs

A large number of security attacks on essentially all types of PUFs have been explored. They can be classified into two groups: reverse engineering (also called characterization) and manufacturing or emulation attacks. Non-invasive characterization attacks mainly target the delay-based PUFs (e.g. APUF and RO-PUF). These attacks mainly use numerical algebra and machine learning techniques. For example, Majzoobi et al. demonstrated how linear programming can be used to characterize delay PUFs [22]. By far the most popular statistical attack was reported by Rührmair at al. in which a relatively small number of challenge-response pairs yielded highly accurate prediction models [23]. Most recently, Xu and Burleson proposed coordinated side-channel and machine learning attacks [24].

There are a number of well studied side-channel attacks either on cryptographical protocols and devices or directly on PUFs including timing, power, electromagnetic emanation, optical, and variety of memory reading attacks including the use of focused ion beams [25] [26]. Note that attacks such as cache behavior attacks are not applicable to PUFs. For instance, it has been practically demonstrated that several side-channel attacks can read data stored in DRAM and SRAM cells [27]. For example, the security research group at Technische Universität Berlin reported successful physical cloning of SRAM PUFs [3].

Side-channel attacks use a variety of physical phenomena and sophisticated engineering approaches, often with high effectiveness. Still, there is a strong belief that APUFs and other delay-based PUFs are either safe or at least much more resilient against side-channel attacks due to their small difference in physical signals and dependency on difficult-to-measure threshold voltages that depend on the number of dopants and their distribution in transistor channels along with other physical characteristics of the device.

CHAPTER 3

Delay-based PUF Stability

3.1 Stable Challenges and Outputs

Figure 3.1 depicts an example of a 3-bit delay-based PUF. Each challenge bit controls the inputs of two multiplexers. An output bit is generated by assigning a challenge vector and sending a rising edge through the PUF. The two paths traverse the three delay segments, swapping positions (top and bottom) depending on the input bit at each segment, before arriving at the arbiter which determines the final output. For example, an input challenge of 011 generates the blue and red paths depicted. An arbiter will set its value to 0 or 1 depending on which path (top or bottom) arrives first, effectively selecting the path that has the smaller delay. Table 3.1 consists of the delay differences between the top and bottom paths for all possible paths in the example PUF in Figure 3.1.

A key observation is that for each unique delay-based PUF there exists a

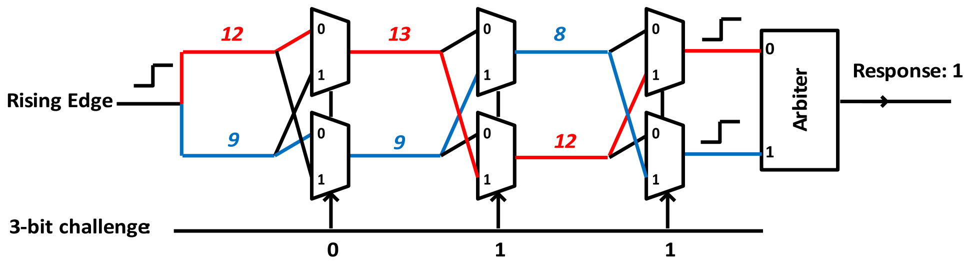

Figure 3.1: Applying a 3-bit input challenge to a delay-based PUF. The challenge is intentionally chosen in this example in such a way that the delay difference between the two paths (red and blue) are maximized.

Challenge	Delay Difference
000	3
001	-3
010	-11
011	11
100	-3
101	3
110	-5
111	5

Table 3.1: Delay differences between all possible paths in the example delay-based PUF in Figure 3.1.

set of challenges that produce stable outputs. Consider the situation in which environmental conditions affect the physical characteristics of the circuit. For example, variations in temperature cause variations in individual gate delays, thereby affecting the overall path delays in the analog PUF. Since challenges 011 and 010 result in a large difference in delay between the two racing paths, it is still with high possibility that the red path will have a larger delay compared to the blue path despite the effects temperature may have on the individual gate delays. We label this challenge, and any other challenges that are resilient to such environmental changes, as stable inputs.

For path delay analysis we introduce a *delay ratio* metric, which is defined as the delay differences of two paths divided by the delay of the shorter path. For the purposes of testing, we assume that gate delays follow a normal distribution due to the effects of process variation.

$$Delay\ Ratio = \frac{Delay_{p1} - Delay_{p2}}{min(Delay_{p1}, Delay_{p2})} \tag{3.1}$$

	$P(R \geq 0.04)$	$P(R \geq 0.06)$	$P(R \geq 0.08)$	$P(R \geq 0.1)$
32-bit PUF	12.51%	4.27%	1.07%	0.21%
64-bit PUF	9.34%	2.44%	0.43%	0.05%

Table 3.2: Probability that the delay ratio (R) is larger than the labelled threshold value for a 32-bit and 64-bit PUF.

We use the the delay ratio, as defined in Equation 3.1, to evaluate the relative delay difference between the two PUF paths. In the following test we assume that the gate delays of the PUF follow a normal distribution due to the effects of process variation.

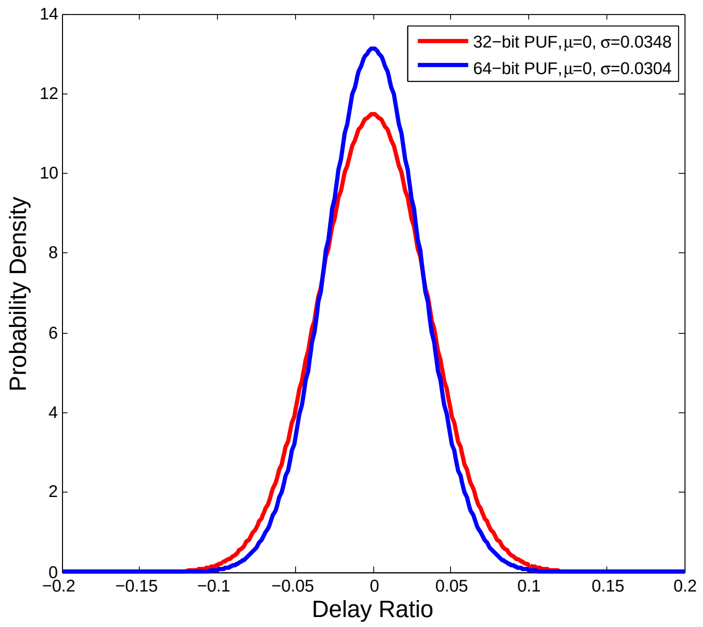

Figure 3.2: Distributions of delay ratios for a 32-bit PUF and a 64-bit PUF.

The distribution of the delay ratio for random challenges on a 32-bit PUF and a 64-bit PUF are depicted in Figure 3.2. In both cases, they follow a normal distribution with their means at 0. The standard deviation of the 64-bit PUF is smaller than the 32-bit PUF. To better visualize the probability that the delay ratio is larger than some value, we use Table 3.2 to show the quantified result.

Temperature	Delay Ratio (T=300K)						
	0.04	0.05	0.06	0.07	0.08	0.09	0.1
250K	0.979	0.987	0.994	0.997	1	1	1
350K	0.969	0.975	0.989	0.994	0.997	1	1
400K	0.937	0.951	0.959	0.977	0.988	0.996	1

Table 3.3: Probability that outputs of the 32-bit PUF are stable over varying temperatures for different delay ratios.

Temperature	Delay Ratio (T=300K)						
	0.04	0.05	0.06	0.07	0.08	0.09	0.1
250K	0.984	0.986	0.996	0.998	1	1	1
350K	0.982	0.986	0.993	0.998	1	1	1
400K	0.954	0.974	0.986	0.991	0.997	1	1

Table 3.4: Probability that outputs of the 64-bit PUF are stable over varying temperatures for different delay ratios.

We simulate and test the stability of our PUF under different environmental temperatures. In our test, we assume that the original distribution of the gate delay D_{old} at temperature 300K follows the Gaussian distribution $D_{old} \sim \mathcal{N}(\mu, \sigma^2)$. When temperature changes, the delay changes: $D_{change} \sim \mathcal{N}(\alpha\mu, |\alpha|\sigma^2)$ where α is a ratio which is decided by the new temperature, thus yielding the new delay $D_{new} = D_{old} + D_{change}$. We use the Hotspot tool [28] to compute α under different temperatures. For example, α under 400K is approximately 1. Based on this assumption, under different original delay ratios and after applying temperature changes, we measure the probability that the same challenge produces the same stable outputs. Table 3.3 shows the results of a 32-bit delay-based PUF. As expected, a higher original delay ratio yields higher probabilities for stable outputs. For example, for an original delay ratio of 0.1, the probability that the PUF output remains stable across temperatures ranging from 250K to 400K remains 1.

The results of our 64-bit PUF tests are shown in Table 3.4. Compared to the 32-bit test case, the 64-bit test case demonstrates a similar trend and exhibits even better stability under the same conditions. As long as the original delay ratio reaches a particular threshold (e.g. 0.1 in this experiment), the outputs remain stable for a wide range of temperatures. Hence, we select those challenges that satisfy this delay ratio threshold as the stable challenges.

3.2 Achieve Stable Challenges

One important issue is how to obtain these stable challenges. We have proposed two easy but feasible solutions. The first is to use programmable delay line and the second is to use statistical information of delay profile. Before describing the approaches, we assume that a n-bit delay-based PUF has n stages in total. Each stage is in charge of one bit and it has two delays racing through each other.

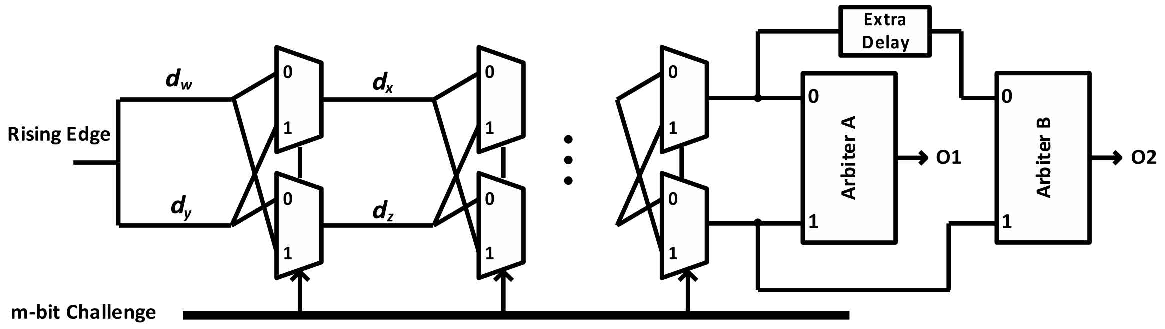

Figure 3.3: Architecture for stable challenge-response testing.

3.2.1 Programmable Delay Line

The programmable delay line allows the user to put adjustable additional delay to the PUF paths. In this approach, we take advantage of the programmable delay line as shown in Figure 3.3. Before the two paths reach the arbiter, we intentionally put certain extra delay to one of the paths, then we apply random challenges to the PUF. For example, for a k-bit delay-based PUF, if the expectation of the delay for each stage is D, then the expectation of the total delay for a path is $k*D$. Now we put $0.1*k*D$ extra delay to one of the paths, then apply random challenges. Suppose under such circumstance, the path being added extra delay can still reach the arbiter earlier than the other path, then we claim that the current challenge enables one path approximately 0.1 times faster than another path if the extra delay is not added, hence, a stable challenge. The detailed algorithm is described in Algorithm 1. According to Table 3.2, although the portion of stable challenges is small, but due to the fact that each trial for a random challenge only requires one clock cycle, the time required to get a large number of stable challenges is still negligible.

3.2.2 Delay Profile

This approach takes advantage of the statistical information of delay profile. This approach involves two steps, the fist is to acquire the delay profile information

Algorithm 1 Programmable Delay Line Approach

Input: D - additional delay by programmable delay line.

Input: C_i - random challenges.

Input: $path0$ - path that outputs 0 in arbiter.

Input: $path1$ - path that outputs 1 in arbiter.

```
1: Add additional delay $D$ to $path0$ ($path1$)
2: for random challenge $C_i$ do
3:   if $path0$ ($path1$) is shorter than $path1$ ($path0$) then
4:     $C_i$ is stable
5:   end if
6: end for
7: Output:  all the acquired stable challenges.
```

and the second is to use the profile information to get stable challenges.

We first assume that in the delay-based PUF, $path0$ and $path1$ represent the two paths in the PUF. $path0$ is the path that outputs 0 in the arbiter while $path1$ is the opposite. For stage i, 2 delays d_i and $\hat{d_i}$ are racing with each other while the challenge bit in this stage determines in which path each delay is included. For example, when the challenge bit is 1, d_i is contained in $path0$ and $\hat{d_i}$ is contained in $path1$ and vice verse. Therefore, with this as a starting point, we apply random challenges to the delay-based PUF and build statistical model in this process. To be more specific, we calculate the possibility that the path that includes d_i is shorter and the possibility that the path that includes $\hat{d_i}$ is shorter. Suppose that d_i is smaller than $\hat{d_i}$, then after a large enough testing set of random challenges, the path that contains d_i will be shorter than the path that contains $\hat{d_i}$ with higher likelihood than the vice verse. The specific probability is determined by the delay difference between d_i and $\hat{d_i}$. For instance, after 1000 trials of random challenges, the paths with d_i are shorter with 550 times and the paths with $\hat{d_i}$ are shorter with 450 times. From there, we can conclude that d_i is more likely to be shorter than d_i. Therefore, after this step, the relation between the two delays in each stage can be determined.

The second step is to produce the stable challenges based on the achieved delay profile. The way we do it is to first assign some random bits as part of the challenge and based on the already filled bits, then we adjust the rest bits to achieve the largest delay difference between the two paths. For example, in Figure 3.1, we assume that the 4-bit challenge from left to right is b_1 to b_4. Then we randomly fix 1 bit, e.g., $b_1 = 1$ or $b_1 = 0$ and adjust the rest 3 bits b_2, b_3, and b_4. We adjust the bits from the stages that are closer to the arbiter to the stages that are away from the arbiter so that as the adjustment goes on, we know exactly in each stage, each of the 2 delays contributes to which output bit in the arbiter. For instance, in Figure 3.1, we start the adjustment from b_4 and continues towards b_2. Due to the fact that we already have the delay profile information, we know which delay is larger in each stage. Therefore, we can assign $b_2b_3b_4 = 010$ so that the maximum delay difference is achieved. However, the solution for adjustment is not unique, e.g., when $b_2b_3b_4 = 011$, the challenge is also stable. The detailed algorithm is described in Algorithm 2. Note that when more bits are fixed in the beginning of the algorithm, the space for adjustment is diminished, so that the stability of the challenge is decreased, but correspondingly, the diversity of the acquired stable challenges increases.

Assume that we have m random bits in the beginning for a 64-bit delay-based PUF, and we can adjust n bits later where $m + n = 64$. The two delays for stage i is d_i and $\hat{d_i}$ where both delays are random values following the same Gaussian distribution, then the delay difference between the two paths can be expressed as illustrated in Equation 3.2. In Figure 3.4, for each hamming distance, we plot the distribution of the delay difference between the two paths. We can see that as the hamming distance gets smaller which means m is smaller in the

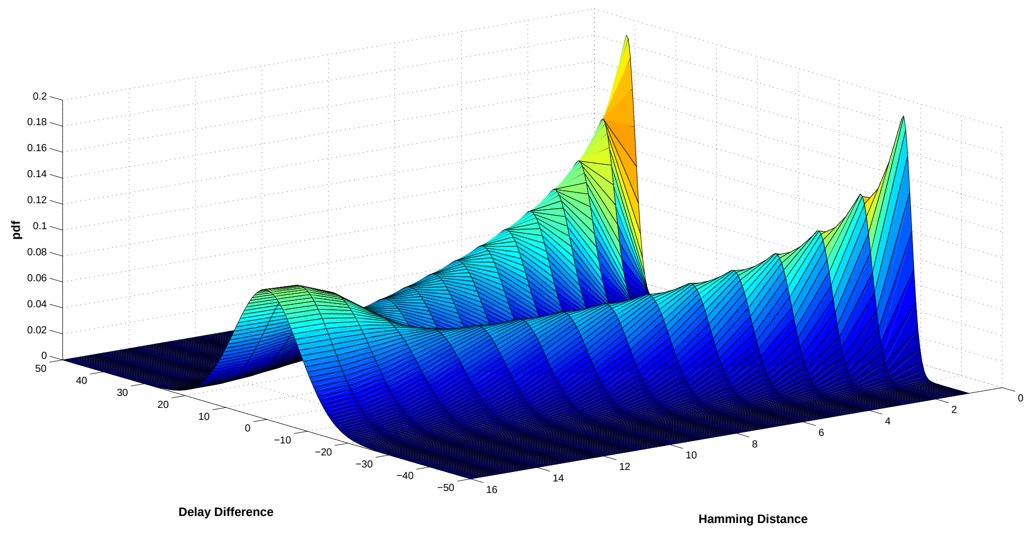

Figure 3.4: The distribution of delay difference given different hamming distance.

beginning, the average delay difference increases. Therefore, we can observe an obvious tradeoff between the hamming distance of the achieved challenges and the challenge stability.

$$Delay\ Diff = \sum_{i=1}^{i=m} |d_i - \hat{d}_i| + \sum_{j=m+1}^{j=m+n} (d_i - \hat{d}_i) \tag{3.2}$$

Algorithm 2 Delay Profile Approach

Input: delay profile information.

Input: C_i - random challenges.

Input: m - number of fixed bits.

Input: B_j - adjustable bits.

Input: $path0$ - path that outputs 0 in arbiter.

Input: $path1$ - path that outputs 1 in arbiter.

Input: $d_i \& \hat{d_i}$ - two delays in stage i.

```
1: for random challenge $C_i$ do
2:   assign $m$ bits to $C_i$ randomly.
3:   for all $B_j$ in $C_i$ (from the bit closest to arbiter to the
     farthest) do
4:     if $d_i \geq \hat{d_i}$ then
5:       assign $B_j$ to put $d_i$ to $path0$ ($path1$).
6:     else
7:       assign $B_j$ to put $\hat{d_i}$ to $path0$ ($path1$).
8:     end if
9:   end for
10: end for
11: Output: all the acquired stable challenges.
```

CHAPTER 4

Digital Bimodal Functions

4.1 A Motivational Example

The concept of the digital bimodal function (DBF) was first proposed by Xu et al. [20]. The essential idea behind the DBF is to represent a set of binary functions in two forms, one which is fast and compact ($f_{compact}$) and the other which is slow and complex ($f_{complex}$). Both forms have exactly the same functionality, in other words, given the same inputs, both forms produce the same outputs.

Equation 4.1, 4.2, and 4.3 illustrate an example of a DBF. As a prerequisite, a_i, b_i, and c_i are binary values, and the function sets f and g are Boolean functions in the form of sums of products (SOP) and/or products of sums (POS) representing $f_{compact}$ and $f_{complex}$, respectively. Equation 4.1 represents the relationship between a_i and b_i and Equation 4.2 represents the relationship between b_i and c_i. Note that each function f has 4 binary inputs assigned in a random and permanent order.

Equation 4.3 is generated by substituting 4.1 into 4.2, yielding a direct relationship between a_i and c_i. Note that substitutions are expanded and simplified so that each sub function in g is in the form of a SOP or a POS. The key observation here is that while both f and g implement the same functionality, f can be computed much more rapidly than g since it is in a compact format in which each subfunction requires only four inputs, while g is in an expanded format in which each subfunction requires up to n variables. It has been shown that the size

$$\begin{array}{ll} \text{Inputs:} & a_i \in \{0,1\}, i \in \{0,1,2...n-1\} \\ \text{Outputs:} & c_i \in \{0,1\}, i \in \{0,1,2...n-1\} \\ \text{Variables:} & b_i \in \{0,1\}, i \in \{0,1,2...n-1\} \\ & r_j \in \{0,1,2...n-1\}, j \in \{0,1,2...8n-1\} \end{array}$$

$$\begin{cases} b_0 = f_0(a_{r_0}, a_{r_1}, a_{r_2}, a_{r_3}) \\ b_1 = f_1(a_{r_4}, a_{r_5}, a_{r_6}, a_{r_7}) \\ b_2 = f_2(a_{r_8}, a_{r_9}, a_{r_{10}}, a_{r_{11}}) \\ \dots \\ b_{n-1} = f_{n-1}(a_{r_{4n-4}}, a_{r_{4n-3}}, a_{r_{4n-2}}, a_{r_{4n-1}}) \end{cases} \tag{4.1}$$

$$\begin{cases} c_0 = f_n(b_{r_{4n}}, b_{r_{4n+1}}, b_{r_{4n+2}}, b_{r_{4n+3}}) \\ c_1 = f_{n+1}(b_{r_{4n+4}}, b_{r_{4n+5}}, b_{r_{4n+6}}, b_{r_{4n+7}}) \\ c_2 = f_{n+2}(b_{r_{4n+8}}, b_{r_{4n+9}}, b_{r_{4n+10}}, b_{r_{4n+11}}) \\ \dots \\ c_{n-1} = f_{2n-1}(a_{r_{8n-4}}, a_{r_{8n-3}}, a_{r_{8n-2}}, a_{r_{8n-1}}) \end{cases} \tag{4.2}$$

$$\begin{cases} c_0 = g_0(a_0, a_1, a_3, \dots, a_{n-1}) \\ c_1 = g_1(a_0, a_1, a_3, \dots, a_{n-1}) \\ c_2 = g_2(a_0, a_1, a_3, \dots, a_{n-1}) \\ \dots \\ c_{n-1} = g_{n-1}(a_0, a_1, a_3, \dots, a_{n-1}) \end{cases} \tag{4.3}$$

difference between $f_{compact}$ and $f_{complex}$ increases exponentially with an increase in input variables and additional levels of substitution [20].

In order to visualize the size difference between $f_{compact}$ and $f_{complex}$, we first set up a few premise, then quantify their size difference.

The first premise is that we put both $f_{compact}$ and $f_{complex}$ into the form of a sum of products and simplify them by using the tool Simple Solver. We then compare the total number of products. Note that the simplification procedure reduces the size of the Boolean functions, but only by a constant factor. In Example I, after simplification, $f_{compact}$ can be expressed by the group of functions in (1c), which has 19 products, and $f_{expanded}$ can be expressed by the functions in (1e), which have 67 products.

The second premise is that each single sub-function in $f_{compact}$ has at most 4 inputs, just as shown in Example I, which is used to limit the size of $f_{compact}$.

The third one is that the number of outputs is the same as the number of inputs in each iteration, e.g., in Example I, the number of a_i equals to the number of b_i and $c_i, i \in \{0, 1, 2, ..., 7\}$.

The last premise is that the number of iterations is half of the number of inputs. While iterations can create size difference between $f_{compact}$ and $f_{complex}$, the size is also limited by the number of inputs. Therefore the number of iterations should be proportional to the number of inputs, as an example, we set it to be half.

Based on the above definition, with the increase in the number of inputs, we randomly generate a group of functions as $f_{compact}$ and the corresponding $f_{complex}$, then compare the number of products.

The size difference between $f_{compact}$ and $f_{complex}$ determines their difference in computation complexity. According to Table 4.1, the number of products in $f_{compact}$ grows linearly while the number of products in $f_{complex}$ grows exponentially. For example, when the number of inputs is 20, the average # of products in $f_{compact}$ is 57.8 while the average # of products in $f_{complex}$ reaches approximately 370000 which is 6400 times larger. Therefore, by increasing the number of inputs

# of iterations	# of inputs	avg. # of products ($f_{compact}$)	avg. # of products (f_{expand})
4	8	23.2±2.8	259.2±16.1
5	10	28.7±3.1	765.6±68.7
6	12	34.6±3.5	3816.0±245.2
7	14	39.0±3.8	11454.8±758.1
8	16	46.7±4.1	49206.4±2684.8
9	18	52.1±4.3	142491.6±7520.1
10	20	57.8±4.6	369656.8±19265.5

Table 4.1: Size comparison between $f_{compact}$ and $f_{complex}$ with different number of iterations and different number of primary inputs. The average number of products are tested with 95% interval confidence.

as well as the number of iterations, it is very easy to create a huge computation gap between $f_{compact}$ and $f_{complex}$.

4.2 FPGA-based Implementation

Figure 4.1 depicts the FPGA-based implementation of the $f_{compact}$ of the DBF defined in Equations 4.1, 4.2, and 4.3. The architecture is composed of two levels of 4-input LUTs. Note that each 4-input LUT implements a 4-input Boolean function from f. A hierarchy structure is constructed by feeding the outputs of the previous level of LUTs to the inputs of the next level of LUTs which is equivalent to the function substitution. Therefore, the LUT network directly implements $f_{compact}$ in the DBF. As the number of inputs and number of levels in the LUT network grows, the expanded form of $f_{complex}$ becomes very difficult to implement in hardware (grows exponentially) while $f_{compact}$ remains in a relatively compact form (grows linearly).

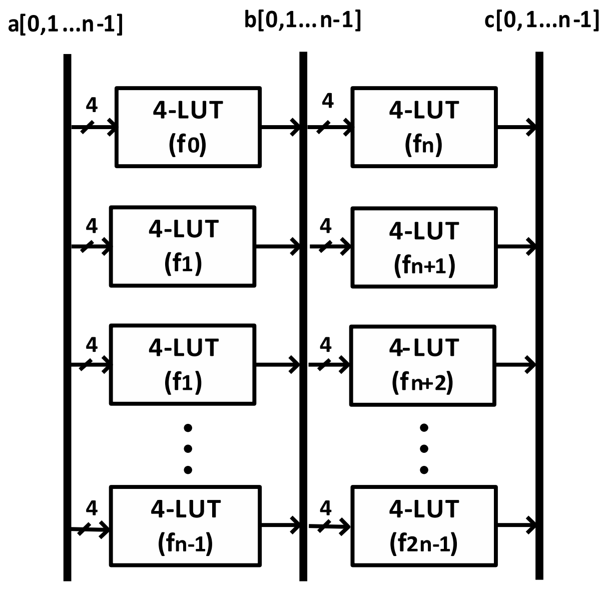

Figure 4.1: An example of the FPGA-based DBF $f_{compact}$ LUT network.

An even more compact FPGA implementation is to use sequential logic. The key idea is to iteratively use one level of LUT networks. This requires each level of sub functions in $f_{compact}$ to have the same format. For example, the combinational logic in Figure 4.2 can be replaced with the sequential logic in Figure 4.3. Compared to the combinational logic, the sequential logic saves the number of LUTs required. It takes iterations (each iteration is corresponding to a clock cycle) to produce the outputs while the number of clock cycles is equal to the levels of LUT in the combinational logic.

Now we consider the hardware implementation of $f_{complex}$. We use the Xilinx ISE Design Suite to synthesize $f_{complex}$ and compare the resources it requires with $f_{compact}$. For a SLC with a given number of inputs and cycles, we generate the corresponding $f_{complex}$, then convert it to a netlist and synthesize it to acquire the number of LUTs required in order for it to be implemented on the FPGA. We change the input # in the experiment and set the cycle # to the half of the inputs #. Table 4.2 indicates that $f_{complex}$ requires many more LUTs than $f_{compact}$ and the difference keeps growing with the increase of the input #, which

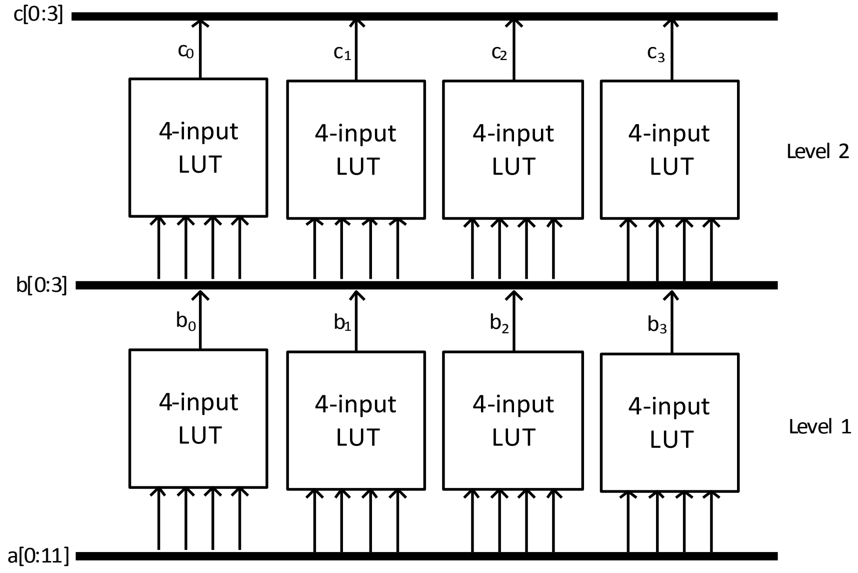

Figure 4.2: A combinational logic implementation of DBF $f_{compact}$ LUT network.

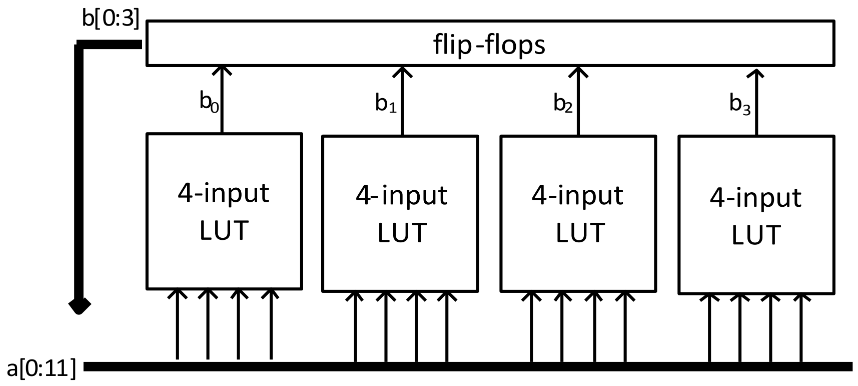

Figure 4.3: A sequential logic implementation of DBF $f_{compact}$ LUT network.

could easily reach the extent that the hardware implementation of $f_{complex}$ costs so much that it can only be simulated. Through this technique, due to the time difference between implementation and simulation, the time difference between $f_{compact}$ (private key) and $f_{complex}$ (public key) can be further expanded.

Levels	Inputs	$f_{compact}$ LUT #	$f_{complex}$ LUT #
4	8	8	81
5	10	10	396
6	12	12	1616
7	14	14	4353
8	16	16	13576
9	18	18	31155
10	20	20	98282

Table 4.2: Average synthesis resources compared between DBF form $f_{compact}$ and form $f_{complex}$. The Input # does not have to be twice as the circle #, we set it here as an example. The tests are based on the Spartan-3 XC3S50-5 FPGA and synthesized using the Xilinx ISE.

4.3 $f_{compact}$ and $f_{complex}$ Comparisons

Based on the above discussion, Table 6.2 shows the comparisons between $f_{compact}$ and $f_{complex}$. Both of them can be simulated, but only $f_{compact}$ can be implemented by the sequential logic on an FPGA using a reasonable amount of resources. As implementation is generally faster than simulation, we choose to implement $f_{compact}$ rather than to simulate it. When applying 20 inputs and 10 iterations, the implementation of $f_{compact}$ takes only 77.2ns while the simulation of $f_{complex}$ takes more than 1.6×10^7ns. The time difference reaches 2×10^5. Note that for an SLC with a larger size, the time difference can grow easily very large. For different applications, the size of SLC can vary.

	$f_{compact}$	$f_{complex}$
Operation	implementation	simulation
Time (8 inputs, 4 iterations)	29.2 ± 3.67(ns)	$(1.18 \pm 0.08) * 10^4$(ns)
Time (10 inputs, 5 iterations)	37.0 ± 4.02(ns)	$(4.33 \pm 0.49) * 10^4$(ns)
Time (12 inputs, 6 iterations)	45.1 ± 5.48(ns)	$(1.63 \pm 0.12) * 10^5$(ns)
Time (14 inputs, 7 iterations)	53.9 ± 5.51(ns)	$(5.77 \pm 0.48) * 10^5$(ns)
Time (16 inputs, 8 iterations)	61.4 ± 6.49(ns)	$(2.31 \pm 0.25) * 10^6$(ns)
Time (18 inputs, 9 iterations)	69.5 ± 7.36(ns)	$(6.75 \pm 0.49) * 10^6$(ns)
Time (20 inputs, 10 iterations)	77.2 ± 7.99(ns)	$(1.61 \pm 0.19) * 10^7$(ns)

Table 4.3: Comparisons between $f_{compact}$ and $f_{complex}$. The results show an average implementation/simulation time with the standard deviation.

The LUT network architecture based on configurable logic blocks (CLBs) on FPGA provides a very intuitive and simple way to implement $f_{compact}$, while for $f_{complex}$ the implementation on FPGA takes too many resources, requiring it to be simulated. The time and efficiency difference between $f_{compact}$ and $f_{complex}$ on FPGAs, which would be utilized for security purposes, offers us a strong reason to choose FPGA as an ideal platform.

CHAPTER 5

Digital PUF

5.1 Architecture

The ideal of digital PUF is first proposed by Xu et al. [29][30]. Figure 5.1 depicts the architecture of the digital PUF. At startup, the user selects and applies stable challenge vectors, supplied by the digital PUF manufacturer, to an array of delay-based PUFs. The resultant stable outputs are then used to initialize and configure individual LUT cells in the DBF. This procedure is applied to a random subset of LUT cells, while the remaining cells are initialized by the user. This bifurcation in initialization enables self trust by preventing malicious manufacturers from completely controlling the DBF configuration process.

After PUF initialization, the user generates an input-output mapping for the DBF which serves as a specification of $f_{complex}$. This is easily done by traversing all the possible inputs and generating the corresponding output. The mapping is stored as Boolean functions in both SOP and POS forms.

By applying only stable challenges to the delay-based PUF at initialization we ensure that the entire digital PUF system is completely stable. Furthermore, the intrinsic unclonability of the delay-based PUF along with its integration with the DBF guarantees that the overall architecture is unclonable. Since the delay-based PUF is used only at initialization and is subsequently disregarded and the rest of the digital PUF operation is delegated to the DBF, we inherit the small power, area, and low delay properties of the DBF.

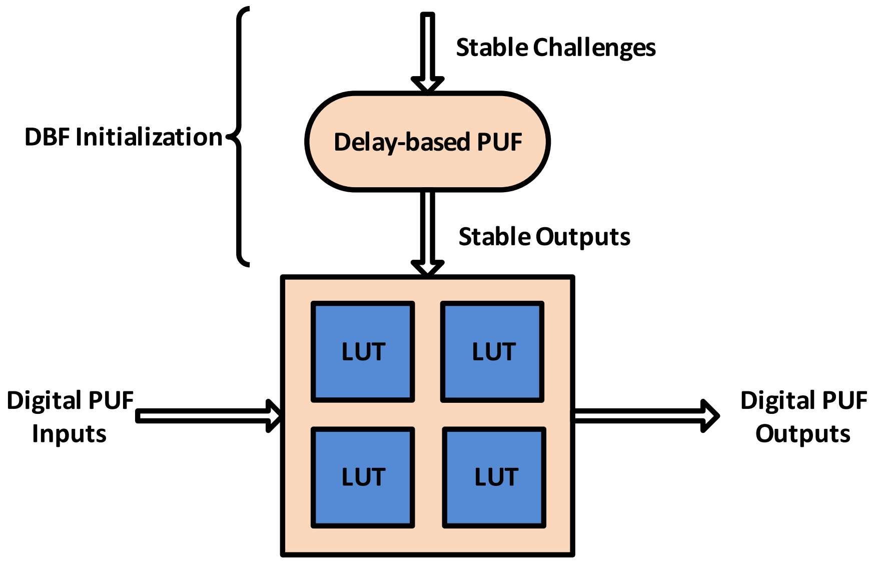

Figure 5.1: Architecture of the digital PUF. Note that the stable outputs from the analog PUF are used only once at startup to initialize and configure the LUTs in the DBF.

5.2 Operations

In order to use the FPGA-based digital PUF, a set of operations need to be done, we divide them into the following two steps: (i) FPGA configuration and (ii) DBF generation.

5.2.1 FPGA Configuration

The essential step before using the DBF is to reconfigure the FPGA for DBF initialization. As we mentioned in the previous section, in every clock cycle the user needs to choose a stable challenge vector $\{C_0, \ldots, C_{k-1}\}$ and feed it to the delay-based PUF. Each challenge is randomly chosen from a pool of stable challenges that is provided by the manufacturer. Note that if we reverse all the challenge bits to $\{\overline{C_0}, \ldots, \overline{C_{k-1}}\}$, the output is reversed too. Hence, there will not be any bias between the number of stable challenges that produce the 0 output and the 1 output. As a result, the output randomness of the delay-based PUF is not reduced because of the use of only stable challenges. Each of the generated output will be used to initialize one cell in a LUT. This procedure is repeated until a random portion of the LUT cells in the DBF are initialized, and the rest are initialized by the user. The reason to choose only a random part of LUTs to initialize is to enable self trust in order to prevent malicious manufacturers, because the manufacturers have no clue how the rest of the LUTs are initialized.

5.2.2 DBF Generation

After initialization of the DBF in the FPGA, the user generates the input-output mapping for the DBF which serves as $f_{complex}$. This can be easily done by traversing all the possible inputs and generating the corresponding outputs. The mapping is stored in the form of Boolean functions. Therefore, until now, the initialized DBF embedded in the FPGA serves as $f_{compact}$ and the mapping is $f_{complex}$.

5.3 Side-channel Attacks

In this section we discuss solutions for protecting the digital PUF against side-channel attacks on the LUT memory cells.

MicroSemi/Actel's antifuse-based FPGA employs one-time programmable connections that are non-volatile [31]. After each fuse is programmed, its probe and programming interface is automatically disabled. Actel fuses are smaller than the regular feature size of the FPGA and therefore are much less susceptible to destructive reverse engineering techniques. They have a very small power footprint (below $40\mu J$) that is significantly smaller than the footprint of a transistor. There are many millions or fuses and recovering their values is at best a very time consuming task. Note that in our approach, the fuses would be programmed in a unique way for each circuit. While dynamic reprogramming of fuses is not feasible, one can easily organize several combinations of fuses in such a way that their software activated combination produces a unique digital PUF.

The second potential approach to side-channel prevention is the use of inspection resistant memory proposed by Valamehr et al. [4]. They employ a combination of secret sharing and secure hashing to reduce the probability of correct key or device recovery to 10^{-12} even if the probability of incorrectly recovering a value from a particular location is only 5%. The overall hardware overhead is equal to slightly more than 7 SRAM cells. Note that as observed by Valamehr et al. their techniques can be combined with antifuse mechanisms.

Our most preferable solution against side-channel attacks is the use of 3D integrated circuit technology [32]. Recently, 3D integrated circuits have emerged as a practical industrial option that reduces some design constraints, such as long interconnect, yield, and levels of integration. 3D has been proposed several times for security application [33] [34], but only very recently has it been advocated as a platform for the detection of intrusive side-channel attacks [35]. Also, the use of configurable shields against side-channel attacks has been explored [36].

We propose the use of 3D implementation in which the security device, in our case, the digital PUF, is placed in the middle-most layers. Devices with the same architecture but with randomly selected parameters are placed both above and below the actual device. Another alternative is that all these devices are used in a standard secret sharing mode. Therefore, no backside access is possible, and the performance of electromagnetic attacks are drastically reduced or even eliminated. Finally, to prevent attacks through the same layers we can employ an active shield [36]. Our final observation is that all three discussed techniques are orthogonal and can be combined. Each technique does introduce some overhead but is relatively small. 3D techniques are applicable only if 3D technology is used.

CHAPTER 6

Security Properties

In this section, we adopt a set of standard statistical tests to analyze the security properties of the digital PUF. We describe possible statistical attacks and test the resilience of our digital PUF against such attacks. We use the standard digital PUF structure with 64-bit inputs and outputs and 32 levels of substitution. We assume that the digital PUF is initialized randomly.

6.1 Output Randomness

We quantify the output randomness of the digital PUF by applying the industry standard statistical test suite provided by the National Institute of Standards and Technology (NIST). We generate a stream of outputs in the following way: a random seed is used as the primary inputs to the digital PUF after random configuration and the corresponding outputs are generated. In each subsequent clock cycle, the outputs are XORed with the previous inputs to generate the inputs for the next clock cycle. We repeat the process until we collect enough outputs required by the benchmark suite. The results in Table 6.1 indicate that the output stream of the digital PUF passes the NIST randomness tests.

6.2 Avalanche Effect

In this attack, an adversary attempts to predict the outputs of the digital PUF using the knowledge of outputs for similar inputs. In cryptography, cipher diffusion

Statistical Test	Avg. Success Ratio
Frequency	100%
Block Frequency (m=128)	98.7%
Cusum-Forward	97.8%
Cusum-Reverse	97.9%
Runs	98.4%
Longest Runs of Ones	97.9%
Rank	99.3%
Spectral DFT	97.5%
Non-overlapping Templates ($m = 9$)	97.5%
Overlapping Templates ($m = 9$)	97.5%
Universal	100%
Approximate Entropy ($m = 8$)	98.1%
Rand. Excursions ($x = 1$)	98.8%
Rand. Excursions Variant ($x = -1$)	97.6%
Serial ($m = 16$)	99.3%
Linear Complexity ($M = 500$)	98.0%

Table 6.1: NIST randomness test results on the digital PUF. 1,000 bitstreams of 10,000 bits are provided to each test. Each test passes for p-value$\geq \sigma$, where $\sigma = 0.01$.

is achieved if a change in the input by one bit results in a dramatic change in the outputs in an unpredictable manner. This is otherwise known as the avalanche effect. To test this, we measure the hamming distance between two output vectors whose input vector differ by one bit. Ideally, the distribution should be in the form of a binomial distribution with the peak at half of the number of output bits. The result in Figure 6.1 shows an almost perfect binomial distribution which indicates our matched device satisfies the avalanche criterion and is highly resilient against this type of attack.

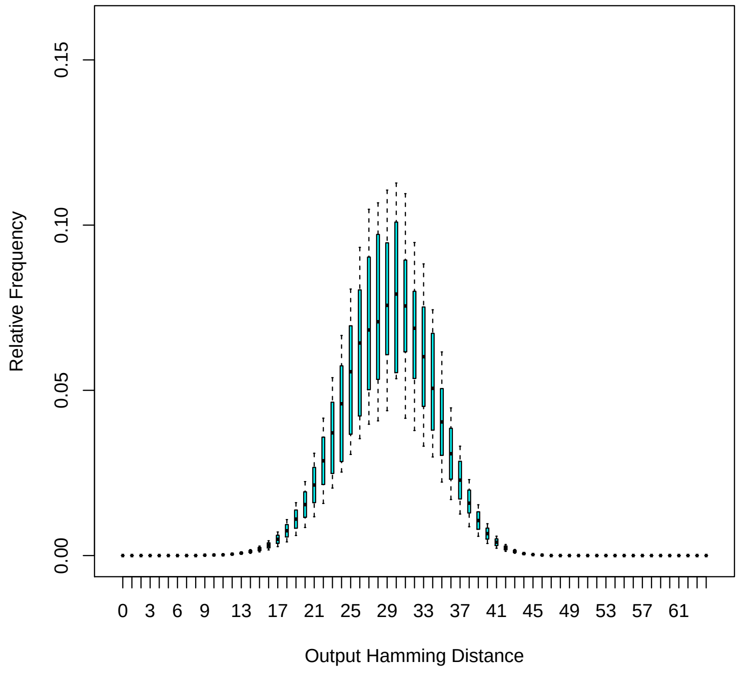

Figure 6.1: Distribution of output hamming distances testing the avalanche effect. The error bars depict the max, 0.75 quantile, mean, 0.25 quantile, and min frequencies.

6.3 Input-based Correlation

Another type of attack utilizes correlations between individual output bits, O_i, and input bits, I_j, for prediction. The goal in this attack is to predict the conditional probability, $P(O_i = c_1|I_j = c_2)$, where c_1 and c_2 are either 1 or 0. For example, if the attacker observes that output O_i is equal to 1 when the input I_j is 1 a large majority of the time, then he can guess with a high probability that output O_i is 1 when I_j is 1. The ideal situation is when all conditional probabilities are 0.5. Figure 6.2 and Figure 6.3 depicts the distribution of conditional probabilities, $P(O_i = 1|I_j = 1)$, for the digital PUF. The majority of probabilities cluster around 0.5, thus indicating low potential for prediction.

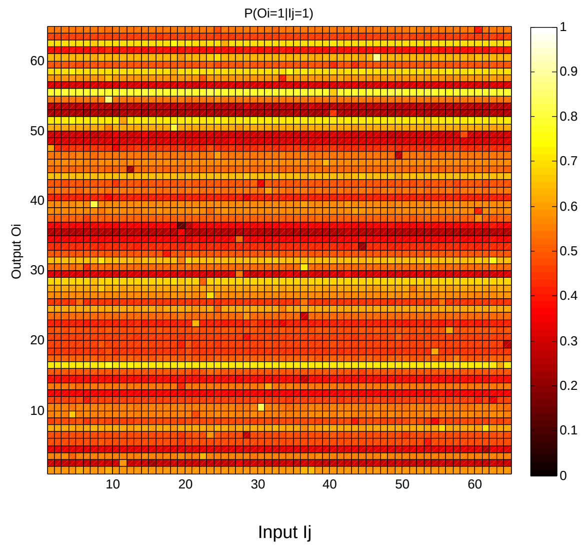

Figure 6.2: Colormap of conditional probabilities between output bits O_i and input bits I_j.

6.4 Output-based Correlation

Similar to the previously described attack, this attack attempts to predict an output bit O_i according to the value of a corresponding output bit O_j. In this case, if two output bits have a strong correlation, then the attacker can deduce the output vector through knowledge of a subset of output bits. We present the distribution of conditional probabilities, $P(O_i = 1|O_j = 1)$ in Figure 6.4 and the

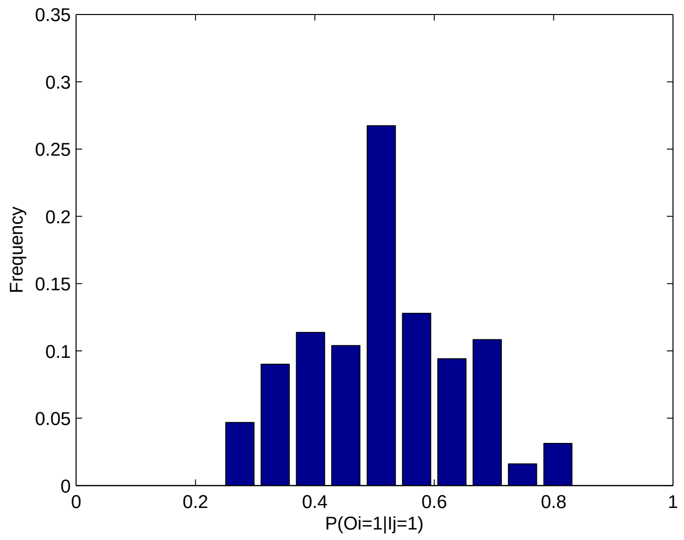

Figure 6.3: Distribution histogram of conditional probabilities between output bits O_i and input bits I_j.

corresponding histogram of the probability distribution Figure 6.5 which depicts low potential for prediction based on output to output correlation.

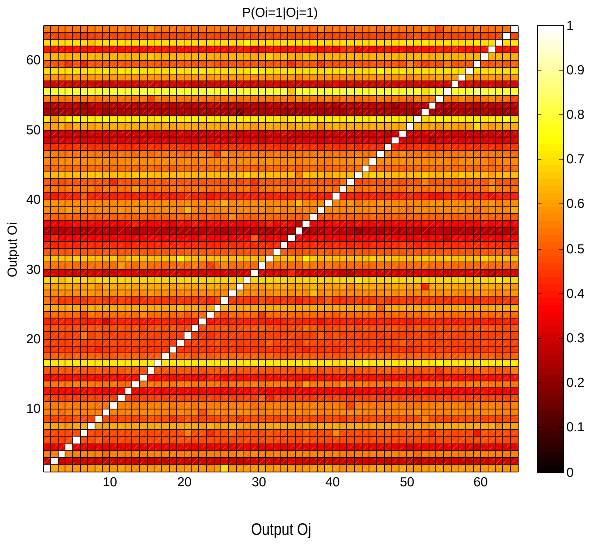

Figure 6.4: Colormap of conditional probabilities between output bits O_i and Output bits O_j.

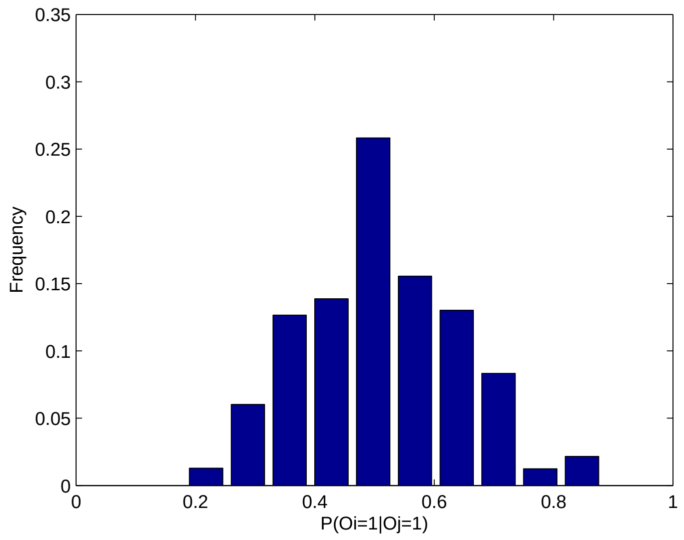

Figure 6.5: Distribution histogram of conditional probabilities between output bits O_i and Output bits O_j.

6.5 Comparisons

Finally, we briefly compare the statistical test results of our FPGA-based digital PUF with the traditional delay PUF. The results depicted in Table 6.2 are tested on the 64-input 64-output FPGA-based digital PUF and 64-input 64-output traditional delay PUF. We conclude that both our PUF and the traditional delay PUF demonstrate excellent properties regarding output frequency and conditional probability, but only our digital PUF demonstrates an ideal output hamming distance satisfying the avalanche criterion.

	Output Freq.	Hamming Dis.	$P(O_i = 1 \| I_j = 1)$
Digital PUF	0.5 ± 0.07	30.9 ± 3.6	0.49 ± 0.01
Delay PUF	0.5 ± 0.09	1.4 ± 0.4	0.52 ± 0.01

Table 6.2: Statistical test results comparison between the FPGA-based digital PUF and the traditional delay PUF. The ideal case for output frequency is 0.5, for hamming distance is 32, and for $P(O_i = 1|I_j = 1)$ is 0.5. The results shown in the table are the average values and corresponding standard deviations.

CHAPTER 7

Structure Exploration

The core architecture of the digital PUF is the randomly connected LUT network. In this section we address how to connect the LUTs to achieve optimal security, while ensuring that the the structure has a small area and delay overhead. The following factors can directly influence the LUT structure:

- **Number of Inputs.** The size and the complexity of $f_{complex}$ is directly dependent on the number of inputs to $f_{compact}$. Thus, the number of inputs should be selected in such a way so as to satisfy the application requirements for security, delay, and area.
- **Number of Levels.** Adding more LUT levels to the digital PUF causes more diffusion, however, also increases the delay and area costs.
- **LUT Connections.** In Figure 5.1, LUTs are connected in such a way that all the outputs feed into the inputs of the next level. However, this is not a mandatory requirement for the digital PUF. For example, feed forward structures can be used. Specifically, the inputs to a specific level of LUTs can come from or be controlled by the outputs from any previous level up to and including the primary inputs. Figure 7.1 illustrates two examples of feed forward structures for the digital PUF.

We explore the impact of input size on the digital PUF by measuring the success of the NIST tests for varying input sizes. Only when the digital PUF's output stream can pass all NIST tests do we claim that the configuration is acceptable.

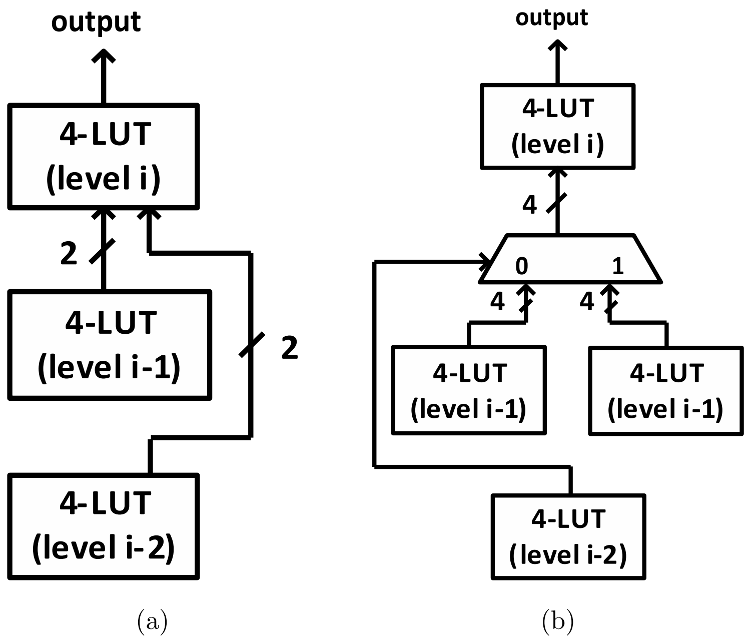

Figure 7.1: Examples of feed forward structures applied to the digital PUF. (a) Inputs arrive from all previous levels. (b) Inputs arrive from and are controlled by previous levels.

We begin with a digital PUF with 8 bit inputs and increment its input size by 8 after each test. We find that once the digital PUF reaches an input size of 32 bits it can consistently pass all NIST randomness tests.

We compare the different structures depicted in Figure 7.1 against the original digital PUF with varying levels of LUTs. We use the output hamming distance test from the avalanche criterion to compare the structures. Since the input size of each structures is 32 bits, the best structure will yield an average output hamming distance of close to 16.

Table 7.1 shows the results across the three types of structures. Two observations can be drawn from this table. The first is that the average hamming distance increases with more LUT levels for smaller sized digital PUFs but eventually stabilizes as the digital PUF grows. It can be concluded that after some growth, more levels would not significantly increase overall input diffusion. The second observation is that the feed forward structure in Figure 7.1b is the most secure of the three structures (in terms of satisfying the avalanche criterion) since its average output hamming distance reaches closest to 16.

Levels	4	8	12	16	20	24
Original Structure	8.8 ± 0.6	12.7 ± 0.2	12.8 ± 0.5	13.1 ± 0.4	13.3 ± 0.5	12.9 ± 0.7
Feed Forward in Figure 7.1a	6.7 ± 0.4	10.9 ± 0.7	12.6 ± 0.6	13.3 ± 0.4	12.9 ± 0.5	13.1 ± 0.9
Feed Forward in Figure 7.1b	9.9 ± 0.9	13.9 ± 0.8	14.8 ± 0.8	15.1 ± 0.5	15.1 ± 0.5	14.8 ± 0.7

Table 7.1: Output hamming distance averages and standard deviations across 20 random instances of each digital PUF structure. The input size is 32 bits. Each column corresponds to a given number of LUT levels in the PUF structure.

CHAPTER 8

Protocol

8.1 Public Key Communication

Public key communication is one of the most widely used communication protocols. We use it as an example to explain how digital PUF works as a security primitive in security protocols. The basic setting for this protocol is shown below.

- Private Key – $f_{compact}$, denoted by K_{priv}.
- Public Key – $f_{expanded}$, denoted by K_{pub}.
- Alice – the owner of K_{priv}. The party to receive and decrypt messages.
- Bob – the party to send and encrypt messages.
- TTP – trusted third party. The party that administrates K_{pub}.

Before explaining how this protocol works, we need to note that the private key $f_{compact}$ is actually a piece of the digital PUF, in which the hardware implementation of $f_{compact}$ is offered but not detected. By using K_{priv}, given input vectors, the output vectors can be calculated promptly. Meanwhile, the public key is the functions in $f_{expanded}$.

The core idea in Protocol 1 is to take advantage of the calculation time difference of $f_{compact}$ and $f_{expanded}$, making only the holder of K_{priv} be the only person who can calculate R_j ($j \in \{1, 2, ..., N\}$) in a reasonable amount of time. Another

Algorithm 3 Public Key Communication

```
1: Bob has a message $m$ to send to Alice, $m$ is in the form of binary vector.
2: Suppose there are $l$ single Boolean sub-functions in $K_{pub}$, all of which have the form of a sum of products and the form of a product of sums. In the first round, for the $ith$ ($i \in \{1, 2, ..., l\}$) function $f_i$ in $K_{pub}$, Bob randomly requests either one product from the a sum of products or one sum from the a product of sums from TTP. According to that product or sum, Bob generates a binary input vector $p_i$, making $r_i = f_i(p_i) = 1$ (case of product) or $r_i = f_i(p_i) = 0$ (case of sum). Then Bob concatenates $p_1$ to $p_l$ to generate binary vector $P_1$ and concatenates $r_1$ to $r_l$ to generate binary vector $R_1$.
3: Bob repeats step 2 for $N$ rounds, generates $P_j$ and $R_j$ ($j \in \{1, 2, ..., N\}$).
4: Bob calculates $E = m \oplus R_1 \oplus R_2 ... \oplus R_N$.
5: Bob broadcasts $E$ and all $P_j$ ($j \in \{1, 2, ..., N\}$).
6: Alice computes all the sub vector $p_i$ ($i \in \{1, 2, ..., l\}$) in each $P_j$ ($j \in \{1, 2, ..., N\}$) with $K_{priv}$ to find out the corresponding value of $r_i$, then Alice concatenates all the $r_i$ to form each vector $R_j$.
7: Alice computes $m = E \oplus R_1 \oplus R_2 ... \oplus R_N$ and gets Bob's message.
```

important design is that we use the trusted third party to administrate K_{pub}. Note that whenever Bob wants to send a message to Alice, he only needs to request one product or one sum of each function in K_{pub} from TTP, then he can create the binary vectors to encrypt his messages. This makes use of the sum of products and product of sums; by knowing one product in a sum of products, one input vector that makes the output to be 1 can be deduced and by knowing one sum in a product of sums, one input vector that makes the output 0 can be deduced. This design minimizes the key size as well as the energy for calculation that Bob requires during the encryption of public key communication. However, for an attacker, if he/she wants to find out the right c_x in C, since he/she does not have K_{priv}, he/she can only request all the information of K_{pub} from TTP to simulate. Therefore, for an attacker, the public key size and the calculation scale are not minimized at all, the expected effort of an attacker is to scan half of a sum of products as well as half of a product of sums. In Protocol 1, N rounds of operations are used to boost the calculation expense of $f_{compact}$ and $f_{expanded}$ N times simultaneously. As it takes much more time to calculate $f_{expanded}$ than $f_{compact}$, after both calculation expense increasing N times, the calculation time for $f_{compact}$ is still trivial but the time for $f_{expanded}$ increases significantly. N is a flexible number that can be adjusted according to the size of $f_{compact}$ and $f_{expanded}$.

8.1.1 Time Gap

We estimate the decryption time gap between the private key holder and the attacker in Protocol 1. Suppose the private key holder uses an SLC with 64 inputs and 32 cycles to implement $f_{compact}$ and the attacker simulates the corresponding $f_{expanded}$. For $f_{compact}$, the implementation time is tested on the Spartan-3 XC3S50-5 FPGA and was measured at approximately $239ns$. For the computation of $f_{expanded}$, the simulation time is too long and can only be estimated. It is obvious that the simulation time is proportional to the size of the $f_{expanded}$. According to Table 4.1, we find that the size of $f_{expanded}$ increases by at least 2.5 times with an increase of 1 iteration and 2 inputs. As a consequence, we can assume the simulation time for $f_{expanded}$ also grows similarly. Therefore, by combining the results in Table 6.2, the simulation time for an SLC with 64 inputs and 32 iterations can be estimated at $1.61 \times 10^7 \times 2.5^{22} = 9.15 \times 10^{15}(ns)$. We further assume that the number of rounds N in public key communication is 10^3, therefore, the time for private key calculation is $239 \times 10^3 = 2.39 * 10^5(ns)$ while the time for public key calculation is $9.15 \times 10^{15} \times 10^3 = 9.15 * 10^{18}(ns) \approx 290 years$, which is not acceptable. Note that N can be designed to be smaller with a larger size public key.

8.1.2 Performance Comparisons

We estimate the performance of the digital PUF based public key communication protocol and compare it with other cryptographic methods. For decryption, suppose the digital PUF uses a 64 input SLC, as a result, the area of 64 LUTs is required. The static power value of the Xilinx FPGA is approximately $24\mu W/CLB$. Suppose each CLB contains 4 LUTs, then 16 CLBs are required with a total power of approximately $384\mu W$. According to the synthesis result, the maxmium delay of the SLC circuit is $7.47ns$ and 32 cycles are needed to compute the outputs. We repeat step 2 in Protocol 1 N=10^3 times, so that the total clock cycle that decryption requires is 3.2×10^4. For encryption, according to the protocol, the public key user only needs to calculate one product or one sum of each function in $f_{expanded}$ in step 2 of protocol 1, compared to the decryption part, the expense of the encryption can be neglected. We therefore have an energy consumption estimation for digital PUF based public key communication through equation (2).

$$Energy = Power \times ClockCycle \ \# \times MaximumDelay \tag{8.1}$$

Design	Flip Flops	LUTs	Area (Slices)	Maximum Delay (ns)	Clock Cycles	Energy (μJ)	Block Size (bits)	Throughput (Mbps) at f_{max}	Throughput/Energy (Mbps/μJ)	Device
Present[37]	114	159	117	8.78	256	3.16×10^{-3}	64	28.46	9.01×10^{3}	xc3s50-5
HIGHT[37]	25	132	91	6.12	160	1.07×10^{-3}	64	65.48	6.12×10^{4}	xc3s50-5
AES[37]	338	531	393	14.21	534	3.58×10^{-2}	128	16.86	4.71×10^{2}	xc3s50-5
RSA[38]	1870	2811	1553	7.62×10^{3}	907	128.80	-	0.15	1.16×10^{-3}	xc3s500e
RSA radix-2[38]	7564	11496	6282	8.21×10^{3}	1058	654.80	-	0.12	1.83×10^{-4}	xc2v6000
RSA radix-4[38]	9944	14907	8328	4.23×10^{3}	560	236.73	-	0.43	1.82×10^{-3}	xc2v6000
DBF	64	64	32	7.47	32000	9.18×10^{-2}	-	267.74	2.92×10^{3}	xc3s50-5

Table 8.1: Comparisons for DBF based cryptography with the traditional block cyphers and RSA. The results for Present, HIGHT and AES are cited from [37], the results for RSA are the parts of multiplication modular and are cited from [38], the results for DBFs are tested on the Spartan-3 XC3S50-5 FPGA and generated by the Xilinx ISE Design Suite 14.3.

Table 8.1 shows the comparisons for DBF based public key communication with traditional block cyphers and RSA with respect to area, delay, and energy in FPGAs. The DBF in the table utilizes one SLC with 64 primary inputs and 32 iterations, and we suppose that the number of rounds N in the protocol is 10^3. We can obviously conclude from the table that DBF, as a security primitive, owns the implementation of ultra low energy that is competitive with the traditional security key block cyphers and outperforms the RSA with at least three orders of magnitude.

8.2 Remote Trust

When using a data centre to adopt remote computations, trust plays a very important role. On one hand, users need to authenticate the data centre, on the other hand, users also want to monitor the flow of their requested calculations to ensure the processing of the data is being carried out correctly. Digital PUFs provide an ultra low-energy and easy solution to authenticate the data centre as well as monitor the calculation flow. The basic approach is to use a hash tree.

The calculation flow shown in Figure 8.1 is an example of how a data centre processes calculations to generate the outputs. Four basic calculations $(+, -, \times, \div)$ are adopted as operation nodes in the flow. The data centre is required to randomly choose n places to "cut" the calculation flow, and generate n corresponding intermediate results. The calculation expense between the adjacent intermediate results is trivial. A hash tree is constructed based on the n intermediate results. Figure 8.2 shows an example of a hash tree with 4 leaf nodes, where each node represents a intermediate result from the data flow. A binary tree structure is generated in which every non-leaf node is the hash of the its two children nodes. Depends on the scale of the calculation, a hash tree may includes millions of intermediate results, and the leaf nodes from left to right coordinates with the

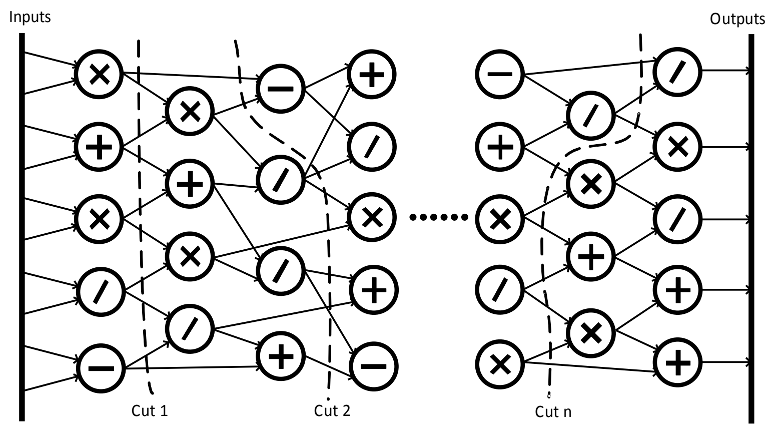

Figure 8.1: Example calculation flow and corresponding cuts. Each node in the graph represents a basic operation (e.g. $+, -, \times, \div$) or even blocks of operations (e.g. if-else, while, functions). Cut 1 to cut n represent random cuts in the calculation flow which can be thought of as intermediate results or states of the procedure.

sequence of the intermediated results generated in the data flow.

To apply a digital PUF on the structure of hash tree, we us the following basic settings.

- Use DBF as the hash function.
- Only the data centre has the digital PUF, which is the hardware implementation of DBF form $f_{compact}$.
- DBF form $f_{expanded}$ is public.

Based on these settings, every time a client wants to monitor the calculation, he/she randomly chooses a leaf node of the hash tree and requests the data centre to offer the "corresponding hashed results" to calculate the path from the leaf node to the top node. For example, in Figure 8.2, the top value of the hash tree can be verified by iteratively hashing intermediate result 2 with the results in hash

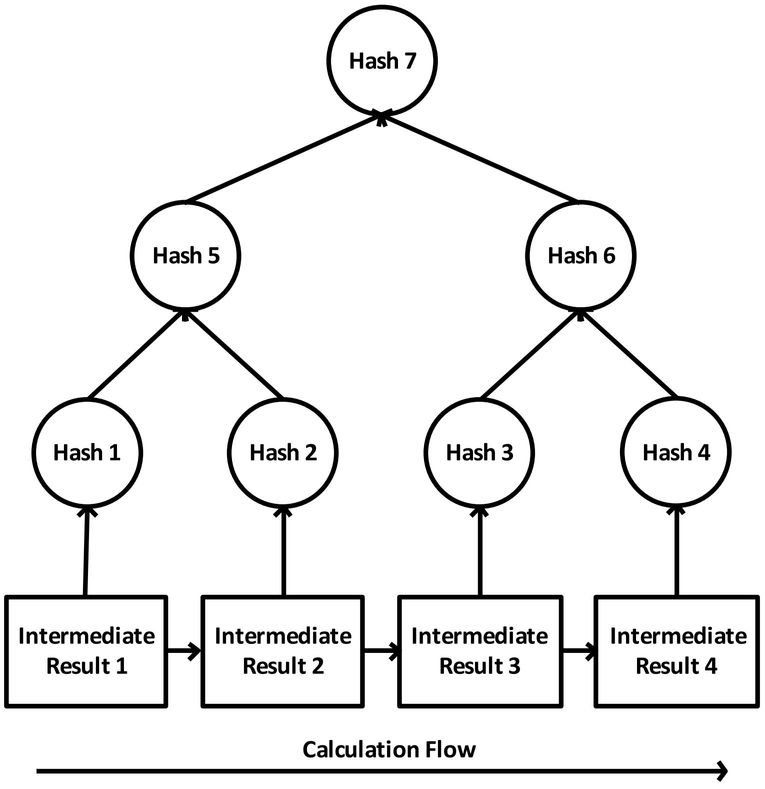

Figure 8.2: An example of a Hash tree for our low overhead remote trust protocol. The intermediate results at each cut in the calculation flow (e.g. cut i, $i \in\{1, 2, ..., n\}$ in Figure 8.1) are hashed as leaf nodes. The arrow shows the direction of calculation flow.

1 and hash 6. In this case, the results in hash 1 and hash 6 are the corresponding hashed results for intermediate result 2. After acquiring the "corresponding hashed results", the client hashes the chosen intermediate result and confirms the rightness of the hashing path afterwards. By repeating this procedure the consistence of the whole hash tree is checked. Another verification that a client can do is to check the calculation between the adjacent intermediate results, e.g., intermediate result 2 and 3. By knowing intermediate result 2, the client can follow the calculation flow to calculate and confirm the intermediate result 3. As the calculation expense between adjacent results is trivial, the client can easily verify the consistence of the adjacent nodes. Therefore, the client can detect the hash

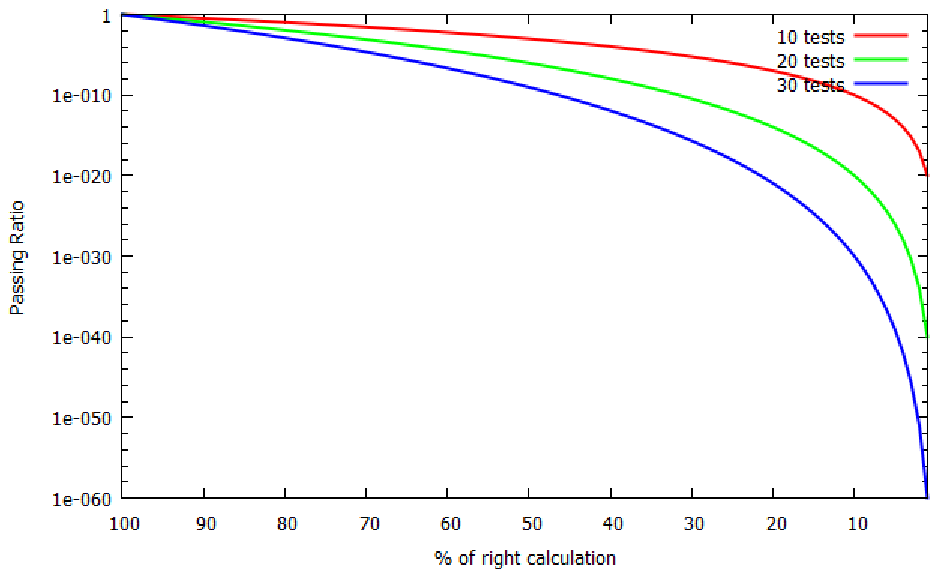

Figure 8.3: Passing ratio with the proportion of right calculation. The 3 curves respectively shows the passing ratio under the circumstance that the client requests 10, 20, and 30 pairs of adjacent nodes to test.

tree both horizontally and vertically. Due to the randomness of every request, only the party that has the whole structure of the hash tree can respond to all the requests correctly, therefore, the calculation flow is monitored and detected.

Monitoring of the calculation flow is used to verify whether the data centre is processing the flow correctly. Now suppose the data centre is not honest and mixes some wrong or irrelevant calculations in the flow. Figure 8.3 shows the possibility that the data centre can pass all the adjacent nodes tests from the client with only some percentage of right calculations in the flow. The results show that with a linear decrease of the proportion of right calculation, the passing ratio drops exponentially. For example, when the right calculation proportion reaches 1% in the case of 10 test pairs, the passing ratio drops to be 10^{-20}, which is ridiculously small.

Note that since the data centre needs to hash millions of intermediate results during the calculation flow, to complete the hashes in a reasonable short time,

$f_{compact}$ must be used. Any unauthenticated party who only has $f_{expanded}$ suffers a long time of hashing. As an example, suppose we use the SLC with 20 inputs and 10 iterations in remote trust. Again, we emphasize that we have flexibility in choosing SLC's size for different applications. Suppose that the number of intermediate results is 10^6 and the client makes 10 hash operations. According to Table 6.2, the calculation time for the data centre is $77.2 \times 10^6 ns = 0.0772 s$ and the calculation time for the client is $1.61 \times 10^7 \times 10 ns = 0.161 s$. However, the calculation time for an attacker reaches $1.61 \times 10^7 \times 10^6 ns = 1.61 \times 10^4 s = 4.47 hrs$. As a consequence, only the party that can offer requested hash results correctly in a reasonably short period of time will be certified by the client. In this way, the only party that can be authenticated is the data centre with the digital PUF.

CHAPTER 9

Conclusion

We have presented the concept of digital PUF which resolves two essential problems in traditional analog PUF. The first problem is stability. Analog PUF is unstable in the same sense that analog system is unstable. Digital PUF resolves this by leveraging a stable delay-based PUF for initializing its connected network of LUTs of digital bimodal functions (DBFs). The stability in the delay-based PUF is ensured by selecting challenges that have a delay ratio of at least 10% which ensures that the output is always stable for temperatures ranging from 250K to 400K. This process guarantees the system to be both unclonable and stable. The second problem of analog PUF is hard to be integrated with digital logic. Digital PUF resolves this problem by employing completely digital system.

Digital PUF is built based on the strutter of digital bimodal functions. DBF has two forms of functions, among which one form is fast and compact, the other form is slow and complex. We proposed the architecture of DBF on FPGA and analyzed the security properties. The security analysis indicates that DBF can pass all benchmark tests from the NIST randomness suite, as well as the avalanche criterion. Based on the standard statistical tests, we further compared and discussed the possible structures of digital PUF.

Finally, Two security protocols are demonstrated, respectively security protocols and remote trust. For security protocols, a fast speed, low overhead public key communication protocol is proposed. It employs the huge computation gap between the two forms of functions in DBF. We have also demonstrated the application of the digital PUFs in a remote secret key exchange protocol in which both communicating parties experience very low overhead in terms of both time and energy.

References

[1] "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," National Institute of Standards and Technology (NIST) Special Publication 800-22, Rev. 1a, Apr. 2010.

[2] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656715, 1949.

[3] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning physically unclonable functions," in *HOST*, pp. 16, 2013.

[4] J. Valamehr et al., "Inspection resistant memory: architectural support for security from physical examination," in *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 130141, 2012.

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

[6] 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.

[7] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *CHES*, pp. 6380, 2007.

[8] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *DAC*, pp. 914, 2007.

[9] J. W. Lee et al., "A technique to build a secret key in integrated circuits for identification and authentication applications," in *Symposium on VLSI Circuits*, pp. 176179, 2004.

[10] S. Devadas et al., "Design and implementation of PUF-based unclonable RFID ICs for anti-counterfeiting and security applications," in *IEEE International Conference on RFID*, pp. 5864, 2008.

[11] E. Simpson and P. Schaumont, "Offline hardware/software authentication for reconfigurable platforms," in *CHES*, pp. 311323, 2006.

[12] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *USENIX Security Symposium*, pp. 291306, 2007.

[13] M. Potkonjak, S. Meguerdichian, and J. L. Wong, "Trusted sensors and remote sensing," in *IEEE Sensors*, pp. 11041107, 2010.

[14] J. B. Wendt and M. Potkonjak, "Nanotechnology-based trusted remote sensing," in *IEEE Sensors*, pp. 12131216, 2011.

[15] G. E. Suh et al., "Design and implementation of the AEGIS single-chip secure processor using physical random functions," in *ACM SIGARCH Computer Architecture News*, vol. 33, pp. 2536, 2005.

[16] N. Beckmann, M. Potkonjak, "Hardware-Based Public-Key Cryptography with Public Physically Unclonable Functions," *Information Hiding: 11th International Workshop*, pp. 206-220, Darmstadt, Germany, 2009.

[17] U. Rührmair, "SIMPL systems, or: can we design cryptographic hardware without secret key information?," in *SOFSEM*, pp. 2645, 2011.

[18] S. Meguerdichian, M. Potkonjak, "Matched public PUF: ultra low energy security platform," *IEEE/ACM ISLPED*, pp. 45–50, 2011.

[19] S. Meguerdichian, M. Potkonjak, "Security Primitives and Protocols for Ultra Low Power Sensor Systems," *IEEE SENSORS*, pp. 1225-1228, October 2011.

[20] T. Xu, J. B. Wendt, M. Potkonjak, "Digital Bimodal Function: An Ultra-Low Energy Security Primitive," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 292-297, 2013.

[21] M. Fyrbiak, C. Kison, and W. Adi, "Construction of software-based digital physical clone resistant functions," in *International Conference on Emerging Security Technologies*, pp. 109112, 2013.

[22] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 1, pp. 5, 2009.

[23] U. Rührmair et al., "Modeling attacks on physical unclonable functions," in *Computer and Communications Security*, pp. 237249, 2010.

[24] X. Xu and W. Burleson, "Hybrid side-channel/machine-learning attacks on PUFs: a new threat?," in *DATE*, pp. 349, 2014.

[25] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *CHES*, pp. 2945, 2003.

[26] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *CHES*, pp. 212, 2003.

[27] J. A. Halderman et al., "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 9198, 2009.

[28] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, “Hotspot: A compact thermal modeling methodology for early-stage VLSI design,” *IEEE Transations on VLSI Systems*, vol. 14, no. 5, pp. 501-513, 2006.

[29] T. Xu, M. Potkonjak, “Robust and Flexible FPGA-based Digital PUF”, *International Conference on Field Programmable Logic and Applications*, pp. 1-6, September, 2014.

[30] T. Xu, J. B. Wendt, M. Potkonjak, “Secure Remote Sensing and Communication using Digital PUFs”, *Symposium on Architectures for Networking and Communications Systems*, pp.1-12, October, 2014.

[31] “Implementation of security in Actels ProASIC and ProASICPLUS flash-based FPGAs,” `"http://www.actel.com/documents/Flash_Security_AN.pdf"`, 2003.

[32] D. H. Kim, K. Athikulwongse, and S. K. Lim, “A study of through-silicon-via impact on the 3D stacked IC layout,” in *ICCAD*, pp. 674680, 2009.

[33] T. Huffmire et al., “Hardware trust implications of 3-D integration,” in *Proceedings of the 5th Workshop on Embedded Systems Security*, pp. 1, 2010.

[34] J. Valamehr et al., “A qualitative security analysis of a new class of 3-D integrated crypto co-processors,” in *Cryptography and Security: From Theory to Applications*, pp. 364382, 2012.

[35] S. Briais et al., “3D hardware canaries,” in *CHES*, pp. 122, 2012.

[36] U. Guvenc, “Active shield with electrically configurable interconnections,” in *SECUREWARE*, pp. 4345, 2013.

[37] P. Yalla, J-P. Kaps, “Lightweight cryptography for fpgas,” *Reconfigurable Computing and FPGAs, 2009. ReConFig’09. International Conference on.* IEEE, 2009.

[38] E. Oksuzoglu, E. Savas, “Parametric, secure and compact implementation of RSA on FPGA,” *Reconfigurable Computing and FPGAs, 2008. ReConFig’08. International Conference on.* IEEE, 2008.