# DOSCrack: Deobfuscation Using Oracle-guided Symbolic Execution and Clustering of Binary Security Keys

Jiaming Wu, University of Florida, jiaming.wu@ufl.edu
Olivia Dizon-Paradis, University of Florida, paradiso@ufl.edu
Sazadur Rahman, University of Central Florida, mohammad.rahman@ucf.edu
Damon Woodard, University of Florida, dwoodard@ufl.edu
Domenic Forte, University of Florida, dforte@ece.ufl.edu

***Abstract*—Design-for-test/debug (DfT/D) introduces scan chain testing to increase testability and fault coverage by inserting scan flip-flops. However, these scan chains are also known to be a liability for security primitives. In previous research, dynamically obfuscated scan chains (DOSC) were introduced to protect logic-locking keys from scan-based attacks by obscuring test patterns and responses. In this paper, we present DOSCrack, an oracle-guided attack to de-obfuscate DOSC using symbolic execution and binary clustering, which significantly reduces the candidate seed space to a manageable quantity. Our symbolic execution engine employs scan mode simulation as well as satisfiability modulo theories (SMT) solvers to reduce the possible seed space, while obfuscation key clustering allows us to effectively rule out a group of seeds that share similarities. An integral component of our approach is the use of sequential equivalence checking (SEC), which aids in identifying distinct simulation patterns to differentiate between potential obfuscation keys. We experimentally applied our DOSCrack framework on four different sizes of DOSC benchmarks and compared their run-time and complexity. Our research effectively addresses critical vulnerabilities in scan-chain obfuscation methodologies, offering insights into DfT/D and logic locking for both academic research and industrial applications. Our framework emphasizes the need to craft robust and adaptable defense mechanisms against scan-based attacks.**

***Index Terms*—logic locking, scan-based attack, clustering, symbolic execution**

## I. Introduction

Scan-based testing is a commonly practiced design-for-test (DfT) scheme that facilitates the detection and diagnosis of faults in integrated circuits (ICs) because of its high controllability and observability [1]. By replacing registers with scan flip-flops and connecting them into scan chains, DfT allows access to internal nets and assists in the extraction of register values in sequence. DfT and scan chain architectures are integral components that contribute significantly to the efficiency and effectiveness of IC testing processes and yield improvement. For example, in the pre-silicon design of IC, Synopsys Tetramax [2] is popularly used for automatic test pattern generation (ATPG) and silicon testability analysis, which automates the process of generating test patterns to test digital ICs for potential defects. In post-silicon testing, the JTAG [3] and Nexus standards are widely adopted, using ATPG test patterns to perform boundary scan tests and debugging.

By increasing controllability and observability, scan flip-flops also introduce weaknesses to scan-based attacks. Scan-based attacks, which are types of side-channel attacks, aim to extract secret keys through the analysis of scan data obtained from scan chains. In order to secure crypto-chips from scan-based attacks, multiple countermeasures have been proposed. These are mainly categorized into two strategies: *scan chain obfuscation* and *scan I/O restriction*. Scan chain obfuscation aims to prevent attackers from controlling the scan chain by modifying the scan structure, inserting obfuscation gates, or adding sub-chains alongside the original scan chain. Agrawal et al. [4] proposed an obfuscated scan chain structure that incorporates XOR gates at random points in the scan chain. Atobet et al. [5] proposed the state-dependent scan flip-flop that replaces scan flip-flop at random points to prevent attackers from identifying the correct scan timing. Lee et al. [6] proposed subchain modification techniques that allow Lock & Key controls and scan order obfuscation to prevent attacks from accessing the scan structure. The Dynamic Obfuscated Scan Chain [7] integrates the permutation of scan chains with XOR gates and employs logic locking techniques using dynamic keys. Moreover, DOSC incorporates a shadow chain that restricts the dynamic keys from leakage to the scan output. This means that both obfuscation and scan I/O restriction are applied in DOSC, providing a robust defense mechanism against unauthorized access or attacks.

In this paper, we propose the DOSCrack, a novel strategy to deobfuscate dynamically obfuscated scan chains (DOSC). Our contributions are listed below.

- We proposed DOSCrack, a novel deobfuscation framework that incorporates structural analysis, symbolic execution, and key candidate clustering.
- We incorporated sequential equivalence checking to pinpoint distinguishing patterns that effectively differentiate potential keys between clusters.
- We experimentally applied our framework on different DOSC benchmarks with different bits of seed. The framework demonstrates scalability, with the recorded run time exhibiting a proportional increase as the seed size grows.

The rest of the paper is organized as follows. Section II gives the necessary background of techniques as well as the

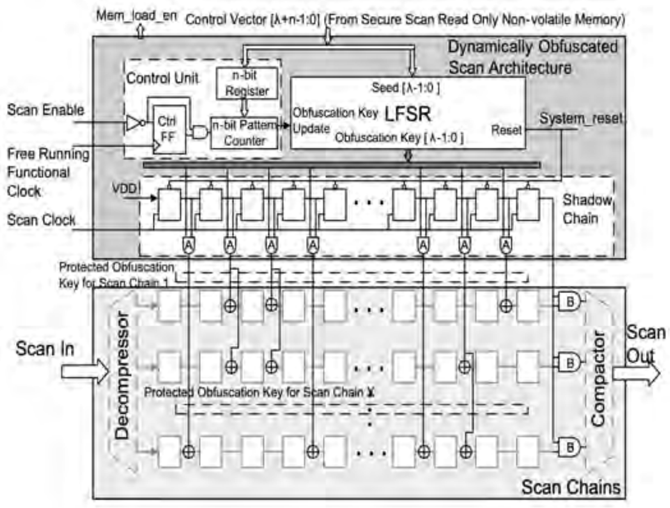

Mem_load_en
Control Vector [λ+n-1:0] (From Secure Scan Read Only Non-volatile Memory)
Dynamically Obfuscated Scan Architecture
Control Unit
n-bit Register
Seed [λ-1:0]
Scan Enable
Ctrl FF
n-bit Pattern Counter
Obfuscation Key Update
LFSR
Obfuscation Key [λ-1:0]
Reset
System_reset
Free Running Functional Clock
VDD
Shadow Chain
Scan Clock
Protected Obfuscation Key for Scan Chain 1
Scan In
Decompressor
Protected Obfuscation Key for Scan Chain X
Compactor
Scan Out
Scan Chains

Fig. 1. Dynamically obfuscated scan chain architecture. [7]

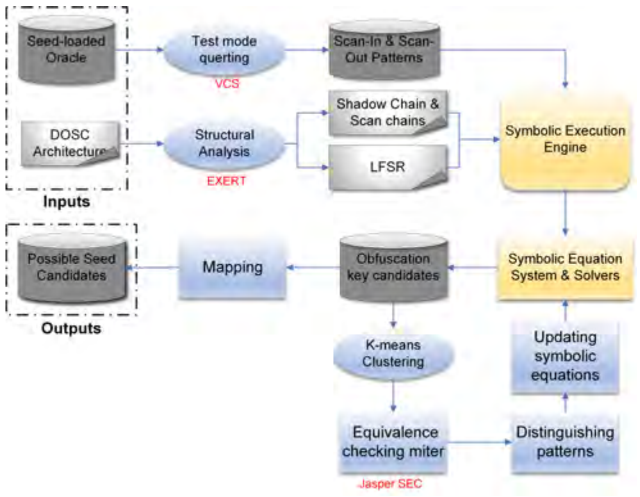

Seed-loaded Oracle
Test mode querting
VCS
Scan-In & Scan-Out Patterns
DOSC Architecture
Structural Analysis
EXERT
Shadow Chain & Scan chains
LFSR
Symbolic Execution Engine
Inputs
Possible Seed Candidates
Mapping
Obfuscation key candidates
Symbolic Equation System & Solvers
Outputs
K-means Clustering
Updating symbolic equations
Equivalence checking miter
Jasper SEC
Distinguishing patterns

Fig. 2. Workflow of DOSCrack deobfuscation.

structure of the dynamically obfuscated scan chain. Section III introduces the novel DOSCrack deobfuscation framework. Section IV analyzes the results of applying our framework to DOSC benchmarks. Finally, Section V concludes the paper with key takeaways and future works.

## II. Background and Preliminary Concepts

In this section, we provide an overview of the architecture of DOSC and introduce the fundamental concepts of symbolic execution as well as our threat model.

### A. DOSC Architecture

The DOSC [7] architecture is shown in Figure 1. It consists of four parts: the control unit, the LFSR (linear feedback shift register), the shadow chain, and the obfuscated scan chain. The control unit generates signals that load the seed from non-volatile memory and regulates the clock frequency of the shadow chain. Then the LFSR takes the seed for obfuscated key sequence generation, and the shadow chain protects the obfuscated key from potential differential attacks. The DOSC seed is considered confidential information to interpret correct test responses. If DOSC is used to protect logic-locked circuits, knowing DOSC's seed would allow an attacker to perform Boolean satisfiability (SAT) attacks [8] against the locked functional circuit.

Previously. the Boolean satisfiability (SAT) attack was performed against the DOSC architecture itself in an attempt to obtain the LSFR's seed. To do so, sequential circuit unrolling was utilized [9]. It was found that DOSC architecture was robust against SAT attacks because such unrolling inevitably results in scalability issues for Boolean SAT solvers.

### B. Symbolic Execution

Symbolic execution is a program analysis technique used in testing, debugging, and verification. Instead of executing a program with concrete input values, symbolic execution operates on symbolic values and expressions. Symbolic execution engines are often combined with simulations to generate feasible execution paths and further utilize satisfiability (SAT) or satisfiability modulo theories (SMT) solvers to find the executing patterns. In our deobfuscation framework, we leverage the built-in symbolic execution engine from EISec [10] to convert our target netlist into C code. Subsequently, the generated C code undergoes symbolic modeling.

### C. Threat Model

In this section, we briefly review the threat model of Design-for-Test/Debug (DfT/D) and present the assumptions of our DOSCrack attack framework. In the context of semiconductor supply chains, the design house typically dispatches the DfT/D inserted netlist to contract third-party fab/foundries for the production of chips and performing scan testing (i.e., JTAG) on fabricated chips. These contract foundries, therefore have full access to the scan chains embedded in the design as well as the DfT/D insertion techniques. This accessibility presents a potential risk as it allows these facilities to conduct scan-based attacks. Based on this context, our DOSCrack framework operates under the assumption that potential attackers have knowledge of the DOSC architecture and access to the scan chain. This threat model also aligns well with Kerckhoffs's principle which states that the security of a cryptosystem must only lie in the secrecy of its keys and everything else should be considered public knowledge. In our threat model, the attacker's goal is to find DOSC's seed.

While many scan-based attacks (i.e. differential scan-based attacks) necessitate certain understanding of the chip's functional logic to be effective, our method exclusively relies on running an unlocked chip (or simulating an unlocked design) in scan mode and does not require any knowledge of the functional logic. This feature sets our DOSCrack framework apart from many other oracle-guided attacks.

## III. DOSCrack Framework

An overview of the DOSCrack framework is illustrated in Figure 2. Our objective is to narrow down the candidate seed space to a manageable quantity, where we can eliminate incorrect obfuscation keys until only a singular valid seed remains.

**Algorithm 1** Structual Analysis

```
Input: DOSC netlist $N$;
Output: LFSR netlist, scan chain netlist;
1: DOSC netlist $\rightarrow$ EXERT interaction analysis
2: if feedback nets detected then
3:     FSM registers $\leftarrow$ LFSR
4:     feedback nets connectivity $\leftarrow$ XOR gate inputs
5: else if feedback nets not detected then
6:     datapath registers $\leftarrow$ scan chains
7: end if
8: return LFSR netlist, Scan chain netlist; feedback nets connectivity
```

Our framework is composed of four integral components: First, we use an unlocked chip (or equivalently simulate an unlocked chip's DOSC and scan chain) to act as an oracle; second, we employ structural analysis to distinguish between the Linear Feedback Shift Register (LFSR) and scan chains; third, we conduct symbolic execution followed by employing an SMT solver to rule out keys; and fourth, we utilize clustering algorithms to efficiently categorize the remaining key candidates. This process iterates until the number of candidate seeds/keys is small enough to brute force.

### A. Structural Analysis

As explained in the threat model (Section II-C), we assume that there is access to an open-source DOSC architecture and the attacker's goal is to obtain the LFSR seed. Structural analysis begins by identifying the LFSR, shadow chain, and scan chains so that symbolic engines can modulate each part separately. The LFSR is typically implemented as a finite state machine (FSM) in a physical context. This implementation means that the LFSR is designed to transition between a finite number of states denoted by state registers. On the other hand, scan chains are implemented with datapath registers. These registers are used to store and shift data through the scan chain during testing or debugging processes, which allows for sequential loading and shifting of test data. Therefore, structural analysis distinguishes the LFSR and scan chains based on their distinct physical implementations and functional roles.

Algorithm 1 shows the detailed steps of our structural analysis. During structural analysis, the interaction analysis tool EXERT [11] is employed to identify FSMs and datapaths (line 1). This identification is based on the characteristic features of the FSM that align with the expected behavior of an LFSR, primarily its feedback network typically constructed from XOR gates (line 2 to 6). This process effectively separates the LFSR and scan chains (line 8), allowing for more precise modeling with symbolic engines.

### B. Oracle Interaction in Test Mode

To apply the proposed deobfuscation framework, random input sequences are applied to the target oracle which already has an LFSR activation seed for its scan chain. These patterns/responses are provided/collected in test mode, where the random inputs with a certain number of sequences are fed into

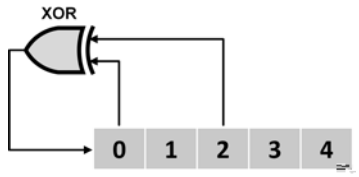


Fig. 3. Structure of a 5-bit LFSR.

TABLE I
TABLE OF SYMBOLIC REPRESENTATION FROM SEED TO OBFUSCATION KEY FOR EACH LFSR BIT ($0 \leq i \leq 4$) AT CLOCK CYCLES 1,2, ..., $t$ FOR THE EXAMPLE IN FIG. 3.

| $i$ / Cycle | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
| 1 | $s_0 \oplus s_2$ | $s_0$ | $s_1$ | $s_2$ | $s_3$ |
| 2 | $s_0 \oplus s_2 \oplus s_1$ | $s_0 \oplus s_2$ | $s_0$ | $s_1$ | $s_2$ |
| $\vdots$ | | | $\vdots$ | | |
| $t$ | $L^0_{t-1} \oplus L^2_{t-1}$ | $L^1_{t-1}$ | $L^2_{t-1}$ | $L^3_{t-1}$ | $L^4_{t-1}$ |

the scan input (SI) port, while an equivalent number of patterns are shifted out from the scan out (SO) port after certain clock cycles. We utilize the oracle in test mode, which ensures that the resulting patterns do not incorporate functional logic. This also improves the performance of our deobfuscation framework by focusing the attack only on the scan chain.

### C. Symbolic Execution Engine and Symbolic Equation System

In this section, we delve into the methodology employed by our symbolic execution engine, with a specific focus on the process of recovering the symbolically assigned $n$-bit seed, with bits denoted as $(s_0, s_1, \cdots, s_{n-1})$ from the symbolic equation system. This system is constructed through the symbolic modeling of both the LFSR and the scan chain.

*1) Symbolic Execution Engine:* The symbolic execution engine modulates the LFSR and the scan chains by converting the target netlist to functionally equivalent C code, where every bit of the unknown seed is represented as a symbolic variable. As our structural analysis identifies the LFSR and the scan chains, the symbolic execution engine proceeds to model the LFSR and the scan chains separately. An LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. Structural analysis provides detailed information about the connectivity of the feedback nets, which represent the locations within the LFSR register where the XOR gate receives its inputs. To symbolically represent LFSR outputs, we denote these two inputs as $(x, y)$. We describe the $i$-th bit output of LFSR at cycle $t$ as $L^i_t$. Thus, a general LFSR output at cycle $t$ is represented as:

$$L^i_t = \begin{cases} L^x_{t-1} \oplus L^y_{t-1}, & \text{when } i = 0; \\ L^i_{t-1}, & \text{otherwise} \end{cases}$$

This equation modeling the LFSR outputs can be described as follows. The first bit of the LFSR is always connected to the output of the XOR gate. At every clock cycle, the first bit of the LFSR is updated based on the output from the XOR gate,

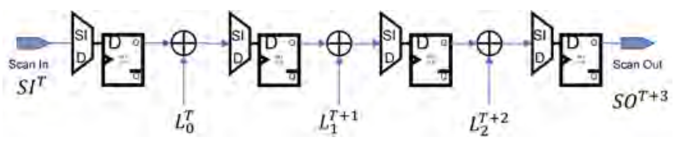


Fig. 4. Example of scan chain modeling with $N = 3$ at any clock cycle $T$. The scan out is denoted as $SO^{T+3} = SI^T \oplus L_0^T \oplus L_1^{T+1} \oplus L_2^{T+2}$

with the inputs of the XOR gate identified through structural analysis as being at positions $(x, y)$ within the LFSR. For the rest of the bits in the LFSR, from position 1 to $n-1$, the shifting process occurs when each bit shifts from its previous state to the next position.

Figure 3 shows an example of 5-bit LFSR with $x = 0$ and $y = 2$. Table I shows the cycled output based on seed $(s_0, s_1, \cdots, s_4)$. At cycle 0, the seed $(s_0, s_1, \cdots, s_4)$ is loaded into the LFSR cells and the output of any cycle $t$ can be computed based on the connectivity information $(x, y)$ obtained from structural analysis. Thus we define the function $f$ which represents the LFSR output given the seed and cycle $t$ as $\vec{L}_t = f[(s_0, s_1, \cdots, s_{n-1}), t]$, where $\vec{L}_t$ denotes the LFSR output in a vector form.

The symbolic engine models the shadow chain and the scan chains together. Both scan chains and shadow chains can be conceptualized as cascading datapath registers, receiving inputs from the SI port of the scan chain and the obfuscation key input, whereas the outputs are directed to the SO port. The symbolic engine then models the transformed C code, treating it as a symbolic equation representing the relationship between the inputs and the outputs.

Table II shows an example of conversion from netlist to C code to symbolic equation for a single scan chain cell. This scan chain cell is located at the end of the scan chain that connects directly to the SO port. By generating every symbolic equation for all scan cells, the symbolic equation that connects the SI port to the SO port is formulated and represented as $SO_{T+N} = SI_T \bigoplus_{t,i=0}^{N} L_t^{T+i}$, where $N$ denotes the length of scan chains and $T$ denotes the cycle when scan in patterns are shifted into the scan chain. The $\bigoplus_{t,i=0}^{N}$ denotes the continuous XOR operation of $L_t^i$ and is derived by symbolically modeling the scan chains.

Figure 4 shows an example of scan chain modulation under this equation with $N = 3$, where $SO_{T+3} = SI_T \oplus L_0^T \oplus L_1^{T+1} \oplus L_2^{T+2}$. This equation captures the relationship between the Scan In (SI) and Scan Out (SO) patterns, effectively encapsulating the dynamics of how the obfuscation key values are processed within the scan chain. Given the tracking of SI and SO patterns with correlated obfuscation keys, we define the function $g$ where $SO_{T+N} = g[SI_T, (L_0^T, L_1^T, \cdots, L_t^{T+N-1})]$ which is derived such that the SO pattern is symbolically represented with corresponding SI pattern with a certain sequence of XOR operation on obfuscation key.

*2) Symbolic Equation System and Solver:* In the previous section, the obfuscation keys are symbolically represented in Table I, and the scan chains are modeled with symbolic equations in Table II in our running example. The two symbolic equations then form simultaneous equations $f$ and $g$:

TABLE II
TABLE OF CONVERSION FROM NETLIST TO C CODE AND SYMBOLIC EQUATION SETUP

| | |
|---|---|
| Netlist | wire shadow_chain/N1, Scan_out;<br>SDFFQX_scan_reg[1] (.D(n1),.SI(out[1] ),.SE(test_se), .CK(CK),.Q(Scan_out);<br>AND2X1 U1(.A(in[0],.B(shadow_chain/N1,.Y(out[1]); |
| C code | bool shadow_chain/N1, Scan_out;<br>out[1] = in[0] & shadow_chain/N1;<br>if(test_se == 0) {Scan_out = n1;}<br>else {Scan_out = out[1]; m+=1;} |
| Symbolic equation | Scan_out == shadow_chain/N1 & in[0] |

$$\begin{cases} \vec{L}_t = f\left[(s_0, s_1, \cdots, s_{n-1}), t\right]; \\ SO_{T+N} = g\left[SI_T, (L_0^T, L_1^T, \cdots, L_t^{T+N-1})\right] \end{cases}$$

As discussed in Section III-B, the scan-in and scan-out patterns are generated from test mode interactions with the oracle. By putting $m$ pairs of simulated corresponding SI and SO patterns into the simultaneous equations, we construct the symbolic equation system that encompasses $m$ equations, each representing their relationship and transformation. As a result, in the symbolic equation system we developed, the assigned seed constitutes the only set of symbolic variables. By applying an SMT solver to this system of equations, we can effectively solve for possible solutions of these variables and recover candidate seeds.

## *D. Obfuscation Key Clustering*

Our system utilizes the SMT solver to generate solutions for a symbolic equation system, which in turn produces the space of obfuscation keys and the corresponding seed values. To effectively narrow down the possible seed space, it is essential to introduce more symbolic equations by further querying more SI patterns to the oracle to produce more symbolic equations. However, the effectiveness of generating SO and SI patterns from random simulations diminishes over iterations. This diminishing effect occurs because the candidates for obfuscation keys begin to form similarities, making it increasingly difficult for random SI vectors to effectively differentiate between them based on SO patterns. As a result, the random simulation approach becomes less capable of exploring and identifying differences among potential obfuscation keys.

To address this problem, we applied the K-means clustering algorithm to categorize existing candidates of obfuscation keys into groups to rule them out as a whole. The overall workflow is illustrated in Figure 5. After using symbolic execution and equation solvers, we produce a certain number of key candidates as a starting point. We then utilize the *Hamming distance* (HD) metric, ideal for binary data, for the K-means clustering algorithm to cluster key candidates. From two distinct clusters, we select centroid keys and map them back to their corresponding symbolic seeds. We then construct a miter with an XOR gate for Sequential Equivalence Checking (SEC), which compares DOSC architectures loaded

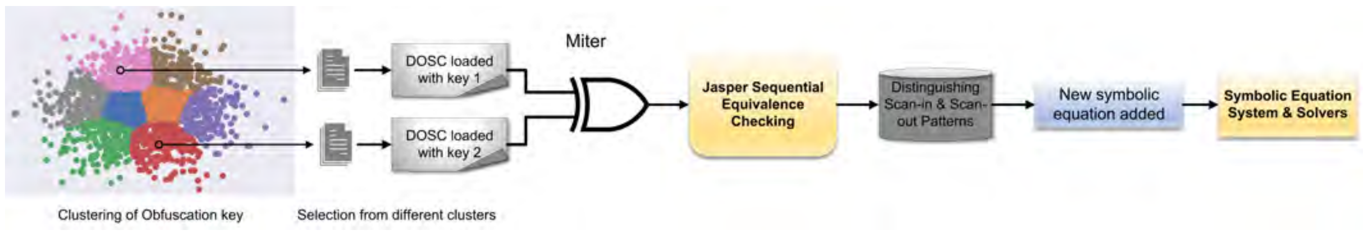

Miter
DOSC loaded with key 1
DOSC loaded with key 2
Jasper Sequential Equivalence Checking
Distinguishing Scan-in & Scan-out Patterns
New symbolic equation added
Symbolic Equation System & Solvers
Clustering of Obfuscation key
Selection from different clusters

Fig. 5. Workflow of generating distinguishing patterns with Jasper SEC.

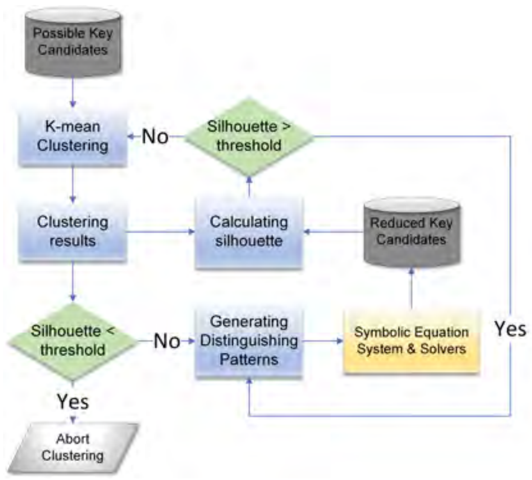


Fig. 6. Flow chart with checkpoints that evaluate silhouette scores. Two silhouette score thresholds determine when to abort clustering and when to redo clustering.

with different seeds, generating distinguishing input patterns tailored to each cluster. We query the oracle with this tailored input sequence to form a new symbolic equation that is then added to our symbolic equation system and solvers. This approach effectively addresses the limitations of random simulation. The generated input sequence is specifically designed to produce distinct outputs, which implies that the corresponding symbolic equation can eliminate one of the two key candidates. Furthermore, due to the similarity of keys within the same cluster, this equation is likely to rule out even more keys in that cluster! Thus, this method efficiently solves the issue of diminishing returns in random simulation. By iteratively selecting different keys from various clusters, we can formulate more equations, effectively reducing the key candidate space to a size suitable for brute force.

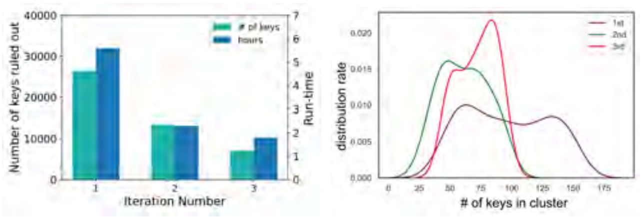
Our process involves iterative re-clustering as more obfuscation keys are ruled out and the clarity of cluster margins decreases. Figure 6 shows the flowchart of the conditions under which re-clustering should be invoked, alongside decision points for transitioning to brute-force elimination. During the iterative process of selecting clusters, we incorporate two critical checkpoints to assess the silhouette score, a common metric for evaluating the quality of clustering results. The initial checkpoint after K-means clustering ensures effective clustering results. A silhouette score below 0.6 triggers a switch to brute-force key elimination. A subsequent evaluation occurs after eliminating a certain number of keys. At this point, we calculate the silhouette score, taking into account both the reduced key set under the initial cluster formations. If the silhouette score remains above the threshold, indicates that clustering is still effective and the generation of distinguishing patterns is still efficient. Conversely, a score drop below this threshold signals the need for re-clustering. These silhouette score checkpoints ensure our approach dynamically adapts, maintaining efficiency in isolating key groups with each iteration.

## IV. Experimental Results and Evaluation

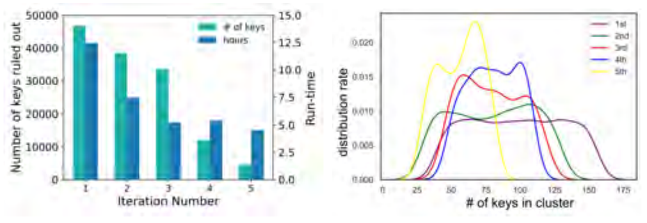
We synthesize four DOSC benchmarks with different LFSR seed sizes (8-bit, 16-bit, 32-bit, and 64-bit) to evaluate DOSCrack. Our framework is tested exclusively based on test mode simulation of a seed-loaded oracle netlist with the functional input port of the scan flip-flop open. To further simplify the application of our DOSCrack method, we establish only a single scan chain for each benchmark and the length of scan cells is identical to its seed size for every benchmark. We operate under the assumption that these seeds are not reachable or accessible during normal operation as would be the case with a physical oracle chip. Test mode simulation is performed by Synopsys VCS. The benchmarks are synthesized with Synopsys Design Compiler. We used an Intel(R) Xeon E5-2450L CPU for the synthesis of DOSC benchmarks, test mode simulation, and running of DOSCrack. The run time results are shown in Table III and illustrated in Figure 7.

*1) DOSC with 8-bit Seed:* Our testing of the DOSCrack framework starts with a DOSC benchmark configured with an 8-bit seed and a corresponding 8-bit LFSR. Both the scan chains and the shadow chain comprise eight scan cells each, aligning with the 8-bit size of the seed. Given the modest 8-bit size of the seed, our symbolic execution engine operates efficiently and the SMT solver has generated just 3 potential key candidates. Consequently, there is no requirement to employ clustering algorithms to manage the obfuscation key candidates' space. The total run time is 10.2 minutes and the simulation time for SI/SO patterns is 8.3 minutes. The remaining 3 keys were trivially brute forcing pruned to find DOSC's actual LFSR seed.

*2) DOSC with 16-bit Seed:* In our evaluation of a DOSC benchmark with a 16-bit seed under the same configuration and simulation, the SMT solver identified 356 potential seed candidates, apt for clustering analysis. Initial clustering formed 14 groups with a silhouette score of 0.68, suggesting effective grouping. By selecting cluster combinations for the Jasper SEC system, 91 distinguishing patterns were generated, each



(a)



(b)

Fig. 7. Number of keys ruled out with run-time bar charts and Clustering histograms for (a) 32-bit DOSC and (b) 64-bit DOSC per iteration of clustering. With each successive re-clustering iteration in our process, the distribution of keys within clusters becomes increasingly narrow.

TABLE III
RESULTS OF DOSCRACK WITH DIFFERENT SEED SIZES. × MEANS NOT APPLICABLE. IN THE TOTAL RUN TIME ROW, 'M', 'H', AND 'D' DENOTE MINUTES, HOURS, AND DAYS.

| **DOSC seed size** | 8-bit | 16-bit | 32-bit | 64-bit |
|---|---|---|---|---|
| # of Scan IO patterns | 400 | 400 | 500 | 500 |
| # of Obfuscation Key Candidates | 3 | 356 | 50288 | $\leq 2^{17}$ |
| # of Clusters | × | 14 | 107 | 2440 |
| # of Seed Candidates | 3 | 23 | 327 | 6508 |
| Total Run Time | 10.2m | 37.9m | 858m | 3d23h |

forming a symbolic equation to help eliminate roughly 4 possible key candidates. This reduced the key candidates' space to 23 keys without the need for further clustering. Subsequent brute-force methods then identified the correct seed candidates within the total runtime of 37.9 minutes, demonstrating the approach's efficiency.

*3) DOSC with 32-bit Seed:* Unlike the previous test, we increase the number of initial SI/SO patterns generated from simulation as we aim to explore the diminishing issues explained in Section III-D. We generated 400 SI/SO patterns to build symbolic equations and further generated another 100 patterns to compare the number of solutions from the SMT solver. This led to the SMT solver identifying 50,354 solutions with 400 patterns and 50,288 with 500, demonstrating that a 25% increase in patterns only reduced key candidates by 66. This highlights the diminishing issue, underscoring the necessity of clustering and SEC miter techniques for producing distinguishing patterns. Through clustering, we narrowed down the key candidates from 50,288 to 327 across 107 clusters. As the process progressed, a linear decrease in the HD between clusters was observed 7(a), indicating the remaining keys within each cluster become more similar.

*4) DOSC with 64-bit Seed:* The biggest benchmark we tested emphasizes the 64-bit seed of DOSC. The initial outcome from the SMT solver reveals approximately 150,000 obfuscation key candidates, which equates to around $2^{17}$ solutions. Clustering results show the number of 2440 groups and we go through iterations of clustering whenever the silhouette score is below 0.6. Figure 7(b) shows the run-time results in the bar chart and the clustering distribution in the histogram. The 64-bit DOSC takes 5 iterations of clustering and the number of keys distribution becomes more narrow through each iteration. The final phase of our analysis involved pruning out the remaining 6,508 key candidates using a brute-force approach. The average time taken to brute force test 1,000 random keys in our analysis, often referred to as the 'pruning time', is approximately 2.5 hours.

## V. SUMMARY AND FUTURE WORK

In this paper, we present DOSCrack, an oracle-guided attack that utilizes symbolic execution and binary clustering to break DOSC. By applying DOSCrack on different sizes of benchmarks, our framework successfully reduced the key candidates' number and broke the 64-bit seed DOSC in 3d23h, which is much more efficient compared to the run-out threshold of 10 days for traditional SAT attacks. Additionally, in our testing, the brute-force effort to rule out 1,000 keys took approximately 2.5 hours. This duration highlights the significantly greater complexity of the brute-force method compared to our approach. Future work in this area could concentrate on developing more advanced algorithms for intelligently selecting different clusters to generate distinguishing patterns. By enhancing the method of choosing clusters, we may further improve the distinctiveness of the patterns generated, thereby more effectively ruling out obfuscation keys.

## REFERENCES

[1] J. Aerts and E. J. Marinissen, "Scan chain design for test time reduction in core-based ics," in *Proceedings International Test Conference 1998 (IEEE Cat. No. 98CH36270)*, pp. 448–457, IEEE, 1998.

[2] "Testmax atpg:advanced pattern generation." https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-atpg.html.

[3] "Ieee standard for test access port and boundary-scan architecture," *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pp. 1–444, 2013.

[4] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay, "Scan based side channel attacks on stream ciphers and their counter-measures," in *Progress in Cryptology - INDOCRYPT 2008* (D. R. Chowdhury, V. Rijmen, and A. Das, eds.), (Berlin, Heidelberg), pp. 226–238, Springer Berlin Heidelberg, 2008.

[5] Y. Atobe, Y. Shi, M. Yanagisawa, and N. Togawa, "State dependent scan flip-flop with key-based configuration against scan-based side channel attack on rsa circuit," in *2012 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 607–610, 2012.

[6] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 325–336, 2007.

[7] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically obfuscated scan for protecting ips against scan-based attacks throughout supply chain," in *2017 IEEE 35th VLSI Test Symposium (VTS)*, pp. 1–6, 2017.

[8] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137–143, IEEE, 2015.

[9] M. S. Rahman, A. Nahiyan, S. Amir, F. Rahman, F. Farahmandi, D. Forte, and M. Tehranipoor, "Dynamically obfuscated scan chain to resist oracle-guided attacks on logic locked design." Cryptology ePrint Archive, Paper 2019/946, 2019. https://eprint.iacr.org/2019/946.

[10] F. Fowze, M. Choudhury, and D. Forte, "Eisec: Exhaustive information flow security of hardware intellectual property utilizing symbolic execution," in *2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6, 2022.

[11] J. Wu, F. Fowze, and D. Forte, "Exert: Exhaustive integrity analysis for information flow security," in *2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6, 2022.