EigenCircuit: Divergent Synthetic Benchmark Generation for Hardware Security Using PCA and Linear Programming

Sarah Amir and Domenic Forte, *Senior Member, IEEE*

***Abstract*—Benchmarks are the standards by which technologies can be evaluated and fairly compared. In the field of digital circuits, benchmarks were critical for the development of CAD and FPGA tools decades ago. Hardware security is an emerging field of research where new techniques of security and vulnerability of hardware designs are being proposed in higher volume each year. Using decade-old VLSI/CAD oriented benchmarks for analyzing the techniques has many issues as these benchmarks were not developed for security research. Additionally, the rise of statistical analysis or machine learning to model vulnerabilities and solve security issues demands a very large set of samples for training purposes. Since the number of available VLSI/CAD benchmarks is limited, such volume can only be obtained through synthetic benchmark generation tools. To accommodate both of these needs, the first hardware security oriented synthetic circuit benchmark generation framework is developed in this paper. With the use of principal component analysis (PCA) and linear optimization tool, the benchmarks generated by the proposed framework are "divergent", that is having maximum variation in structures from each other. By accommodating user inputs for desired features, the framework offers customization for generating richer and more challenging benchmarks for data-driven hardware security. With thorough experimentation, we demonstrate our framework's scalability, the structural and functional variations in the generated benchmarks, and the advantage of structurally variant synthetic benchmarks in hardware security applications.**

***Index Terms*—benchmark testing, hardware, security, tools, optimization**

I. Introduction

CYBERSECURITY aims to protect Internet-connected systems from attacks and has become one of the most important fields of research. While the significance of software security was realized decades ago, it cannot provide reliable protection without trustworthy hardware. However, globalization of the electronics supply chain has resulted in untrusted entities handling sensitive intellectual property (IP), fabricating integrated circuits (ICs), and testing ICs. Thus, even when the software is trusted and verified, weaknesses in untrusted hardware can circumvent it. Researchers in hardware security are therefore addressing threats that were not considered before. The various topics in hardware security research include hardware Trojan detection and avoidance [1], hardware obfuscation [2, 3], fault injection[4], etc. Hardware Trojan research focuses on detecting or preventing malicious additions or modifications made by an adversary during design, integration, or fabrication. The resulting threats includes leakage of information, intentional malfunction, or kill switch trigger. Hardware obfuscation consists of modifying an IC design to intentionally conceal its functionality, which makes it more difficult to reverse engineer or use without permission. In pre-silicon phase, it makes the circuit unintelligible, protecting against IP infringement, malicious modifications and other threats. In post-silicon phase, locks introduced by obfuscation are used for access control, thus preventing IC overproduction.

Because of the high impact of these issues, research in hardware security has become a hot topic in academia, industry, and government. Identification of new threats and development of countermeasures are resulting in a large volume of research publications every year. In order to fairly evaluate these newly proposed methods and compare them with existing ones, researchers need standardized benchmarks. Using arbitrary benchmarks makes it hard to have a wholesome idea of their effectiveness and almost impossible to reciprocate the results. Unfortunately, the benchmarks currently being used by the hardware security community are CAD/ VLSI-oriented benchmarks, mostly developed decades ago. For example, we looked into the highly cited hardware obfuscation schemes and attacks on them that were published between 2008 and 2020. The findings are presented in Fig. 1. More than 70% of those evaluated their works using ISCAS'85 [5], ISCAS'89 [6], MCNC'91 [7] benchmarks which were already more than two decade old. Whereas the usage of well-known benchmarks can help better understand-ability of the schemes, concerns with using such benchmarks for hardware security include:

- *Size:* Design complexity in number of transistor in commercial ICs is increasing exponentially according to Moore's law [8]. While a thousands gate design represented real world ICs decades ago, present day designs hold millions of gates. The smaller benchmarks developed earlier were sufficient for the CAD development at that time, but they are not enough to evaluate technology developed today. Often the new proposals using traditional benchmarks report higher percentile overhead and appear infeasible, but could have very low overhead when applied on a large industrial design. Vice versa is true as well. This questions the interpret-ability and fair comparison of such overhead analyses.
- *Function:* Another issue of using traditional CAD benchmarks is function of each of the benchmark is well known to everyone. From a hardware security standpoint, this can bias the research results as well as the directions taken by researchers. It is often easier to visualize, extract and

remove security schemes embedded in them than it might be for a arbitrary ASIC design.

- *Objective:* The fact that CAD-oriented benchmarks were not originally designed for security research, they often lacks the criteria significant for such analysis. For example, a benchmark for Trojan insertion should have many low observable nodes [9, 10, 11], while a benchmark for obfuscation, side channel analysis or fault analysis should have assets/IP worth protecting [12]. While these benchmarks performed excellently for CAD development works, which they were designed for, they hardly can offer sufficient usability for such security research. There have been some good works in [9, 10, 11, 12] that modifies the benchmarks to include Trojans and locking mechanisms, but these works too were limited by the original benchmarks that they modify.
- *Quantity:* A trending topic in nearly every analytical field these days is machine learning (ML) and artificial intelligence (AI). In hardware security, new ML/AI-based security scheme and attacks are just now being introduced for Trojan detection [13], obfuscation [14, 15], and side channel attacks [16]. However, to properly train and evaluate such approaches, especially when deep learning is involved, a very large number of benchmarks is necessary. Even thousands of samples are way too few and the hardware security community is only relying on a few hundred. When more benchmarks are generated by slightly modifying the existing ones, they do not provide enough structural and functional variation for ML/AI to explore. While one could argue that industry and government have access to large numbers of past designs, these are often proprietary or classified and thus cannot be shared with researchers.

Synthetic benchmark generation is the best way to address these limitations because it allows one to obtain a large sample set which varies in size, function, structure and objective. In [17], we introduced a linear programming (LP) formulation for generating "divergent" combinational circuit benchmarks and demonstrated them in hardware security applications. Here, we dramatically improve not only the scalability of our original approach but also the divergence of the benchmarks it generates. To be more specific, our main contributions can be summarized as follows:

- We introduce a novel alternative way to expedite the optimization process that uses *Principal Component Analysis (PCA)*. Our new framework is thus called EigenCircuit as it performs eigendecomposition of circuit.
- We implement and compare EigenCircuit to our previous framework [17] and show that it not only preserves the quality of the generated benchmarks, but offers better scalability with thousand times faster execution time.
- We also experimentally show the additional challenges provided by our synthetic benchmarks compared to the existing reference benchmarks for logic locking and hardware Trojan research.

The rest of the paper is organized as follows - Section II summarizes the existing works in related field and the works used in our framework. Section III describes the idea and theoretical and mathematical details of our proposed framework. In Section IV, we focus on experimentally demonstrating the proposed approach and analyzing its results. Section V concludes the paper with a summary and future directions.

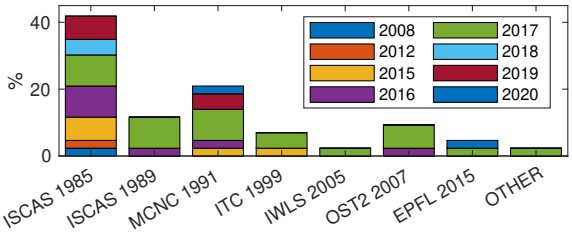


Fig. 1: A summary of the benchmarks used in hardware obfuscation techniques and attacks from 2008–2020.

II. Related Works and Preliminaries

A. Synthetic Benchmark Generation

Generation of benchmarks synthetically was introduced in the 1990s, to support the development and evaluation of CAD/EDA tools. The initial works were done as logic graph modification [18] and circuits were processed as graphs [19]. A more detailed generation process that investigated the idea of representing circuit as mathematical array of numbers was presented in [20]. The structural information of a reference design was extracted with C_{CIRC} tool [21], which returns arrays of numbers in a statistical file which can sufficiently represent the tree-like structure of a combinational design. The details of the representation can be found in Section III-B, as it contains close relation with our work. The tool C_{CIRC} accompanied another tool, named C_{GEN} [21] which can read the statistical file and generate a new circuit from scratch. In step-by-step generation process, gates and input-outputs were assigned in levels, and then the wires were placed to complete the design. Though the C_{GEN} tool generates a new circuit from scratch, it highly depends on the data in statistical file. The arrays of numbers referring to simple structural information can be modified to ensure the generation be different than the reference, but the accompanying detailed data of the structure in the file, which are not manually calculable, makes the possibility of proper alteration very minimal. As a result, the generated circuit cannot be too different from the original reference. However, it nevertheless shines light on how to properly represent a circuit with mathematical arrays and reconstruct circuit from such data. The benchmarks acquired with this method would alleviate the problem of limited number of available benchmarks, but it still lacks in the variance and customization. In many advanced research, detailed in Section III-A, the variance of features are critically important for proper performance.

In our work, we ensured the variance of features along with enabling customization of basic circuit size and dimension of external connectivity. The benchmarks suite obtained with our method would not just offer unlimited quantity, but also ensure quality by having large spectrum of structural and functional divergence in the features. Fig. 2 offers a simplified overview

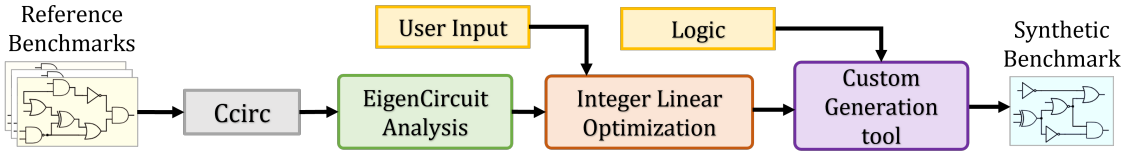


Fig. 2: Simplified overview of the proposed flow.

of our framework. It takes reference benchmarks as input along with other user input and incorporates the works of [22, 21] and linear programming to achieve all these qualities.

B. Hardware Security Benchmarks

In previous works, significant contribution has been done to customize existing benchmarks to make them more suitable for hardware security researches [9, 10, 11, 12]. Trust-HUB holds a repository of specialized benchmarks [23]. Authors of [9, 10, 11] published benchmarks specifically for hardware Trojan research, where they introduced small to large Trojans into designs that can offer a spectrum of challenges for Trojan detection algorithms. Works on secured or locked electronic design benchmarks for the purpose of hardware obfuscation study have been introduced in [12]. In this work, existing benchmarks were modified to include logic locking schemes that can work as protection for the IP. However, there are limitations such as (1) the benchmarks can be used to evaluate attacks and threats, but not new locking techniques, (2) few hundred benchmarks were published, so it's not sufficient for any method requiring larger sample set, and (3) the benchmarks are modified versions of previously existing benchmarks, and the physical features of new benchmarks vary only little from the existing ones. All these insufficiency of existing work drives as a motivation for the work presented in this literature.

Above mentioned hardware security topics would benefit from benchmarks with special characteristics. For example, for a hardware Trojan to be stealthy, it is expected that the benchmark used in experimentation have low value of observability, hiding the signals from getting detected easily. Detection of Trojans in such benchmarks would be more challenging for any detection algorithm. Thus, ensuring that the detection algorithms can be improved and more reliably detect similar malicious modifications in real circuits.

Looking closely into the hardware obfuscation techniques, it is apparent that they are closely related to various features of designs. For example, fault analysis based obfuscation [24] places locking gates in places where the output corruption would be highest, and corruption is estimated by fault analysis. To test how this methods works against various designs, one needs a large amount of benchmarks, that vary in size and in features such as controllability and observability. Neither is satisfied with existing suite of benchmarks. Further, as mentioned earlier, machine learning based attacks [14, 15] need to be trained with millions of sample obfuscated benchmarks in order to work properly. Otherwise the claim of effectiveness of such algorithm becomes questionable. Our proposed benchmark generation framework is designed with these requirements in mind.

Apart from controllability and observability, other features that are often useful for security research are number of clause, reconvergence, delay, design size etc. For generated benchmarks to be useful for security research, it is desirable that the framework offers customization or prioritization of these features. Our proposed methodology is capable of directly set value for features like delay or size. Number of clauses can be varied by varying size of design. Controllability and observability can be changed by varying size, changing depth for a fixed size design or the logic gates used in the design. Reconvergence can be altered with maximum fanin-fanout values. More concise control over indirect parameters are still an ongoing work. If the complex relation between attack resiliency and these features can be formulated, it can be used to optimize the benchmark generation to produce either highly resilient or vulnerable designs. Such relationships can be discovered using AI/ML as long as a large and divergent number of circuit samples are available. This is the purpose of the objective functions in Section III-E where new benchmarks that are structurally different from prior ones are generated.

C. Linear Programming (LP)

An important feature of our proposed framework is utilizing linear optimization techniques to determine the structure of synthetic benchmark and to integrate security features in the process. LP is a powerful way to determine an optimal solution for a linear problem under linear constraints. An integer LP optimization technique [25] finds the minimum of a problem specified using the following formulation

$$\min_x f^T x \;\; subject\, to \begin{cases} Ax \leq b \\ A_{eq}x = b_{eq} \\ lb \leq x_i \leq ub \;\forall i \\ x_i \in \mathbb{Z} \;\forall i \end{cases}$$

where f, x, b, b_{eq}, lb, and ub are vectors. $f(x)$ is the term needs to be minimized, A and A_{eq} are matrices that represents the coefficients of x in linear inequality and equality constraints, respectively. The third and fourth relations restrict the range of each feature x_i to $[lb, ub]$ and to integers, respectively.

D. Principal Component Analysis (PCA)

Principal component analysis or PCA is a well-known linear projection operator that we use in our framework. PCA maps a variable from one coordinate system to a new coordinate

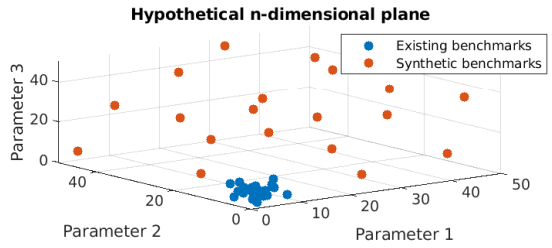


Fig. 3: A hypothetical $n = 3$ dimensional plane showing the structural parameter distribution of existing (blue) vs. ideal synthetic benchmarks (orange).

frame where the axes represent maximal variability [26, 27]. With PCA, an input data matrix X is transformed back and forth to an output matrix Y through the transformations:

$$Y = XP \tag{1}$$

$$X = P^{-1}Y \tag{2}$$

where P is the projection matrix and each column of P is a principal component. A truncated P maps each point of X to a low-dimensional space. C is the covariance matrix of X that is used to find P. It is given by

$$C = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)(x_n - \mu)^T \tag{3}$$

where μ is the mean, defined with

$$\mu = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{4}$$

Performing *eigendecomposition*, the covariance matrix C can be expressed with the projection matrix P as

$$C = P\Lambda P^T \tag{5}$$

where Λ is a diagonal matrix with elements $\lambda_1, \lambda_2, ..., \lambda_D$ and $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_D$. Here each λ, corresponding to each column of principal component in P, indicates the variance explained by projecting the data onto that component. P_i is the eigenvector and λ_i is the eigenvalue, where i refers to the column number of P [27].

III. Proposed Synthetic Benchmark Framework

A. Hypothesis and Goal

Let's consider a multidimensional space, like the one shown in Fig. 3, where each axis represents a unique structural feature of circuit designs. We hypothesize that almost all existing circuit benchmarks as well as those generated by minor perturbations or mutations of them would fall in a close concentrated cluster (represented by blue dots in in Fig. 3). If one wants a design that lies far from the cluster, the horizon for the parameters must be extended. The goal of the proposed approach is just that – *to create combinational circuit benchmarks with structural parameters that are very different than the existing benchmarks*. In other words, we want to create new benchmarks outside of the cluster like those represented by orange dots in Fig. 3.

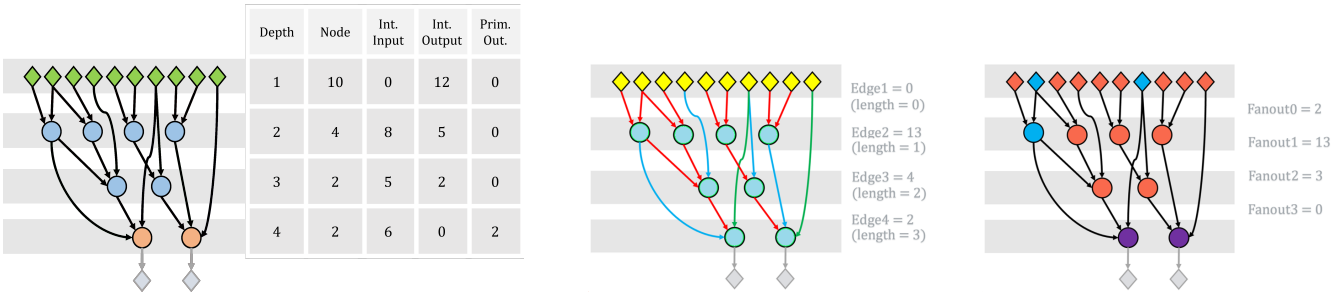
There are multiple reasons why this goal is valuable as previously mentioned. Most notably, data-driven methods based on ML/AI need such "diverse" samples to form accurate predictions and avoid over-fitting. For hardware security topics discussed earlier, AI/ML could be used to optimize countermeasures, e.g., insertion of obfuscation/locking gates. There are many different attacks against obfuscation and logic locking that might be difficult to analytically model. By using a large set of diverse obfuscated samples and applying attacks against them, ML/AI can better learn how to perform attack-resistant logic locking. As another example, many weaknesses and vulnerabilities in hardware security, like the more general cybersecurity, are "unknown unknowns". That is, they are unforeseeable situations which pose a potentially greater risk simply because they cannot be anticipated. By attempting to cover the entire "space" of circuits, it is more feasible to identify such patterns and challenging corner cases where unknown unknowns might occur.

B. Mathematical Representation of Circuit

In order to run optimization methods such as LP for generation of *valid* digital circuits[1], the first requirement is to represent a design mathematically with quantifiable variables. These variables necessarily would represent the entire structure sufficiently. Both the constraints and objective could then be defined using those variables. In our work, we have developed a modified version of the structural representation used in [20] for these variables. Figure 4 shows the variables in [20] which are explained below.

1) Circuit depth (n): The maximum depth of a circuit is the length of the longest path from the primary input to the primary output in terms of number of gates in the path. Each structural parameter extracted from circuits are arrays of length n where n represents the maximum logical depth. For optimization, we denote n as the desired maximum depth or delay for the synthetic benchmark. All reference benchmarks are adjusted to this length for our optimization flow. Depth is denoted on the first column in the table of Fig. 4a.
2) Node distribution (N): Node refers to a logic gate in this representation. Node distribution is defined as the number of gates in each depth. Primary inputs are placed in the first level (i.e., at depth of 1), and their quantity is considered as N_1. N_i is the number of gates that are on $i-1$ delay from the input (see Fig. 4a).
3) Internal inputs distribution (I): This is the number of internal inputs at each depth. In our notation, I_i represents this variable for depth i. As no internal inputs go to primary inputs, $I_1 = 0$ (see Fig. 4a).
4) Internal outputs distribution (O): This is an array of number of internal outputs drawn from each depth, where O_i refers to number of internal outputs coming from the nodes in depth i. The last depth n does not have any internal outputs, so $O_n = 0$ (see Fig. 4a).

[1]Valid means that circuit does not consist of floating nets, nets driven by multiple nodes, etc.



(a) Depth(n), N, I, O, PO (b) Edge (c) Fanout

Fig. 4: Example of structural features from the graph representation of a circuit. The diamonds on top and bottom represent primary inputs and outputs respectively. The circles represent gates.

5) Primary outputs distribution (PO): This array represents the number of primary outputs drawn from each depth. Alternatively, PO_i would refer to number of primary outputs that have a delay of $i-1$ from the primary inputs, or primary outputs that are drawn from nodes placed in depth i (see Fig. 4a).
6) Wire length or edge distribution (E): This is an array that contains the number of wires (i.e., edges that connects nodes) of each length in the entire circuit. Here, length is the difference in depth level between the nodes that are connected with that wire or edge. E_i represents the total number of wires in the design that has length $i-1$ (see Fig. 4b).
7) Fanout distribution (F): This array contains the number of gates with specific fanout size in the entire circuit. F_i refer to total number of gates in the design that has fanout of i. This fanout is the number of nodes to which each node output is directly connected to. In a typical reference design, F_1 is found to be the largest number, followed by F_2 (see Fig. 4c).

In our framework, we also incorporated more detailed variables which breaks down edges (E) into *edges per depth level* and fanout (F) into *fanout per depth level*. The elaboration helps in both defining the structure and translating it into a gate level netlist. Both of these are represented as two-dimensional arrays or matrices.

1) Edges per depth level (Ed): Two dimensional array that contains number of wires of each length in each depth.
2) Fanout per depth level (Fd): Two dimensional array of gates with each possible fanout starting from each depth.

The vectors N, I, O, PO, E, and F contain n elements. Ed and Fd contain n^2 elements and, for simplicity, are converted to one dimensional arrays by concatenating rows back to back. We construct the optimization variable vector x as the concatenation of vectors N, I, O, PO, E, F, Ed, and Fd, thus consisting of $2n^2+6n$ structural parameters.

$$x = [N, I, O, PO, E, F, Ed, Fd]$$

As will be discussed in Section III-D , the designer has the opportunity to influence these parameters through the framework's user-defined inputs.

C. Problem Formation Options

Though it seems easy to determine an array of data that is most different from some reference array of data using basic optimization techniques, the process gets complicated when the resultant data must be meaningful and refer to realistic circuit. A *constrained* optimization approach is critical to enforce the rules for validity where the objective function of optimization is utilized to implement the preferences. As the circuit parameters are real integer numbers, we found integer linear programming to be the most appropriate choice.

D. LP Constraints

The constraints are mathematical representations of physical limitations of a real world circuit. In this work, we implemented more than two dozen such mathematical constraint types that ensure the synthetic benchmark structure refers to a valid circuit. For example, a three input gate cannot have more than three different input signals. If we examine common technologies, we can see, logic units may have up to four inputs. Another example of a constraint can be maximum fanout constraint of a design. In this case, designer selects a number which refers to the maximum allowable loading of output of any gate. In other words, it indicates how many other gates can be driven by the output of a single gate. Some additional constraints were also employed to capture user-defined data to customize the resultant benchmark. Provided in a optimization tool, even without any specific objective, these constraints are enough to generate random synthetic benchmarks.

For brevity, we only cover a few simple constraints here:

(i) User defines certain data for the desired synthetic benchmark. These are number of gates in the design (S), primary inputs (PI), primary outputs (PO) and depth ($n-1$). As discussed in Section III-B, the nodes in first depth are primary inputs, and in depth 2 to n are logic gates, the constraints to enforce these user-defined inputs are -

$$PI = N_1 \tag{6}$$

$$S = \sum_{i=2}^{n} N_i \tag{7}$$

$$PO = \sum_{i=1}^{n} PO_i \tag{8}$$

(ii) Maximum fanin (mFI) and fanout (mFO) are also parameters defined by the designer. These are constraints for each node. These links number of internal inputs (I_i) and outputs (O_i) in each depth to the number of nodes in that depth (N_i) with-

$$N_i \leq I_i \leq N_i \times mFI \; \forall \; i \tag{9}$$

$$N_i \leq O_i + PO_i \leq N_i \times mFO \; \forall \; i \tag{10}$$

(iii) Nodes at any depth N_i should not be more than the maximum possible internal inputs and POs available to take in the output of this depth and should not be more than maximum possible internal outputs that can feed this depth. This can be defined as constraints with:

$$N_i \leq mFI \times \sum_{j=i+1}^{n} N_j + \sum_{j=i+1}^{n} PO_j \; \forall \; i \tag{11}$$

$$N_i \leq mFO \times \sum_{j=1}^{i-1} N_j \; \forall \; i \tag{12}$$

There are dozens more constraints which we withheld from here for brevity and are as vital as these to impose correctness of the structure sufficiently. Only with the proper constraints, the optimizer can determine the optimum value for the objective function that refers a structure that is not just unique and significant, but can also be translated into a working circuit.

E. LP Objective Functions

The objective function helps to prioritize the features that result in the random synthetic benchmark structure. Along with a set of valid constraints, the objective gives the control over choosing more desirable circuit according to the need of the designer. This can prioritize one criteria or feature over another when generating a circuit. In this paper, we present two objectives, both of which is developed to acquire a diversified synthetic benchmark suite. While one objective (from [17]) is more detailed and resource-heavy, the other one (EigenCircuit) is faster and more convenient. The workflow illustrated in Fig. 5 highlights the different paths for two objectives.

1) Objective 1: Maximize distance from all references: This objective is based on the concept of finding the most distant point in a $p * n$ dimensional space. Each axes in that space represent one structural parameter at one specific depth. As mentioned earlier, these structural parameters or variables are nodes, inputs, outputs, primary outputs, edges, and fanout arrays (per depth), each being an array of length n. So, for p parameters and n depth, the total number of axes are $p * n$. The distance between reference and resultant ILP benchmark are measured in all axes independently. The objective is to maximize the total distance between reference and resultant, summed up for all axes. So, the linear optimization problem for this objective can be defined as follows

$$\begin{aligned} &\max_x f_1^T(x) \; where \\ &f_1(x) = \sum_{i=1}^{n} \sum_{j=1}^{r} |x_i - x'_{i,j}| \\ &subject\,to \begin{cases} Constraints \\ Bounds \\ x \in \mathbb{Z}^{0+} \end{cases} \end{aligned} \tag{13}$$

where i corresponds to a single depth level for each structural feature, x is concatenated parameters per depth, r is the number of reference designs. $x'_{i,j}$ is the ith feature from the jth reference benchmark, and x_i is optimal ith feature for the synthetic benchmark that can later be translated to a circuit in netlist form. Optimization constraints have already been discussed in detail in Section III-D. Bounds for variables are used to ensure efficiency of the optimization. Setting the highest and lowest possible value as upper and lower bounds respectively reduces the optimization time by reducing range of sweep for each variable.

With this objective, the first resultant benchmark is placed in a position which causes the summation of distances for reference benchmarks maximum [17]. For generation of later benchmarks, this newly generated structure can be considered as another reference point. The distance from original references and from already synthesized benchmarks are both considered for the placing of next benchmark. Hence, consecutive generations place benchmarks in different positions, both from original references and from each other.

For this formulation, the depth of each reference must be the same. Since many of the references may have different depth, adjustment is done to equalize them to the resultant. This is done by either adding buffer layers in a uniform fashion or truncating the circuit.

Fig. 5 with the blue hyphenated path shows the simplified flow for this objective function. The reference benchmarks are processed using Ccirc tool [21] to generate statistical data file in stats format. These files contains all necessary information about the structure of the designs (nodes, inputs, etc.) where the circuit structure is represented mathematically with numeric arrays as explained in Section III-B. This data along with user inputs are fed into linear optimization tool. The objective in the optimization is set as Equation (13) [17]. The resultant structural data is then translated into a gate level netlist by a custom tool. This tool determines the logic according to user choice (see Section III-F).

This objective is variation oriented and results in a diverse set of benchmarks. The only drawback is, the optimization may take a long time. The number of variables and equations for the optimization increases with number of references and depth. For the optimizer, number of variables is $3n^2 + (31 + 18m)n$ and number of constraint equations is $17n+25mn+3$, where n is the depth of resultant benchmark and m is the number of reference benchmarks. It is more applicable for a system where a small set of more diversified benchmarks are desired.

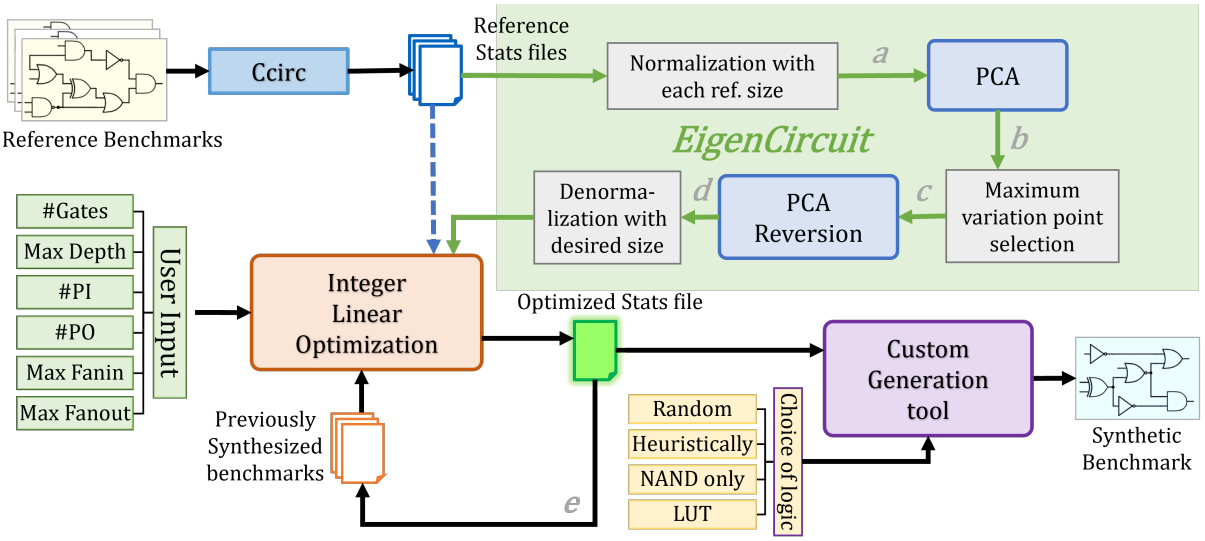


Fig. 5: Flow chart of the framework in version 1 with objective 1 (blue path) and version 2 with objective 2 (green path).

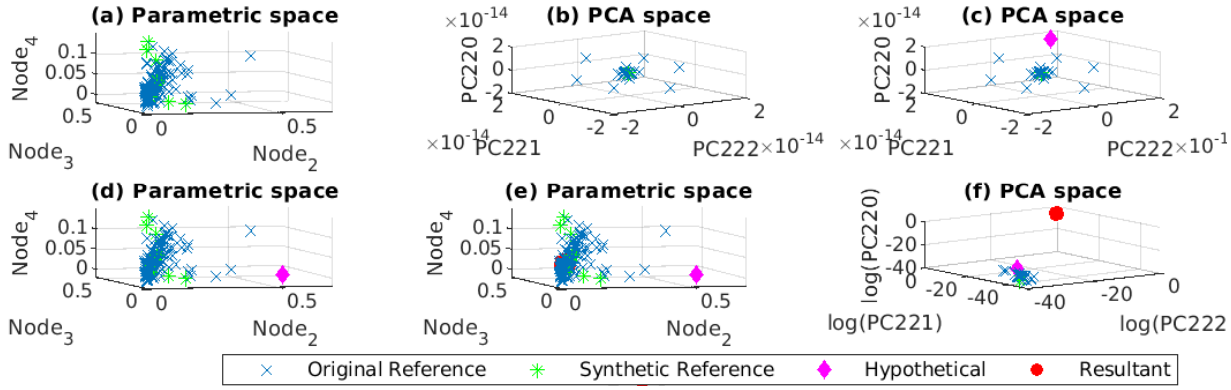


Fig. 6: Determining the position of hypothetical point. Details are in section III-E2.

2) Objective 2: Minimizing the distance from hypothetical maximum distant point (derived in PCA space): The second version of the framework has been developed with a new objective. In order to expedite the time taken for optimization, other than handling each reference separately, a wholesome technique is applied. In this version, the reference designs defined in original axes of structural features are processed with PCA. As new axes of PCA holds the information of variations between designs, a hypothetical point that lies most distant contains most difference. Instead of calculating distance from every reference due to unavailability of boundary conditions and to reduce time and complexity, we selected a hypothetical point which is equal to the maximum data for each axis independently among the reference points. A hypothetical point that has maximum value along all PCA axes might not refer to a real circuit. Once this hypothetical point is reverted back to original physical parametric space, it might contain unrealistic negative values for circuit structural features like number of gate or input per depth. Even though this point is unrealistic, this holds the most difference, so the objective is to *get a valid circuit structure as close to the hypothetical point as possible*. Hence, in this objective, the optimizer tries to minimize the distance between the hypothetical point and the resultant solution, i.e.,

$$\begin{aligned} &\min_x f_2^T(x) \; where \\ &f_2(x) = \sum_{i=1}^{n} |x_i - x_{hypo,i}| \\ &subject\,to \begin{cases} Constraints \\ Bounds \\ x \in \mathbb{Z}^{0+} \end{cases} \end{aligned} \tag{14}$$

where $x_{hypo,i}$ denotes a hypothetical maximum point at corresponding to this axis for the structure and the remaining terms, constraints, etc. are the same as Equation (13).

The framework displayed in Fig. 5 highlights the difference of steps in the second version with green paths and box. Additionally, the status of analysis in internal steps, marked with $a-e$, are elaborated in Fig. 6 for a better understanding.

- In Fig. 6a, the references are placed in original parametric space, where the axes are defined as distribution of structural parameters like nodes each depth, internal inputs in each depth and so on. For visualization, only first three elements of x are shown here.

- In Fig. 6b, reference benchmarks are transformed to the PCA space using Equation (1). In the presented plot, the axis are chosen to be a least significant principal components, referencing to the components of reference benchmarks that are having least variation in values. Our objective is to maximize the variation in all axes. We are displaying the effect in places where the variation are least in existing references.
- In Fig. 6c, a hypothetical point (magenta diamond) is chosen in the PCA space. The hypothetical point is the maximum value of references on each axis, for all axes. (It can be multiplied with any number to intensify the difference if the designer desires.)
- In Fig. 6d, the references, along with the hypothetical point, are transformed back to parametric space using Equation (2). Note that, in this state, the hypothetical point might not represent a valid design, hence may include negative values on any parametric axis.
- In Fig. 6e, the optimizer determines the closest benchmark to hypothetical point that satisfies all constraints (hence refers to a valid circuit). This optimum point is represented with red dot.
- In Fig. 6f, the position of the optimum selected position, along with references and hypothetical point are shown in a PCA space. Logarithmic scale had to chosen to clearly capture the distance. This farthest placement of optimum resultant indicates that it holds a large variation even at the axes where the variation was small originally.

In this EigenCircuit version, for the optimizer, number of variables is $3n^2 + 49n$ and number of constraint equations is $42n + 3$, where n is the depth of resultant benchmark. The complexity does not vary with the number of references. So the optimization is fast, and the number of reference can be very large. This aids in generating larger benchmarks and larger number of benchmarks more quickly as compared to [17].

F. Logic Assignment

The optimization determines the best and most different graph-like structure. But the structure does not contain information about the logic the nodes would hold, i.e., the functionality is not optimized. The structure is more like a LUT-based netlist on which any logic can be assigned. The exact output of the optimizer is a statistical file, where the structural parameters are mathematically represented with arrays of numbers. This file is then translated to a workable library dependent or independent Verilog netlist by the our *Custom Generation tool* (see Fig. 5). This tool translates the mathematically represented tree structure into a netlist by introducing the functionality of each node. Now, depending on the objective of the designer, this functionality determination can have different assignment.

- In random selection, a set of logic gates or a library is provided to the generation tool. From the optimum structure, the generation tool determines the number of input for a specific gate, and randomly selects logic gate from available options that matches the structure. All available gates with same number of inputs have same rate of occurrence.
- In heuristically driven selection, certain functionality can be selected to be maximized. For example, to increase controllability or observability, following the findings of [28], the logic choices can be chronologically sorted for preference. As 'AND' / 'NAND' gates have larger value for controllability or observability, selecting only to maximize these parameters results in every gate in the design to be 'AND' / 'NAND'. Such structure may seem be reduced to a smaller one as many parts of the design would have no controllability at all. But, if only a certain imbalance is maintained in the equalized distribution of random logic assignment, in a way that would help increasing certain feature like controllability or observability, like increasing the rate occurrence for 'AND'/'NAND' logic over that for 'OR'/'NOR' logic, the resultant benchmark would be more reliable. Demonstration of this phenomena is included in Section IV-C3.
- A lookup table (LUT) based translation is also possible. The structure is only interpreted into a netlist with LUTs which can be defined later with logic on a FPGA board.

IV. Experimental Results

The theoretical details explained in the previous section has been verified and implemented. Both objective functions were utilized to generate two benchmarks suites. The generated designs are inspected, tested and analyzed to ensure the optimization is enhancing the divergence in features as it is designed to do theoretically. This section discusses the experimental findings in detail.

The synthetic generation and result analysis are performed in a Linux operated server machine. As samples, we have generated 12 benchmarks with objective 1: Direct Distance optimization (referencing 10 ISCAS benchmarks) and 61 benchmarks with objective 2: EigenCircuit (referencing 186 benchmarks from ISCAS, MCNC and EPFL suites). The sizes of the benchmarks, both synthetic and references are presented in Fig. 7. In this chart, red, orange and yellow bars represent the sizes of reference combinational benchmarks, respectively from ISCAS, MCNC and EPFL suites. The green and blue bars are for synthetic benchmarks with objective 1 and 2. As the designer can select the size of synthetic benchmark when generating, these synthetic benchmark sizes are arbitrarily set. But it is apparent here that even if reference benchmarks are small in size, the synthetic benchmarks can be generated as large as required.

Along with the generations, we have farther experimented on the benchmarks to evaluate them. We executed experiments to verify the variance in both structural and functional features on the generated synthetic benchmarks and compared with the references. The findings are explained in Sections IV-A and IV-B. We also ran experiments with the generated benchmarks on security research topics and analyzed the difference of their applicability with existing reference benchmarks in Section IV-C. We also analyzed the scalability of the framework against the feature size of the generations in Section IV-D.

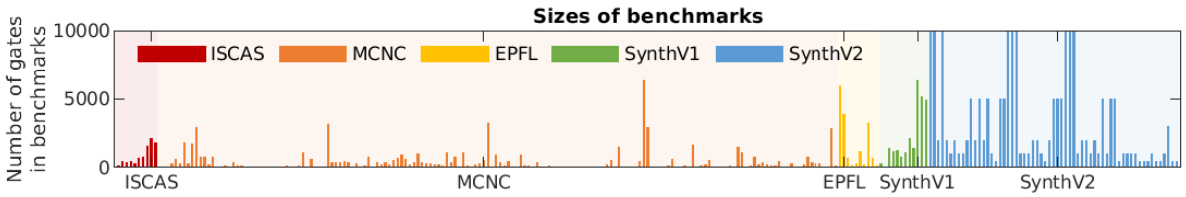


Fig. 7: The sizes of reference and synthesized benchmarks.

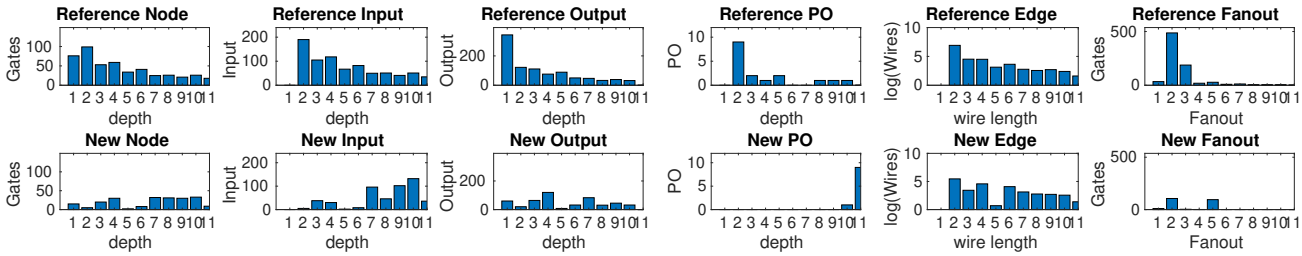


Fig. 8: Distribution of structural features where (top row) represents the average distribution for the reference designs and (bottom row) represents the distribution for a synthetic benchmarks generated by the proposed flow using objective 1 (Direct Distance optimization

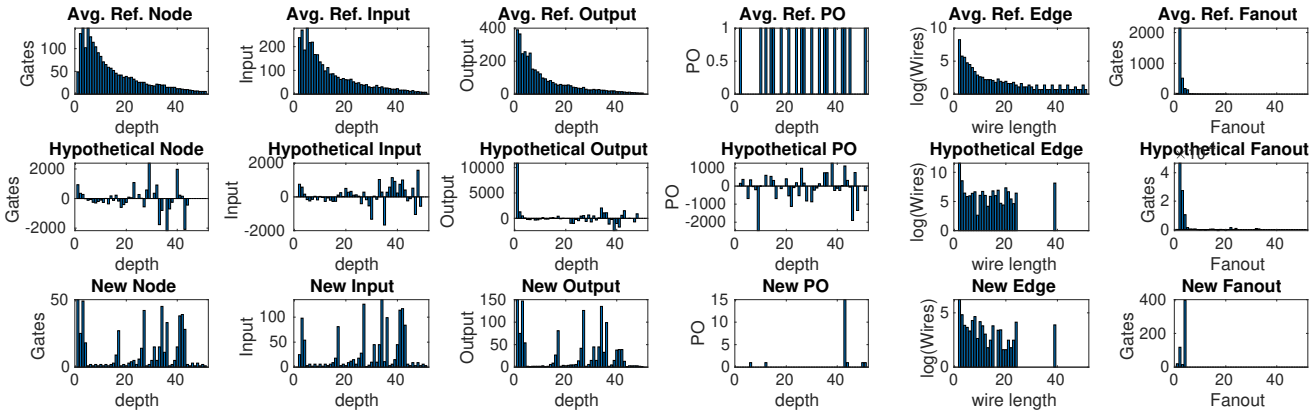


Fig. 9: Distribution of structural features where top row represents the average distribution for the reference designs, middle row refers to hypothetical target point (containing unreal negative values) and bottom row represents the distribution for a synthetic benchmarks generated by the proposed flow using objective 2 (EigenCircuit).

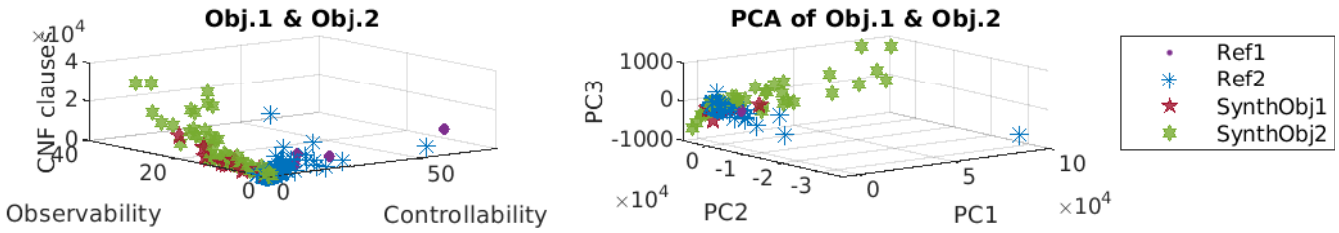
A. Structural Variance

The optimizer compares the structure and maximizes the differences in them. The representation of the structure of a reference circuit can be visualized in Fig. 8 top row. The plots displays the distribution of parameters discussed in Section III-B (nodes, internal inputs, etc.) per depth. For displaying, we plot the average values of each bar for all references. The bottom row is the same plots for the generated benchmark using objective 1 (Direct Distance optimization). For simplicity, if we consider there is only one reference, the optimizer tries to maximize the difference between the bar heights between top and bottom plots for each of the depth in horizontal axis, individually for each parameter.

Fig. 9 shows a similar comparison plot for generation with EigenCircuit framework. Other than maximizing difference from references, the framework chooses a hypothetical distant most point, which is represented with the middle row. As this point is a theoretical point only, it contains negative values for parameters like node per depth. In this framework, the optimizer tries to find a realistic solution (presented in bottom row) that minimizes the differences between middle and bottom row. So, the synthetic benchmark is as similar as realistically possible to the hypothetical point. The obvious structural variation between this synthesized one with the reference benchmarks in top row (averaged) is visible in the plots.

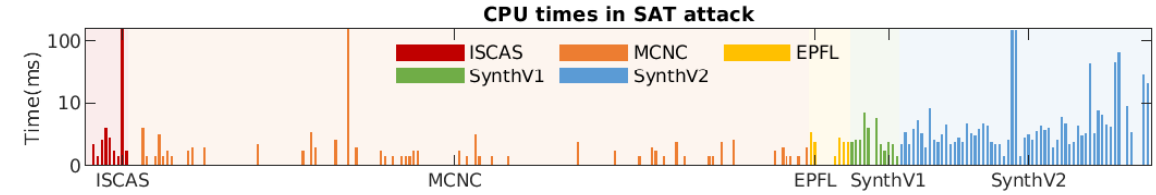
B. Feature Divergence Comparison

The structural variance ensured in optimization results in functional variation in the circuits too. To measure this variation, we collected functional parameters of benchmarks, like controllability, observability (evaluated with Synopsis

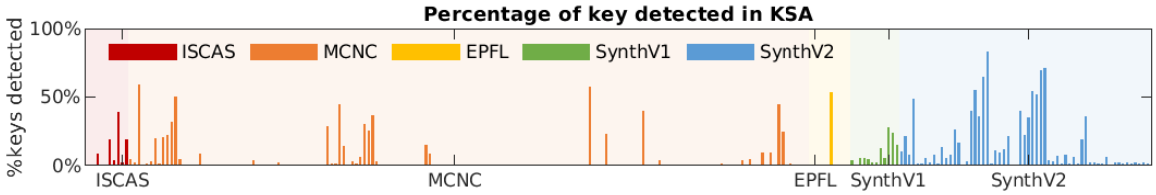


(a) (b)

Fig. 10: Comparing existing and synthetic benchmarks on the basis of structural parameters. Benchmarks are compared based on (a) controllability, observability and number of CNF clauses; and (b) three principal components of PCA performed on multiple analyzed structural parameters. Pentagrams are for synthetic benchmarks generated with objective 1 and hexagrams are for synthetic benchmarks generated with objective 2. Dots and asterisks are for references that were used for generation with objective 1 and 2 respectively.



(a) Execution time of SAT attack (in seconds) on obfuscated benchmarks.



(b) Percentage of keys retrieved by key sensitization attack on obfuscated benchmarks (without brute force attack).

Fig. 11: Performance of benchmarks against logic locking attacks.

TetraMAX [29]), number of conjunctive normal form (CNF) clauses (calculated with ABC tool [30]) and many other direct and indirect features. For comparative observation, we placed benchmarks along these parameters as axes in Fig. 10a where dots and asterisks represents existing CAD reference benchmarks, pentagrams are for synthetic benchmarks generated with objective 1 and hexagrams are for synthetic benchmarks generated with objective 2. To summarize the variations in all parameters we have measured, we performed a PCA on the resultant data set. The three most significant primary components holds the maximum variations in all benchmarks, regardless of which suite they belong to. In Fig. 10b, we placed the same benchmarks, represented with similar marker coding as before, along the three most significant primary components. Both the original parametric and PCA plots shows that synthetic benchmark markers are spread out on more space in the 3D planes than the reference markers. This indicated that the divergence of features in synthetic benchmarks are more widely dispersed than the existing benchmarks. Also, from comparing markers for synthetic benchmarks of both objectives, it is apparent that even though the second objective is faster in optimization, it does not deteriorate the benchmark quality as it presents the same type of variance in the features as the first objective. Note that, with the generation of more synthetic benchmarks, more divergence can be obtained than the ones shown here.

C. Demonstration of Synthetic Benchmarks in Hardware Security Applications

To verify the usage of the synthesized benchmarks in hardware security research and explore the difference of characteristics these possess in contrast to the existing reference benchmarks, we performed some hardware security assessments.

1) Attacks on Obfuscation: Firstly, the benchmarks are obfuscated randomly with 128-bit key. Two popular attacks on logic locking are applied to break the obfuscation. To minimize the effect of randomness of key placement, the tests were repeated ten times and average of attack results has been reported for each benchmark.

In the satisfiability or SAT attack [31], the CPU times in seconds to break the locking are compared in Fig. 11a. Note that, random obfuscation method is not resilient to SAT attack, hence the attack can almost always break the locking scheme unless the circuit has some inherent qualities that can restrict the attack. The timeout was set as 12 hours. From the results, we could identify that synthetic benchmarks are showing more resiliency in general. Without farther analysis of features, it is not possible to comment on the reasons behind

the resiliency. However, for the use of these benchmarks, it can be said that for a thorough analysis, for example, to model the circuit characteristics and resiliency with AI tools, the generated synthetic benchmarks are offering much more qualitative variations than the existing ones can.

A second attack on logic locking, named as key sensitization attack [32] has also been applied on the locked benchmarks. This attack tries to isolate the effect of each key and propagate it to the primary outputs. Fig. 11b displays the percentage of keys that could be broken with the attack without performing the brute force attack at the end. In this attack, the synthetic benchmarks offered less resiliency than reference benchmarks. This attack is a functional attack. Hence, the distinction in performances of attack with synthetic and existing benchmarks suit is another demonstration that — synthetic benchmarks have variation in functional behavior too though the optimization have been designed to maximize structural variance.

2) Hardware Trojan detection: We have tested hardware Trojan stealthiness using reference and synthetic benchmarks. We designed small Trojans which are *trigger*ed by the rarest signals in the designs. The rare signal triggers are sourced from nodes with highest or lowest static probability of either binary values in the entire unit under test. The *payload*s are XOR gates as that would ensure equal probability of alteration as the trigger. These payloads are placed randomly in the original design. Chronologically lesser rare triggers are used to insert more Trojans sequentially. Because of the fact that the exact number of Trojan payload XOR gates are same in reference and synthetic benchmarks, we found side channel based detection schemes are not beneficial in comparative analysis. So, for detecting the existence of Trojans, we ventured two simulation based techniques. First, we ran functional simulation of Trojan infested design using ten thousand random patterns and compared with golden output patterns. Second, we generated test patterns for fault detection for all nodes in the golden design using an ATPG tool. We used these patterns to run functional simulation and compare with golden output patterns like the first technique.

The detection rates for both techniques and types of benchmarks are shown in Fig. 12. The number of Trojans inserted are represented as the percentage of design size in term of logic gates. The bars shows average rate of output alteration in simulation for different amount of Trojans. Blue and orange bars refer to results with existing CAD benchmarks (ISCAS, MCNC and EPFL) in ATPG based and random pattern testbenches respectively. The yellow and purple bars refer to SynthGen and EigenCircuit synthetic benchmarks for similar testbenches. The success rate is defined as the percentage of bits that flips or detects the presence of Trojans. ATPG based detection scheme is more successful in detecting the Trojans. Yet none of the techniques was much successful in detecting Trojans in synthetic benchmarks. Trojans appear better hidden or stealthy in these benchmarks compared to the common reference benchmarks. As the optimization sets higher rate of gate with three or more inputs, synthetic benchmarks have more interconnection between gates, hence, more nestedness. That in turn lowers the controllability and results in lesser chance of Trojan detection. This is a demonstration that this

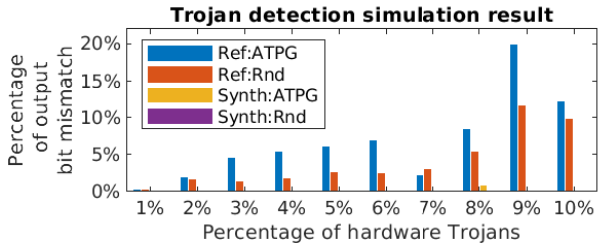


Fig. 12: Success rate of triggering and observing randomly placed hardware Trojans by simulation with random testbenches and ATPG-derived fault analysis based testbenches.

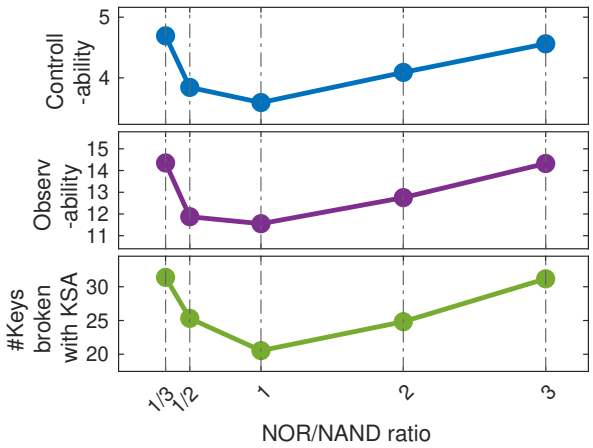


Fig. 13: Variation of controllability, observability and resiliency against Key sensitization attack with NOR/NAND ratio of synthetic benchmarks.

synthetic benchmark suite offers more challenge for functional Trojan detection schemes. They would therefore make excellent samples for future AI/ML methods to learn from.

3) Logic ratio alteration: One of the major benefits of application of synthetic benchmarks in hardware security research is the opportunity to customize features. For example, varying controllability and observability of designs to change resiliency against attacks that observes outputs changes. We have found the distribution of logic gates relates to controllability and observability. Since our framework has the flexibility to control the logic gates and their distribution, we have provided an experiment to highlight the usage of the generation scheme in hardware security. The default generation tool selects logic functions to fit in the optimal structure pseudo-randomly, with equal probability of NAND and NOR gates. We varied the probability of NOR-to-NAND logic when constructing designs from the optimized structure. We collected the controllability and observability values from the benchmarks, obfuscated and ran key sensitization attack (KSA), which depends on those values, and compared the findings with equal NOR-to-NAND ratio design. Fig. 13 presents the relation we found with the same sample optimized structures we used in other experiments in this paper. There is a clear correlation between KSA resiliency and controllabilty/observability. This flexibility to control logic and vary features is exclusive to the synthetic generation scheme. The

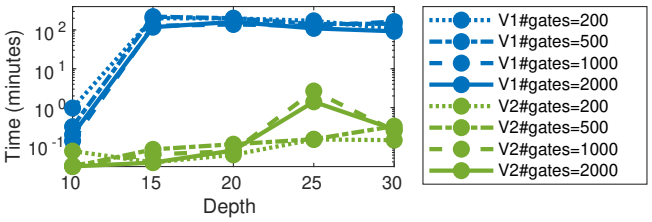


Fig. 14: Optimization time taken with objective 1 (blue) and objective 2 (green) with increasing depth of resultant design.

controlled variation of resiliency against the attack, as shown in the bottom plot, proves the unique applicability of our approach in hardware security research.

The experiments of hardware security on the benchmarks presents two findings. First, the generated benchmarks are sound and workable; they can be used in research for security analysis. Second, synthetic generation can offer benchmarks that are different than existing ones to the study and evaluation of security features. Along with the fact that researchers can generate as many benchmarks they need, this synthetic benchmark generation framework can be highly valuable to be used in evaluation and AI modelling of many hardware security theories.

D. Scalability

With the Direct Distance objective, both the number of variables and equations for the optimizer to consider increases with the number of references as we have discussed in Section III-E1. The algorithm of linear programming is solvable in polynomial time, which means it takes very large time when the inputs are large. The dominating input parameter is the depth, as that defines the number of variables. As a result, time to reach a solution with this framework can get larger with a lot of references. In experiments we referenced only ISCAS suite with 10 benchmarks, and still the generations took large times as shown with blue lines in Fig. 14. The EigenCircuit framework, on the other hand, solves the scalability issue by making the number of optimization variables and equations independent of number of reference. With the EigenCircuit analysis in first part, a single hypothetical point is determined. For optimization, this works as a single reference point, reducing the complexity greatly as explained in Section III-E2. As a result of the complexity simplification, the optimization runs up to thousands times faster, as shown with green lines in Fig. 14, without compromising the quality of the generation as highlighted in other part of the experimentation.

V. Conclusion and Future Work

The relation between circuit structure and vulnerability or attack resiliency is complex. It is not a direct one-to-one situations that one can easily prove or formulate analytically. Machine intelligence is the best way to derive such complex relations. A synthetic benchmark generation framework is not just the backbone needed for training the machine, it is the platform to incorporate the derived relation in future. The optimization to ensure the diversity of generated benchmark is essential to cover the boundary condition and issues in such learning algorithms. The contributions of this paper both help innovative techniques to demonstrate effectiveness with numerous sample benchmarks and provide the means to properly utilize machine intelligence towards determining structural aspects of mitigating vulnerability.

Specifically, in this paper, we have provided comprehensive details on the synthetic benchmark generation frameworks. We proposed EigenCircuit, a major upgrade from our previous work [17] with different objective function. We presented theoretical details and experimental results of features and scalability of both versions. The experimental results compared the resultant synthetic benchmarks with existing CAD benchmarks structurally and functionally. We also presented the experimental findings on the applicability of the proposed method. We have published both versions of the framework as open-source in Trust-Hub website for researchers to use. Also, from the analysis of performance of generated benchmarks against security researches, we plan to publish the benchmarks that are on the extreme ends on the feature spectrum, i.e., showed best and worst resiliency against attacks and so on. As a continuation, we are investigating the effect of varying circuit structure on attack resiliency, and training ML tool with numerous generated synthetic benchmark to predict vulnerability. This can lead to AI to detect and mitigate vulnerability in designs. Along with that, in future, we would extend the work to generate sequential circuit benchmarks with state elements and feedback loops. We would also like to focus on co-optimization of structure and functionality in one platform.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1651701. The authors would also like to acknowledge the Python parts of EigenCircuit and Custom Generation tool performed by Daniel Capecci and Princess Lyons.

References

[1] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, pp. 10–25, 2010.

[2] D. Forte, S. Bhunia, and M. M. Tehranipoor, *Hardware protection through obfuscation*. Springer, 2017.

[3] M. Yasin, J. J. Rajendran, and O. Sinanoglu, *Trustworthy Hardware Design: Combinational Logic Locking Techniques*. Springer, 2020.

[4] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[5] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *Proceedings of IEEE Int'l Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[6] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Circuits and Systems, 1989., IEEE International Symposium on*, May 1989, pp. 1929–1934 vol.3.

[7] K. Koźmiński, "Benchmarks for layout synthesis—evolution and current status," in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991, pp. 265–270.

[8] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[9] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *2013 IEEE 31st international conference on computer design (ICCD)*. IEEE, 2013, pp. 471–474.

[10] S. Wei, K. Li, F. Koushanfar, and M. Potkonjak, "Hardware trojan horse benchmark via optimal creation and placement of malicious circuitry," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 90–95.

[11] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, 2017.

[12] S. Amir, B. Shakya, X. Xu, Y. Jin, S. Bhunia, M. Tehranipoor, and D. Forte, "Development and evaluation of hardware obfuscation benchmarks," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 142–161, 2018.

[13] K. G. Liakos, G. K. Georgakilas, S. Moustakidis, P. Karlsson, and F. C. Plessas, "Machine learning for hardware trojan detection: A review," in *2019 Panhellenic Conference on Electronics & Telecommunications (PACET)*. IEEE, 2019, pp. 1–6.

[14] P. Chakraborty, J. Cruz, and S. Bhunia, "Sail: Machine learning guided structural analysis attack on hardware obfuscation," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018, pp. 56–61.

[15] L. Alrahis, S. Patnaik, F. Khalid, M. A. Hanif, H. Saleh, M. Shafique, and O. Sinanoglu, "Gnnunlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," *arXiv preprint arXiv:2012.05948*, 2020.

[16] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016, pp. 3–26.

[17] S. Amir and D. Forte, "Adaptable and divergent synthetic benchmark generation for hardware security," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[18] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Transactions on computers*, vol. 100, no. 12, pp. 1469–1479, 1971.

[19] H. Van Marck, D. Stroobandt, and J. Van Campenhout, "Towards an extension of rent's rule for describing local variations in interconnection complexity," in *Proc. 4th Intl. Conf. for Young Computer Scientists*, 1995, pp. 136–141.

[20] M. Hutton, J. P. Grossman, J. Rose, and D. Corneil, "Characterization and parameterized random generation of digital circuits," in *Proceedings of the 33rd annual Design Automation Conference*. ACM, 1996, pp. 94–99.

[21] M. Hutton, J. Rose, and D. Corneil, "Clustered And Iterative Synthetic Circuit Generation," 2003. [Online]. Available: http://www.eecg.toronto.edu/~jayar/software/Cgen/Cgen.html

[22] M. D. Hutton, J. Rose, J. P. Grossman, and D. G. Corneil, "Characterization and parameterized generation of synthetic combinational benchmark circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 985–996, 1998.

[23] (2017) Trust-hub website. [Online]. Available: https://www.trust-hub.org

[24] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic encryption: A fault analysis perspective," in *DATE 2012, Dresden, Germany, 2012*, 2012, pp. 953–958.

[25] D. G. Luenberger, Y. Ye *et al.*, *Linear and nonlinear programming*. Springer, 1984, vol. 2.

[26] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[27] J. Shlens, "A tutorial on principal component analysis," 2014.

[28] L. Goldstein, "Controllability/observability analysis of digital circuits," *IEEE Transactions on Circuits and Systems*, vol. 26, no. 9, pp. 685–693, 1979.

[29] S. U. Manual, "TetraMAX ATPG user guide," *Version X-2005.09*, pp. 249–264, 2005.

[30] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification," 2004. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[31] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *IEEE Intl. Symposium on HOST 2015, Washington, DC, USA, 2015*, 2015, pp. 137–143.

[32] M. Yasin, J. J. V. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.