EXERTv2: Exhaustive Integrity Analysis for Information Flow Security with FSM Integration

Jiaming Wu, *Student Member, IEEE* and Domenic Forte, *Senior Member, IEEE*, University of Florida

***Abstract*—Hardware information flow analysis detects security vulnerabilities resulting from microarchitectural design flaws, design-for-test/debug (DfT/D) backdoors, and hardware Trojans. Though information flow violations can be manifested through a multitude of possible ways, prior research has only focused on detecting the existence of such vulnerabilities and no approach has been proposed to exhaustively activate all vulnerable points and reduce false positives. In this paper, we propose EXERTv2, a novel analysis framework that combines ATPG, SAT, and FSM analysis as well as FSM integration to detect information flow violations and perform exhaustive analysis that reports the complete set of integrity-violating input patterns for vulnerable control points. Compared with the original version of EXERT, the significant contribution of EXERTv2 is its algorithm for integrating FSMs, which simplifies the process of constraining multiple FSMs. The FSM analysis and integration, in particular, consider the behavior of all the FSMs in the design as a whole, which can be performed offline and helps resolve scalability limitations in prior approaches while remaining exhaustive. We also demonstrate EXERT's usage in the application of fault injection vulnerability analysis and attacks. As a proof-of-concept, EXERTv2 is evaluated on multiple Trojan benchmarks from Trust-Hub and two additional ciphers. It detects rare Trojan triggers (activation probability $\approx$ 1.4243e-70), generates all activation patterns within minutes, and shows a 15$\times$ to 110$\times$ faster run time compared with Cadence Jasper Security Path Verification (SPV). EXERT is also applied to a larger RISC-V benchmark to identify instruction sequences with and without fault injection that result in privilege escalation.**

***Index Terms*—information flow tracking, hardware Trojan, pattern generation**

I. Introduction

As the semiconductor industry moves from vertical to horizontal supply chains, modern computing hardware is often designed and fabricated in discrete locations by third parties with different levels of trust. As a result, hardware security threats might occur at any point, from functional specifications to manufacturing and finally in-field usage [1]. The system-on-chip (SoC) design and fabrication flow involves the following major parties: third-party intellectual property (IP) Vendors (e.g., ARM, Rambus, etc.), SoC design houses (e.g., Qualcomm, Apple, etc.), commercial CAD tool providers (e.g., Cadence, Synopsys, etc.), and contract fabs/foundries (e.g., TSMC, GlobalFoundries, etc.). Untrusted third parties involved in a horizontal supply chain have full access to hardware IPs and can possibly modify them during design or fabrication. Such intentional malicious modifications are often referred to as hardware Trojans [2]. Even though some parties are trustworthy. such as CAD tool providers, synthesis and optimization processes can also introduce unintentional design flaws that make hardware assets vulnerable.

Jiaming Wu and Domenic Forte are with the Electrical and Computer Engineering (ECE) Department, University of Florida, Gainesville, FL, USA.
Manuscript received XXX; revised YYY

Recent research has discovered multiple forms of attacks that utilize the vulnerabilities of modern processors and SoCs to perform malicious operations, steal valuable assets, and hijack control flows. For example, Spectre [3] exploits timing side-channels to speculatively perform actions that leak private data from memory. Meltdown [4] exploits architectural design vulnerabilities to break the memory isolation and allow unauthorized access to memory data. Design for Test (DfT) structures allow external and direct access to certain nets within an SoC which adds additional vulnerabilities to the system [5]. As modern processors and SoCs are the cores of electronic devices, such attacks performed on chips can effectively leak private information, leak proprietary software, recover secret keys, and allow remote access to control our increasingly networked devices [6]. Another concern is that intentional vulnerabilities can be added to SoCs through hardware Trojans from third-party IP (3PIP) vendors as they do not share the same level of trust. These hardware Trojans are designed by adversaries to be difficult to detect which results in stealthy backdoors that leak important information or act as a killswitch. These backdoors can remain undetected for years, creating huge risks for technology companies, banks, and even military defense systems.

To mitigate such issues, there is a significant need to develop scalable frameworks that detect micro-architectural flaws as well as hardware Trojans for application in pre-silicon and post-silicon stages. Further, we emphasize that such a framework should be capable of performing *exhaustive analysis*. First, exhaustiveness is the only way to *completely eliminate all flaws* and achieve a secure system. For example, exhaustive patterns search the entire state of activation vectors that trigger stealthy (difficult to activate) hardware Trojans. By exhaustively checking all paths and patterns, design and verification engineers can even remove false positives. Second, exhaustion can help to *better quantify how vulnerable a system is*. For example, existing tools often find a single counterexample that demonstrates a vulnerability. If an engineer knows that this pattern will never appear in real-time, it might provide the designer with a false sense of security because other patterns were not identified. Moreover, through exhaustion, one can compare the security of two systems/designs and identify the better one based on the number and/or complexity of triggering patterns. Third, recent research is exploiting AI to find vulnerabilities, generate test patterns, perform fuzzing tests, etc [7]. A well-known bottleneck in AI, particularly deep learning, is data. That is, a neural network may need hundreds of thousands of samples to converge. Exhaustive pattern generation can *obtain the sufficient amount of data needed to train AI-based approaches*. Besides, exhaustive patterns can allow design and verification engineers to check if all vulnerable control patterns can ever appear in real time which confirms a true vulnerability. As the fact that not all activation vectors are possible to occur in run-time, if the limited patterns never appear then it might provide the designer with a false sense of security. Last but not least, the exhaustive patterns can be used post-silicon by *real-time monitors that filter illegal input patterns/sequences* during execution. When such patterns are generated, the system can respond to avoid a security issue, e.g., through zeroization of sensitive data, system reset, etc.

In this paper, we propose EXhaustive IntEgRiTy Analysis (EXERT), a SAT and IFT (information flow tracking)-based approach that produces a complete set of I/O patterns given a netlist under test and target assets. Finite state machine (FSM) analysis and automatic test pattern generation (ATPG) method are also involved to make EXERT more scalable. Also, EXERT is extended to version 2 with FSM integration module. Through EXERTv2, it is possible to generate all patterns that trigger a Trojan and cause confidentiality/integrity issues in processors and other IPs. Our contributions are summarized as follows:

- We propose an exhaustive pattern generation framework based on SAT and IFT. The novelty of our approach is that we decompose sequential data-paths with fan-in cones and generate exhaustive patterns without false positives at each level during backward propagation based on IFT techniques. Also, unlike other previous techniques, our approach can work with DfT-inserted netlists.
- We utilize offline FSM extraction techniques to analyze the information flows and the input patterns driving them in FSMs. FSM extraction is performed only once based on the interaction analysis (Section III-B) of registers. This helps avoid the state explosion problem and makes EXERT's pattern generation scale.
- To improve FSM analysis for solving multiple FSM interaction scenarios, we expand EXERT to version 2 (v2). FSM integration techniques (Section III-E) are used to handle the analysis of all the FSMs in the design as a whole. This allows EXERT to handle additional ciphers, Subterranean, and SAEAES.
- We introduce the fault injection testing framework expanded from EXERTv2 which generates fault injection points with corresponding patterns causing an integrity violation. We experimentally perform a fault injection attack on RISC-V privilege escalation utilizing this framework.
- We experimentally run EXERT on multiple sequentially triggered Trojan benchmarks with different target designs from Trust-hub. Our proposed technique can efficiently and exhaustively detect Trojan trigger patterns for sequential Trojans with triggering probability as low as 1.4243e-70. We also compare the run time of the proposed framework with the state-of-the-art commercial tool, JasperGold. Our experiments show that it is more than 15 times faster than Cadence Jasper Security Path Verification (SPV) [13].
- For further demonstration of scalability and application scope, we test EXERT and the fault injection testing framework on a 25k+ gate RISC-V processor to generate instruction sequences that lead to privilege escalation.

The rest of the paper is organized as follows. In Section II, we give the necessary background and related work. In Section III, we discuss our proposed EXERT framework and its complexity while Section IV shows EXERT's application to fault injection analysis. Section V analyzes the results from our framework on benchmarks with very complex sequential triggers. Finally, Section VI provides a conclusion and directions for future work.

II. Preliminary and Related Work

A. Threat Model

In this section, we briefly review the threat model of IC supply chains and present how our framework can address common threats (Hardware Trojans and malicious design flaws) under certain assumptions. We assume that potential adversaries can insert Hardware Trojans at any stage of IC supply chains including during the use of third-party IPs (3PIP), third-party EDA tools or vendors (3P-EDA), and external design expertise. Additionally, the synthesis and optimization process can unintentionally introduce design flaws. We assume that the Hardware Trojans and design flaws are of extremely low activation probability and hard to activate through purely functional testing. We also assume we can obtain a list of interest points that are potential asset points to activate Hardware Trojans and design flaws by using existing testability measurement tools. For example, SCOAP [14] and COTD [15] test for malicious signals by measuring the controllability and observability of all signals and further utilizing clustering analysis based on testability reports, imprecise IFT [16] allows a quick profile of potential information flow security vulnerabilities that identifying security violation points [17]. Under these assumptions, our framework relies on access to the synthesized gate-level netlist and any points of interest specified by a user for analysis, generating the exhaustive patterns that activate the points of interest. In this paper, we aim to present that these patterns certainly help designers understand how privileged asset points are accessed and explore the violation of integrity.

B. Information Flows Tracking (IFT)

IFT is a well-formulated formal method for verifying security properties with confidentiality or integrity violations. Many projects have utilized IFT to build verification frameworks (e.g., GLIFT. [8], RTLIFT. [9], and SecVerilog [10], etc.). GLIFT assigns a label (tainted or not) to the target bit of a design under test and models how a single data bit propagates. Such assignments provide the foundation for modeling how a single data bit tangles with other labeled bits. RTLIFT improves upon GLIFT to provide tracking information flows in a higher level of abstraction. Another verification framework

TABLE I
COMPARISON BETWEEN EXISTING METHODS AND PROPOSED EXERT. $\surd$ ($\times$) DENOTES THAT A METHOD DOES (DOES NOT) POSSESS A FEATURE.

Previous Work	GLIFT [8]	RTLIFT [9]	SecVerilog [10]	Trojan Activation [11]	IFS Verification [12]	EXERT (v2)
Target	Taint bit in netlist	Taint bit in RTL	Taint bit in RTL	Rare branch activation	Malicious observation, control point	Malicious control point activation
Methods	Taint labeling	Taint labeling	Dynamic labeling	Model checking, ATPG	ATPG-based IFT	ATPG-based IFT,SAT, FSM
Scan-chain required	N/A	N/A	N/A	$\surd$	$\surd$	N/A
ATPG mode	N/A	N/A	N/A	Full-sequential	Partial-scan	N/A
Exhaustive	$\times$	$\times$	$\times$	$\times$	$\times$	$\surd$
Scalable	Overhead	Overhead	$\surd$	$\times$	$\times$	$\surd$
Multipe FSMs	N/A	N/A	N/A	No	No	No (Yes)

SecVerilog extends Verilog with information flow flags that support comprehensive, precise reasoning about information flows at compile time [10]. It adds a timing tag to Verilog through a type system to verify the security of the timing information flow offline. SecVerilog takes Verilog code with security labels as input. Once a target design passes verification, security labels are removed and normal Verilog code is generated. The generated Verilog code complies under information flow policies defined by SecVerilog yet without any overhead constraining code.

Existing IFT methods tend to take a qualitative approach and only enforce binary security properties. In previous work, a binary answer (yes or no) is provided regarding the flow of information between design elements. Nevertheless, GLIFT and RTLIFT also suffer from overhead in producing the shadow logic for their propagation of security tags. SecVerilog does not have such overhead as it generates information flow policy-defined RTL, but it requires the designer to hard-code information flow policies when adding labels to variables for each specific security policy. In [12], Nahiyan et al. proposed the framework of backward propagation-based IFT and utilized commercial ATPG tool Tetramax (Synopsys) [18] to generate potential patterns for propagating information flow. However, this work only generates a single pattern (which may be a false positive) and suffers from scalability issues due to sequential ATPG reliance. The summarized comparison of these works with our proposed framework is shown in Table I.

C. Boolean Satisfiability (SAT)

SAT approaches are formal methods for finding an assignment of 0s and 1s to a Boolean function's variables which make it evaluate to 1. In order to utilize SAT to verify the functions of a circuit, the circuit must first be transformed into formulas that can be handled by SAT solvers. SAT-based ATPG [19] turned out to be a robust alternative to classical structural ATPG algorithms [20]. The number of unclassified faults can be significantly reduced using SAT-based ATPG. However, previous SAT-based ATPG algorithms can only be applied to combinational logic as the timing nature of sequential logic is not transformable to a Boolean representation. The robustness attribute of SAT-based ATPG algorithm is therefore not applicable to sequential pattern generation. By transforming combinational logic into a Boolean formula and further conjunctive normal form, the solution of this CNF(conjunctive normal form) represented SAT problem is a valid test pattern. Such serial transformations are infeasible for sequential logic because of the timing nature. Thus, the robustness attributes of SAT-based ATPG algorithm is not applicable to sequential pattern generation.

D. Jasper Gold Security Path Verification (SPV)

Commercial CAD/EDA tools are also equipped with information flow tracking capabilities. The Cadence Jasper Security Path Verification (SPV) App [13] is a formal verification tool that takes the design (RTL/gate-level) to prove whether a secured area can be accessed through paths or not. It proves the leakage with mathematical certainty and a detailed waveform of how security assets are leaked if a path is detected. Based on its own path-sensitization technology, it finds paths that propagate data from sources and destinations the user provides. However, the path-sensitization technology is not pattern-driven and requires hard coding of the policies to use it. In Section V, we show that this is more than one to two orders of magnitude slower than EXERT.

E. Finite State Machines (FSMs)

For convenience, an FSM is typically represented as a directed graph where each vertex represents a state and each edge represents the transition from the current state to the next state.Transition conditions consist of an input pattern for creating a valid transition in an FSM from its initial state to its final state. While FSMs can be modeled as graphs in high-level description, their physical implementation in low-level netlists is a set of registers with combinational logic and feedback nets connecting them. Tools have been developed in the prior work to convert gate-level netlist descriptions into FSM graphs [21]. We make use of these open-source tools to aid EXERT.

F. Exhaustive Analysis

Exhaustive analysis, also known as exhaustive search or exhaustive testing, is a method of evaluating or exploring all possible options systematically and comprehensively. Exhaustive testing guarantees that all detectable faults will be detected [22]. Exhaustiveness implies every possible solution or outcome is examined or tested to ensure that no valuable information or potential solution is neglected. In EXERT, the integrity violations are validated by the exhaustiveness of activating patterns.

III. EXERT FRAMEWORK

The objective of this paper is to develop a framework called EXERT, which analyzes the integrity violations of any malicious points and generates exhaustive patterns to activate such violations. Our proposed framework is the first approach that could perform exhaustive analysis while maintaining scalability. EXERT framework aims to perform integrity analysis statically on a synthesized netlist with quantitative patterns

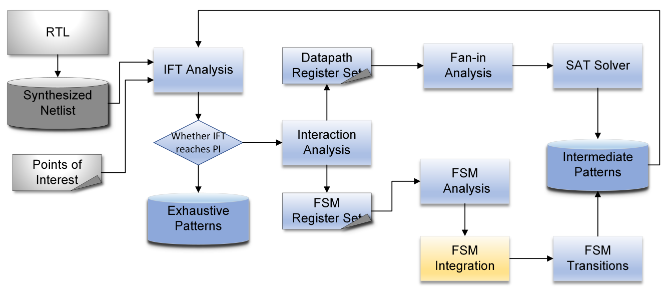


Fig. 1. Block diagram of EXERT framework.

to support our analysis from input ports to internal nets. In contrast to existing quantitative information flow tracking tools, such as Qflow [23] and QIF-Verilog [24], which perform confidentiality analyses with quantitative leaked information at compile time. EXERT takes the target netlist[1] of the design and points of interest as input, then outputs the malicious points and their propagation patterns. Our target points of interest can be either Trojan's triggering nets or any potential malicious point that violates an integrity policy (e.g., modifying an asset through an unauthorized input port). The overall EXERTv2 framework is shown in Figure 1, which introduces an additional module of FSM integration based on the original framework of EXERT. The EXERTv2 framework primarily consists of five modules: IFT analysis, interaction analysis, fan-in analysis, FSM analysis, and FSM integration. Starting from synthesis, the target design is verified first with our IFT analysis module. This analysis operates in a cycle-based manner, systematically examining the information flow at each clock cycle until it reaches the primary inputs. During every clock cycle, the Interaction analysis module is invoked, responsible for classifying the registers present in that specific cycle. Depending on the results of the Interaction analysis, either the Fan-in analysis or the FSM analysis will be invoked. The Fan-in analysis aims to generate exhaustive patterns on data-path asset points while the FSM analysis aims to generate propagating patterns by analyzing a group of registers as FSMs. Besides, by introducing FSM integration, multiple FSMs are integrated and analyzed as a whole by consolidating the state transitions and states of individual FSMs. The detailed explanations of each module are shown in the subsections below.

A. Information Flow Tracking Analysis

As shown in Algorithm 1 and illustrated in Figure 1, the first step of EXERT is IFT analysis which performs a taint propagation from the asset points of interet to the inputs. IFT analysis takes the nets of interest and target gate-level netlist as input, and reports the register control points and primary input control points which potentially represent the existence of information flows that violate an integrity policy. Our proposed analysis first takes the net of interest as the starting point and

[1]Our framework analyzes designs at the gate level instead of RTL because the synthesis process may introduce vulnerabilities due to optimization of don't cares [25], DfT insertion [5], etc. Nevertheless, EXERT can analyze RTL after it is synthesized into a netlist.

Algorithm 1 IFT Backward Propagation Algorithm

```
Input: Nets of interest, gate-level netlist;
Output: Register control points, Primary input control points;
 1: start point FF level: $FFlevel \longrightarrow 1$
 2: for all $Asset$ in Nets of interest do
 3:     $FaninReg \longleftarrow \{\}$
 4:     $FaninReg$ at $FFlevel \longleftarrow fanin(Asset, FFlevel)$
 5:     $FaninPI$ at $FFlevel \longleftarrow fanin(Asset, FFlevel)$
 6: while $FaninReg$ exists do
 7:     invoke Interaction Analysis
 8:     Update Asset: $Asset$ = $FaninReg$
 9:     $FFlevel$ += 1
10:     $FaninReg$ at $FFlevel \longleftarrow fanin(Asset, FFlevel)$
11:     $FaninPI$ at $FFlevel \longleftarrow fanin(Asset, FFlevel)$
12:     append $RegisterControlpoints \leftarrow FaninReg$
13:     append $PrimaryInputControlpoints \leftarrow FaninPI$
14: return $RegisterControlpoints, PrimaryInputControlpoints$
```

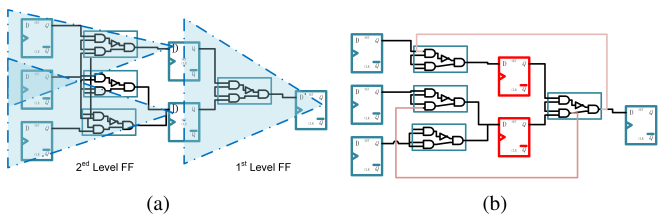


Fig. 2. Interaction analysis showing how IFT propagates backward through levels of registers: (a) without feedback nets, registers are classified as data-path registers; and (b) with feedback nets, the registers connected by feedback nets are classified as FSM state registers.

records all fan-in registers and fan-in primary inputs (lines 3 to 5 in Algorithm 1). When fan-in registers exist, the interaction analysis module is invoked to determine which set of registers these fan-in points belong to. Next, the nets of interest are updated with fan-in registers (lines 8 to 9) and keep looking for fan-ins in the prior level of flip-flops (FFs) (lines 9 to 11) until the complete set of control points with all their corresponding FF levels are recorded.

B. Interaction Analysis

This module, which is invoked in IFT analysis, takes the whole netlist as input and targets the feedback nets in a design for classification. In Section II-E, we discussed that an FSM's physical implementation is a set of state registers with feedback nets connecting them (see Figure 2). The objective of interaction analysis is to classify registers into datapath and FSM register sets to better analyze them with fan-in analysis and/or FSM analysis described in Sections III-C and III-D. This allows for tracking information flows with better performance. Simply tracking information flows in FSMs inevitably results in a loop or reverse flow. The registers in Figure 2(a) are classified as data-path registers since there are no feedback nets connected. Figure 2(b) shows the feedback nets and associated FSM/state registers in red. Note the interaction analysis will only be performed once and is conducted offline. This further promotes efficiency in EXERT.

C. Fan-in Analysis

In this section, we describe the fan-in analysis module applied on the data-path registers that were identified by interaction analysis in Section III-B. It aims to generate exhaustive patterns on data-path asset points (classified by

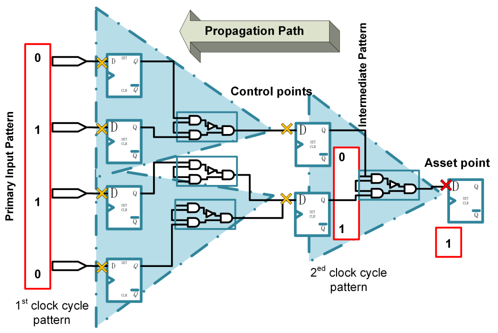


Fig. 3. Illustration of fan-in analysis. The red cross denotes the target asset point of interest. Yellow crosses denote the detected control points. The intermediate patterns match every control point detected.

interaction analysis) with scalable effort. Our fan-in analysis is capable of tracking information flows through integrity verification and generating complete sets of patterns that propagate them. The fan-in analysis engine is based on the concept of decomposing a sequential circuit into cascading combinational circuits by tracking the fan-in cone of every asset point. As illustrated in Figure 3, the two-stage data path is decomposed into three cascading combinational cones. The inputs of the decomposed combinational circuit are the *control points* set of the output, which is the target asset point. The control point set will give us a new asset point set in the next level of registers and keep looking for fan-in cones until no more control points are detected. During the fan-in analysis, SAT engines [26] and ABC synthesis [27] are utilized to generate intermediate patterns while AIGer [28] compresses and constrains them. AIGer is fast and scales effectively for creating and manipulating circuits at each cycle [29].

1) *Pattern-Driven Information Flow Tracking*

The proposed fan-in analysis module is presented in Algorithm 2 and illustrated in Figure 3. The algorithm first takes the netlist under test and target asset point (e.g., red cross in Figure 3) as inputs. Then, it looks for fan-in registers of the asset point and extracts the combinational block with fan-in registers as inputs and asset point as output using Design Compiler (Procedure I of Algorithm 2). Given the fan-in registers from IFT analysis (see Section III-B), it shows if there are information flows from fan-ins to target asset; hence, there may exist a set of patterns to propagate them (see *Intermediate Pattern* in Figure 3). Next, the extracted combinational block is transformed into CNF and fed into a SAT engine to generate the complete set of patterns that propagate the information flow (Procedure II in Algorithm 2). At this point, the list of control points that present information flows and the intermediate patterns driving them are reported. With the IFT analysis (see Section III-B) detecting control points at the next level, this procedure will keep being invoked until no more fan-in registers are detected, i.e., where our analysis reaches primary inputs. The patterns generated at every cycle make up the exhaustive pattern sequence as the IFT analysis aborts. The patterns generated here expand our IFT analysis with propagating patterns which makes our analysis a pattern-driven information flow framework. In order to keep our algorithm exhaustive but still scalable, the intermediate patterns generated at each level need compression and con-

Algorithm 2 Fan-in Cone Extraction and Analysis

```
Procedure I: Fan-in Cone Extraction
Input: Gate-level netlist, Asset point, fan-in register set from Interaction
    analysis;
Output: Combinational block in RTL, Asset Control Points
1: for all $register$ in fan-in Register set do
2:     Create Primary Input at $register$
3: Create Primary Output at $TargetAsset$
4: Re-synthesis $\longrightarrow$ Combinational block in RTL
5: return Combinational block in RTL
Procedure II: SAT-based Pattern Generation
Input: RTL file, Asset point
Output: Intermediate Patterns;
1: Transform RTL into CNF format $\longrightarrow$ CNF expression
2: Feed to SAT solver $\longrightarrow$ $SatisfyingPattern$
3: repeat
4:     CNF expression $\longleftarrow$ Addclause($SatisfyingPattern$)
5:     SAT($CNFexpression$) $\longrightarrow$ $SatisfyingPattern$
6:     append $IntermediatePatterns \leftarrow SatisfyingPattern$
7: until $SatisfyingPattern = \emptyset$
8: return Intermediate Patterns
Procedure III: Constrain and Compress
Input: Intermediate Patterns
Output: Exhaustive Primary Input Patterns, Compressed Block in AIGer,
    Asset Control Points
1: Classify patterns into Primary Input Control Point Patterns and Register
    Control Point Patterns
2: for Primary Input Control Point Patterns do
3:     Stored as exhaustive Patterns at current $FFlevel$
4: for Register Control Point Patterns do
5:     Construct in AIGer formate as constraints for next $FFlevel$
6: $AIGer(IntermediatePattern, Constraints) \longrightarrow$ Delete invalid in-
    termediate patterns
7: $AIGer$ $\longrightarrow$ Exhaustive Patterns
8: return Exhaustive Patterns at current $FFlevel$, $AIGer$ at current
    $FFlevel$
```

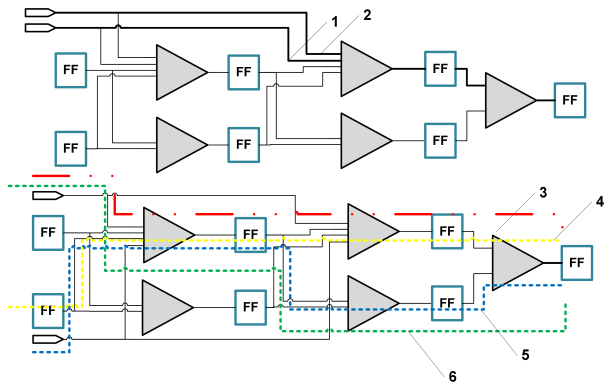


Fig. 4. State explosion with the growth of register level. At the first level, only 2 paths are detected. At the second level, 4 paths are detected. At the n-th level, 2^n paths are detected.

straining so that invalid/reproduced patterns can be discarded (see Procedure III of Algorithm 2). This procedure is necessary to discard the false positive information flows without their associated propagating patterns.

2) *Preventing State Explosion*

As the number of state variables in the system increases, the size of the system state space grows exponentially. This is called the "state explosion problem" [30]. As illustrated in Figure 4, the number of paths that represents the information flow grows exponentially with register depth, with 2 paths in the first level and 4 more in the second level. This makes our pattern-driven information flow tracking a state explosion problem as tracking information flows performs is similar in

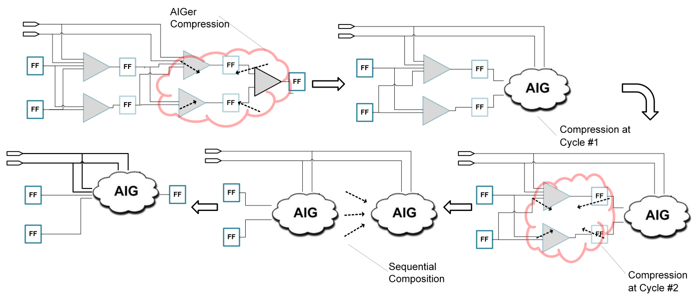


Fig. 5. AIGer Compression and constraints. At every cycle, the intermediate patterns at the control points are compressed in AIGer format. By sequentially compositing in the next cycle, invalid intermediate patterns are discarded.

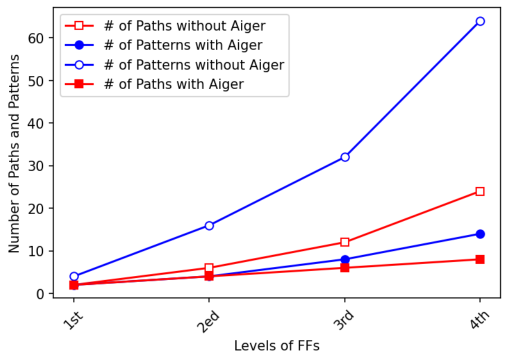


Fig. 6. Plots of path/pattern growth with FF level.

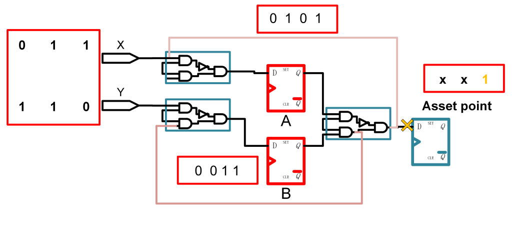
complexity as tracking paths. In order to make EXERT more scalable, we compress the intermediate patterns and paths at every level to analyze them as a whole in AIGer format. As illustrated in Figure 5, after generating the intermediate patterns, the patterns recognized in fan-in cones are compressed at every level and sequentially composed for the next level. As a pattern-driven information flow analysis framework, paths without valid propagating patterns are discarded.

3) *Time Complexity Analysis*

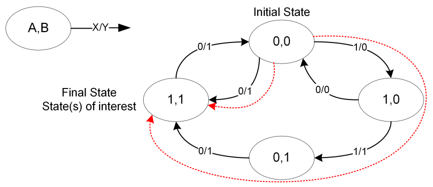
With our compression and constraining in AIGer, our analysis is more scalable compared to an uncompressed one. As shown in Figure 6, since paths in the former levels are analyzed as a whole, the newly detected paths in every level of the register grow linearly instead of exponentially. Let m be the number of paths in a single level of registers and N as the level of registers. To search all flows, the timing complexity without compression is $O(m^N)$ while the timing complexity with the proposed AIGer compression is $O(N)$, which shows a significant reduction in complexity. The number of patterns detected is also reduced. That is, AIGer compression and composition reduce patterns that were valid in the previous level of control points but actually become invalid in the next level of registers. For example, this can be seen in Figure 3. We may detect *multiple* intermediate patterns at the first level of FFs, which represent the states of current control points. However, some of the intermediate patterns may not be valid as these states of control points may not be reachable in the next level of fan-in analysis.

4) *Space Complexity Analysis*

In Section III-C3, timing complexity is analyzed to prove how patterns are reduced with our AIGer compression. In this section, space complexity is analyzed to validate the scalability



(a)



(b)

Fig. 7. Example of FSM analysis where two registers A,B denote the state variables and X/Y denote the two inputs. (a) Netlist level abstract of the FSM with a valid sequence of patterns; and (b) Extracted graph of the FSM with 4 states made of A,B and 6 state transitions made of X/Y .

of our approach. Let m denote the number of registers in a single level of FFs. As illustrated in Figure 5, because the AIGer compression only compresses the current level of intermediate patterns to the next level, the number of fan-in blocks utilized and stored in memory is only $2m$. Therefore, the space complexity of the approach with AIGer compression is indeed $O(m)$. That is, the space complexity of this approach is linear with respect to the number of registers in a single level of FFs (also the fan-in blocks in a single level of FFs), indicating that it is scalable and can handle larger designs efficiently.

D. *FSM Analysis*

In Section III-B, our interaction analysis module classifies registers into datapath set and FSM set. In order to modulate the information flows in feedback nets (also referred to as the FSM registers set), we model the feedback nets along with their connected registers together as FSMs. The FSM analysis aims to generate propagating patterns by analyzing a group of registers as FSMs. Our FSM analysis module takes the netlist and the set of registers (see Figure 7(a)) classified by our interaction analysis (see Section III-C) as inputs and constructs an FSM (see Figure 7(b)) that represents the behavior of the register set over the course of the operation of the netlist in a graph [21].

When an FSM register is detected during our information flow analysis, EXERT will discard some information flows by tracking its fan-ins. This is because if the register belongs to an FSM, tracking its information flows through fan-ins will inevitably result in a looping flow or path since every register in an FSM is capable of reaching the others. Thus, we analyze all the registers in the FSM to generate patterns as a whole. By modeling FSM with a state transition graph, we can find the sequences of patterns that will force an FSM to transition from an initial state to a final desired control state at a higher level of abstraction than the netlist itself. When combined with our fan-in analysis, the "initial state" is defined from the register values after reset, and the "final state" is defined for the "control

TABLE II
RESULTS OF EXERT ANALYSIS ON SEQUENTIAL TROJAN BENCHMARKS.

Benchmark	# Gates	Trojan Trigger	Activation Probability	Trojan Payload	# Patterns	Length in Clock Cycles	Time(sec)
AES-T1400(M)	513	Predefined sequence	8.8162e-39	Leaks keys	12	4	46.78
AES-T1400(Max)	1336	Predefined sequence	≤4.4750e-142	Leaks keys	600	400	4743.49
PIC16-T200(M)	2015	Counting predefined instructions	2.8231e-118	Manipulates instruction register	729	100	453.90
s38417-T100	5431	Comparator	1.4243e-70	Manipulates scan mode	16	64	339.34
MC8051-T400	2040	Predefined instruction sequence	2.7105e-20	Manipulates flag registers	1	5	187.68
Subterranean(M)	1446	Counting predefined patterns	1.4131e-124	Leaks keys	81	100	567.56
SAEAES(M)	1521	Counter	2.6814e-66	Leaks keys	81	49	467.48

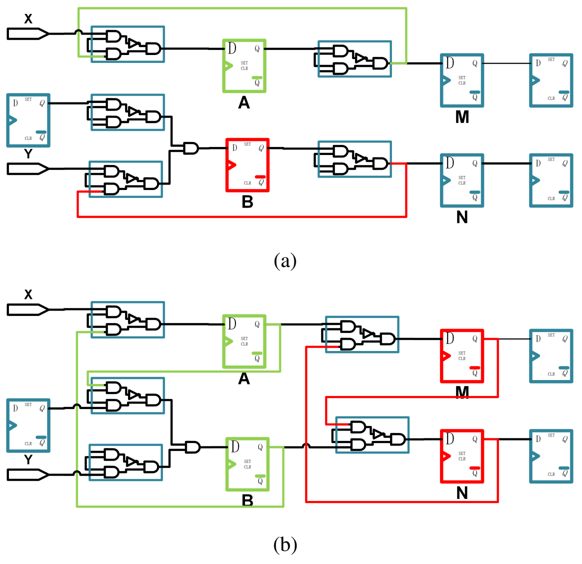


Fig. 8. Multiple FSMs integration: (a) FSMs in parallel; and (b) FSMs in series

state(s) of interest" which enables the information flow in the paths/patterns. In order to make the patterns exhaustive and scalable, we always look for the shortest paths in the state transition. Any looping paths or repeated states are discarded. As illustrated in Figure 7, two paths are detected from the initial state 0,0 to the final state 1,1; thus two pattern sequences consisting of these state transitions are generated (Figure 7 shows one of the two sequences in red). The two sequences of patterns will force the asset point to be 1 in the last clock cycle while the state values in the other two clock cycles don't matter in our FSM modeling. The patterns in FSM state transitions are also constrained along with intermediate patterns from fan-in analysis as shown in Figure 1. The combined patterns are compressed and moved to the next level of IFT analysis.

Note that alternative methods for generating patterns on sequential circuits without DfT, such as sequential ATPG, search for a sequence of test vectors through the huge space of all test vector sequences. Such methods are not scalable and our evaluation with Synopsys Tetramax [18] (not shown) terminates as sequential depth increases due to an abort limit.

E. FSM Integration

Our FSM analysis detects FSMs in a design and provides cycle-based patterns that propagate the information flows in a single FSM. However, it becomes more challenging to track patterns within a single FSM when multiple FSMs are detected in parallel or in series. As our cycle-based information flow analysis requires every pattern with a valid cycled tag, tracking parallel FSMs individually could result in propagating patterns with different clock cycles. In particular, a parallel interaction occurs when both FSMs are in states that can potentially make simultaneous transitions on the same input symbol. The behavior of parallel FSMs can interact in complex ways and it's important to ensure that the clock cycles are synchronized across all FSMs. In this context, it is necessary to develop an intra-FSM analysis technique that can track the cycle of propagation patterns across multiple FSMs. One approach to addressing this issue is to develop an analysis technique that considers the behavior of all the FSMs in the design as a whole, rather than analyzing them individually. Thus, we propose an algorithm that integrates multiple FSMs into a single one to analyze them as a whole using the original EXERT framework.

Algorithm 3 shows how we composite two FSMs into a single one that can be applied to integrate multiple FSMs by repeating the algorithm. As shown in the Figure 8, there are two scenarios of the FSMs interaction: series (see Figure 8(b)) and parallel (see Figure 8(a)). The green and red colors denote two FSM and the colored nets denote their corresponding feedback nets. Our algorithm first identifies the interaction scenario by the intersection nodes of two FSMs (line 1). If the algorithm identifies series interaction (line 2), a series graph composition function will be applied to composite two FSMs in graph format at the intersection node (line 3 to 4). For example, Figure 9 denotes the series composition of the two FSMs in Figure 8(b). Let's say we have two FSMs constructed by four registers A,B and M,N. After we model the FSMs with graphs and traditions, we always set the states 0,0 as our initial states as well as the connecting point for series graph composition since we assume that the FSMs are activated in order and the initial state for every register is 0. We then perform a Fan-in analysis at the intersection node register to determine the state values of A and B that activate M and N to be 0,0 (line 3). This means that the two FSMs are connected at this point of states, and a transition will be added between the activating state and the initial 0,0 state of the two FSM graphs. Thus we have the two FSM graphs for series composition by adding a transition between that activating state and the initial 0,0 state of two FSM graphs. Then we can search for all paths from "initial state" to "final state" in the merged graph (red arrows denote one possible path) which gives us the exhaustive sequence patterns. The resulting merged FSM states will include all of the states and transitions from the original FSMs without creating any new states or state transitions (line 5).

Otherwise, if the algorithm identifies parallel interaction (line 6), merged FSM states are created for every pair of states

Algorithm 3 FSM Integration Algorithm

```
Input: FSM1 states set $A$, FSM1 transitions set $q$, FSM2 states set $B$, FSM2
    transitions set $r$, FSM intersection nodes set;
Output: Merged FSM states, Merged FSM transitions;
 1: FSM interaction analysis: Identify the interaction scenario between FSM1
    and FSM2
 2: if FSM interaction analysis is series then Construct FSM1 graph and
    FSM1 graph;
 3:     Intersection nodes $\longrightarrow$ Connecting states
 4:     Series composition of FSM1 and FSM2 with intersection nodes
 5:     Merged FSM states and transitions: $(A + B, q + r)$
 6: else if FSM interaction analysis is parallel then
 7:     for all FSM states $A$ and $B$ do
 8:         Create merged FSM state $C == A \times B$
 9:     for all FSM transitions $q$ and $r$ do
10:         Create merged FSM transition $z == q \times r$
11:     Merged FSM states and transitions: $(A \times B, q \times r)$
12: return Merged FSM states and transitions
```

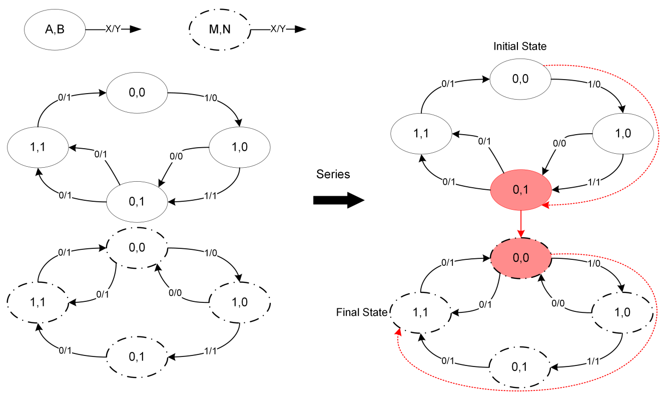


Fig. 9. Illustration of two series FSM composition in graph format based on the intersection nodes. A,B and M,N denote the state register in Figure 8(b). Solid nodes denote states of A, B; Dotted nodes denote states of M, N. Red nodes denote the connecting states and red arrows denote one possible path.

(one from each of the two FSMs being composed)(line 7 to 8). New transitions are added from the corresponding transitions in the original FSMs for every pair of state transitions (line 9 to 10). Specifically, the resulting FSM will have a state set that is the Cartesian product of the state sets of the original FSMs and the new transitions set which is the concatenation of the two transitions in the original FSMs. For example in Figure 10, let's say we have two FSMs A and B, and we are composing them in parallel. Suppose FSM A has a transition from state A0 to A1 with transition q, and FSM B has a transition from state B0 to B1 with transition r. If q and r can be simultaneously active (i.e., a parallel interaction), then the algorithm will create a new merged state (A0, B0) and add a transition from (A0, B0) to (A1, B1) on input pattern (q,r). The resulting FSM will then transit to the merged state (A1, B1), which matches the transition to A1 in FSM A and B2 in FSM B. If q and r cannot be simultaneously active, the merged transition is a two-clock-cycle pattern merged by (q,r). Figure 10 shows the whole graph and transitions after merge.

IV. Application in Fault Injection Testing

Our EXERTv2 framework is expanded to the application of fault injection attacks which allows designers to conduct in-depth analyses of potential vulnerabilities from the viewpoint

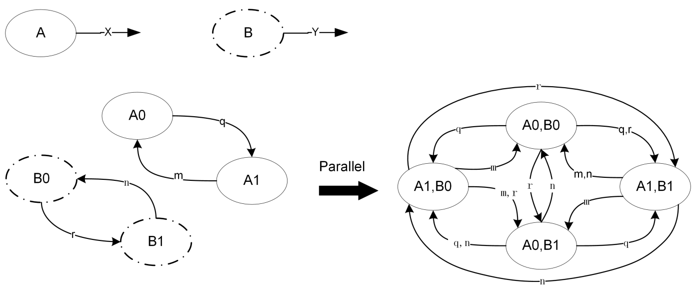


Fig. 10. Illustration of two parallel FSMs merge into a single one by creating merged states and making concatenation of transitions. A,B denote the state registers in Figure 8(a). Solid nodes denote states of A; Dotted nodes denote states of B.

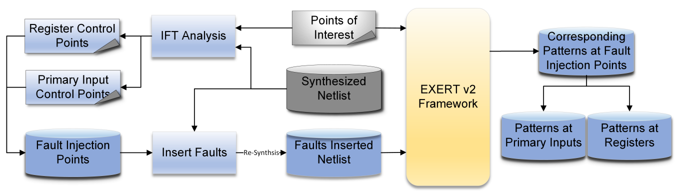


Fig. 11. Overview of fault injection analysis expansion from EXERTv2 framework. The fault injection points are vulnerable control points reported from the IFT Analysis module.

of attackers. The expanded framework is shown in Figure 11. We utilize the IFT Analysis module (Section III-A) to report the vulnerable points to the target interest point, which consists of Register control points and Primary Input control points. After locating these vulnerabilities, we designate them as the target locations for fault injection. Then we insert faults to the target netlist (shown in Figure 12), where additional inputs are added at the register fault injection point. During re-synthesis, the flip-flops situated at the register points where faults are injected are removed along with any associated logic that is no longer relevant, while the added inputs (Green Input) and primary inputs (Grey Input) that have been intentionally subjected to faults are retained in the design. This process aims to prepare the target benchmark for our EXERTv2 framework by introducing faults exclusively as the primary inputs because our framework exclusively generates patterns for these primary inputs. After this, the fault-inserted netlist along with the target interest points are fed into our EXERTv2 framework to generate corresponding patterns at primary inputs fault injection ports and registers fault injection points.

V. Evaluation

We applied our EXERT framework to generate exhaustive patterns that trigger Trojans on multiple Trust-Hub benchmarks [31] and that exploit escalation in a RISC-V processor. Besides, EXERTv2 is applied on two additional ciphers as multiple FSMs are detected during analysis. We used an Intel(R) Xeon E5-2450L 32 cores CPU with 128GB memory operating at 1.80GHz for synthesis and running EXERT. Table II summarizes the Trojan benchmarks, their sizes, and our results.

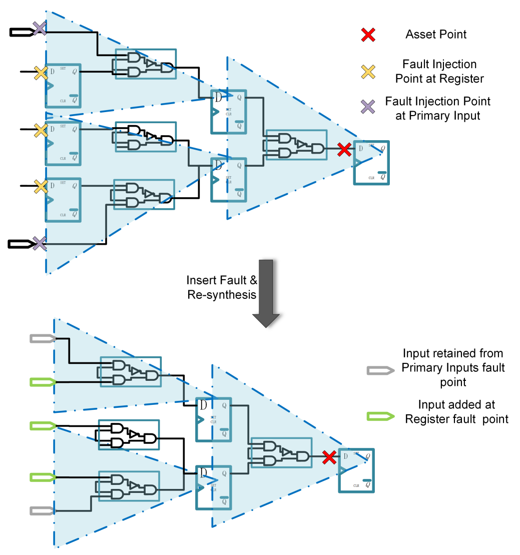

Asset Point
Fault Injection Point at Register
Fault Injection Point at Primary Input
Insert Fault & Re-synthesis
Input retained from Primary Inputs fault point
Input added at Register fault point

Fig. 12. Illustration of Inserting Faults. The red cross denotes the target asset point of interest. The yellow cross denotes the fault injection point at the registers. The purple cross denotes the fault injection point at the primary inputs. Green input denotes inserted inputs at register fault injection points. Grey input denotes primary inputs fault injection points.

A. *AES-T1400*

The first Trojan benchmark is AES-T1400. AES-T1400 Trojan's trigger circuit is composed of a finite state machine (FSM) and it triggers when a sequence of plaintexts is observed. After synthesis, the Trojan triggering module is formulated in the netlist and the triggering condition is hard to extract. By applying our framework, not only are the four plaintexts detected but also the right sequence from the netlist level of design.

In order to test the exhaustiveness and scalability of our framework, we modify the triggering circuit of AES-T1400 and increase the predefined plaintext sequences. We also incorporate more states into the triggering circuit up to eight states and create 25 modified benchmarks (denoted with '(M)' in Table II) by increasing the sequence length of predefined plaintext. Nevertheless, we modified the sequence length to 400 cycles with 600 predefined patterns to test the worst-case run time (denoted with Max), as a maximum of unrolling on AES benchmark in prior work with commercial tools is 400 clock cycles [32]. Then we apply EXERT on all these AES-T1400(M) Trojan benchmarks as well as AES-T1400(Max) and record the run time. As illustrated in Figure 13(a) and Table II, the run time shows a linear growth with the increase of sequence length, along with a minor growth trend when we increase the number of predefined patterns and keep the length the same.

Besides, to better validate and quantify the scalability of our framework, the maximum memory usage is recorded for all AES-T1400(M) benchmarks. As illustrated in Figure 14(a), the maximum memory usage remains approximately the same level as we increase the size of the benchmark. In addition, the maximum memory usage for AES-T1400(Max) is 1463.26Mb, which is still scalable with a maximum of 400 cycles of sequence. This is because our space complexity analysis shows linearity with respect to the number of registers in a single level of FFs. As the size of the circuit increases, the number of registers and fan-in blocks may also increase, but in a single level, the number of fan-in blocks should remain relatively constant. Therefore, the EXERT approach remains linear with respect to the overall size of the circuit, making it scalable and able to handle larger and more complex designs.

B. *PIC16-T200*

PIC-T200 manipulates the instruction register of the PIC microprocessor and is triggered by counting the number of instructions executed. The Trojan payload of PIC-T200 is triggered by a predefined instruction set read from the microprocessor's RAM [33], which requires RAM initialization. Therefore, in order to apply our EXERT framework on PIC-T200 benchmark, we create a 4-bit input bus where the microprocessor reads the instruction bits to bypass the RAM initialization. Note that our current framework is not able to read predefined RAM instructions and reading instructions from primary input will not affect the correctness of pattern generation. The maximum memory usage is also recorded in Figure 14(b). The memory usage remains approximately the same level as we increase the size of the benchmark, which shows similar results with AES-T1400(M). This result further validates that the space complexity of this approach is linear with respect to the number of registers in a single level of FFs.

Note that our framework only detects the shortest activating paths of the Trojan. In normal operation, the majority of instructions will not trigger the counter and create don't care states in the Trojan's activation paths. EXERT automatically discards don't care states as there is no information flow to the malicious counter. Similar to AES-T1400, we modify the Trojan triggering circuit size by changing the counter overflow threshold and length of instruction set. We created 25 benchmarks (denoted by PIC16-T200(M)) and the run time results are shown in Figure 13(b) and Table II. Once again, linear growth is observed.

C. *s38417-T100*

Unlike previous benchmarks from Trust-Hub, s38417-T100 is scan-chain inserted netlist and the Trojan is triggered by a sequential counter in functional mode. The s38417-T100 Trojan payload enables the scan enable signal of a part of one scan chain in the functional mode which allows an adversary to leak internal signal values. Scan chain testing utilizes scan flip-flop to access any registers by shifting in any input vector. However, such scan chain access creates additional unauthorized information flows through a whole scan chain, resulting in false positive control points. In order to apply EXERT on designs with scan chain access while reducing false positives from scan flip-flops, we discard the information flows from the scan chains. In our analysis, we identify the scan-in port of the scan flip-flop and discard its fan-in register as a control point. Table II shows that all 16 triggering patterns are detected in less than 6 minutes.

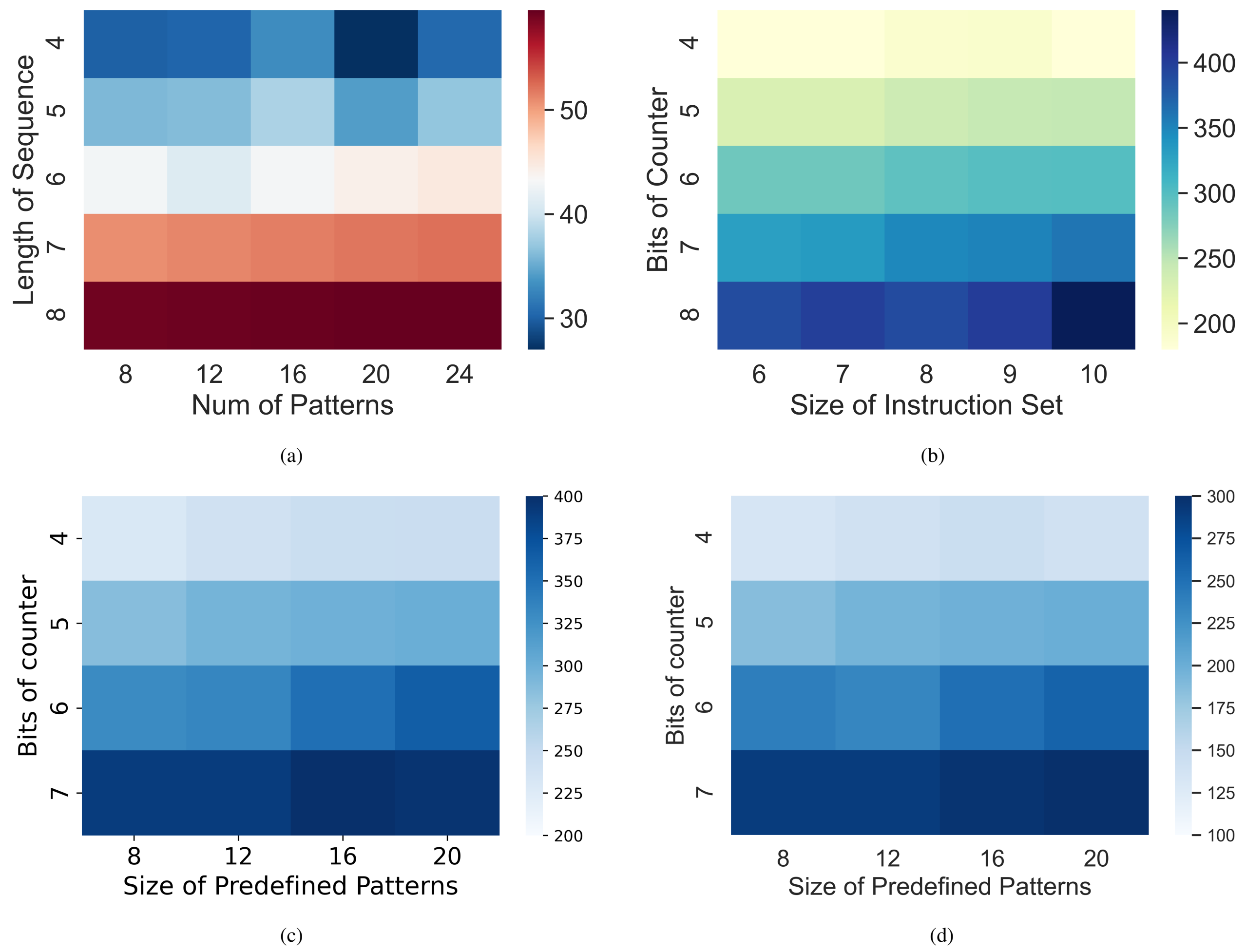


Fig. 13. Heatmap of runtime result (seconds): (a) AES-T1400(M); (b) PIC16-T200(M); (c) Subterranean(M); and (d) SAEAES(M).

D. MC8051-T400

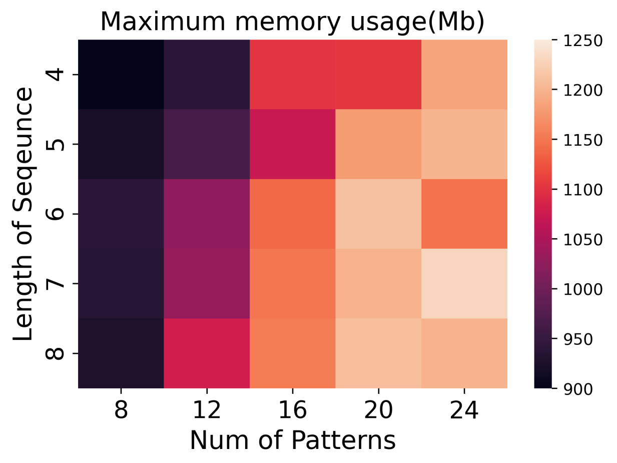
We apply EXERT on another Trojan benchmark, MC8051-T400, to further test its scalability on processor designs. The Trojan in MC8051 is triggered when a specific sequence of commands is executed and the Trojan payload disables interrupt handling after activation. Similar to PIC16-T200, MC8051-T400 also requires a RAM initialization to execute instructions. EXERT successfully detects the predefined sequence of instructions. Note that MC8051-T400 is the largest microprocessor benchmark on Trust-hub with more than 2000 gates. EXERT maintains scalability on two benchmarks with similar triggering conditions while having huge differences in benchmark size.

E. Subterranean Cipher

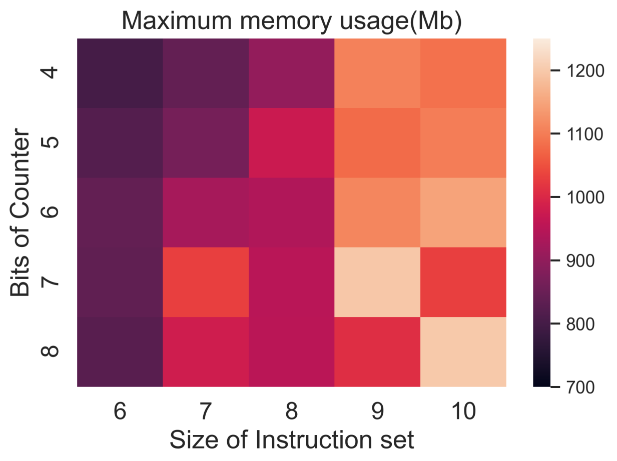
The Subterranean encryption benchmark is a cryptographic algorithm that was developed as part of the NIST Lightweight Cryptography Competition [34]. We apply EXERTv2 on the Subterranean encryption benchmark as we detect two FSMs in this cipher, including a control unit made of 6 registers and an encryption core made of 257 registers that perform round and duplex functions. In order to generate meaningful patterns, we incorporate a counter-triggered Trojan into the subterranean cipher and modify its counter size and instruction set as in the prior section. The Trojan payload leaks one byte of keys directly to the output. Our FSM analysis module in EXERTv2 detects three FSMs (the Trojan activation counter makes the third FSM), which are successfully integrated as a single one. Our FSM integration module is performed offline which takes 148.85 seconds. The runtime results are shown in Table II and Figure 13(c).

F. SAEAES Cipher

SAEAES (Short Authenticated Encryption with AES) is another cryptographic algorithm that was developed as part of the NIST Lightweight Cryptography Competition. The SAEAES is based on SAEB and AES, which is a mode of operation that extends a block cipher into authenticated encryption cone [35]. We utilize the same technique to incorporate a counter-triggered Trojan into the subterranean cipher and modify its counter size and predefined pattern set, creating 16 modified benchmarks to test the runtime results. Since we detect three FSMs, the SAEAES control FSM made of 10 registers, an 8-round unit made of 512 registers that perform keys schedule and expansion, and a key update FSM made of 8 registers. EXERTv2 is used which takes 178.60 seconds for the integration of FSM. It takes more time in FSM



(a)



(b)

Fig. 14. Maximum memory usage result (Mb) for AES-T1400(M) and PIC16-T200.

TABLE III
RESULTS OF EXERTV2 ANALYSIS ON RISC-V PRIVILEGE ESCALATION.

Benchmark	Asset Point	Sequential Length
RISC-V	CSR	115 (Median)
Fault inserted RISC-V	CSR	3

integration compared with the runtime of Subterranean Cipher as three FSMs integration requires both series and parallel compositions. The total runtime results are shown in Table II and Figure 13(d).

G. RISC-V Privilege Escalation and Fault Injection Testing

EXERT can also generate patterns on much larger benchmarks like RISC-V (25786 gates after synthesis, which can further test EXERT's scalability. To better establish a secure hardware platform, RISC-V is designed with four privileged levels (User, Machine, Supervisor, and Reserved level) that aim to provide isolation between different components of instruction stacks. Any attempts to perform operations not permitted by the current privilege mode will cause an exception to be raised [36] while only legal instructions could change the privileged state. As each privileged state is encoded with a Control and Status Register (CSR), we set the Machine mode CSR as our asset to apply the EXERT framework since Machine mode is supposed to have the highest level of security. Our EXERT framework successfully detects more than 76 control points that potentially break the isolation policy between different levels of privileged states with 1769 patterns in 75 minutes. Note that such patterns cannot be generated by sequential ATPG as the register depth exceeds its abort threshold.

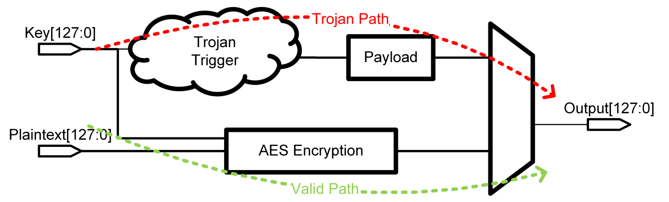


Fig. 15. Activation paths of AES encryption module. The red line denotes Trojan path and the green line denotes the valid encryption path.

To perform the privilege-level escalation attack more easily, we experimentally perform fault injection testing at vulnerable points with our fault injection framework from Section IV. By selecting two register control points as fault injection locations, we successfully obtain two patterns of 3 clock cycles that directly set the CSR registers and cause the privilege escalation. The comparison of patterns generated with/without fault injection is shown in Table III. Since the faults are introduced at control register points situated in close proximity (in terms of level distances) to the asset CSR, the associated fault patterns exhibit significantly shorter sequential length (number of clock cycles).

H. Run Time Comparison with Jasper Gold SPV

In this section, we apply Jasper Gold SPV on AES-T1400 and PIC16-T200 Trojan benchmarks to compare the run-time result with EXERT. The AES-T1400 benchmarks consist of Trojan paths and Valid paths. In JasperGold SPV, we set the complete bus of key input ports as the source and one bit of the output port as the destination to verify whether a sensitization path exists for the Trojan paths. Figure 15 illustrates how we apply JasperGold on the Trojan Path to validate a pattern. However, as a policy-driven validation tool, JasperGold requires us to hardcode the generated pattern as policy insertions so that it can keep generating different patterns. The result shows JasperGold takes 60.06 seconds to validate a path with a *single pattern*. On the other hand, EXERT can generate *all 12 patterns* in only 46.78 seconds. As for PIC16-T200, Jasper Gold requires 76.86 seconds to validate a *single pattern* on a path while our EXERT framework only takes 453.90 seconds to generate *all 729 patterns*. We also validate the patterns of the Subterranean and SAEAES cipher to compare the EXERTv2 runtime results. JasperGold takes 240.40 seconds to validate a single pattern for Subterranean and 330.48 seconds for SAEAES.

VI. Conclusion and Future Work

In this paper, we proposed a novel framework called EXERT which performs exhaustive analysis to report all patterns causing information flows without the need for white-box knowledge of the IP. We have experimentally validated our framework by applying it to Trojan benchmarks from Trust-Hub as well as a large RISC-V processor benchmark. EXERT exhaustively analyzes integrity violations and provides patterns while remaining scalable.

Our initial EXERT framework utilizes FSM analysis to revolve looping information flows in feedback nets [37]. However, these FSMs can interact with each other which requires constraining them with graph transition modeling. With EXERTv2, we propose the FSM integration algorithm that analyzes multiple FSMS as a whole, reducing the complexity of developing an intra-FSM constraining algorithm. Moreover, as mentioned earlier, our EXERT framework also has the potential for real-time applications. In future work, we plan to use EXERT to build monitors that filter illegal patterns during execution.

VII. Declarations

Ethical Approval Not applicable.

Competing Interests Not applicable.

Authors' Contributions Authors A and B contributed text and literature reviews to the paper. All authors reviewed the paper.

Funding This research was partially supported by NSF under award # 2016624 and AFOSR under Award ID FA8650-20-C-1719.

Availability of Data and Materials Not applicable.

References

[1] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[2] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 67–70, 2011.

[3] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 973–990, USENIX Association, Aug. 2018.

[5] G. K. Contreras, A. Nahiyan, S. Bhunia, D. Forte, and M. Tehranipoor, "Security vulnerability analysis of design-for-test exploits for asset protection in socs," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 617–622, 2017.

[6] R. Buhren, H.-N. Jacob, T. Krachenfels, and J.-P. Seifert, "One glitch to rule them all: Fault injection attacks against amd's secure encrypted virtualization," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, (New York, NY, USA), p. 2875–2889, Association for Computing Machinery, 2021.

[7] S. Roy, S. K. Millican, and V. D. Agrawal, "Training neural network for machine intelligence in automatic test pattern generator," in *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, pp. 316–321, 2021.

[8] W. Hu, D. Mu, J. Oberg, B. Mao, M. Tiwari, T. Sherwood, and R. Kastner, "Gate-level information flow tracking for security lattices," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, nov 2014.

[9] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 1691–1696, 2017.

[10] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," *SIGARCH Comput. Archit. News*, vol. 43, p. 503–516, mar 2015.

[11] J. Cruz, F. Farahmandi, A. Ahmed, and P. Mishra, "Hardware trojan detection using atpg and model checking," in *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, pp. 91–96, 2018.

[12] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor, "Hardware trojan detection through information flow security verification," in *2017 IEEE International Test Conference (ITC)*, pp. 1–10, 2017.

[13] "Jasper security path verification app." https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/.

[14] L. Goldstein and E. Thigpen, "Scoap: Sandia controllability/observability analysis program," in *17th Design Automation Conference*, pp. 190–196, 1980.

[15] H. Salmani, "Cotd: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Transactions on Information Forensics and Security*, vol. 12, pp. 338–350, 2017.

[16] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, (New York, NY, USA), p. 85–96, Association for Computing Machinery, 2004.

[17] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware information flow tracking," *ACM Comput. Surv.*, vol. 54, may 2021.

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

[19] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.

[20] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved sat-based atpg: More constraints, better compaction," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 85–90, 2013.

[21] T. Meade, J. Portillo, S. Zhang, and Y. Jin, "Neta: When ip fails, secrets leak," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, (New York, NY, USA), p. 90–95, Association for Computing Machinery, 2019.

[22] L.-T. L.-T. Wang, "Chapter 3 - design for testability," in *Electronic Design Automation* (L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, eds.), pp. 97–172, Boston: Morgan Kaufmann, 2009.

[23] L. M. Reimann, L. Hanel, D. Sisejkovic, F. Merchant, and R. Leupers, "Qflow: Quantitative information flow for security-aware hardware design in verilog," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, (Los Alamitos, CA, USA), pp. 603–607, IEEE Computer Society, oct 2021.

[24] X. Guo, R. G. Dutta, J. He, M. M. Tehranipoor, and Y. Jin, "Qif-verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 91–100, 2019.

[25] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "Avfsm: A framework for identifying and mitigating vulnerabilities in fsms," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[26] "The glucose sat solver." https://www.labri.fr/perso/lsimon/glucose/.

[27] "Abc: A system for sequential synthesis and verification." http://people.eecs.berkeley.edu/~alanmi/abc/.

[28] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

[29] "pyaiger: A python library for manipulating sequential and combinatorial circuits.." https://github.com/mvcisback/py-aiger.

[30] Z. Xin-feng, W. Jian-dong, L. Bin, Z. Jun-wu, and W. Jun, "Methods to tackle state explosion problem in model checking," in *2009 Third International Symposium on Intelligent Information Technology Application*, vol. 2, pp. 329–331, 2009.

[31] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pp. 471–474, 2013.

[32] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[33] "Pic16f84a data sheet." https:ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf.

[34] J. Daemen, P. M. C. Massolino, A. Mehrdad, and Y. Rotella, "The subterranean 2.0 cipher suite," *IACR Transactions on Symmetric Cryptology*, vol. 2020, p. 262–294, Jun. 2020.

[35] Y. Naito, M. Matsui, T. Sugawara, and D. Suzuki, "Saeb: A lightweight blockcipher-based aead mode of operation." Cryptology ePrint Archive, Paper 2019/700, 2019. https://eprint.iacr.org/2019/700.

[36] "The risc-v instruction set manual." https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf.

[37] 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.