Adaptable and Divergent Synthetic Benchmark Generation for Hardware Security

Sarah Amir and Domenic Forte
sarah.amir@ufl.edu,dforte@ece.ufl.edu
University of Florida. Gainesville, FL, USA, 32611

ABSTRACT

Benchmarking can drive the development of technologies by facilitating standardization of features for comparison of different methods. While hardware security has seen an exponential growth in innovation throughout the last decade, the lack of sufficient benchmarks for data-driven analysis is prominent. Researchers must currently rely on decades-old VLSI benchmarks, which in most cases were not designed with security evaluation in mind. Considering the present day computational power, these benchmarks lack in both quality and quantity for usage in hardware security topics such as obfuscation and hardware Trojans. Many advanced techniques, like statistical analysis and machine learning, require a large number of samples in order to sufficiently examine the feature space. In an attempt to resolve this issue, we have developed the first synthetic benchmark generation process flow. This paper describes our novel technique that utilizes linear optimization to generate an endless number of synthetic combinational benchmarks that are adaptable to user input constraints and divergent in quantifiable structural features from input reference benchmarks. Thus, our framework offers customization for generating richer and more challenging benchmarks for data-driven hardware security. Through experimentation, we verify that our benchmarks offers more structural variation than the current benchmark suites.

CCS CONCEPTS

• **Security and privacy → Security in hardware**; • **Hardware → Circuit optimization**; **Software tools for EDA**; • **Mathematics of computing → Integer programming**.

KEYWORDS

Synthetic benchmark, hardware security, linear programming

ACM Reference Format:
Sarah Amir and Domenic Forte. 2021. Adaptable and Divergent Synthetic Benchmark Generation for Hardware Security. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Benchmarks are standardized designs that allow fair evaluation and comparison of different techniques/tools on a common problem. These designs should possess a certain level of difficulty in processing them that enables quantifiable performance measure, e.g., overhead or speed, for the techniques under comparison. Benchmarks can also be utilized to validate and to determine limitations of existing or new methods. Development and utilization of common benchmarks has enabled progress in fields such as computer architecture, multimedia, machine learning, digital signal processing [11, 17, 19]. In electronic design, benchmarks played a key role in development of computer aided design (CAD) and electronic design automation (EDA) tools [14, 15]. In hardware security, benchmarks inserted with hardware Trojans are widely used to compare detection methods [23, 24, 32]. Hardware obfuscation benchmarks [3] have also been proposed to standardize evaluation of attacks on obfuscation.

The existing electronic design benchmarks [1, 2, 6–8, 16] were sufficient for their original purposes, e.g., automatic test pattern generation (ATPG), FPGA optimization, floorplanning, place-and-route, etc. Due to the lack of dedicated hardware security benchmarks, researchers in this field often rely on existing benchmarks that are not well-equipped for their need. For example, attacking simple designs like multiplier or ALU does not reflect realistic security threats or protections. For hardware obfuscation, side-channel analysis, fault injection and many other hardware security topics, benchmarks should be challenging, should contain or imitate secrets/assets, or represent valuable intellectual property (IP) that is worth protection. Further, machine learning and statistical analysis are being increasingly used in hardware security [4, 9]. To correctly model a system, millions of proper sample are often needed especially for deep learning. To both use machine learning and verify the methods being proposed that use it, it is undeniable that large number of circuit benchmarks are essential. Further, such benchmarks should have diverse characteristics in order to fully explore corner cases.

In this work, we propose a novel synthetic benchmark generation framework to tackle this issue. While our framework supports qualitatively superior and quantitatively unbounded synthetic benchmarks for any application, we focus the demonstration on hardware security since it is a timely topic. The contributions of this work can be summarized as follows

- We outline the limitations of current benchmarks and synthetic benchmark generation tools.
- We propose and implement a framework for generating combinational benchmarks[1]. Features are extracted from preprocessed circuits and an integer linear programming (ILP)

[1]Should this paper be accepted, we pledge to make our tools publicly available for the research community.

is used to generate benchmarks that are optimally divergent from a set of reference benchmarks in terms of structure. The framework can generate an arbitrarily large number of benchmarks with characteristics such as number of nodes, primary inputs, etc. set by the user.
- We also introduce a second logic customization step that assigns operators (NAND, XOR, etc.) to each node in the circuit to order to make the benchmarks more targeted to specific applications in hardware security.
- We demonstrate the scalability of the proposed approach and show that the resulting benchmarks contain different structures and security features compared to existing ones.

The rest of the paper is organized as follows. The motivation and necessity of synthetic benchmarks in hardware security is further explained in Section 2. We also discuss the limitations of prior synthetic benchmark generation tools in the field of EDA and why they leave a gap. The synthetic benchmark generation flow with associated features, optimization techniques and generation processes, and limitations are described in Section 3. The experimental results are presented in Section 4 while Section 5 concludes the paper and discusses future work.

2 MOTIVATION AND RELATED WORK

In the design of systems and tools, a set of benchmarks is often needed to validate the output and to evaluate the performance, especially against the current state-of-the-art. In the 1980s and 1990's, the era of EDA tool development, standard circuit benchmarks were introduced in renowned conferences [6–8, 16]. With continual growth of development of EDA, numerous larger designs were sought, hence started the process of synthetic benchmark generation [12]. These benchmarks were constructed automatically, by modifying some existing reference circuits. Application specific algorithms were later published to meet new objectives [27, 29].

With the introduction of machine learning and artificial intelligence into design and optimization of circuits with traditional as well as security constraints, versatile and large (both in size and number) benchmark sets are necessary. Machine learning is data-driven and demands many training samples to successfully learn from and make predictions on; deep learning in particular can require millions of samples. This is why in applications such as face recognition, a great deal of effort is devoted to data augmentation [22, 25, 26]. Even small circuits can have exponentially large number permutations in both logic and structure that should be explored by machine learning methods. However, the existing benchmarks are simply not enough – both in number and variety.

2.1 Benchmarks in Hardware Security

In hardware security research, researchers have been forced to either evaluate/compare approaches using these three-decade old obsolete circuit, or use open-source circuit designs. We have looked into 43 combinational hardware obfuscation techniques and attack publications, and Figure 1 represents the benchmarks they used to validate their approaches. Even though the methods claims their efficacy for modern designs, the use of decade old benchmarks in most of them question their scalability and more. This does not facilitate the need of significant number of samples or variety for data-driven approaches in hardware security.

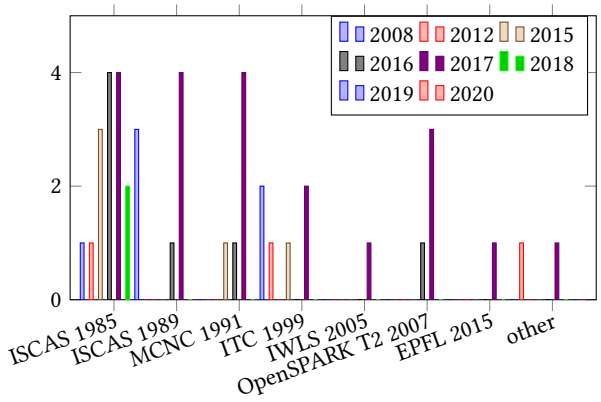


Figure 1: A summary of the benchmarks used in hardware obfuscation techniques and attacks from 2008–2020.

Another key issue of using benchmarks intended for CAD development in security analysis is objective. Hardware obfuscation, hardware Trojan insertion, side channel analysis, etc. are significantly different applications and have separate set of criteria for the benchmarks to be considered "good" gauges of usefulness. For example, where a benchmark for Trojan insertion should have many low observable nodes [23, 24, 32], while a benchmark for obfuscation or side channel analysis should have assets worth protecting [3]. None of these criteria can be explicitly found in CAD oriented benchmarks that researchers are using since they were not originally designed for such applications. Note that this is not a direct criticism of the aforementioned works [3, 23, 24, 32] which do an excellent job of modifying benchmarks for security purposes by inserting Trojans and key gates; nevertheless, they are severely limited by the original benchmarks that they modify.

Methods that allow a designer or researcher to emphasize specific hardware security criteria in generation process can result in such objective-oriented benchmarks. The only solution we envision to support this is an automated system that generates synthetic benchmarks.

2.2 Synthetic Benchmark Generation

In light of the above, we briefly introduce preliminary work on synthetic circuit generation here. Synthetic benchmark generation was initially performed as logic graph modification [18]. Circuits were analyzed as graphs [30] until one of the earliest works to offer a detail generation process from scratch in [12]. Additional works used partitioning and clustering of designs [14, 31].

Synthetic combination circuit benchmark generation in [12] followed a method where the structural data of reference circuits was extracted by C_{CIRC} tool [13], as arrays of parameters, and another generation tool, C_{GEN} [13], utilized that data to construct a new circuit based on these parameters. Some of the key parameters extracted from the circuit were maximum depth level, number of logical unit or gates at each depth level, number of inputs to and outputs from each depth level, number of wires of each possible functional length (measured in depth levels), etc. The step-by-step generation process used this data. The gates and input-outputs were assigned in each depth level, and then the wires were placed

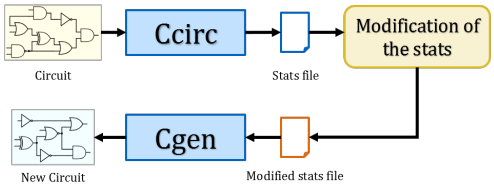


Figure 2: Synthetic benchmark generation flow in [12, 13]

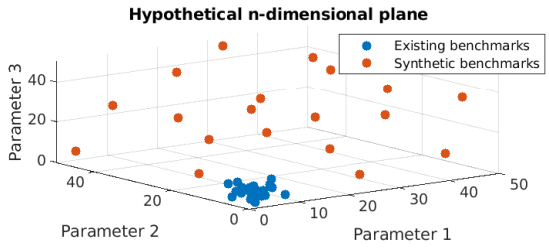


Figure 3: A hypothetical $n = 3$ dimensional plane showing the structural parameter distribution of existing (blue) vs. ideal synthetic benchmarks (orange).

to complete the design. A simplified overview of the generation process is shown in Fig. 2. As only the extracted parameters are used, the new synthetic circuit are structurally very similar to the original reference circuit.

The lack of versatility in such existing synthetic generation systems [12, 13] is a key problem in using them for data-driven applications. Some prior works are based on synthesizing new benchmarks by mutation of an existing one [3, 12]. The mutated benchmarks are somewhat useful, but do not significantly differ in structure from their references. For example, if we observe the reconvergence, a parameter representing the nestedness of a circuit, of some common benchmarks, we can see most of them lies within the range of 0.1 to 0.4 on the scale of 0 to 1. However, a design of multiplier, which is found very resilient to state-of-art SAT attack against logic locking, has a higher reconvergence of around 0.8 [3]. In order to study the relation between structure and attack resiliency, one needs large number of benchmarks which are significantly different, in size, delay, shape, reconvergence, and so on. If we imagine a hypothetical multidimensional space, like the example in Fig. 3, where each axis represents a unique structural quality of circuit designs, almost all existing benchmarks as well as those generated by mutation would fall in a close concentrated cluster. If one wants a design that lies far from the cluster, the horizon for the parameters must be extended.

In order to construct such "divergent" benchmarks, one possible way would be to modify the data extracted by the C_{CIRC} tool, and then use the C_{GEN} tool to generate the modified design. However, in our preliminary experiments, we found that the extracted data also included other detailed information that was not explained in the accompanying literature. If the key parameters are altered, the modification is unlikely to result in a feasible circuit and thus the C_{GEN} tool fails to generate a modified benchmark from the manually modified data. For example, increasing or decreasing the maximum depth level of the intended circuit is not possible

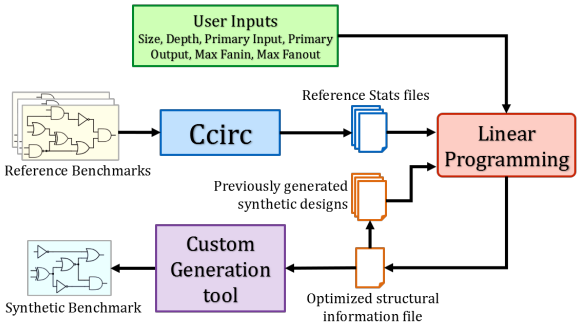


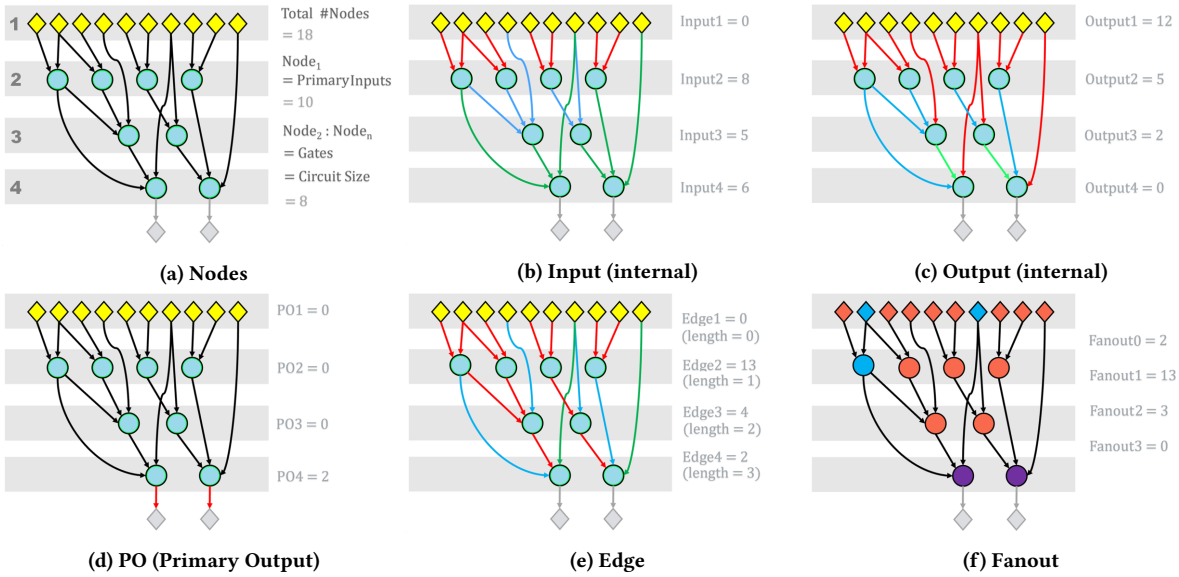
Figure 4: Overview of proposed synthetic benchmark generation framework.

without the detailed knowledge of every parameter presented in the extracted data.

As research on EDA development is past its peak, further development has largely ceased on synthetic benchmark generation. Thus, as part of this work, we developed a new generation tool, partially based on the idea used in C_{GEN} tools [12]. In the next section, we describe our automated flow for generating synthetic, combinational circuit benchmarks that overcomes the above limitations. The proposed approach is *adaptive*, allowing the user to specify parameters of the desired circuit (e.g, number of gates, depth, number of primary inputs, etc.) and associated gates (e.g., maximum fanin and fanout). Such control is important in generating circuits are both realistic and allow one to explore the scalability of approaches to be benchmarked or evaluated. The primary method of optimization is linear programming which allows the solution – a synthetic benchmark set – to match this input spec, meet the requirements of a feasible circuit, e.g., no floating nets or nodes, and achieve the main objective, i.e., have fundamentally different structural parameters than reference benchmarks. The latter can, therefore, enable realization of the *divergent* cluster shown in Figure 3. Our approach is also complemented by a procedure for assigning logical operations for the nodes in the resulting circuits based on the needs of the benchmarking application, such as logic locking or hardware Trojan insertion. As proof-of-concept, we explore features like observablity, Hamming distance, canonical form [28], etc. that can play valuable roles in evaluating attack/detection resilient qualities of circuits.

3 PROPOSED METHODOLOGY AND IMPLEMENTATION

A simplified overview of the proposed synthetic benchmark generation flow and its components is presented in Figure 4. Three inputs are provided to the flow: (1) one or more reference circuits – these could be existing benchmarks; and (2) the basic features of the synthetic benchmarks produced – in this paper, these are number of nodes (i.e., circuit *Size*), depth (n), number of primary inputs and outputs (PI and PO), and maximum fanin/out of nodes (mFI and mFO) in the circuit; and (3) the desired number of synthetic benchmarks to be produced as output. The generation flow builds on the basic idea of C_{GEN} tool [12], but with critical modifications



(a) Nodes **(b) Input (internal)** **(c) Output (internal)**

(d) PO (Primary Output) **(e) Edge** **(f) Fanout**

Figure 5: Example of structural features from the graph representation of a circuit. The diamonds on top and bottom represent inputs and outputs respectively. The circles represent gates.

to satisfy our objectives. Specifically, we construct synthetic benchmarks from more direct and detailed structural information than the one generated by the original C_{CIRC} tool. For example, along with number of logical units per depth level, it includes number of wires of each length generating from each depth level. This detailed data is thorough and holds a complete information of a netlist structure. Any valid modification in the data would result in a different design. From these references stats, integer linear programming is used to generate a synthetic benchmark that is the most structurally different from the reference stats. Stats from the synthetic benchmark are extracted and included as feedback to the linear programming to generate additional synthetic benchmarks in an iterative fashion. In this way, generated synthetic benchmarks are more divergent and can cover the whole space as shown in Figure 3. Finally, the structural parameters of these synthetic benchmarks are passed to a custom generation tool that assigns logical operations (NAND, NOR, etc.) to the nodes in these benchmarks based on another objective (e.g., higher controllability).

3.1 Structural Features

Most of the structural features that we extract from references and optimize are vectors of length n where n represents the maximum logical depth. Circuit depth is the length of the longest path from the input to the output (in terms of number of gates in the path).

(1) Node distribution (*Nodes*): Defined as array of gates in each depth. Primary inputs are placed in the first level (depth = 1). $Nodes_i$ represents the number of nodes with depth i in the circuit. $Node_i$ is the number of gates that are on $i-1$ delay from the input (figure 5a).
(2) Internal inputs distribution (*Input*): Array of number of internal inputs to each depth. $Input_i$ is the internal input to depth i. As no internal input goes in primary inputs, $Input_1 = 0$ (figure 5b).
(3) Internal outputs distribution (*Output*): Array of internal outputs, where $Output_i$ refer to internal outputs coming from depth i. The last depth n does not have any internal outputs, so $Output_n = 0$ (figure 5c).
(4) Primary outputs distribution (*PO*): Array of primary outputs drawn from each depth. PO_i would refer to number of primary outputs that have a delay of $i-1$ from the primary inputs (figure 5d).
(5) Wire length distribution (*Edge*): This is an array that contains number of wires of each length in the entire circuit. Here length is the difference in maximum depth level between a node and its fanout node. Here, $Edge_i$ represents total number of wires in the design that has length $i-1$ (figure 5e).
(6) Fanout distribution (*Fanout*): Array containing number of gates with each fanout in the entire circuit. $Fanout_i$ refer to total number of gates in the design that has fanout = i. In a practical design $Fanout_2$ is the largest number, followed by $Fanout_3$ (figure 5f).

3.2 Constrained Optimization

Seemingly, it would be easy to generate a set of features than differs from reference features using basic optimization techniques. However, because the resulting benchmarks must be genuine circuits, a constrained optimization approach is critical. In this section, we discuss the objective functions and constraints (some withheld for brevity) in order to achieve divergent benchmarks.

3.2.1 Nonlinear vs. Linear Programming. Initially, we explored nonlinear optimization methods where the objective is to find $\min_x f(x)$ where $f(x)$ is a nonlinear function [20]. For example, the equation representing the Euclidean distance between two vectors in a n

dimension space is defined by

$$\sqrt{\sum_{i=1}^{n}(x_i - x_i')^2}$$

where $x_1, x_2, ..., x_n$ are dimensions of one vector and $x_1', x_2', ..., x_n'$ are elements of a second vector. For our purposes, each x' would represent a structural feature per depth of the reference benchmarks while each x would represent a structural feature per depth of the resulting synthetic benchmark. Unfortunately, we found that nonlinear optimization was not suitable for our application because the resultant solution (x_1,x_2, etc.) contains fractions. Our structural features used in real circuits such as number of nodes, edges, inputs, etc. can only take on integer values. A naïve approach to resolve this could be to round up the optimized fractional solution but this introduces two major errors. First, the solution is no longer optimal; second and even worse, rounding can cause the resulting circuit to be invalid, introducing one or more discrepancies.

As the circuit parameters are real integer numbers, we found integer linear programming to be the more appropriate choice. This optimization technique [20] finds the minimum of a problem specified by

$$\min_x f^T x \text{ s.t. } Ax \leq b \tag{1}$$
$$A_{eq}x = b_{eq} \tag{2}$$
$$lb \leq x_j \leq ub \ \forall j \tag{3}$$
$$x_j \in \mathbb{Z} \ \forall j \tag{4}$$

where f, x, b, b_{eq}, lb, and ub are vectors. A and A_eq are matrices that represents the coefficients of x in linear inequality and equality constraints, respectively. The third and fourth constraints restrict the range of each feature x_j to $[lb, ub]$ and to an integer, respectively. In order to use distances in linear optimization, linearization of the distance equation is required. Therefore, the workable distance equation is defined by

$$\sum_{i=1}^{n} \left| x_i - x_i' \right|$$

where the symbols represent the same meaning as in the non-linear distance equation.

3.2.2 Problem Formulation. The linear optimization problem can be defined as finding the maximum of a problem specified by-

$$\begin{aligned} &\max_x f^T(x) \ where \\ &f(x) = \sum_{param} \sum_{i=1}^{n} \left| param_i - \frac{\sum_{j=1}^{N} param_{j_i}}{N} \right| \\ &\forall param \in \{Node, Input, Output, PO, Edge, Fanout\}, \\ &subject\ to \begin{cases} x\ integer \\ Constraints \\ Bounds \end{cases} \end{aligned} \tag{5}$$

where N is the number of reference designs; x is the concatenated array of the optimal distribution of parameters that can later be translated to circuit in netlist form. Optimization constraints are discussed in section 3.2.3. 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.

Here, the dimension i corresponds to a single depth level for each structural feature. If the intended synthetic benchmark has maximum depth n and the number of feature types is p, then the total number of dimensions would be $d * p$. For example, for *param = {Node, Input, Output, PO, Edge, Fanout}*, $p = 6$. The resultant distribution of parameters reflects maximum possible difference in each depth for each parameter.

Note that for this formulation, the depth of each reference must be the same. Since many of the references may have different depth, we developed a pre-processing step to adjust their depth. This is done by either adding buffer layers in a uniform fashion or truncating the circuit (more details in Appendix A.2).

3.2.3 Constraints. The constraints and bounds ensure the resultant data represents a valid circuit. In our work, we have employed *more than 25 validity constraints* (not shown for brevity) to ensure that the resulting features result in a valid circuit solution. For brevity, we only cover a few of the most significant of those constraints here:

(i) As mentioned earlier, the circuit size (*Size*) in terms of total number of nodes, number of primary inputs (*PI*) and number of primary outputs (*PO*) are defined by the user as input. Nodes at the first depth represents primary inputs, hence should be equal to user input of such. The total nodes from depths 2 to n should be equal to user-defined circuit size. The total number of primary outputs of all the depths should be equal to the total number of primary outputs. These are expressed by the following equality constraints:

$$PI = Node_1 \tag{6}$$
$$Size = \sum_{i=2}^{n} Node_i \tag{7}$$
$$PO = \sum_{i=1}^{n} PO_i \tag{8}$$

(ii) The number of nodes in any depth should not be more than maximum possible internal inputs and primary outputs in the later part of the circuit. Similarly, the nodes in any depth should not be more than maximum possible outputs in the prior depths. Both are also related to the maximum fanin (*mFI*) and fanout (*mFO*) of nodes in the circuit. These can be expressed using the following inequality constraints:

$$Node_i \leq \sum_{j=i+1}^{n} Node_j \times mFI + \sum_{j=i+1}^{n} PO_j \ \forall \ i \tag{9}$$
$$Node_i \leq \sum_{j=1}^{i-1} Node_j \times mFO \ \forall \ i \tag{10}$$

(iii) The maximum possible input to a depth is $Node_i$ times *mFO*, where *mFO* refers to maximum fanout (user-defined parameter). Inputs to depth i ($Input_i$) should not be more than maximum possible input to that depth. Also, total outputs of any depth, internal and external, should not be more than

maximum possible output of that depth, which is the number of nodes in that depth times maximum fanout of each node. These can be expressed using the following inequality constraints:

$$Node_i \leq Input_i \leq Node_i \times mFI \; \forall \; i \tag{11}$$

$$Node_i \leq Output_i + PO_i \leq Node_i \times mFO \; \forall \; i \tag{12}$$

With an additional dozens of constraints (more shown in Appendix), the $f(x)$ is maximized to determine the unique circuit structure that is farthest from the reference in every axis in a hypothetical circuit plane. *In this way the optimization makes sure that the generated benchmark suite is diversified, unique, arbitrarily large, and designer-specified to match the application.* Our results in Section 4 shows how these optimal benchmarks contrast in structure from input reference clusters.

3.3 Custom Logic Assignment

The above optimization determines the structure of synthetic benchmark in terms of logical units (nodes) and their interconnections (edges), like a graph. It is represented by an array of numbers, similar to the output of C_{CIRC} tool. However, the logic to be defined in the logical units are not specified by the optimizer. This numerical representation is converted to a workable library dependent or independent verilog netlist by the our Custom Generation tool (see Figure 3). The assessment of logic in the generation stage can be random or heuristically driven.

- For random logic assignment, the generation tool determines logic for each node, based on the number of input and output to that node. Given a set of choices, a pseudo-random function is used to choose the logic. In this assessment, rate of occurrence for all logic choices are same.
- For heuristically driven, we assume that there is some hardware security application. For example, one can generate good synthetic benchmarks for hardware Trojan insertion, by assigning logic that to maximizes controllability or observability. Following the equations of determining controllability or observability of logic gates [10], the logic choices can be chronologically sorted. From our preliminary experiments, we have seen an strict law of maximizing these parameters (i.e. making the design harder to control or observe) results in a selection of all 'AND'/'NAND' logic for all *nodes*. This type of design can be significantly reduced to smaller one, and not a good realistic design. Thus, we suggest a better selection to increase controllability or observability by increasing the rate occurrence for 'AND'/'NAND' logic over that for 'OR'/'NOR' logic. Such heuristics will be explored for applications such as logic locking and side channel analysis in future work.

3.4 Current Limitations

In this initial implementation of the proposed benchmark generation flow, we only consider combinational benchmarks for simplicity, but plan to extend it to sequential designs in future work. Regarding the implementation, the objective function of the optimization is the distance between target design and average of reference design. This can be improved later by optimizing the total distances between target design and each reference designs. Optimization of structure and logic are preformed separately resulting in the need for custom generation tools. In future work, we'd prefer to combine this into one optimization step. This is important because synthesis tools can optimize and alter the structure of our synthetic benchmarks based on their logic, thus resulting in sub-optimal differences from the references.

4 EXPERIMENTAL RESULTS

Through detailed analysis with existing and novel structural parameters, we tested how the generated combinational benchmarks possess differentiating qualities compared to existing standard benchmarks. We also analyzed the scalability of our generation flow for design parameter variation, and relations between controllable design parameters of our method and significant features of resultant benchmarks.

In our experimental setup, we have used integer linear programming from MATLAB on RedHat Linux Server to perform the optimization. The complete structure of the netlist is defined by the optimizer, except the logic assignment. The optimized structural data is converted to a verilog netlist by our generation tool developed on Java 1.8 platform. Performance of the framework and synthetic benchmarks are elaborated in next subsections.

4.1 Divergence of Synthetic Benchmarks

Each synthetic benchmark is generated by optimizing the maximum distances of parameter distributions between the average distribution of all references along with already generated benchmarks and that of the design to be generated. Figure 6 plots one such sample of reference and resultant distribution of parameters. The difference in the structure is clearly visible.

The synthetic benchmarks are analyzed for structural parameters along with existing benchmarks. To compare, we worked with existing combinational benchmarks for ISCAS [7] benchmark suites. As for structural parameters, we analyzed each circuit with C_{CIRC} tool [13] to acquire the node distribution. Multiple ad hoc metrics were used to compare the benchmarks to quantify the differences in figure 6. First, the depth at which maximum number of node is placed, normalized with maximum depth to per unit value, is termed as "peak" of the node distribution. Second, another parameter that is significant to express node distribution is the standard deviation. To compare, we normalized this standard deviation to per unit value. Finally, normalized average logic cone size is a metric that reflects how nested the logic gates are in the design. It is defined as

$$\sum_{i=1}^{N} (Fanin_i + Fanout_i)/N^2 \tag{13}$$

where N is the total number of logic gates in the circuit, and $Fanin_i$ and $Fanout_i$ are calculated as number of logic gates in corresponding cones of the ith node/gate.

The experimental results, as in figure 7, show that the synthetic benchmarks are more divergent than existing reference benchmarks. The blue dots represents the ISCAS combinational benchmarks and the red dots are for synthetic benchmarks. Even with the few instances, we can observe synthetic benchmarks are expanding to areas outside the clusters of existing benchmarks. As the features like gate per depth of synthetic benchmarks are customizable, the positions of red dots in this plots are even more flexible.

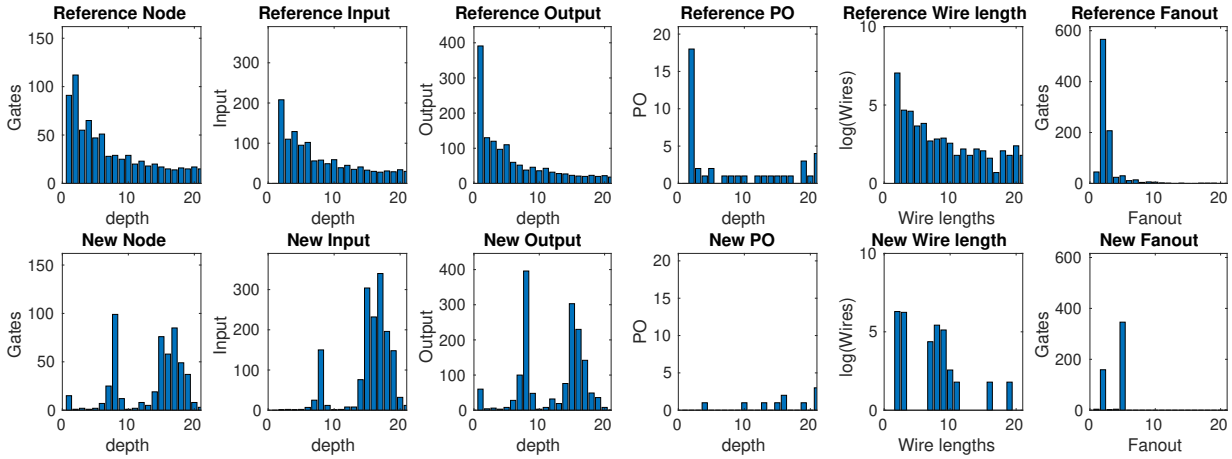


Figure 6: Distribution of structural features where (top row) represents the average distribution for the reference designs and (bottom row) represents the distribution for a synthetic benchmarks generated by the proposed flow.

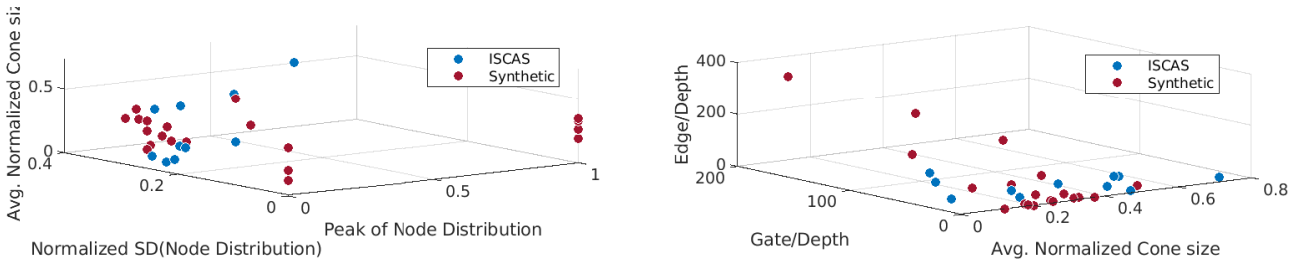


Figure 7: Comparing existing and synthetic benchmarks on the basis of structural parameters. Benchmarks are compared based on (a) Node distribution parameters (position of peak and standard deviation) and normalized average logic cone size and on (b) gates and edges per depth and average logic cone size

4.2 Scalability

The times needed for ILP optimization per benchmark versus the depth is presented in figure 8. The total number of gates is also varying and represented by a different line. Since the the number of optimization variables and number of equations are quadratically dependent on intended maximum depth, the ILP optimization time increases with depth. However, the size of design in terms of number nodes/gates does not have significant effect on this time. While benchmarks need only be generated once and can be used in many applications, the optimization time can be improved in future work.

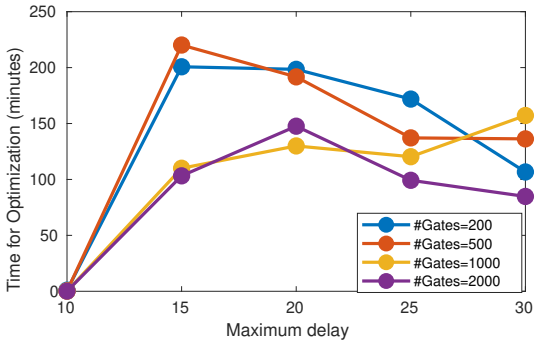


Figure 8: Generation times of circuits with varying depth and number of gates (size).

4.3 Relation Between Controllable Features and Security Critical Parameters

There are security critical structural features that directly relates to resiliency against attacks in hardware security. For example, the larger the number of clauses and literals in CNF form of a circuit, the longer it takes for SAT attacks to execute on logic locked circuits [28]. The controllability and observability of a design also relates to how easily an obfuscation or logic locking key gate can be excited and its key value be propagated to primary outputs in Key sensitization attack [33]. These critical features are found to be directly proportional to circuit depth or maximum delay level, as shown in Figure 9. In experiment, the controllability and observability are calculated with Synopsys TetraMAX [21] and number

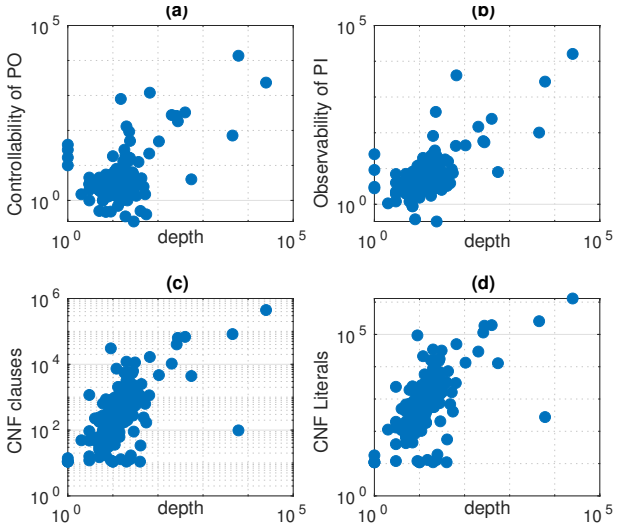


Figure 9: Relationship of controllable features in synthetic benchmark generation and observable parameters that can impact security of logic locking: (a) Controllability of primary outputs, (b) observability of primary inputs, (c) Number of clauses and in CNF form (d) literals in CNF form versus maximum depth.

of clauses and literals of CNF form is measured with Barkley ABC tool [5].

In our synthetic benchmark generation framework, depth is a user defined parameter. Thus, in order to study the resiliency of a logic locking scheme against those attacks, the depth of the synthetic benchmarks can be made arbitrarily small. Whereas if one wants to ignore such attacks and study some other logic locking attacks, the depth can be made larger.

5 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a novel adaptable and divergent synthetic benchmark technique utilizing linear optimization. We have outlined the problem, detailed the proposed methodology and elaborated on the experimental outcomes. It is our believe that the synthetic benchmarks developed through this framework would be crucial in development of many fields, especially data-driven and machine learning for hardware security. We plan to release the initial tool to the public to enable such research. In future work, we aim to expand its capabilities to sequential circuits and improve on its current limitations. Examples include alternative objective functions and depth adjustment methods, more scalable optimization methods, and combined logic/structure optimization.

ACKNOWLEDGMENTS

This work is supported by NSF under grant CNS 1651701. Authors would also like to thank Shahin Tajik, Bicky Shakiya, Farhaan Fowze, Sazadur Rahman for their support.

A APPENDIX

A.1 Additional Constraints

A.1.1 Edge Table. We construct an edge table where each row represents the wires starting from each depth level and each columns represents the length of wires. Here $d = depth$, and $l = length$.

$$Edge\,Table = \begin{bmatrix} E_{1,1} & E_{1,2} & \dots & E_{1,n} \\ E_{2,1} & E_{2,2} & \dots & E_{2,n} \\ \dots & \dots & E_{d,i} & \dots \\ E_{n,1} & E_{n,2} & \dots & E_{n,n} \end{bmatrix}$$

Edge is a vector of number of edges of each length. The $Edge_i$ represents total number of edges of $length = i$ in the entire circuit. Each node in each depth should be connected to nodes of prior depths with at least one edge of $length = 1$. This is mandatory to ensure the node is placed according to the optimizer, not floating to other depths. Edges starting from $depth = d$, and $length = l$ is the input to $depth = d + l$. The sum of any length of edges starting from a depth is the output for that depth.

$$\begin{aligned} Node_i &\le E_{i,1}; \quad \forall\, i \\ Input_i &= \sum_{d=1}^{i-1} \sum_{l=1}^{i-1} E_{d+l=i} \quad \forall\, i \\ Output_i &= \sum_{l=1}^{n} E_{i,l}; \quad \forall\, i \\ Edge_i &= \sum_{d=1}^{n} E_{d,i} \quad \forall\, i \end{aligned} \tag{14}$$

There is no edge starting from final depth. Edges starting from any depth should not be longer than possible lengths. Edge starting from $depth = d$, with $length = l$ is input to $depth = d + l$. So, $d + l$ should not be more than n, i.e. number of edge of d+l more than n is null.

$$E_{d=n} = E_{d+l>n} = 0 \;\forall\, d \;\forall\, l \tag{15}$$

A.1.2 Buffer Layer Avoidance. This is not an essential constraint, but is an additional one to avoid repetitive buffer layers caused by the optimization. Though design containing the chain of buffer is not an impossible circuit, but is considered not a good divergent design, as buffer layers can easily be removed. The condition for buffer layer $(i + 1)$ after layer i can be identified as $\forall$i $\in$ {1,...,n} $Node_i = Output_i = Node_{i+1} = Input_{i+1}$, but not exclusively (i.e. all buffer layer would satisfy this condition, but all layers satisfying this condition are not buffer layers).

$$Node_i \neq Node_{i+1} \;\forall\, i \tag{16}$$

A.2 Depth Adjustment

The reference design parameters are either truncated or extended to match the user-defined intended depth. Assume, the reference design has maximum depth = $depth_{ref}$ and the intended design depth is d. To truncate, the circuit is split at the depth d ($d < depth_{ref}$), and the internal outputs from depth d is considered primary outputs. All other parameters are adjusted to reflect the split. To extend when $d > depth_{ref}$, new buffer layers are inserted in between existing delay layers. If $a_1 > \dfrac{d}{depth_{ref}} > a_2$, then $a_1 - 1$ buffer layers are inserted after first n_1 layers, and $a_2 - 1$ layers are inserted after the rest of the n_2 layers, where $n_1 * a_1 + n_2 * a_2 = d$.

REFERENCES

[1] Christoph Albrecht. 2005. IWLS 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS)*. 9.

[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*.

[3] Sarah Amir, Bicky Shakya, Xiaolin Xu, Yier Jin, Swarup Bhunia, Mark Tehranipoor, and Domenic Forte. 2018. Development and evaluation of hardware obfuscation benchmarks. *Journal of Hardware and Systems Security* 2, 2 (2018), 142–161.

[4] Johanna Baehr, Alessandro Bernardini, Georg Sigl, and Ulf Schlichtmann. 2020. Machine learning and structural characteristics for reverse engineering. *Integration* 72 (2020), 1–12.

[5] Berkeley Logic Synthesis and Verification Group. 2004. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/

[6] F. Brglez, D. Bryan, and K. Kozminski. 1989. Combinational profiles of sequential benchmark circuits. In *Circuits and Systems, 1989., IEEE International Symposium on*. 1929–1934 vol.3. https://doi.org/10.1109/ISCAS.1989.100747

[7] F. Brglez and H. Fujiwara. 1985. 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., 677–692.

[8] F. Corno, M.S. Reorda, and G. Squillero. 2000. RT-level ITC'99 benchmarks and first ATPG results. *Design Test of Computers, IEEE* 17, 3 (Jul 2000), 44–53. https://doi.org/10.1109/54.867894

[9] Rana Elnaggar and Krishnendu Chakrabarty. 2018. Machine learning for hardware security: opportunities and risks. *Journal of Electronic Testing* 34, 2 (2018), 183–201.

[10] L Goldstein. 1979. Controllability/observability analysis of digital circuits. *IEEE Transactions on Circuits and Systems* 26, 9 (1979), 685–693.

[11] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 3–14.

[12] Michael Hutton, Jerry P Grossman, Jonathan Rose, and Derek Corneil. 1996. Characterization and parameterized random generation of digital circuits. In *Proceedings of the 33rd annual Design Automation Conference*. ACM, 94–99.

[13] Michael Hutton, Jonathan Rose, and Derek Corneil. 2003. Clustered And Iterative Synthetic Circuit Generation. http://www.eecg.toronto.edu/~jayar/software/Cgen/Cgen.html

[14] Michael D Hutton, Jonathan Rose, Jerry P Grossman, and Derek G Corneil. 1998. Characterization and parameterized generation of synthetic combinational benchmark circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17, 10 (1998), 985–996.

[15] Nevin Kapur, Debabrata Ghosh, and Franc Brglez. 1997. Towards a new benchmarking paradigm in EDA: analysis of equivalence class mutant circuit distributions. In *Proceedings of the 1997 international symposium on Physical design*. 136–143.

[16] Krzysztof Koźmiński. 1991. Benchmarks for layout synthesis—evolution and current status. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*. 265–270.

[17] Martin Kutter and Fabien AP Petitcolas. 1999. Fair benchmark for image watermarking systems. In *Security and Watermarking of Multimedia Contents*, Vol. 3657. International Society for Optics and Photonics, 226–239.

[18] Bernard S Landman and Roy L Russo. 1971. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on computers* 100, 12 (1971), 1469–1479.

[19] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*. IEEE, 330–335.

[20] David G Luenberger, Yinyu Ye, et al. 1984. *Linear and nonlinear programming*. Vol. 2. Springer.

[21] Synopsys User Manual. 2005. TetraMAX ATPG user guide. *Version X-2005.09* (2005), 249–264.

[22] Luis Perez and Jason Wang. 2017. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).

[23] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. 2013. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st international conference on computer design (ICCD)*. IEEE, 471–474.

[24] Bicky Shakya, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor. 2017. Benchmarking of Hardware Trojans and Maliciously Affected Circuits. *Journal of Hardware and Systems Security* 1, 1 (2017), 85–102.

[25] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 60.

[26] Patrice Y Simard, David Steinkraus, John C Platt, et al. 2003. Best practices for convolutional neural networks applied to visual document analysis.. In *Icdar*, Vol. 3.

[27] Dirk Stroobandt, Peter Verplaetse, and Jan Van Campenhout. 2000. Generating synthetic benchmark circuits for evaluating CAD tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19, 9 (2000), 1011–1022.

[28] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *IEEE Intl. Symposium on HOST 2015, Washington, DC, USA, 2015*. 137–143.

[29] Tao Wan and M. Chrzanowska-Jeske. 2004. Generating random benchmark circuits for floorplanning. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, Vol. 5. V–V.

[30] Herwig Van Marck, Dirk Stroobandt, and Jan Van Campenhout. 1995. Towards an extension of Rent's rule for describing local variations in interconnection complexity. In *Proc. 4th Intl. Conf. for Young Computer Scientists*. 136–141.

[31] Peter Verplaetse, Jan Van Campenhout, and Dirk Stroobandt. 2000. On synthetic benchmark generation methods. In *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No. 00CH36353)*, Vol. 4. IEEE, 213–216.

[32] Sheng Wei, Kai Li, Farinaz Koushanfar, and Miodrag Potkonjak. 2012. Hardware Trojan horse benchmark via optimal creation and placement of malicious circuitry. In *Proceedings of the 49th Annual Design Automation Conference*. 90–95.

[33] Muhammad Yasin, Jeyavijayan J. V. Rajendran, Ozgur Sinanoglu, and Ramesh Karri. 2016. On Improving the Security of Logic Locking. *IEEE Trans. on CAD of Integrated Circuits and Systems* 35, 9 (2016), 1411–1424.