Sweep to the Secret: A Constant Propagation Attack on Logic Locking

Abdulrahman Alaql, Domenic Forte, and Swarup Bhunia
Dept. of ECE, University of Florida, Gainesville, FL 32608
$alaql89@ufl.edu$, $\{forte, bhunia\}@ece.ufl.edu$

***Abstract*—The development of hardware intellectual properties (IPs) has faced many challenges due to malicious modifications and piracy. One potential solution to protect IPs against these attacks is to perform a key-based logic locking process that disables the functionality and corrupts the output of the IP when the incorrect key value is applied. However, many attacks on logic locking have been introduced to break the locking mechanism and obtain the key. In this paper, we present SWEEP, a constant propagation attack that exploits the change in characteristics of the IP when a single key-bit value is hard-coded. The attack process starts with analyzing design features that are generated from the synthesis tool and establishes a correlation between these features and the correct key values. In order to perform the attack, the logic locking tool needs to be available. The level of accuracy of the extracted key mainly depends on the type of logic locking approach used to obfuscate the IP. Our attack was applied to ISCAS85, and MCNC benchmarks obfuscated using various logic locking techniques and has obtained an average accuracy of 92.09%.**

I. Introduction

The integration of hardware intellectual property (IP) has vastly grown in the past several years, which is being utilized in different critical applications such as military, health-care, and the Internet of things (IoT). Due to the globalization of the semiconductor industry, many stages of the development cycle are outsourced to third-party facilities, which may not always be trusted. Hence, a rising concern regarding the security of hardware IPs has emerged as estimates suggest that $250 billion each year is lost because of IP piracy and counterfeit components [1]. The main objective for adversaries is to gain design information in order to reverse engineer the IP and to create counterfeits. The other goal is to inject these systems with Trojans that may reduce the performance, deny service, or leak sensitive data. To prevent these attacks, many techniques have been considered, where different protection mechanisms are used to secure the system in specific development cycles, while others focus on preventing particular attacks.

One potential solution is to apply hardware logic locking, which can be used to add a layer of protection against many attacks on hardware IPs [2]. Hardware logic locking is a pre-silicon measure that aims to control the functionality of the system-under-attack and corrupt the outputs by adding logic elements and/or dummy routing. The locking mechanism is usually controlled by a key that is only known by the IP owner and can be used to unlock the design and regain the correct functionality. Logic locking is typically applied to the gate-level netlist, which makes it suitable for a secure soft-IP exchange since the IP is not going to function correctly unless the key is obtained. On the other hand, several attacks on logic locking have also emerged. Many of these attacks are able to completely or partially unlock the functionality, to bypass the locking mechanism, or to extract the key using functional and structural analysis [3][4][5][6][7].

Although the research for improved logic locking techniques is critical, such techniques cannot be considered ‘strong’ unless no clear vulnerability is found. Hence there is an urgent need for diverse and comprehensive attack/security assessment tools. In this paper, we introduce SWEEP, a constant propagation attack on hardware logic locking that exploits the changes that occur to the system during the synthesis process. The attack performs the synthesis analysis to each key input by optimizing the locked system when hard-coding both correct and incorrect values for that key bit. The attack is scalable and does not require an unlocked circuit (oracle), but having an unlocked system is recommended to perform functional tests and to verify the correctness of the extracted key. The proposed approach performs the analysis on information provided by the synthesis process in a generated report. The attack is designed to be precise, meaning the correctly extracted key is identified by value and location with high confidence, which reduces the brute force effort to identify the key. It is difficult to prevent such an attack due to the naivety and predictable patterns of current logic locking techniques. Unlike many existing attacks in the literature, SWEEP is designed to exploit the structural effects of the locked circuit rather than performing functional analysis, which exposes more vulnerabilities to techniques that have always been considered strong. This paper makes the following main contributions:

- Introduces SWEEP, a constant propagation attack that is applicable to any IP type, size, and complexity.
- Provides a methodology for applying the attack by analyzing the obfuscation tool and performing a weight scoring process for each design feature.
- Demonstrates the attack on several popular types of logic locking techniques and shows the accuracy results for ISCAS85 and MCNC benchmarks.

The rest of the paper is organized as follows. Section II provides a background of logic locking and relative techniques and attacks. Section III talks about an overview of the synthesis-based attack, and Section IV presents the attack results. Section V discusses the significance of the attack as well as its limitations, and Section VI provides the conclusion.

II. Background

A. Threat Model

The goal for adversaries is to obtain critical design information about the IP to reverse engineer or to maliciously modify it. In this paper, our proposed attack is applied with the following assumptions:

978-1-7281-3544-1/19/$31.00 ©2019 IEEE

- The attacker has access to the locked gate-level netlist, which can be obtained using invasive reverse engineering techniques or from an untrusted foundry.
- The attacker has access to the obfuscation tool that is used to perform the logic locking mechanism to the obtained locked netlist.
- The attacker can identify all key inputs in the netlist.

Similar to encryption algorithms and tools, obfuscation tools should be publicly available and easily accessible to the end-user in order to consider them practical. Hence, the assumption that attackers are able to obtain the obfuscation tool is reasonable. Although the attacker needs to obtain the full locked netlist, familiarity with the function is not required to perform the attack. Additionally, the attack does not necessarily require an unlocked system or a set of input/output pairs (oracles). However, obtaining an unlocked system can help perform additional attacks, such as brute-force or functional hill-climbing [12].

B. Attacks on Logic Locking

1) ***Boolean Satisfiability (SAT) Attack****:* SAT attack is applied to locked combinational circuits using a key-based hardware obfuscation [4]. The attack algorithm applies a set of inputs and key values to find a distinguishing input pattern (DIP), which is defined as an input that changes the output value when two or more different keys are used. A SAT solver is then applied to eliminate key values associated with that DIP, and this process iterates until the entire system functionality is covered. The attack is only applicable to combinational circuits or sequential circuits that have access to scan-chains. Additionally, the SAT attack is not able to unlock systems that contain SAT-hard functions, such as multipliers.

2) ***AppSAT Attack****:* AppSAT is a version of the SAT attack that is developed to provide an approximated key instead of searching for the correct one [5]. The key finding algorithm may accept a key that provides correct functionality for most tested input patterns. AppSAT can break obfuscation techniques that focus on increasing the number of DIPs while reducing the amount of output corruptibility.

3) ***KSA Attack****:* Key sensitization attack (KSA) requires the locked netlist and an unlocked IC. By identifying key gates that can block the effect of other key gates (muting gates), the attacker may be able to find a pattern that sensitizes a key input to the output [3]. The attack can then focus on sensitizing the key values of those gates from an unlocked IC. The attack is not scalable to extremely large and complex systems, especially if the number of primary outputs is limited.

4) ***Desynthesis Attack****:* The desynthesis attack has been introduced in [7] to exploit the changes caused by guessing a key vector and applying the synthesis process. The guessed key is then partially modified, and the analysis of the functional and structural behavior is applied again. The attack does not require an unlocked circuit, but the obfuscation tool is needed to perform the analysis. Unlike our proposed attack, the desynthesis attack focuses on the functional and structural changes that occur when applying guessed vectors to the key inputs, while our attack focuses on the features obtained from the synthesis report. The other main difference is that the desynthesis attack performs the analysis to the entire key vector, while our attack performs the analysis to individual key inputs, which gives high confidence to the extracted keys. Compared to the desynthesis attack, our attack is also designed to avoid wild guesses and places an ‘X’ symbol to keys that are not being guessed, which can be used to perform further attacks, such as reduced brute force.

5) ***SAIL Attack****:* This machine-learning attack uses the obfuscation tool and some training sets to extract design properties that can lead to the extraction of key values [6]. SAIL is independent of the system type or size, and it does not require an unlocked system. However, the attack is limited to systems that are locked using XOR/XNOR techniques only, as it is not able to handle MUX-based obfuscation techniques.

C. Comparison to Existing Attacks

Unlike the SAT attack, our proposed method is able to extract the key from complex and SAT-hard functions that the SAT attack may not be able to handle. The proposed attack does not require an unlocked circuit, which widens the scope of attack to designs in development stages as early as the foundry, whereas most attacks are not viable unless the unlocked function is deployed and available to attackers. Additionally, the scalability of our approach makes it more practical to apply compared to the KSA attack. Although the proposed attack does not handle XOR/XNOR gates, it can be complemented by SAIL (which focuses on XOR/XNOR based obfuscation). A brief comparison between existing attacks and our proposed one is shown in Table I.

TABLE I: A Summary of Attacks on Obfuscated Gate-level Netlists

Attack	Obfuscation Tool	Oracles Required	Differences
KSA [3]	Not Required	Yes	Not Scalable
SAT* [4]	Not Required	Yes	Cannot Handle Complex Functions*
AppSAT* [5]	Not Required	Yes	Cannot Handle Complex Functions*
Desynthesis [7]	Required	Recommended	Not Scalable
SAIL [6]	Required	Recommended	Only Attacks XOR/XNOR
SWEEP (Proposed)	Required	Recommended	Does not Attack XOR/XNOR

* Such as multipliers and large arithmetic operations.

III. Attack Overview

SWEEP can be considered an attack on the obfuscation technique used for logic locking, where a set of obfuscated benchmarks with known key values are used to perform the analysis (training). The basic idea of the attack is to assign a key-value (logic 0 or logic 1) to one key input and to synthesize the obfuscated design with that key embedded in the circuit as a constant value. The synthesis report is then analyzed for any design features that are correlated to the correct key value. The attack algorithm can capture a number of features that are directly or indirectly correlated to the correct key value. These features are then analyzed for each key-value, which can indicate the correct value of the analyzed key input. The analysis of these features is done for each key input individually by comparing them to the synthesis report of the original obfuscated design. A feature scoring algorithm is used to evaluate each design feature and assigns a signed weighting value, which helps enhance the accuracy of SWEEP. Fig. 1 shows an overview of the attack process.

A. Generation of Training Data Points

The first step of the introduced attack is to generate a set of data points and sample circuits. Different popular benchmark suites can be used to generate the required data

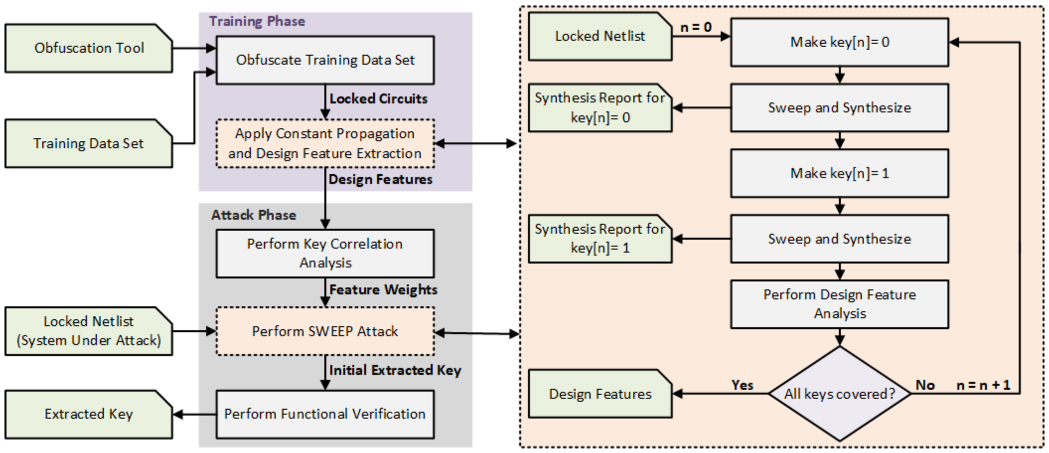

Fig. 1: Process flow of the synthesis-based attack.

points. Additionally, the obfuscated system-under-attack can be used as training data points, where an additional round of obfuscation is applied to the locked design and used to perform the analysis. After obtaining the locked netlists, the synthesis report will then show a number of design features that represent the performance, timing-limitations, and resource allocation of the design. These features include (but are not limited to) the amount of power consumption, latency of the critical paths, the total area, gate counts, or the number of binary decision diagram (BDD) branches.

B. Constant Propagation and Feature Extraction

In this stage, the generated training data sets are modified by hard-coding each key input with the correct and the incorrect key value. In each instance, the attacker performs a round of synthesis and extracts design features from the generated synthesis report. Different synthesis and optimization parameters and library modifications can be used to enhance the sensitivity of the extracted features and make them easily observable. We have also observed that limiting the synthesis tool to only use an And-Inverter Graph (AIG) representation of the attacked IP has been observed to provide refined values for each feature. AIGs are semi-canonical as they remove any dependence (or decorrelation) caused by technology mapping algorithms. The synthesis reports are generated, where each key input has two associated synthesis reports (a report for logic 0 and another for logic 1). For each feature, two reported values are extracted (called $F_{zero}[n]$ and $F_{one}[n]$) with corresponding key location n. Eqn. 1 and 2 outline how the feature values are calculated when the key value is zero and one, respectively, and Eqn. 3 is the difference (delta) between the two values:

$$V_{zero}[n] = \frac{F_{REF} - F_{zero}[n]}{F_{REF}} \tag{1}$$

$$V_{one}[n] = \frac{F_{REF} - F_{one}[n]}{F_{REF}} \tag{2}$$

$$\Delta V[n] = V_{zero}[n] - V_{one}[n] \tag{3}$$

Where F_{REF} is the feature value for the reference obfuscated netlist, $F_{zero}[n]$ and $F_{one}[n]$ are the feature values when $KEY[n] = 0$ and $KEY[n] = 1$ respectively $\Delta V[n]$ is the difference between the two values. When considering multiple features, a "feature matrix" that contains the deltas for each key location (column) and each feature type (row) is generated, Eqn. 4 shows how the feature matrix is expressed:

$$\Delta V_{kn} = \begin{bmatrix} \Delta V_{11} & \Delta V_{12} & \Delta V_{13} & \dots & \Delta V_{1n} \\ \Delta V_{21} & \Delta V_{22} & \Delta V_{23} & \dots & \Delta V_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Delta V_{k1} & \Delta V_{k2} & \Delta V_{k3} & \dots & \Delta V_{kn} \end{bmatrix} \tag{4}$$

where k is the number of features and n is the number of key-bits. The feature matrix is a concatenation of data sets of all used benchmarks generated by the obfuscation tool.

C. Feature Weighting Algorithm

This algorithm takes the generated feature matrix and its corresponding key values and generates the optimum weights that evaluate the correlation between every feature and the correct key. The basic function of the weighting process is shown in Algorithm 1.

For each feature, the process in Algorithm 1 compares the sign of delta values (ΔV) with its corresponding key value. In other words, if the delta sign for most feature values is positive when $KEY = 1$ and negative when $KEY = 0$, then the overall weight W for that feature is positive. Similarly, if the sign for most delta values is negative when $KEY = 1$ and positive when $KEY = 0$, then the overall weight is going to be negative. The value of the weight depends on how many delta value signs are consistent with either a positive or a negative weighting score. Features that almost have an equal amount of positive and negative scores indicate that they are not correlated with the correct key value, and will be assigned with a low weighting value.

D. Perform the SWEEP Attack

In this stage, the obfuscated netlist of the system-under-attack is used to extract its design features. By applying the same constant propagation technique used for the training data set, a feature matrix for the locked design is generated. Key locations used in this stage correspond to the locked netlist

Algorithm 1 Key-Feature Correlation

```
1:  procedure PRODUCING FEATURE WEIGHTS
2:  Input: $\Delta V_{kn}$: Feature matrix
3:  Input: $KEY[n]$: Correct key vector
4:  for $j = 1$ to $k$
5:  $positive\_score = 0$, $negative\_score = 0$
6:  for $i = 1$ to $n$
7:      if $\Delta V_{ji} > 0$ and $KEY[i] = 1$ then
8:  $positive\_score = positive\_score + 1$
9:      else if $\Delta V_{ji} < 0$ and $KEY[i] = 0$ then
    $positive\_score = positive\_score + 1$
10:     else if $\Delta V_{ji} > 0$ and $KEY[i] = 0$ then
    $negative\_score = negative\_score + 1$
11:     else if $\Delta V_{ji} < 0$ and $KEY[i] = 1$ then
    $negative\_score = negative\_score + 1$
12:     end if
13: end for
14: $W[j] = [positive\_score - negative\_score]/n$
15: end for
16: Output: $W[k]$: Feature weights
```

under attack. An overall sum of all features for each key location is calculated using Eqn. 5:

$$\Delta V'[n] = \sum_{t=1}^{k} V_{tn} \times W[t] \tag{5}$$

where $\Delta V'[n]$ is the weighted feature delta for key location n, k is the number of extracted features. This vector is used to perform the SWEEP attack. The key extraction process is explained in Algorithm 2:

Algorithm 2 Weighted Feature Values Comparison

```
1:  procedure COMPARING FEATURE VALUES
2:  Input: $\Delta V'[n]$: Weighted feature values
3:  Input: $m$: Acceptable margin
4:  for $i = 1$ to $n$
5:      if $|\Delta V'[n]| < m$ then
          $Extracted\_key[i] \leftarrow X$
6:      else if $\Delta V'[n] > 0$ then
          $Extracted\_key[i] \leftarrow 1$
7:      else if $\Delta V'[n] < 0$ then
          $Extracted\_Key[i] \leftarrow 0$
8:      end if
9:  end for
10: Output: $Extracted\_key$: Extracted key
```

The inputs to Algorithm 2 are the weighted delta values $\Delta V'[n]$, and the acceptable margin m. This margin is an adjustable parameter that prevents key guesses for values that are close to each other, which will reduce the chance of performing a wild guess. Increasing this margin improves how precise key extraction is at the cost of fewer keys being successfully extracted out of the total key vector. The symbol 'X' indicates that the attack was not able to make a safe decision for that key input due to the absolute value of $\Delta V'[n]$ being less than the specified margin m. The output of the algorithm is the extracted key vector.

E. High-Order Locking

Up until now, we have assumed that the obfuscation technique uses a two-choice locking mechanism, such as a MUX2, which reroutes the signal to either fan-in A or fan-in B. However, many locking techniques utilize a higher-order of route selection, such as MUX4, MUX8, etc. In that case, SWEEP is designed to structurally analyze the circuit and to obtain "key clusters", which are analyzed at the same time. For example, if the locking mechanism uses MUX8, that means three key inputs control each key-gate, and the attack will be performed on a set of 3-bit key clusters. For every key cluster, we generate all possible key values and extract a vector of features for that cluster. Instead of using $V_{zero}[n]$ and $V_{one}[n]$ from Eqn. 1 and 2, we use the minimum and maximum values of $V_i[g]$ for the obtained vector of features, where g is the total number of key clusters in the design, and i corresponds to all possible key values in one cluster (a cluster of n-bit keys has i of 0 to $2^n - 1$).

The weighting process in Algorithm 1 will be modified to evaluate $minimum\_score$ and $maximum\_score$ instead of $positive\_score$ and $negative\_score$. In other words, instead of comparing the delta of features generated by applying the two possible values for a single key-input, we apply all possible values (0 to i) and capture the set of design features for each key entry. We then repeat this process for each cluster. For example, a 3-bit key cluster has 8 possible values with each possible key value produces its own set of features $V_i[g]$. Out of the 8 possible values, we focus on the two critical feature values, the minimum $min\{V_i[g]\}$ and the maximum $max\{V_i[g]\}$. We also examined whether their corresponding key values match the ones of the original key. To identify the key value associated with $min\{V_i[g]\}$ and $max\{V_i[g]\}$, we use the argument of the minima and maxima ($argmin$ and $argmax$), which locates the position of minimum and maximum values of a function. Hence, $KEY_{min}[g] = argmin_i\{V_i[g]\}$ and $KEY_{max}[g] = argmax_i\{V_i[g]\}$. To obtain the weights, $minimum\_score$ is incremented if the correct key cluster matches KEY_{min},

Algorithm 3 High-Order Feature Values Comparison

```
1:  procedure COMPARING KEY CLUSTER VALUES
2:  Input: $V_i[g]$: Feature matrix
3:  Input: $W[k]$: Feature weights
4:  Input: $m$: Acceptable margin
5:  for $j = 1$ to $k$
6:  for $h = 1$ to $g$
7:      if $W[j] = 0$ then
          $Extracted\_key[j, h] \leftarrow X$
8:      else if $max\{V_i[h]\} - min\{V_i[h]\} < m$ then
          $Extracted\_key[j, h] \leftarrow X$
9:      else if $W[j] > 0$ then
          $Extracted\_key[j, h] \leftarrow argmax_i\{V_i[g]\}$
10:     else if $W[j] < 0$ then
          $Extracted\_key[j, h] \leftarrow argmin_i\{V_i[g]\}$
11:     end if
12: end for
13: end for
14: Output: $Extracted\_key[k]$: Extracted key vector
```

and $maximum\_score$ is incremented if KEY_{max} matches the correct key. $W[k]$ are the weights obtained in the training phase, where k is the number of features. For the feature comparison process, Algorithm 3 is used:

The output of Algorithm 3 is a set of extracted key values, where each feature k has its own $Extracted\_key[k]$. The overall extracted key can be obtained by performing a weighted majority voting on all extracted key vectors.

F. Perform Functional Verification

After the attack is performed, there is a chance that some guessed key bits have a value of X. In that case, all key bits that are assigned with X are attacked using brute-force or functional hill-climbing attack [12].

IV. Security Analysis

The proposed attack has been applied to multiple obfuscation techniques using ISCAS85 benchmarks and the combinational circuits from the MCNC benchmark set. In this section, we demonstrate the security analysis results when applying the attack to the following obfuscation techniques:

- **TOC (MUX-Based):** MUX2 key-gates are inserted with dummy functions selected to maximize the Hamming distance between correct and the incorrect outputs [9].
- **IOLTS:** Key gates are inserted to locations with low controllability [10].
- **SARLock:** Comparators are added to mitigate the SAT attack by masking the output value [11].
- **Random:** MUX key-gates are inserted in random locations, and the dummy routes are also selected at random. For this locking technique, two variants have been generated, MUX2 and MUX4 key-gates.

A. Evaluation Metrics

The security analysis results include the accuracy and the precision metrics. We define accuracy as the percentage of correctly extracted key values out of the entire key size, and the precision as the percentage of correct keys while considering Xs as correct guesses. They are shown in Eqns. 6 and 7:

$$Accuracy = \frac{KEY_{correct}}{KEY_{total}} \times 100\% \tag{6}$$

$$Precision = \frac{KEY_{correct} + KEY_X}{KEY_{total}} \times 100\% \tag{7}$$

where $KEY_{correct}$ is the number of key bits that are correctly extracted, KEY_{total} is the total number of bits in the original key, and KEY_X is the total number of Xs in the that are generated by SWEEP. For example, let us consider the correct key value of a locked design to be ”00110”, and SWEEP extracted ”00X10”. Then the accuracy is 80%, and the precision is 100%. Whereas if the extracted key is ”00010”, then the accuracy is 80%, and the precision is 80%. We focused on the precision metric as it shows the level of confidence that SWEEP provides when a key value is extracted.

B. Selecting Margin Values

The proposed framework for the attack provides an adjustable parameter m that controls the margin when comparing weighted feature values. Choosing this parameter makes the attacker consider the tradeoff between the precision of the attack and the accuracy. Due to the varying nature of circuits used for data set generation, the precision values for some benchmarks did not reach 100% unless the margin value is extremely high. In other words, to achieve 100% precision for the entire feature matrix, the overall margin value had to be very high, which means the overall accuracy would drop dramatically. To overcome this issue, we created a two-stage process that obtains two margin values. The first value m_{100} is the one that provides a 100% precision, this margin will be used as the initial step that will highlight extracted keys with high confidence. The second margin m_{90} is set to maintain the an overall precision of at least 90%, which is used to extract more key values at the cost of risking some incorrect guesses.

C. Key Extraction Results

After performing SWEEP on the selected benchmarks, a set of accuracy and precision values are obtained for every obfuscation scheme. Each benchmark has been obfuscated four times to generate variants with an overhead allowance of 5%, 10%, 25%, and 50%; the results use a weighted average based on the number of key-bits in every variant. The results in Table II present the accuracy and precision as well as the CPU runtime. The average accuracy results for each technique suggests that most of the key values are correctly extracted. It is worth noting that depending on the locking mechanism, some techniques are more vulnerable to SWEEP than others. The nature of the design can also affect the accuracy of the attack, where designs with more convoluted structures and large pipelines (such as multipliers) exhibit a larger change in design features when the incorrect key value is applied.

V. Discussion

A. Key Value Correlation to Design Features

The correlation of the key to each design feature extracted from the synthesis report differs for each locking scheme. For example, the MUX-based TOC locking technique is designed to maximize the Hamming distance between the correct and the incorrect route, applying the incorrect key value will always result in a large area reduction, the number of edges, power consumption, and the gate count. This reduction occurs when the incorrect route is selected, which leaves the correct node unconnected, and is hence removed when sweeping the circuit. Similarly, the random insertion technique is also shown to have the same vulnerability. On the other hand, timing characteristics are usually less correlated to the key values when performing the locking by selecting less observable and controllable nodes (such as in IOLTS). These nodes are usually placed in non-critical paths and do not cause a significant change in the data arrival delay.

B. Runtime and Scalability

Table II clearly demonstrates that no matter how large the system-under-attack is, the run time is almost unaffected. Even designs that are complex for other types of attacks (such as SAT) will not increase the complexity of SWEEP. However, in

TABLE II: Accuracy and Precision Results for SWEEP for Different Locking Techniques

Benchmark	TOC			IOLTS			SARLock			Random (MUX2)			Random (MUX4)		
	Acc.	Pres.	Run Time (s)	Acc.	Pres.	Run Time(s)	Acc.	Pres.	Run Time(s)	Acc.	Pres.	Run Time(s)	Acc.	Pres.	Run Time(s)
c432	63.84	97.60	0.93	100	100	0.21	100	100	0.34	90.62	93.75	0.22	100	100	0.27
c499	54.79	98.67	0.92	100	100	0.22	100	100	0.33	96.87	96.87	0.16	90.62	96.8	0.28
c880	64.66	91.68	0.96	100	100	0.19	100	100	0.32	100	100	0.17	87.50	100	0.29
c1355	63.51	95.25	0.84	100	100	0.19	100	100	0.32	93.75	93.75	0.23	100	100	0.31
c1908	72.74	87.65	0.79	100	100	0.21	100	100	0.38	96.87	96.87	0.20	87.50	100	0.28
c2670	65.23	90.50	1.02	100	100	0.20	100	100	0.36	93.75	93.75	0.23	93.75	100	0.27
c3540	72.15	90.39	0.97	100	100	0.18	100	100	0.35	93.75	93.75	0.23	100	100	0.29
c5315	75.36	86.75	0.92	100	100	0.23	100	100	0.36	100	100	0.21	87.50	100	0.31
c6288*	**87.26**	**91.27**	**0.77**	**100**	**100**	**0.20**	**100**	**100**	**0.35**	**90.62**	**92.18**	**0.22**	**87.50**	**96.87**	**0.29**
c7552	78.39	90.29	0.84	100	100	0.20	100	100	0.31	96.87	98.43	0.19	93.75	100	0.33
apex2	71.40	91.54	0.89	100	100	0.23	100	100	0.34	96.87	98.43	0.23	90.62	100	0.31
apex4	90.17	90.47	0.73	100	100	0.20	100	100	0.36	100	100	0.19	87.50	100	0.3
dalu	83.70	97.65	0.81	100	100	0.23	100	100	0.35	90.32	90.32	0.21	93.75	100	0.32
des	86.86	96.34	0.72	100	100	0.19	100	100	0.30	96.87	96.87	0.18	100	100	0.28
ex1010	90.39	99.10	0.75	100	100	0.23	100	100	0.33	78.12	78.12	0.21	75	93.75	0.27
ex5	76.17	96.34	0.71	100	100	0.19	100	100	0.34	87.50	87.50	0.19	96.87	96.87	0.29
i4	70.29	95.81	0.72	100	100	0.17	100	100	0.30	100	100	0.17	100	100	0.31
i7	79.31	99.04	0.75	100	100	0.15	100	100	0.31	90.62	90.62	0.18	96.87	96.87	0.29
i8	82.97	96.79	0.81	100	100	0.19	100	100	0.33	96.87	96.87	0.17	93.75	100	0.29
i9	73.98	95.84	0.72	100	100	0.21	100	100	0.35	78.12	78.12	0.23	93.75	93.75	0.3
k2	82.84	98.86	0.65	99.88	99.88	0.20	100	100	0.34	99.88	96.87	0.15	65.62	96.87	0.29
seq	86.86	98.59	0.71	100	100	0.20	100	100	0.34	90.32	90.32	0.24	71.87	90.62	0.31
Average	**76.3**	**94.38**	**0.81**	**99.99**	**99.99**	**0.20**	**100**	**100**	**0.33**	**93.57**	**93.79**	**0.20**	**90.62**	**98.29**	**0.29**

* The highlighted benchmark is a 32-bit-multiplier that SAT attack is not able to break.

order to extract the features from the generated training data set, a large number of training data sets have been generated and attacked. The run time for the entire process is around 7 hours using an Intel Xeon 4-core 2.80GHz processor and 32GB of RAM.

C. Accuracy vs. Precision

To better understand the relationship between accuracy and precision, a simple analysis has been performed in Fig. 2, which visualizes the results when a number of attacked locking techniques when adjusting the acceptable margin m value.

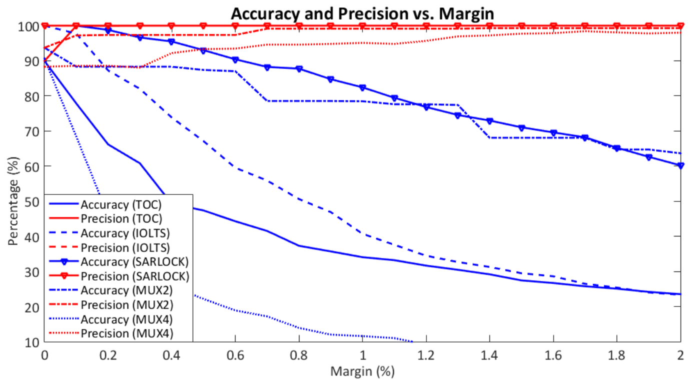

Fig. 2: Accuracy and Precision vs. Margin.

Fig 2 clearly indicates that as precision gets higher, the key extraction accuracy falls sharply. The applied locking technique, and the type of locked IP, play a major role in the relation between accuracy and precision. When fixing the value of accuracy, higher precision values indicate that the used locking technique is highly breakable by SWEEP, while a lower accuracy value suggests that only a partial-key can be extracted using our attack.

VI. Conclusion

In this paper, we have presented SWEEP, a constant propagation attack on logic locking that exploits the behavior of the system's performance when key values are hard-coded to the obfuscated system and re-synthesized. The proposed methodology starts with evaluating design features and weights them based on their correlation to the correct key values. Different analytical approaches can be applied, from a simple area comparison to the analysis of multiple features produced by assigning a value to each key location. The results specifies how many keys are correctly extracted, where the value and the location of these keys are accurately determined. This way, even a low accuracy result would still reduce the key-space dramatically. Future work should focus on the extraction of more design features in order to improve the accuracy of the attack, performing the attack sequential locking techniques, and finding a countermeasure that eliminates the correlation between correct key values and design features.

VII. Acknowledgment

This work is supported in part by the National Science Foundation grants: 1603483, and 1651701.

References

[1] OECD, "The Economic Impact of Counterfeiting and Piracy," in *Organisation for Economic Co-operation and Development,* 2007.
[2] R. S. Chakraborty, and S. Bhunia, "Hardware Protection and Authentication through Netlist Level Obfuscation," in *ICCAD,* pp. 674-677. 2008.
[3] M. Yasin, J. J. V. Rajendran, O. Sinanoglu, and R. Karri., "On improving the security of logic locking," in *IEEE Trans. on CAD of Integrated Circuits and Systems,* 1411-1424, 2016.
[4] P. Subramanyan, S. Ray and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *HOST,* pp. 137-143, 2015.
[5] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST),* 2017.
[6] P. Chakraborty, J. Cruz, and S. Bhunia "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," in *Asian Hardware Oriented Security and Trust Symposium (AsianHOST),* 2018.
[7] M. Massad, J. Zhang, S. Garg and M.V. Tripunitara, "Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism," *in arXiv preprint arXiv:1703.10187,* 2017.
[8] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits, in *2008 Design, Automation and Test in Europe,* 2008.
[9] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault Analysis-Based Logic Encryption. IEEE Transactions on Computers, 64(2), Feb 2015.
[10] S. Dupuis, P. Ba, G. Di Natale, M. Flottes and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans," in *IOLTS,* pp. 49-54, 2014.
[11] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST),* pp. 236-241, 2016.
[12] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 2015.