Obfuscation of Digital Systems using Isomorphic Cells and Split Fabrication

Mohamad A. Masoud
Computer and Systems Engineering
Ain Shams University
Cairo, Egypt
Email: mohamad.abdelaziz@eng.asu.edu.eg

Yousra Alkabani
Computer and Systems Engineering
Ain Shams University
Cairo, Egypt
Email: yousra.alkabani@eng.asu.edu.eg

M. Watheq El-Kharashi
Computer and Systems Engineering
Ain Shams University
Cairo, Egypt
Email: watheq.elkharashi@eng.asu.edu.eg

***Abstract*—Hardware security has become a vital issue, and has recently gained increasing attention. In this paper, we propose obfuscation algorithms that improve the security of digital systems, and protect them against reverse engineering attempts. Metrics for measuring the obfuscation level of digital designs that implement our approaches are also introduced. The introduced algorithms are mainly based on the concepts of split fabrication, isomorphic gates and optimized isomorphic gates, and a new concept we introduce based on the inclination of logic gate outputs towards a specific output (1 or 0). Experimental results show improvements in the figures that represent the security of designs based on the introduced metrics.**

I. Introduction

Digital systems are widely used in various applications. Due to the complex flow of production of digital systems, many companies prefer to get their chips produced at third-party foundries. It is possible for untrusted foundries to insert malicious circuits in these designs to disrupt the functionality of the original circuit, or get unauthorized access to confidential information. It is difficult to detect these circuits using traditional methods like testing due to the malicious circuits being triggered under rare conditions. Therefore, it is necessary to provide solutions to this problem. Split fabrication involves splitting designs into two separate entities: A complex entity, which usually contains the main algorithm, and a relatively simple entity. It is effective for obfuscating designs to make them more resistant to reverse-engineering attempts. This method can make use of isomorphic cells, which are special types of logic cells that can be configured to implement various functionalities. After the designs are split, the complex part is fabricated normally, whereas a less advanced trusted foundry implements the simple part. The key is that the simpler part would contain the connections which represent the configurations of designs without which the design's function would be challenging to figure out, and that would protect the designs from being tampered with, while protecting the foundries' industry secrets. Obfuscation causes reverse-engineering attempts to be so costly that anyone with malicious intent would refrain from such attempts since the gain would be much less than the necessary effort and cost. We introduce some algorithms for obfuscation, and metrics that measure the degree of obfuscation of digital systems that are secured using the proposed algorithms. Experimental results show improvements in the values of the proposed metrics upon usage of the proposed obfuscation algorithms. Section II gives a summary of some related work on similar topics. Section III provides the definitions of the concepts used in the paper, and a theoretical background on the proposed algorithms and metrics. Section IV illustrates our proposed algorithms using pseudo code. Section V discusses the environment setup, as well as some information on the tools used. Section VI shows some experimental results.

II. Related Work

Valamehr et al. [6] introduced a security monitoring method using split fabrication for improving security in 3D ICs. The design was divided into two parts, a complex computational part that has to be manufactured by an advanced foundry, which is usually untrusted, and a relatively simple control part that monitors internal signals in the first part. The control part can be manufactured by a less advanced foundry since it is relatively simple. This also provides more choices as it is easier to find a trusted foundry that way since a less advanced level of technology is required. This method relies on monitoring designs at runtime to detect unexpected behavior due to malicious changes that could be introduced by an untrusted foundry. Imeson et al. [5] introduced a design partitioning method that targets the obfuscation of designs for 2.5D/3D integrated circuits. Jagasivamani et al. [4] introduced several obfuscation techniques based on split fabrication, as well as metrics to measure the degree of obfuscation of digital systems. These techniques include limiting the gate types used in implementing digital systems, inserting dummy cells in designs, and using isomorphic gates, which can be configured by trusted foundries to implement specific logic functions. The metrics also include system entropy, which measures the amount of information present. Yousra Alkabani [1] introduced an optimized model for isomorphic gates, where these gates are restricted to implement specific logic functions in exchange for optimizations to reduce the overhead introduced by using isomorphic gates. F. Koushanfar et al. [13] and A. Sengupta et al. [7] introduced protection techniques based on the idea of watermarking designs in order to prove ownership and protect the confidentiality of hardware designs. Techniques based on the physical properties of designs have been used to detect hardware trojans. For example, Y. Jin et al. [8] used delay analysis in order to detect malicious trojans. A. Vijayakumar et al.[12] investigated obfuscation techniques based on alterations of circuit elements that are challenging to observe. Vaidyanathan et al. [11] introduced obfuscation techniques for analog IPs to protect them against potential tampering by untrusted foundries. A. Al-Anwar et

al. introduced techniques for protecting designs that include third party IPs against hardware trojans without using a golden design a reference [9][10]. Otero et al. [3] introduced a special cell library and a tool for automating the obfuscation of layouts for split fabrication. Dummy transistors were added for equalizing the number of PMOS and NMOS transistors, and some constraints on the placement of cells were applied. Xiao et al. [2] suggested combining split fabrication with built-in self authentication.

III. Theoretical Background

This section provides a background necessary to understand the introduced algorithm and metrics.

A. Isomorphic Cells

Jagasivamani et al. [4] used isomorphic gates to obfuscate the real type of cells in the designs fabricated by an untrusted foundry. An isomorphic gate implementing logic gates with 2 inputs could be represented using a 4x1 multiplexer. The gate's inputs are placed on the multiplexer's selection lines, while values that correspond to the truth table of the logic gate that needs to be implemented are placed on the multiplexer's other inputs. i.e, a 2-input isomorphic gate has 6 inputs and 1 output. These isomorphic gates are used in our approach to hide the true function of gates that give away highly critical information on the design by replacing those gates with isomorphic gates, and applying the connections needed to configure the function later on at the trusted foundry, therefore hiding the true functionality of the gate, and protecting the design by obfuscating critical information.

B. Optimized Isomorphic Gates

Yousra Alkabani [1] introduced an improved model of isomorphic gates. By restricting the functions that can be represented by isomorphic gates to a specific set of functions, it is possible to minimize the number of configuration wires that need to be connected by the trusted foundry in order to implement the required functionality without revealing any information on the design. Therefore, this optimized model makes it easier for trusted foundries to do their work, while retaining the security of designs.

C. Truth Table Inclination (TTI)

All digital designs consist of combinations of logic gates connected to each other. With the exception of XOR, XNOR, and NOT gates, all logic gates are inherently "inclined" towards specific outputs. For example, AND gates are more inclined towards an output of "0" with a probability of 75% for a 2-input AND gate, and OR gates are more inclined towards an output of "1", with the probability of 75% for a 2-input OR gate. Consequently, digital designs inherit this inclination property from the gates inside them. i.e, digital designs are inherently inclined towards specific outputs depending on the gates that constitute them, and depending on how these gates are connected to each other. This abstract concept can be used to find situations where a specific gate (or a few gates) in a design can have a deep effect on the functionality that can give away crucial information on how the circuit works, and consequently unintentionally help attackers find out details on the design, and facilitate reverse engineering attempts. Figure 1 shows an exemplary circuit that shows this concept. GATE2 is an AND gate that greatly influences the output of the design. i.e, if IN3 is set to 0, the output is definitely going to be 0 regardless of any other inputs. Otherwise, the output would either be 0 or 1 depending on IN0, IN1, and IN2. GATE2 ensures that at least 50% of the input combinations lead to a 0 output, and the other gates might produce an output of 1 or 0 if IN3 is set to 1. By inspecting the circuit's truth table, it can be deduced that the circuit favors an output of 0 with a percentage of 75% (12 out of 16 combinations). This inclination can be partially justified by the presence of GATE2, which can be interpreted as a switch to enable or disable the output of the remaining part of the circuit. Also, IN3 can be interpreted as an enable signal which controls the output of the circuit. Such information is critical, and can help an attacker find out part of the functionality of this circuit. The truth table of this circuit shows this fact by clarifying the inclination towards an output of 0. This is the main concept used in our algorithm. However, this inclination is calculated for the whole circuit. Some gates could neutralize the effects of others in the overall truth table, and therefore the effects would not be completely clear just by inspecting the overall truth table of the circuit. That is why this concept is extended to cover the exact effect of each gate in the design in its own position in section III-H.

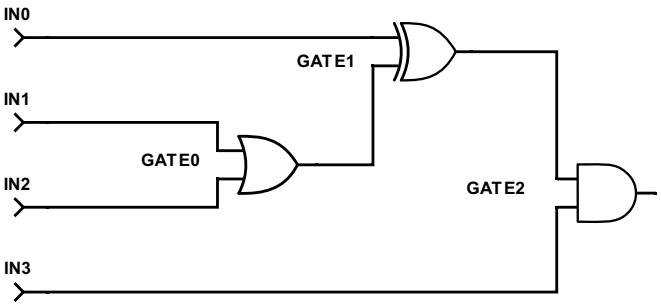

Fig. 1: Truth Table Inclination Example

D. Cluster

We use a method of clustering that involves grouping related logic gates together. This has two benefits: Splitting groups of gates from each other, therefore supporting split fabrication by severing the connections between different clusters, and reconnecting them at the trusted foundry. This provides extra security to the whole design, while maintaining the relation between the gates of each cluster. The other benefit is limiting the required processing per cluster since our algorithm relies heavily on truth table calculations which scale exponentially, and can be impractical for large designs due to the huge amount of required processing. The clustering technique we use here is based on depth first search. This is to ensure that each cluster consists of a connected set of gates to be able to apply our truth table analysis illustrated in section III-C.

E. Isomorphic Replacement

Our algorithm is based on replacing some gates that fulfill specific conditions with isomorphic gates. We use isomorphic replacement as a term that means the replacement of these specific gates with isomorphic gates. Jagasivamani et al. [4] proposed replacing all gates in the design with isomorphic gates. Although this is excellent from a security-oriented perspective, this is impractical since this implies a huge overhead in terms of resources, design area, timing parameters, and cost. Our algorithm provides a compromise that replaces only the gates with the highest impact on security (from the algorithm's viewpoint), while limiting the overhead in terms of design area, cost, and timing parameters.

F. Replacement Threshold

Replacement threshold is a parameter used in our algorithm that defines the minimum size of a cluster to process it. i.e, if a cluster's size is less than the defined replacement threshold, this cluster is ignored and not considered when applying isomorphic replacement.

G. Deviation Threshold

Deviation threshold is a parameter used in our algorithm that defines the minimum deviation from the ideal distribution of a truth table. Our view of an ideal truth table distribution is that the number of combinations that trigger an output of 0 is equal to the number of combinations that trigger an output of 1. If any gate in a processed cluster causes a deviation greater than or equal to the defined deviation threshold, that gate is replaced with an isomorphic gate.

H. Gate Normalization

In order to study the exact effect of each gate on the truth table of the whole cluster, we introduce **Gate Normalization**, which is a concept that utilizes the fact that some logic gates like XOR and XNOR have an ideally distributed truth table. So, in order to measure the effect of each gate including its position in the cluster as well as its type, each gate in a cluster is replaced by an XOR gate separately. i.e, we iterate over the whole cluster N times, where N represents the total number of gates in the cluster, and then during each iteration we replace 1 gate with an XOR gate, and calculate the truth table output for all binary combinations. Afterwards, we compare each of these truth tables with the truth table of the original circuit without replacement. Finally, we apply the filtering parameter **Deviation Threshold** to choose the gates that shall be replaced with isomorphic gates. Gates whose normalized truth tables deviate by values greater than or equal to the allowed deviation threshold are replaced. Deviations from the original truth table are calculated by finding out the number of combinations that produce an output that does not match the output of the same binary combination for the original circuit.

I. Entropy

Jagasivamani et al. [4] proposed using entropy as a metric for measuring the amount of information present. A uniform design utilizing a fewer number of types of cells for implementing all required logic functions results in lower entropy and yields less information on the design's functionality, which in turn increases the effort needed for reverse engineering. Entropy is defined as

$$E = -\sum_{i=1}^{N} p_i log(p_i) \tag{1}$$

Where p_i is the percentage that each gate type represents of the overall number of gates in the design, and N represents the overall number of gates in the design. We use entropy as one of our metrics to measure the amount of information in the design before isomorphic replacement. Moreover, we extend the concept of entropy to define a more specialized entropy that can be used to measure the amount of information in designs after applying our truth table inclination method as shown in section III-J.

J. Isomorphic Entropy

Isomorphic entropy provides a metric for measuring the amount of obfuscation provided by isomorphic replacement. Isomorphic entropy is calculated per cluster and is defined as

$$E_{iso} = E * (1 - p_{isomorphic}) * (1 + log_{10} \frac{1}{N_{types}}) \tag{2}$$

Where $p_{isomorphic}$ is the percentage of isomorphic gates of the total number of gates in the cluster, N_{types} is the total number of types of gates used in the design, and E is the entropy. According to equation 2, the higher the value of $p_{isomorphic}$, the lower the isomorphic entropy and the lower the amount of information revealed. This is consistent with the fact that isomorphic gates hide information from attackers. If $p_{isomorphic}$ equals 1, then the isomorphic entropy is 0, which means that it is very difficult to find out any information on the design's functionality. Also, the number of gate types represented by N_{types} affects the value of isomorphic entropy. The higher the number of gate types used in the design, the lower the isomorphic entropy after applying isomorphic replacement. This is justified by the fact that the attacker would normally assume that an isomorphic gate actually implements one of the gate types present in the design only. Consequently, the higher the number of gate types, the higher the number of possibilities that the attacker needs to try, and the more difficult any reverse engineering attempt would be.

K. Cluster Forms

Isomorphic gates can be configured to implement various logic functionalities. Each gate that is replaced by an isomorphic gate in a cluster creates various possibilities for that cluster. i.e, that gate could take up several forms, and would cause the whole cluster to have several possible forms. By increasing the number of isomorphic gates in a cluster, the number of forms increases exponentially, since there are several combinations for all possibilities of each gate. Also, the higher the number of gate types in the design, the higher the number of forms each isomorphic gate would be expected to take, and the higher the number of combinations. The number of cluster forms actually grows so rapidly that reverse-engineering attempts would be extremely difficult even at

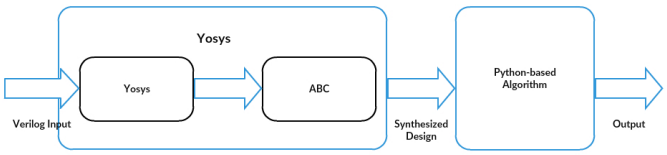

Fig. 2: Algorithm Flow

relatively smaller numbers of isomorphic gates per cluster. The number of cluster forms is defined by equation 3.

$$NumForms = NumGateTypes^{NumIsomorphicGatesInCluster} \quad (3)$$

IV. Algorithms

A. DFS-based Isomorphic Replacement

Based on the concepts of Truth Table Inclincation, Split Fabrication, Isomorphic Gates, and Isomorphic Replacement, we propose the following depth-first clustering algorithm. Basically, the algorithm creates clusters based on depth-first search, and then evaluates the requirements of isomorphic replacement for each cluster. If the requirements are met, then the algorithm applies isomorphic replacement to all gates matching the configured conditions. Figure 2 shows the flow used to implement this algorithm. Algorithm 1 shows a pseudo code for our replacement algorithm. Lines 6 through 10 get a node out of the stack, and if it has not been visited before, it is added to the current cluster. Otherwise, a new cluster is created, and the existing cluster is processed and added to the list of clusters. Lines 15 through 21 evaluate all nodes that follow the current node. If one or more nodes exist that are not outputs of the circuit, they are added to the cluster. Otherwise, they are ignored, and a new cluster is created, while the existing cluster is processed and added to the list of clusters. If the cluster's size is smaller than the replacement threshold, the cluster is ignored. Lines 29 through 32 calculate the difference in truth table output between the original and normalized cluster, and replaces the gate if the difference is greater than the deviation threshold.

B. High Fan-Out Articulation Points

In this section, we propose a theory for another replacement algorithm that could be used to improve the security of digital systems. Using the concept of articulation points, it is possible to find pivotal gates in designs without which the design would be disconnected. In order to filter out low-impact articulation points, we would use a threshold for the required fan-out above which a gate would be replaced. For example, if we used a threshold of 4, and there are articulation points whose fan-out is less than 4, then they are not replaced. The result would be improved security at low costs of isomorphic replacement since only the gates with the highest impact would be replaced. The reason why articulation points are chosen is that they relay critical information to the remaining parts of the circuit, and high fan-out is chosen in order to confuse attackers by increasing the amount of receivers of the obfuscated information. i.e, if a gate's output is connected to 8 different gates, then obfuscating that gate would affect all 8 gates that follow, which increases the degree of obfuscation since it would be

Algorithm 1 DFS-based Isomorphic Replacement

```
1:  procedure ISOMORPHIC REPLACEMENT
2:      stack ← [1'b0, 1'b1]
3:      clstList = list
4:      gateClst = list
5:      while stack not empty do
6:          elem = stack.pop()
7:          if elem visited previously then
8:              pClust = processCluster(gateCluster)
9:              clstList.append(pClust); gateClst = list
10:             continue
11:         if elem not visited then
12:             Mark elem as visited
13:         if elem is neither input nor output then
14:             gateClst.append(elem)
15:         unVisited = unvisited nodes following elem
16:         if unVisited contains no gates then
17:             pClust = processCluster(gateCluster)
18:             clstList.append(pClust); gateClst = list
19:             continue
20:         else
21:             stack.append(all nodes in unVisited)
22: procedure PROCESSCLUSTER(clstr)
23:     replacedClst = list
24:     OrigDev = number of ones in original truth table
25:     for all idx, node in enumerate(clstr) do
26:         clstCpy = copy of clstr with XOR at idx
27:         dev = number of ones in truth table
28:         replaceGate = False
29:         if OrigDev − dev > deviationThreshold then
30:             replaceGate = True
31:         replacedClst.append((cluster[idx], replaceGate))
32:     return replacedClst
```

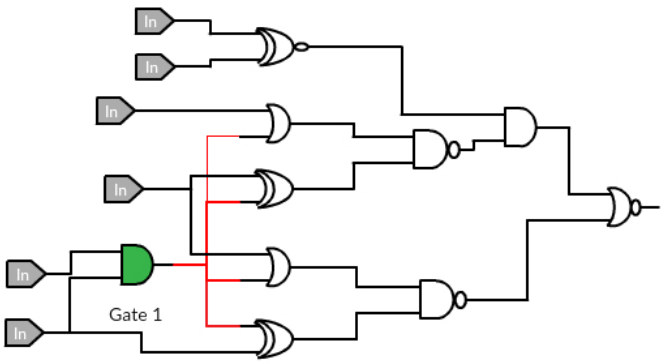

Fig. 3: Exemplary High Fan-Out Articulation Point

difficult to figure out how the following gates work without knowing the functionality of this articulation point. Figure 3 shows an example of this. Also, the idea of using high fan-out gates on its own regardless of articulation points could be quite effective.

V. Environment Setup

We used a machine running Windows 8.1 with Intel Core i7-3632QM CPU, as well as Ubuntu 14.04 on a virtual

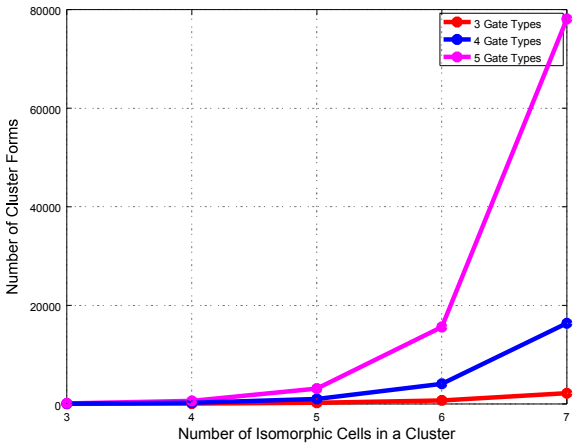

Fig. 4: Number of cluster forms vs number of isomorphic gates in it for different values of types of gates in design.

machine. Yosys [14], which is a framework for Verilog RTL synthesis was used to synthesize our Verilog-based designs. Yosys is based on ABC [15], which is a growing system for system synthesis. We also used Python 3.5 to implement our algorithms and metrics.

VI. Experimental Results

We consider 3 different designs as case studies. an i2c slave, a DES model, and an 8-bit multiplier. Tables I, II, and III show the results of applying DFS-based Isomorphic Replacement to an i2c slave, a DES model, and an 8-bit multiplier respectively for different values of replacement threshold and deviation threshold, where replacement threshold is represented by RT, and deviation threshold is represented by DT. Also, Figure 4 represents a graph that illustrates how the number of cluster forms scales with the number of isomorphic gates in the cluster. It is clear from the graph that the number of cluster forms grows exponentially, and quickly becomes so high that reverse-engineering attempts based on signal probabilities can be very time-consuming. In the provided tables, we refer to entropy as E, isomorphic entropy as IE, total number of isomorphic gates used as IG, the overall number of gates as NG, the number of gate types used to implement the design as GT, the maximum number of isomorphic gates per cluster as MIGC, the maximum number of possible cluster forms as MCF, the average number of isomorphic gates per cluster as AIGC, and the average number of cluster forms as ACF.

VII. Conclusion

Isomorphic replacement is a powerful method of obfuscating digital systems. Replacing all cells in a design would theoretically provide a huge improvement to security, but at large costs in terms of design area and signal delays. DFS-based isomorphic replacement provides a compromise between security and cost that targets critical gates that give off important information to improve security, while avoiding the overhead of replacing all cells. Experimental results show improvements in security represented by isomorphic entropy in values as low as 0.03977, and in number of cluster forms in values as large as 1594323, which render reverse engineering attempts based on signal probabilities very difficult.

TABLE I: Results of applying DFS-based Isomorphic Replacement on an i2c slave with max cluster size = 13

	RT=1 DT=1	RT=5 DT=1	RT=2 DT=7	RT=2 DT=1
E	0.4213	0.4213	0.4213	0.4213
IE	0.06344	0.1209	0.14819	0.08652
IG	707	448	325	603
NG	993	993	993	993
GT	3	3	3	3
MIGC	12	12	10	12
MCF	531441	531441	59049	531441
AIGC	2.3333	1.47854	1.0726	1.99009
ACF	12.9802	5.07512	3.2491	8.90263

TABLE II: Results of applying DFS-based Isomorphic Replacement on a DES model with max cluster size = 13

	RT=1 DT=1	RT=5 DT=1	RT=2 DT=7	RT=2 DT=1
E	0.36423	0.36423	0.36423	0.36423
IE	0.05907	0.13602	0.13949	0.07743
IG	3785	1568	1468	3256
NG	5487	5487	5487	5487
GT	3	3	3	3
MIGC	13	13	11	13
MCF	1594323	1594323	177147	1594323
AIGC	2.34655	0.9721	0.9101	2.01859
ACF	13.17022	2.90944	2.71788	9.18578

References

[1] Yousra Alkabani, "Hardware security and split fabrication," Design & Test Symposium (IDT), 2016 11th International

[2] K. Xiao and D. Forte and M. M. Tehranipoor, "Efficient and secure split manufacturing via obfuscated built-in self-authentication," 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)

[3] C. T. O. Otero and J. Tse and R. Karmazin and B. Hill and R. Manohar, "Automatic obfuscated cell layout for trusted split-foundry design," 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)

[4] Meenatchi Jagasivamani and Peter Gadfort and Michel Sika and Michael Bajura and Michael Fritze, "Split-Fabrication Obfuscation: Metrics and Techniques," 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)

[5] Frank Imeson and Ariq Emtenan and Siddharth Garg and Mahesh V. Tripunitara, "Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," 22nd USENIX Security Symposium

[6] Jonathan Valamehr and Timothy Sherwood and Ryan Kastner and David Marangoni-Simonsen and Ted Huffmire and Cynthia Irvine and Timothy Levin, "A 3-D Split Manufacturing Approach to Trustworthy System Development," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[7] A. Sengupta and S. Bhadauria and S. P. Mohanty, "Embedding low cost optimal watermark during high level synthesis for reusable ip core protection," IEEE International Symposium on Circuits and Systems (ISCAS)

[8] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, HOST '08

[9] A. Al-Anwar and Y. Alkabani and M. W. El-Kharashi and H. Bedour, "Hardware trojan detection methodology for fpga," 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)

TABLE III: Results of applying DFS-based Isomorphic Replacement on an 8-bit multiplier

	RT=1 DT=1	RT=5 DT=1	RT=2 DT=7	RT=2 DT=1
E	0.44060	0.44060	0.44060	0.44060
IE	0.03977	0.17684	0.17317	0.05996
IG	623	175	187	557
NG	753	753	753	753
GT	3	3	3	3
MIGC	13	13	11	13
MCF	1594323	1594323	177147	1594323
AIGC	2.20921	0.62056	0.66312	1.97517
ACF	11.32571	1.97736	2.07199	8.75788

[10] A. Al-Anwar and Y. Alkabani and M. W. El-Kharashi and H. Bedour, "Hardware trojan protection for third party IPs," 2013 Euromicro Conference on Digital System Design

[11] K. Vaidyanathan and R. Liu and E. Sumbul and Q. Zhu and F. Franchetti and L. Pileggi, "Efficient and secure intellectual property (IP) design with split fabrication," 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)

[12] A. Vijayakumar and Vinay C. Patil and Daniel E. Holcomb and Christof Paar and Sandip Kundu, "Physical Design Obfuscation of Hardware: A Comprehensive Investigation of Device and Logic-Level Techniques," IEEE Transactions on Information Forensics and Security Volume: 12, Issue: 1, Jan. 2017 Page(s): 64 - 77

[13] F. Koushanfar and Y. Alkabani, "Provably secure obfuscation of diverse watermarks for sequential circuits," IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)

[14] Clifford Wolf, "Yosys Open Synthesis Suite," http://www.clifford.at/yosys/

[15] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/ alanmi/abc/