QuEST: Quantitative Entropy based Security and Trojan Detection Framework for Confidentiality Verification

Jiaming Wu and Domenic Forte
ECE Dept., University of Florida
Email: *jiaming.wu@ufl.edu* and *dforte@ece.ufl.edu*

***Abstract*—Modern semiconductor design heavily relies on the integration of IPs from 3PIP vendors to improve design efficiency and reduce time to market. However, such collaboration introduces security concerns, including unintentional bugs and opportunities for adversaries to insert hardware Trojans. Confidentiality verification is widely applied for detecting design weaknesses capable of leaking sensitive information through output ports of a chip or IP module. In this paper, we present QuEST, a novel confidentiality verification framework that identifies data leakage by analyzing statistical dependencies between multiple input and output ports. Moreover, QuEST augments traditional leakage detection techniques through Shannon entropy-based metrics, most notably mutual information and conditional mutual information, to quantify the extent of data leakage. This quantitative feature enables designers to systematically verify and assess security vulnerabilities more effectively that existing approaches. Experiments show that QuEST successfully detects data leakage caused by hardware Trojans in 11 Trust-hub benchmarks. In general, QuEST serves as a promising confidentiality analysis tool that enables designers to detect and quantify data leakage, thus bolstering the security posture of modern hardware designs.**

I. Introduction

Incorporating 3PIP vendors is a common approach in modern hardware design life-cycle. This strategy enables the SoC design house to reduce development time and costs by leveraging IP blocks built and tested by specialized vendors rather than designing each component from scratch. However, the integration of 3PIPs into the design of SoCs poses increasing security risks, which have become a growing concern in the industry. Notably, as third-party vendors could possibly insert malicious operations into their IP, such as hardware Trojans and backdoor access points. These hardware Trojans can leak sensitive information, raise user privileges, reduce system reliability, and introduce a variety of other critical security vulnerabilities.

The confidentiality principle in security aims to ensure that sensitive information within a system remains protected and do not leak to unauthorized entities [1]. Confidentiality analysis is widely utilized to detect hardware Trojans, as common Trojan payloads are designed to leak secret information to the output. In prior research, confidentiality-based hardware Trojan detection has primarily been implemented through two main categories: path-oriented information flow tracking and formal verification methods. These techniques focus on monitoring data flows, verifying security properties, and analyzing system behavior to identify potential information leaks that indicate the presence of hardware Trojans.

Information flow tracking (IFT) focuses on detecting potential data leakage by systematically tracing data flows from a specific input to a specific output. IFT is developed to perform on different levels of abstraction, including gate-level and RTL level depending on the tracking methods tailored to each abstraction level. Hu et al. [2] proposed the fundamental theorem of gate-level information flow tracking (GLIFT), a method that monitors propagations of individual bits through logic functions at the gate level. GLIFT addresses confidentiality properties by assigning security labels to data (tainted or not tainted) and propagating the security labels through shadow logic, additional logic that mirrors the primary circuit and computes the security label at the outputs. Yu et al. introduced Multi-Flow [3], an advanced version of GLIFT designed to track simultaneous information flow behaviors on multiple bits at the gate level. Multi-Flow achieves this by expanding both the security lattice and the shadow logic to accommodate a wider label width to manage the propagation of multiple tainted bits concurrently.

Quantification in IFT enables the measurement of the extent to which sensitive information propagates through a system. In recent work, Lennart et al. introduced QFlow [4], a quantitative tool designed to assess information flow leakage and detect hardware Trojans at the RTL level. QFlow parses Verilog code into a bind tree graph structure, where outputs serve as the root nodes, and inputs as the leaf nodes. QFlow then performs its quantitative analysis using Posterior Bayes Vulnerability (PBV) [5] on the parsed bind tree structure. Guo et al. proposed QIF-Verilog [6], an extended Verilog hardware description language designed to assess the reliability of hardware systems. QIF-Verilog introduced a quantitative information flow model using an extension of the Verilog type system to label sensitive signals and to evaluate leakage at compile time.

Recent research also focused on the detection of hardware Trojans and the analysis of confidentiality in hardware systems using formal verification methods. Instead of physically exploring the path, the formal method detects confidentiality violations through Bounded Model Checking (BMC), which checks for a binary answer of whether critical information get leaked or not. Jeyavijayan et al. [7] proposed a formal verification framework to detect hardware Trojans that corrupt data by defining data corruption properties and validating properties using ATPG (Automatic Test Pattern Generation) [8]. Although existing confidentiality frameworks have made their efforts to effectively detect hardware Trojans, they still have certain limitations and drawbacks.

In this paper, we introduce a novel statistical verification framework which detects confidentiality violations through dependency checking and quantifies them using entropy-based calculations. Unlike previous work, our framework defines confidentiality violations based on the dependence of outputs on inputs and introduces the Shannon entropy for quantification. Our contributions are summarized as follows:

- We establish a foundation for confidentiality analysis by detecting data leakage through an examination of dependency relationships between the input and output ports. We propose the formal definition of dependency in mathematical theorems.
- We apply Shannon entropy-based information theory calculations, including mutual information and conditional mutual information, to quantitatively analyze dependencies between the inputs and outputs under evaluation. This approach enables designers to rigorously measure the strength of leakage on the output side.
- We further develop a quantification analysis mechanism based on entropy calculations to systematically quantify the "leakage control" points of the target design, enabling identification of the root causes for design weaknesses and Trojan triggering.
- Our proposed QuEST framework is performed on 11 Trojan benchmarks from Trust-Hub and successfully detects their data leakage.

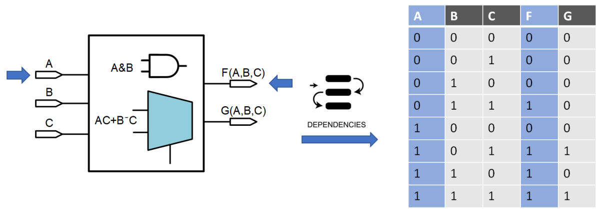
The remainder of the paper is organized as follows. Section II provides necessary background and threat model. Section III introduces the state-of-art techniques and their limitations. In Section IV we propose the novel QuEST statistical dependency verification framework as well as its mathematical proofs. Section VI analyzes the results when our framework is applied on multiple Trojan benchmarks and in Section VII we compare our QuEST with previous work. Finally, we conclude with limitation and future work in Section VIII.

II. Preliminaries

In this section, we provide an overview of the threat model and then introduce the necessary concepts of hardware Trojan and statistical dependence modeling.

A. Threat Model

We consider all third-party vendors with access to the IP as potential adversaries capable of inserting hardware Trojans. These vendors include IP vendors, DfT vendors, and third-party EDA tool suppliers. Each of these entities has privileged access to critical aspects of the IP, making them viable insertion points for malicious modifications or hardware Trojans that could compromise functionality or leak sensitive information. In our framework, we consider the SoC design house as the defender, responsible for conducting verifications to detect potential weaknesses introduced by adversaries. We assume that the designer has access to the netlist file but only possesses black-box knowledge of the purchased IP, as vendors often mask the internal structure to safeguard their intellectual property. Detecting weaknesses in netlist abstraction would be extremely difficult as a netlist only consists of basic logic gates that are not human readable after synthesis.



A	B	C	F	G
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

Fig. 1. Circuit model for statistical dependence. The input and output are treated as discrete variables. The output F is dependent on A given the statistical distribution.

B. Hardware Trojans

Hardware Trojans are malicious modifications introduced into the design or manufacturing process of integrated circuits with the goal of compromising their security, functionality, or reliability [9]. These Trojans often remain hidden and inactive during standard operations, but under specific conditions, they can be triggered to perform harmful actions. The motivation behind embedding hardware Trojans varies, but common objectives include espionage, IP theft, denial-of-service, or bypassing security mechanisms.

A hardware Trojan is constructed by two main components: the Trojan trigger and the Trojan payload. The primary objective of the Trojan payload is often to leak sensitive information, thereby violating the confidentiality security property of the system. As a result, confidentiality verification becomes an effective approach for detecting hardware Trojans, as it can identify unauthorized data leakage and assess the integrity of the design regardless of the trigger condition. Later we test on hardware Trojan benchmarks from Trust-Hub [10] that leak encryption keys to the outputs in our experiment setup.

C. Circuit Model

Our primary objective is to detect information leakage through dependency checks. In general, the probability of an event is calculated by summing the probabilities of all outcomes that satisfy the event's criteria. In the context of our circuit model – where inputs and outputs are treated as discrete variables in $\{0,1\}$ – this means identifying which outputs correspond to specific primary input values in the sample space of its truth table, as shown in Figure 1. Therefore, to compute the probability density function for the circuit, one must count occurrences of 0 and 1 and enumerate all relevant subsets of the input and output variables. The example shown

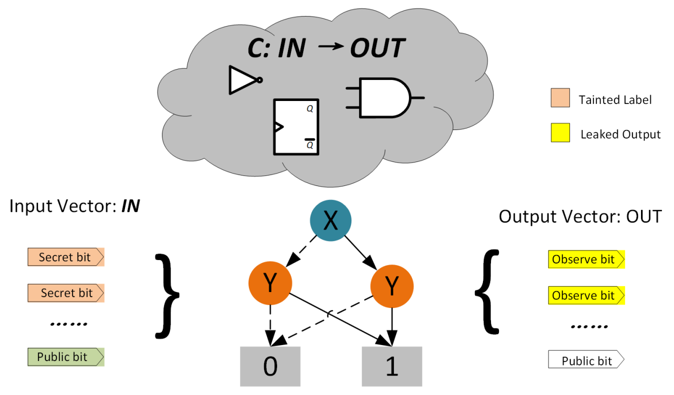


Fig. 2. Dependency model based on BDD diagram. The circuit is parsed and the dependency is calculated based on input vector and output vector.

TABLE I
SUMMARY OF LEAKAGE MEASUREMENTS IN SIDE-CHANNEL ANALYSIS

Metric	Definition	What it Measures
Mutual Information	$I(A;B) = \sum P(a,b) \log\left(\frac{P(a,b)}{P(a)P(b)}\right)$	Average-case leakage
Maximal Leakage	$\mathcal{L}(A \to B) = \log \sum_b \max_a P(b\|a)$	Worst-case gain in guessing
g-Leakage	$L_g = \log\left(\frac{\max_{\hat{a}} \mathbb{E}[g(A,\hat{a})]}{\mathbb{E}_B[\max_{\hat{a}} \mathbb{E}[g(A,\hat{a})\|B=b]]}\right)$	Task-specific adversarial leakage

in Figure 1 demonstrates that the output F exhibits statistical dependency on the input A. This is calculated by treating the inputs and outputs as discrete variables and compares their marginal probability distributions and joint probability distribution, as discussed in detail in Section IV

which is detaily exxplained in Section IV

To make this approach more practical, rather than relying on truth-table–based calculations (which require circuit simulation), we use the high-performance Sylvan [11] MTBDD package, leveraging its built-in functions to compute dependencies. To achieve this, we model a deterministic circuit C using the Binary Decision Diagram (BDD) with discrete variables. The circuit C is derived from an $m \times n$ Boolean function, represented in vector form as $A_m \to F_n$, where A denotes the set of inputs, F represents the set of outputs, m is the number of inputs, and n is the number of outputs. As a result, the circuit model is illustrated in Figure 2. The inputs and outputs are represented as discrete variables in vector form, and their dependency can be computed.

D. Statistical Dependence-based Verification

Statistical dependency analysis has been used to verify potential vulnerabilities in side-channel attacks by identifying and quantifying the leakage of sensitive information through dependent correlations [12], [13]. Moreover, dependency analysis is used to perform probing security verification by evaluating whether intermediate signals remain statistically independent from secret variables under a bounded number of observations [14]. Table I shows three leakage measurement metrics used in side-channel analysis. By leveraging statistical tools to assess the dependence between observed signals and potential probing points, one can determine its resilience to probing-based side-channel attacks. In our QuEST framework, we expand this statistical analysis to detect data leakage caused by hardware Trojans, thereby determining whether such weaknesses exist within a given design.

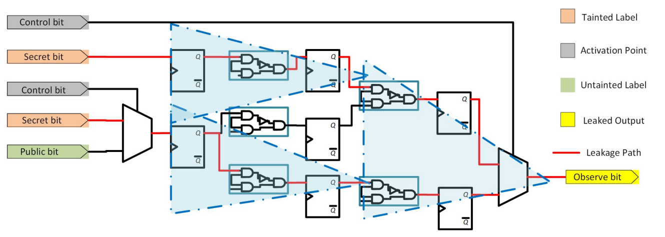


Fig. 3. Path-oriented confidentiality verification. The confidentiality is verified on connectivity from single start point to single end point.

III. Related Work and Limitations

Research on hardware Trojans focuses on both the development and the detection of Trojan, with research works advancing in parallel to enhance both attack and defense perspectives. Traditional hardware Trojan detection methods such as feature analysis-based approaches [15], [16] are comprehensively discussed in [17]. This section explores the limitations of two leading hardware Trojan detection approaches: GLIFT-based information flow analysis and formal verification techniques.

A. Limitation of GLIFT-based Verification

IFT for hardware Trojan detection is inherently a *path-oriented tracking approach* as shown in Figure 3, requiring exploration of connectivity throughout the target design to prove confidentiality violations. This leads to two main limitations:

- It only detects violations originating *from a single input*, leaving potential violations caused by correlations among multiple inputs unverified.
- It is *unable to distinguish between valid and malicious paths* [18]. For example, GLIFT finds that it is normal that the key flows to the ciphertext outputs in cryptographic functions [19]. As long as the passes through all the round operations and diffusion is achieved to make the key non-recoverable, such paths are treated as valid paths to outputs. However, if an adversary designed a Trojan that leaks the key through the ciphertext output without diffusion, the path-oriented tracking approach would not be able to identify it as a malicious path.

The security lattice of IFT approaches also grow exponentially with the number of input ports. This results in another limitation of scalability, leading to a *path explosion problem* during the shadow logic generation, and requires exponentially longer simulation time. To validate the patterns that cause the leakage, actual simulations on the shadow logic are required. This process can be time-consuming and computationally intensive, as it requires comprehensive testing to cover all possible scenarios.

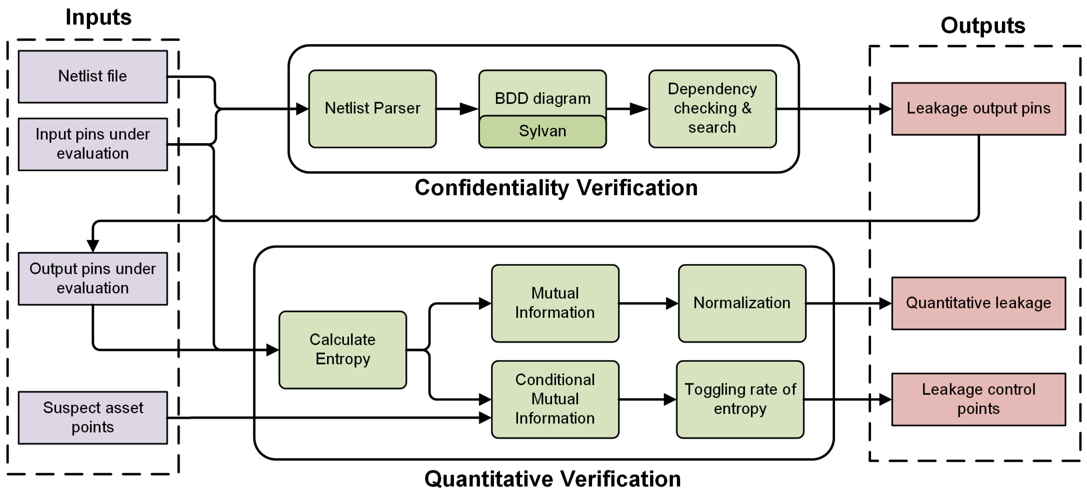


Fig. 4. Overview of QuEST verification framework. (top) Confidentiality verification identifies leakage output pins; (bottom) Quantitative verification reports the amount of leakage at the outputs identified by confidentiality verification.

B. Limitation of Formal Verification

Some commercial formal verification tool shares the same limitations with GLIFT as it also checks confidentiality violations through path-oriented tracking method. For example, the JasperGold Security Path Verification (SPV) tool [20] validates whether security-critical information can be leaked or not, and provides the traces if there is a violation. JasperGold SPV offers a straightforward command, `check_spv -from start_point -to end_point`, which verifies whether information leaks along a specified path from the start point to the end point. However, it inherits the limitation of path sensitization, as it can only detect single-ended paths and requires the designer to validate whether the path is valid or malicious. Traditional formal verification examines confidentiality violations through security property assertions and model checking to determine if the property is violated using, for example, JasperGold FPV. However, this *requires designer's expertise to formally representing security features in property assertions, as well as extensive knowledge of the target design.* Another limitation of formal verification is its scalability. Since formal verification checks all input space vectors for violations of security properties, detecting sequence-triggering Trojans requires unrolling the design, which is exponentially time-consuming. Normally, a formal verification tool, such as JasperGold FPV *can only provide a bounded proof* where a sequential limit is set, and it will abort once this limit is reached.

C. Comparison and Motivation

Compared to path-oriented verification approach, our QuEST framework detects information leakage through dependency checking. Figure 2 shows the statistical model for checking leakage. This approach shifts the focus from tracking the physical paths data might take through the hardware to analyzing the statistical relationships between data elements. First, unlike the path-oriented approach, which can only track leakage through single start-to-end point paths, *our dependency checking method can detect leakage across multiple inputs and outputs simultaneously*, capturing correlated data flows that path-oriented methods might miss. Second, the path-oriented method may miss malicious paths if they share the same start and end points as valid paths. In contrast, our framework's *quantitative dependency feature allows it to distinguish between paths based on quantified dependencies*. Even if they share the same endpoints, their degree of dependency varies. In summary, the motivation behind QuEST's approach is to overcome the limitations of path-oriented tracking, enabling the detection of correlated leakage patterns that traditional methods might overlook, and providing quantitative analysis of security vulnerabilities in hardware designs.

IV. QuEST Verification Framework

In this section, we introduce the QuEST verification framework, which leverages a statistical model to ensure both confidentiality verification and quantitative analysis, supported by mathematical validation. As depicted in Figure 4, the QuEST verification framework consists of two components: **confidentiality verification** and **quantitative verification**. In the confidentiality verification process, the input netlist and target pins are first parsed by a netlist parser to generate a Binary Decision Diagram (BDD). Leveraging the Sylvan library, the framework then performs dependency analysis on the BDD to systematically identify output pins that may leak sensitive information. In quantitative verification, output pins and suspected leakage control points are analyzed by calculating entropy and performing Mutual Information (MI) and Conditional Mutual Information (CMI) analysis. The results are further processed for standardization to quantify leakage and the toggling rate of entropy to identify root-cause-of-leakage points, providing both qualitative and quantitative assessments of confidentiality breaches.

A. *Confidentiality Verification*

The Confidentiality Verification component takes the netlist and secret input pins as inputs and identifies which output pins exhibit leakage by performing dependency checking using a BDD-based statistical model. This section provides a comprehensive explanation of the Confidentiality Verification process, including the mathematical proofs of dependency and the search algorithm for leakage output [11].

1) Dependency Checking Model: The Confidentiality Verification relies on dependency checking to detect and prove leakage. After parsing the netlist and modeling the circuit using a BDD, the input and output pins can be further modeled as discrete random variables, and their dependencies are analyzed based on the probability mass function (PMF) and the joint probability mass function (JPMF). Instead of directly checking for dependence between variables, we mathematically define independence and demonstrate that its violation serves as proof of dependency.

To formalize this, we define the notation $P_r[X = x]$ to represent the probability that a discrete random variable X takes the value x, where x is binary (0 or 1) in the context of the circuit model. Additionally, we define $p_X(x)$ as the probability mass function of the discrete variable X, which assigns probabilities to the possible values of X.

The formal definition of the probability mass function $p_X(x)$ is introduced as

$$p_X(x) = P_r[X = x], \quad \text{where } x \in \{0, 1\},$$

and the joint probability mass function between two discrete variables X and Y is defined as

$$p_{X,Y}(x, y) = P_r[X = x \,\text{and}\, Y = y], \quad \text{where } x, y \in \{0, 1\}.$$

Thus we can introduce the mathematical condition for independence between two discrete variables.

Lemma. Specifically, X and Y are independent if and only if their joint probability mass function equals to the product of their individual probability mass functions:

$$p_{X,Y}(x, y) = p_X(x) \cdot p_Y(y), \quad \text{where } x, y \in \{0, 1\}.$$

This condition provides the foundation for determining whether statistical independencies exist between inputs and outputs, violation of independence demonstrates leakage from inputs to outputs. With this definition, the example process in Figure 1 of detecting leakage can be exemplified as follows: If we want to detect whether leakage exists from A to F, we can simply calculate the probability mass function between A and F by the definition of independency theorem shown above.

In order to verify possible leakage between multiple ports and their correlations, we expand the theorem to vectors. This allows us to analyze the relationships between multiple inputs and outputs simultaneously, providing an assessment of potential data leakage caused by correlated ports. For example, let $\vec{A}$ be a vector of inputs $[A_1, A_2, \cdots, A_n]$, and $\vec{F}$ be a vector of outputs $[F_1, F_2, \cdots, F_n]$. The independence checking theorem can then be improved as discussed next.

TABLE II
EXAMPLE OF JOINT PROBABILITY MASS FUNCTION (JMPF) AND MARGINAL PROBABILITY MASS FUNCTION (PMF) CALCULATION BASED ON TRUTH TABLE FOR INPUTS A_1, A_2 AND OUTPUTS F_1, F_2

A_1	A_2	F_1	F_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Lemma. The inputs and outputs are independent if and only if the joint probability distribution of the input-output pairs equals the product of their individual probability distributions. Mathematically, this is expressed as:

$$p_{\vec{A},\vec{F}}(a, f) = p_{\vec{A}}(a) \cdot p_{\vec{F}}(f), \quad \forall a \in \vec{A},\, f \in \vec{F}.$$

Therefore, the independency checking theorem between vector $\vec{A}$ and $\vec{F}$ applies when the joint probability mass function is equal to the multiply product of the marginal probability mass function for all combinations of a and f.

Assumption. All input variables $\{A_1, A_2, \ldots, A_n\}$ to the system are independent and identically distributed (i.i.d.) random variables following a normal distribution. This assumption reflects the statistical behavior of inputs as commonly represented in truth tables.

As shown in Table II's example, the JPMF is described by:

- $P(\vec{A} = [0, 0], \vec{F} = [0, 0])$
- $P(\vec{A} = [0, 1], \vec{F} = [0, 1])$
- $P(\vec{A} = [1, 0], \vec{F} = [1, 0])$
- $P(\vec{A} = [1, 1], \vec{F} = [1, 1])$

While the marginal PMF is given by:

- $P(\vec{A} = [0, 0]), P(\vec{A} = [0, 1]), P(\vec{A} = [1, 0]), P(\vec{A} = [1, 1])$
- $P(\vec{F} = [0, 0]), P(\vec{F} = [0, 1]), P(\vec{F} = [1, 0]), P(\vec{F} = [1, 1])$

We can then compute $p_{\vec{A},\vec{F}}(a, f)$ with $p_{\vec{A}}(a) \cdot p_{\vec{F}}(f)$ for all combinations. By checking if the JPMF is equal to the multi-product of the marginal PMFs, the output $\vec{F}$ is dependent of $\vec{A}$. By expanding the independency theorem into vectors and using this methodology, we can systematically verify potential data leakage between multiple input and output ports in a circuit netlist.

2) Identifying Leakage Points: In the prior section, we provided the mathematical basis for defining dependency. Building upon this, we now extend the approach to identify all dependent pins, enabling us to explore the complete set of leakage points. The algorithm, presented in Algorithm 1, demonstrates how an exhaustive search of output combinations can reveal leakage points and generate a suspect set for use in our quantitative framework.

The exhaustive search algorithm is designed to systematically analyze dependencies between secret input ports and output ports to verify the confidentiality of a system. It begins by examining subsets of the output ports, starting with the smallest size, For each subset d, the algorithm computes the joint probability and compares it to the product of the individual probabilities. If the probabilities do not match, a statistical dependency is identified, indicating

Algorithm 1 Exhaustive Search Algorithm

```
Input: Set of $n$ secret input ports $A$;
Output: Set of $d$ suspect leakage output ports $F$;
1:  start point: $d \longrightarrow 1$
2:  while True do
3:      for each subset $F'$ of $F$ with size $d$ do
4:          if $P_{A,F}(A, F') \neq P_r[A = 1] \times P_r[F' = 1]$ then
5:              Report dependency found in subset $F'$, $d$ output ports
6:          end if
7:      end for
8:      $d \leftarrow d + 1$
9:      if $d > outputsize$ then
10:         break
11:     end if
12: end while
```

potential information leakage through the subset F . When such a dependency is found, the subset is flagged as part of the suspect set, and the subset size d is incremented to analyze larger groups of output ports. This process continues exhaustively until either all combinations of output ports have been analyzed or the subset size exceeds the total number of output ports. By iterating through all possible subsets, the algorithm ensures comprehensive detection of confidentiality violations, enabling the identification of specific output ports that may leak information about protected inputs.

B. Quantitative Statistical Framework

Our confidentiality framework provides a binary answer to indicate whether suspect output points exhibit leakage, determined by their dependency on the target input points under evaluation. Building upon this, our quantitative framework delves deeper, offering precise metrics to quantify the extent of this dependency, enabling a more comprehensive assessment of potential leakage risks. The leakage points identified by our confidentiality framework can be further analyzed using our quantitative approach, which provides a detailed assessment of the quantified leakage and vulnerability. The general framework is shown in Figure 4. Our framework provides two key quantitative metrics for measuring dependency: **mutual information (MI)** and **conditional mutual information (CMI)**. These metrics, are derived from Shannon Entropy and its extensions in information theory. The quantitative verification framework facilitates both the quantification of information leakage and the identification of leakage control points from a given list of suspect points, e.g., rarely activated nets in hardware Trojan trigger detection.

1) Entropy Calculation: Entropy measures the uncertainty or unpredictability of a random variable in information theory. Entropy also reflects variable dependency, as independent variables exhibit maximum unpredictability, thereby having the highest entropy. The formal definition of entropy is given as follows. For a discrete random variable X with a set of possible outcomes $\{x_1, x_2, \ldots, x_n\}$ and corresponding probabilities $\{p_1, p_2, \ldots, p_n\}$, the entropy $H(X)$ is calculated using the following formula:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

To perform a quantitative analysis application with our framework, we also introduce MI and CMI derived from the basis of entropy definition.

Theorem. The mathematical definition of mutual information $I(X; Y)$ is given as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 \frac{p(x, y)}{p(x)p(y)}$$

MI effectively captures the degree of statistical dependency between the two variables. We apply this metric in quantitative analysis where X represents the inputs and Y represents the outputs. By calculating MI, we can assess the dependency between the inputs and outputs, thereby quantifying the amount of information leaked.

Normalization is essential for better quantification of leakage because MI inherently varies with the size of input and output vectors. When X and Y are perfectly independent, the MI is minimized to 0, On the other hand, when X and Y are perfectly dependent (i.e., Y is a function of X), the mutual information $I(X; Y)$ is maximized. In this case,

$$I(X; Y) = H(X) = H(Y),$$

where $H(X)$ and $H(Y)$ represent the entropy of X and Y, respectively.

For an n-dimensional binary vector X, there are 2^n possible states. The maximum entropy occurs when X is uniformly distributed over all 2^n states:

$$H(X) = \log_2(2^n) = n \,\text{bits}.$$

Thus, the mutual information $I(X; Y)$ reaches its maximum when X and Y are fully dependent, which is equal to the entropy of X (or Y):

$$I(X; Y) = H(X) = n.$$

In order to better quantify leakage, we introduce the **normalized mutual information (NMI)**. The NMI is defined as the mutual information divided by the maximum possible mutual information n:

$$\text{NMI} = \frac{I(X;Y)}{n}.$$

where $I(X; Y)$ is the mutual information between X and Y, and n is the dimensionality of the binary vector $\vec{X}$.

Theorem. The mathematical definition of CMI is given as:

$$I(X;Y|Z) = \sum_{z \in Z} p(z) \sum_{x \in X} \sum_{y \in Y} p(x, y|z) \log_2 \frac{p(x, y|z)}{p(x|z)p(y|z)}.$$

Conditional mutual information $I(X; Y \mid Z)$ quantifies the amount of information a random variable X contains about another random variable Y, given that a third variable Z is already known. In our analysis, X denotes the inputs, Y

TABLE III
APPLICATIONS OF ENTROPY CALCULATION AND THE ASSOCIATED VARIABLE NOTATION.

Entropy Calculation	**Mutual Information**	**Conditional Mutual Information**
Mathematical Expression	$I(X;Y)$	$I(X;Y\|Z)$
Input Variable	X	$X;Z$
Output Variable	Y	Y
Applications	Quantify dependency between X and Y	1) Quantify dependency between X and Y conditioned on Z 2) Identify Z as a "leakage control" point, e.g., Trojan trigger

denotes the outputs, and Z to any signal being evaluated within the hardware system, including inputs, wires, and registers (states). By calculating CMI, we assess how much additional information about the outputs Y is provided by the inputs X, beyond what is already known from the intermediate state Z. This is particularly useful for isolating the contribution of specific inputs or intermediate registers to the observed leakage at the outputs. Higher conditional mutual information indicates that changes in the value of Z result in a higher entropy toggling rate, indicating that Z plays a stronger weight in reducing the uncertainty about Y. This means that Z may possess "leakage control" properties for the circuit, as it increases the information shared between X and Y.

2) Applications of Quantitative Entropy Calculation: The summarized application of the entropy calculation and the associated variable notation is shown in Table III. The quantitative verification component serves a dual purpose: Assessing information leakage and pinpointing potential vulnerabilities within the circuit. By measuring the mutual information between the input pins and output pins, the approach quantifies how much information about the inputs is carried through to the outputs, thus signaling and quantifying leakage.

In addition, the framework takes a set of suspect leakage control points as input to further analyze the source of a design's confidentiality weaknesses. Specifically, it evaluates the CMI of each suspect, which represents how much additional information leakage arises from specific signals once the suspect has been accounted for. Then a threshold for CMI is applied to highlight the points whose leakage contribution is particularly high. If any point exceeds this threshold, it suggests that this point is more vulnerable to system security, thus being labeled a leakage control point. This allows designers to identify critical vulnerable points with quantitative measurements of entropy without the expertise of the design.

V. IMPLEMENTATION OF AN AUTOMATED FRAMEWORK

Subsequently, the entropy calculation metric is developed based on the Sylvan MTBDD library, with state-of-art multi terminal binary decision diagram package in C++. To support automated entropy calculation, we extend Sylvan with a dedicated framework that incorporates a suite of information-theoretic computations such as marginal probability, joint probability, mutual information, and conditional entropy. To maintain compatibility and performance, all extensions are implemented using Sylvan's TASK macros, which support parallel recursive computations. All metrics are defined as recursive functions operating on MTBDDs. We utilize Sylvan's built-in caching interface to store intermediate results and avoid recomputation. Each task takes one or more nodes along with the number of variables as input, and returns a double value representing a probabilistic or information-theoretic quantity.

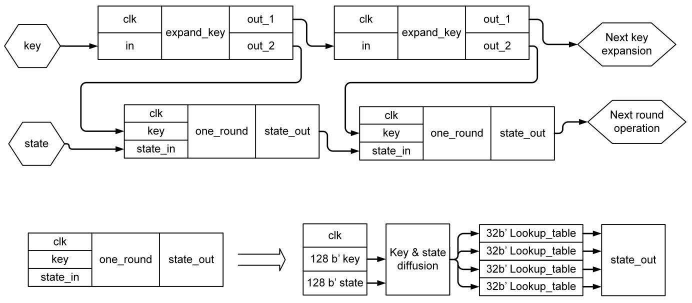


Fig. 5. Synthesized diagram of AES netlist. The expand key and round modules are each identical and duplicated for 10 rounds with symmetric connectivity.

VI. EVALUATION

We applied our QuEST framework on multiple Trust-Hub benchmarks [21] to evaluate its application in Trojan detection. Our QuEST framework applies on gate-level netlists while most benchmarks are in RTL level. Thus, we begin by synthesizing all benchmarks using the open-source Yosys [22] tool, thereby generating a library-independent netlist suitable for our netlist parser. Table IV summarizes the results for different Trojan benchmarks.

A. Scaling to AES Encryption IPs

Encryption algorithms like AES and DES typically operate on 128-bit inputs, comprising a 64-bit plaintext and a 64-bit key, to produce 64-bit outputs, known as ciphertext. Running an exhaustive search on all 128-bit combinations is computationally infeasible due to the sheer scale of possibilities. To address this challenge, our framework deconstructs the AES netlist into its 10 rounds and verifies each round symmetrically. We hypothesize that, as long as each round operates independently, the symmetric arrangement of the round modules ensures inherent independence between the key inputs and the ciphertext outputs. Each round consists of four lookup tables that operate in parallel in 32-bit segments. As every round module is identical except for the final round, there is no need to verify 10 separate instances of the same module. Instead, we can focus on verifying a single representative round and the distinct final round and ensuring that they are symmetrically connected.

TABLE IV
AES-Trojan and RSA-Trojan benchmark leakages. # Detected / # Actual denote the number of detected and actual leaked asset inputs reported from confidentiality verification; Normalized MI denotes the normalized mutual information reported from quantitative verification;# of independent inputs denotes the number of input pins that are not leaked.

Benchmark	Confidentiality Verification	Quantitative Verification	# of Independent Inputs	Time (s)	Trigger	Payload
	# Detected / # Actual	Normalized MI				
AES-T100	8/8	0.4438	120	246	always on	covert channel
AES-T200	8/8	0.4438	120	266	always on	covert channel
AES-T400	128/128	0.4576	0	445	sequence	RF signal
AES-T700	8/8	0.3587	120	234	sequence	covert channel
AES-T800	8/8	0.3587	120	232	sequence	covert channel
AES-T900	8/8	0.3526	120	212	counter	covert channel
AES-T1200	8/8	0.4688	120	241	counter	covert channel
AES-T1600	128/128	0.2220	0	448	counter	RF signal
AES-T1700	128/128	0.2955	0	465	counter	RF signal
RSA-T100	32/32	0.7925	0	346	plaintext	ciphertext
RSA-T300	32/32	0.7925	0	402	counter	ciphertext

To establish the security of the AES implementation, we first verify that the lookup tables are secure against vulnerabilities. We find that the *mutual information for a single lookup table of a Trojan-free AES is calculated to be 0.0475, which is effectively negligible (close to zero).* Next, we ensure that the connectivity between components exhibits sufficient diffusion, effectively propagating changes in the input across the entire state. By proving the security of the lookup tables and the diffusive nature of the connectivity, we can validate that there is no dependence between the key input to the ciphertext output. This approach incorporates the netlist into a graph-based representation and analyzes its symmetric modules using the schematic.

In summary, By breaking down the verification process into smaller and manageable components, we achieve scalability while maintaining a robust evaluation of the functionality and security of the system.

B. AES Trojan Benchmarks

We further applied the QuEST framework to evaluate multiple AES benchmarks with Trojan insertions. To streamline the evaluation process and reduce computational overhead, we simplified the AES operation from 10 rounds to 2 rounds. This reduction was sufficient to capture the critical diffusion properties of the cipher while maintaining the ability to assess Trojan insertion. The results are shown in Table IV. The first benchmark we tested is AES-T100, which leaks its 8-bit secret key inputs directly to 8-bit outputs with no activation needed. The AES-T100 Trojan payload leaks its secret keys through a covert channel. A pseudo-random number generator produces a sequence that is XORed with 8 bits of the key, creating a modulated signal that carries side-channel information. This signal is then transmitted through the leakage circuit using CDMA communication. For confidentiality verification, we labeled the 32-bit input pins, identifying those that carry the 8-bit secret key. Our framework successfully detects the confidentiality violation introduced by the Trojan circuit by identifying the dependency between the covert channel and the 8 bits of the secret key, thereby effectively detecting the hardware Trojan within the AES encryption process. This verification process demonstrates the robustness of our framework in isolating and quantifying leakage points, ensuring that even subtle information leaks can be accurately detected. The ability to identify these leakage points within the ciphertext output is crucial for evaluating the security of AES implementations, particularly in the presence of Trojan insertions.

After identifying the leakage output pins, quantitative verification is employed to assess the extent of information leakage from the inputs to the outputs. We demonstrate that *through the covert channel, a mutual information calculation of 0.4438 is detected, indicating a significant leakage of sensitive data.* In contrast, for the Trojan-free AES, the mutual information between the inputs and outputs is nearly zero, highlighting the absence of such leakage under normal conditions.

We tested AES-T200, AES-T700, AES-T800, and AES-T1200 benchmark. These benchmarks are tested with our QuEST framework as shown in Table IV. Note that, since both benchmarks share the same Trojan payload, the mutual information calculation remains nearly identical. This consistency indicates that both systems leak a similar amount of information, which can be attributed to their having the same Trojan payload. The primary distinction between the two Trojans lies in their activation mechanisms. AES-T700 is triggered when a predefined state sequence is observed, whereas AES-T900 becomes active by counting the preset number of encryption operations. The activation method slightly changes the mutual information calculation.

The AES-T400, AES-T1600, and AES-T1700 Trojan benchmarks focus on the leakage of secret keys through the RF (Radio Frequency) channel, and share the same Trojan payload. The 128-bit keys are XORed with a shift register and leaked by serial transmission through the antenna RF channel. In order to perform our QuEST framework on this, we evaluate the confidentiality of all 128-bit inputs and successfully detect the dependence between the secret inputs to the 1-bit antenna outputs. To perform quantitative verification of secret key leakage through a Trojan payload, calculating mutual information between secret keys and the shift register is an effective method to understand how much information is being leaked. In this context, mutual information quantifies the amount of information shared between the secret key and the output of the shift register, before the data is transformed into a serial stream for transmission.

In the RSA-T100 and RSA-T300 benchmarks, the embedded Trojan payload is the same: once triggered, it replaces the regular ciphertext output with the unencrypted keys. They differ in how the Trojan is activated. In RSA-T100, the trigger is a specific plaintext value, while in RSA-T300, the trigger is a predefined count of operations. As the Trojan payload leaks the key via the legitimate ciphertext outputs, it bypasses traditional confidentiality checks because the Trojan start and end points do not violate the encryption IP's confidentiality property. Consequently, a quantitative verification approach is necessary to detect this Trojan. Using QuEST, we *detect a pronounced spike in mutual information measured at 0.7925*. This is because keys are being directly leaked to the outputs.

C. Identifying Trojan Triggers

A key application of our quantitative verification approach involves the use of CMI to identify critical vulnerable points in the system that have a significant impact on security. This method helps us detect Trojan triggers, which cause a sharp increase (toggling rate of entropy) in the mutual information between the inputs (such as the secret key) and outputs (such as leaked data). These triggers are characterized by a substantial change in mutual information, highlighting points in the system where security is compromised.

Table V shows the computation of conditional mutual information on three AES Trojan benchmarks. The CMI of the Trojan trigger is approximately $3\times$ larger than that of a random input point. As discussed in the previous section, CMI quantifies the amount of information shared between the inputs and outputs, conditioned on a third set of point(s), which can be any input or intermediate register within the design. When we compute the CMI with respect to the target Trojan trigger and a random input, significant differences emerge. These differences arise because the Trojan trigger exerts a much stronger influence on the dependency between the inputs and outputs, compared to a random net.

For a security designer analyzing a design without prior knowledge of the system, identifying the Trojan trigger can be a challenging task. Given a list of potential Trojan trigger candidates, the designer can now calculate the CMI to determine the most significant candidate. The nets with the highest CMI are considered the most likely Trojan trigger signals, allowing the designer to identify potential leakage control points.

VII. Comparisons to State-of-Art

In previous section, we discussed the limitations of the state-of-art confidentiality verification frameworks for hardware Trojan detection techniques. In this section, we experimentally prove that our QuEST framework overcomes these limitations by comparing with previous work GLIFT and QFlow.

TABLE V
Conditional mutual information of AES-Trojan benchmark conditioned on Trojan triggers.

Benchmark	Conditional Mutual Information		Time (s)
	Trojan Trigger	Random input	
AES-T100	0.2120	0.0876	238
AES-T200	0.2287	0.0735	257
AES-T700	0.1344	0.0628	203

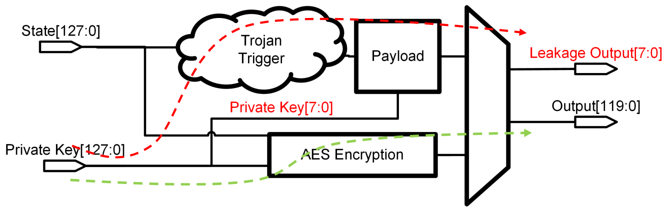


Fig. 6. Example Trojan payload with ciphertext leakage. The red line denotes Trojan path and green line denote valid encryption path.

GLIFT. GLIFT-based path sensitization techniques cannot detect Trojans that leak secret keys to the ciphertext output because the Trojan path and the valid path share the same start and end points. Figure 6 illustrates an example of such an AES Trojan. Eight bits of the secret key are leaked to the 8-bit ciphertext output through the Trojan payload. The 8 bits of the secret key bypass the encryption module and are leaked directly to the outputs without undergoing diffusion.

Unlike GLIFT, our QuEST framework is capable of detecting such a Trojan, as it can quantify dependencies to distinguish different flows. The keys that flow through the valid path are encrypted, ensuring that their diffusion is achieved. As a result, their mutual information calculation would be nearly zero. *In contrast, if the keys are leaked through the Trojan payload, the output will be dependent on the leaked input, and our QuEST quantitative verification can detect this dependency through mutual information.* Table IV shows that our QuEST framework successfully detects mutual information on the RSA-T100 and RSA-T300 benchmarks, which leak the secret key directly to the cipher text output when the Trojan is triggered.

QFlow. QFlow is an information flow-based verification method that also incorporates the quantitative analysis approach through Posterior Bayes Vulnerability (PBV). It categorizes leakage into three levels: leaked bits, unleaked bits, and bits that might be leaked, based on different PBV thresholds. However, *QFlow detects false positive leakage* on the AES-T1400 benchmark, whose results show that the last 32 bits of the secret key are weakly leaked, while the first 8 bits are fully leaked. This contrasts with the Trojan payload design file, where only the first 8 bits of the key are leaked through the covert channel, while the remaining bits are processed through the encryption module. This was explained by the fact that the Trojan exhibits strong leakage, whereas the leakage of other bits is minimal due to the 10 rounds of AES encryption. However, this explanation does not clarify why only the last 32 bits exhibit weak leakage, while the remaining 88 bits (128 -

32 - 8) do not, especially given that both sets undergo the same AES encryption process and should achieve uniform diffusion.

Unlike QFlow, *QuEST identifies that the AES-encrypted keys share the same negligible mutual information, measured at 0.0785, whereas the leaked keys measured at 0.4688.* Additionally, our previous analysis of the Trojan-free AES encryption module in Section VI-A confirms the minimal mutual information and effective diffusion, achieved by dividing the round process into identical lookup-table and key scheduling modules to reduce computational complexity.

VIII. Conclusion and Future Work

In this paper, we introduce QuEST, a quantitative confidentiality verification framework to detect undesired data leakage and to identify leakage control points. Using mutual information and conditional mutual information, our approach effectively quantifies information leakage and identifies vulnerable assets and leakage points within the design. This framework allows security designers to detect bugs and Trojan triggers even in the absence of prior knowledge of the system, providing a reliable method to identify potential vulnerabilities.

We demonstrated the application of our framework on AES benchmarks with Trojan insertions, showing how it successfully pinpointed leakage points and quantified the extent of information leakage. The results show the efficacy of our QuEST framework in both detecting Trojans and quantifying the security vulnerability of hardware designs. Furthermore, by comparing the difference in the calculation of the conditional mutual information, we provided a means to identify the most critical leakage control points, i.e., Trojan triggers.

Although QuEST offers significant advances in hardware Trojan detection, scalability remains a challenge. Our dependency checking model requires evaluating the joint probability mass function for all possible subsets, resulting in complexity limits. In this paper, we overcame this on ciphers by exploiting their redundant structures. In future work, we plan to explore entropy approximation techniques that can address the computational challenges of analyzing large encryption modules without relying on redundancy. Such approaches are also expected to offer a promising solution to verifying large processors that lack small redundant elements. We also plan to extend the QuEST framework to verify vulnerabilities in general-purpose processors. Since our conditional mutual information calculations highlight differences between leakage control points and random points, this approach may also identify micro-architectural vulnerabilities.

References

[1] 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.

[2] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner, "Theoretical fundamentals of gate level information flow tracking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1128–1140, 2011.

[3] Y. Tai, W. Hu, L. Zhang, D. Mu, and R. Kastner, "A multi-flow information flow tracking approach for proving quantitative hardware security properties," *Tsinghua Science and Technology*, vol. 26, no. 1, pp. 62–71, 2021.

[4] 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)*, pp. 603–607, 2021.

[5] M. S. Alvim, K. Chatzikokolakis, A. McIver, C. Morgan, C. Palamidessi, and G. Smith, *Posterior vulnerability and leakage*, pp. 71–100. Cham: Springer International Publishing, 2020.

[6] 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.

[7] 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.

[8] 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.

[9] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, May 2016.

[10] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, pp. 85–102, 2017.

[11] T. van Dijk and J. van de Pol, "Sylvan: Multi-core decision diagrams," in *Tools and Algorithms for the Construction and Analysis of Systems* (C. Baier and C. Tinelli, eds.), (Berlin, Heidelberg), pp. 677–691, Springer Berlin Heidelberg, 2015.

[12] B. Köpf and D. Basin, "An information-theoretic model for adaptive side-channel attacks," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, (New York, NY, USA), p. 286–296, Association for Computing Machinery, 2007.

[13] A. Covic, F. Ganji, and D. Forte, "Circuit masking: From theory to standardization, a comprehensive survey for hardware security researchers and practitioners," 2021.

[14] W. Cheng, Y. Liu, S. Guilley, and O. Rioul, "Attacking masked cryptographic implementations: Information-theoretic bounds," in *2022 IEEE International Symposium on Information Theory (ISIT)*, p. 654–659, IEEE Press, 2022.

[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, no. 2, pp. 338–350, 2017.

[16] M. Tebyanian, A. Mokhtarpour, and A. Shafieinejad, "Sc-cotd: Hardware trojan detection based on sequential/combinational testability features using ensemble classifier," *J. Electron. Test.*, vol. 37, p. 473–487, Aug. 2021.

[17] R. Sharma, G. K. Sharma, M. Pattanaik, and V. S. S. Prashant, "Structural and SCOAP features based approach for hardware trojan detection using SHAP and light gradient boosting model," *J. Electron. Test.*, Sept. 2023.

[18] 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.

[19] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting hardware trojans with gate-level information-flow tracking," *Computer*, vol. 49, no. 8, pp. 44–52, 2016.

[20] "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/.

[21] 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.

[22] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free verilog synthesis suite," 2013.