June 2018

A Key Based Obfuscation and Anonymization of Behavior VHDL Models

Balausha Varshini Kandikonda
University of South Florida, kandikondab@mail.usf.edu

A Key Based Obfuscation and Anonymization of Behavioral VHDL Models

by

Balausha Varshini Kandikonda

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Srinivas Katkoori, Ph.D.
Sriram Chellappan, Ph.D.
Robert Karam, Ph.D.

Date of Approval:
May 24, 2018

Keywords: Intellectual Property, Trojan Horse, Reverse Engineer, Resilience, Overhead

DEDICATION

I dedicate this work to my family and all my beloved ones who helped and supported me.

ACKNOWLEDGMENTS

I would first like to formally acknowledge Dr. Srinivas Katkoori for providing the opportunity to work on this project. I always cherish his support and express my gratitude that he has helped me identify my strengths and use them in a best possible way. I am forever grateful to him for bringing me success with his guidance and constant support. I give thanks to Dr. Sriram Chellappan and Dr. Robert Karam for volunteering their precious time to serve as members on my thesis committee. I thank God for shielding me in every phase of my life and making me pursue a fruitful path. My special thanks to Kiran Kumar Kandikonda and Nagambika Kandikonda (parents), Harshith Kandikonda (brother), Bharath Bukka (uncle) and all my beloved ones for encouraging me in achieving my goals. I would also like to thank my friends Shiekh Ariful Islam, Rohith Challa and all the colleagues I have acquainted for their help and assistance.

TABLE OF CONTENTS

LIST OF TABLES ii

LIST OF FIGURES iii

ABSTRACT iv

CHAPTER 1: INTRODUCTION 1

CHAPTER 2: BACKGROUND AND RELATED WORK 4
- 2.1 Behavioral-level Obfuscation 5
- 2.2 Register Transfer (RT-) Level Obfuscation Methods 6
- 2.3 Gate-level Obfuscation Methods 8
- 2.4 Physical-level Obfuscation Methods 11
- 2.5 Software Obfuscation 13
- 2.6 Chapter Summary 14

CHAPTER 3: PROPOSED APPROACH 16
- 3.1 Proposed Framework 17
  - 3.1.1 Key Based Obfuscation Step 19
  - 3.1.2 Anonymization Step 21
- 3.2 Difficulty to Reverse Engineer 22
- 3.3 Chapter Summary 24

CHAPTER 4: EXPERIMENTAL RESULTS 25

CHAPTER 5: CONCLUSION 29

REFERENCES 30

ABOUT THE AUTHOR ..... END PAGE

LIST OF TABLES

Table 3.1 Resilience Table (see Equation 3.1) 23

Table 4.1 Benchmark Details 25

Table 4.2 Simulation Time for 5 Test Vector Sequence 26

Table 4.3 Simulation Time for 1K Test Vector Sequence 26

Table 4.4 Simulation Time for 10K Test Vector Sequence 27

LIST OF FIGURES

Figure 1.1 Reverse Engineering 2

Figure 2.1 Genetic Algorithm 5

Figure 2.2 RT-level Obfuscation 7

Figure 2.3 Gate-level Obfuscation of D Flip-flop 10

Figure 2.4 An Illustrative Example of Physical-level Obfuscation 12

Figure 2.5 Software Obfuscation 13

Figure 3.1 Man-in-the-Middle Attack Model 16

Figure 3.2 Proposed Obfuscation and Anonymization Flow 17

Figure 3.3 Key Based Obfuscation Algorithm 18

Figure 3.4 Obfuscation of OR Gate 19

Figure 3.5 Obfuscation Step 20

Figure 3.6 Anonymization Algorithm 21

Figure 3.7 Anonymization Step 22

Figure 4.1 Overhead of Designs for 32-bit Key 27

ABSTRACT

Intellectual Property (IP) based Integrated Circuit (IC) design is an established approach for the design of a complex System-on-Chip (SoC). Porting the preparatory designs to third-party without enough security margin exposes an attacker to perform reverse engineering (RE) on the designs and hence counterfeiting, IP theft etc., are common now-a-days. Design obfuscation can reduce RE attempt by an attacker. In this work, we propose a key based obfuscation and anonymization method for a behavioral IP. Given a behavioral VHDL description, the assignment and conditional statements are modified by incorporating random boolean operations with unique random key bits. The obfuscated VHDL is then anonymized by random identifiers. The resultant behavioral model can be simulated correctly upon application of original key sequence. Simulation results with nine datapath intensive benchmarks with three different lengths of test sequences show that the simulation overhead is negligible (only a few seconds). We evaluate the probability of reverse engineering the obfuscated design and show that it is extremely low.

CHAPTER 1: INTRODUCTION

With more building blocks being deployed as Intellectual Property (IP), the problem in regards to their security is also increasing. These are more vulnerable to malicious activities which decreases the hardware protection. Some of the malicious activities are Trojan injection, unauthorized over production, and alteration of the code. Trojan injections can cause a major problem. Once, these are injected, they can be triggered at any time, even during the execution. IC designs are easy targets to the hardware Trojans as they are vulnerable (for eg., due to design outsourcing) and can cause malicious alterations to the circuit. Unauthorized over production occurs when the IP is used or accessed more than required or mentioned without the prior knowledge of the IP owner.

For these reasons, protection and security of the hardware has become important in recent times. Hardware protection primarily provides security against malicious activities like code modifications to increase the difficulty of reverse engineering. At the same time, it also decreases the occurrence of Trojan insertion.

Many methods have been proposed till date to protect the hardware. Encryption of the hardware at behavioral level can be done by changing the names with the help of the keys. Gate level security is achieved by introduction of Physical Unclonable Functions (PUFs) and multiplexers which complicate the design process but does not change the functionality. Physical level security can be achieved by changing the doping concentration of the material. These methods have their own drawbacks such as increasing the time complexity to simulate a program, usage of more power

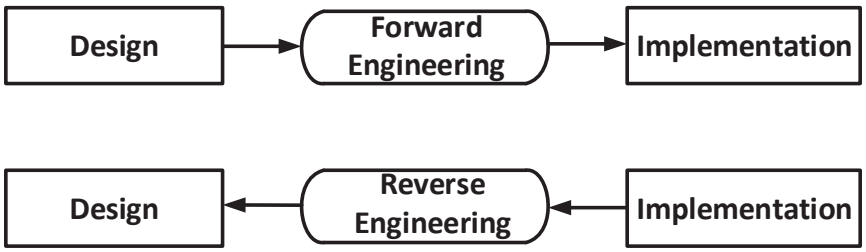

Figure 1.1: Reverse Engineering

than required thus making it less power efficient, and so on. From all these, it is clear that a time and power efficient method is needed.

To achieve an efficient tool that is least vulnerable to attacks, the probability of reverse engineering should be decreased. Reverse engineering can be best explained with the help of Figure 1.1. Consider a design is forward engineered to get its implemented. The process of re-creating the design from the implementation by transforming the implementation is called reverse engineering.

To protect the original design, reverse engineering should be hard for the attacker. That means, even though the attacker can get the data, she cannot access it unless she does reverse engineering. To mitigate reverse engineering of the design, one of the techniques used is to obfuscate the design, which is an easy and inexpensive way. Obfuscation is very common in software. Software obfuscation is performed by renaming the identifiers and encrypting the functionality of the code.

Obfuscation incorporates additional code in the original program without modifying the functionality of the program. It should be implemented tactfully in such a way that the attacker cannot understand the code for reverse engineering. Security against hardware trojans can also be achieved through obfuscation. Anonymization, on the other hand, means renaming the variables of

the program so that reverse engineering becomes merely impossible for the attacker. In this work, we propose a key based obfuscation and anonymization method for behavioral VHDL IP models.

In the proposed approach, even if a third party tries to over produce the code illegally, it cannot be accessed by the users as the key is only disclosed to legitimate users. Due to this type of implementation, illegal over production would be drastically reduced. Difficulty of reverse engineering is inversely proportional to the level of vulnerability i.e., the increase in difficulty of reverse engineering the IP model, decreases the malicious activities such as modifying the code. Not only protection of the IP model is achieved but also, it has been noted that the overhead occurred is only a few seconds.

The rest of the thesis is organized as follows. Chapter 2 presents an overview of hardware trojans, obfuscation and various techniques used to resist reverse engineering at different abstraction levels. Chapter 3 presents in detail the proposed method for key-based obfuscation and anonymization. Chapter 4 reports the experimental results. Finally, Chapter 5 draws conclusions.

CHAPTER 2: BACKGROUND AND RELATED WORK

Hardware obfuscation is a technique used for protecting the design from untrusted execution. It increases the difficulty to understand, to read, and also to modify the functionality of IC/IP model. Software obfuscation is also a similar technique which is used for protecting the logic from third party by generating a machine code so that it will increase the difficulty for reverse engineering. There are many automated tools available for software obfuscation.

Software obfuscation is widely used when compared to that of hardware. Nearly everything in software is ensured by obfuscation which is not the case in hardware design. It is quite challenging to create a tool for different abstraction layers of the hardware. The design that is obtained after obfuscation turns out mildly complex but it will also become exceptionally protected. Hardware obfuscation is very much distinctive when compared to software obfuscation. In software obfuscation, the structure of the code is modified or altered where as in hardware, the providing protection to the functionality of the code is main concern.

This chapter reviews existing methods at behavioral-, RT-, gate-, and layout-levels for hardware obfuscation. It also reviews popular software obfuscation techniques currently in practice.

2.1 Behavioral-level Obfuscation

When obfuscating an IP, a functionally equivalent source file is compiled, which is highly difficult for humans to understand by just viewing the source file and thus is extremely difficult to perform reverse engineering on. Veeranna and Schafer [1] had designed an obfuscation tool for behavioral IP source code to decrease Quality of Results (QoR) degradation due to the obfuscation. The authors proposed two methods based on Genetic Algorithm (GA) and iterative greedy algorithm. The obfuscation based on Genetic Algorithm has two steps. The first step involves

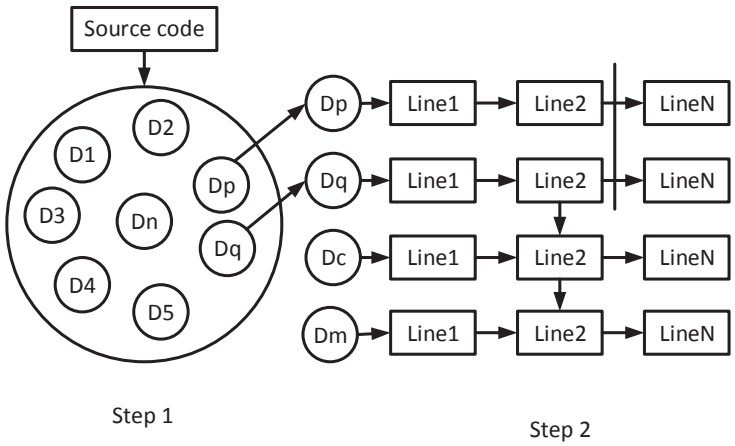

Figure 2.1: Genetic Algorithm

initial population generation, where a random line in the behavior IP is parsed. After parsing that line, a decision is taken whether or not to obfuscate that line and the probability is noted. Then, generation of offspring is carried out. This uses two functions, mutation and crossover. Mutation simply flips the lines of the code in the offspring and crossover happens on two designs of the parent which are chosen randomly from the population. Figure 2.1 depicts the two steps in the algorithm. D1, D2,... are the designs from which the lines are randomly chosen whether to obfuscate or not. Dc and Dm are the crossover and mutation designs.

Fast iterative-greedy technique provides more appropriate QoR and obfuscation level with a single design. This predominantly conveys which of the obfuscation has better QoR. This method has three steps. In the first step the line numbers are extracted where obfuscation has been achieved in a completely obfuscated file. In the second step, all of these lines obtained are analyzed to determine which of them contribute to QoR. Finally, in the third step, a new behavioral IP is generated with the lines contributing to less QoR degradation. It was observed that the iterative-greedy method produced more efficient obfuscation and had lesser run time when compared to GA.

2.2 Register Transfer (RT-) Level Obfuscation Methods

Castillo *et. al.*, [2] proposed a high level design protection scheme, namely, IPP@HDL. It uses electronic signature to protect the designer rights in the development and distribution of reusable VHDL modules. The procedure relies upon facilitating the bits of the digital signature inside memory structures or in combinational circuits that are a part of the framework. Two strategies, namely, *signature embedding* and *signature hosting* are used. These follow the process, signature preparation, spreading, extraction and validation. Error correction codes, additional logic tasks for detecting signature extraction, and strategies to host signatures methods are analyzed to combat the tampering attacks. Storage of the signature bytes costs additional hardware which limits the application of this method.

Islam and Katkoori [3] proposed a key based RTL obfuscation during high-level synthesis (HLS). The CDFG of the behavioral description of the design is analyzed and obfuscation points are embedded with random keys. Multiplexers based keys are stored in additional registers available in RTL datapath. The algorithm consists of two steps. First, all the operations of CDFG are

analyzed for possible obfuscation points. Then, a random operation along with input or output line is selected for all the obfuscation key bits and are replaced with a multiplexer. Inputs to the design are increased by randomly passing a line from the design as an input along with the original inputs. Compared to other techniques, this obfuscation technique has less performance overhead.

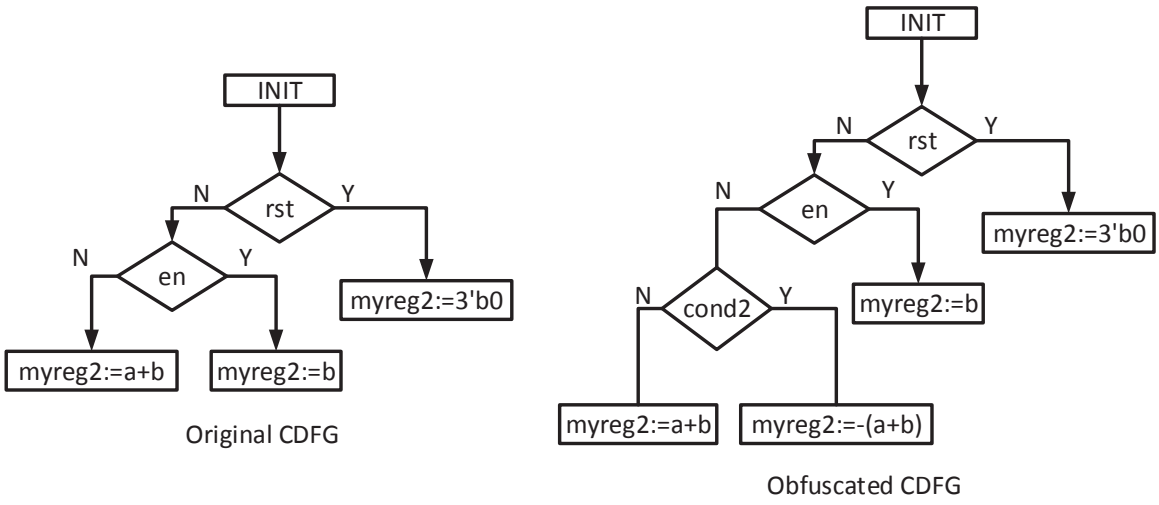

Figure 2.2: RT-level Obfuscation

For RTL obfuscation, Chakraborty and Bhunia [4] proposed low-overhead key-based IP protection. The obfuscation scheme is comprised of four steps. First, blocks of RTL code is parsed and transformed into CDFG. These small CDFGs are merged to generate a larger CDFG. The next part is to host the mode-control FSM by distributing the elements into registers non-contiguously. The third step is to modify the CDFG branches by using control signals from step two and embedded with additional states to hold the obfuscation key. The number of states required increases linearly with the size of key.

Figure 2.2 shows an illustrative example of the original and modified version of CDFG. Additional key based clause is added at the end so that passing a correct key value as input can only produce the correct output. Finally, obfuscated RTL code is generated from the modified CDFG. This mechanism not only prevents IP infringement but also incurs low design overhead.

Koteshwara *et. al.*, [5] presented a dynamic approach to produce an inconsistent circuit behavior when an incorrect key is applied. It increases the time complexity of deciphering and decoding of the correct key while implementing a brute-force attack on it, even if shorter key lengths are used. Key based obfuscation is achieved by adding fixed key bits to the multiplexers of the control flow, time variant based on the input key value and circuit triggering at random time duration for every incorrect key. Trigger circuits are designed to withstand the hardware Trojans using counters. Whenever a Trojan is activated, the signal behaves differently due to trigger modifications. This activates a delay counter and when this overflows, a final counter is incremented which is served as an input to the trigger generator circuit. Through this approach, controls overheads are reduced.

2.3 Gate-level Obfuscation Methods

Chakraborty and Bhunia proposed HARPOON (HARdware Protection through Obfuscation of Netlist) [6], an obfuscation methodology for gate-level netlist of pre-synthesized IP core. Here, obfuscation and authentication are provided by modifying the soft IP core. Obfuscation is used to reduce reverse engineering. During obfuscation procedure, a small finite state machine (FSM) is inserted into the state-transition function of the circuit. This FSM will work normally only when specific input sequence is received. It also modifies the chosen nodes. The chosen nodes are

those having higher fan-out logic cone. For each of the correct input sequence, a digital signature is embedded into the output. Identifying the correct input from these signatures is hard. The HARPOON design methodology is applied for selecting the optimal set of nodes for modification using ranking algorithm. The approach gives a piracy-proof design flow but uses control blocks which require additional area as it fails to analyze the security vulnerable nets of IP core and inserts random key insertion.

Chakraborty and Bhunia [7] also presented obfuscation for protection against hardware trojan with additional states in state transition graph. Obfuscation is done by modifying the state transition graph which provided enhanced protection against hardware trojans.

Tehranipoor and Koushanfar [8] presented several trojan detection techniques. The method includes side-channel analysis or Trojan activation at the chip-level and architectural level. Chip-level trojan detection process merges with power analysis which includes region-free and region-aware trojan activation schemes. Architectural-level trojan detection scheme involves balancing of if-else statements without changing the execution time and power. These techniques can be used at design phase to detect a trojan. Once a trojan is detected, it can be removed.

The obfuscation methodology in [7], involves triggering an arc which connects the normal nodes from the obfuscated state transition graph. All the node states which are reachable are blown up by exponential functions using state elements making them unreachable and the states which are unreachable are made reachable in the obfuscated mode. To decrease the probability of finding the unreachable nodes, the obfuscated state space is made larger by inserting excessive state elements. This makes reverse engineering infeasible for the attacker.

PUF based Challenge-Response Pairs (CRPs) [9] are proposed for unique secret key generation to be used in combinational and sequential logic obfuscation. In combinational logic obfuscation, a part of the original circuit is replaced with a delay-based PUF along with configurable logic circuit. In order to increase the security of the circuit, the delay PUF must be placed where high-activity and timing-stressed regions are not present. This is because of the unstable nature of PUF. A challenge (n-bit) is given to PUF to make it stable.

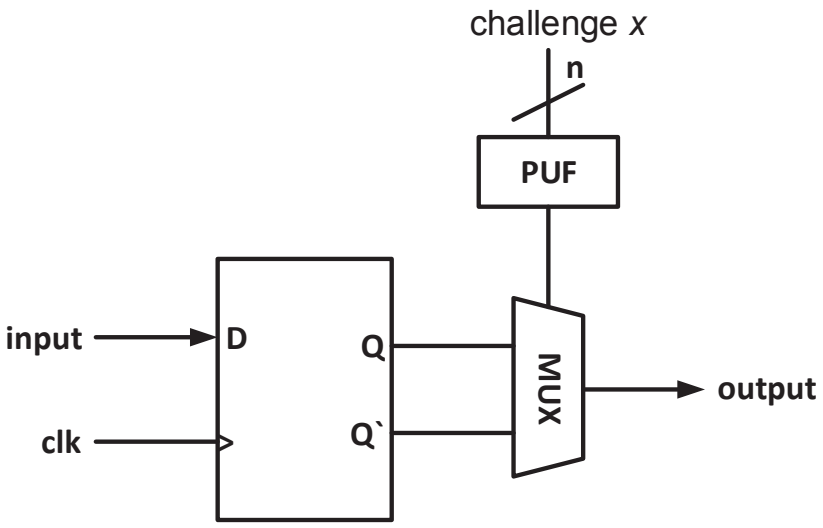

Figure 2.3: Gate-level Obfuscation of D Flip-flop

In sequential logic obfuscation, the outputs of the flip-flop are obfuscated to make the correct output indistinguishable from the wrong one. Figure 2.3 determines the modifications done to D flip-flop in order to obfuscate the output of the flip-flop. Here, an n-bit PUF is added to the multiplexer for obfuscation. The obfuscation should done keeping the delay overhead in mind. Given the unstable nature of PUF and constant prediction of PUF output, obfuscation is less effective.

Dofe and Yu [10] presented *state deflection* method for Finite State Machines (FSMs). In this approach, during a wrong key application, it would dynamically deflect state transitions from the allowable transition path to a black hole cluster. Multiple states are merged to create a black hole and every black hole is mapped to a wrong key. These black holes remain unstable by constantly shifting to other states. Obfuscation is done on the gate-level netlist without the prior knowledge of the original states and signals. This is done by adding additional input signals to the FSM and changing the state bits. If an original state is categorized as a black hole, the output bit is flipped. The key sequence is checked at every state transition. Although this methodology has high success rate in obfuscation, it is limited due to the hardware cost.

2.4 Physical-level Obfuscation Methods

Layout-level obfuscation methods involve camouflaging and change in physical parameters to the circuit components. Vijaykumar *et al.* [11], presented an outline of low-level obfuscation method to protect against image recognition attack. Obfuscating the hardware at physical level has a very large design space. Obfuscation is mainly based on stealthy circuits. These circuits have distinct logic function for the original and the one which is extracted after performing reverse engineering. This is a three layer model of which the bottom layer (device-level mechanism) and middle layer (logic-level mechanism) can be used for physical design obfuscation. The top layer has obfuscation techniques which are at the gate-level.

The bottom layer has device-level mechanisms which re-size the transistors based on its group. The groups are classified depending on the type of effect i.e., stuck-at-faults, stealthy signaling, and delay manipulation. Device-level techniques focus on the creation of stuck-at and delay

faults to harness the reverse engineering. Stuck-at faults can be created by changing the doping concentration in the transistor. This is illustrated in Figure 2.4. The source and drain of the original PMOS transistor is doped with n-type which produces a constant output causing stuck-at faults. The most significant mechanisms in which this can be achieved are device specific mechanisms, where source or channel doping is done. Few of them are interconnect specific mechanisms, where inter-layer dielectric and inter-connect can be manipulated and dummy logic can be inserted. Mechanisms where usage of crosstalk or manipulations done on stealthy signaling and timing faults using lithographic printable features are also used.

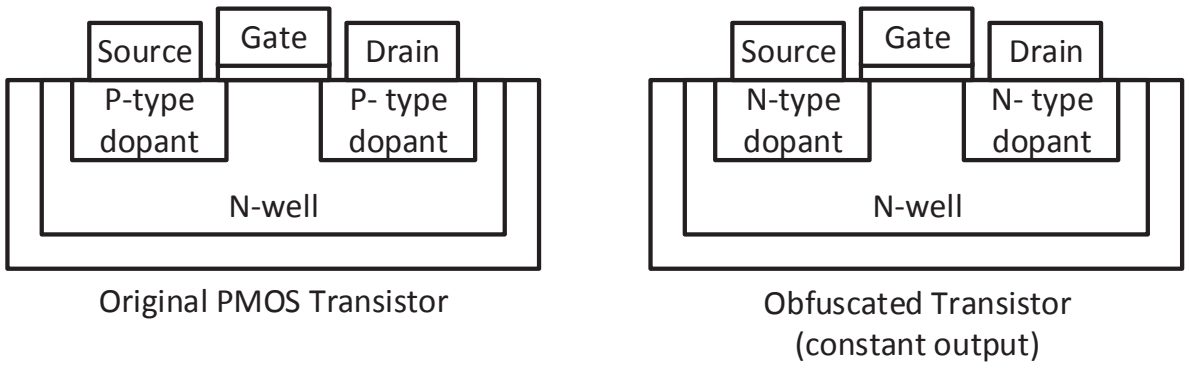

Figure 2.4: An Illustrative Example of Physical-level Obfuscation

The middle layer has logic-level mechanisms which are focused on circuit structures formed from manipulations of the bottom layer. These circuits intervene between the device-level mechanisms and complex logic functions. The techniques used for obfuscating the logic-level mechanisms are circuit transformations which include CMOS circuits, pass transistors, dynamic logic, Differential Cascode Voltage Switch (DCVS) logic, and flip-flops. Promotion of physical mechanism to logic level which include look-up tables, Programmable Logic Array (PLA) cross points, and usage

of gate functions which are restricted and countermeasures for obfuscation mechanism. All these mechanisms provide hardness and stealthiness to the obfuscation of the low-level design.

2.5 Software Obfuscation

JAVA has been the most used programming languages of all times and it still continues to be the best one because of it is open, well defined, and portable. These factors also make it vulnerable to reverse engineering attacks. To counter this we use obfuscation techniques to make the program less readable and more complex to understand by others making this the most secured way to write a program. Software obfuscation can be explained with the help of Figure 2.5. As shown, the original source code is sent through the obfuscation tool as an executable file to the user. If an attacker tries to de-compile the file, she cannot retrieve the original file, thus securing the original one.

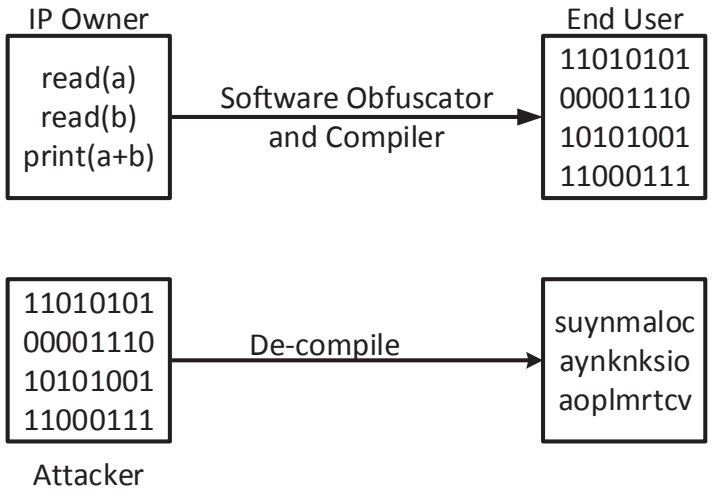

Figure 2.5: Software Obfuscation

The obfuscation technique used in JIRO [12] is an identifier renaming algorithm. It uses four different algorithms each of them receiving an input source 'x' and generated target code 'x'. All these algorithms have three main steps in common, overusing identifiers, overloading unrelated methods, and introducing illegal identifiers.

Overusing identifiers uses randomly generated identifier names instead of using the sequence names. This increases the difficulty of static attacks and at the same time increasing the difficulty to reverse engineer the program. Overloading unrelated method renames the methods of the compiled class with same identifier by considering the complexity of inheritance relation between the sub class and super class. Illegal identifier obfuscation method uses purely keywords and string of keywords combined with illegal characters to replace the common identifiers. This results in a relationship between the size of the program and the obfuscation effects. These have a direct impact with the size of the original program. It can be seen that with the help of the tool developed, the JAVA program can be protected. By this way, the complexity of the program code sharply rises and thus increases the compilation time for the code as well. This procedure can be implemented in hardware obfuscation.

2.6 Chapter Summary

In this chapter, obfuscation techniques at different levels for hardware and software are described. At behavior-level, changes are done to the identifiers. CDFGs are modified at RT-level. At gate-level, PUFs are introduced and at physical-level, doping concentrations are modified.

Although all these approaches provide security against attacks, they propose a complex way to perform obfuscation of a program and hence the time taken for a defender also increases with the obfuscation level. Assuming man-in-the-middle attack for a behavioral IP, we propose an automated approach to obfuscate the VHDL model followed by anonymization with reasonable runtime overhead (few seconds) which will be discussed in next Chapter.

CHAPTER 3: PROPOSED APPROACH

When an IP owner sends his/her proprietary design to fabrication facility, an attacker during system integration can spoof the design and alter the functionality without explicit knowledge of parties involved. We mimic such attack model as man-in-the-middle attack. Although the IP owner and the end user may think that the design is unaltered, attacker may reverse engineer the design for IP theft, counterfeiting etc., or embed HT due to lack of centralized control in the supply chain. To reduce reverse engineering attempt for an attacker, obfuscation and anonymization can enable the designer to protect behavioral VHDL model of a design. The obfuscation and anonymization tool developed helps in reducing the attack depicted in Figure 3.1.

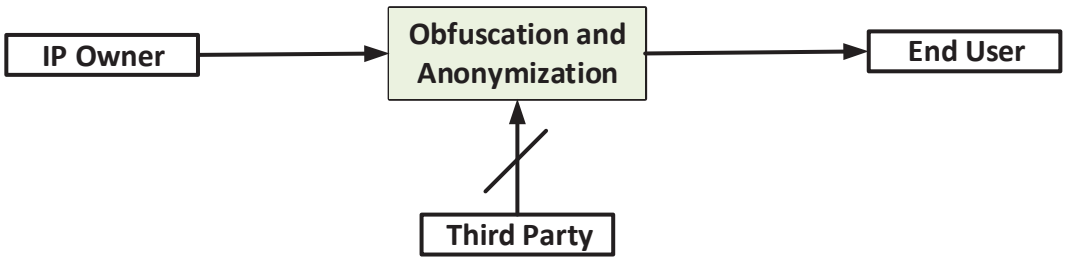

Figure 3.1: Man-in-the-Middle Attack Model

3.1 Proposed Framework

We propose key-based obfuscation and anonymization technique to protect the functionality of a behavioral VHDL model from reverse engineering.

Obfuscation can hide the functionality of a code given a number of key and their length. In our work, the given VHDL code is passed through a parser, built in lex and yacc, which generates a key based obfuscated VHDL model of the original code. Figure 3.2 shows the overall flow. We incorporate random boolean operations with random key bits in the assignment and conditional evaluation statements of the original VHDL without any change in original function.

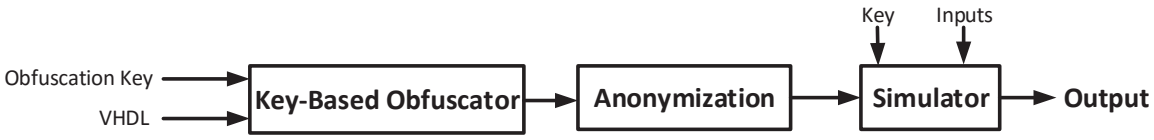

Figure 3.2: Proposed Obfuscation and Anonymization Flow

Next, we perform anonymization as design sanitization for privacy protection. In our work, it is a process of renaming the identifiers so that it becomes difficult for the attacker to find out which of the identifiers (ports in VHDL) are the key and inputs/outputs. The previous key-based obfuscated VHDL is now anonymized with random identifiers of a certain length. The key during obfuscation is provided by the IP designer in boolean format and hence, the generated VHDL of the first step changes dynamically. The obfuscated behavioral model can be simulated correctly only when the valid key is applied.

Unlike, any other obfuscator, we are using a key-based obfuscator which uses designer specified key length and are also dynamic in nature. That is, once the designer embeds key in the behavioral VHDL and generates the obfuscated model, model will be functionally correct only for that particular key. Any other key will result in wrong output. This way the VHDL model is secure. To keep the key and input-output identifier indistinguishable, obfuscated model is anonymized further. Hence, the hardware is protected from reverse engineering. The two steps are explained in detail in Sections 3.1.1 and 3.1.2.

```
Data: V= Original VHDL, L= Key-length, Keys
Result: Obfuscated VHDL
// Step 1 - Identifying 0s and 1s in keys
for bits k_i in keys do
 |  separate 0's and 1's from the keys
end
Add keys as input in port declaration
// Step 2 - Parse the VHDL code
for line in V do
   I ← random number generation from the respected 0's
    and 1's in the keys
   //I value changes randomly each time
   if line = assignment_statement then
      //Modify the statement by adding boolean
       operations
      variable = (((expression) XOR K1(I))AND(K2(I)
       OR K2(I)))XOR K1(I);
   end
   if line = conditional_statement then
      Add "IF(K1(I) = 1) then
      set_of_instructions
      END IF;"
   end
end
```

Figure 3.3: Key Based Obfuscation Algorithm

3.1.1 Key Based Obfuscation Step

To obfuscate a VHDL model, we perform XOR and AND operations on keys and embed them into the assignment statements. Figure 3.3 shows the algorithm to perform such obfuscation. The algorithm takes the original VHDL, key length and the key as inputs and produces the obfuscated VHDL as the output. It performs obfuscation in two stages. First, the 0's and 1's are separated from the key. Keys are then added as inputs in port declaration of the original VHDL along with a few dummy keys to disguise the original key from the attacker for a fixed key length. Step 1 in the algorithm takes constant time to complete due to the for loop. Step 2 needs $O(n)$ time where n is the lines in the model. Hence, the total time complexity of the algorithm is $O(n)$.

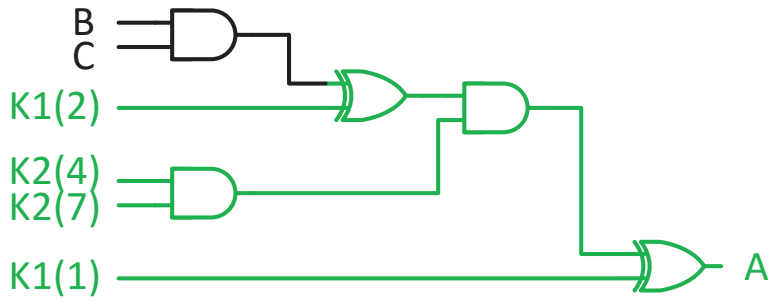

Figure 3.4: Obfuscation of OR Gate

While parsing the original VHDL file, if there is an assignment statement, it is concatenated with a series of boolean operations. For example, if "C=A" is an assignment statement, the corresponding obfuscated version will be "C = ((A XOR $K1(1)$) AND $K2(6)$) XOR $K1(4)$". The particular index of the key will be random (0 or 1) depending on the key value. Figure 3.4 shows an obfuscated form of a two-input "AND" gate. Here, the expression (A = B AND C) can be transformed to "A = ((((B AND C) XOR $K1(2)$) AND ($K2(4)$ AND $K2(7)$)) XOR $K1(1)$". If B

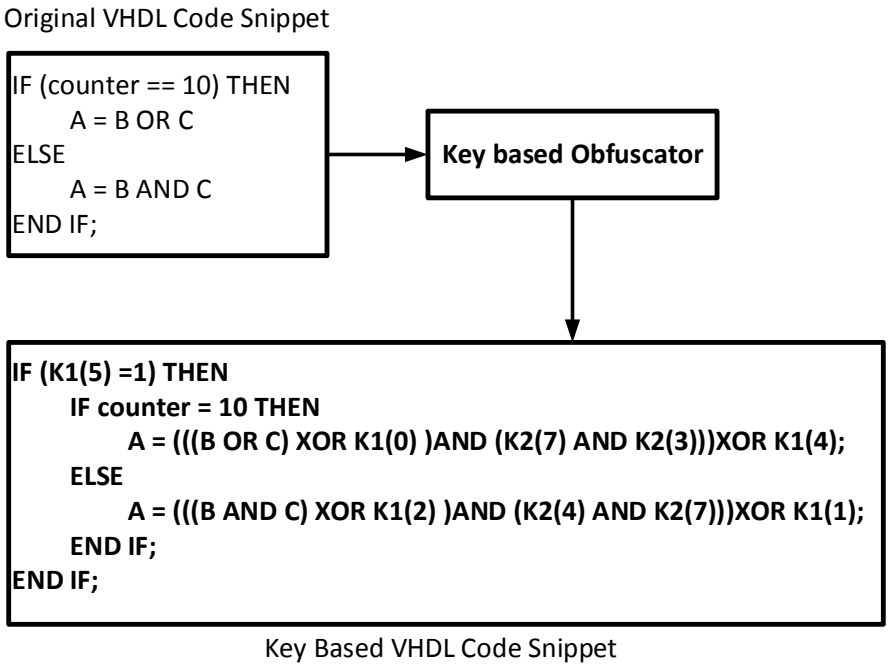

Figure 3.5: Obfuscation Step

and C are n-bit identifiers, random keys are concatenated to get the required n-bit and appended to the regular expression of the gate.

If a conditional statement is found, we insert another conditional statement to check the value of a particular index in given key sequence before the current conditional statement. For example, "if $(C == A)$ then $C = 0$" is the conditional statement, it is obfuscated to "if$(K1(5) == 1)$ then if $(C == A)$ then $C = 0$" by inserting an additional if clause. Figure 3.5 shows an example of obfuscation for a conditional code block. As the particular value corresponding to the index of any key in input is unknown, the attacker has to enumerate all possible key values and the positional value as well.

3.1.2 Anonymization Step

Anonymization is done such that a functional equivalent source is compiled and formulated for the given source, which is highly difficult or say impossible for humans to understand by just viewing the source file and is extremely difficult to reverse-engineer through different on-line and off-line tools. We carry the anonymization step with specified length of random string m. In this process, the identifiers are renamed and hence, the difficulty of reverse engineering increases linearly with the identifier length.

```
Data: V= Obfuscated VHDL,
      m=anonymized-string-length
Result: Key Based Obfuscated and Anonymised VHDL
// Step 1 - Create a table for identifiers
Table[id, old_value, new_value]
// Step-2 - Search for an identifier
for identified iden in V do
    if identifier = (old_value) in table then
        iden = new_value;
    else
        Add Table [id, iden, random(id, m)]
        iden = new_value;
    end
end
```

Figure 3.6: Anonymization Algorithm

The pseudo-code of the anonymization algorithm is shown in Figure 3.6. It takes the obfuscated VHDL from Section 3.1.1 as input and generates an anonymized VHDL model as output. The algorithm performs anonymization in two stages. First, a 2-dimensional table is created for the identifiers. While parsing the obfuscated VHDL, if an identifier is found in the table, the anonymised value is taken from the table, else a random string is generated for that identifier and inserted to the table. For example, "A : IN BIT" can be anonymized as "hnjksgthenbd : IN BIT" by an m-bit random string.

The anonymization algorithm takes $O(1)$ time for step 1 as the size of the addition of identifier to the table can be performed in constant time. Step 2 has a time complexity of $O(n)$ where n is the number of identifiers in the model. Hence, the total time complexity of the algorithm is $O(n)$. Figure 3.7 shows anonymization of the key-based obfuscated code from Figure 3.5.

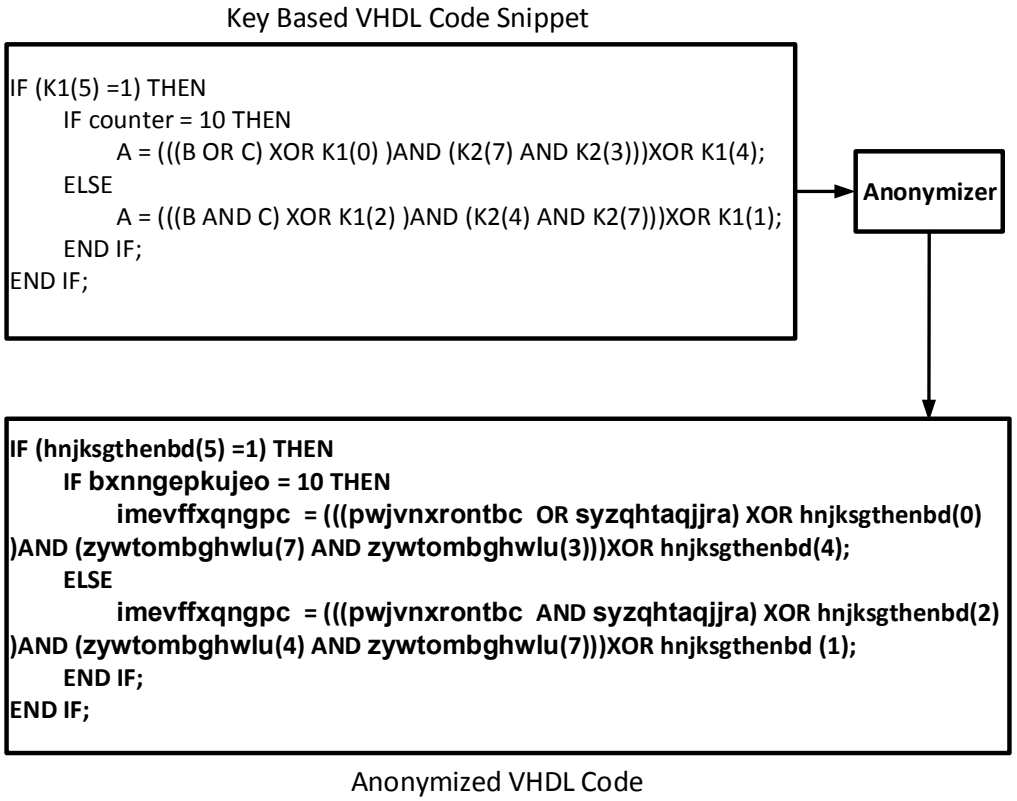

Figure 3.7: Anonymization Step

3.2 Difficulty to Reverse Engineer

We evaluate the resilience of the proposed obfuscation and anonymization technique. Resilience determines the withstanding capability of obfuscated and anonymized VHDL against attacker attempt. If m is the number of input bits and k is the number of bits in obfuscation key

used in the model, the attacker has to try different combinations to know the key. The search is augmented by different ways of arranging k keys into $m+k$ bit registers. The resilience of obfuscated VHDL model can be given by the following equation:

$$P(m,k) = \frac{1}{\binom{m+k}{k} 2^k} \tag{3.1}$$

Table 3.1 shows the resilience value for different benchmarks with various key bits. Lower value of P increases the difficulty for the attacker to correctly guess the the keys and their location in registers. With a larger number of key size and number of input bits, the attackers need more resources (computational time and effort) to reverse engineer the design.

Table 3.1: Resilience Table (see Equation 3.1)

Design	Original VHDL k=0		Obfuscated VHDL					
			8 bit		16 bit		32 bit	
	m	P	m	P	m	P	m	P
ellip8	64	1	96	7.83E-36	128	2.26E-57	192	4.05E-91
ellip16	128	1	160	5.01E-44	192	7.47E-72	256	5.09E-115
ellip32	256	1	288	7.30E-53	320	2.79E-88	384	4.81E144
fir8	80	1	112	2.22E-38	144	8.87E-62	208	3.36E-98
fir16	160	1	192	8.14E-47	224	5.79E-77	288	7.42E-124
fir32	320	1	352	8.49E-56	384	7.20E-94	448	2.79E-154
latt8	64	1	96	7.83E-36	128	2.26E-57	192	4.05E-91
latt16	128	1	160	5.01E-44	192	7.47E-72	256	5.09E-115
latt32	256	1	288	7.30E-53	320	2.79E-88	384	4.81E-144

3.3 Chapter Summary

We presented in detail the proposed approach of the obfuscation and anonymization methods for the behavior VHDL models. The two main steps of the process flow are explained with algorithms and examples. The resilience is calculated for the original and the obfuscated VHDL models to show the difficulty level for doing reverse engineering. The overhead is calculated by measuring the simulation time for all of the VHDL models.

CHAPTER 4: EXPERIMENTAL RESULTS

Nine datapath intensive benchmarks are modified by using different key lengths. The behavioral description of these benchmarks are written in VHDL and have entity, port declarations, signals, process(s), variable declarations, assignment statements, conditional evaluations, loop statements, and wait statements. Table 4.1 shows these benchmark models number of input and output ports, number of variables and number of assignment statements present in the VHDL model.

The obfuscation of these benchmarks is done by using three different key lengths which are, 8, 16, and 32. The test-benches created for these benchmarks along with the original file contain 5, 1K, and 10K test vectors. These files are simulated to get the output and simulation time is calculated. Table 4.2 shows the time taken to simulate the test benches having 5 test vectors with different key values. Table 4.3 shows the time taken to simulate the test benches having 1K test vectors with different key values. Table 4.4 shows the time taken to simulate the test benches having 10K test vectors with different key values.

Table 4.1: Benchmark Details

Design	Ports		Variables	Statements
	# inputs	# outputs		
ellip	8	8	29	37
fir	10	1	8	9
latt	8	3	12	15

Table 4.2: Simulation Time for 5 Test Vector Sequence

Design	Original	User Simulation Time (sec)					
	(sec)	8-bit	Overhead	16-bit	Overhead	32-bit	Overhead
ellip8	0.72	074	0.02	0.76	0.04	0.77	0.05
ellip16	0.72	0.77	0.05	0.78	0.06	0.79	0.07
ellip32	0.71	0.81	0.10	0.83	0.12	0.84	0.13
fir8	0.69	0.71	0.02	0.71	0.02	0.74	0.05
fir16	0.72	0.74	0.02	0.75	0.03	0.75	0.03
fir32	0.73	0.74	0.01	0.75	0.02	0.75	0.02
latt8	0.70	0.72	0.02	0.73	0.03	0.74	0.04
latt16	0.71	0.75	0.04	0.75	0.04	0.75	0.04
latt32	0.72	0.75	0.03	0.76	0.04	0.78	0.06

Table 4.3: Simulation Time for 1K Test Vector Sequence

Design	Original	User Simulation Time (sec)					
	(sec)	8-bit	Overhead	16-bit	Overhead	32-bit	Overhead
ellip8	2.29	3.06	0.77	3.11	0.82	3.28	0.99
ellip16	2.49	3.22	0.73	3.28	0.79	3.39	0.90
ellip32	2.65	3.51	0.86	3.61	0.96	3.72	1.07
fir8	2.63	3.43	0.80	3.49	0.86	3.62	0.99
fir16	2.77	3.57	0.80	3.63	0.86	3.78	1.01
fir32	3.09	3.89	0.80	3.92	0.83	4.03	0.94
latt8	2.26	3.02	0.76	3.11	0.85	3.25	0.99
latt16	2.43	3.17	0.74	3.27	0.84	3.36	0.93
latt32	2.62	3.42	0.80	3.55	0.93	3.63	1.01

Table 4.4: Simulation Time for 10K Test Vector Sequence

Design	Original (sec)	User Simulation Time (sec)					
		8-bit	Overhead	16-bit	Overhead	32-bit	Overhead
ellip8	7.54	10.98	3.44	11.24	3.70	11.66	4.12
ellip16	8.05	11.50	3.45	11.74	3.69	12.16	4.11
ellip32	8.94	12.52	3.58	12.78	3.84	3.14	4.20
fir8	9.04	12.63	3.59	12.90	3.86	13.45	4.41
fir16	9.65	13.16	3.51	13.40	3.75	13.80	4.15
fir32	10.56	14.18	3.61	14.34	3.77	14.79	4.22
latt8	7.53	10.91	3.38	11.07	3.54	11.51	3.98
latt16	8.04	11.37	3.33	11.58	3.54	11.99	3.95
latt32	8.82	12.27	3.45	12.63	3.81	12.93	4.11

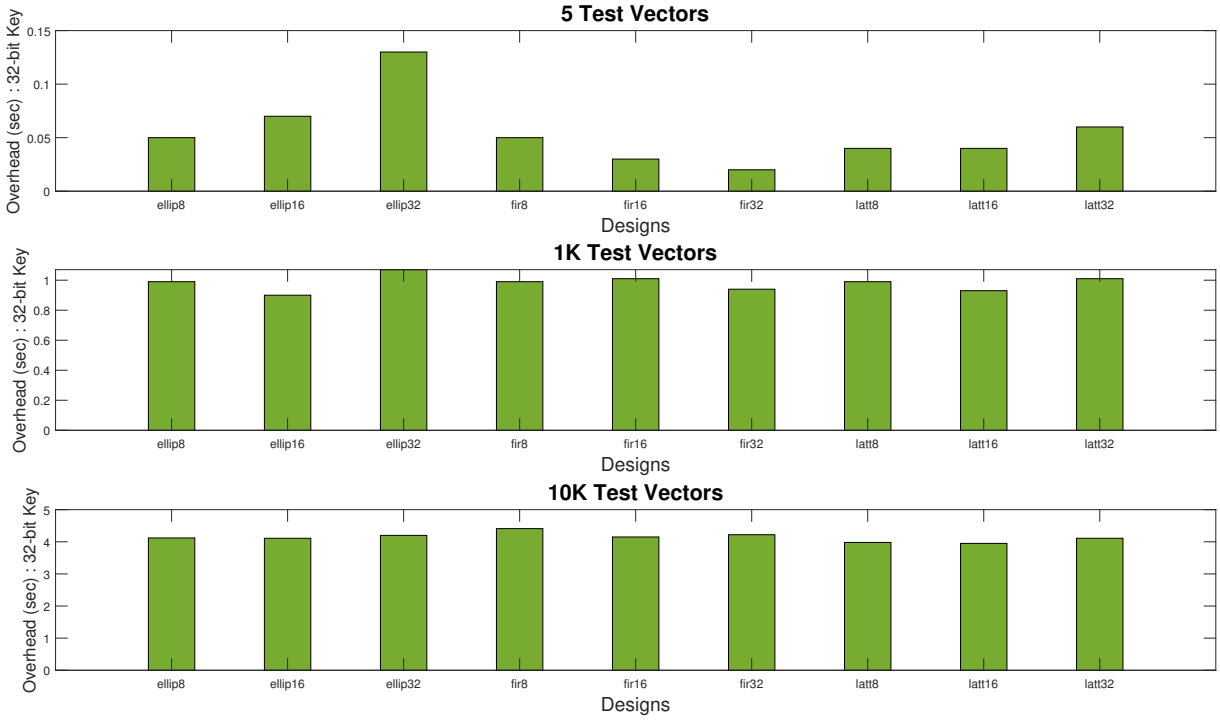

Figure 4.1: Overhead of Designs for 32-bit Key

It is observed that the user time difference between the original and the key based obfuscated VHDL models is only a few seconds. The average overhead for 5, 1000, and 10K test vectors is 0.04, 0.88, and 3.78 seconds respectively. Figure 4.1 shows the overhead for all designs with 5, 1K and 10K test vectors and 32-bit key length.

CHAPTER 5: CONCLUSION

The main agenda of our work is to mitigate reverse engineering of the IP based IC designs for which, we have implemented an obfuscation tool with lex and yacc for behavior VHDL. We successfully demonstrated the key-based obfuscation and anonymization of the behavioral VHDL to protect from malicious attacks and has reduced the probability of reverse engineering by brute-force attacks and is quantified by a resilience metric. The proposed work had given very less overhead, few seconds, when experimented with nine data path intensive design benchmarks.

REFERENCES

[1] N. Veeranna and B. C. Schafer. Efficient behavioral intellectual properties source code obfuscation for high-level synthesis. In *2017 18th IEEE Latin American Test Symposium (LATS)*, pages 1–6, March 2017.

[2] E. Castillo, U. Meyer-Baese, A. Garcia, L. Parrilla, and A. Lloris. IPP@HDL: Efficient intellectual property protection scheme for IP cores. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(5):578–591, May 2007.

[3] S. A. Islam and S. Katkoori. High-level synthesis of key based obfuscated RTL datapaths. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 407–412, March 2018.

[4] R. S. Chakraborty and S. Bhunia. RTL hardware IP protection using key-based control and data flow obfuscation. In *2010 23rd International Conference on VLSI Design*, pages 405–410, Jan 2010.

[5] S. Koteshwara, C. H. Kim, and K. K. Parhi. Key-based dynamic functional obfuscation of integrated circuits using sequentially triggered mode-based design. *IEEE Transactions on Information Forensics and Security*, 13(1):79–93, Jan 2018.

[6] R. S. Chakraborty and S. Bhunia. HARPOON: An obfuscation-based SoC design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, Oct 2009.

[7] R. S. Chakraborty and S. Bhunia. Security against hardware trojan through a novel application of design obfuscation. In *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, pages 113–116, Nov 2009.

[8] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE Design Test of Computers*, 27(1):10–25, Jan 2010.

[9] D. Li, W. Liu, X. Zou, and Z. Liu. Hardware IP protection through gate-level obfuscation. In *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, pages 186–193, Aug 2015.

[10] J. Dofe and Q. Yu. Novel dynamic state-deflection method for gate-level design obfuscation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):273–285, Feb 2018.

[11] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu. Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques. *IEEE Transactions on Information Forensics and Security*, 12(1):64–77, Jan 2017.

[12] Z. Tang, X. Chen, D. Fang, and F. Chen. Research on java software protection with the obfuscation in identifier renaming. In *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pages 1067–1071, Dec 2009.

ABOUT THE AUTHOR

Balausha Varshini Kandikonda completed her Bachelors of Technology in Computer Science and Engineering from Jawaharlal Nehru Technological University (JNTU), Hyderabad, India in 2015. She worked as a Systems Engineer at PurpleTalk Inc., India and has been awarded with "SPARK" (Best Employer) for her outstanding contribution towards the project. In 2016, she joined Department of Computer Science and Engineering at University of South Florida, Tampa to pursue her Master's degree. She worked under the supervision of Dr. Srinivas Katkoori and completed her thesis. She is always motivated to take up any new and challenging tasks.