ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

RELIABILITY AND SECURITY OF ARBITER BASED PHYSICAL UNCLONABLE FUNCTION

M.Sc. Thesis by

Zaur TARIGULIYEV

DEPARTMENT : ELECTRONICS ENGINEERING

PROGRAMME : ELECTRONICS ENGINEERING

Thesis Supervizor : Assist. Professor Dr.Sıddıka Berna ÖRS YALÇIN

DECEMBER 2010

ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

RELIABILITY AND SECURITY OF ARBITER BASED PHYSICAL UNCLONABLE FUNCTION

M.Sc. Thesis by

Zaur TARIGULIYEV

504021229

Date of submission :

Date of defence examination:

Supervisor (Chairman) : Assist. Prof. Dr. Sıddıka Berna ÖRS

Members of the Examining Committee : Prof. Dr. Ece Olcay GÜNEŞ

Members of the Examining Committee : Assist. Prof. Dr. Gökay SALDAMLI

2011

FOREWORD

I would like to extend my sincere gratitude to my advisor, Berna ÖRS, for her inspiring advice and encouragment through the beginning to the end of this search. I would like to express my appreciation and thanks to my best friend, Zafer Işcan, providing me the PR software tools, and supporting me with some ideas. Furthemore, I’m thankful to my friends from Embedded System laboratory for technical support in experiments with FPGA.

Decemeber 2010 Zaur TARIGULIYEV

Electronics Engineer

TABLE OF CONTENTS

1. INTRODUCTON 17
2. DEFINITION AND APPLICATION OF PHYSICAL UNCLONABLE FUNCTIONS . 21
   2.1 Definition of One–Way Function **21**
   2.2 Definition of Physical Unclonable Function **22**
   2.3 PUF Based RFID Authentication **23**
3. DESCRIPTION AND LINEER MODEL OF PUF CIRCUIT 25
   3.1 General Description **25**
   3.3 Delay Paths in the PUF Circuit **28**
      3.3.1 Switch delay 28
   3.4 Symmetrical Structure of PUF circuit **31**
   3.5 Lineer Model of an Arbiter-based PUF **33**
4. IMPLEMENTATION OF A PHYSICAL UNCLONABLE FUNCTION ON AN FPGA 35
   4.1 Introduction to a FPGA **35**
   4.2 Implementation of PUF components **36**
      4.2.1 Switch implemented with MUX 36
      4.2.2 Arbiter 39
   4.3 Placement of Switch Box **41**
5. ANALYSIS AND CHARACTERIZATION OF ARBITER-BASED PUF 43
   5.1 Inter-chip Variation **43**
      5.1.1 Information –bearing challenges 43
      5.1.2 Definition and evaluation of inter-chip variation 45
   5.2 Environmental Variations **46**
   5.3 Identification/Authentication Abilites **47**
6. SOFTWARE ATTACKS ON ARBITER-BASED PHYSICAL UNCLONABLE FUNCTION 49
   6.1 Prediction and Calculation Tools **50**
      6.1.1 Lineer Programming method 51
      6.1.2 Lineer Support Vector Machine 52
      6.2.3 Radial Base Functions Support Vector Machine 54
   6.2 Attacks on PUF Circuit Using Linear Programming Aproach **56**
      6.2.1 Experiments using data generated by Matlab 56
      6.3.2 Experiments using data measured from the PUF on the FPGA 57
   6.3 Attacks on PUF Circuit Using Support Vector Machine Classification **58**

6.3.1 Experiment using data generated by Matlab..58
6.3.2 Experiments using data measured from the PUF on the FPGA............................59
6.4 Reason of Failure in Prediction of PUF Circuit Responses....................................60
7. CONCLUSION..63
APPENDICES 69
CURRICULUM VITAE..71

ABBREVIATIONS

POWF : Physical One Way Function

PUF : Physical Unclonable Function

MUX : Multiplexer

SVM : Support Vector Machine

FPGA : Field Programmable Gate Array

LUT : Lookup Table

RBF : Radial Basis Function

LIST OF TABLES

Table 5.1: Table form of PUF responses, indexed with j, against randomly chosen i challenges..

Table 6.1: Results of prediction test using linear programming approach for PUF implemented with MUXs and response generated by the mathematical model.. 4

Table 6.2: Results of prediction test using linear programming approach for PUF implemented with MUXs and response measured from the PUF on the FPGA... 4

Table 6.3: Results of prediction test using Lineer SVM classifier for PUF implemented with MUXs and response generated by the mathematical model............................ 4

Table 6.4: Results of prediction test using Lineer SVM classifier for PUF implemented with MUXs and response measured from the PUF on the FPGA.............................. 4

Table 6.5: Results of prediction test using RBF SVM classifier for PUF implemented with MUXs and response measured from the PUF on the FPGA............................. 4

RELIABILITY AND SECURITY OF ARBITER BASED PHYSICAL UNCLONABLE FUNCTION

SUMMARY

Modern cryptographic protocols are based on the premise that only authorized participants can obtain secrets keys and access to information systems. However, various kinds of tampering methods have been devised to extract secret information from smartcards and ATMs. From storage of digital secret key in a chip we came to an idea to use physical property of nonhomogenious material that make him unique . This property make a secret key unclonable and due to this, the structure created from this material are called physical unclonable function. First time, this idea was realized using optical micro-structure with bubbles. Then, silicon material was used to realize the Arbiter-based Physical Unclonable Functions (PUFs). This technique exploit statistical delay variation of wires and transistors across integrated circuits (ICs) in the manufacturing processes to build a secret key unique to each IC. We implemented Arbiter-based PUFs in Xilinx FPGA and investigated the identification capability, reliability, and security of this scheme. Experimental results and theoretical studies show that a sufficient amount of variation exists across ICs. This variation enables each IC to be identified securely and reliably over a practical range of environmental variations such as temperature and power supply voltage.

FIZIKSEL KOPYALANAMAZ FONKSIYONUN GÜVENLİĞİ VE GÜVENİRLİĞİ

ÖZET

Modern kriptografik protokoller sadece yetkili kişilere sistemdeki bilgiye ve gizli anahtara erişimini sağlamaktadır. Ancak, smart kartlardan ve ATM’lerden gizli bilgileri elde etmek amacıyla değişik teknikler geliştirilmiştir. Gizli anahtarın dijital veri yerine maddenin homojen olmayan ve o maddeyi tek yapan fiziksel özelliği ile ilişkilendirilmesi fikir olarak sürülmüştür. Bu özellik gizli anahtarın kopyalanmasını imkânsız kıldığı için bu maddeden oluşan yapıya fiziksel kopyalanamaz fonksiyon adı verilmiştir. Bu fikri önce içinde kabarcık olan cam, sonra silikon maddesi kullanılarak hayata geçirilmiştir. Bu yöntem her bütünleşmiş devresinin ait tek gizli anahatarı oluşturmaktadır. Bunu için bütünleşmiş devrelerin üretim sürecindeki oluşan hat ve tranzistorlardaki geçikme varyasyonları kullanılmaktadır. Arbiter tabanlı PUF devresi Xilinx FPGA’de gerçeklendi. PUF devresi kimlik belirleme yeteneği, güvenirlik ve güvenlik açısından incelendi. Deneysel sonuçlar ve teorik çalışmalar entegre devre içinde varyasyon yeterli miktarda var olduğunu göstermektedir. Bu varyasyonlar her entegre devreyi, ısı ve güç kaynağı gerilimin değişimi makul sınırların içinde kalmak kaydıyla, güvenilir ve sağlam şekilde tespit edilmesini sağlıyor.

1. INTRODUCTON

Typically, cryptography is used to secure communication between two parties connected by an untrusted network[1]. In such communication, each party has privately stored key information which allows it to encrypt, decrypt, and authenticate the communication. It is implicitly assumed that each party is capable of securing its private information. This assumption is reasonable when a party is a military installation, or even a person, but breaks down completely for low-cost consumer devices. Once a secret key is attained, eavesdropping and impersonation attacks become possible[2].

In low cost devices such as RFIDs and smartcards is essential that sensitive information is safely stored and communicated. However, the inherent power and footprint limitations of such devices, prevent us from employing standard cryptographic techniques for authentication which were originally designed to secure high end systems with abundant power. In practice, the implementation cost of cryptographic hash functions is near that of block ciphers which is around 10K logic gates [3,4]. For RFIDs the footprint allotted for security is less than 1K gates [5]. Public-key cryptography bears significant computational overhead when compared to secret key techniques. Furthermore, even if the footprint problem is solved, each time an authentication takes place, the device has to transmit large amounts of data through the channel to the reader, which will unnecessarily consume the power of the device.

Another issue of secret key technique is resilience against invasive and non-invasive physical tampering attacks. Laser cutting, microprobing and power analysis have made it possible to extract digitalized secret information from ICs and compromise conditional access systems by using illegal copies of the secret information[2]. Focusing on the problem of invasive attacks, it is apparent that once a device has been opened, the large difference in state between a 0 and a 1 makes it relatively easy to read out the device’s digitally stored secrets. Traditionally, such attacks are avoided by detecting intrusion and erasing the key memory when an intrusion is detected [6]. However, tamper-sensing environments are expensive to produce and, as long as a key is being protected, the intrusion sensors need to be powered, further increasing costs. In order to resist to the cloning and invasive attacks random functions based on the randomness in physical materials were proposed.

Chronology in development of unclonable artifacts show that the first powerful notion of a physical one-way functions (POWF) was introduced by Papu et. al [7]. In the study of the physical one way functions he used transparent optical medium with a 3 –dimensional micro-structure containing bubbles. The input-challenge of the POWF is an incomming laser beam and the output /response is a fixed-length bit vector derived from resulting interference patterns. The interference pattern depends on the angle and frequency of incoming beam and the speckle pattern in the optical medium. After that, Gassend et. al. introduced the concept of a silicon physical unclonable function or silicon PUF [8,9]. Modern and future silicon technology-based integrated circuits may serve as PUFs due to their intrinsic manufacturing variability. Essentially, a number of unavoidable physical and chemical phenomena, such as silicon lattice imperfection, uneven distribution of dopants, imperfect mask alignment, and non-uniform chemical mechanical polishing, result in gates with sharply different characteristics. Already in 45 nanometer technology, it is common that the delay of the same gate in different ICs differs by 1/3 from the nominal value [7]. Therefore two silicon PUFs, having the same structure and designed to be sensitive to circuit delays, implemented on the same IC or different ICs have different responses to the same inputs.

Since process variation is beyond manufacturers' control, even an adversary who has detailed information of the PUF circuit cannot physically clone the silicon PUF of a given IC. So the authentication in PUF circuits are based on hidden delay or timing information corresponding to a circuit rather than digitial information [10]. Since there are several types of PUFs with different structure and complexities from a security point of view(XOR PUF, Feed Forward PUF, Ring Oscilator PUF) we consider only basic type called arbiter-based PUF.

The main purpose of this thesis is to investigate the reliability and resistance of a PUF, implemented on FPGA, to software attacks. Also, we aim to implement PUF structures based on MUXs and make analysis on data obtained from them. These data enable us to study the characteristics of the PUFs and sensitivity to changes to environmental factor such as the ambient temperature and voltage fluctuations.

This thesis is structured as follows. Chapter 2 defines physical unclonable functions and physical one-way functions, gives an example of one-way mathematical function. Furthemore, in this chapter one of the most common application of PUF is considered.

In Chapter 3 general overview of PUF system and notion of the delays is considered. In this we cover the fundamental components containing in the differential structure of PUF. Also, we show how delay path can be configured using additive property of the delay and underline the conditions that are providing more sensitivity to inherit process variation in PUF circuits. We provide a linear model of PUF circuit where responses are expressed in terms of challenges and delay variables.

Chapter 4 introduces a detailed circuit implementation of arbiter-based PUF on a Xilinx FPGA device using program Xilinx ISE 9.2 . We shortly touches on the primitive logic elements that enable us to implement the main components of PUF circuit. Furthemore, the hardware description language codes(VHDL) that generate these components are also given. In this chapter we show how to adjust “synthesize” tool that comes inside Xilinx ISE 9.2 in order to prevent from converting MUX. Also with the integrated in ISE environment “Floorplanner ” tool we try to achieve symmetrical placement of switch blocks.

In Chapter 5 the experimental results for PUFs implemented on FPGA are shown. We analyze the important, for security and reliability, characteristics of PUFs such as the inter-chip variation and environmental noise.

Chapter 6 studies the vulnerability of the arbiter-based PUFs against possible software attack models. Based on the linear model we make attacks using a limited number of linear inequalities and linear programming technique. Using this method, we try to solve for uknown delay parameters of the circuit. Then we compares the responses produced by our theoric model and the model extracted by linear programming algorithm. In the second attack we use artificial neural network. Providing challenge-response pairs for training of the Supported Vector Machine neural networks, the prediction capabilites of these method are investigated. Also, in this chapter is told about causes of inconsistency between responses of a lineer model of a PUF with the responses obtained from FPGA.

Finally, in the last chapter all the parameters that have effects on security of PUF on FPGA are summarized.

2. DEFINITION AND APPLICATION OF PHYSICAL UNCLONABLE FUNCTIONS

2.1 Definition of One–Way Function

Information security requires a mechanism that provides significant asymmetry in the effort required to make intended and unintended uses of encoded information. Modern cryptographic practice rest on the use of one-way functions(OWF). If f is one–way function i.e for any argument it is easy to compute but extremely hard to invert, then, even if f and $f\ (PW)$ were made public, it would be nearly impossible for a reasonable adversary to compute the password (PW) from $f(PW)$. Here, a reasonable adversary is one that does not have access to exponential computing resources.

Here is the formal definition of one-way functions.

A function f:{0,1}→{0,1}* is called strongly one-way if the following two conditions hold*[2].

- Easy to compute: *There exist a deterministic polynomial time algorithm A such that on input x, A outputs* $f(x)$
- Hard to invert: *For every probabilistic polynomial time algorithm A', every polynomial* p*, and all sufficiently large* n

$$\mathbf{Pr}\left(\boldsymbol{A}'(\boldsymbol{f}(\boldsymbol{y})) \in \boldsymbol{f}^{-1}(\boldsymbol{f}(\boldsymbol{y}))\right) < \frac{\mathbf{1}}{\boldsymbol{p}(\boldsymbol{n})} \qquad \textbf{(2.1)}$$

Saying that a function f is easy to compute means that there exist a P-time algorithm A which, given an input x , output $f(x)$. The second condition means that the probability that algorithm *A'* will find an inverse of y under f is negligible[1]. Weak one-way functions require only that all efficient algorithms fail with some non-negligible probability.

In order to give an example of OWF candidate let's define a function as $f(p, q) = p \times q$, where p and q are k-bit primes and $*$ is the regular integer multiplication[2]. (So , our domain is the set of pairs of k-bit primes, and $f\colon Pk \times\ Pk \rightarrow X$ where Pk is the set of k-bit primes, and X is set of $2 \times k$-bit numbers). So clearly, f is not a permutation. And let's assume that n=p∗q, so that $f(p, q) = p \times q\ = n$. There's no known polynomial time algorithm A such that $A(n)$ output values p' and q' so $that\ p' \times q'\ = n.$ Of course, we can object this claim offering test all the number from 2 to $\sqrt{n}$. And propose a program that works like the following:

```
For i=2 to $\sqrt{n}$ do
If (i divides n ) then output (i,$\frac{n}{i}$);
```

And we would claim that our program runs in time O($\sqrt{n}$) which is polynomial in temrs of n[2]. However, keep in mind taht the number n inputted in this algorithm is of magnitude roughly 2^{2k} and of size 2k and since $\sqrt{n}\ \approx\ \sqrt{2^{2k}} \approx\ \ 2^{k}$, this algorithm runs in time O(2^{k}) , which is exponential in terms of input size. Thus this algorithm runs in exponential time. And, no algorithm that's polynomial in terms of k is known that can factor n. Therefore, this function f is easy to compute and difficult to invert .

2.2 Definition of Physical Unclonable Function

Instead of using computational complexity of algorithm, we can exploit the physical randomness in nature, such as heterogeneous optical medium, electrical noise, and process variation in silicon manufacturing, to construct unclonable functions[2]. A physical unclonable function has common properties as one way function but differently it's implemented in a physical device.

Here is a definition of physical unclonable functions based on the definition of one-way functions. The term challenge refers to the input to the functions and response refers to the output[1].

A physical Random Function is the function embodied by a physical device, and maps challenges to responses. A physical unclonable function satisfies the following properties:

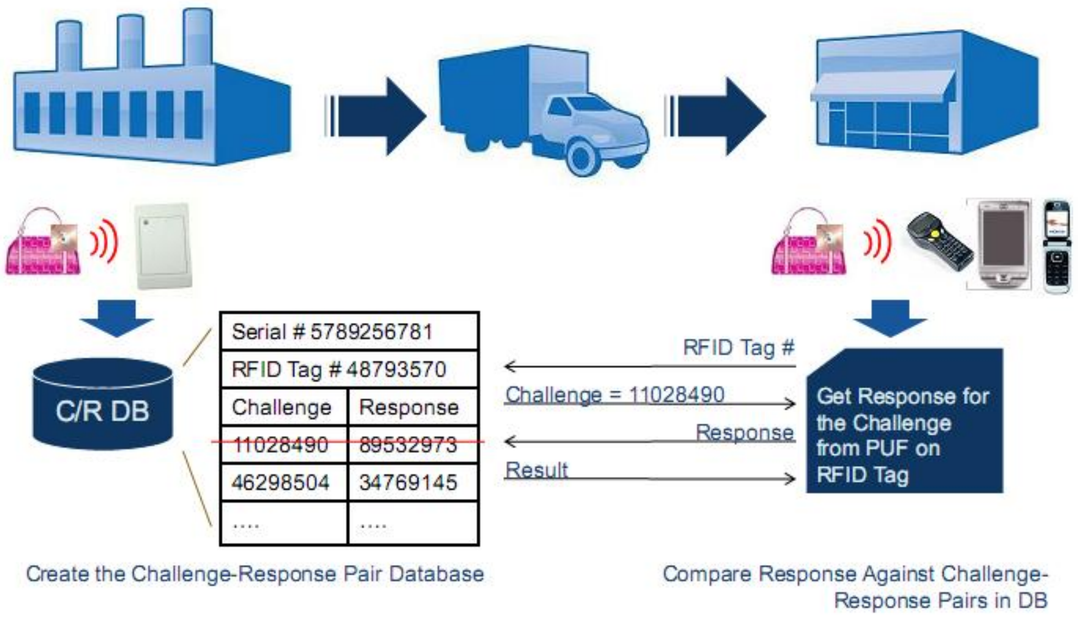
- Easy to evaluate: *The physical device can easily evaluate the function in a short period.*
- Hard to predict: *From a polynomial number of physical measurements (in particular, determination of chosen challenge – response pairs(CRPs)), an adversary who no longer has the device and can only use a polynomial amount of resources (time, matter) can extract only a negligible amount of information about the response to a randomly chosen challenge*.

By the tern “easy”, we mean that the function can be computed in polynomial time.

2.3 PUF Based RFID Authentication

PUFs are tiny electrical circuit primitives that exploit the unavoidable IC fabrication process variations to generate unlimited number of unique, unpredictable, though reliable "secrets" from each chip. These secrets are dynamically generated, using a challenge response scheme. A PUF is queried with a challenge vector (input vector) - a random 64-bit (or longer) number. It almost instantly generates a unique response vector(output) - a 64-bit (or longer) number.

In the figure 2.1, an application of PUF based RFID authentication is given[24]. As shown in the figure below, a set of challenge response pairs are collected from the chip, and stored in a database. This may usually happen at an initial stage in the life of the chip, perhaps at a secure location. To authenticate the chip at a later time, one of the stored challenges from the database is sent to the chip, the response generated is compared against the one initially recorded in the database. If the two match, the chip is authentic. Since each chip can have multiple challenge response pairs, each challenge response pair is used just once, as a one-time pad. This prevents replay attacks on PUF authentication.



C/R DB
RFID Tag #
Challenge = 11028490
Response
Result
Get Response for the Challenge from PUF on RFID Tag
Create the Challenge-Response Pair Database
Compare Response Against Challenge-Response Pairs in DB

Serial # 5789256781	
RFID Tag # 48793570	
Challenge	Response
~~11028490~~	~~89532973~~
46298504	34769145
....	....

Figure 2.1: PUF based RFID authentication procedure

3. DESCRIPTION AND LINEER MODEL OF PUF CIRCUIT

3.1 General Description

An arbiter-based PUF is a $\{0, 1\}^n \rightarrow \{0, 1\}$ mapping, that takes a n-bit challenge (a) and produces a single bit output (r). The basic idea of a PUF circuit is to create a race between two signals which originate challenge (C) as an input and produces a single bit output (R). The arbiter based PUF circuit consists of n consecutive MUX blocks. Each MUX block consist of two MUXs and has two input and two output bits and a control or select bit. If the control bit of a MUX block is logical 0, the two inputs are directly passed to the outputs through a straight path. However, if the control bit is set to logical 1, the two input signals are switched before being passed to the outputs of the MUX block. Based on the control bit of the MUX block, each of the two input signals will take one of two possible paths. As can be seen from Figure 3.1, there are n MUX blocks where the output of each block is connected to the input of the following block. After the last block, the two output signals are connected to an D Flip-flop. The two inputs of the first block are connected to each other, and the connection is sourced by a pulse generator.

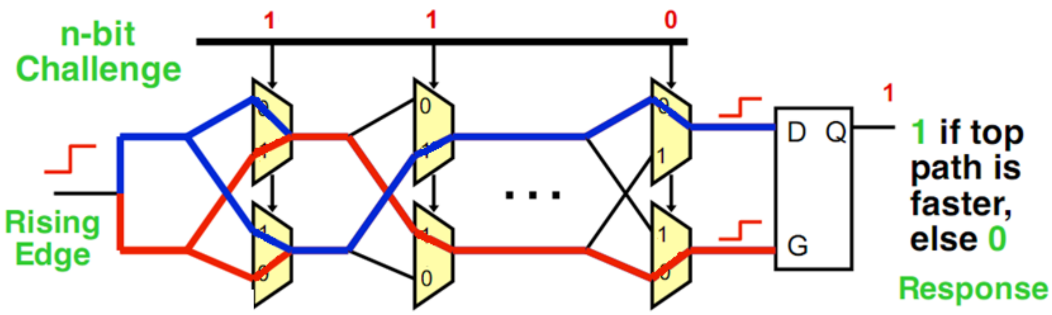


Figure 3.1: Arbiter-based PUF circuit

Afterwards, a pulse is generated and fed into the inputs of the first block. Since the inputs of the first block are connected, the pulses traveling through each of the two paths(red and blue lines) are expected to be simultaneous. Although these two paths have been supposed to be perfectly symmetrical, manufacturing variations of these paths will cause a small mismatch. As the two pulses pass through the consecutive MUXs, they will start acquiring a time delay. The arbiter at the end of the delay paths determines which rising edge arrives first and sets its output to 0 or 1 depending on which of the pulses comming first .

3.2 Components of PUF Circuit

As it’s seen in the Figure3.1 the main components of PUF circuit are MUX blocks and D-type Flip Flop. Due to their functions they can be representated as a switch block and an arbiter. In the next chapters MUX blocks and switch blocks are used interchangeably. The same is valid for a D-type Flip- flop and an arbiter component.

So, switches as MUX block have 2 inputs, 2 outputs and control or select pins. Applying logic “0” to control pin provide direct connection between input and output which means the signal stay on the same path. On the contrary, if control signal is logic “1” the connection are cross coupled which means the input signals interchange their path. The described operation of switch block component are shown in the Figure 3.2.

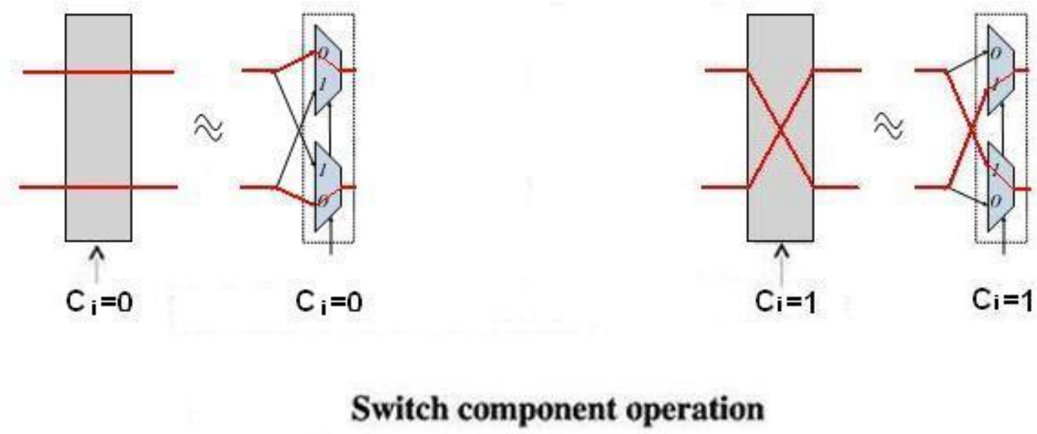


Figure 3.2: Operation of MUX blocks or switch block components

A positive edge-triggered D-type flip-flop or arbiter has two input and 1 output pins as it shown in the Figure 3.3. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D input. This logic states are hold until the next transition of clock input.

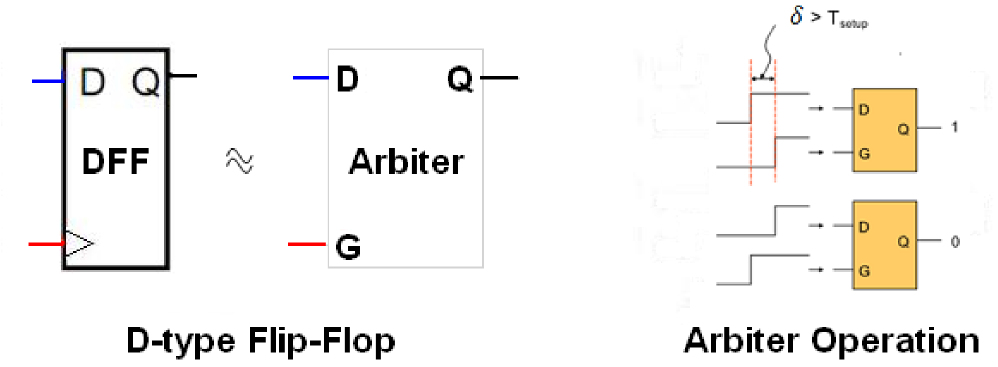


Figure 3.3: Operation of a D-type Flip-Flop or an arbiter

Flip – flop have setup and hold time that must be satisfied[25]:

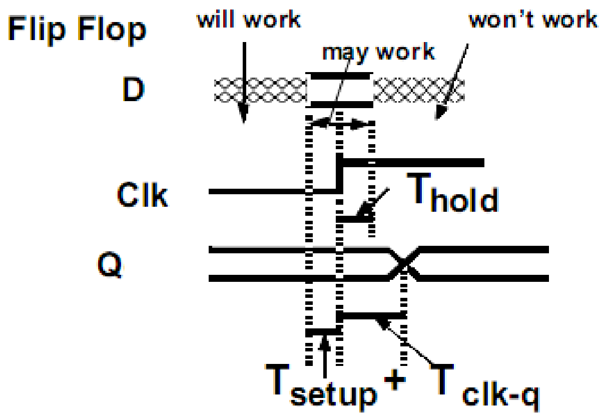


Figure 3.4: Setup and Hold times in a Flip-Flop

If D will arrives before setup time and is stable after the hold time Flip-Flop will work. FF will slow the signal by the setup and “clk to Q” delay in the worst case. In a PUF circuit, if the delay difference between signals in G and D pins is more than setup time, T_{setup} , the output will be logic ”1” ,otherwise the output will be logic ”0”.

3.3 Delay Paths in the PUF Circuit

3.3.1 Switch delay

CMOS non-linear model let us define the total(switch) delay of a logic gate i.e the delay between input of the first gate and the input of the next gate as shown in the Figure 3.5. This model propose us to divide the switch delay in two components as cell delay and connect delay which is expressed below by the formula (3.1)

$$D_{switch} = D_{cell} + D_{connect} \tag{3.1}$$

The D_{cell} delay contributed by the MUX gate itself, is typically defined as the 50 percent input pin voltage to 50 percent output voltage[26]. Cell delay is usually a function of both output loading and input transition time. The connect delay $D_{connect}$ of an element is the time it takes the voltage at an input pin to charge after the driving output pin has made a transition. In brief, it's the time neccesary for a waveform to travel along a wire. The connect delay are the function of wire capacitance, pin capacitance and wire resistance. The wire capacitance and wire resistance are related with wire length.

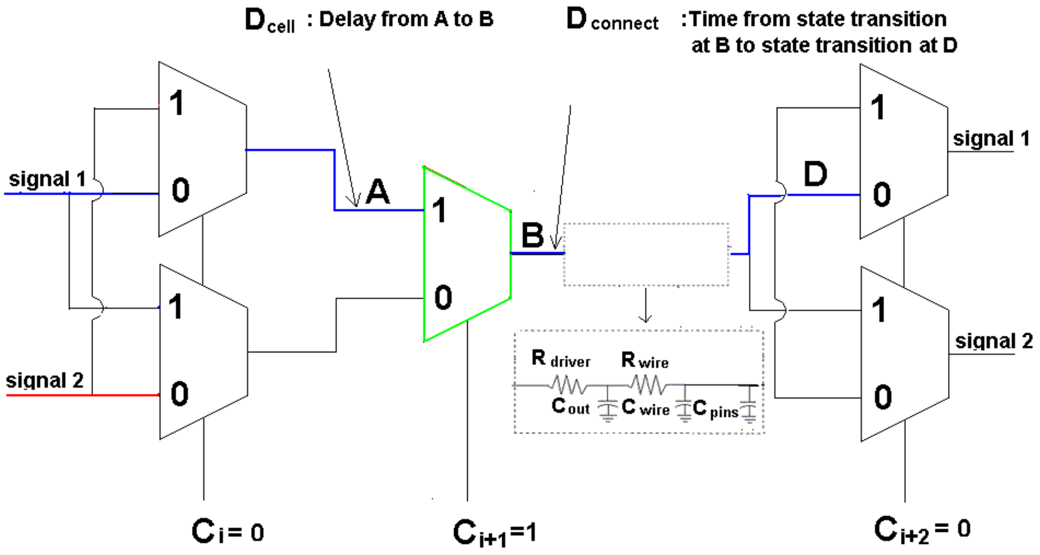
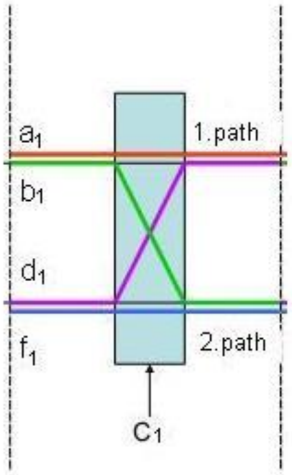


Figure 3.5: Cell and Connect delays in a PUF circuit

3.3.2 Delay path configurations

Unique delay paths consist of unique delay of MUXs. Different challenges will impose different paths on the propagating pulses. In order to see different configurations of delay path in PUF circuit we can consider a circuit that consist of two switch components without an arbiter part. In this example we assume that the delay behavior's of circuit obeys additive delay model[10].



When $C_1=0$ 1.path delay is a_1

2. path delay is f_1

When $C_1=1$ 1.path delay is d_1

2. path delay is b_1

Figure 3.6:. Delays in a switch block

We label the two paths which the upper signal can take as a_i and b_i, and for the lower signal as d_i and f_i. Paths a_1 and $f_1$are chosen when the challenge bit c_1 is 0, whereas b_1 and d_1 are chosen when the challenge bit is 1.

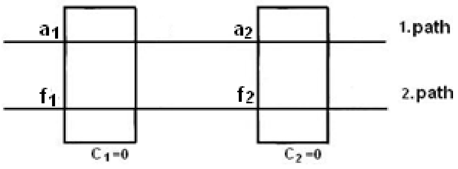


Figure 3.7: Configured delay paths with $C_1=0$ and $C_2=0$ control inputs

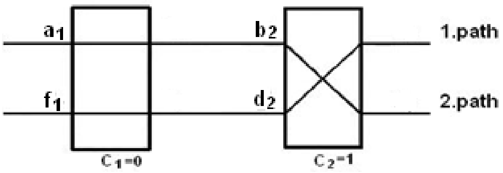


Figure 3.8: Configured delay paths with C_1=0 and C_2=1 control inputs

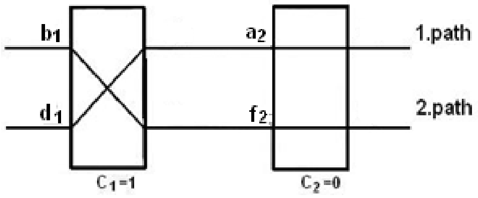


Figure 3.9: Configured delay paths with C_1=0 and C_2=0 control inputs

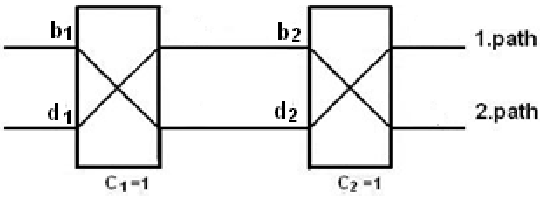


Figure 3.10: Configured delay paths with C_1=1 and C_2=1 control inputs

a_i, f_i is the delays of a signal gained at the end of direct passing through switch to the input of next switch and b_i, d_i is the delays of a signal gained at the end of cross passing through switch to the input of the next switch .

- For 1. path all set of total delay will be

$$m_{11}(b)= a_1 + a_2 \quad (C_1=0, C_2=0) \tag{3.1}$$

$$m_{12}(b)= f_1 + d_2 \quad (C_1=0, C_2=1) \tag{3.2}$$

$$m_{13}(b)= b_1 + d_2 \quad (C_1=1, C_2=1) \tag{3.3}$$

$$m_{14}(b)= b_1 + f_2 \quad (C_1=1, C_2=0) \tag{3.4}$$

- For 2. path all set of total delay will be

$$m_{21}(b)= f_1 + f_2 \quad (C_1=0, C_2=0) \tag{3.5}$$

$$m_{22}(b)= a_1 + b_2 \quad (C_1=0, C_2=1) \tag{3.6}$$

$$m_{23}(b)= d_1 + b_2 \quad (C_1=1, C_2=1) \tag{3.7}$$

$$m_{24}(b)= b_1 + f_2 \quad (C_1=1, C_2=0) \tag{3.8}$$

If we set $C_1 = 0$ and $C_1 = 1$, the delay of path 1 is $m_{1,i}$ $=f_1 + d_2$ and $m_{2,i}$ $=a_1 + b_2$. So, if the delay difference, δ, between path 2 and path 1 is greater than T_{setup} the response will be logic “1” , otherwise the response will be logic “0”as depicted in the Figure 3.10 . As we see for n=2 challenges we get $2x2^2$ different configurations or equations for 1 and 2 path. So if we generalize n challenge bits lead to 2^n equations for each path with 4n unknow delay variables.

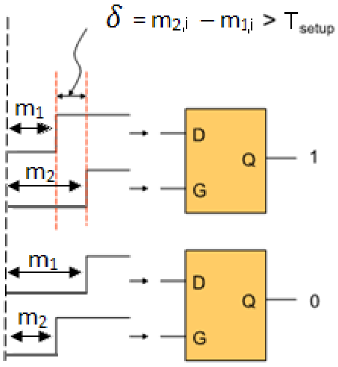


Figure 3.11: Arbiter operation in the example with m_1 and m_2 delay paths

3.4 Symmetrical Structure of PUF circuit

Analyzing the PUF circuit, which is shown in the Figure 3.1, and path delays, which are described in previous section, allow us to propose more general structure of PUF as shown in the Figure 3.11, below:

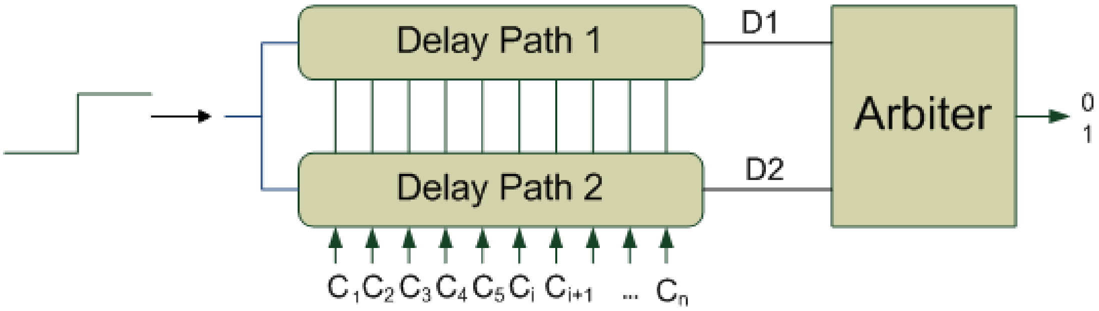


Figure 3.12: More general structure of arbiter based PUF

The delay pathes are configured by using n-bit challenge vectors. These challenges select different routing for signals. The signal travelling on this route passing through different switches or MUXes. So, the total delay of signal1 and signal2 at the end of delay paths is sum expression where each entry belongs to unique combination of unique delays.   In order to find more general expression for the total delay path 1 and 2 we adds up all the switch delays that signal passing through. After that we separate all switch delays to cell and connect delays components. All this operation are reflected in the formula 3.9 and 3.10. The arbiter gives the output according to measured differences between these delay paths. Formula 3.11 gives the general expression for these difference.

$$D_1=\sum_{i=1}^{i=n} D^{(1)}_{switch}(C_i) = \sum_{i=1}^{i=n} D^{(1)}_{cell}\,(C_i) + D^{(1)}_{connect}(C_i) \qquad \textbf{(3.9)}$$

$$D_2=\sum_{i=1}^{i=n} D^{(2)}_{switch}(C_i) = \sum_{i=1}^{i=n} D^{(2)}_{cell}\,(C_i) + D^{(2)}_{connect}(C_i) \qquad \textbf{(3.10)}$$

$$\delta = D_2\text{-}D_1=\sum_{i=1}^{i=n}(D^{(2)}_{cell}\,(C_i) - D^{(1)}_{cell}(C_i)) + (\,D^{(2)}_{connect}(C_i) - D^{(1)}_{connect}(C_i)) \qquad \textbf{(3.11)}$$

The last expression show that if difference connect delay components are much more than difference of cell delay component and setup time of D Flip-flop then the response of PUF circuit are biased to logic”1” or logic “0”. As a result the circuit become insensitive to process variations in silicon. That is why we need to reduce the contribution of a connect delay. From section 3.3.1 we know that connect delays are mainly the function of wire length. That’s why the only way to do it is to make the structure of PUF symmetric i.e make wire length connecting the MUXs equal. Figure 3.12 help us to depict our statement

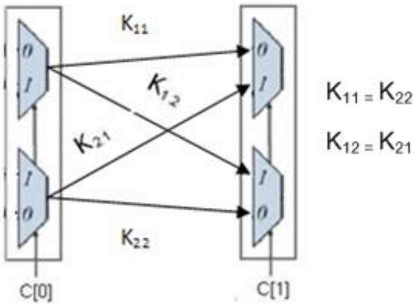


Figure 3.13: The symmetrical wiring between two switch blocks

The symmetrical structure must be maintaned through all PUF circuit in order to produce random responses against random challenge vectors.

3.5 Lineer Model of an Arbiter-based PUF

In this section we derive a linear model for arbiter based PUF [1].

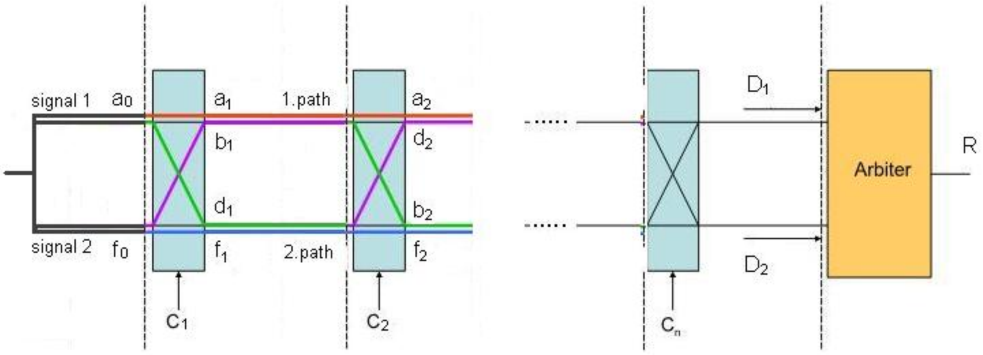


Figure 3.14: PUF circuits represented with switch blocks and switch delays

To compute the total delay, the path of the upper signal is followed from the initial input pulse. The signal will start traveling in a separate path to get to the first switch. Let us label this initial delay as a_0. For the signal going through the lower path we label this delay as f_0. In the first switch the delay of the signal 1 will be ($\bar{c}_1 a_1$ + $c_1 b_1$), where $\bar{c}_1$ is the complement of c_1. The delay in the second switch will depend on whether the signals switched paths in the first stage or not. For signal 1 if it doesn't change path (c_1 =0) the delay at switch2 will be $\bar{c}_1$($\bar{c}_2 a_2$ + $c_2 b_2$). If it changes (c_1 =1) the delay at switch will be c1($\bar{c}_2 f_2$ + $c_2 d_2$) . So, the total delay of signal 1 at switch 2 will be $\bar{c}_1$($\bar{c}_2 a_2$ + $c_2 b_2$) + c_1($\bar{c}_2 f_2$ + $c_2 d_2$) . Let's specify a new variable x_i which represents the parity of the first i−1 challenge bits, and will signify if the signal starting at the upper path stays in that path or moves to the lower path after i − 1 switches. The expression for x_i is

$$x_i = c_1 \oplus c_2 \oplus \ldots \oplus c_{i-1} \qquad \textbf{(3.12)}$$

So the delay of i th switch can be denoted as $\bar{x}_i$($\bar{c}_i a_i$ + $c_i b_i$) +x_i($\bar{c}_i f_i$ + $c_i d_i$)

. The total delay in the pulse of signal 1 is

$$D_1 = a_0 + \sum_{i=1}^{n} \bar{x}_i\,(\bar{c}_i a_i +\ c_i b_i) +\ x_i(\bar{c}_i f_i +\ c_i d_i) \qquad \textbf{(3.13)}$$

Similarly the delay for the signal 2 initiated in lower path can be derived to be equal to

$$D_2 = f_0 + \sum_{i=1}^{n} \bar{x}_i\,(\bar{c}_i f_i + c_i d_i) + x_i(\bar{c}_i a_i + c_i b_i) \qquad \textbf{(3.14)}$$

The difference between these two delays δ is the main variable for our model. This difference will decide whether the output of the PUF is 0 or 1. Since we do not know which of the two signals will end up in the upper path, we will need to incorporate the parity of all the challenge bits which we label P. The difference between the two delays becomes

$$\delta = (-1)^P(D_1 - D_2) = (-1)^P(\sum_{i=1}^{n}(\overline{x_i} - x_i)\left(\overline{c_i}(a_i - f_i) + c_i(b_i - d_i)\right) +$$

$$(a_0 - f_0)) \qquad \textbf{(3.15)}$$

$$\delta = (\sum_{i=1}^{n}(-1)^{P \oplus x_i}\left(\overline{c_i}(a_i - f_i) + c_i(b_i - d_i)\right) +$$

$$(-1)^P(a_0 - f_0)) \qquad \textbf{(3.16)}$$

Finally , we define $v_i = \frac{(a_i + d_i) - (b_i + f_i)}{2}$ and $u_i = \frac{(a_i - d_i) - (b_i - f_i)}{2}$ for

i=1...n and $v_0 = (a_0 - f_0)$. We define the parity of the challenge bits from a reverse order as $p_i = P \oplus x_i = c_i \oplus c_{i+1} \oplus \dots \oplus c_n$. The delay equation becomes:

$$\delta = (\sum_{i=1}^{n}(-1)^{p_i}(u_i + (-1)^{c_i} v_i) + (-1)^P v_0) = (-1)^{p_i} u_i + (-1)^{p_i \oplus c_i} v_i +$$

$$(-1)^P v_0 \qquad \textbf{(3.17)}$$

Since connection delays in direct and cross wiring between two following switch boxes ideally suppose to be equal, due to symmetry, we can conclude that the defined variable u_i must be close to zero. Extracted expressions for u_i and v_i variable are given in appendix A. So, in the above expression for δ(delay difference) we can omit the parameter u_i . Note that p_1=P, and that $p_i \oplus c_i = c_{i+1}$. After defining $y_i = v_{i-1}$ the final delay equation becomes

$$\delta = \sum_{i=1}^{n}(-1)^{p_i}(y_i + y_{n+1}) \qquad \textbf{(3.18)}$$

Equation 1 uses only n+1 rather than 2n+1 variables to desribe the delay between the upper and lower signal paths. It’s important to note that the delay variation y_i will depend on the fabrication process of the PUF circuit. Therefore, one would expect these variable to follow a normal distribution. Without loss of generality, we can normalize these values and assume they belong to a normal distribution of mean 0 and variance 1. We invoke the arbiter condition for the response bit R. We have

$$\delta > T_s \rightarrow R=1 \tag{3.19}$$

$$\delta < T_s \rightarrow R=0 \tag{3.20}$$

Where T_s is the setup time for the arbiter.. Finally , we can use Equation to write response equation.

$$(-1)^R \sum_{i=1}^{n} (-1)^{p_i} (y_i + y_{n+1}) < 0 \tag{3.21}$$

Equation (3.21) is an inequality relating the challenge vector C which consist of n input bits C_i to the output bit R. This inequality has n+1 variables which characterize the PUF circuit. So, for a single PUF, we may form the following linear equation:

$$\delta_j = (-1)^{p_1^{(j)}} y_1 + \dots + (-1)^{p_2^{(j)}} y_2 + \dots + (-1)^{p_n^{(j)}} y_n + y_{n+1} \tag{3.22}$$

Using Equation 2 we may write the following linear inequalitiy

$$(-1)^{R^{(j)}} [(-1)^{p_1^{(j)}} \quad (-1)^{p_2^{(j)}} \dots (-1)^{p_n^{(j)}} \quad 1]\, Y < 0 \tag{3.23}$$

where $Y = [y_1 \; y_2 \dots y_{n+1}]$

4. IMPLEMENTATION OF A PHYSICAL UNCLONABLE FUNCTION ON AN FPGA

4.1 Introduction to a FPGA

For implementation of PUF circuit we preferred FPGA Virtex2Pro designed and produced by Xilinx Corporation. FPGA devices are preffered to custom IC because of some advantages such as flexibility in reprogrammability and short amount of time for implementation of circuits [11].

FPGAs contain programmable logic components called "configurable logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory[12].

CLB resources include four slices and two 3-state buffers. Each slice is equivalent and contains:

- Two function generators (F&G)
- Two storage elements
- Arithmetic logic gates
- Large multiplexers
- Fast carry look-ahead chain
- Horizontal cascade chain (OR gates)

The function generators F & G are configurable as 4-input look-up tables (LUTs), as 16 bit shift registers, or as 16-bit distributed SelectRAM+ memory. In addition , the two storage elements are either edge-triggered D type flip flops or level-sensitive latches. Each CLB has fast internal interconnect and connects to a switch matrix to access general routing resources [12].

4.2 Implementation of PUF components

Xilinx ISE 9.2 is the one of a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimulus, and configure the target device with the programmer. In order to synthesize a component and implement it on FPGA we should know the VHDL design flow and understand VHDL codes [13] Any primitive logic are implemented in FPGA by using function generators or look-up tables(LUTs)[12].

4.2.1 Switch implemented with MUX

The structure of this switch unit is shown on the Figure 4.1. Each switch unit includes two MUX elements. So this structure enable us to connect directly or cross connect inputs “dI0” and “dI1” to outputs “dQ0” and “dQ1” depending on a challenge input bit c_i .

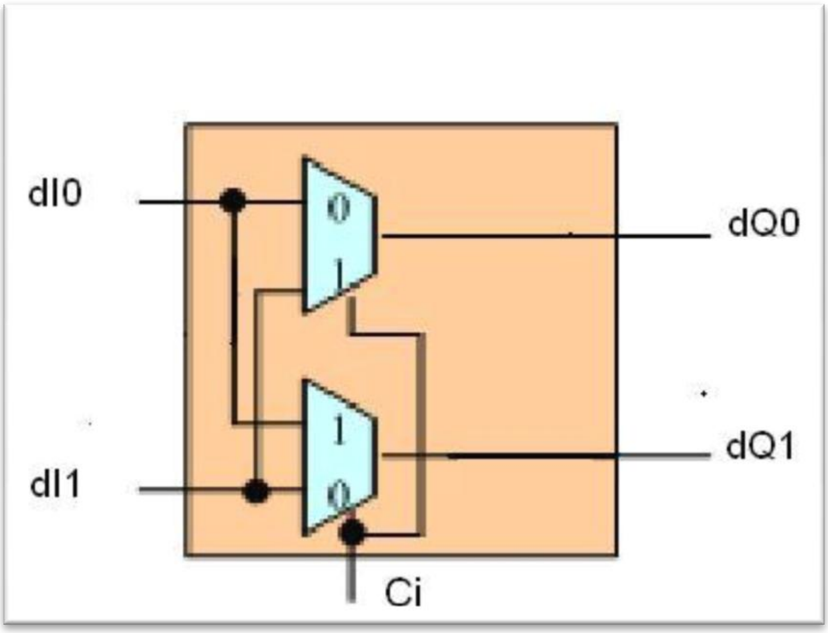


Figure 4.1: A Switch block implemented with a MUX

For synthesis of MUX switch we need to define the entity with three inputs (one of them is challenge bit) and two outputs. Inside of the entity we describe the MUX unit and how they connected. In the architecture part of VHDL codes we define the loop statement "$for\ \ i\ in\ 1\ to\ n\ generate$" ,which is necessary to obtain a chain of serially connected MUX switch[15] . n spesifies the number of switches and the size of input vector of challenge bits. The VHDL codes for implementation of switch blocks on FPGA are given in the CD, which is attached with the thesis. As a result of synthesizing this code we get RTL schematic as shown on Figure 4.2.

A Synthesizer integrated in ISE tool always try to optimize the logic. There are usually many ways to implement logic with a given functionality. If the synthesis/mapping/place & route tools recognize that certain blocks do not fulfil a timing requirement, these blocks may be optimized in terms of placement and logic design, possibly at the cost of an increased area or slow speed. That is why as the result of synthesis the MUX gate are simplified and the routing between gates are not as shown in the figure 3.1. To avoid it we must change the constraints of synthesizer as it shown on the figures 4.7, 4.8.

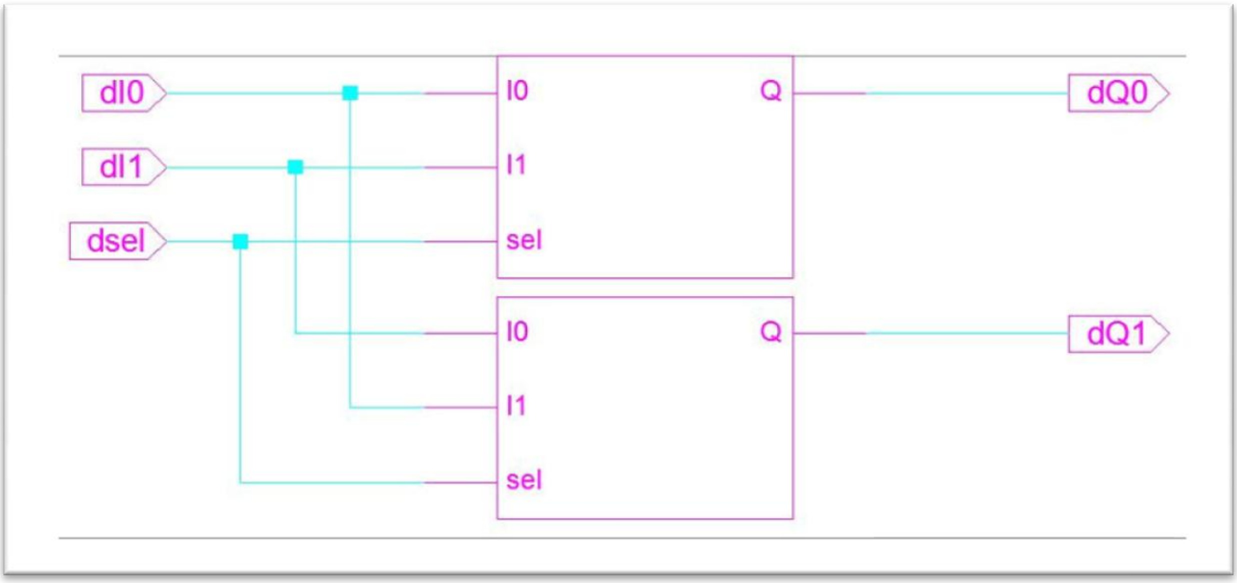


Figure 4.2 : RTL schematic of a switch implemented with a MUX

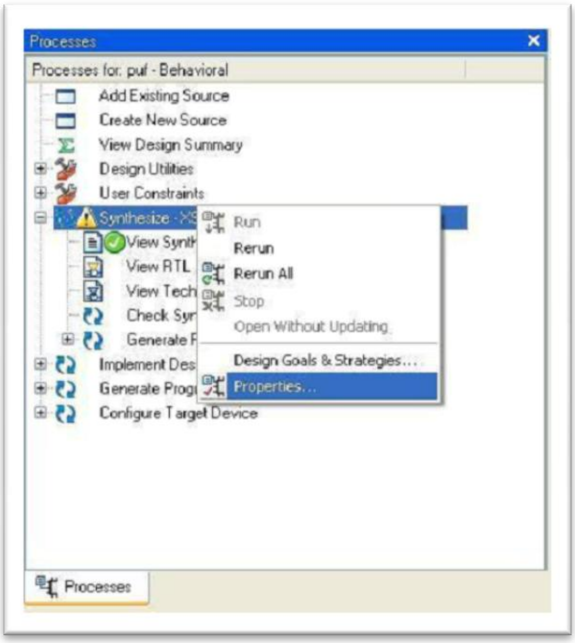
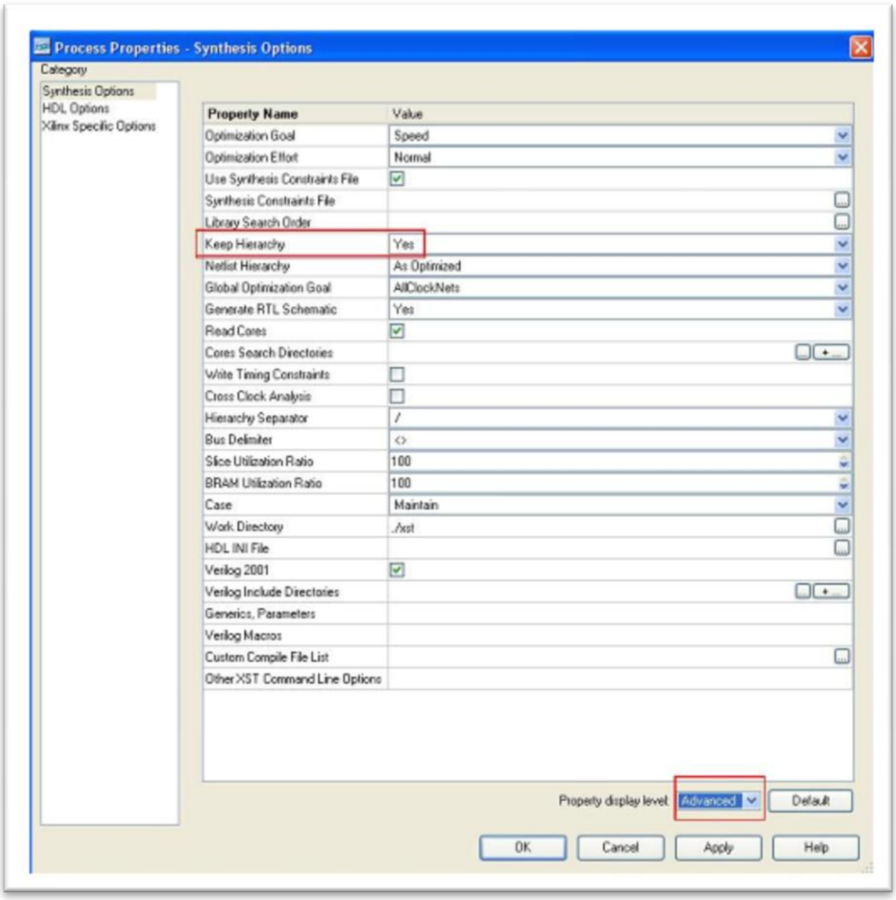


Figure 4.7: Access the “Property” option of a synthesize process



Property Name	Value
Optimization Goal	Speed
Optimization Effort	Normal
Use Synthesis Constraints File	☑
Synthesis Constraints File	
Library Search Order	
Keep Hierarchy	Yes
Netlist Hierarchy	As Optimized
Global Optimization Goal	AllClockNets
Generate RTL Schematic	Yes
Read Cores	☑
Cores Search Directories	
Write Timing Constraints	☐
Cross Clock Analysis	☐
Hierarchy Separator	/
Bus Delimiter	<>
Slice Utilization Ratio	100
BRAM Utilization Ratio	100
Case	Maintain
Work Directory	./xst
HDL INI File	
Verilog 2001	☑
Verilog Include Directories	
Generics, Parameters	
Verilog Macros	
Custom Compile File List	
Other XST Command Line Options	

Property display level: Advanced

Figure 4.8: Setting “Keep Hierarchy” parameter to “YES” value

4.2.2 Arbiter

For an arbiter we use pozitive edge triggerred D type Flip Flop. Unwilling wiring of clock input of flip flop to an internal system clock signal done by synthesizer make us to implement own D type flip flop. The schematic of the Flip flop are shown on the figure 3.10 [15]

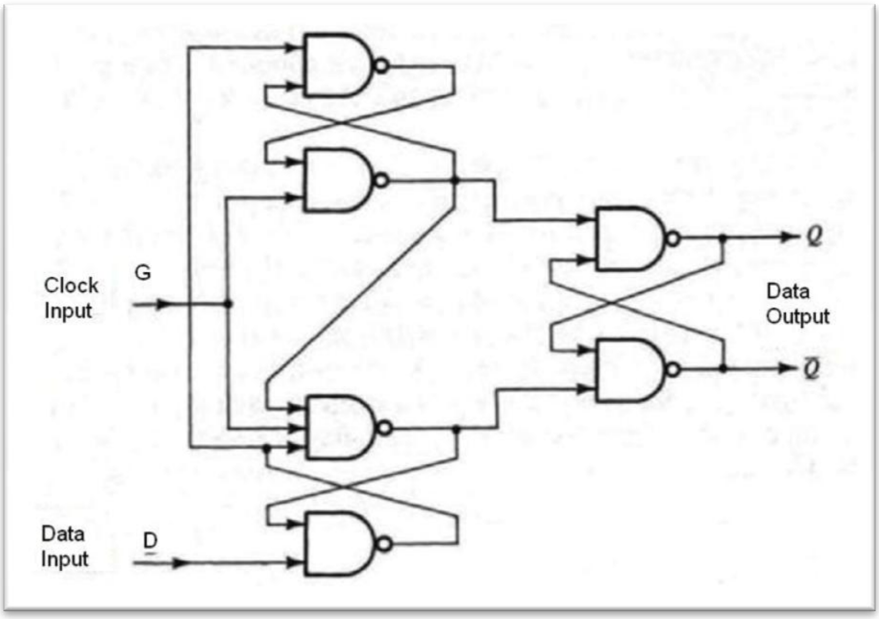


Figure 4.10: The D type edge-triggered flip flop

The operation of D Flip- flop are illustrated in the Figure 3.3.

As the rising edge of D comes earlier than that of G, an ideal arbiter output must be 1. However, if the time differnece between two signals is less than the setup time of the Flip-flop , an output remains at ‘0’ instead of being switched to ‘1’. This property introduces an skew factor or imbalance in ratio of numbers of 1 and 0 at the output. The desiring probability of comming ‘1’ or ‘0’ at the output must be the same and equal to 0,5. Frequently most of flip flops favor the path to output ‘0’. So, to compensate for the skews we fix some of the most significant challenge to effectively lengthen the delay path connected to a gate input[2]. Also, in the graduation project[15] the number of switch blocks or challenge bits, n, giving almost equal rate of logic ‘1’ and logic’0’ in response, has been determined experimentally. In this experiment, the responses has been measured against randomly generated challenge vectors. Implementing PUF with n= 64 switch blocks almost solve our skew problem. In our experiments, with 64 switch blocks we obtain the rate as 61 %.

The two signals emitted from the source at the same time after passing through switch blocks arrives to data and clock input of the arbiter. . For implementation of the flip-flop shown in figure 4.10 we need a primitive NAND gates. That can be accomplished by configuration of LUT as result of synthesizing the next code:

```
LUT2_inst : LUT2
generic map (
INIT => X"7")
port map (  O => O,   -- LUT general output
I0 => I0, -- LUT input
I1 => I1  -- LUT input
);
```

4.3 Placement of Switch Box

In order to maximize the inter-chip variation, the delay path must be placed and routed as symmetrically as possible so as to minimize the delay differences between two paths[1]. Since Xilinx ISE places the gates automatically not considering the wire length or wire delay, the implemented PUF always gives response that biased to logic ‘1’ or logic ’0’. So symmetrical placement of switch box minimizing the wire delay shown on the figure 4.12 For that placement we should manually drag and place the MUXs in the according slices that contains this logic gates in Xilinx Floorplanner tool. Having done this placement the tool automatically write this constraint into “ucf” file,which is designated for defining pozition, time, connection constraints. In this file must be inserted the constraints that fix upper and lower MUXs. In order to keep them close one of them are placed in F LUT the other in G LUT in the according slices. For that we must enter the following codes in “ucf” file shown on the figure 4.13:

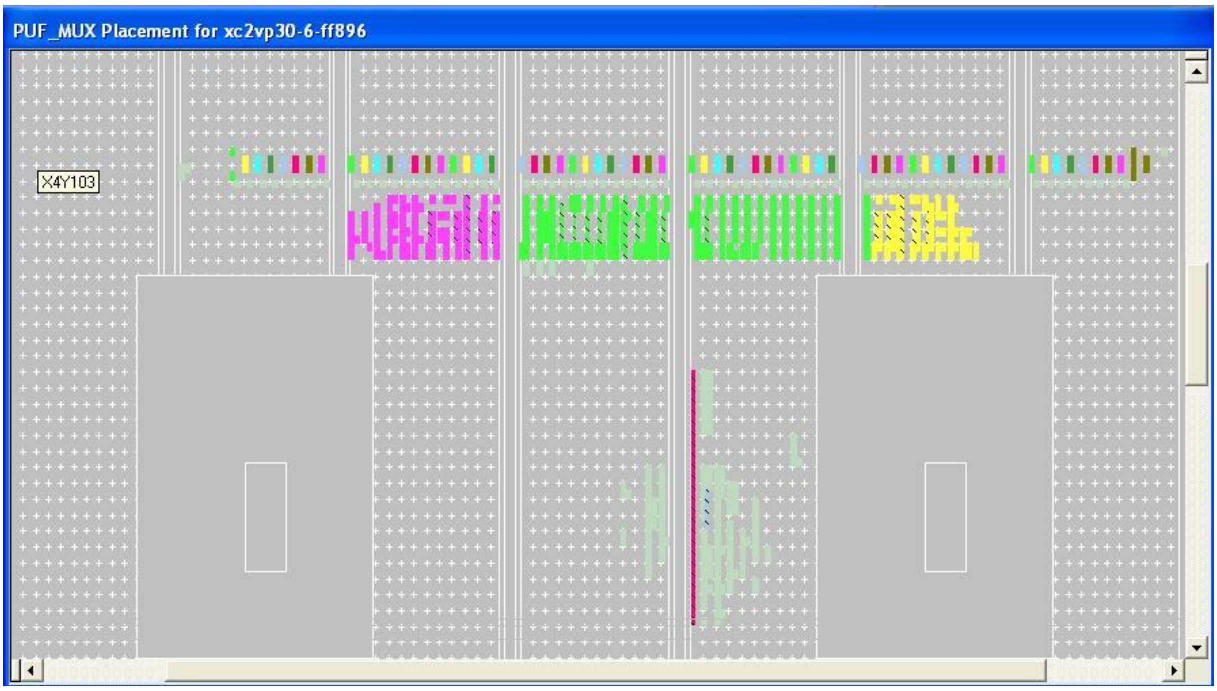
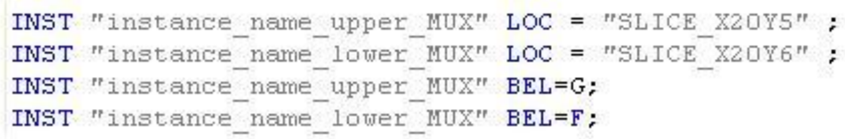


Figure 4.12: The placement of MUX switches in Floorplanner



```
INST "instance_name_upper_MUX" LOC = "SLICE_X20Y5" ;
INST "instance_name_lower_MUX" LOC = "SLICE_X20Y6" ;
INST "instance_name_upper_MUX" BEL=G;
INST "instance_name_lower_MUX" BEL=F;
```

Figure 4.13: Constraints in “UCF” file to select LUT F and LUT G for placement of MUXs

5. ANALYSIS AND CHARACTERIZATION OF ARBITER-BASED PUF

In this section we present the primary characteristics of arbiter-based PUF such as inter-chip variation, measurement noise, and environmental variations, which have an important effects on identification process between different PUFs.

5.1 Inter-chip Variation

5.1.1 Information –bearing challenges

Information –bearing challenges can be defined as the challenges whose responses in a number of different PUFs are not equal. In order to identify a large amount of PUF on FPGA we need considerable numbers of information-bearing challenges. In an experiment where we applied 2000 randomly chosen challenges to 37 chips we could define the ratio of information bearing challenges. Due to the lack of so many FPGAs in our laboratory I decided to make experiments on available two FPGA chips but with different positions of PUF circuit across the FPGA chip wafers. The results of measurements done in another graduation project [2] approve our decision. The results obtained there shows that the variations across different wafers is similar to the variations across one wafer.

Putting the obtained data in matrix form enable us to facilitate some manipulations over data. The entries $R_{i,j}$ in this matrix are the PUF function $F^{(j)}$ (corresponding to j FPGA chip) which takes $C^{(i)}$ as an input vector.

For each challenges, we have calculated the probability of a response being 1 as follows, in other words we count the occurence of 1 across a row and divide it by the number of tested FPGA chips or j.

Table 5.1: Table form of PUF responses, indexed with j , against randomly chosen i challenges

	Response of each PUF, $F^{(j)}$				
Challenges, $C^{(i)}$	$F^{(1)}$	$F^{(2)}$	$F^{(3)}$	…	$F^{(h)}$
$C^{(1)}$	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$	$R_{1,\ldots}$	$R_{1,h}$
$C^{(2)}$	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$	$R_{2,\ldots}$	$R_{2,h}$
…	$R_{i,1}$	$R_{i,2}$	$R_{i,3}$	$R_{i,j}$	$R_{i,h}$
$C^{(v)}$	$R_{v,1}$	$R_{v,2}$	$R_{v,3}$	$R_{v,j}$	$R_{v,h}$

$$p_i = Pr\left(R_{i,j}\left(C^{(i)}\right) = 1\right) = \frac{k}{19} \qquad \textbf{(5.1)}$$

where k is the number of PUFs that output 1 against each challenge vector $C^{(i)}$, $i \in \{1,2,\ldots,v\}$ and $j \in \{1,2,\ldots,g\}$. Figure 5.1 shows the density function of the random variable p_i for 2000 challenges. When p_i=0 or p_i=1 , the challenge does not generate any information since all PUF response are equal. Except for the cases when p_i=0, p_i=1 , more than 84 % of the total challenges are information-bearing challenges.

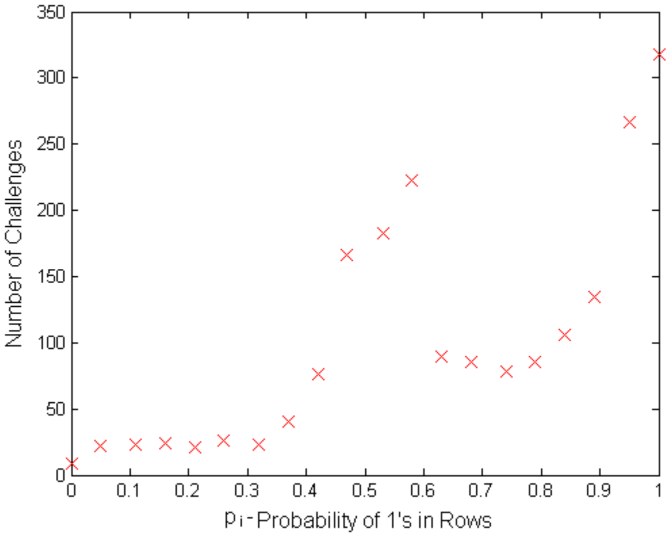


Figure 5.1: The density function of the random variable $p_i = Pr\left(R_{i,j}(C_i) = 1\right)$

The changes in responses of PUFs across a row in the table[2]are resulted from differences in a process variation. The changes in responses of PUFs across a column are arised from different challenges.

5.1.2 Definition and evaluation of inter-chip variation

Here we define the inter-chip variation between two different PUFs as below . For two different PUF responses $R_{i,m}(C_i)$ and $R_{i,p}(C_i)$ to a challenge C_i we can define a function that compare the responses which are the inputs to this function . This function behaves like XOR logic function.

$$D_{m,p} = XOR(R_{i,m}(C^{(i)}), R_{i,p}(C^{(i)})) = R_{i,m}(C^{(i)}) \oplus R_{i,p}(C^{(i)} \quad \textbf{(5.2)}$$

where m ∈ $\{1,2 \dots h\}$, p ∈ $\{1,2 \dots h\}$, i ∈ $\{1,2 \dots v\}$

Simply by this formula we estimate the Hammming distance between responses of two different FPGAs.

Inter-chip variation is one of the basic functions tha make the PUF ouputs unique. It is very important for security of PUF based identification. If the PUF produces uniformly distributed independent random bits, the inter-chip variation should be 50 % on average[2].

For a random challenge set C, we define the inter-chip variation $y_{i,j}$ between PUF i and j as

$$y_{m,p} = \frac{1}{|C^{(i)}|} \sum D_{m,p}(C^{(i)}) \quad \textbf{(5.3)}$$

For convenience, we denote the inter-chip variation by (100* $y_{i,j}$)

Let ‘s assume that our responses from different FPGAs are in matrix or table form as it’s shown in the table 1. Finding inter-chip variations between PUF^i snd PUF^j is equvalent to finding the hamming distance between two columns that corresponds to FPGA with i index and FPGA with j index.

We have evaluated $y_{i,j}$ from 91 arbiter-based PUF pairs using a random challenge set C, where |C|=20000. Figure 5.2 shows the density function of 91 evaluated inter-chip variation. Our test-chips have 34% inter-chip variation on average, and the minimum inter-chip variation is 2%. So minimum value of inter-chip is not allowable since it is less than our environmental noise, which will be described in the next section. It means that we can not distinguish any of two different FPGA using this challenges. This PUF based system is vulnerable to impersonation attacks.

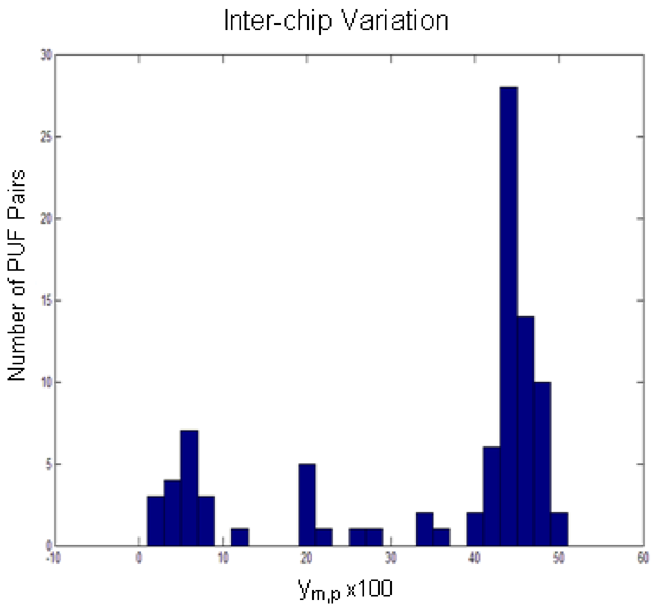


Figure 5.2 : The density of the inter-chip variation $y_{i,j}$ for a number of PUF pairs.

5.2 Environmental Variations

Environmental or intra-chip variation is the property that make outputs of a PUF reproducibile. It is important for reliability in applications based on PUFs. We desire 0 % intra-chip variation[15].

Noise, which may cause unreliability in measured PUF responses are arised from temperature and power supply variation[16].

Temperature or power supply voltage variations can significantly change circuits delay and lead to unreliable responses. Since we exploit relative delay measurement, arbiter-based PUFs are robust to such environmental variations. Figure 5.3 shows the amount of environmental variation introduced by temperature (μ_t) and voltage variation (μ_v) . The reference responses are measured at 27 °C and 1,8 V power supply voltage [2]. In this experiments 10000 challenges are used to estimate environmental variations. Even if the temperature increases more than 40 degrees to 70 °C , $\mu_t \approx$ 4,82 %[2]. Also with ± 2% power supply voltage variation , $\mu_v \approx$ 3,74% .Both μ_t and μ_v are below the average inter-chip variation.

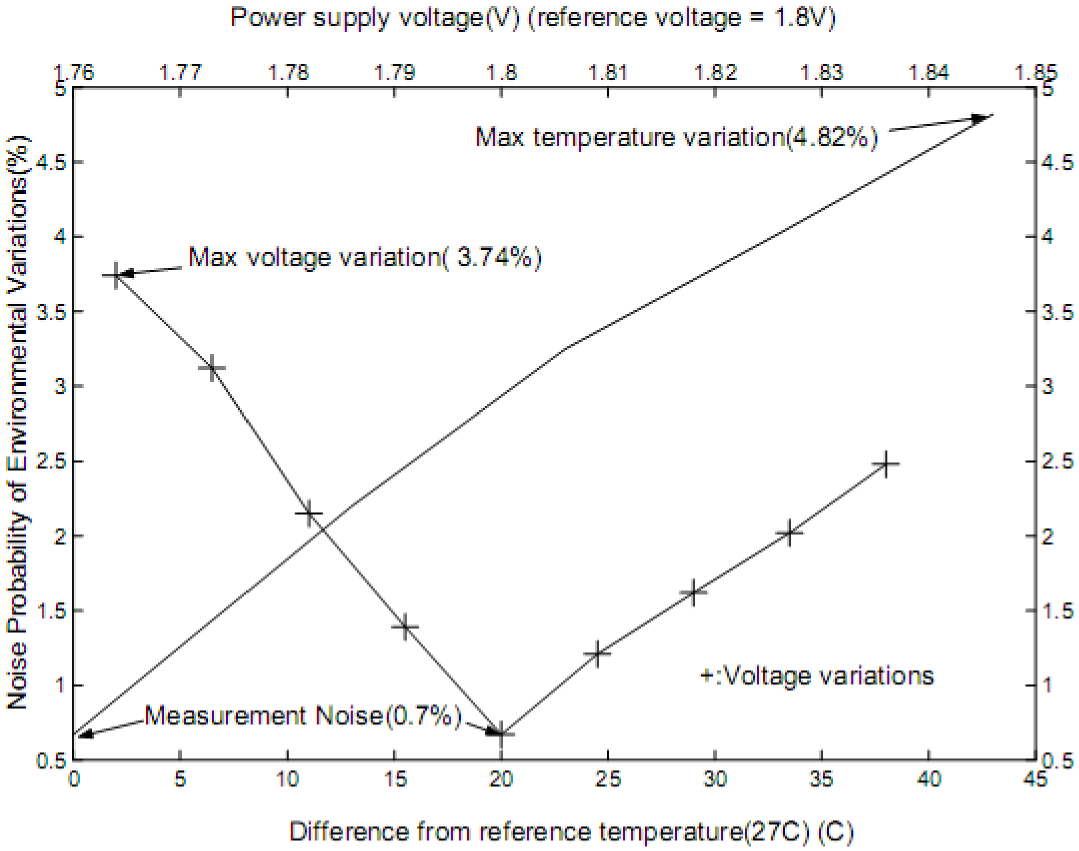


Figure 5.3: The variation of PUF responses due to temperature and power supply changes[2]

5.3 Identification/Authentication Abilites

When we use PUFs for the identification of registered users , there exists the server that stores CRP profile of N registrered PUFs in the database . Each CRP profiles has k CRPs. When a user present a PUF to the server , the server generates CRPs using the presented PUF and compare it with all CRP profiles of the registered users in the database. The server identifies the user by finding the minimum distance CRP profile from generated CRPs of the presented PUF.

In this identification scheme we are trying to find the number k that enable us to identify $N=10^9$ different chips with the negligible error probability. It’s apparent that before finding the sufficient challenge number k we need to calculate the number of information bits or delay stages of PUF to distinguish between 1 billion components. The birthday phenomenon will guide us to the right answer. It ‘s so named because the problem of finding two random challenges which gives the same response value is identical to finding two people in a group of people who share the same birthday with probability greater than a certain threshold. The probability of one collision is expressed in the next inequality [17]

$$P>1-e^{\frac{-n(n-1)}{2m}}$$

where n - the number of evaluation, m-possible outputs ($m=2^l$, l –is the number of information bits)

If we require that the probability of a collision be greater than 0.5, all we have to do is to find that value n in terms of m which makes it happen. We found that n approximately equal to $m^{1/2}$. This solution say that in the set of m possible outputs after n times evaluation the ouput will repeat with the probability more than 0,5. Therefore n is the maximum number of times when we can get different output. Similarly in our problem of finding l information bits to distinguish 1 billion components we need m (the number of all possible challenges) which is equal to n^2 . So the number of all possible challenges is 10^{18} which is aproximately equal to 2^{60} . Thus to distinguish 10^9 chips we need more than 60 information bits.

In the master thesis[1] identification/authentication capability of arbiter-based PUFs based on inter-chip variation ζ and noise probability μ have been studied. Using ζ=0.22 and μ=0.048 the number of sufficient challenges k is calculated as 443 with error probability $p_e < \frac{1}{N} = 10^{-9}$. The conclusion that using less than 450 CRPs we can authenticate 1 billions of PUF with negligible probability[2].

To imagine how many combinations can be created with the responses to 450 challenges where the ratio of 1's in 450 bit output vector is 50 % we solve the combination problem where we try to place 225 numbers of 1 in 450 cells. Our expected expression will be

$$Nc = \frac{450!}{225!\,225!}$$

The maximum number of components Nc that can be distinguished by 450 challenges is approximately equal to 10^{134} . In this expression we don't consider inter-chip variations and environmental variations. Including these factors significantly lower this value.

Using the inter- chip factors and measurement noise will make changes to our identification scheme[17].Suppose that a PUF wants to authenticate itself as Alice. Then she server asks the PUF to generate a list of CRPs corresponding to CRP profile of Alice in server's database. If the PUF is indeed Alice, then each generated response bit differs from the corresponding profile's response bit with probability μ . If the PUF is not who it claims to be, then this probability is equal to the inter-chip variation $\zeta(>\mu)$.

Inter-chip and intra-chip variation(environmental noise) give us new definitions in identification and authentication process which are false pozitive and false negative. False positive is probability that PUF A will be authenticated as PUF B when PUF A produce the same output as PUF B. False negative is probability that a correct PUF will fail to be authenticated when PUF fail to regenerate a consistent output [15].

6. SOFTWARE ATTACKS ON ARBITER-BASED PHYSICAL UNCLONABLE FUNCTION

A software attack is one of the non-invasive attacks. It can be realized using the lineer programming technique or using machine learning algorithm. In the first case an adversary can formulate the response of a PUF as a function of challenges and delay parameters. If the challenge-response model of the PUF circuit is linear, then an adversary can apply a polynomial number of random challenges and monitor the response to estimate circuit delay parameters in linear model. If the model can predict the response of the circuit with error probability lower than the maximum environmental variation, the adversary can impersonate the original PUF with model. So the main goal of an adversary must be to find out all delay parameters because in this case he could calculate response for any input challenges.

Using Simplex method and Matlab program in lineer programming problem we tried to simulate attacks of an adversary. In this attack, Matlab produces random challenges and calculate the responses against each 64 bit challenge. In attack using linear programming approach a certian number of challenge-response pairs or inequalities are used. Simplex algorithm help us to solve for delay parameters. To analyze the success of this attack we increase the number of input CRPs step by step. The result of attacks using CRPs collected from PUF on FPGA show us the consistency between CRPs created by lineer model in Matlab and measured from PUF on FPGA.

The second attack using support vector machine (SVM) classifier, which is one of the machine learning algorithm, show us the possibility to predict the response of challenges in a test set after training the classifier with CRPs in training set. In neural network the feature vectors with 1 and 0 labels are separated by threshold plane.

In the last part of the chapter we mention about causes of failure in linear modeling.

6.1 Prediction and Calculation Tools

In order to make attacks or to find out the all delay parameters of the system having only limited number of CRPs we need calculation tools. Having found all parameters we will able to estimate the response of PUF against each challenge in CRP list of database. This problem can be solved using classification and optimization methods.

We will talk about Simplex algorithm which can be categorized as one of the widely used optimization tools for linear programming model.

The other prediction tool that we will use is based on a machine learning algorithm for classification problem. The classification normally refers to a a procedure that learns to classify new instances based on learning from a training set of instances that have been properly labeled by hand with the correct classes. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. One of the most widely used classifiers is support vector machine (SVMs) classifier [18].

6.1.1 Lineer Programming method

A Linear Programming problem is a special case of a *Mathematical Programming* problem. More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear_equality and linear inequality constraints. Linear programs are problems that can be expressed in canonical form[19]:

$$Maximize\quad f^T x$$

$$object\ to\ Ax \le b$$

where $\boldsymbol{x}$ represents the vector of variables (to be determined), $\boldsymbol{f}$ and $\boldsymbol{b}$ are vectors of (known) coefficients and A is a (known) matrix of coefficients. The expression to be maximized or minimized is called the *objective function* ($\boldsymbol{f^T x}$ in this case). The equations $A\boldsymbol{x} \le \boldsymbol{b}$ are the constraints which specify a convex polytope over which the objective function is to be optimized. Linear programming can be applied to various fields of study. It is used most extensively in business and economics, but can also be utilized for some engineering problems.

Let's consider a simple example of linear programming (LP) problem[19]:

Find numbers x_1 and x_2 that maximize the sum $x_1 + x_2$ subject to the constraints $x_1 \ge 0$, $x_2 \ge 0$ and

$$x_1 + 2x_2 \le 4$$
$$4x_1 + 2x_2 \le 12$$
$$-1x_1 + 1x_2 \le 1$$

In this problem there are two unknowns, and five constraints. The first two constraints $x_1 \ge 0$, $x_2 \ge 0$, are special . These are called nonnegativity constraints and are often found in linear programming problems. The other constraints are then called the main constraints. The function to be maximized (or minimized) is called the objective function. Here, the objective function is $x_1 + x_2$. We can solve this problem by graphing the set of point in the plain that satisfies all the constraints and then finding which point of this set maximizes the value of the objective function. Each inequality constraint is satisfied by a half-plane of points, and the constraint set is the intersection of all the half-planes. In the present example, the constraint set is the five-sided figure shaded in Figure 6.5

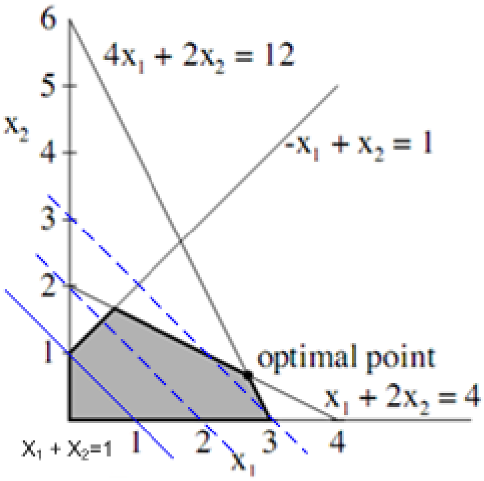


Figure 6.5: Example of solving linear programming problem

We seek the point (x_1, x_2), that achieves the maximum of $x_1 + x_2$ as (x_1, x_2) ranges over this constrain set. Therefore, we seek the line of slope -1 that is farthest from the origin and still touches the constraint set. This occurs at the intersection of the lines $4x_1 + 2x_2 = 12$ and $x_1 + 2x_2 = 4$, namely (x_1, x_2)=(8/3,2/3) . It is easy to see in general that the objective function, being linear, always takes its maximum (or minimum) value at a corner point of the constraint set, provided the constraint set is bounded.

The simplex algorithm, developed by George Dantzig in 1947, solves LP problems by constructing a feasible solution at a corner of the polytope and then walking along a path on the edges of the polytope to corners with non-decreasing values of the objective function until an optimum is reached[20].

For solving LP problems in MATLAB we need Optimization Toolbox including LINPROG routine[21].

6.1.2 Lineer Support Vector Machine

Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Classification is achieved by a linear or nonlinear separating surface in the input space of the dataset. When the feature vectors of the training set are linearly separable by a hyperplane, we can build a linear SVM that uses the structural risk minimization principle to decrease classification errors.

Here we gives a description of linear SVM. Let's suppose that we are given training data D, a set of n points of the form [22]

$$D = \left\{\left(c^{(i)}{}_i, s^{(i)}\right) \middle| c^{(i)} \in R^p, s^{(i)} \in \{-1,1\}\right\}_{i=1}^{n} \tag{6-20}$$

where the $s^{(i)}$ is either 1 or −1, indicating the class to which the point $c^{(i)}$ belongs. Each $c^{(i)}$ is a p-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having $s^{(i)}$ = 1 from those having $s^{(i)}$ = − 1. Any hyperplane can be written as the set of points x satisfying

$$\boldsymbol{w \cdot c + w_0 = 0} \tag{6-21}$$

where · denotes the dot product. The vector $\boldsymbol{w}$ is a normal vector. It's perpendicular to the hyper plane.

We want to choose the **w** and b to maximize the margin, or distance between the parallel hyperplanes that as far apart as possible while still separating data. The hyperplanes can be divided by two equations

$$\boldsymbol{w' \cdot c + w_0 = 1} \tag{6-22}$$

$$\boldsymbol{w' \cdot c + w_0 = -1} \tag{6-23}$$

We find the distance between these two hyper planes is $\boldsymbol{\frac{2}{||w||}}$**.** So we want to minimize **||w||** . In some cases, the dataset can not be classified clearly because of non-linearity at the boundary of each class. Considering this case, our goal is not only to make the distance, as large as possible, but also minimize the number of misclassifications.

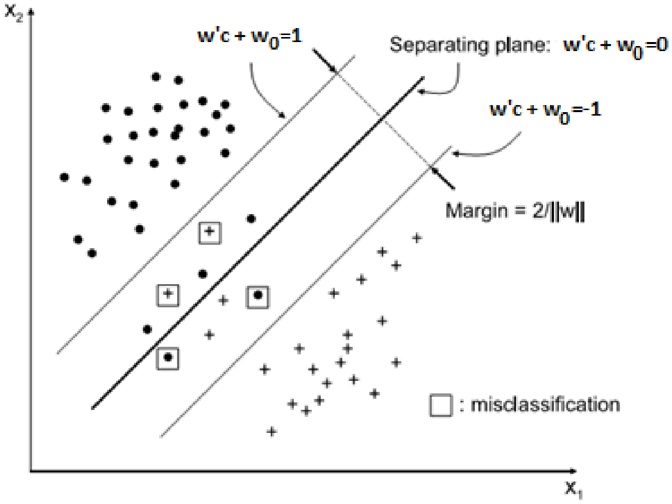


Figure 6.6: Maximum- margin hyperplane separating classes

This is equivalent to minimizing the cost function[23]

$$J(w, w_0, n_e) = \frac{1}{2}\,||w||^2 + \vartheta n_e \qquad \textbf{(6.24)}$$

where n_e is the number of mis-classifications. The parameter ϑ is the positive constant reducing of which give allows more data to lie on the wrong side of hyper plane and would be treated as outliers which give smoother decision boundary[6]. Lagrangian Support Vector Machines(LSVM) provide fast converging algorithm to the minimal point of the cost function [2].

6.2.3 Radial Base Functions Support Vector Machine

There may be another situation where the points are clustered such that the two classes are not linearly separable as shown in the Figure 6.7 . In such cases, one prefers non-linear mapping of data into some higher dimensional space called ‘feature space’, where it is linearly separable. The original space of data points is called ‘input space’. The hyperplane in ‘feature space’ corresponds to a highly non-linear separating surface in the original input space, in the Figure 6.9 [29].

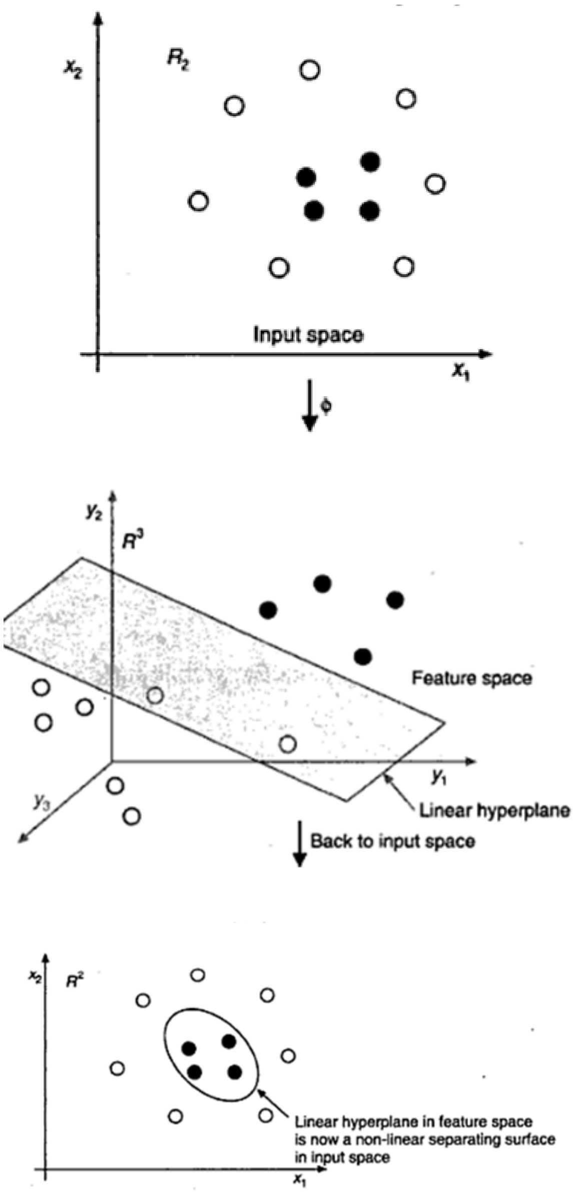


Figure 6.7: Mapping Process

Given a training D set of n data point, $D=\left\{\left(c^{(i)}{}_{i}, s^{(i)}\right)\middle|c^{(i)}\in R^{p}, s^{(i)}\in\{-1,1\}\right\}_{i=1}^{n}$ the support vector method approach aims at constructing a classifier of the form[30]:

$$s(c) = sign[\textstyle\sum_{i=1}^{n} \alpha_i \, s_i \psi\left(c, c^{(i)}\right) + b] \tag{6.25}$$

Where α_i are positive real constant and b is a real constant. $\psi\left(c, c^{(i)}\right)$ is a row of radial basis function and for RBF SVM can be expressed as $\psi\left(c, c^{(i)}\right) = \exp\{-\left\|c - c^{(i)}\right\|^2/\sigma^2\}$. $\psi(.,.)$ is also called kernel function and $\| \, . \, \|$ represents a norm that is generally Euclidean. The known data points $c^{(i)} \in R^p$ i=1,2…n are the center of radial basis functions.

Expensive calculation of dot products in a high-dimensional space can be avoided by introducing a kernel function.

6.2 Attacks on PUF Circuit Using Linear Programming Aproach

6.2.1 Experiments using data generated by Matlab

In order to test our derived mathematical model for PUF implemented with MUXs , we ran a number of tests using MATLAB and the function linprog(f,A,b) included in optimization tool of Matlab [4]. Let's formulate the lineer programming problem one more:

$$min\ f^t x \text{ such that} \begin{Bmatrix} Ax & \le b \\ lb \le x \le ub \\ \end{Bmatrix} \qquad \textbf{(6.26)}$$

In our case we need x (delay parameter) satisfying constraint inequality $Ax \quad \le b$, where b is zero vector size and A is matrix with row elements expressed in inequality (6.x). We can take f equal to 0 since objective function is not used in our case.

Our test were performed for a PUF circuit with n=64 stages. We describe test procedure step by step as follow:

- First step: We produce 50000 random challenge vectors with 64 bit length using unifrom distribution1. After application the challenges to the PUF model programmed in Matlab, obtained responses are saved in a file with challenges.

- Second step: We read N_s number of CRPs from the file
- Third step: We used these N_s rows to solve for $n+\ 1$ variables. For this purpose we use the Simplex algorithm supported by Matlab
- Fourth step: We generate Test =25000 numbers of response using saved test challenges in the file and estimated delay variables. Then we compare the responses produced by our model and the responses saved in the file. The percentage of error are given in the Table 6.1 :

- Table 6.1: Results of calculation test using linear programming approach for PUF implemented with MUXs and response generated by the lineer model. The results are expressed in terms of percentage error for each N_s number of CRPs

	Number of challenge-response pairs N_s								
n=64	32	64	128	256	512	1024	2048	4096	8192
	29.09	15.71	7.38	5.49	4.57	2.91	1.55	0.97	0.49

This test shows that if the relation between response and challenge vector strictly follows mathematical model an adversary can break scheme based on both PUFs having only 1024 CRPs. In order to identify a chip the prediction error rate of the model must be less than the environmental noise, which has maximum value 4.82 %

6.3.2 Experiments using data measured from the PUF on the FPGA

For the test using data measured from the PUF on the FPGA we apply the next procedure:

- First step: We produce 50000 random challenge vectors with 64 bit length using uniform distribution. After application the challenges to the PUF measured responses are saved in a file with applied challenges.
- Second step: We read N_s CRPs from the saved file and apply them to the mathematical model in order to find A matrix in a linear programming problem for $Ax < b$ expression(look at the formula 6.1)
- Third step: We used these N_s CRPs to solve for $n +$ 1 variables. For this purpose we use the Simplex algorithm supported by Matlab
- Fourth step: We generate Ts =25000 numbers of response using saved test challenges in the file and estimated delay variables. Then we compare the responses produced by our model and the PUF on the FPGA

  We apply this test only to the PUF implemented with MUXs. The percentage of prediction error is given in the Table 6.2 below:

Table 6.2: Results of prediction test using linear programming approach for PUF implemented with MUXs and response measured from the PUF on the FPGA . The results are expressed in terms of percentage error for each N_s number of CRPs

	Number of challenge-response pairs N_s								
n=64	32	64	128	256	512	1024	2048	4096	8192
	49.36	49.06	48.70	47.99	47.92	47.93	47.93	47.93	47.93

The results in Table 3 show that the relation between input vectors and output bits of the implemented PUF doesn’t follow the derived mathematical model, formulated in 6.10

We will talk about the reason of failure in prediction of responses in the last section.

6.3 Attacks on PUF Circuit Using Support Vector Machine Classification

6.3.1 Experiment using data generated by Matlab

We performed test on CRPs generated according to linear model programmed in Matlab . Our PUF length is n=64 bits . We apply the same test procedure as it has been done in section 6.3.1 . The one difference is that in step 3 instead of linear programming technique we are using SVM classifier. As a software I have used LIBSVM (Library for Support Vector Machines)tool, which is developed by Chang and Lin[27]. The percentage of prediction error is given in the Table 6.4 below. The results are almost similar as in Table 6.1. The results show that an adversary having only 1024 CRPs can break authentication scheme based on PUF

Table 6.4: Results of prediction test using SVM classifier for PUF implemented with MUXs and response generated by the mathematical model. The results are expressed in terms of percentage error for each N_s number of CRPs .

	Number of challenge-response pairs N_s								
n=64	32	64	128	256	512	1024	2048	4096	8192
	30.20	22.34	19.98	9.96	5.81	2.69	1.99	1.16	0.71

6.3.2 Experiments using data measured from the PUF on the FPGA

6.3.2.1 *Results of software attack for Lineer SVM classifier*

In order to compare the result of prediction for data collected from implemented PUF on the FPGA we apply the same procedure as in section 6.3.2 Instead of linear programming tool we use lineer SVM classifier. The percentage of prediction error is given in the Table 6.5 below:

Table 6.5: Results of prediction test using lineer SVM classification for PUF implemented with MUXs and response measured from the PUF on the FPGA . The results are expressed in terms of percentage error for each N_s number of CRPs

	Number of challenge-response pairs N_s								
n=64	32	64	128	256	512	1024	2048	4096	8192
	50.45	50.06	49.9	49.56	49.40	49.37	49.27	48.95	49.42

The differences between the results shows inconsistency between the data collected from PUF implemented on FPGA and the data generated by Matlab as in section 6.3.2

6.3.2.2 Results of software attack for RBF SVM classifier

In order to compare the result of prediction for data collected from implemented PUF on the FPGA we apply the same procedure as in previous section. The percentage of prediction error is given in the Table 6.6 below:

Table 6.6: Results of prediction test using RBF SVM classification for PUF implemented with MUXs and response measured from the PUF on the FPGA

	Number of challenge-response pairs N_s									
n=64	32	64	128	256	512	1024	2048	4096	8192	17600
	52.31	47.75	47.06	47.03	45.96	41.96	32.34	21.76	14.32	9.87

In spite of data inconsistency between real measured data and lineer model of PUF it possible to predict responses of PUF on FPGA using a lot of CRPs for training of the machine learning algorithm. In this attack we could not reduce the error rate under environmental noise rate as it is supposed to be in order to succeed. However, this result verify that PUF can be more vulnerable to more sophisticated software that may be already exist or will be developed in near future. Variable resistance to different software attacks is undesirable from the security aspect.

6.4 Reason of Failure in Prediction of PUF Circuit Responses

High error rate in classification problem using lineer SVM and linear programming approach applied to the data measured from implemented PUF on the FPGA show inconsistency between responses of linear model and responses from PUF on the FPGA. The result of analysis made below approve that we could not satisfy the main condition of symmetry in implementation of PUF circuit on FPGA.

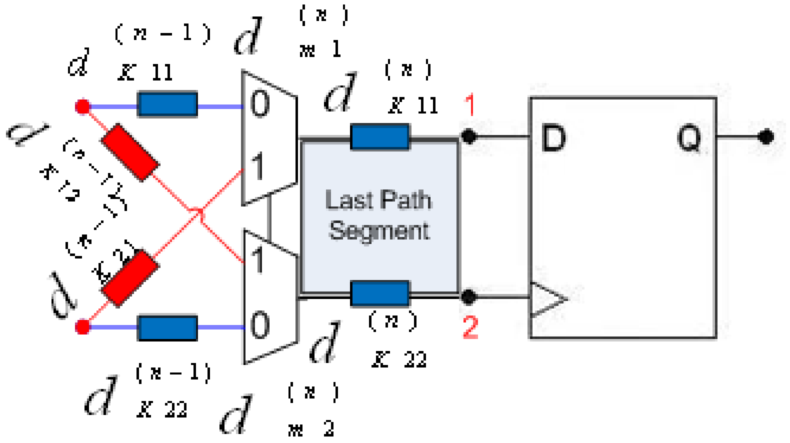


Figure 6.8: Last stage of PUF with cell and connect delays

To identify the critical path in a design and to check whether the timing constraints could be fulfilled, timing analysis tools are used. This tool uses the difference between required time and actual path delays. Actual path delays which consist of on the delay of the design elements and interconnects are read from SDF (Standard Delay Format) file. If we use approach of statical static timing analysis all the delays can be described in terms of mean and standart deviations as it is shown on the Figure 6.9 . Worst case operating condition correspond to the extreme 3σ corners. So, in worst case timing model any delay can be separated as a static delay component and a random delay component. Random delay component are caused by process variation which can reach 3,5 % of nominal (mean) value and refer to 3σ corner in statistical delay model.

For investigation of effect of static delay and random delays on the response of PUF we can use the figure (6.8). Proceeding from previous knowledge let us write expressions for delays at point 1 and point 2 as it is done in (6.25) and (6.26). The result of PUF circuit will depend on the difference Δd ,which is expressed in (6.27)

$$d_1 = d_{1,s} + d_{1,R} \quad \textbf{(6.27)}$$

$$d_2 = d_{2,s} + d_{2,R} \quad \textbf{(6.28)}$$

$$|\Delta d| = |d_1 - d_2| = |\Delta d_s + \Delta d_R| \quad \textbf{(6.29)}$$

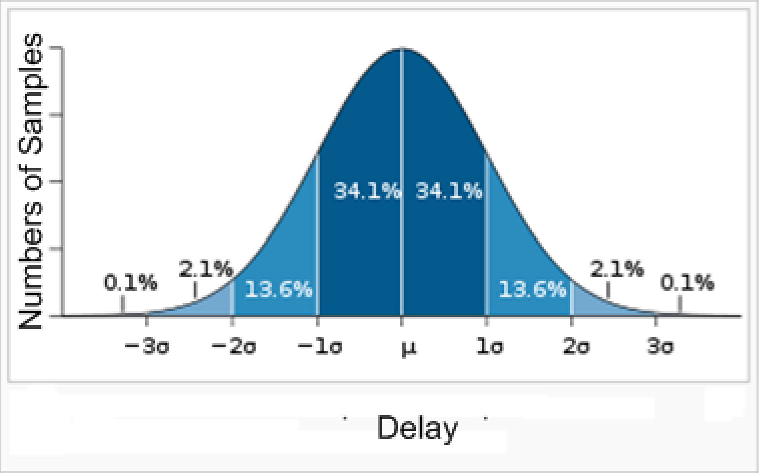


Figure 6.9: Statistical model of delay capturing process variations between ICs

From the figure 6.8 we can derive expressions for $d_{1,s}, d_{2,s}, d_{1,R}, d_{2,R}$ is a function of $d_{K11}^{(n-1)}$, $d_{K21}^{(n-1)}$, $d_{K11}^{(n1)}$ and $d_{m1,S}^{(n)}$

$$d_{1,s} = \overline{C_n} d_{K11}^{(n-1)} + C_n d_{K21}^{(n-1)} + d_{K11}^{(n1)} + d_{m1,S}^{(n)} \quad \textbf{(6.30)}$$

$$d_{2,s} = \overline{C_n} d_{K22}^{(n-1)} + C_n d_{K12}^{(n-1)} + d_{K11}^{(n1)} + d_{m2,S}^{(n)} \quad \textbf{(6.31)}$$

$$d_{1,R} = d_{m1,R}^{(n)} \quad \textbf{(6.32)}$$

$$d_{2,R} = d_{m2,R}^{(n)} \quad \textbf{(6.33)}$$

In the figure (6.10) the result of assymetry in wiring cause a big differences between static delays of path 1 and path 2 before the arbiter.. By using these equations and static or mean delay form timing analysis we can find cumulative delay at point 1 and 2 . The result are graphycally displayed in the figure (6.11) and (6.12). From the figure (6.11) we find the ratio of static component to random component as

$^{\triangle\, d_S}/_{\triangle\, d_R} \approx 50$ and from the figure(6.12) $^{\triangle\, d_S}/_{\triangle\, d_R} \approx 0.4$. As it is seen from the figure (6.11) in direct connection of switch block the response of PUF are not determined by Δd_R but Δd_S since Δd_S value are considerably more than Δd_R. Despite this situation, in the figure (6.12) the value of Δd_S is less than Δd_R which means that our response are completely depends on random process variation. The last case show that by accident we can satisfy the symmetry condition that enable us to get the right response.

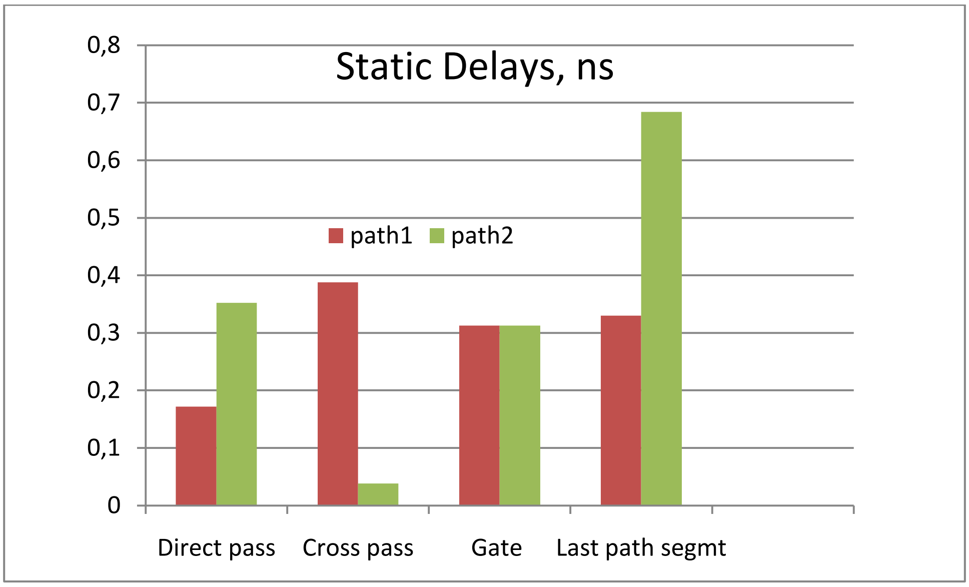


Figure 6.10: Calculated static delays for last stage of PUF circuit

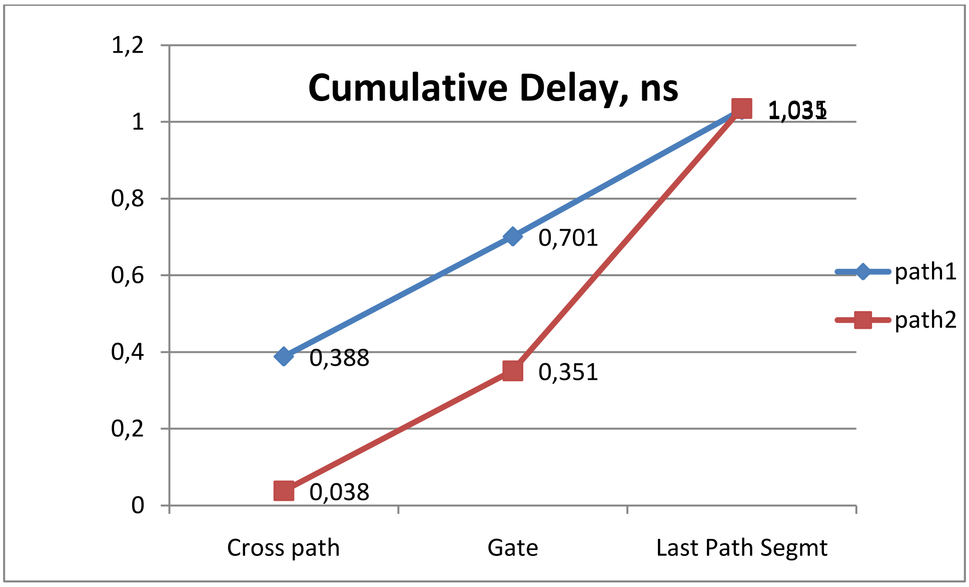


Figure 6.11: Cumulative delay for last stage of PUF with cross connected switch block

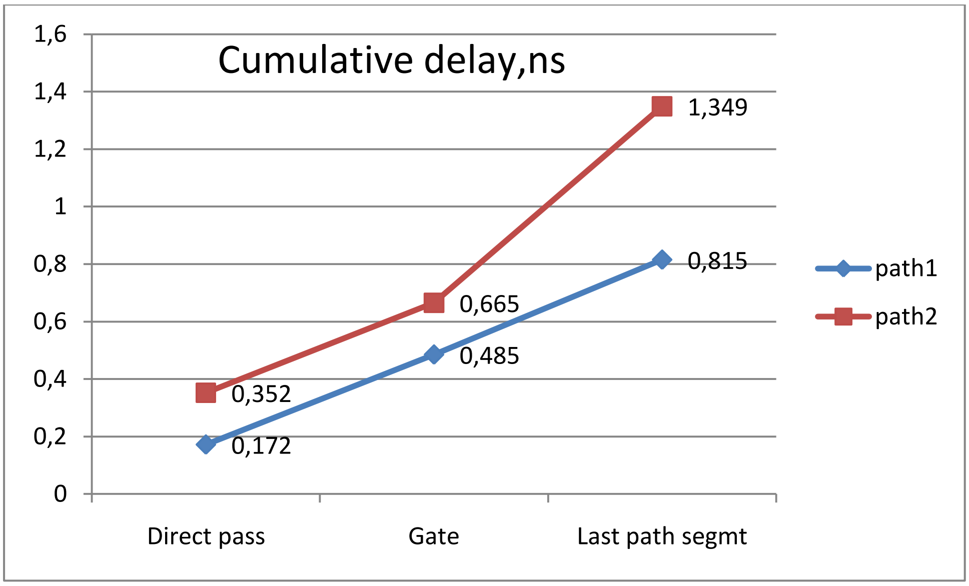


Figure 6.12: Cumulative delay for last stage of PUF with direct connected switch block

7. CONCLUSION

In this thesis we have investigated the security and reliability for arbiter-based PUF circuit and conducted preliminary experiments. By using authentication method based on PUF circuit it is possible to store secrets on a chip that is less vulnerable to invasive attacks than traditional digital methods.

We have implemented arbiter-based PUF using MUX on Xilinx FPGA. Experiment results have shown that there are enough variation between programmable gates not only on different FPGA chip but also on the same chip for identification purposes. The undesired effect of temperature and power supply voltage variations can change the delay characteristic of PUF circuit making worse the reliability parameter. Experiments of applying different challenges to PUFs circuit which are in different position across FPGA chip show that not any challenge can be used for identification purposes. Some of the challenges produce the same outputs in different FPGA circuits. The assymetry in wiring reduce the numbers of challenges that provide high inter-chip variations.

We have tested the security of linear model PUF circuit, described by equation in section 3.5, against software attacks. For this purpose we have used the linear model the security analysis of linear model created in Matlab suggest that the device could be vulnerable to model building software attack. In fact, we see that

Our experiments where we use the responses produced by linear model and response measured from implemented PUF have shown different resistance to software attacks. In linear model, 1024 CRPs is enough to solve for all delay variable or to predict the outputs for any next challenges. Hovewer, lineer programming and lineer SVM haven’t succeeded in prediction of responses, which bring the fact of inconsistency between reponses of lineer model and measured from FPGA responses. Delay analysis report presented by Xilinx Timing Analyzer tool have shown that in spite of symmetrical position arrangement of MUXs across a chip we could not achieve desired symmetry in wire delays between switch blocks. It lead to corruption of responses in PUF on FPGA. In spite of that, application of SVM using radial based kernel have considerably increased prediction rate up to 90 percent where we need 17600 CRPs for a training procedure. This fact verify that PUF based system can be vulnerable to more sophisticated attacks, which is unacceptable from the security aspect. Especially in case of man in the middle attack . Since authentication process of PUF occures in untrusted and open environment the adversary can easily collect the data, which will be used for software attacks. More authentication process provide him more CRPs. The low intra-chip variation can complicate the aim of adversary since in this case we need more CRPs to train our machine learning algorithm.

In order to prevent the predictions of responses, we can employ non-linear arbiters such as feed-forward arbiter PUFs. It is difficut for an adversary to build an appropriate software model of these arbiters[1].

REFERENCES

[1] Erdinç Öztürk, Ghaith Hammouri and Berk Sunar “Toward Robust Low Cost Authentication for Pervasive Device” , IEEE computer society , 2008

[2] Daihyun Lim, “Extracting Secret Key from Integrated Circuits”, April 5, 2005

[3] M. Feldhofer, S. Dominikus, J. Wolkerstorfer, “Strong Authentication for RFID Systems Using the AES Algorithm”: CHES 2004, LNCS 3156, pp. 357–370, 2004.

[4] J.-P. Kaps, B. Sunar, “Energy Comparison of AES and SHA-1 for Ubiquitous Computing”, in: X. Z. et al. (ed.), Embedded and Ubiquitous Computing (EUC-06) Workshop Proceedings, vol. 4097 of Lecture Notes in Computer Science (LNCS), Springer, 2006

[5] A. Poschmann, G. Leander, K. Schramm, C. Paar, *"A Family of Light-Weight Block Ciphers Based on DES Suited for RFID Applications"*. Workshop on RFID Security 2006, Graz, Austria. 12.-14. July, 2006.

[6] “IBM 4764 PCI-X Cryptographic Coprocessor Custom Software Interface Reference Release 3.30” http://www-03.ibm.com/security/cryptocards, June 11 2009.

[7] Nathan Beckman and Miodrag Potkonjak, “Hardware-Based Public-key Cryptography with Public Physically Unclonable Functions ”, : IH 2009,LNCS 5806, p-206-220 , 2009 Springer-Verlag Berlin Heilderberg 2009

[8] Blaise Gassend, “Physical Random Functions" M.S. thesis, Massachusetts Institute of Technology, Jan. 2003.

[9] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas, “Silicon PhysicalRandom Functions ," in Proceedings of the Computer and Communication Security Conference,

[10] B.Gassend, D.Clarke,M.van Dijk and S.Devadas. Delay-based Circuit Authentication and Application. Massachusetts Institute of Technology ,Laboratory of Computer Science Cambridge

[11] Kharagpur 1,” *Lesson 20 : Field Programmable Gate arrays and Applications*”, Design of Embedded Processors Version 2 EE IIT, www.scribd.com/doc/31436874/FPGA

[12] Virtex-2 Pro and Virtex-2 X Platform FPGAs: Complete Data sheet, November 5, 2007

[13] Volnei A Pedroni, “*Circuit Design with VHDL*”, pp.1-4, MIT Press Cambridge, Massachusetts,London, England, 2004

[14]Erdinc Ozturk, Ghaith Hammouri, Berk Sunar, “*Physical unclonable function with tristate buffers”.* In: Proceedings of the International Symposium on Circuits and Systems (ISCAS 2008), 18-21 May 2008, Seattle, Washington, USA, pp. 3194-3197, IEEE, Washington, DC, USA (2008)

[15] Mehmet Soybalı, “*Implementation of Reliable RFID Applications”* ,Undergraduate thesis,May 2010

[16] Blaise Gassend,Dwaine Clarke,Marten van Dijk and Srinivas Devadas, “Silicon Physical Random Function”, CCS’o2 November 18-22, 2002, Washington DC, USA

[17] Ravikanth Pappu, Ben Recht, Jason Taylor, Neil Gershenfeld,” Physical One-Way Functions”, VOL 297 SCIENCE, www.sciencemag.org, 20 SEPTEMBER 2002

[18]]”Classification”, Wikipedia ,http://en.wikipedia.org/wiki/Classification_(machine_learning), October 2010

[19] Thomas S.Ferguson,”Linear Programming” ,Electronic Text, http://www.math.ucla.edu/~tom/

[20] ”Linear Programming”, Wikipedia, http://en.wikipedia.org/wiki/Linear_programming, December 2010

[21] “Solve Linear Programming Problem”, The MathWorks Inc, 1984-2010 http://www.mathworks.com/help/toolbox/optim/ug/linprog.html

[22] ”Support Vector Machine”,Wikipedia, http://en.wikipedia.org/wiki/Support_vector_machine, December, 2010

[23] J.P.Lewis, Tutorial on SVM, CGIT Lab, USC, 2004

[24] “PUF Based RFIDs ”http://www.verayo.com/product/pufrfid.html

[25] R. Saleh, “Flip-Flop and Clock Design”, Lecture 6, www.ece.ubc.ca/~elec579/clock**flop**.pdf

[26] V.Jeyaraman, ”Design ,Characterization and Automation of Ultra-high Temperature Standard Cell Library for Harsh Environment” , Msc Thesis, December 2004

[27] Chih-Chung Chang and Chih-Jen Lin,”LIBSVM - A Library for Support Vector Machines”

[28] Rakesh Chadha, J. Bhasker,”Statistical Static Timing Analysis-A Better Alternative”, 2/3/2009

[29]K.P.Soman,Y.Ajay,R.Loganathan, ”Machine Learning with SVM and Other Kernels”, PHI Learning Pvt. Ltd, 2009 ,p.122-124

[30] J.A.K. SUYKENS and J. VANDEWALLE,” Least Squares Support Vector Machine Classifiers”, Neural Processing Letters 9: 293–300, 1999

APPENDICES

APPENDIX A.1: Extraction of Expressions for u_i and v_i Variables in Equation 3.17

APPENDIX A.1

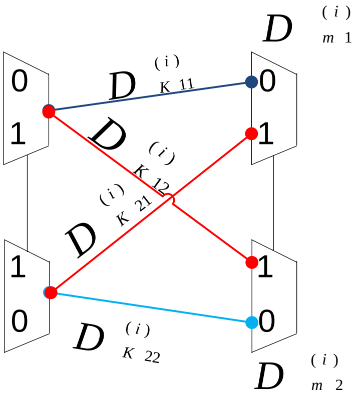


Figure A.1: Two successive MUX blocks with delay variables

Let's try to express $u_i$variable defined in section (3.5) in terms wire delays $D^{(i)}_{K11}$, $D^{(i)}_{K12}$, $D^{(i)}_{K21}$, $D^{(i)}_{K22}$and upper MUX and lower MUX gate delays which are $D^{(i)}_{m1}$, and $D^{(i)}_{m2}$, respectively. We can achieve it by simple substitution of a_i, d_i, b_i, f_i parameters in terms of wire and gate delays which are shown in Figure A.1 . Comparing two figures 6.x1 and 6.x1 give us the expressions stated below:

$$a_i = D^{(i)}_{K11} + D^{(i)}_{m1} \qquad \textbf{(A.1)}$$

(A.1)

$$b_i = D^{(i)}_{K12} + D^{(i)}_{m2} \qquad \textbf{(A.2)}$$

(A.2)

$$f_i = D^{(i)}_{K22} + D^{(i)}_{m2} \qquad \textbf{(A.3)}$$

(A.3)

$$d_i = D^{(i)}_{K21} + D^{(i)}_{m1} \qquad \textbf{(A.4)}$$

(A.4)

$$\boldsymbol{u_i} = \frac{(\boldsymbol{a_i} - \boldsymbol{d_i}) + (\boldsymbol{b_i} - \boldsymbol{f_i})}{\mathbf{2}} = \frac{\left(D^{(i)}_{K11} + D^{(i)}_{m1} - D^{(i)}_{K21} - D^{(i)}_{m1}\right) + \left(D^{(i)}_{K12} + D^{(i)}_{m2} - D^{(i)}_{K22} - D^{(i)}_{m2}\right)}{\mathbf{2}} = \frac{(D^{(i)}_{K11} - D^{(i)}_{K22}) + (D^{(i)}_{K12} - D^{(i)}_{K21})}{\mathbf{2}} \qquad \textbf{(A.5)}$$

(A.5)

$$\vartheta_i = \frac{(\boldsymbol{a_i} + \boldsymbol{d_i}) - (\boldsymbol{b_i} + \boldsymbol{f_i})}{\mathbf{2}} = \frac{\left(D^{(i)}_{K11} + D^{(i)}_{m1} + D^{(i)}_{K21} + D^{(i)}_{m1}\right) - (D^{(i)}_{K12} + D^{(i)}_{m2} + D^{(i)}_{K22} + D^{(i)}_{m2})}{\mathbf{2}} = \frac{\left(D^{(i)}_{K11} - D^{(i)}_{K22}\right) + (D^{(i)}_{K21} - D^{(i)}_{K12})}{\mathbf{2}} + (D^{(i)}_{m1} - D^{(i)}_{m2}) \qquad \textbf{(A.6)}$$

CURRICULUM VITAE

Candidate's full name **:** Zaur TARIGULIYEV

Place and Date of Birth : Baku / AZERBAIJAN, 07.05.1979

Permanent Address **:** M.Hadi str,23/136, Baku/ AZERBAIJAN

Universities and
Colleges attended **:**Istanbul Technical University (1998-2002)