Hardware Trojan Detection for Gate-level ICs Using Signal Correlation Based Clustering

Burçin Çakır
Department of Electrical Engineering
Princeton University, Princeton, NJ 08544
Email: bcakir@princeton.edu

Sharad Malik
Department of Electrical Engineering
Princeton University, Princeton, NJ 08544
Email: sharad@princeton.edu

***Abstract*—Malicious tampering of the internal circuits of ICs can lead to detrimental results. Insertion of Trojan circuits may change system behavior, cause chip failure or send information to a third party. This paper presents an information-theoretic approach for Trojan detection. It estimates the statistical correlation between the signals in a design, and explores how this estimation can be used in a clustering algorithm to detect the Trojan logic. Compared with the other algorithms, our tool does not require extensive logic analysis. We neither need the circuit to be brought to the triggering state, nor the effect of the Trojan payload to be propagated and observed at the output. Instead we leverage already available simulation data in this information-theoretic approach. We conducted experiments on the TrustHub benchmarks to validate the practical efficacy of this approach. The results show that our tool can detect Trojan logic with up to 100% coverage with low false positive rates.**

I. Introduction

The development and supply chain of Integrated Circuits (ICs) is increasingly being spread globally to lower manufacturing costs. Outsourcing of the manufacturing process makes the controlling of the design cycle impossible, while making it vulnerable to malicious modifications. This situation raises serious concerns about the integrity and security of ICs.

The part of the circuit which alters the intended circuit behavior is often referred to as a hardware Trojan. Trojans, by definition, have a stealthy nature, i.e., they are hard to detect using conventional pre-silicon verification and post-manufacture tests. Further, they are small in size and occupy only a small fraction of the circuit. They can be inserted without changing the physical characteristics of the circuit like circuit area, die size or pin count. Trojans generally behave like a monitor in the chip which waits for certain events or a sequence of events to trigger the malicious circuitry. Although they stay dormant for most of the circuit operation, they can eventually cause the failure of the device, change the system behavior or leak confidential information. The size and complexity of modern ICs make their exhaustive testing infeasible, and limits the controllability and observability of circuit gates, while making it easier for an adversary to hide the malicious logic using the gates that are not associated with highly controllable or highly observable parts of the chip. Therefore, detection of Trojans has risen as a concern for possible threats to military systems, space facilities, financial infrastructure and medical devices.

This work was supported in part by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

Previous work in the area explored side-channel analysis as a way for IC authentication and Trojan detection. Several new techniques have been suggested to extract IC signatures based on non-functional characteristics such as delay and power in [2]–[6]. In [6], the authors propose a successful model to de-noise collected power signals and build a side-channel fingerprint for a family of ICs. However, due to degraded signal-to-noise ratio, the accuracy of these approaches reduces significantly with decreasing Trojan size and increasing process variation [7].

Logic verification, on the other hand, is independent of process variations. Extensive logical testing can detect very small Trojans even in multi-million gate circuits. However, given the large scale of the problem, it becomes almost impossible to exhaustively test the whole circuit or find the input condition that triggers the Trojan [8]. On-chip test structures like scan-chains or Built In Self Test (BIST) are widely used to reduce the testing time and detect chip failures and defects [1], [9]–[11]. Yet, these tests still may be unable to put the circuit to the triggering state. The test patterns are usually generated based on the controllability and observability of gates at the outputs. Hence, a Trojan logic may not be found if it is not activated, or even if the tests activate it and the malicious circuitry does not change the state of the circuit or the expected behavior of the outputs, an information leaking circuit for example.

In [12], a logic based analysis is presented. They propose a metric calculation to identify nearly unused logic in the design by sampling the corresponding truth table. The success of their method is sensitive to the number of sampled rows in the truth table and this brings about a trade-off between practicality and the accuracy. As the authors mention in the paper, compared to other techniques, this tool also has a potential failure on detection for distributed Trojans where the trigger condition is stretched over a chain of logic.

In this paper we propose a novel approach to detect Trojans employing a statistical-correlation-based clustering. Using the simulation data, we compute a correlation-based similarity weight for the input-output pairs of each gate in the circuit. We convert the gate-level design to a circuit graph, and weigh each edge this similarity value. The triggering wire of a Trojan is associated with gates with very low controllability, and the payload of such a malicious logic, typically, appears as a gate with a low observability value. Therefore, a correlation based weighting is likely to help identify such gates as outliers of the graph. To detect that, we apply a density-based clustering algorithm and flag the outliers in the graph, which are then used

978-3-9815370-4-8/DATE15/©2015 EDAA

to indicate Trojan logic in the circuit. Compared to traditional logic-based verification, our approach does not require the circuit to be pushed the actual triggering state. As long as there is enough activity on the gate outputs, our method can capture the statistical similarities between signal pairs. Experimental results on TrustHub benchmarks [13], show that the proposed technique has an accuracy of ~ 0.01 false positive rate with Trojan coverage up to 100%.

The rest of the paper is organized as follows. Section II provides background material on the classification of hardware Trojans, and discusses the Trojan detection techniques and challenges. In Section III, the preliminary material about the proposed clustering method is introduced. Section IV presents the Trojan detection algorithm and simulation analysis flow. Section V illustrates the experimental results on different TrustHub benchmarks. In Section VI, the proposed approach is discussed in comparison with other logic testing based Trojan detection techniques. Finally, Section VII provides some concluding remarks.

II. Proposed Approach

We assume that the hardware design that we are given is in the form of soft intellectual property (IP). It is either extracted from the actual physical chip, as a netlist, provided as the behavioral description written in a hardware description language (HDL) or the gatelist obtained by logic synthesis of the source code. Our tool works on the gate-level representation of the design, and applies to the cases where the Trojan is visible in the netlist/gatelist.

Overview

The main insight behind the simulation-based Trojan detection approach is that Trojan logic has weak statistical correlation with the rest of the circuit. Functionally related signals merge at certain nodes inside the circuit before propagating the data to the next block. They share the same fan-in cone, exchange information and finalize the computation at reconvergence points. Finding the regions, where the pairwise correlation of signals at the inputs and the output of a gate are low helps us find statistically uncorrelated group of signals as this indicates some level of functional divergence. The similarity measure is calculated by taking the cross-correlation of the input-output pairs of each gate, and computing the energy of the resulting signal. We use a graph representation of circuits. In a $circuit\ graph$, the outputs/inputs of each gate/latch and all primary I/O ports are represented by a vertex. An edge exists if and only if one of the vertices of the edge is an output and the other one is an input of the same gate/latch. The similarity measures that we compute are used to assign a weight to each edge in this graph. Fig. 2(a) shows a small circuit where the edges associated with the Trojan backdoor circuit are weighted. Here, we do not use a directed graph, since the similarity that we are trying to capture is symmetric. The reason why this computation captures the functional relation between the signals is based on information theory and will be explained in detail in *Step 1*.

Once the above setup is complete, we use the weights that we assigned to the edges of the graph to obtain a local connectivity distance and apply a density-based clustering algorithm called Ordering Points To Identify the Clustering Structure (OPTICS) [14] to reveal the clusters of functionally related structures. OPTICS is a a widely used clustering technique to compensate the shortcomings of many clustering methods on detecting clusters with varying densities. It has been successfully used in applications such as protein sequencing and data mining. The algorithm outputs a special kind of dendrogram, called a reachability plot, which we use to detect the clusters and mark some of the nodes as Trojans. Additional explanation on OPTICS is provided in the Appendix, and it is suggested that reader go through the provided summary to understand the details of our algorithm.

Step 1 - Functional Simulation based Statistical Correlation

The initial setup requires the computation of edge weights on the circuit graph. In order to compute the weights, we use functional tests and generate digital stimuli on different regions of the circuit with high coverage. We believe that test sets developed during design verification of a design should suffice for this step. The target, here, is to excite as many nodes as possible to estimate the statistical correlation between neighbouring nodes on the circuit graph. The waveforms obtained from logical tests take only binary values and transition at different times during the simulation. An output of a gate will changes its value only if at least one of its inputs switches. Therefore, the transition count depends on the inputs of the gate. However, this dependency is not even among all input-output pairs. Whichever input has statistically higher correlation to the output it will have more control on the switching behavior. Therefore the weight of the edges on the graph representation of the circuit will have different values. Recognizing this relation, we developed a model to capture the similarity between each adjacent pair, i.e. nodes connected by an edge on the graph. We obtain a new signal from the waveform of each node such that for every transition in the waveform, a unit pulse is added to the corresponding point in the new signal. In this way, we model the transition-triggering signal of each waveform. If an input of a gate has higher control on its output compared to the other inputs, then this input-output pair has higher correlation than the others. Hence, if their Fourier Transform is taken, we see that they share many common frequency components, which we can use as a similarity measure.

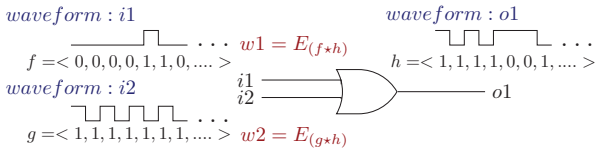


Fig. 1. Weight calculation for the input-output pairs of an OR gate from the simulation waveforms by calculating the energy of the cross-correlation signal

In order to compute such a metric, we take the cross-correlation of all adjacent signal pairs, which, by definition, eliminates the uncommon frequency factors [15] and tells us how similar the two waveforms are. Then, we calculate the energy of the resulting signal as follows.

$$E_x = \sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |\Psi_x[k]| \quad (1)$$

where $x[n]$ is the signal obtained from the cross correlation of the signals g and h as $(g \star h)[n] = \sum_{m=-\infty}^{\infty} g[m]h[m+n]$. $|X(k)|$ denotes the N-point Discrete Fourier Transform (DFT) of the sequence $x[n]$. The quantity, $|X[k]|^2$, is an estimation for the energy spectral density (ESD) of x and is conventionally given the symbol Ψ. Parseval's identity allows us to write the right hand side of the above equality by summing the squared magnitude of its transform [15]. The ESD of a signal describes how the energy of x is distributed at the various frequencies. It can be shown that if x is a real-valued signal, the ESD is also real, and non-negative.

As given in (1), the energy of a signal is equal to the sum of its distributed energy over different frequencies. Therefore, the energy of the cross-correlation can quantify how much common frequency components the two signals share. Due to Parseval's equality (1), we do not actually need to take the Discrete-Time Fourier Transform, and can do the calculation in the time domain. However, since, we work with discrete signals, we require as many samples as possible, meaning high switching activity in the circuit in order to get a good approximation. Fig. 1 illustrates how to compute the similarity weights from simulation waveforms.

This simulation-based similarity model is thought as a way to capture the actual mathematical correlation between input-output pairs in the netlist written in terms of the primary inputs. This is why we expect higher similarities at reconvergence points and between functionally related nodes, and can also detect weakly correlated signals in the circuit.

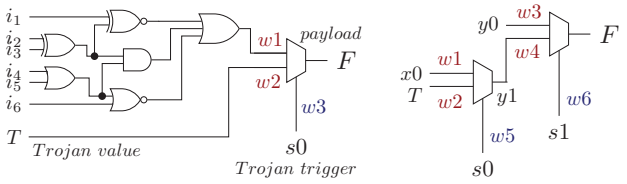
Step 2 - Weight Normalization & Clustering

The calculations that we do so far gives us a similarity measure between neighboring nodes based on pure simulation analysis. For the clustering algorithm that we will use, however, the structural connectivity of the graph is needed to be considered to come up with an accurate clustering, since it provides important information to identify the hubs and outliers in the graph. To take into account the effect of the graph structure, in [16], the notion of structural similarity is introduced. It is defined as the local connectivity density of two adjacent nodes in a weighted graph and formalized as follows:

$$\sigma(u,v) = \frac{\sum\limits_{x \epsilon \Gamma(u) \cap \Gamma(v)} w(u,x) \cdot w(v,x)}{\sqrt{\sum\limits_{x \epsilon \Gamma(u)} w^2(u,x)} \sqrt{\sum\limits_{x \epsilon \Gamma(v)} w^2(v,x)}} \quad (2)$$

where $u \in V$ and $v \in V$ are adjacent vertices connected with the edge $e \in E$ and weight $w(u,v) \in W$ in the weighted graph $G = (V, E, W)$. The weight $w(u,u)$ is defined as 1. $\Gamma(u)$ denotes the neighborhood of the node u including u and its all adjacent vertices, and formally stated as $\Gamma(u) = \{v \in V | \{u,v\} \in E\} \cup \{u\}$.

After we obtain the σ values for every edge, we define the distance between each vertex u and its adjacent vertices $x \in \Gamma(u) - \{u\}$ to be inversely proportional to the corresponding similarity rate, $\sigma(u,x)$. The distance between non-adjacent vertices u and v, s.t. $v \notin \Gamma(u)$ but u is connected to v through multiple edges, $v \rightsquigarrow u$, is defined as the sum of the distances on the shortest path which connects them. Now, we are ready to run OPTICS and get the reachability plots.



(a) Simple Trojan payload with weighted edges (b) Distributed Trojan

Fig. 2. Examples of gate-level hardware Trojan templates with T denoting the Trojan value to be propagated, with the control wires $s0$ & $s1$

Step 3 - Trojan Detection based on Reachability Plots

Trojans show some kind of functional divergence from the rest of the circuit. The reachability plots generated by OPTICS shows this weak relation either by pushing them with high reachability-distances to the borders of the clusters, i.e. the Gaussian bumps on the plot, where the reachability-distance as explained in the Appendix, measures the proximity of a node to dense regions in the graph; or explicitly showing the malicious logic as a different cluster in case of a large Trojan. Two types of reachability plots observed in conducted experiments are illustrated in Fig. 3, and discussed in more detail in Section III.

Typically, a Trojan backdoor consists of a good and a malicious part fed into some logic semantically equal to a multiplexer [12] which, when enabled, propagates the load of the Trojan or activates an invisible circuitry which may just, for example, increase the power consumption of the circuit without changing any of the outputs. The triggering condition of such an attack should happen very rarely, otherwise it is likely detected during design validation. Therefore, these wires are associated with nodes which have low enough controllability to serve as a backdoor. Also, if the Trojan is intended to change the behavior of the circuit, then, the payload has also to be chosen carefully to make sure that it has low observability value. These conditions imply that these wires, even if they have some activity, have a certain statistical distance from the rest of the circuit, and likely to appear as noise in the graph. Here, by noise, we refer to the points in the graph which look like outliers in a density-based clustering algorithm. Hence, it is reasonable to expect these nodes at the end of the ordered list with high reachability-distances. In order to detect such peak points, we use an outlier detection technique proposed in [17] based on the definitions of OPTICS.

$$lof(p) = \frac{\sum\limits_{o \epsilon N_{cd}(p)} lrd(o)}{|N_{cd}(p)|} / lrd(p) \quad (3)$$

where $lrd(p)$, the $local\ reachability\ density$, is the average reachability-distance of vertex p from its neighbors which are at most its core-distance away. The set of neighbors of p which are in its core-distance range are denoted by $N_{cd}(p)$. As explained in the Appendix, the core-distance of a node is computed according to the *MinPts* parameter of the algorithm. It denotes the minimum radius for a node to have at least *MinPts* in its neighborhood. Outliers lie between relatively denser regions of a graph and show some local deviation from their neighbors [17]. The idea behind lof is comparing

the local density of a vertex with respect to its neighbors, finding the regions of similar density and scoring the vertices which have lower densities than their neighbors for anomaly detection. After we calculate the lof values, for each node we derive a rough probability value for being an outlier or not by fitting a Gaussian model to the lof scores as proposed in [18].

Given that we have enough simulation data in practice, our method is quite effective finding suspicious nodes, since it estimates real statistical distributions of the circuit. Normally, it is difficult to activate a Trojan, but we can detect suspicious stealthy logic even if the Trojan is not fully activated, or is activated but the resulting effect cannot be propagated to the output due to low observable gate selection for payload. Sometimes, although the Trojan is triggered during regular logic testing, it may go undetected, because it does not directly change any of the ports in the circuit, but has some invisible action (such as leaking information through some side channel or scan chain outputs). In all cases, the reachability plots reflect the Trojan logic as a rise in reachability-distance along the triggering path. Further, if the malicious logic inserted by the adversary is larger than a couple of gates, it can even be seen as a separate cluster in the reachability plot. Fig. 3 (a & b) show the examples of such reachability plots and will be discussed in Section III.

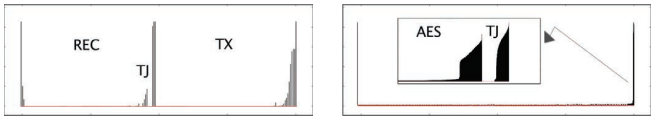
Fig. 2(a) shows a simple template for a Trojan payload with correlation based weights on the edges. Here, the weights of the input-output pairs of the payload, F, are denoted by $w1$, $w2$ and $w3$. We expect $w1 \gg w3$ and $w1 \gg w2$ due to the low activation probability of the Trojan. Therefore, the Trojan related logic is driven apart from F, and we compute higher lof values for these nodes. Together with the triggering wires, our tool can also help to identify the payloads of the Trojans, since these nodes are usually wires with low observability values. Although this is not always the case, most of the time the adversary will pick low observable nodes in order to decrease the detection probability during functional tests, and indirectly, keep them weakly connected to the rest of the circuit.

Another type of Trojan template is illustrated in Fig. 2(b) where things get a bit more interesting. Instead of one wire to trigger the malicious logic, here, we have distributed control on the propagation of the Trojan value T to the payload F. The Trojan gets activated at a specific state of the control inputs $s0$ and $s1$. Lets take the worst case where although the triggering combination is rare, the switching probability of $s0$ and $s1$ is not low; meaning they do not have low correlation with the other nodes and their weights $w5$ and $w6$ are high. In such a case, detecting the control wires based on stealthiness will be more difficult. However, the low probability of carrying a Trojan value to the payload is still hidden in the statistics of the graph and our tool can detect it. If $w1 > w2$, then the Trojan value has already a relatively higher distance to the payload. As the value of $w2$ gets larger; due to the stealthy nature of the Trojan, $w3$ gains almost the same amount of increase, and the structural similarity between $y1$ and F decreases. Therefore, overall, the distance between T and F cannot get shortened. The adjustment between the weights can be thought as similar to the mechanism of a spring of which the middle point is $y1$. Depending on the weight distribution it moves either right or left. As long as there is some activity in the nodes, it cannot stay in the middle due to the nature of Trojans as discussed above. It bridges two nodes T and F with unequal distances and behaves like an outlier between two clusters. Hence, our algorithm can identify such nodes easily since they have lof values larger than 1. This is one of the strengths of our algorithm, because we neither require the circuit to be brought to the triggering state nor need the Trojan effect to be propagated to the output to be observed. Some activity to estimate the pairwise correlation is all we need.

III. Experimental Results

Our technique was evaluated on the eight different TrustHub groups of verilog circuits. In order to test out tool on different designs, we picked one circuit from each of the groups, and used Synopsys Design Compiler to synthesize/translate them to IBM/ARM cell library for 45nm SOI process. In our circuit graphs, we replaced the buffer and inverters with one vertex since they have the exact same signals according to our model. We also ignored the clock and reset signals, since they connect the whole circuit and makes the clustering more noisy.

In order to generate the test patterns with high circuit coverage, for the designs for which we do not have access to the behavioral description, and the ones for which is not easy to write functional tests manually, we used the Synopsys TetraMAX ATPG tool to generate tests for manufacturing faults. For these circuits, we did not completely remove the scan-chain, instead treated them as regular circuit gates/latches. Although this increases the probability of bringing a circuit to a state which is supposed to happen on very rare conditions, even with this added controllability and observability our approach was able to separate the Trojan during clustering.



(a) Reachability plot for RS232-800 showing the receiver (REC) and the transmitter (TX) modules of the uart circuit with Trojan (TJ) logic pushed to the border of the REC cluster

(b) Reachability plot for AES-1800 with the Trojan (TJ) logic appearing as a separate cluster at the end of the ordered list

Fig. 3. Types of reachability plots observed with TrustHub Trojan benchmarks

Brief information about the benchmarks used is provided in Table I. AES-1800 is a cryptographic IP core which implements an AES cipher. Its Trojan payload is a shift register which continuously rotates and increases the power consumption after activation.

In our experiments, we were able to detect the Trojans for all designs. Fig. 3 shows the typical types of reachability plots that we observed. As seen in Fig. 3(b), the part of the Trojan logic in AES-1800 benchmark which consumes power after the Trojan gets activated is clearly separated from the rest of the circuit as a different cluster where the trigger wires appear on the borders, i.e. the begin and end, of the cluster bump. Fig. 3(b) shows the clustering result of a UART circuit (RS232-800), where we can distinguish the receiver (REC) and transmitter (TX) parts easily. The Trojan logic in the receiver part of RS232-800 consists of only a couple of gates,

TABLE I. RESULTS FOR TRUSTHUB BENCHMARKS

Design Information		Trojan Detection			
name	gate/latch	*MinPts*	TPR(%)	SPC(%)	Notes
s15850-100	3478	50	61	99	Two comparators (distributed trigger). Enabled in functional mode. Leaks information through output port.
s35932-200	8107	10	27	99	A comparator over 16 wires, triggered only in functional mode. Changes the output of four gates.
s38417-100	8422	50	100	99	A comparator over 16 wires. While active, changes the value of an internal gate.
s38584-200	9548	50	99	98	Compares 8 wires to increases a counter. Leaks information when the counter is between 100 & 110.
AES-1800	164800	50	92	99	Gets activated after observing a predefined input plaintext. Increases the power consumption.
wb-conmax-200	20224	50	28	96	A state machine. Activated when a specific state is reached. Changes priority of the first wishbone master.
PIC16F84-100	1616	20	75	96	A counter that increases based on the executed instruction. Changes the program memory address.
RS232-800	205	5	80	94	A counter over 19 wires. On activation, manipulates one of the outputs in receiver circuit.

† As seen from TPR values, in each case, at least a quarter of the nodes of each Trojan have been identified.
Note that TPR: True positive ratio, Specificity: 1-False positive ratio, MinPts: Parameter to the OPTICS clustering algorithm

therefore, we do not see an explicit Gaussian bump. However, the triggering logic is still pushed to the clustering boundary with relatively higher reachability-distances as labelled on the plot.

The sensitivity of the results is measured by the true positive rate (TPR), i.e. the number of Trojan nodes correctly detected as a percentage of the total number of Trojan gates. Note that in each case, even though we may not include every node on the Trojan, we successfully identify the Trojan. The exact nature of the Trojan can then be determined through code review of the nodes flagged as suspicious. To demonstrate that our tool whitelists most of the design successfully, we also provide the specificity (SPC) results, which tells us the ratio of the true negatives over the number of non-Trojan gates. (1-SPC) is the fraction of gates that are falsely flagged as being suspicious and will need to be ruled out in detailed review. The Fig. 4 shows the TPR and SPC values of the RS232-800 benchmark with varying *MinPts*. On the figure, the x axis represents the threshold to label nodes as malicious according to their outlier probabilities.

As explained in Appendix, the parameter ε is used to make an n-dimensional sphere around each point and limit the reachability computation to the data points in the sphere. For our calculations, we take ε as infinity, since for OPTICS, ε is usually needed for performance reasons, not clustering. We also do not want the reachability-distance of any node to be flagged as $undefined$, which is what OPTICS computes if the number of points in the ε-neighborhood of a node is less than *MinPts*. The choice of *MinPts*, however, cause some variations on the accuracy. There is a range of acceptable values for the *MinPts* parameter. As described in [14], it has a smoothing effect on the clustering and hence, the reachability plot of the objects. The lower *MinPts* means, the algorithm is likely to build more clusters and add more noise to the specificity. However, if *MinPts* is chosen to be a large number, then there is also a possibility that it will start labelling a lot of nodes as outliers. As *MinPts*, gets even larger, due to its smoothing effect, there will be saturation in the clustering algorithm. The reachability plot will be almost flat and nothing will look like an outlier.

For a good clustering, the suggested way of choosing *MinPts* may be to estimate the size of the smallest circuit module/cluster in the dataset. However, in our case, it is even enough to flag a small part of the Trojan. Our target is to minimize the false positive ratio, i.e., (1-SPC), while still detecting some part of the Trojan circuitry which we always can due to rare statistics of the Trojan logic. Recall that we are not really interested in maximizing TPR as long as we can detect the Trojan, because the activation logic of a Trojan can usually be detected during region specific code review once we flag some part of it. Therefore, to demonstrate the experimental results, for each benchmark, we choose the *MinPts* values such that the the number of nodes flagged as malicious is manageable (or a relatively small set) with the chosen threshold for manual inspection. Table I shows the true positive rate (TPR) and specificity (SPC) values of each circuit for a fixed threshold 1. As seen in the table, even with the highest threshold, our tool can flag the Trojans for all benchmarks successfully with high SPC values, i.e. true negative rates.

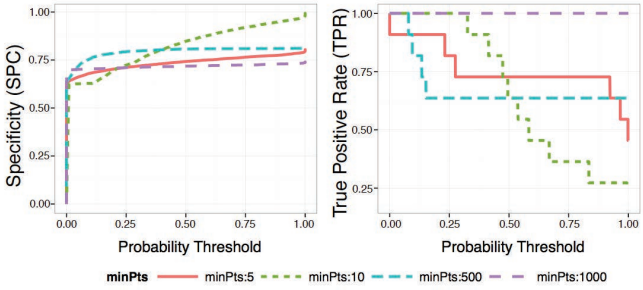


Fig. 4. The TPR (*sensitivity*) and SPC (*specificity*) values for s35932-200 bechmark with different *MinPts*

IV. CONCLUSION

There is a growing body of research on hardware security. The ability to identify malicious insertions to trusted designs is becoming increasingly important. In this paper, we address this problem by proposing a simulation-based clustering technique to detect hardware Trojans in gate-level circuits. We present a methodology to find weakly-correlated nodes or functionally isolated sections in the netlist. We convert the gate-level netlist to a circuit graph, and use simulation data to weight each edge. Then, by using a density-based clustering algorithm, we can detect Trojan-related nodes, which appear as outliers in our graph model, with low false positive rates. In this work, we did not attempt to find all Trojan logic, but to flag a small subset of gates to be reviewed which can help reduce the authentication time. We believe that our tool will yield even better results for real chips with with extensive test sets that provide higher coverage and better statistics about the activity across the chip.

REFERENCES

[1] M. Banga and M. S. Hsiao, "A novel sustained vector technique for the detection of hardware trojans." in *VLSI Design*. IEEE, 2009, pp. 327–332.

[2] J. Li and J. Lach, "At-speed delay characterization for ic authentication and trojan horse detection." in *HOST*, M. Tehranipoor and J. Plusquellic, Eds. IEEE Computer Society, 2008, pp. 8–14.

[3] B. Gassend, "Physical random functions," Master's thesis, Massachusetts Institute of Technology, Cambridge, 2003.

[4] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM Press, 2002, pp. 148–160.

[5] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference*. New York, NY, USA: ACM Press, 2007, pp. 9–14.

[6] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using ic fingerprinting." in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007, pp. 296–310.

[7] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, June 2008, pp. 51–57.

[8] R. Chakraborty, S. Paul, and S. Bhunia, "On-demand transparency for improving hardware trojan detectability," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, June 2008, pp. 48–50.

[9] C. Fagot, O. Gascuel, P. Girard, and C. Landrault, "On calculating efficient lfsr seeds for built-in self test," in *European Test Workshop 1999. Proceedings*, May 1999, pp. 7–14.

[10] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. S. M. Hassan, and J. Rajski, "Logic bist for large industrial designs: real issues and case studies." IEEE Computer Society, 1999, pp. 358–367.

[11] W.-T. Cheng, M. Sharma, T. Rinderknecht, L. Lai, and C. Hill, "Signature based diagnosis for logic bist." in *ITC*, S. Davidson and A. Gattiker, Eds. IEEE, 2006, pp. 1–9.

[12] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: Identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, New York, NY, USA, 2013.

[13] "Trusthub benchmarks," accessed: Sep. 2014. [Online]. Available: http://www.trust-hub.org/resources/benchmarks

[14] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 49–60.

[15] M. Roberts, *Fundamentals of signals and systems*. Boston: McGraw-Hill Higher Education, 2008.

[16] J. Huang, H. Sun, J. Han, H. Deng, Y. Sun, and Y. Liu, "Shrink: A structural clustering algorithm for detecting hierarchical communities in networks," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, ser. CIKM '10. New York, NY, USA: ACM, 2010, pp. 219–228.

[17] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Optics-of: Identifying local outliers," in *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, ser. PKDD '99. London, UK, UK: Springer-Verlag, 1999, pp. 262–270.

[18] H. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Interpreting and unifying outlier scores," in *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, 2011, pp. 13–24.

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996, pp. 226–231.

APPENDIX
OPTICS - ORDERING POINTS TO IDENTIFY THE CLUSTERING STRUCTURE

OPTICS is a hierarchical clustering algorithm for finding density based clusters in a given dataset [14]. The algorithm is based on the notions introduced in DBSCAN [19] where the data space is divided based on global density parameters ε and *MinPts*. DBSCAN forms clusters such that each one contains at least *MinPts* number of points (dense regions) in a radius of ε. The points which do not belong to any cluster are treated as noise. OPTICS can be considered as a generalized version of DBSCAN to compensate for its shortcomings in detecting clusters with varying densities. It computes a walk through the dataset based on the parameters ε and *MinPts*.

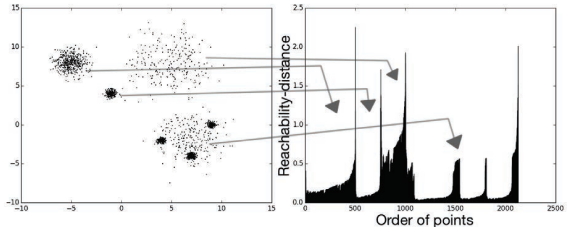


Fig. 5. An example reachability plot (right) generated by OPTICS for a data set (left) with hierarchical clusters of different sizes, densities and shapes [14]

The density of an area is measured by the number of objects in it. The basic idea behind density-based clustering is dividing the dataset into clusters such that each has at least a minimum number of points (*MinPts*) in a given radius ε. An object which has at least *MinPts* objects in its ε-neighborhood, $N_\varepsilon(p)$, is called a *core object*. The clusters are defined as the maximal sets of density-connected objects. An object p is said to be $density\ connected$ to object q, if there is an object o in the dataset such that both p and q are density-reachable from o with respect to ε and *MinPts*. Object p is $density\text{-}reachable$ from an object q, if there is chain of objects $p_1, ..., p_n, p_n = q$ such that p_{i+1} is directly density-reachable from p_i with respect to ε and *MinPts*. Object p is defined as $directly\ density\text{-}reachable$ from an object q with respect to ε and *MinPts*, if q is a core object and p lies in the ε-neighborhood of q. The DBSCAN algorithm proposed in [19], provides a single flat density-based clustering. It is useful in many cases, but not for datasets with varying densities or hierarchical structures. OPTICS overcomes this problem by producing an augmented order of the dataset to reflect the clustering structure of the dataset at different clustering levels. It adds two new notions to density-based approach explained above: $core\text{-}distance$ and $reachability\text{-}distance$. The core-distance of an object p is the smallest distance $\varepsilon' \leq \varepsilon$ such that the neighborhood of p contains at least *MinPts* objects, $|N_\varepsilon(p)| \geq$ *MinPts*. The reachability-distance of an object p w.r.t. o is defined as the maximum of the core-distance of o and the actual distance between p and o [14]. If $\varepsilon' > \varepsilon$, then the core-distance of o and the reachability-distances of all objects according to o are $undefined$.

OPTICS computes a walk on the dataset by calculating the core-distances of all objects and ordering each one based on the smallest reachability-distance with respect to an object processed before it. Therefore, a low reachability-distance means that the corresponding object is part of a dense region, whereas a high reachability-distance indicates noise or start of a new cluster. The ordered list of objects are represented a bar plot called $reachability\text{-}plot$ based on the reachability-distances of each object. As illustrated in Fig. 5 of [14], the dense regions consisting of objects which are likely to form a cluster correspond to the areas, so called Gaussian bumps, in the plot with low reachability-distances [14].