Article

Protecting Dynamically Obfuscated Scan Chain Architecture from DOSCrack with Trivium Pseudo-Random Number Generation †

Jiaming Wu [1,*], Olivia Dizon-Paradis [1], Sazadur Rahman [2], Damon L. Woodard [1] and Domenic Forte [1]

[1] Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA; paradiso@ufl.edu (O.D.-P.); dwoodard@ece.ufl.edu (D.L.W.); dforte@ece.ufl.edu (D.F.)

[2] Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816, USA; mohammad.rahman@ucf.edu

* Correspondence: jiaming.wu@ufl.edu

† This paper is an extended version of our paper published in 2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST).

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, the dynamically obfuscated scan chain (DOSC) was 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 and 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 runtime and complexity. Finally, we propose a low-cost countermeasure to DOSCrack which incorporates a nonlinear feedback shift register (NLFSR) to increase the effort of symbolic execution modeling and serves as an effective defense against our DOSCrack framework. Our research effectively addresses a critical vulnerability in scan-chain obfuscation methodologies, offering insights into DfT/D and logic locking for both academic research and industrial applications. Our framework highlights the need to craft robust and adaptable defense mechanisms to counter evolving scan-based attacks.

Keywords: scan-based attack; clustering; symbolic execution; logic locking

check for updates

Academic Editor: Christoforos Ntantogian

Received: 18 October 2024
Revised: 7 December 2024
Accepted: 7 January 2025
Published: 14 January 2025

Citation: Wu, J.; Dizon-Paradis, O.; Rahman, S.; Woodard, D.L.; Forte, D. Protecting Dynamically Obfuscated Scan Chain Architecture from DOSCrack with Trivium Pseudo-Random Number Generation. *Cryptography* **2025**, *9*, 6. https://doi.org/10.3390/cryptography9010006

1. Introduction

Scan-based testing is a commonly practiced design-for-test (DfT) scheme that facilitates detection and diagnosis of faults in integrated circuits (ICs) because of the high controllability and observability it provides [1]. By replacing common flip-flops with scan flip-flops and connecting them serially, DfT allows access to internal nets and assists in the extraction of test response values in sequence. Design for testability and scan chain architecture play crucial roles in the realm of IC design and testing. They 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 ICs, 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.

While the scan chain architecture incorporates DfT principles to improve IC testing, the controllability provided by scan flip-flops also introduces weaknesses in on-chip security primitives to non-invasive attacks, and the observability of scan chain outputs may create paths that leak information, thus introducing further security risks. 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 prior research, scan-based attacks against advanced encryption standard (AES), data encryption standard (DES), and Rivest–Shamir–Adleman (RSA) crypto modules have been proposed to extract secret keys through scan chains [4]. Scan-based attacks are commonly classified into two groups: differential scan-based attacks and signature scan-based attacks. Differential scan-based attacks (DSAs) take advantage of a specific characteristic in the round function of block ciphers, where two unique inputs can lead to output vectors that exhibit a distinctive Hamming distance following a single encryption round [5]. Yang et al. [6,7] detail a two-phase procedure in their research. This procedure is designed to first determine the position of intermediary registers within the scan chain and then apply DSAs to retrieve the round key. On the other hand, signature scan-based attacks require the generation of a set of signatures by subjecting various chosen plaintexts to encryption simulations using a range of potential encryption keys. Kodera et al. [8] describe a procedure that observes changes in the scan data, providing particular plaintexts to form the scan signature matrix and thus reducing the key candidate number from 2^{48} to 512.

In order to secure crypto-chips from scan-based attacks, multiple countermeasures are proposed. Scan-based attack countermeasures 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 [9]. Agrawal et al. [10] proposed an obfuscated scan chain structure that incorporates XOR gates at random points in the scan chain. Atobet et al. [11] proposed the state-dependent scan flip-flop (SDSFF) that replaces scan flip-flops at random points to prevent attackers from identifying the correct scan timing. Lee et al. [12] proposed subchain modification techniques that allow Lock and Key controls and scan order obfuscation to prevent attacks from accessing the scan structure. Gaikwad et al. [13] proposed InvisibleScan architecture, which utilizes FSM obfuscation with access control mechanisms to protect the scan chain by preventing direct access to the SI, SO, and SE logic. The dynamically obfuscated scan chain (DOSC) [14] incorporates a method that integrates the permutation of scan chains with XOR gates and employs logic-locking techniques using dynamic keys. Moreover, the 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 DOSCs, providing a robust defense mechanism against unauthorized access or attacks.

In this paper, we propose DOSCrack, which stands for Deobfuscation using Oracle-guided Symbolic Execution and Clustering of Binary Security Keys. In addition, we also propose a modification to the DOSC that substantially increases the computational effort required to break it. Our contributions are listed below.

- We propose DOSCrack, a novel deobfuscation framework that incorporates structural analysis, symbolic execution, key candidate clustering, and sequential equivalence checking to non-invasively recover the DOSC's seed—the foundation of its security.

- The DOSCrack framework is designed to target the DOSC architecture while being equally effective at breaking static scan chain obfuscation mechanisms by exploiting the scrambled scan chain through symbolic execution.
- We experimentally apply our framework on implementations of the DOSC with different seed bit-lengths. The framework demonstrates scalability, with the recorded runtime exhibiting a proportional increase as the seed size grows.
- We propose a countermeasure against our deobfuscation framework that incorporates a nonlinear shift feedback register (NLSFR) to improve the DOSC's robustness against symbolic execution modeling.
- The NLFSR countermeasure achieves this enhanced security with minimal design overhead, making it a practical solution for industrial DfT/T applications such as the BIST (Built-In Self-Test) architecture.
- We analyze the complexity of NLFSR-based countermeasures by examining the growth in the number of clauses, as well as the time complexity for NLFSR compared to LFSR in the original DOSC architecture.
- Our experiments demonstrate that the NLFSR-based countermeasure effectively defends against the DOSCrack deobfuscation attack when the seed size exceeds 32 bits, with the attack timing out after 10 days.

The rest of the paper is organized as follows. Section 2 gives the necessary background of techniques, as well as the structure of the dynamically obfuscated scan chain. Section 3 introduces the novel DOSCrack, our proposed deobfuscation framework, and describes each step. Section 5 provides a countermeasure and explains how it improves the DOSC. Section 6 analyzes the results of applying our framework to DOSC benchmarks. Finally, Section 7 concludes the paper with key takeaways and future works.

2. Background and Preliminary Concepts

In this section, we will provide a concise overview of the architecture of the DOSC and then introduce the fundamental concepts of symbolic execution and binary clustering. In the end, we give the threat model of our DOSCrack framework, outlining the potential security risks from the attacker's perspective and the assumptions of the DOSCrack attack.

2.1. DOSC Architecture

The DOSC [14] 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. In DOSC, the seed must be kept a secret from the attacker. Otherwise, the attacker can generate test patterns and interpret test responses. If the DOSC is used to protect logic-locked circuits, knowing the DOSC's seed would allow an attacker to perform Boolean satisfiability (SAT) attacks [15] 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 [16]. The DOSC architecture was found to be robust against SAT attacks because such an unrolling inevitably results in scalability issues for Boolean SAT solvers. As a result, the SAT attack targeting the DOSC seed timed out after 20 days.

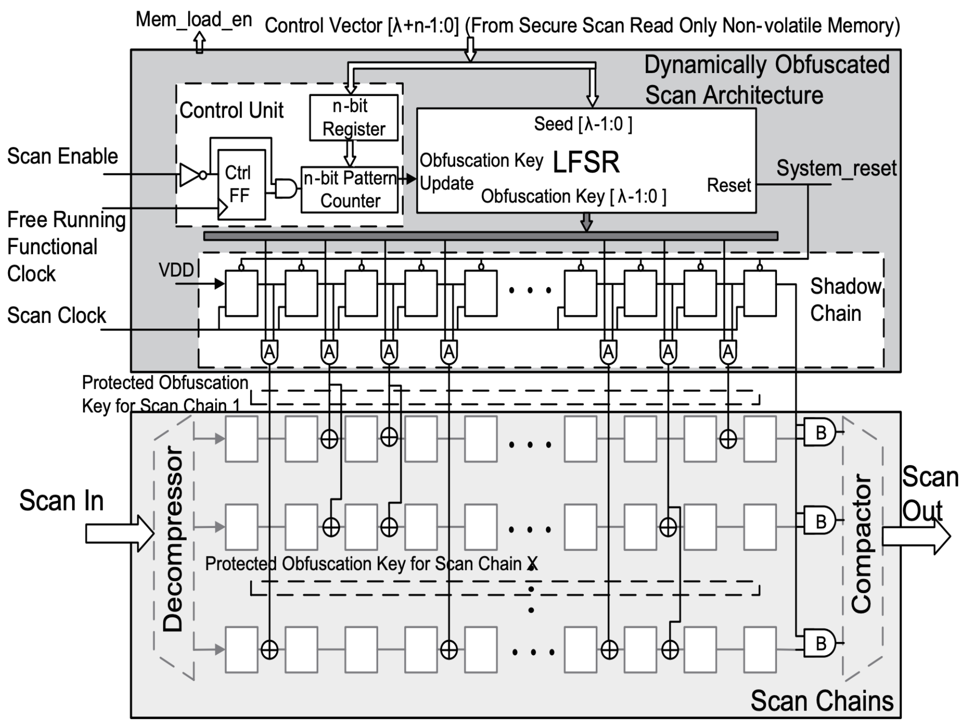


Figure 1. Dynamically obfuscated scan chain architecture [14].

2.2. 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. These symbolic values represent input variables, and the program's execution path is explored symbolically by tracking how these symbolic values change. 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 prior research, Ahmed et al. [17] proposed a framework that combines symbolic execution with simulation data to generate test vectors that activate rarely triggered hardware Trojans. Vafeei et al. [18] proposed another framework that utilizes random simulations in symbolic models to identify critical execution branches for potential hardware Trojans. In our deobfuscation framework, we leverage the built-in symbolic execution engine from EISec [19] to convert our target netlist into C code. Subsequently, the generated C code undergoes symbolic modeling.

2.3. Binary Clustering

Binary clustering refers to grouping binary data into clusters based on pattern similarity or distance measurements such as Hamming distance (HD) or Euclidean distance. Binary data typically imply that the data being clustered consists of only two potential states. Thus, this concept is well suited for hardware applications, as logic gate values are limited to 1s and 0s. In hardware security, binary clustering is often employed to identify patterns in binary vectors that indicate malicious behaviors or hardware Trojans. For example, SCOAP [20] and COTD [21] test malicious signals using clustering analysis based on testability reports. In [20], Zhao et al. introduced a hardware Trojan detection method that relies on clustering gates and registers value distributions using the density-based spatial clustering of applications with noise (DBSCAN) algorithm. He et al. [22] proposed Fuzzy C-Means clustering in conjunction with fusion distance algorithms to detect anomalies indicative of hardware Trojan in integrated circuits. The choice of clustering algorithm depends on the specific characteristics of the data and the goals of the analysis. Commonly used binary clustering algorithms include hierarchical clustering, k-means clustering, and DBSCAN. In this paper, we utilize binary clustering to group similar candidate keys, which allows us to further rule out similar keys (or associated seeds) as a group.

2.4. 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. These contract foundries therefore have full access to the scan chains embedded in the design. This accessibility presents a potential risk as it allows these facilities to conduct scan-based attacks to extract sensitive information or compromise the security of the fabricated semiconductor devices. Additionally, the assembly and test facilities, which are responsible for performing scan testing (i.e., JTAG) on fabricated chips, have access to the knowledge of DfT/D insertion techniques. As assumed in many other logic-locking papers, it is also possible for the design to be reverse-engineered or leaked by untrusted parties. Therefore, end users and debuggers may also have access to this information and scan chains. 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. However, they do not know the value of the LFSR seed. Obtaining the seed is therefore the goal of their attack. 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.

While many scan-based attacks (i.e., differential scan-based attacks) necessitate some degree of understanding of the chip's functional logic to be effective, our method exclusively relies on running the 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. Additionally, DOSCrack performs a non-invasive attack, preserving the integrity of the oracle chip throughout the process. Note that once DOSCrack has recovered the seed, an attacker can proceed with SAT attacks against the functional circuit through the scan chain to recover its logic-locking key.

3. DOSCrack Framework

An overview of DOSCrack, the proposed DOSC deobfuscation framework, is illustrated in Figure 2. The inputs to the framework are the target obfuscated oracle and the netlist of the stand-alone DOSC architecture. Using this information, the framework then generates the minimized obfuscation key candidates set as outputs, which can subsequently be mapped to the corresponding seeds. 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. Our framework is composed of four integral components:

1. We use an unlocked chip (or equivalently simulate an unlocked chip's DOSC and scan chain) to act as an oracle;
2. We employ structural analysis to distinguish between the linear feedback shift register (LFSR) and scan chains in the netlist;
3. We conduct symbolic execution followed by employing an SMT solver to rule out keys and their associated seeds;
4. We utilize clustering algorithms to efficiently categorize the remaining key candidates, find distinguishing patterns, and iteratively rule out more keys/seeds.

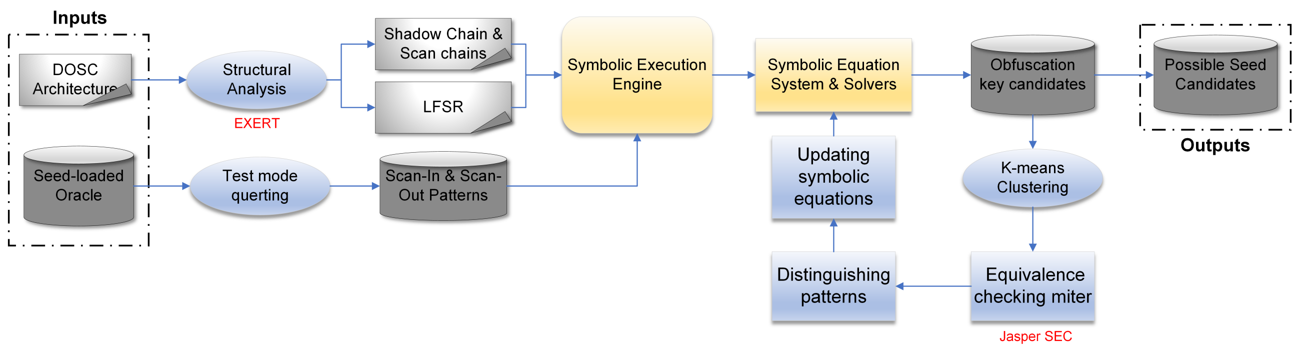

Inputs
DOSC Architecture
Seed-loaded Oracle
Structural Analysis
EXERT
Test mode querting
Shadow Chain & Scan chains
LFSR
Scan-In & Scan-Out Patterns
Symbolic Execution Engine
Symbolic Equation System & Solvers
Updating symbolic equations
Distinguishing patterns
Obfuscation key candidates
K-means Clustering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

Figure 2. Workflow of DOSCrack deobfuscation.

3.1. Structural Analysis

As explained in the threat model (Section 2.4), 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 taking the netlist of the DOSC architecture as input to identify 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 based on current state and input. On the other hand, scan chains are implemented using 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. The FSM structure is suitable for LFSRs because it can efficiently handle the linear feedback mechanism that defines the LFSR's behavior. Therefore, structural analysis distinguishes the LFSR and scan chains on the basis of 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 [23] 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 modulation with symbolic engines.

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
```

3.2. Oracle Interaction in Test Mode

To apply the proposed deobfuscation framework, random input sequences are applied to the target oracle that 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 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 unlocked chip (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 (DOSC).

3.3. 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.

3.3.1. 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). Figure 3 shows an example of a 5-bit LFSR with $x = 0$ and $y = 2$.

We describe the i-th bit output of the LFSR in cycle t as L_t^i. Thus, a general LFSR output at cycle t is represented as:

$$L_t^i = \begin{cases} L_{t-1}^x \oplus L_{t-1}^y, & \text{when } i = 0; \\ L_{t-1}^i, & \text{otherwise} \end{cases} \tag{1}$$

This equation models the LFSR outputs and can be described as follows. The first bit of the LFSR is always connected to the output of the XOR gate. This means that at every clock cycle, the first bit of the LFSR is updated based on the output from the XOR gate, 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 $n-1$ (excluding the first bit which is directly updated from the XOR gate), the shifting process occurs when each bit shifts from its previous state to the next position with each clock cycle.

Table 1 shows the cycled output based on seed $(s_0, s_1, \cdots, s_4)$ for the example in Figure 3. 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 scan in the (SI) port of the scan chain and the obfuscation key input, whereas the outputs are directed to the scan out (SO) port. The symbolic engine then modulates the transformed C code, treating it as a symbolic equation representing the relationship between the inputs and the outputs.

Table 2 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

$$SO_{T+N} = SI_T \bigoplus_{t,i=0}^{N} L_t^{T+i}, \tag{2}$$

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.

Table 1. Table of symbolic representation from seed to obfuscation key for each LFSR bit ($0 \leq i \leq 4$) at clock cycles $1, 2, \ldots, t$ for the example in Figure 3.

Cycle \ i	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	$L_{t-1}^0 \oplus L_{t-1}^2$	L_{t-1}^1	L_{t-1}^2	L_{t-1}^3	L_{t-1}^4

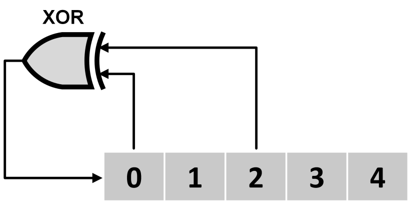


Figure 3. Structure of a 5-bit LFSR.

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 SI and SO patterns, effectively encapsulating the dynamics of how the obfuscation key values are processed within the scan chain. This encapsulation is conducted using our symbolic execution engine. In the conversion process to C code, we have integrated a flag variable, labeled m. This variable increases each time the code representing the scan flip-flop behavior is executed. The primary purpose of this flag variable is to keep track of the location of the scan patterns. 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 scan-out pattern is symbolically represented with a corresponding scan-in pattern with a certain sequence of XOR operation on the obfuscation key.

Table 2. Table of conversion from Netlist to C code and symbolic equation setup.

Netlist	wire shadow_chain/N1, Scan_out;
	SDFFQX_scan_reg[1] (.D(n1),.SI(out[1]),.SE(test_se),
	    .CK(CK),.Q(Scan_out);
	AND2X1 U1(.A(in[0],.B(shadow_chain/N1,.Y(out[1]);

C code	bool shadow_chain/N1, Scan_out;
	out[1] = in[0] & shadow_chain/N1;
	if(test_se == 0) {Scan_out = n1;}
	    else {Scan_out = out[1]; m+=1;}

Symbolic equation	Scan_out == shadow_chain/N1 & in[0]

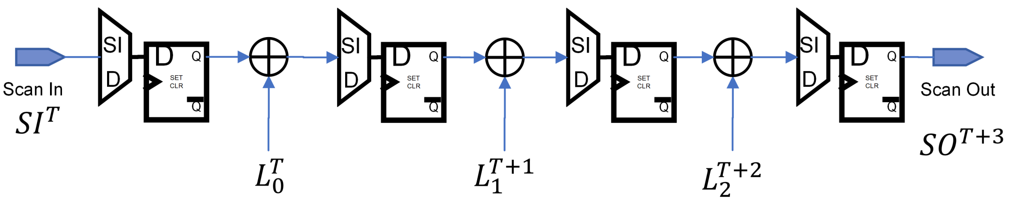


Figure 4. Example of scan chain modeling with $N = 3$ at any clock cycle T. The scan out is denoted as $SO^{T+3} = SI^T \oplus L_0^T \oplus L_1^{T+1} \oplus L_2^{T+2}$.

3.3.2. Symbolic Equation System and Solver

In the previous section, the obfuscation keys are symbolically represented in Table 1, and the scan chains are modeled with symbolic equations in Table 2 in our running example. The two symbolic equations then form simultaneous equations f and g:

$$\vec{L}_t = f[(s_0, s_1, \cdots, s_{n-1}), t]; \tag{3}$$

$$SO_{T+N} = g[SI_T, (L_0^T, L_1^T, \cdots, L_t^{T+N-1})] \tag{4}$$

As discussed in Section 3.2, the SI and SO patterns are generated from test mode interactions within the oracle. Upon providing a sequence of m-bit SI patterns, we collect the corresponding m-bit SO patterns from the oracle. By putting m pairs of these corresponding SI and SO patterns into the simultaneous equations, we construct a symbolic equation system that encompasses m equations, each representing their relationship and transformation. As a result, in the symbolic equation system that we have 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.

Note that as our SI and SO patterns are generated randomly, there is no assurance that all symbolic equations in the above system are independent. This issue might result in a large number of potential solutions provided by the SMT solver. Despite this, these initial results serve as a starting point for applying the clustering algorithm to explore similarities in the solution space and utilize sequential equivalence checking (SEC) to produce distinct SI and SO patterns. Future work can focus on more careful selection of SI patterns.

3.4. 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. The symbolic equation is built on SO and SI patterns formulated by functions f and g that are produced by the symbolic execution engine. As the functions f and g are fixed based on the inherent nature of the circuit implementation, more SO and SI patterns need to be obtained by further queries to the oracle in order to produce more symbolic equations. However, the effectiveness of generating SO and SI patterns from random simulations diminishes over time. This diminishing effect occurs because the candidates for the 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 designed to categorize existing candidates for obfuscation keys into distinct groups. Following this categorization, we construct a miter circuit specifically for SEC, which generates distinguishing patterns that are tailored to each group of key candidates.

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 applied the K-means clustering algorithm to categorize the space of key candidates. We utilized the Hamming distance (HD) metric for the K-means clustering algorithm, as it is the most suitable for binary data. As the obfuscation key dynamically changes over time, we select and group the first four cycles of the obfuscation key that produce the clustering data, as shown in the first step in Figure 5. From two distinct clusters, we select centroid keys and map them back to their corresponding symbolic seeds. The two seeds are subsequently loaded to two DOSC architectures and compared by the miter circuit. The miter is constructed with an XOR gate, which compares two copies of the DOSC architecture loaded with different seeds. This setup, as shown in the second step of our process, forms the basis for further analysis. Next, the miter circuit feeds to the Jasper SEC engine [24] that generates the input sequence that can differentiate between the two DOSC benchmarks. We then 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.

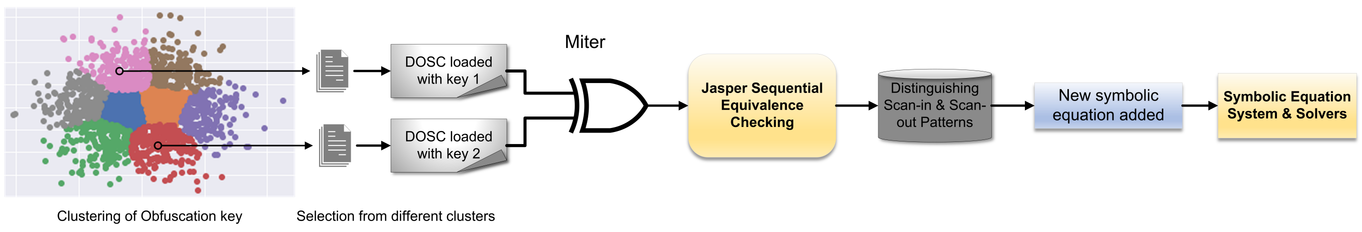

DOSC loaded with key 1
DOSC loaded with key 2
Miter
Jasper Sequential Equivalence Checking
Distinguishing Scan-in & Scan-out Patterns
New symbolic equation added
Symbolic Equation System & Solvers
Clustering of Obfuscation key
Selection from different clusters

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

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 same cluster. Thus, this method efficiently solves the issue of diminishing returns in random simulation, ensuring a more effective elimination of potential key candidates. 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.

The re-clustering is needed as the process of ruling out obfuscation keys progresses, and the distinct margins of the clusters become less defined or maintained as more keys/seeds get ruled out. Figure 6 shows the flowchart of the specific conditions under which re-clustering should be invoked. Moreover, it outlines the decision points for determining when to abandon the clustering approach and switch to ruling out keys by brute force. During the iterative process of selecting clusters for generating distinguishing patterns using Jasper SEC, we incorporate two critical checkpoints to assess the *silhouette score* (the silhouette score is a common metric for evaluating the quality of clustering results. It operates by measuring how similar objects are to others in their own cluster compared to other clusters. The silhouette score ranges from -1 to +1, where a high value indicates that the clustering parameters (number of clusters, among others) are appropriate). The first check occurs right after the completion of the K-means clustering. At this point, we evaluate the silhouette score to gauge the effectiveness of the clustering, ensuring that the clusters formed are distinct and meaningful. If the silhouette score is below our threshold score of 0.6, we take this to mean that the clustering is not successful and we should switch to rule out keys by brute force. The second check occurs during the iterations. This occurs after a certain number of keys have been eliminated. At this point, we calculate the silhouette score, taking into account both the reduced key sets under the initial cluster formations. If the silhouette score remains above our set threshold, it conveys that the existing clustering maintains distinct margins and indicates that the generation of distinguishing patterns is still efficient. Then, another iteration of selecting clusters is performed to feed into the miter to generate distinguishing patterns. On the other hand, if the silhouette score drops below our threshold, it suggests a deterioration in the clustering structure, and re-clustering is needed to re-establish distinct grouping. By inserting two checkpoints of the silhouette score to evaluate the clustering margins, we ensure a dynamic and responsive approach to generate effective and distinct patterns in ruling out groups of keys in each iteration.

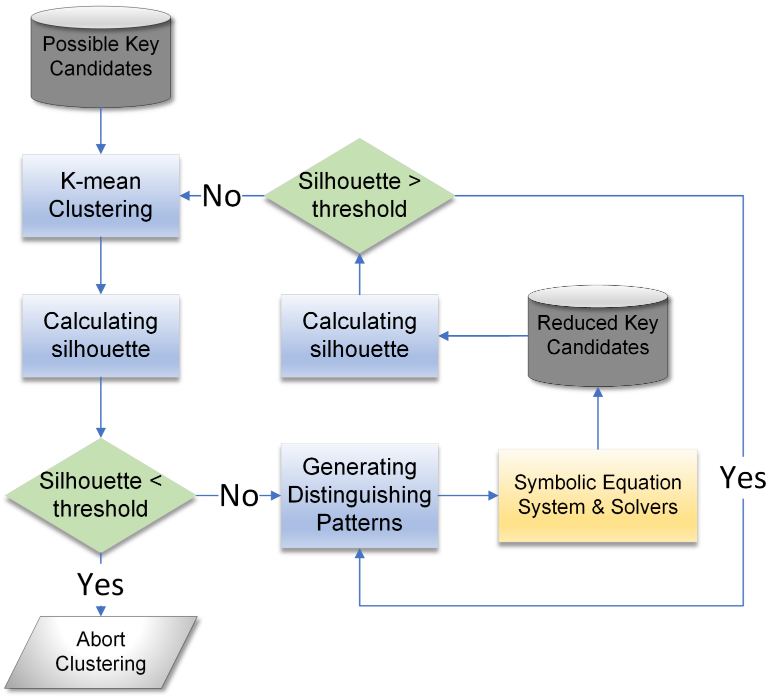


Figure 6. Flowchart with checkpoints that evaluate silhouette scores. Two silhouette score thresholds determine when to abort clustering and when to re-perform clustering.

4. Exploiting Alternative Scan Chain Obfuscation Techniques with DOSCrack

In addition to targeting dynamically obfuscated scan chains, DOSCrack also serves as an effective method for breaking existing static scan chain obfuscation frameworks such as SeqL [25] and XOR-obscuring scan chains [10]. Both static obfuscation methods employ MUX or XOR gates inserted within the scan chain to scramble its architecture, as illustrated in Figure 7. Static scan chain obfuscation relies on the assumptions of security that the keys and the locations of the scrambling gates remain undiscoverable by attackers. DOSCrack leverages symbolic execution as a powerful reverse engineering technique to explore the locations of scrambling gates, making it an exceptionally effective method for breaking static obfuscation. Once the scan chain architecture is modeled using DOSCrack symbolic execution engine, the XOR scrambling mechanism is effectively breached. For MUX-based scrambled scan chains, the select bits of the MUX are treated as the scrambling key inputs (sck0 and sck1). Breaking this type of scan chain involves determining the correct scrambling keys, which can be attacked by DOSCrack by constructing symbolic equations and solving them to retrieve the correct key values. An example of exploiting the scan chain architecture of the static obfuscation techniques is shown in Table 3.

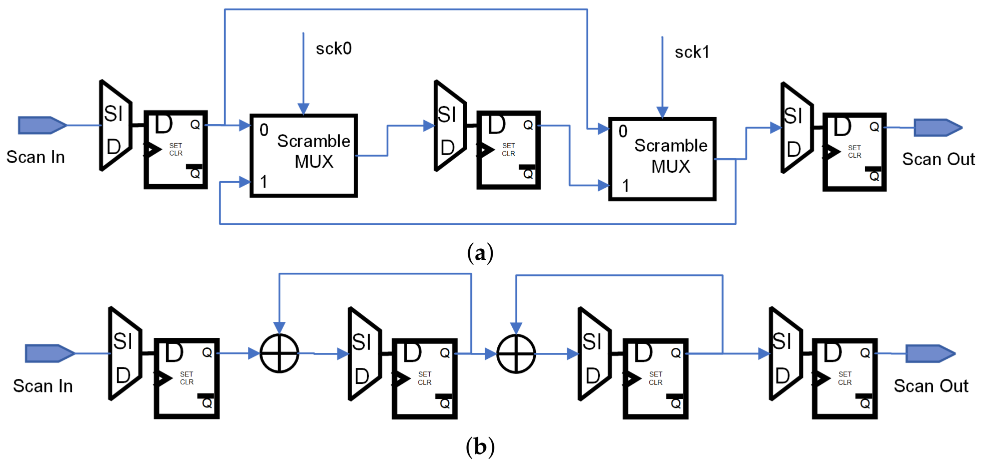


Figure 7. Example of static obfuscation scan chain based on scrambling. (**a**) Scrambling of scan chain with MUX and (**b**) scrambling with XOR gate.

Recent research has further advanced the concept of the DOSC architecture by proposing enhanced designs that incorporate additional layers of security. Lee et al. [26] proposed a scan chain architecture that integrates dynamic keys with PUF keys, along with bypass mechanisms and fake response generators, creating a more robust and secure scan chain architecture. DOSCrack also has the potential to compromise this enhanced scan chain architecture. If the PUF key-based authentication is successfully bypassed using a developed method, the dynamic key generation mechanism can be attacked independently with DOSCrack.

Table 3. Table of conversion from Netlist to C code for static scan chain obfuscation. (Top) SeqL scrambling with MUX, and (Bottom) XOR-chain scrambling with XOR. Scb1 denotes the scrambling nets that feed backwards to the MUX or XOR inputs.

Netlist	wire sck[0], Scb1;
	SDFFQX_scan_reg[1] (.D(n1),.SI(Y[1]), .SE(test_se),.CK(CK),.Q(out[1]);
	MX2X1 U4(.A(out[0],.B(Scb1),.S0(sck[0]),.Y(Y[1]);

C code	bool sck[0], Scb1;
	if(test_se == 0) {out[1] = n1;}
	    else {out[1] = Y[1];};
	if(sck[0] == 0) {Y[1] = out[0];}
	    else {Y[1] = Scb1;}

Netlist	wire Scb1;
	SDFFQX_scan_reg[1] (.D(n1),.SI(Y[1]),.SE(test_se),.CK(CK),.Q(out[1]);
	XOR2X1 U2(.A(out[0],.B(Scb1),.Y(Y[1]);

C code	bool Scb1;
	Y[1] = out[0] ˆ Scb1;
	if(test_se == 0) {out[1] = n1;}
	    else {out[1] = Y[1];}

5. Incorporation of NLFSR to Increase DOSCrack Complexity and Effort

In this section, we present a potential design countermeasure to our DOSCrack attack. This countermeasure aims to increase the complexity of symbolic execution modeling thus extending the runtime of the DOSCrack attack. Given the inherent design of scan chains, which are composed of cascading scan flip-flops, altering their structural layout to enhance robustness is not a feasible approach. Therefore, we turn to modifying the LFSR to improve its robustness against symbolic execution modeling by incorporating the nonlinear feedback shift register (NLFSR) into the DOSC architecture. The Trivium cipher is well regarded for its efficiency in generating large sequences of keys. In our application, it enables the generation of key streams extending to less than 2^{64} cycles. This feature of Trivium makes it an ideal candidate for our purpose, as it significantly enhances the complexity of the key generation process.

5.1. Trivium-Based DOSC

The Trivium random generation algorithm [27] is shown in Algorithm 2. The algorithm is initialized by loading an 80-bit key and an 80-bit initialization vector (IV) into the 288-bit initial state and setting all remaining bits to 0 (line Input:). The algorithm generates a random bit z_i which can be configured from 1-bit to 64-bit (line Output:). The t_1, t_2, and t_3 are intermediate Boolean variables that perform the nonlinear operations (line 6 to 8).

In our application, to integrate the Trivium cipher algorithm effectively within the DOSC framework, we adopt both the 80-bit key and the 80-bit IV as the seed. The significant length and complexity of this 160-bit seed, combined with the nonlinear properties of NLFSR, render the DOSC system highly resilient against symbolic execution modeling. This approach results in a 160-bit seed for the DOSC equipped with NLFSR. As an NLFSR, Trivium has been a particularly attractive target for algebraic attacks [28]. In LFSR-based systems, the state evolves linearly, allowing for the application of efficient linearization techniques to solve the corresponding systems of equations. However, for NLSFRs, these linearization techniques become less effective. As a result, the scalability of solving the systems of equations increases. To compare the effort of DOSCrack in breaking the DOSC with LFSR vs. NLFSR, we build different sizes of Trivium according to its original 288-bit architecture, where Algorithm 2 shows an example of the 8-bit Trivium architecture.

Algorithm 2 Trivium random generation [27]

```
288-bit seed Trivium:
Input: Initialization Vector: $(s_1, \cdots, s_{288})$
Output: Random bit: $z_i$
 1: for $i = 1$ to N do
 2:     $t_1 \leftarrow s_{66} + s_{93}$
 3:     $t_2 \leftarrow s_{162} + s_{177}$
 4:     $t_3 \leftarrow s_{243} + s_{288}$
 5:     $z_i \leftarrow t_1 + t_2 + t_3$
 6:     $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
 7:     $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
 8:     $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
 9:     $(s_0, s_1, \cdots, s_{93}) \leftarrow (t_3, s_1, \cdots, s_{92})$
10:     $(s_{94}, s_{95}, \cdots, s_{177}) \leftarrow (t_1, s_{94}, \cdots, s_{176})$
11:     $(s_{178}, s_{279}, \cdots, s_{288}) \leftarrow (t_2, s_{178}, \cdots, s_{287})$
12: end for
8-bit seed Trivium:
Input: Initialization Vector: $(s_0, s_1, s_3, s_4, s_5, s_6, s_7, s_8)$
Output: Random bit: $z_i$
 1: for $i = 1$ to N do
 2:     $t_1 \leftarrow s_3 + s_4$
 3:     $t_2 \leftarrow s_7 + s_8$
 4:     $z_i \leftarrow t_1 + t_2$
 5:     $t_1 \leftarrow t_1 + s_2 \cdot s_3 + s_6$
 6:     $t_2 \leftarrow t_2 + s_6 \cdot s_7 + s_2$
 7:     $(s_0, s_1, s_3, s_4) \leftarrow (t_2, s_1, s_2, s_3)$
 8:     $(s_5, s_6, s_7, s_8) \leftarrow (t_1, s_5, s_6, s_7)$
 9: end for
```

5.2. Complexity of LFSR vs. Trivium

When comparing the time complexity of solving LFSR and Trivium as ciphers, LFSRs are vulnerable due to their linear nature, making them susceptible to attacks like the Berlekamp–Massey algorithm [29], which can efficiently reconstruct the internal state with a time complexity of $O(n^2)$. This makes LFSRs easier to solve in polynomial time given a sufficient sequence of output bits. In contrast, Trivium is designed with a large 288-bit state and nonlinear feedback taps, making it highly resistant to linear cryptanalysis and significantly harder to solve. Attacks on Trivium often involve algebraic or guess-and-determine methods, which are computationally expensive due to the nonlinearity, typically resulting in exponential time complexity.

5.2.1. Complexity of LFSR and Trivium Ciphers

Algebraic attacks aim to exploit the algebraic structure of a cipher to recover the secret key or internal state. The DOSC architecture can be treated as a logic-locking mechanism of the scan chain using obfuscation keys generated from LFSR and Trivium ciphers. Thus, analyzing the complexity of using algebraic attacks to recover the seed from LFSR and Trivium ciphers is crucial when evaluating the effectiveness of our NLFSR countermeasure in the DOSC architecture.

Table 4 summarizes the comparisons in complexity from previous research. For LFSR-based ciphers, the attacker can generate a system of n linear equations. Solving this system using Gaussian elimination requires $O(n^3)$ time complexity, where n is the length of the output sequence [29]. In some optimized cases, this can be reduced to $O(n^2)$ time complexity using sparse matrix solvers [30]. The space complexity is $O(n^2)$ for storing the system of equations [31].

Table 4. Comparison of attack complexity on LFSR-based and Trivium ciphers.

Cipher Type	Time Complexity of Attack	Space Complexity of Attack
LFSR Cipher	$O(n^2)$ to $O(n^3)$	$O(n^2)$ for storing the linear system [29]
Trivium Cipher	$O(2^{80})$ to $O(2^{115})$	$O(2^n)$ for storing the nonlinear system [31]

Unlike an LFSR, the Trivium cipher incorporates nonlinear feedback taps between three shift registers set, which significantly complicates algebraic attacks. The equations of Trivium are quadratic (due to the nonlinear feedback) rather than linear and the system of quadratic equations can be solved using the Gröbner basis approach [32]. Solving the system has time complexity of $O(2^k)$, where k is the number of internal state bits. For Trivium, this typically results in a complexity of $O(2^{80})$ to $O(2^{115})$ [31]. The space complexity for Trivium-based attacks can be as large as $O(2^{288})$, due to its 288-bit internal state.

5.2.2. SAT Solver Complexity

Both algebraic attacks and our symbolic execution framework utilize SAT solvers to find the solution of the equation system. Solving of an equation system is mostly referred to as the MP (Multivariate polynomial) problem, with a set of functions:

$$f_i(x_1, \ldots, x_n) = y_i, 1 \leq i \leq m \tag{5}$$

where each f_i is a multi-degree polynomial. We analyze the complexity difference between LFSR and Trivium by examining their conversion from the MP problem to an SAT problem [33]. The complexity can be quantified by the number of clauses in the conjunctive normal form (CNF) representation and the SAT solver's runtime. A linear equation with α variables (monomials) converts to a CNF containing $2^{(\alpha-1)}$ clauses. In a nonlinear equation, each unique quadratic monomial introduces one new variable. This substitution creates three extra clauses in the CNF. In general, an equation with α variables and β monomials (γ of which are quadratic) converts to a CNF with $2^{(\beta-1)} + 3 \cdot \gamma$ clauses. The total number of clauses in the CNF is determined by [34]:

- Number of variables.
- Density of equations.
- Number of unique quadratic monomials.

Thus, for an n-bit LFSR size, the number of clauses of an n-bit LFSR is $(2^{(n-1)} + 3)$, given that the LFSR only contains a single feedback tap that creates a quadratic. In addition, the number of clauses also depends on the roll-up of internal nodes. For a scan chain with length N, the unrolling of the scan chain introduces $(2N-1)$ times clauses. The total number of clauses for the LFSR is therefore $(2^{(n-1)} + 3)(2N-1)$, as shown in Table 5. However, given the nonlinear component from Trivium Algorithm 2, the z_i consists of three nonlinear XOR operations between different shift registers (line 6 to 8), which makes every monomial in Trivium contain a quadratic component. The number of clauses of Trivium results in a total number of $(3 \cdot 2^{(n-1)} + 3n + 9)(2N-1)$, as shown in Table 5.

Table 5. SAT solver mathematical complexity for LFSR and Trivium. n denotes size of seed and N denotes length of scan chain.

Solver	Number of Clauses
LFSR	$(2^{(n-1)} + 3)(2N-1)$
Trivium	$(3 \cdot 2^{(n-1)} + 3n + 9)(2N-1)$

The significant difference in the number of clauses between LFSR and Trivium during CNF conversion (approximately three times) can still result in exponential growth in SAT solver runtime.This is because the complexity of SAT solvers scales exponentially with the presence of XOR constraints in random CNF-XOR formulas [35]. The number of clauses alone is only a mathematical complexity variable but not an accurate predictor of whether a SAT solver will be able to solve a problem in reasonable time. The structure and nature of the constraints, especially the presence of XOR clauses [36], play a more significant role in determining the performance of solvers. The runtime comparison based on two open source SAT solvers, CryptoMiniSAT and Grasp, is further analyzed in Section 6. We selected the CyptoMiniSAT solver as it is a state-of-the-art solver designed to efficiently handle large problems in determining the satisfiability of Boolean formulas and has been used in multiple algebraic attacks. GRASP (Generic seaRch Algorithm for the Satisfiability Problem) is another SAT solver that employs a systematic search technique based on conflict analysis to determine the satisfiability of Boolean formulas.

6. Experimental Results and Evaluation

In the section, we first 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, which allows us to synthesize the scan chains only by keeping 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 the scan cells is identical to its seed size for every benchmark. In our approach, the seed values are synthesized as static inputs, and 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.

The overview of our synthesized DOSC is shown in Figure 8. The test mode simulation is performed by Synopsys VCS with 500 clock cycles of random SI patterns. The benchmarks are synthesized with Synopsys Design Compiler. We used an Intel(R) Xeon E5-2450L 32-core CPU with 128 GB memory operating at 1.80 GHz for the synthesis of DOSC benchmarks, test mode simulation, and running of DOSCrack. The runtime results are shown in Table 6 and illustrated in Figure 9. We discuss the DOSCrack results on 8-bit, 16-bit, 32-bit, and 64-bit seeds for the LFSR in Sections 6.1–6.4, while the results for the DOSC containing the NLFSR are discussed in Section 6.6. Section 6.7 discusses the overhead of our countermeasure under different assumptions.

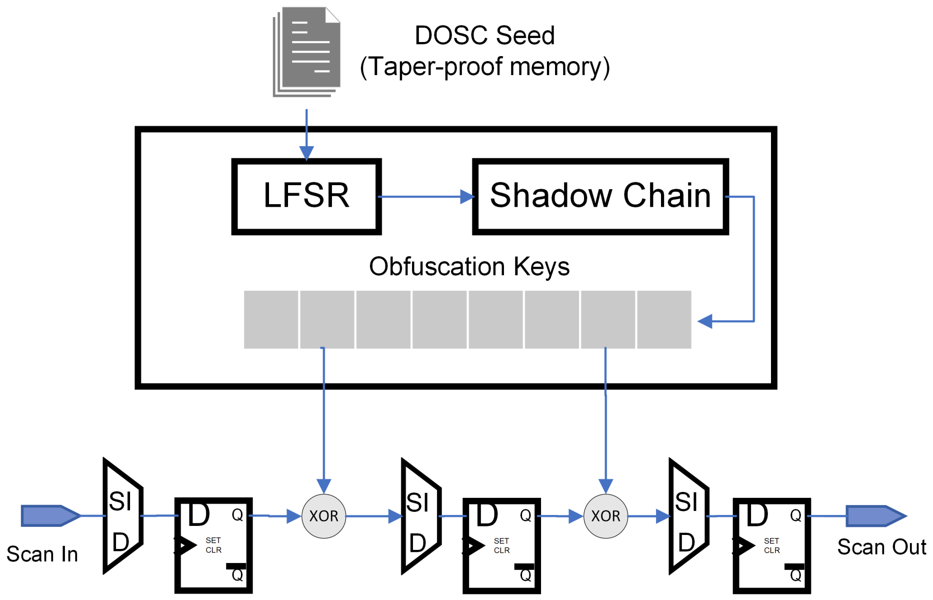


Figure 8. Synthesized version of DOSCrack netlist used in our experiments.

Table 6. Results of DOSCrack with different seed sizes. × means not applicable. In the total runtime row, 'm', 'h', and 'd' denote minutes, hours, and days, respectively.

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	50,288	$\leq 2^{17}$
# of Clusters	×	14	107	2440
# of Seed Candidates	3	23	327	6508
Total Runtime	10.2 m	37.9 m	858 m	3 d 23 h

6.1. 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 to maintain consistency across the system. Given the modest 8-bit size of the seed, our symbolic execution engine operates efficiently and the SMT solver generates just three potential key candidates. Consequently, there is no requirement to employ clustering algorithms to manage the obfuscation key candidates' space. The total runtime is 18.5 min but this includes simulation time for SI/SO patterns which is 8.3 min. Since querying an oracle chip would be much faster than simulations, the real runtime for symbolic execution and SMT solvers is only 10.2 min. The effective handling of an 8-bit seed by our symbolic execution engine results in only three possible key candidates from the SMT solver, which serves as evidence of the scalability of our stand-alone symbolic execution modeling approach. The break time for a traditional SAT attack on the 8-bit DOSC benchmark exceeds 5 h, demonstrating a significant 30-fold reduction in runtime compared to previous approaches. For the 8-bit DOSC, there is no need for clustering because there are only three remaining seeds. These were trivially brute forced to prune DOSC's actual LFSR seed.

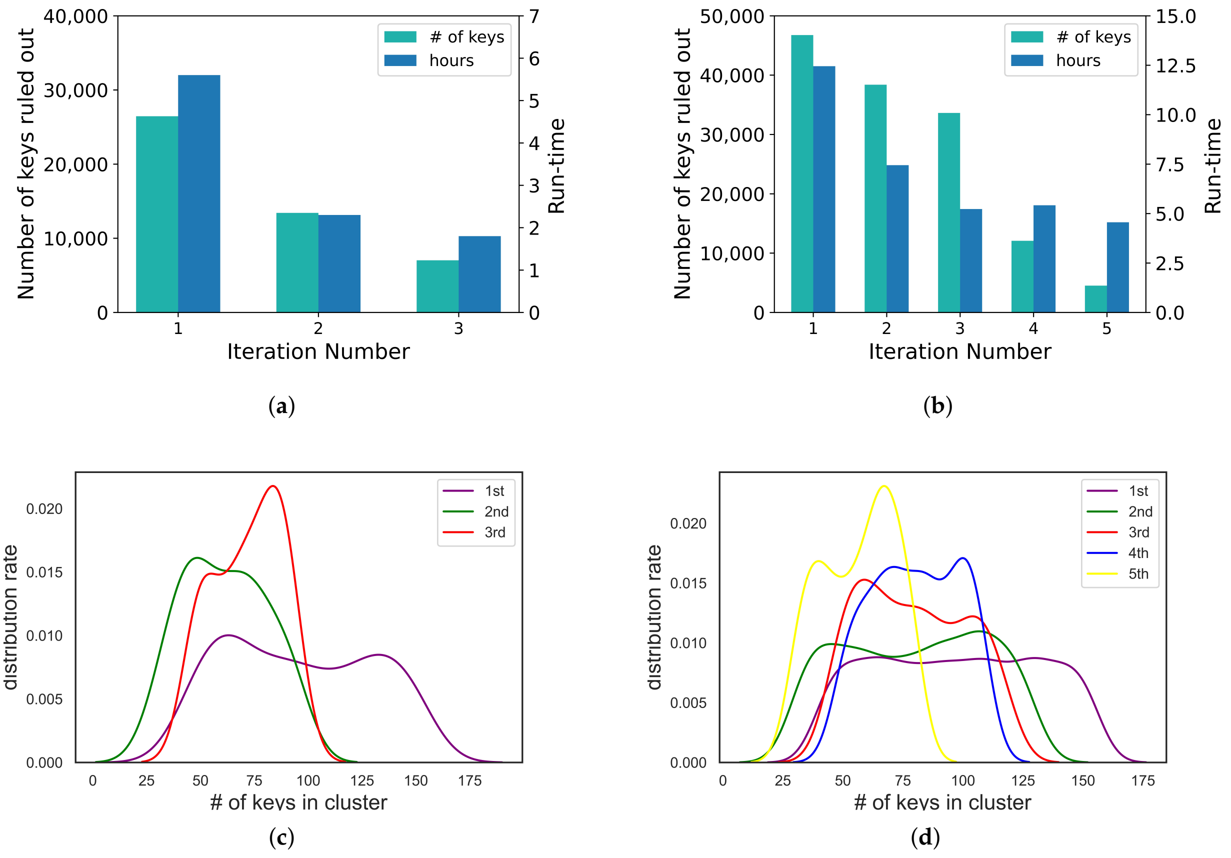


Figure 9. Number of keys ruled out and runtime bar charts for (**a**) 32-bit DOSC and (**b**) 64-bit DOSC per iteration of clustering. Clustering histograms (**c**) 32-bit DOSC and (**d**) 64-bit DOSC for each iteration of clustering. With each successive re-clustering iteration in our process, the distribution of keys within clusters becomes increasingly narrow.

6.2. DOSC with 16-Bit Seed

We further tested a DOSC benchmark configured with a 16-bit seed under the same configuration of 16-bit LFSR and sixteen scan cells. With the same number of simulation SI/SO patterns, our SMT solver has generated 356 potential key (seed) candidates. This quantity is suitable for performing clustering analysis. The initial clustering results identify 14 clusters with a silhouette score of 0.68, indicating a successful clustering outcome. By carefully choosing combinations of clusters and inputting their centroid keys into the Jasper SEC system, we were able to successfully generate 91 distinguishing patterns. The addition of these new equations significantly enhanced the efficacy of our analysis. As a result of this strategic approach, each newly generated equation proved effective in ruling out approximately four potential key candidates. The possible obfuscation key candidates are reduced to 23 without re-clustering. The corresponding seed candidates then are ruled out by brute force and the total runtime is 37.9 min.

6.3. DOSC with 32-Bit Seed

Under the same configuration of DOSC, we perform the deobfuscation on a 32-bit seed benchmark. Unlike the previous test, we increase the number of initial SI/SO patterns generated from simulation as we aim to explore the diminishing effect issues explained in Section 3.4. 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. As a result, the SMT solver produced 50,288 solutions with 500 patterns and 50,354 solutions with 400 patterns. The observation here is that the increase of 100 patterns, which is a 25% increase in simulation patterns, only led to a reduction of 66 in the number of possible key candidates. This result validates the issue of diminishing effect and the necessity of utilizing clustering and SEC miter circuits to produce distinguishing patterns. With clustering, the possible obfuscation key candidates were reduced from 50,288 to 327, with a successful clustering of 107 groups. Figure 10 shows the HD distribution between clusters in three re-clustering iterations. As keys are selectively ruled out in the clustering process, there is a noticeable linear decrease in the HD between clusters. This trend indicates that, over time, the remaining keys within each cluster become more similar to each other.

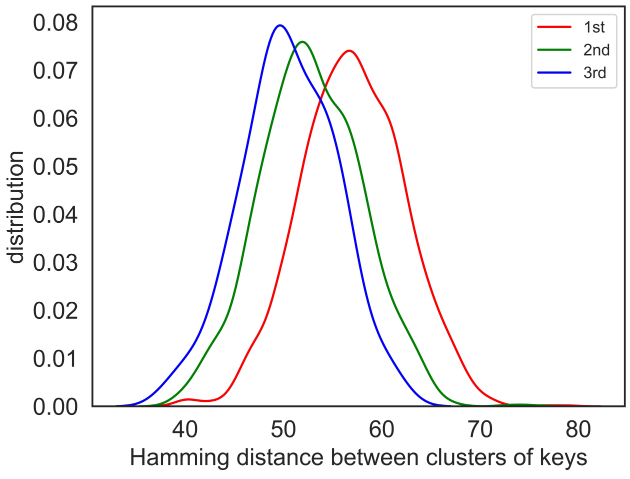


Figure 10. Hamming distance distribution of three consecutive re-clustering processes. The Hamming distance between cluster are reduced after ruling out keys.

6.4. DOSC with 64-Bit Seed

The biggest benchmark we tested uses a 64-bit seed with DOSC. We have decided to maintain the number of SI/SO patterns obtained from the simulation at 500 as we presume the diminishing issue will occur that increasing the number of patterns further would not contribute to our symbolic modeling. The initial outcome from the SMT solver reveals approximately 150,000 obfuscation key candidates, which equates to around 2^{17} solutions. Clustering results in 2440 groups and we go through iterations of clustering whenever the silhouette score is below 0.6. Figure 9b,d show the runtime results as a bar chart and the clustering distribution in a histogram, respectively. The 64-bit DOSC takes five iterations of clustering and the number of the key distribution becomes more narrow through each iteration. The final phase of our analysis involved pruning out the remaining 6508 key candidates using a brute-force approach. The average time taken to brute-force test 1000 random keys in our analysis, often referred to as the 'pruning time', is approximately 2.5 h. The entire attack takes almost 4 days to find the seed.

6.5. Comparisons to Other Attacks

While DOSC architectures have proven to be robust against traditional SAT attacks, they time out after 20 days. State-of-the-art attacks on logic-locked scan chains with dynamic keys rely on customized SAT-based techniques [37,38] designed to target scan chains. In these works, the attack framework was tested on various benchmarks and demonstrated that the seed can be successfully recovered in an hour. However, their attack assumes that the functional logic is not logic-locked, thereby allowing access to the primary I/O pairs needed to generate DIPs [39]. This contrasts with the original threat model of the DOSC [16], where the functional logic should be logic-locked and inaccessible from being exploited to attack the scan chain. In contrast, our DOSCrack framework adopts a stricter and more realistic threat model, focusing exclusively on performing attacks in scan mode. By performing an oracle-based attack that operates solely within the scan chain, DOSCrack eliminates the dependency on functional logic access.

6.6. Trivium-Based DOSC Architecture

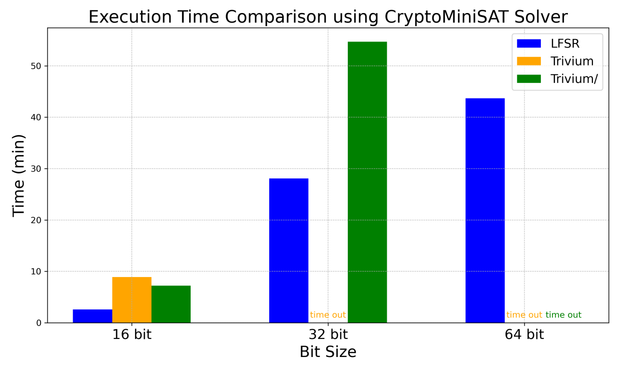
To assess the effectiveness of our proposed NLFSR-based countermeasures, we extend our evaluation to include applying the DOSCrack framework to different bit sizes of the Trivium-based DOSC architecture. We build Trivium-based DOSC benchmarks with different bit-lengths based on Algorithm 2, which shows an example of our 8-bit Trivium and the original 288-bit Trivium architecture. In our evaluation, it was found that the total time taken to crack the 8-bit Trivium-based DOSC using the DOSCrack framework amounted to 52.8 min. Compared to the runtime observed with a system utilizing an LFSR, the 8-bit Trivium-based DOSC exhibits approximately five times the complexity in terms of runtime. The increased complexity observed in the 8-bit Trivium-based DOSC can be attributed to the complexity analysis in Section 5.

In the case of the 288-bit Trivium architecture, our approach involves maintaining its full 288-bit state while specifically loading an 8-bit seed into the system. The remaining bits of the state, which are not occupied by the 8-bit seed, are set to 0. The execution time of solving the symbolic equation system using CryptoMiniSAT times out, even when maximized to 6 days, similar to the other benchmarks shown in Figure 11. The comparison of DOSC benchmarks in the aspect of area and power consumption is shown in Table 7. In the case of symbolic modeling, the NLFSR introduces a higher degree of nonlinearity and complexity in the relationships between the key bits and the output.

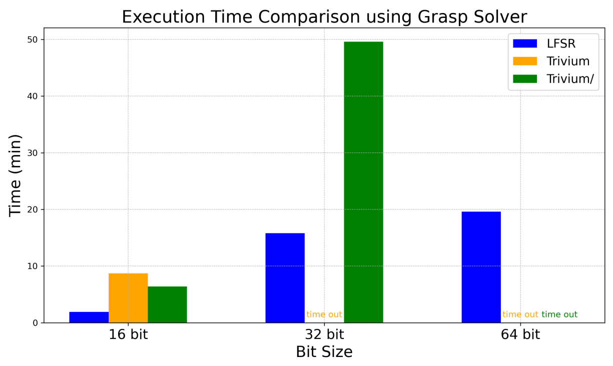
The mathematical and runtime complexity comparisons between the LFSR- and Trivium-based DOSC of the other size benchmarks are shown in Figure 11. The number of clauses generated by both the LFSR and Trivium methods shows an exponential increase with bit-length. However, our symbolic execution approach reduces the mathematical complexity by approximately 75%. The runtime results also demonstrate the effectiveness of the countermeasures when using Trivium for the DOSC. Both SAT solvers time out after 24 h when attempting traditional SAT-based attacks on 64-bit benchmarks. In contrast, DOSCrack, enhanced by our symbolic execution, achieves a great reduced runtime by completing in 49.6 min with the CryptoMiniSAT solver and 54.7 min with the Grasp solver. It is important to note that this runtime reflects only the time required to solve the symbolic equation system once and derive the candidate key space initially. To fully execute DOSCrack, multiple iterations are needed to generate additional symbolic equations, which will further increase the runtime.

Table 7. Benchmark comparison of LFSR-based DOSC and Trivium-based DOSC.

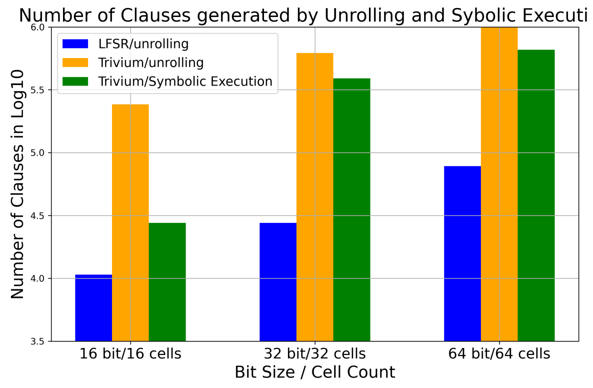
DOSC Benchmarks	8-Bit LFSR	8-Bit Trivium	288-Bit Trivium
# of gates	288	298	597
Total Area (μm^2)	8057	8630	15,729
Total Power (μW)	3.17×10^3	3.17×10^3	5.06×10^3



(**a**)



(**b**)



(**c**)

Figure 11. Mathematical complexity and attack runtime comparisons between LFSR and Trivium: (**a**) runtime comparison of CryptoMiniSAT on unrolling the LFSR (blue), unrolling the Trivium (orange), and symbolic execution of Trivium (green); (**b**) runtime comparison of Grasp solver on unrolling the LFSR (blue), unrolling the Trivium (yellow), and symbolic execution of Trivium (green); (**c**) number of clauses on a logarithmic scale generated from unrolling the LFSR (blue), unrolling the Trivium (yellow), and symbolic execution of Trivium (green).

6.7. Combining DOSC and BIST

Figure 12 illustrates an architecture that integrates Built-In Self-Test (BIST) with DOSC functionality. In industry DfT application, BIST and scan chain insertion are typically implemented concurrently. Since BIST requires a test pattern generator to apply input stimuli and a response analyzer to compare the outputs, we can optimize the design by sharing the NLSFR between both the test pattern generator and the key generator for DOSC obfuscation (the seeds used by the BIST and DOSC should probably be different for security purposes). Table 8 provides the area overhead associated with our integrated design. The results show that incorporating the DOSC architecture into the BIST results in only a 1.1% increase in area overhead, demonstrating the efficiency of our approach.

Table 8. Area increase comparison for s38576 alone, with BIST, and with BIST and scan obfuscation.

Benchmark	**Area/μm^2**	**Increase (%)**
s38576 stand alone	408,055.99	-
s38576 with BIST	429,730.68	6.2%
With BIST & scan obfuscation	434,645.79	1.1%

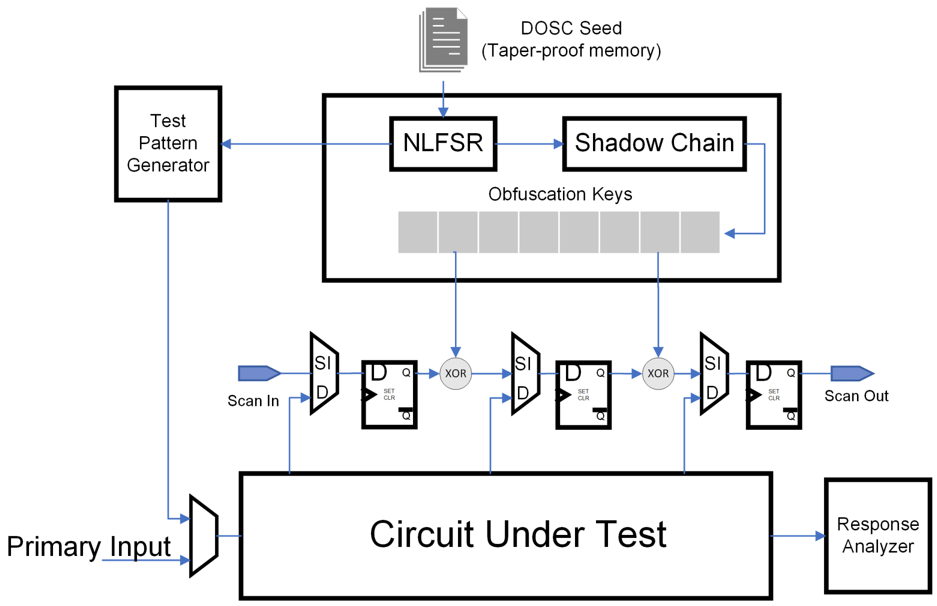


Figure 12. Experiment setup of DOSCrack.

7. Summary and Future Work

In this paper, we present DOSCrack [40], an oracle-guided attack that utilizes symbolic execution and binary clustering to break the DOSC. By applying DOSCrack on different sizes of benchmarks, our framework successfully reduced the number of key candidates and broke the 64-bit seed DOSC in 3 d 23 h, which is much more efficient compared to the time-out threshold of 10 days for traditional SAT attacks. Additionally, in our testing, the brute-force effort to rule out 1000 keys took approximately 2.5 h. This duration highlights the significantly greater complexity of the brute-force method compared to our approach. To evaluate our framework from the designers' perspective, we proposed countermeasures to symbolic execution modeling by introducing the NLFSR trivium cipher as a random bit generator. This innovative approach significantly improves the security of the DOSC. Our Trivium-based DOSC, as a result of this integration, demonstrates higher complexity compared to a DOSC with a traditional LFSR.

Future work in this area could concentrate on developing more advanced algorithms for intelligently selecting different clusters to generate distinguishing patterns. Currently, our approach relies on iterating through various combinations of keys from different clusters to create these patterns. By enhancing the method of choosing clusters, we could significantly improve the distinctiveness of the patterns generated, thereby more effectively ruling out obfuscation keys. One promising direction for this enhancement is the development of a reinforcement learning algorithm. Such an algorithm would learn and adapt over time, based on the success of previous pattern generations, to make more informed decisions about which clusters to select. Another area on improvement is generation of independent SI/SO patterns rather than giving random ones to the oracle during the symbolic equation system step of the framework.

Author Contributions: Conceptualization; methodology; software; validation; formal analysis, J.W., S.R., O.D.-P. and D.F.; writing—original draft preparation, J.W.; writing—review and editing, O.D.-P.; supervision, D.L.W.; project administration; funding acquisition, D.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NSF Grant NO. 2016624.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

References

1. Aerts, J.; Marinissen, E.J. Scan chain design for test time reduction in core-based ICs. In Proceedings of the Proceedings International Test Conference 1998 (IEEE Cat. No. 98CH36270), Washington, DC, USA, 18–23 October 1998; pp. 448–457.
2. TestMAX ATPG:Advanced Pattern Generation. Available online: https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-atpg.html (accessed on 1 January 2025).
3. *IEEE Std 1149.1-2013*; IEEE Standard for Test Access Port and Boundary-Scan Architecture. IEEE: Washington, DC, USA, 2013; pp. 1–444. [CrossRef]
4. Sao, Y.; Ali, S.S. Security Analysis of Scan Obfuscation Techniques. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 2842–2855. [CrossRef]
5. Sao, Y.; Ali, S.S.; Mazumdar, B. DefScan: Provably Defeating Scan Attack on AES-Like Ciphers. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2024**, *43*, 2326–2339. [CrossRef]
6. Yang, B.; Wu, K.; Karri, R. Scan based side channel attack on dedicated hardware implementations of Data Encryption Standard. In Proceedings of the 2004 International Conferce on Test, Charlotte, NC, USA, 26–28 October 2004; pp. 339–344. [CrossRef]
7. Yang, B.; Wu, K.; Karri, R. Secure Scan: A Design-for-Test Architecture for Crypto Chips. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2006**, *25*, 2287–2293. [CrossRef]
8. Kodera, H.; Yanagisawa, M.; Togawa, N. Scan-based attack against DES cryptosystems using scan signatures. In Proceedings of the 2012 IEEE Asia Pacific Conference on Circuits and Systems, Kaohsiung, Taiwan, 2–5 December 2012; pp. 599–602. [CrossRef]
9. Xu, X.; Yang, H.; Zou, J.; Cai, Z.; He, L. A High Security Encryption Circuit Based on Ring Oscillator PUF and Secure Scan Chain. In Proceedings of the 2023 6th International Conference on Electronics Technology (ICET), Chengdu, China, 12–15 May 2023; pp. 58–62. [CrossRef]
10. Agrawal, M.; Karmakar, S.; Saha, D.; Mukhopadhyay, D. Scan Based Side Channel Attacks on Stream Ciphers and Their Counter-Measures. In Proceedings of the Progress in Cryptology—INDOCRYPT 2008, Kharagpur, India, 14–17 December 2008; Chowdhury, D.R., Rijmen, V., Das, A., Eds.; Spinger: Berlin/Heidelberg, Germany, 2008; pp. 226–238.
11. Atobe, Y.; Shi, Y.; Yanagisawa, M.; Togawa, N. State dependent scan flip-flop with key-based configuration against scan-based side channel attack on RSA circuit. In Proceedings of the 2012 IEEE Asia Pacific Conference on Circuits and Systems, Kaohsiung, Taiwan, 2–5 December 2012; pp. 607–610. [CrossRef]
12. Lee, J.; Tehranipoor, M.; Patel, C.; Plusquellic, J. Securing Designs against Scan-Based Side-Channel Attacks. *IEEE Trans. Dependable Secur. Comput.* **2007**, *4*, 325–336. [CrossRef]
13. Gaikwad, P.; Slpsk, P.; Bhunia, S. Invisible Scan for Protecting Against Scan-Based Attacks: You Can't Attack What You Can't See. In Proceedings of the 2023 IEEE International Test Conference India (ITC India), Bengaluru, India, 23–25 July 2023; pp. 1–6. [CrossRef]
14. Zhang, D.; He, M.; Wang, X.; Tehranipoor, M. Dynamically obfuscated scan for protecting IPs against scan-based attacks throughout supply chain. In Proceedings of the 2017 IEEE 35th VLSI Test Symposium (VTS), Las Vegas, NV, USA, 9–12 April 2017; pp. 1–6. [CrossRef]
15. 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: Washington, DC, USA, 2015; pp. 137–143.
16. Rahman, M.S.; Nahiyan, A.; Amir, S.; Rahman, F.; Farahmandi, F.; Forte, D.; Tehranipoor, M. Dynamically Obfuscated Scan Chain to Resist Oracle-Guided Attacks on Logic Locked Design. Cryptology ePrint Archive, Paper 2019/946. Available online: https://eprint.iacr.org/2019/946 (accessed on 19 August 2019).
17. Ahmed, A.; Farahmandi, F.; Iskander, Y.; Mishra, P. Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution. In Proceedings of the 2018 IEEE International Test Conference (ITC), Harbin, China, 15–17 August 2018; pp. 1–10. [CrossRef]
18. Vafaei, A.; Hooten, N.; Tehranipoor, M.; Farahmandi, F. SymbA: Symbolic Execution at C-level for Hardware Trojan Activation. In Proceedings of the 2021 IEEE International Test Conference (ITC), Shanghai, China, 18–20 August 2021; pp. 223–232. [CrossRef]
19. Fowze, F.; Choudhury, M.; Forte, D. EISec: Exhaustive Information Flow Security of Hardware Intellectual Property Utilizing Symbolic Execution. In Proceedings of the 2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Singapore, 14–16 December 2022; pp. 1–6. [CrossRef]
20. Zhao, P.; Liu, Q. Density-based Clustering Method for Hardware Trojan Detection Based on Gate-level Structural Features. In Proceedings of the 2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Xi'an, China, 16–17 December 2019; pp. 1–4. [CrossRef]

21. Salmani, H. COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 338–350. [CrossRef]
22. He, C.; Lei, D.; Wu, H.; Cheng, L.; Yan, G.; Huang, Q. A Side-Channel Hardware Trojan Detection Method Based on Fuzzy C-Means Clustering and Fusion Distance Algorithms. *IEEE Internet Things J.* **2024**, *11*, 13927–13937. [CrossRef]
23. Wu, J.; Fowze, F.; Forte, D. EXERT: EXhaustive IntEgRiTy Analysis for Information Flow Security. In Proceedings of the 2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Singapore, 14–16 December 2022; pp. 1–6. [CrossRef]
24. Jasper Sequential Equivalence Checking. Available online: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/jaspergold-sequential-equivalence-checking-app.html (accessed on 1 January 2025).
25. Potluri, S.; Aysu, A.; Kumar, A. SeqL: Secure Scan-Locking for IP Protection. In Proceedings of the 2020 21st International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 25–26 March 2020; pp. 7–13. [CrossRef]
26. Lee, K.J.; Liu, C.A.; Wu, C.C. A Dynamic-Key Based Secure Scan Architecture for Manufacturing and In-Field IC Testing. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 373–385. [CrossRef]
27. De Cannière, C. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Proceedings of the Information Security, Samos Island, Greece, 30 August–2 September 2006; Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 171–186.
28. Courtois, N.T.; Meier, W. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Proceedings of the Advances in Cryptology—EUROCRYPT 2003, Warsaw, Poland, 4–8 May 2003; Biham, E., Ed.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 345–359.
29. Hawkes, P.; Rose, G.G. Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers. Available online: https://eprint.iacr.org/2004/081 (accessed on 16 March 2004).
30. Palit, S.; Roy, B.K.; De, A. A Fast Correlation Attack for LFSR-Based Stream Ciphers. In Proceedings of the Applied Cryptography and Network Security, Kunming, China, 16–19 October 2003; Zhou, J., Yung, M., Han, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 331–342.
31. Méaux, P.; Wang, Q. Extreme Algebraic Attacks. Available online: https://eprint.iacr.org/2024/064 (accessed on 17 January 2024).
32. Faugére, J.C. A new efficient algorithm for computing Gröbner bases (F4). *J. Pure Appl. Algebra* **1999**, *139*, 61–88. [CrossRef]
33. Bard, G.V.; Courtois, N.T.; Jefferson, C. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. Available online: https://eprint.iacr.org/2007/24 (accessed on 26 January 2007).
34. Teo, S.G.; Wong, K.K.H.; Bartlett, H.; Simpson, L.; Dawson, E. Algebraic analysis of Trivium-like ciphers (poster). In Proceedings of the Twelfth Australasian Information Security Conference (AISC '14),—Volume 149, AUS, Auckland, New Zealand, 20–23 January 2014; pp. 77–81.
35. Dudek, J.M.; Meel, K.S.; Vardi, M.Y. The Hard Problems Are Almost Everywhere For Random CNF-XOR Formulas. *arXiv* **2017**, arXiv:1710.06378.
36. Creignou, N.; Daude, H. Satisfiability threshold for random XOR-CNF formulas. *Discret. Appl. Math.* **1999**, *96–97*, 41–53. [CrossRef]
37. Limaye, N.; Sinanoglu, O. DynUnlock: Unlocking Scan Chains Obfuscated using Dynamic Keys. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 270–273. [CrossRef]
38. Alrahis, L.; Yasin, M.; Limaye, N.; Saleh, H.; Mohammad, B.; Al-Qutayri, M.; Sinanoglu, O. ScanSAT: Unlocking Static and Dynamic Scan Obfuscation. *IEEE Trans. Emerg. Top. Comput.* **2021**, *9*, 1867–1882. [CrossRef]
39. Chen, D.; Lin, C.; Beerel, P.A. GF-Flush: A GF(2) Algebraic Attack on Dynamically Secured Scan Chains. In Proceedings of the 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Athens, Greece, 6–8 October 2021; pp. 1–6. [CrossRef]
40. Wu, J.; Dizon-Paradis, O.; Rahman, S.; Woodard, D.; Forte, D. DOSCrack: Deobfuscation Using Oracle-Guided Symbolic Execution and Clustering of Binary Security Keys. In Proceedings of the 2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 6–9 May 2024; pp. 227–232. [CrossRef]