cryptography

MDPI

Article

Reevaluating Graph-Neural-Network-Based Runtime Prediction of SAT-Based Circuit Deobfuscation

Guangwei Zhao and Kaveh Shamsi *

Department of Electrical and Computer Engineering, University of Texas at Dallas, Richardson, TX 75080-3021, USA; guangwei.zhao@utdallas.edu
* Correspondence: kaveh.shamsi@utdallas.edu

Abstract: Logic locking is a technique that can help hinder reverse-engineering-based attacks in the IC supply chain from untrusted foundries or end-users. In 2015, the Boolean Satisfiability (SAT) attack was introduced. Although the SAT attack is effective in deobfuscating a wide range of logic locking schemes, its execution time varies widely from a few seconds to months. Previous research has shown that Graph Convolutional Networks (GCN) may be used to estimate this deobfuscation time for locked circuits with varied key sizes. In this paper, we explore whether GCN models truly understand/capture the structural/functional sources of deobfuscation hardness. In order to tackle this, we generate different curated training datasets: traditional ISCAS benchmark circuits locked with varying key sizes, as well as an important novel class of synthetic benchmarks: Substitution-Permutation Networks (SPN), which are circuit structures used to produce the most secure and efficient keyed-functions used today: block-ciphers. We then test whether a GCN trained on a traditional benchmark can predict the simple fact that a deeper SPN is superior to a wide SPN of the same size. We find that surprisingly the GCN model fails at this. We propose to overcome this limitation by proposing a set of circuit features motivated by block-cipher design principles. These features can be used as stand-alone or combined with GCN models to provide deeper topological cues than what GCNs can access.

Keywords: circuit deobfuscation; Graph Convolutional Network; SAT attack; substitution-permutation network

Citation: Zhao, G.; Shamsi, K. Reevaluating Graph-Neural-Network-Based Runtime Prediction of SAT-Based Circuit Deobfuscation. *Cryptography* **2022**, *6*, 60. https://doi.org/10.3390/cryptography6040060

Academic Editor: Jim Plusquellic

Received: 24 August 2022
Accepted: 8 November 2022
Published: 22 November 2022

1. Introduction

Integrated circuits (ICs) are the most significant raw materials for modern computing systems, yet the semiconductor industry has mostly adopted a fabless business model due to rising production costs. Such an outsourced fabrication can result in severe security and privacy issues, such as IP theft, IC overproduction, and worse, the possibility of malicious modification of the design (the hardware Trojan problem). One technique proposed to hinder some of these threats is logic locking. Logic locking is based on adding additional key inputs into a design so that the circuit is unintelligible absent a correct secret key, which is programmed post-fabrication by a trusted party.

Most locking schemes are secure against naive brute-force attacks. In 2015, the Satisfiability-Based attack (SAT attack) [1,2] against locking was introduced. The SAT attack is an Oracle-Guided (OG) attack in that it uses functional queries to an unlocked IC to learn the value of the secret key bits. The attack employs a modern SAT solver to search for input patterns that can exclude incorrect keys, query them on the oracle, and use the same SAT solver to find keys that satisfy the input-output relations. The runtime of the attack is a function of the number of needed queries and the solving time for each instance. Both of these values are incredibly difficult to compute for a general circuit as they reduce to hard computational problems. Running the attack to observe its runtime can take prohibitively long, especially for hard-to-deobfuscate circuits. Then, the challenge

becomes: given restricted time and hardware resources, can we estimate the deobfuscation time without fully performing the SAT attack?

While there has been some research on runtime prediction of SAT instances, OG attacks require multiple SAT calls, some of which may be hard. Recent work [3] has suggested that a Graph Convolutional Network (GCN) [4] based solution called ICNet can be used to predict the runtime of SAT attacks. This was applied in [3] to single circuits with varying key sizes and gate types, demonstrating that key size and gate type are features that affect SAT attack run time. However, it is not clear yet if such a GCN is truly learning the topological and functional circuit properties that are the source of deobfuscation hardness for arbitrary circuits and distinguish meaningfully between different locked circuits that are not in the immediate training dataset of the model.

This paper focuses on the above question and on the need for metrics that capture such generic deobfuscation hardness. We specifically deliver the following:

- We set up a control experiment. We develop a novel synthetic circuit benchmark set of Substitution-Permutation Networks (SPN), which is the basis for the most efficient yet secure keyed-functions used today: block-ciphers. We then ask whether a GCN fully trained on traditional ISCAS benchmarks can predict the simple fact that a deep SPN is more secure than a wide SPN of the same size, to which we surprisingly discover the answer to be negative.
- Motivated by block-cipher design principles, we develop a set of engineered features extracted from the circuit graph. These include various measures of depth, which are critical in SPNs, along with width. Furthermore, we propose a partitioning scheme that allows mapping the circuit to a possible SPN analog and measures cryptanalysis metrics such as differential statistics and bit propagation width. We show how simple combinations of these metrics can beat the GCN on the SPN hardness prediction task.
- We take these metrics back to the non-SPN benchmark set and show that many positively and consistently correlate with runtime. This means that there is an important utility in extracting deeper engineered features for deobfuscation hardness. If not to supplant end-to-end learning, such metrics can certainly aid it.

The paper is organized as follows. Section 2 presents backgrounds for logic locking, SAT attack, GCNs, and relevant backgrounds. Section 4 presents the analysis of the prediction model and features. Section 5 shows experimental results. Section 6 draws the conclusion for the paper.

2. Background

Logic locking. Formally, a logic locking scheme is an algorithm that transforms an n-input m-output original circuit $c_o(x)$ to a locked circuit $c_e(k, x)$ by adding l new key inputs, in a way that hides/corrupts the functionality of c_o absent some correct key k_*. The locked circuit is fabricated by the untrusted foundry, and the correct key is programmed post-fabrication. Randomly inserting key-controlled XOR/XNOR gates, lookup-tables, AND-trees/comparators, and MUX networks are among common ways to implement locking [5].

SAT attack. An oracle-guided attack such as the SAT attack allows the attacker to query a black-box implementation of c_o. This corresponds to purchasing a functional/unlocked IC from the market and accessing its input/outputs/scan-chain. The attack begins by forming an SAT formula $M = c_e(k_1, x) \neq c_e(k_2, x)$. This is derived by constructing the circuit for M and then using the Tseitin transform to convert it to an SAT Conjunctive-Normal-Form (CNF) formula. Satisfying M returns a Discriminating Input Pattern (DIP) $\hat{x}$, which is queried on the oracle obtaining $\hat{y} = c_o(\hat{x})$. The resulting input-output relation is appended to M. The process continues until the formula is no longer satisfiable at which point solving the IO-relations will return a functionally correct key k_*.

Graph Neural Networks (GNN). Most deep learning algorithms specialize in dealing with fixed-dimensional data. Many types of data, such as social networks, chemical

structures, and circuits, are graph-like in nature. Graph Neural Networks (GNN) are a class of recent machine learning models that can directly operate on graphs and perform various learning tasks. Many of these models operate by taking in an initial set of node features as the node encoding. They then iteratively apply operations on the node features and the graph's adjacency matrix that correspond to some variant of data being passed around and accumulated in the graph nodes along edges. This is all achieved without explicitly storing the dense representation of the graph's adjacency matrix. GNNs have been shown to be effective for both node-level and graph-level regression and classification tasks [4]. Graph Convolutional Networks (GCNs) are a type of GNN layer that is analogous to the convolutional operation in deep neural networks.

Substitution Permutation Networks (SPN). SPNs are the basic structure used in block-ciphers. Figure 1 demonstrates a general structure of an SPN. An SPN is often made up of several rounds, with each round consisting of a key mixing layer, and a group of substitution boxes, known as sboxes, which create the substitution layer and a permutation layer. The key mixing layer is typically a bit-wise XOR with the key vector, which means that to have a non-linear SPN, the sbox is the main source of non-linearity. The sbox is typically a look-up table, mapping its input bits to an equal or smaller number of output bits. The permutation in its simplest form performs a reversible bit shuffle. In a block-cipher, the keys for each round of the SPN are typically different and derived from a master key via key-expansion. It is not precisely known formally what kinds of sbox operations produce the most secure SPNs, but given the right substitution+permutation operation, the security of SPNs against practical attacks increases somewhat exponentially as the number of rounds increases.

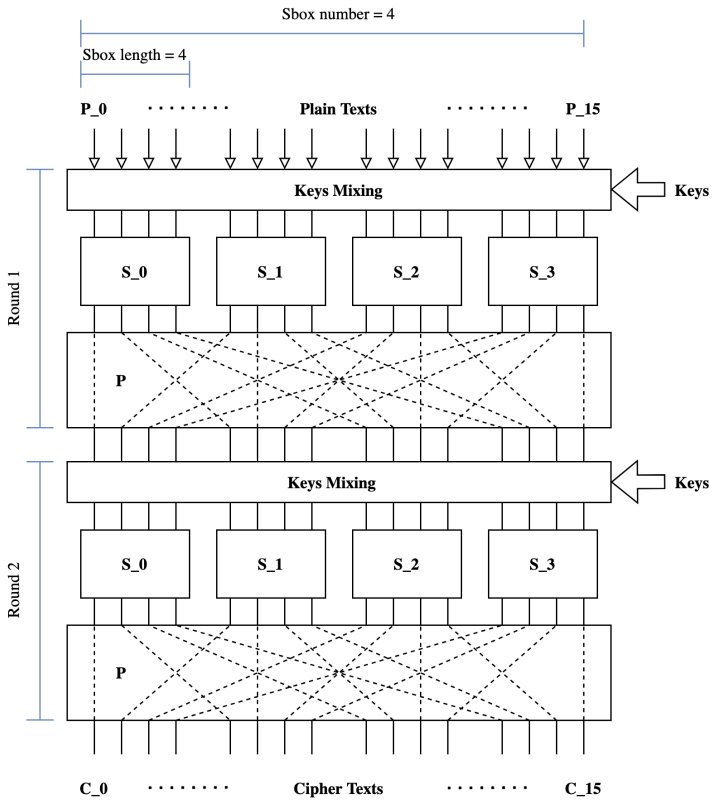

Figure 1. An SPN with two rounds.

3. Putting GCN-Based Deobfuscation Hardness Prediction to the Test

GCN is one of the most effective current deep learning models for detecting graph-like structures. It has multiple message-passing layers and pooling layers, akin to CNN (Convolutional Neural Network). The message-passing layers filter each node in the graph to acquire information from its neighbors with a one-hop distance, while the pooling layer

reduces computational complexity. The following rule is typically used by a GCN message passing layer to update the node encoding in step l, $H^{(l)}$:

$$H^{(l+1)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where $\tilde{A}$ is the adjacency matrix added with self-connections, D is the degree matrix and W is the trainable layer weight. The weight matrix which is the tunable parameter here controls the strength of this combination. This corresponds to combining the information of one node with those collected from the node's neighbors mentioned before. As a result, a GCN model with n layers implies it can aggregate node messages from at most n-hop away.

For a graph-level regression task such as predicting the runtime of the SAT attack from an obfuscated circuit graph, the node-level encodings $H^{(l)}$ have to be combined into a single graph-level number. This aggregation step is called global pooling. The result of the global pooling can then be passed to a fixed neural network layer such as a dense layer with an activation function. For the runtime prediction/regression task this is simply a dense layer with no activation function.

Regressing this GCN on a dataset of SAT attack times with a collection of traditional benchmarks such as ISCAS will show a drop in training loss over iterations. However, here we are interested in whether the GCN in this setting is in fact learning the true source of deobfuscation hardness. To test this, we propose to test a well-trained SAT attack predictor GCN on the task of comparing the difficulty of breaking two circuits— a task that is obvious to a human engineer.

Figure 1 shows a two-round SPN. Figure 2 shows the same two rounds of the SPN in Figure 1 but rather than being placed back-to-back, the rounds have been placed next to one another in parallel. The parallel placement of the rounds increases the number of inputs and the number of keys in the circuit while reducing its depth. It is a well-known fact that the deeper sequential SPN is more secure than the parallel analog. If one takes the 10 rounds of AES and places them in parallel, the key and block size may expand by 10 times, yet security is all but removed. We hence ask a well-trained GCN to predict the deobfuscation difficulty of these two SPNs and compare the results, i.e., whether $Prediction(Sequential\ SPNs) > Prediction(Parallel\ SPNs)$. This will demonstrate whether the GCN is simply blindly counting key inputs or has gained the simple topological heuristic that deeper circuits tend to be more secure than their parallel counterparts. We describe this experiment in Section 5 and found that unfortunately the answer is negative.

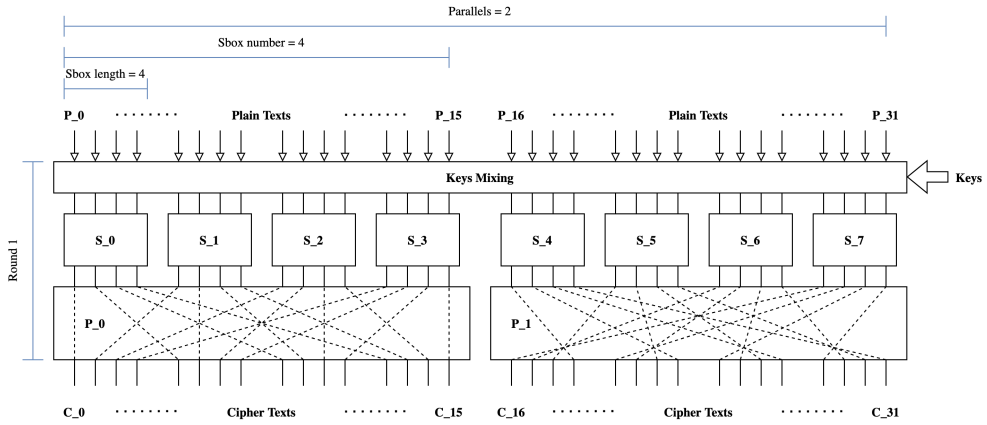

Figure 2. The two rounds SPN in Figure 1 laid down in parallel. This SPN is less secure than the sequential one in Figure 1.

4. SPN-Motivated Features

The inability of the end-to-end GCN to learn the simple facts about the relationship between circuit structure and deobfuscation hardness calls for the development of better metrics/features. We explore some possible metrics here motivated by the SPN structure.

We first begin by exploring some traditional alternatives to the end-to-end machine learning approach. We note that in a theoretical sense, predicting the runtime of an SAT instance is a hard computational problem. If it were not, this would violate certain long-standing assumptions; however, heuristically guessing the difficulty of practical instances of SAT problems has been a topic of research in other domains besides hardware security. The authors of [6] highlight a wide set of features, some simple to compute and some themselves requiring potentially exponential exploration. These range from variable to graph-level features, as well as some novel features, such as closeness to the Horn formula and so on.

As for locked circuit level features, an early work in logic locking proposed analyzing the key–dependency graph of the locked circuit and searching for cliques in this graph [7]. Deriving this metric itself reduces to the NP-complete problem of finding cliques in a graph. Subsequent work has shown that such cliques can be easily achieved by simple tricks. Metrics such as hamming distance (average number of bit-flips at the output given some input difference) were used prior to the invention of the SAT attack [5].

In this paper, given that we showcased the failure of the GCN to compare SPN structures to one another, we derive several metrics based on SPN/block-cipher long-standing design heuristics. If one can map a generic circuit to an SPN equivalent form, then one can use metrics used in the cryptography domain for decades to design better and better SPNs as correlates of deobfuscation hardness.

Assuming that the circuit under study is locked with XOR/XNOR insertion. Here key bits are being XOR/XNORed with internal wires. We can hence imagine that the XOR/XNOR gates are in fact part of a key-mixing layer, with the non-key gates being parts of substitution/permutation operations. While partitioning the non-key logic of the circuit into substitution/permutation blocks does not necessarily produce a unique mapping, the partitioning into key and non-key parts can be performed uniquely.

An example of this partitioning is seen in Figure 3. Here, we take the key-controlled XOR/XNOR gates as key-parts. The remaining logic is then in the form of at least one connected non-key part. Algorithm 1 shows the pseudo-code for this partitioning.

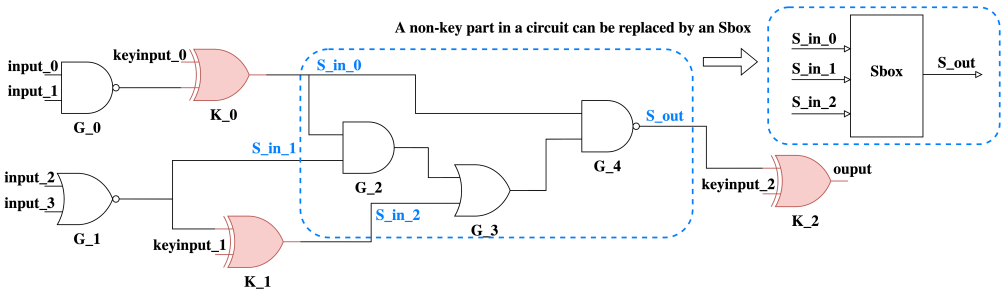

Figure 3. A non-key part in a circuit. Non-key parts (which serve as sboxes when viewing the circuit as an SPN), are parts of the circuit that lie between key-gate logic.

The algorithm starts by from the set of all key gates immediately connected to the key input set *Keys* (line 3). For each key gate in this set, we perform a backward breadth-first-search (BFS) until we reach other key gates or primary inputs. The logic explored as part of this BFS forms a new non-key part (*NkeyPart*). All circuit partitions (*Parts*) will be completed once the aforementioned procedure has been applied to all keys in *Keys* (line 13). We record these different partitions, which allow us to then compute a set of SPN-motivated features listed herein:

- **Non-key Part Structural Statistics**: this includes structural information about the non-key parts. This is primarily, the average number of inputs, outputs, and gates.
- **Number of Active Non-key Parts**: the average number of "active" non-key parts when passing a signal randomly from one of circuits' input to an output. This is derived from the important notion of **active-sboxes** in block-cipher design. Given a change in some input bit, the change will propagate through the SPN touching many

sboxes. The sboxes touched during this propagation are called active sboxes. The difficulty of differential cryptanalysis is directly tied to the number of active sboxes. This metric shows a surprising consistently high correlation with SAT attack time.

- **Non-Key Part Differential Statistics**: the average output flip rate when randomly flipping a bit in non-key parts. This is derived also from differential cryptanalysis, in which changes in the input of some sbox will produce changes at its output. The closer the change at the output is to half of all bits, the more propagation there is in the circuit. Furthermore, changes in the input that produce highly-likely/impossible changes at the output can facilitate differential cryptanalysis.

Algorithm 1: Circuit Partitioning

```
Function Partition(Circuit Cir):
    Parts ← ∅ // Set of all partitions
    KeyGates ← get_key_gates(Cir) // set of gates connected
    // immediately to key wires.
    for kg ∈ KeyGates do
        Q ← queue() // BFS queue
        NkeyPart ← ∅ // Non-Key partition
        Q.push(kg) // start BFS exploration from kg
        while Q ≠ ∅ do
            cg ← Q.pop() // explore the fanin gates within a one-hop distance
                of cg
            for fin ∈ Cir.get_fanin_gates(cg) do
                // those that are not visited nor are key-gates must
                // belong to the non-key part
                if fin ∉ visited and fin ∉ KeyGates then
                    visited ← visited ∪ fin NkeyPart ← NkeyPart ∪ fin Q.push(fin)
        Parts ← Parts ⋃ NkeyPart
    return Parts
```

Note that while we only explore these metrics in this paper, there is a dearth of other SPN-related metrics that can be explored which we aim to study in our future work. Examples are the nonlinearity of each non-key part, the BDD (Binary Decision Diagram) size of each non-key part, and so on.

In [8], it was shown that the original circuit itself can have a huge impact on deobfuscation hardness. One can think of AES itself as a deep and complex original circuit that is locked by simply inserting key-mixing XOR gates and the original circuit (which includes the sboxes and AES permutation) is doing the heavy-lifting for security. The above metrics by focusing on non-key parts capture aspects of the original circuit that relate to security.

5. Experiments

We now present our experimentation. All experiments are carried out on an AMD EPYC dual-CPU 128-core, 256GB server running Ubuntu with the neos C++ SAT attack tool, which uses Glucose as its SAT solver. We use spektral [9] for graph machine learning, and sklearn for traditional ML techniques.

5.1. Training Datasets

We generate a set of datasets that help us determine the transfer-learning ability of the GCN. Hence, we propose and build 4 datasets containing either benchmark circuits locked with the same key size or varying key sizes or datasets comprised of sequential and parallel SPNs. The goal is to train the GCN model on one dataset and evaluate it on another. Successful transfer-learning (generalization) here will be captured by a model being trained on non-SPN circuits being able to differentiate the hardness of parallel versus sequential SPNs.

SPN Generation. We generate a novel training and evaluation dataset consisting of sequential and parallel SPN circuits. The SPN circuits are constructed using a Python script. The SPN sboxes are generated as random truth tables that are synthesized to gates using ABC. The truth-table generation is repeated several times and disconnected or highly skewed SPNs are discarded. Each time an SPN is generated the sboxes are each assigned new random truth tables.

Each SPN is parameterized (and named) according to the following:

- *seq*/*par*: type of SPN structure (sequential/parallel).
- p: number of parallel rounds.
- n: number of sboxes in each round.
- l: length for an sbox. i.e., number of inputs/outputs to each sbox.
- r: SPN round number (number of sequential rounds).

As such, Figure 1 can be labeled as *seq*_n4_l4_r2 while Figure 2 may be named as *par*_p2_n4_l4_r1. This principle is applied to all datasets in our paper that follow.

Our traditional circuit datasets consist of benchmark circuits locked with XOR/XNOR random insertion locking. Deobfuscation times are extracted from the exact SAT attack algorithm. We average the runtime of the SAT attack across a few runs to obtain a more accurate reading. We develop four different datasets in total, a more detailed description is as follows:

- **Dataset_1**. We first select a set of benchmarks from ISCAS-85, ISCAS-89, and ITC-99 benchmarks, respectively, with diverse depth/width statistics to form a benchmark set (c432, c499, c880, c1355, c1980, c2670, c3540, c5315, s832, s953, s1196, s1238, s1488, s1494, s3271, s5378, b04, b07, b11, b12, b13, b14). Since some of these benchmark circuits are sequential, we convert them to combinational by removing flip-flops and replacing them with primary input/output pairs. Then, each circuit in the dataset is locked using XOR/XNOR locking with 64 key bits. The locking is repeated 100 times with different random seeds that results in the XOR/XNOR key-gates being inserted in different locations. These circuits have the same key size as the evaluated SPN datasets that will be introduced later. This dataset contains $22 \times 100 = 2200$ circuits.
- **Dataset_2**. This dataset is comprised of a subset of the above benchmark circuits that are difficult to break for the SAT attack. The dataset is built by adjusting key sizes from 1 to 700 for the c3540, s5378, and 1 to 500 for the b14 benchmark circuits (b14 with more than 500 key bits was not deobfuscatable with our SAT attack). The locking is repeated 10 times for each key size. This leads to a total of $((2 \times 700) + (1 \times 500)) \times 10 = 19{,}000$ locked circuits.
- **Dataset_3**. This dataset is derived by taking each circuit in the benchmark set in Dataset_1 (minus the b14 benchmark due to its size and deobfuscation time) and locking it with XOR/XNOR with the key size ranging from 1 to 200. The locking is repeated 5 times with different key gate locations. This leads to $21 \times 200 \times 5 = 21{,}000$ locked circuits.
- **Dataset_4**. This is a unique and novel dataset that solely comprises SPN circuits. This is built by constructing SPNs with the following parameters. The number of sboxes in each round is for different sbox number ($n \in \{4, 8, 16\}$), sbox length ($l \in \{4, 6, 8\}$), and number of rounds ($r \in \{1, 2, 3, 4, 5\}$). Each SPN is regenerated 20 times. This leads to a total of $20 \times 5 \times 3 \times 3 = 900$ SPN circuits.

5.2. Evaluation Datasets

We generate a specific SPN dataset for evaluation (evaluating transfer learning ability). These are generated correspondingly so that they will produce pairs of sequential versus parallel SPNs having roughly the same number of gates but different topologies (see Table 1). The goal is to see if a model trained on another dataset can detect that deeper SPNs are more secure when the number of gates and keys is the same within this evaluation dataset.

Table 1. Average number of gates in evaluation datasets.

Dataset	Avg #Gates	Dataset	Avg #Gates
seq_n8_l8_r1	7189.60	par_p2_n4_l8_r1	7156.48
seq_n8_l8_r2	14,345.92	par_p2_n8_l8_r1	14,368.32
seq_n8_l8_r3	21,621.12	par_p3_n8_l8_r1	21,582.72
seq_n8_l8_r4	28,844.16	par_p4_n8_l8_r1	28,794.24
seq_n16_l4_r1	512.00	par_p2_n8_l4_r1	520.96
seq_n16_l4_r2	1056.00	par_p2_n16_l4_r1	1039.68
seq_n16_l4_r3	1567.04	par_p3_n16_l4_r1	1538.56
seq_n16_l4_r4	2065.92	par_p4_n16_l4_r1	2089.28

5.3. GCN Evaluation Results

We use the ICNet technique to train the GCN model with three input features: key gate mask (set to 1 for key gate, else 0), gate type, and the circuit's sparse adjacency matrix. The model has three layers and is trained until it converges using the ADAM optimizer with the MSE loss criterion. To test the model's performance under various conditions, we train it on four training datasets established in Section 5.1 and make predictions on the SPN comparison evaluation datasets developed in Section 5.2.

The average SAT attack run time for this dataset is represented by the real value in Table 2 (the real value of 1800 in the *seq_n*8_*l*8_*r*3 and *seq_n*8_*l*8_*r*3 circuits simply denotes an SAT attack time-out). We can conclude the following from real values: (1) Whether the SPN structure is sequential or parallel, the SAT attack execution time grows non-linearly with the rise in-depth; (2) SPNs with longer sbox lengths but the same depth take longer to attack than smaller ones, regardless of whether they are sequential or parallel, e.g., $time(seq\_n8\_l8\_r2) > time(seq\_n16\_l4\_r2)$, it is well established in cryptography that larger sboxes are better; (3) The sequential SPN should be much harder to attack than its corresponding parallel SPN, e.g., $time(seq\_n16\_l4\_r4) > time(par\_p4\_n16\_l4\_r1)$. These findings are consistent with block-cipher design principles. As a result, if GCN can truly estimate a circuit's run time or security, the outcomes of its prediction should at the very least fulfill the aforementioned observations.

However, Table 2 shows that even if we train the GCN on our prepared four types of datasets, including normal and time-out situations the GCN fails at this comparison task. The evaluation results indicate that the trained GCN can only estimate the SAT attack time at a linear rate as the circuit size grows. Even when training on *Dataset*_4, which is developed with only SPNs using random parameters, predictions on evaluation datasets still do not perform well. The GCN expects the parallel SPNs in some cases to be even more secure than their sequential counterpart.

Table 2. GCN evaluation results.

	GCN Prediction Values (in s)			
Training Dataset	seq_n8_l8_r1	seq_n8_l8_r2	seq_n8_l8_r3	seq_n8_l8_r4
Real value	1.78	30.84	1800	1800
Dataset_1	32.27	41.58	42.47	43.76
Dataset_2	711.07	724.06	685.49	662.13
Dataset_3	108.42	113.92	113.92	108.97
Dataset_4	291.16	673.13	807.61	1727.29
Training Dataset	seq_n16_l4_r1	seq_n16_l4_r2	seq_n16_l4_r3	seq_n16_l4_r4
Real value	0.13	0.51	5.51	1110.11
Dataset_1	−10.98	−7.54	−5.88	−5.33
Dataset_2	1309.36	1547.26	1641.45	1662.68
Dataset_3	406.98	336.88	286.55	275.77
Dataset_4	−13.64	82.06	139.67	352.17

Table 2. *Cont.*

	GCN Prediction Values (in s)			
Training Dataset	par_p2_n4_l8_r1	par_p2_n8_l8_r1	par_p3_n8_l8_r1	par_p4_n8_l8_r1
Real value	1.42	3.75	6.03	8.87
Dataset_1	39.52	39.06	42.20	43.09
Dataset_2	696.53	677.57	702.76	668.33
Dataset_3	110.38	115.10	110.59	112.51
Dataset_4	712.06	715.10	844.00	919.86
Training Dataset	par_p2_n8_l4_r1	par_p2_n16_l4_r1	par_p3_n16_l4_r1	par_p4_n16_l4_r1
Real value	0.03	0.11	0.19	0.29
Dataset_1	−11.18	−11.21	−10.83	−10.16
Dataset_2	1465.58	1496.57	1544.90	1584.23
Dataset_3	431.60	436.15	431.13	430.95
Dataset_4	69.94	74.06	90.76	107.14

5.4. SPN-Motivated Metrics

We then evaluated our SPN-motivated metrics from Section 4 on the same dataset. We ensure that our model was trained on the same training datasets as GCN and tested on the same evaluation datasets. First, we leverage our training datasets to assess the correlation between our SPN-motivated features and SAT attack time to see whether our metrics are effective. As seen in Table 3, the average signal walk depth and max active non-key portions, two proposed metrics directly derived from SPN structures have high correlation coefficients and are both positively proportional to all training datasets, indicating that these two metrics in addition to their theoretical relation to long-standing block-cipher design principles are consistent predictors of deobfuscation hardness for locked circuits. In the last column of Table 3, we put the average of correlations to demonstrate the metrics performance among four datasets. We picked two metrics that had interestingly high correlations with the SAT attack time and plot their scatter map in Figure 4. Figure 5 shows scatter plots and correlation lines for high-correlating metrics in the SPN dataset (dataset_4).

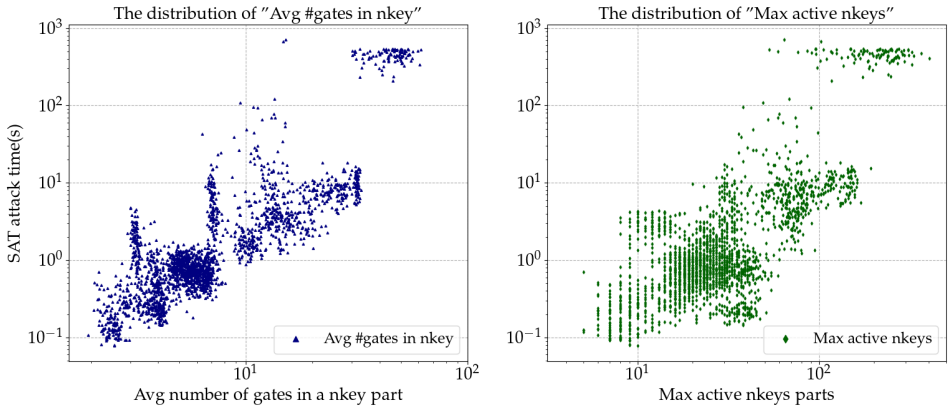

Figure 4. Distributions of metrics with high correlations on dataset_1.

Then, we pass these SPN-like metrics as feature vectors to various ML models. Since these proposed metrics form fixed-size vectors, we choose to fit them using several popular regression models, which include Linear Regression (LR), LASSO, Support Vector Machine (SVM), Multiperception (MLP), Ridge Regression (RR), Elastic Net(EN), Theil–Sen Regression (TS), and Random Forest (RF). We create input vectors from four training datasets, train them on the aforesaid regression models, and then use these trained models to predict evaluation datasets. Table 4 shows the most accurate prediction data for each regression model.

Table 3. SPN-motivated metric correlations with SAT attack time on the 4 different datasets. **Learnability** is the ability of an sklearn MLP regressor to learn the output of the locked circuit, as measured by the average of the probability of correctly predicting output bits from the input vector. "#Keys with $\leq X$ keys in fanin" is the number of key gates in each circuit that have fewer than X other key gates within their transitive fanin. The signal probability biases are captured as the distance of the non-key part output signal probability to 0.5.

Metrics	D1	D2	D3	D4	Average
#Gates	0.89	0.63	0.04	0.58	0.54
Key Depth	0.08	−0.22	0.25	0.70	0.20
#Keys with <2 keys in fanin	0.22	0.29	0.11	−0.35	0.07
#Keys with <4 keys in fanin	−0.24	−0.14	0.20	−0.19	−0.09
#Keys with <8 keys in fanin	−0.14	0.06	0.15	−0.26	−0.05
#Keys with <16 keys in fanin	−0.10	0.33	0.05	−0.06	0.06
#Keys with <32 keys in fanin	−0.07	0.34	0.17	−0.03	0.10
#Keys with <64 keys in fanin	−0.02	0.48	0.07	0.57	0.28
#Keys with <128 keys in fanin	-	0.53	0.03	0.45	0.34
#Keys with <256 keys in fanin	-	0.35	-	0.27	0.31
Avg signal walk depth	−0.06	−0.11	0.19	0.81	0.21
Avg #inputs in nkeys	0.77	0.10	−0.04	0.61	0.36
Avg #gates in nkeys	0.74	0.02	−0.03	0.60	0.33
Avg #outputs in nkeys	-	-	-	0.43	0.43
Min active nkeys	-	-	-	0.39	0.39
Max active nkeys	0.74	0.65	0.19	0.59	0.54
Avg active nkeys	0.07	0.26	0.11	0.56	0.25
Avg #nkeys	-	0.31	0.19	0.74	0.41
Min active nkeys(in percentage)	-	-	-	0.31	0.31
Max active nkeys(in percentage)	0.74	0.09	−0.02	0.62	0.36
Avg active nkeys(in percentage)	0.07	−0.07	−0.02	0.59	0.14
Avg nkeys dout/din	−0.12	−0.13	−0.02	0.01	−0.07
Min nkey signal probability bias	−0.01	−0.02	−0.03	−0.24	−0.08
Max nkey signal probability bias	0.12	0.10	0.06	0.42	0.18
Avg nkey signal probability bias	−0.05	0.35	−0.13	0.36	0.13
Hamming distance	−0.24	−0.29	−0.05	0.65	0.02
Learnability	0.39	0.28	−0.05	−0.05	0.14
Circuit key diameter	0.56	−0.09	−0.07	0.60	0.25
Signal probability	−0.14	−0.51	−0.11	−0.21	−0.24
Circuit maximum depth	0.08	−0.41	0.24	0.72	0.16
Circuit maximum width	0.93	0.60	−0.08	0.40	0.46

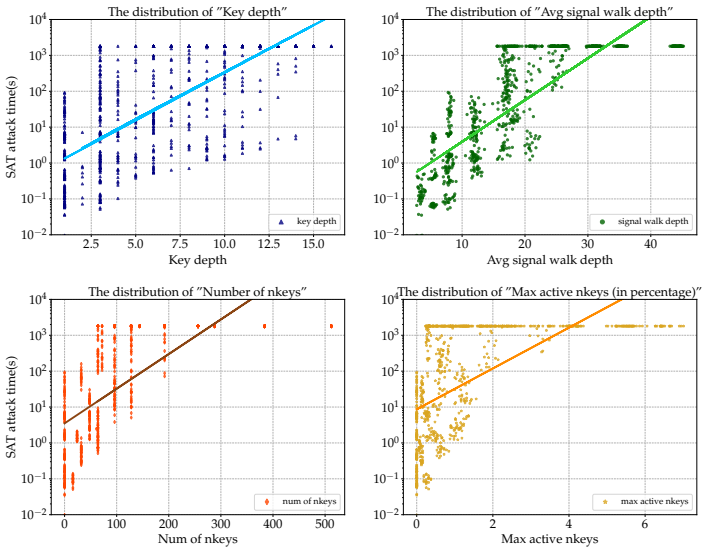

Figure 5. Distributions of metrics with high correlations on dataset_4.

Table 4. Regression models on proposed metrics prediction results.

	Regression Models Prediction Values (in s)			
Regression Models	seq_n8_l8_r1	seq_n8_l8_r2	seq_n8_l8_r3	seq_n8_l8_r4
Real value	1.78	30.84	1800	1800
LR	61.56	473.14	1654.00	1677.23
LASSO	70.71	489.33	1654.54	1678.10
SVM	91.53	193.47	320.25	460.60
MLP	7.05	607.30	1613.23	1708.41
RR	71.31	489.38	1654.50	1667.04
EN	49.64	530.97	1686.88	1696.33
TS	37.69	256.79	1835.84	1834.37
RF	10.18	223.65	1799.11	1800.00
Regression Models	seq_n16_l4_r1	seq_n16_l4_r2	seq_n16_l4_r3	seq_n16_l4_r4
Real value	0.13	0.51	5.51	1110.11
LR	−35.94	79.59	416.89	925.24
LASSO	−34.47	72.17	396.85	927.61
SVM	12.18	12.78	13.46	14.17
MLP	−24.75	−49.10	288.59	1014.93
RR	−29.94	65.36	361.61	921.57
EN	−85.26	22.98	427.86	980.49
TS	−28.71	259.12	236.89	929.86
RF	0.13	1.03	49.04	1055.95
Regression Models	par_p2_n4_l8_r1	par_p2_n8_l8_r1	par_p3_n8_l8_r1	par_p4_n8_l8_r1
Real value	1.42	3.75	6.03	8.87
LR	−281.38	−270.51	−362.75	−395.15
LASSO	−300.70	−309.73	−439.31	−504.49
SVM	90.98	329.35	733.55	1295.49
MLP	−2.14	−13.89	−42.66	−61.09
RR	−316.35	−331.93	−474.88	−550.25
EN	−271.31	−190.10	−214.69	−184.25
TS	−173.28	−164.54	−228.56	−250.09
RF	9.1463	59.40	284.55	683.83
Regression Models	par_p2_n8_l4_r1	par_p2_n16_l4_r1	par_p3_n16_l4_r1	par_p4_n16_l4_r1
Real value	0.03	0.11	0.19	0.29
LR	−197.69	−209.09	−276.35	−318.07
LASSO	−206.98	−229.49	−315.49	−376.14
SVM	12.19	13.46	15.50	18.59
MLP	−27.73	−74.42	−123.07	−169.48
RR	−213.29	−228.19	−310.11	−368.13
EN	−231.59	−16.78	150.09	332.30
TS	−127.84	−105.34	−116.13	−111.35
RF	0.14	7.38	7.65	8.49

We can deduce from these findings that most regression models outperform the GCN. All regression models have non-linearity that matches the real data and most models can discover that sequential SPNs are more secure than their parallel counterparts. Figure 6 shows the comparison results of sequential/parallel SPNs with sbox number of four and eight accordingly. The log values of SAT attack time are shown on Y axis, while the SPN rounds/parallels are shown on the X axis. RF regression estimates all clearly reflect a similar trend as the real non-linear data while the GCN prediction does not and overestimates the parallel SPN attack time.

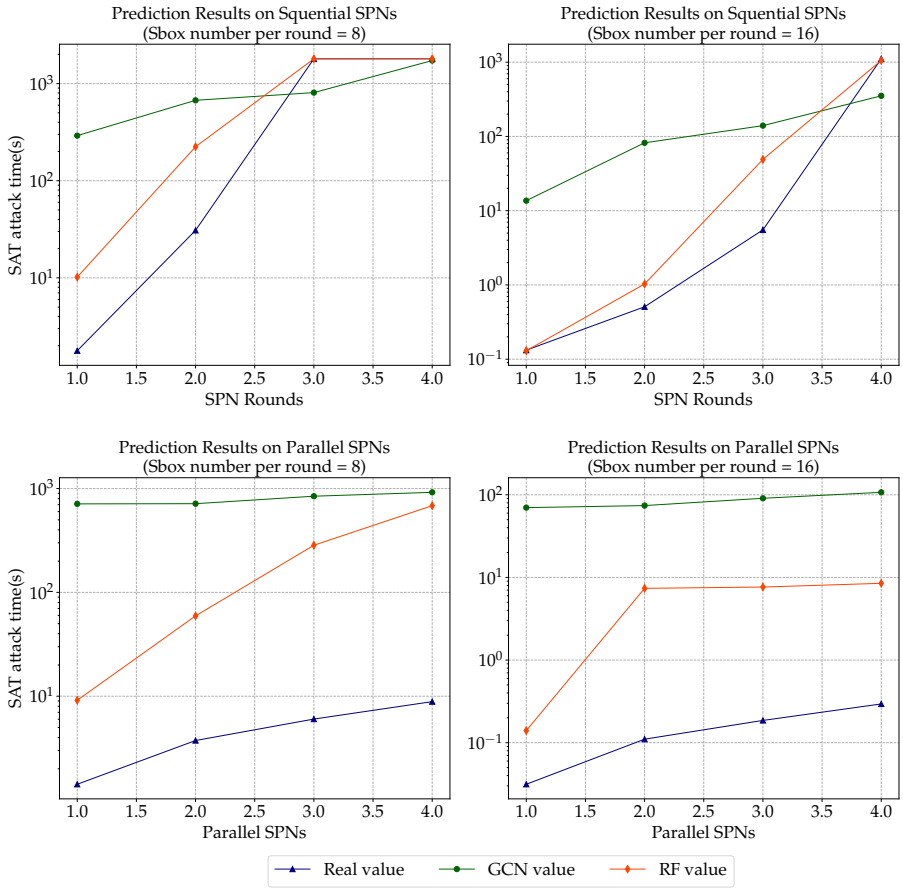

Figure 6. Evaluation results comparison. The GCN consistently overestimates the deobfuscation difficulty of parallel SPNs.

6. Conclusions

In this paper, we demonstrated that GCN-based SAT attack runtime prediction does not seem to be able to capture simple facts about deobfuscation hardness, such as the fact that a deeper SPN is more secure than a parallel one. We proposed a set of alternative metrics derived from SPN design principles and demonstrated their superior ability in capturing circuit complexity. Most of the regression models have shown non-linearity that matches the real data and the majority of the models tested can discover that sequential SPNs are more secure than their parallel counterparts. Among these models, random forest regression produces the most accurate results (as measured in distance from predicted to real values). We leave a deeper analysis of these metrics and the application of the above approach to non-XOR/XNOR locking schemes to future work.

Author Contributions: Conceptualization: G.Z. and K.S.; Software: G.Z. and K.S.; Writing—Original: G.Z.; Writing—Review: K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research receive no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Subramanyan, P.; Ray, S.; Malik, S. Evaluating the security of logic encryption algorithms. In Proceedings of the 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 5–7 May 2015; IEEE: Manhattan, NY, USA, 2015; pp. 137–143.
2. El Massad, M.; Garg, S.; Tripunitara, M.V. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 8–11 February 2015.
3. Chen, Z.; Kolhe, G.; Rafatirad, S.; Lu, C.T.; Manoj, S.; Homayoun, H.; Zhao, L. Estimating the circuit De-obfuscation runtime based on graph deep learning. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; IEEE: Manhattan, NY, USA, 2020; pp. 358–363.
4. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.
5. Yasin, M.; Sinanoglu, O. Evolution of logic locking. In Proceedings of the 2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Dhabi, United Arab Emirates, 23–25 October 2017; IEEE: Manhattan, NY, USA, 2017; pp. 1–6.
6. Hutter, F.; Xu, L.; Hoos, H.H.; Leyton-Brown, K. Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.* **2014**, *206*, 79–111.
7. Rajendran, J.; Pino, Y.; Sinanoglu, O.; Karri, R. Security analysis of logic obfuscation. In Proceedings of the 49th Annual Design Automation Conference, San Francisco, CA, USA, 3–7 June 2012; pp. 83–89.
8. Shamsi, K.; Pan, D.Z.; Jin, Y. On the Impossibility of Approximation-Resilient Circuit Locking. In Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 5–10 May 2019; IEEE: Manhattan, NY, USA, 2019; pp. 161–170.
9. Grattarola, D.; Alippi, C. Graph Neural Networks in TensorFlow and Keras with Spektral. *arXiv* **2020**, arXiv:2006.12138.