On Application of One-class SVM to Reverse Engineering-Based Hardware Trojan Detection

Chongxi Bao
ECE Department
University of Maryland, College Park
chongxi.bao@gmail.com

Domenic Forte
ECE Department
University of Connecticut
forte@engr.uconn.edu

Ankur Srivastava
ECE Department
University of Maryland, College Park
ankurs@umd.edu

***Abstract*—Due to design and fabrication outsourcing to foundries, the problem of malicious modifications to integrated circuits known as hardware Trojans has attracted attention in academia as well as industry. To reduce the risks associated with Trojans, researchers have proposed different approaches to detect them. Among these approaches, test-time detection approaches have drawn the greatest attention and most approaches assume the existence of a "golden model". Prior works suggest using reverse-engineering to identify such Trojan-free ICs for the golden model but they did not state how to do this efficiently. In this paper, we propose an innovative and robust reverse-engineering approach to identify the Trojan-free ICs. We adapt a well-studied machine learning method, one-class support vector machine, to solve our problem. Simulation results using state-of-the-art tools on several publicly available circuits show that our approach can detect hardware Trojans with high accuracy rate across different modeling and algorithm parameters.**

I. Introduction

More and more integrated circuit (IC) designers are outsourcing fabrication to foundries and incorporating third-party intellectual property (IP) cores into their designs. This practice helps to lower costs and reduce time-to-market pressure, but can also lead to security problems such as malicious modifications to ICs, also known as hardware Trojans (HTs). HTs can change IC functionality, reduce IC reliability, leak valuable information from the IC, and even cause denial of service [1]. Depending on the applications, the consequences of HTs range from loss of profit if used in consumer-electronic devices to life-threatening if used in military devices.

To reduce the risks associated with Trojans, researchers have proposed different approaches to detect them. Test-time detection approaches [2]–[5] have drawn the greatest amount of attention from researchers. In these approaches, functional and/or side-channel behavior of suspect ICs are compared to a "golden model" that represents the expected behavior of a Trojan-free IC. If the suspect IC deviates sufficiently from the golden model, it is classified as being Trojan-infected. While these approaches have met with some success, obtaining such golden models/data is mostly an open problem.

Motivation. Prior works such as [6] suggest that reverse-engineering (RE) be used to identify Trojan-free ICs and verify the data used for the golden models. However, our literature survey did not find any work describing how to do this accurately and efficiently. Reverse-engineering is actually a very complex, error-prone, and time-consuming process. It consists of 5 steps [7]: decapsulation, delayering, imaging, annotation, and schematic creation. The first 3 steps essentially obtain images of the physical layout for test ICs. The last 2 steps extract netlists for the circuit/design based on the images. A naive approach to RE-based Trojan detection would apply all 5 RE steps to extract netlists for comparison with a golden netlist, but this approach is flawed. First, some Trojans (eg. parametric Trojans [8]) can actually be missed by only comparing netlists. Second, the effort required by this naive approach is unnecessarily excessive because generating the netlist via the last 2 RE steps is time-consuming and requires manual input.

Contributions. In this paper, we propose a more efficient and robust RE approach for solving the above problem. In our approach, we avoid extracting netlists altogether. Instead, we develop a machine learning approach that classifies ICs as Trojan-free and Trojan-inserted based on features extracted from the IC images. Our approach essentially eliminates the last two reverse-engineering (RE) steps 4-5, which can save lots of unnecessary effort. The major features of our approach and our contributions are as follows:

- To our knowledge, we propose the first IC classification scheme for hardware Trojan (HT) detection based on reverse-engineering of chips without generating a gate or transistor netlist. As stated above, this saves much effort.
- In our approach, we use images obtained from the imaging step of reverse-engineering and extract features from them to characterize an IC's physical layout. We develop a Support Vectors Machine (SVM) approach that automatically learns how to distinguish between expected and suspicious structures in the ICs and ultimately classifies unknown ICs as Trojan-free or Trojan-infected. Our method does not rely on a Trojan-free (golden) IC which is the underlying assumption of many other approaches.
- Our approach depends heavily on several SVM modeling parameters as well as algorithm parameters related to feature selection. We discuss how to select these parameters in a way that makes our classification approach accurate and robust to noise in the training data. Furthermore, our proposed features account for fabrication and reverse-engineering induced variations that occur within the ICs.
- We perform simulation experiments on 6 publicly available benchmarks, ranging in size from 50 to over 100,000 gates. The results show that our method can detect three different kinds of HTs (inserted transistors, deleted transistors, and parametric changes) with very high accuracy. We also vary the modeling and algorithm parameters to determine their impact on our method.

The rest of the paper is organized as follows: Section II gives a brief review of reverse-engineering, hardware Trojan characteristics and detection, and general SVM. Section III defines the problem we want to solve and discusses the motivation behind it. Section IV explains our method in detail including challenges, feature selection, SVM implementation, parameter selection, and merits. In Section V, experimental results are discussed. Finally, Section VI concludes the paper.

II. Background

A. Reverse Engineering

Reverse-engineering of an IC is the process of analyzing an IC's internal structures, connections, etc. in order to determine how it was designed and how it operates. RE is considered an important tool for learning how to build and improve upon designs as well as proving Intellectual Property (IP) infringement. A typical RE flow includes the following steps [7].

1) *Decapsulation*: The die is removed from its package.
2) *Delayering*: Each layer of the die is stripped off one at a time using chemical methods while polishing the surface to keep it planar.
3) *Imaging*: Thousands of high-resolution images of each exposed layer are taken using scanning electron microscope (SEM). The images are stitched together to form a complete view of the layer using special software. Multiple layers are also aligned at this step so that contacts and vias are lined up with layers above and below them.
4) *Annotation*: All structures in the device (interconnects, vias, transistors, etc.) are annotated either manually or by using image recognition software [7].
5) *Schematic creation, organization, and analysis*: A hierarchical or flat netlist is generated using the annotated images as well as public information (datasheets, papers, etc.).

All of the above steps are time-consuming and error-prone. The first 3 steps essentially extract images of the structures contained in the IC. Removing and planarizing layers affect the structures in the exposed and lower layers, resulting in additional noise in their IC structures. The last 2 steps are required for circuit/design extraction and are also quite challenging. The annotation process and the schematic creation/analysis often requires input from experienced analysts [7].

B. Hardware Trojans

Hardware Trojans (HTs) are malicious modifications to the circuitry of an IC that can be made at any untrusted or outsourced phase of the IC's production (design, synthesis, fabrication, and distribution) [9]. HTs can leak secret information, disable the IC, or even destroy the IC [10] making them very dangerous. HTs can be described in terms of their physical and activation characteristics [1], [10].

Physical: A HT can change the functionality or parameters of the IC. Functional changes include addition and deletion of transistors, gates, interconnects, etc. Parametric changes consist of thinning interconnects, weakening flip-flops, increasing susceptibility to aging, etc. which can reduce the yield and reliability of an IC design [8].

Activation: Some HTs are always active (eg. parametric Trojans) while others rely on triggering mechanism. A triggered-Trojan consists of two parts: a trigger and payload. The trigger is a sensing circuitry that waits for an event, such as a rare input pattern or internal state, to take place. Before the Trojan is triggered, it is said to be in the inactive state and the IC mainly works as intended. Once the HT is triggered, the payload gets activated and executes the Trojans attack.

C. Trojan Detection Techniques

Hardware Trojans (HTs) can seriously degrade the performance and reliability of electronic systems. The consequences of HTs range from loss of profit if inserted in consumer-electronic devices to life-threating if inserted in military devices. As a result, researchers in academia as well as industry have proposed different approaches to detect HTs. These methods fall into the following three categories.

Test-time approaches consist of post-silicon tests and are the most widely studied approaches in the literature. There are two types: functional testing and side-channel fingerprinting [5]. Functional testing [2]–[4] aims to detect HTs that change the functionality (primary outputs) of the IC from the intended one. Side-channel fingerprinting [11], [12] is an alternative approach that measures side channel signals (timing, power, etc.) and uses them to distinguish genuine ICs from Trojan-infested ones. It does not require HT to be triggered to be detected (trigger itself affects the IC's side channels).

Run-time approaches add circuitry that monitors the behavior/state of a chip after it has been deployed. If deviation from the expected golden behavior is detected, additional circuitry can disable the chip or bypass the malicious logic before the HT can do any damage. [6] provides a good survey of different run-time approaches. The major disadvantages of these approaches are their high resource overhead and assumption that the run-time circuitry is Trojan-free.

Most test-time and run-time approaches assume that a golden model (intended functionality and behavior) is available to compare with. They suggest reverse-engineering be used to obtain such a golden model [6].

Reverse-engineering based approaches apply the reverse-engineering (RE) process (discussed above) to ICs in order to detect HTs. Basically, the design/netlist uncovered by RE is compared to an intended (golden) netlist. Since these approaches are not only time-consuming, but destructive, they have been pursued the least for HT detection. Their main use of RE has been to verify the Trojan-free chips used in the golden model development [6].

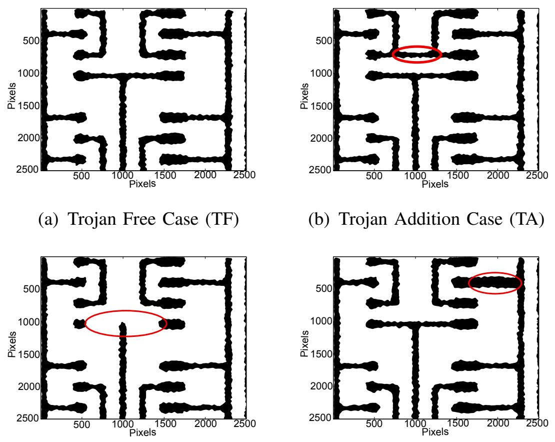
Since all of the above have their own advantages and disadvantages, one proposed direction is to apply each for the highest HT coverage. For example, post-silicon, RE-based approaches can verify golden chips required for test-time and run-time golden models. Functional and side-channel approaches can be used to detect small and large Trojans respectively that were inserted during fabrication. Run-time approaches can act as a last line of defense.

III. Problem Statement and Motivation

Problem Statement. Assume we are given ICs from one or more untrusted foundries. We want to determine which ICs are Trojan-free (TF) and which have Trojans inserted (TI). We consider that there are three-types of TI cases possible:

- *Trojan Addition (TA)*: These HTs add transistors, gates, and interconnects into the original layout.
- *Trojan Deletion (TD)*: These HTs delete transistors, gates, and interconnects from the original layout.
- *Trojan Parametric (TP)*: These HTs perform physical changes to the transistors, gates, and interconnects of the original layout as suggested in [8].

Examples of TF, TA, TD, and TP are shown in Figure 1. Note that the above problem can be viewed as an instance of the classification problem which is given as follows. Assume we have 2 classes of objects denoted by C_0 and C_1. Let each object be represented by a feature vector $\mathbf{x} = (x_1, \ldots, x_n)$ where x_i denotes the ith feature, $x_i \in \mathbb{R}$, and n denotes the # of features. Given an unknown object A, the problem is to determine the correct class of A. In our case, objects are chips under test and TF represents class C_0 while the TI cases (TA,



(a) Trojan Free Case (TF) (b) Trojan Addition Case (TA)

(c) Trojan Deletion Case (TD) (d) Trojan Parametric Case (TP)

Fig. 1: Example of three kinds of trojans. SEM image of metal1 layer is shown.

TD and TP) represent class C_1.

Motivation. As discussed in Section II-C, prior works [6] suggest reverse-engineering (RE)-based methods be used to identify Trojan-free ICs in order to develop the golden models for other HT detection approaches. However, there has not been any work devoted towards performing this classification. One can only assume that papers in the literature would assume a golden netlist and naively apply all 5 RE steps (see Section II-A) to extract netlists from all unknown chips. Only data from those chips with netlists that match the golden netlist would be included in the golden model.

We feel that there are some issues with this naive approach. First, some Trojans (eg. parametric Trojans) can actually be missed by only comparing netlists. Second, the effort required by this naive approach is excessive. RE steps 4-5 are time-consuming and require manual input for annotation and schematic read-back. In this paper, our goal is to develop a more efficient and robust RE approach for solving the above classification problem. In our approach, we avoid extracting netlists altogether. Instead, we develop a one-class SVM-based approach that makes a classification decision based on features extracted from the IC images. The key features of our approach are ability to detect all the above Trojan cases in an *efficient and automated* fashion. Details of our approach and challenges we overcome are discussed in the next section.

IV. Proposed Approach

Assumptions. As discussed above, we wish to solve the following classification problem: Given N ICs, classify each as Trojan-free (TF) or Trojan-inserted (TI). To perform the classification, we assume the following. We assume that the original design is Trojan-free and Trojans have only been inserted during fabrication. Formal verification and trusted (in-house) design are two approaches that guarantee this. We also assume that we have the original physical layout (referred to as the *golden layout*) for all the layers of the IC. Since the designer would ultimately come up with this layout and send it to the foundry, it is reasonable to assume we have this.

Basic idea. Our approach utilizes the first 3 steps of reverse-engineering (Decapsulation, Delayering, and Imaging) to obtain internal images of each layer (poly, metal1, etc.) for the ICs. By omitting reverse-engineering (RE) steps 4-5, we save lots of unnecessary effort. The images recovered represent the physical structures and layout of the ICs. Then we classify the ICs using these images and support vector machines (see below). In contrast to the naive RE approach which requires manual effort, our approach is fully automated and more efficient in terms of computational and storage resources.

Classification via SVM. The key of our approach is to solve a classification problem which is defined in Section III. There are many machine learning approaches to perform classification. Our approach will make use of the popular Support Vector Machine (SVM) [13] approach. Classification via SVM usually involves two phases:

- *Training:* In this phase, SVM takes as input training dataset $DS_T = \{(\mathbf{x}_j, y_j) \mid j = 1, \ldots, |DS_T|\}$ where $\mathbf{x}_j$ and y_j are the jth feature vector and its class label (C_0 or C_1). An optimization problem that relies on the dot product of feature vectors is solved to find a linear decision boundary $\boldsymbol{\omega}$ which separates the feature vectors of class C_0 from class C_1 so that as many objects from DS_T as possible are correctly classified.
- *Classification:* This phase takes as input a feature vector $\mathbf{x}^*$ of an unknown object and outputs its predicted class based on which side of the decision boundary $\boldsymbol{\omega}$ that $\mathbf{x}^*$ resides.

Challenges. The main component of our approach is SVM. Applying standard SVM discussed above to our problem has its own set of unique challenges:

1) *SVM features.* Selecting relevant features for SVM-based classification is important for accuracy and computational efficiency. In our case, we need to extract features (see feature vectors defined in Section III) from the IC images obtained from step 3 of the RE process. These feature vectors would be used during the classification process. Choosing too many features will result in large overheads. Also, the features selected must contain enough relevant information to accurately classify the ICs as Trojan-free (TF) or Trojan-inserted (TI). In our approach, we use a heuristic for selecting features. Features are determined by comparing physical layouts in the RE images with the golden layout. Such features are appropriate because they give us some estimate for how much deviation from the golden layout exists in the ICs. More details on the features are discussed in Section IV-A.
2) *Fabrication and RE-induced variations.* Differences between the ICs are bound to exist because of random fabrication variations. Furthermore, the RE process (delayering) is also imperfect and could result in additional noise in the IC structures. A naive matching of the designed layout with the RE layout would always result in a mismatch due to the existence of variations. The challenge is to detect the Trojans while accounting for the variations caused by manufacturing and RE process. Our SVM classifier must be able to distinguish between these random differences and the systematic differences caused by Trojan insertion. In our approach, we estimate how much random noise is to be expected between the RE layout and the golden layout. Noise margins are introduced when extracting feature vectors. More details are given in Section IV-A.
3) *SVM Training.* Traditional two-class SVM problem requires training samples from both classes to correctly train the classifier. However, in our HT detection problem (as defined in Section III), we cannot know in advance which ICs are Trojan-free and which are Trojan-inserted, nor do

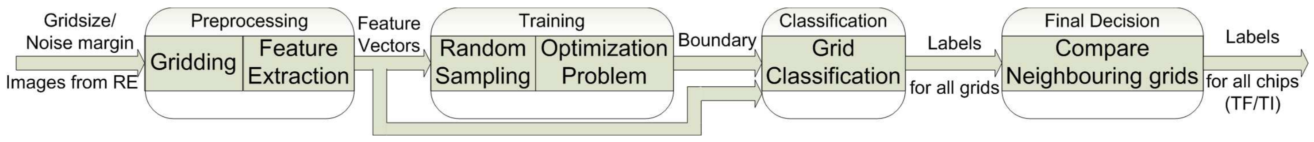


Fig. 2: Block diagram of our Trojan detection approach.

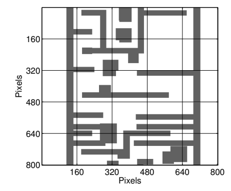


Fig. 3: Gridding techniques. Part of metal1 layer of s298 benchmark with gridsize 160×160 pixels is shown.

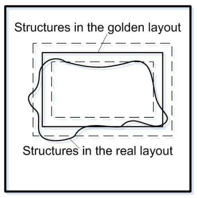

Structures in the golden layout
Structures in the real layout

Fig. 4: Golden layout and SEM image of real layout within one grid. Solid rectangle and curve denote Z and Y respectively while rectangles in dashed line are Z_{in} and Z_{out}.

we know what kind of modifications the attackers will make to those ICs. This leads to a lack of traning samples from one class. We overcome this by using a one-class SVM approach. More details are given in Section IV-B.

4) *Computational Effort.* Another general issue that we encounter is that extracting features, training the SVM, and performing classification *for the entire layout* can be demanding in terms of computational effort and storage. Rather, in our approach we simplify each of these steps by breaking them up into smaller sub-problems. Specifically, we divide the IC images into smaller non-overlapping grids (see Figure 3). Then we independently extract features and classify each grid as Trojan-free (TF) or Trojan-inserted (TI) separately. This griddingapproach is advantageous because it allows us to parallelize or distribute the processing for each grid. A final classification decision for the entire IC (TF or TI) is made by examining the classes determined for its grids.

Overall Algorithm. Our overall approach is summarized in the block diagram shown in Figure 2. We take as input the golden layout, N chips to classify, and parameter values for grid size, noise margin d_{nm}, etc. The N chips undergo only first 3 RE steps, which result in images of each layer for all N chips. The next step is to divide the images for each layer of all N chips into non-overlapping grids. Features are extracted for all N chips for each grid in each layer. We train the classifier and obtain a decision boundary for each layer using a subset of the chips (*possibly even just one chip*). After training, we classify the grids in the each layer of all the N chips as Trojan-free (TF) or Trojan-inserted (TI) based on the ν-SVM decision boundaries of each layer. Finally, we determine a label for each chip based on these grid classifications.

A. Feature Selection

As discussed above, we break the images (layouts) into smaller non-overlapping grids as shown in Figure 3. Features are extracted from each grid by comparing the corresponding grids of the IC under test (ICUT) with the golden layouts grids (which we know from design). Note it is assumed that step 3 of the RE process (see Section II-A) has already properly aligned the images. We will experiment with different grid sizes in Section V.

We illustrate many of our features with the help of Figure 4. The RE image and golden layout of one grid are shown. Let Y denote the structures in the layout image obtained by reverse-engineering, which is the curly shape in the figure. Let Z denote the structures in the golden layout, which is the solid rectangle in the middle. Let Z_{in} be Z scaled by a margin d_{nm}. Let Z_{out} be Z expanded by a margin d_{nm}. Z_{in} and Z_{out} correspond to two rectangles in dashed lines. d_{nm} represents noise in fabrication and reverse-engineering process that is thought to be reasonable. Let $\overline{Z}$ denote the complement of Z, where the universal set is the set containing all the pixels in the grid (hence $\overline{Z}$ is the set of pixels outside Z).

We select five features which are determined based on area and centroid differences between the RE and golden layouts.

Area differences. Our first 3 features are obtained by calculating the intersection of different areas between the golden layout and reverse-engineered layout. They are given by the following equations

$$f_1 = \frac{A(Y \cap Z_{in})}{A(Z_{in})} \tag{1}$$

$$f_2 = 1 - \frac{A(Y \cap \overline{Z_{out}})}{A(Y)} \tag{2}$$

$$f_3 = 1 - \frac{A(Y \cap \overline{Z_{out}})}{A(\overline{Z_{out}})} \tag{3}$$

where $A(Z)$ denotes the area of Z. Equation (1) captures how much from the golden layout is missing in the reverse-engineered layout while Equation (2) and (3) concentrate on whether there is anything additional in the reverse-engineered layout compared to the golden layout. The values of these features should be 1 if no modifications exist. When $A(Z_{in}) = 0$, which means there is no structure at all in the grid, we set $f_1 = 1$ meaning there is no deletion because there is nothing to delete. When $A(Y) = 0$ or $A(Z_{out}) = 0$, which means either there is no addition or there is no way to add because a grid of golden layout is entirely filled with structure, we set $f_2 = 0$ or $f_3 = 0$ respectively.

The reason that we expand and shrink original structure by a margin and use them to get first three features is based on the observation that process variations and noises in RE process tend to produce imperfections only outside Z_{in} and inside Z_{out}. Thus, any differences within these areas are regarded as acceptable while differences outside these areas are suspicious. The value of d_{nm} can be reasonably determined by performing simulations of the fabrication and RE processes. In Section V-B, we shall vary this parameter to see its effects.

Centroid differences. A centroid is defined as the geometric center of a mass (eg. see Figure 5). We calculate features based on the difference between the centroid of the golden layout and ICUT layout because not only do area differences matter, but where these differences occur also matters. Figure 5 explains this in detail. The solid rectangle in the middle is the structure in the golden layout while the shaded area is the structure in the RE image for the ICUT. In both cases, $A(Y)$ is the same. But in Figure 5(b), Y is more biased to

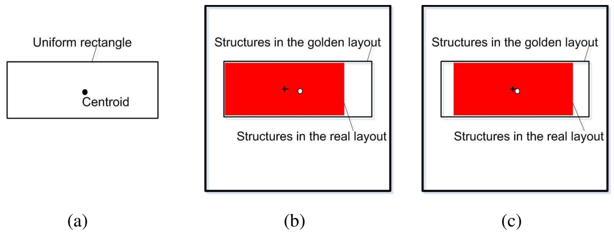

Uniform rectangle
Centroid
(a)
Structures in the golden layout
Structures in the real layout
(b)
Structures in the golden layout
Structures in the real layout
(c)

Fig. 5: Illustration of centroid and use of centroid difference to distinguish two cases. The cross and open dot denote the centroid of structures in the golden layout and real layout respectively.

the left which indicates malicious deletion of the right portion. In Figure 5(c), Y is more towards the center which indicates random reverse-engineering and fabrication noise. Both cases will produce the same f_1, f_2, f_3, but centroids will capture the difference.

Equations for the centroid difference features are as follows

$$f_4 = \frac{|CX(Z) - CX(Y)|}{grid's\ length} \tag{4}$$

$$f_5 = \frac{|CY(Z) - CY(Y)|}{grid's\ height} \tag{5}$$

where $CX(Z)$ and $CY(Z)$ denote the x and y coordinate of Z's centroid.

B. SVM implementation

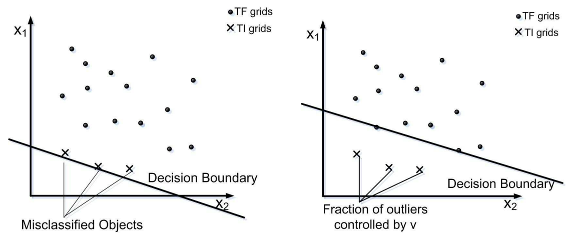
One issue that arises is that we do not have labels for training samples for either class in our HT detection problem because we do not know which IC grids are Trojan-infested, nor do we know what kind of modifications are made. To overcome this, we make a simplifying assumption and use a special implementation called one-class ν-SVM [14].

The assumption is as follows: we assume that most of the IC grids are samples of the Trojan-free (TF) class. This is reasonable because Trojans should only infect a small portion of each IC. Otherwise, if the Trojan was large, it would be easily detectable by other HT detection methods (eg. functional and side-channel analysis). Then one-class SVM will modify the optimization problem to find a decision boundary $\boldsymbol{\omega}$ that closely surrounds those training samples. Given a new sample, it will then be classified as Trojan-free or Trojan-infested depending on which side of the boundary it lies in. [15] proposes different decision boundaries like hyperplane, hypersphere, and hyperellipsoid. In our Trojan detection context, we will use hyperplane because it has a shorter run time and its performance is comparable to the other decision boundaries according to our experimental results.

One issue with this assumption/approach is that a small portion of the grids/samples may actually be Trojan-infested. Hence the boundary chosen by one-class SVM will include these samples (outliers) which will result in misclassification (see Figure 6(a)). ν-SVM introduces a parameter ν to limit the number of outlying samples (in our case TI samples) that are contained within the boundary. This makes our approach more robust to imperfections in the training set labels and should increase the classification accuracy (see Figure 6(b)).

The main details and formulation of the ν-SVM approach are discussed below.

ν-SVM Training and Classification. [14] gives a detailed algorithm of one-class ν-SVM. It introduces a parameter $\nu \in (0, 1)$



(a) Without ν controlling fraction of outliers (b) With ν controlling fraction of outliers

Fig. 6: Parameter ν will control fraction of outliers. Those outliers in the training samples will not be included in the decision boundary with proper choice of ν.

which bounds the chance of accepting outlier sample points. We summarize it below in the context of our IC classification problem.

The first step is training. We assume we have a training dataset $DS_T = \{(\mathbf{x}_j, y_j) \mid j = 1, \ldots, m\}$ where $\mathbf{x}_j$ and y_j are the jth grid's feature vector and its class label (TF). We determine a hyperplane (boundary) whose normal vector is $\boldsymbol{\omega}$ to separate as many training samples (grids) as possible from the origin with a maximum margin ρ. Errors (representing samples lying on the same side of the hyperplane as the origin does) are allowed and are denoted by slack variables ξ_i. To find such a hyperplane, one needs to solve the following quadratic programming problem:

$$\min_{\boldsymbol{\omega}\in F,\, \rho\in\mathbb{R},\, \boldsymbol{\xi}\in\mathbb{R}^m} \quad \frac{1}{2}\|\boldsymbol{\omega}\|^2 + \frac{1}{\nu m}\sum_{i=1}^{n}\xi_i - \rho \tag{6}$$

subject to

$$\boldsymbol{\omega}\cdot\mathbf{x}_i \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, m$$

In the above, $\boldsymbol{\omega}$ and ρ are the normal and bias of the hyperplane respectively; F is the input feature space; ξ_i are slack variables; m is the number of training samples; ν is a modeling parameter. In the above optimization problem, the ν parameter acts as a weighting term for the slack variables. Smaller (larger) ν results in less(more) training samples being on the same side of hyperplane as the origin is. Equation (6) is solved for $\boldsymbol{\omega}$ and ρ which define the hyperplane.

The next step is classification. Let $\mathbf{x}^*$ denote a feature vector of an unknown grid that we wish to classify as TF or TI. $f(\mathbf{x}^*)$ is a function that outputs a label for $\mathbf{x}^*$ given by

$$f(\mathbf{x}^*) = \operatorname{sgn}(\boldsymbol{\omega}\cdot\mathbf{x}^* - \rho) \tag{7}$$

Intuitively, the output of f is based on which side of the hyperplane $\mathbf{x}^*$ falls on in the input feature space F. $\operatorname{sgn}(x)$ is the sign function which is 1 for all $x > 0$ and -1 elsewhere. In our case, an output of 1 and -1 corresponds to the TF and TI class labels respectively.

Parameter values. There are several parameters that we need to decide before solving the above optimization problem:

- *ν parameter:* [14] proves that ν is the upper bound on the fraction of outliers. This means if we know the fraction of grids that are maliciously modified, we can set the value of ν appropriately but the problem is that the fraction is not known beforehand. If we choose ν to be fairly small, say 1e-8, some outliers may not be discovered, causing a high false positive rate (failure to detect HTs). On the other hand, if we choose ν to be a large number, say 0.5, some Trojan-

free (TF) grids may be mislabeled as Trojan-inserted (TI), resulting in a high false negative rate (false alarm of HTs). Our observation is that Trojans are more likely to modify polysilicon layer and interconnection layers while only a small portion of via layers may be modified. Thus, we can choose relatively large ν for polysilicon and interconnect layers while keeping ν small for via layers.
Also, ν should be varied with the number of training samples/grids. For example, if ν is so small that $\nu \times \#grids < 1$, all outliers will be included within the decision boundary which will lead to failure in detecting Trojans. In practice, we found that a good value for ν can be obtained by setting $\nu \times$ #grids equal to about 10 to 15, which results in 10 or 15 outliers being kept outside the boundary.

- *Kernel function and parameters:* The hyperplane decision boundary does not always give the best result. To have a more flexible way of choosing decision boundary, we introduce the use of kernel functions. The idea is to map the feature vectors from the input space F to a high-dimensional feature space H and use a hyperplane decision boundary in the high-dimensional space to separate most of the training samples from the origin. The decision boundary in the original input space, which is the pre-image of that hyperplane, can therefore take on many forms. The corresponding optimization problem can be written as follows:

$$\min_{\boldsymbol{\omega}\in H,\, \rho\in\mathbb{R},\, \boldsymbol{\xi}\in\mathbb{R}^m} \quad \frac{1}{2}\|\boldsymbol{\omega}\|^2 + \frac{1}{\nu m}\sum_{i=1}^{n}\xi_i - \rho \tag{8}$$

subject to

$$\boldsymbol{\omega}\cdot\Phi(\mathbf{x}_i) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, m$$

where Φ is a map of feature vectors from input feature space F to high dimensional feature space H and other parameters are the same as Equation (6).
The label of an unknown vector $\mathbf{x}^*$ is given by:

$$f(\mathbf{x}^*) = \mathrm{sgn}(\boldsymbol{\omega}\cdot\Phi(\mathbf{x}^*) - \rho) \tag{9}$$

The problem is how to find such a mapping. Since the optimization problem above only requires dot products between feature vectors, we can design the mapping in such a way that the dot products in the high dimensional space can be computed in the original input space using a *kernel function* [13] $\mathbf{k}(\mathbf{x}, \mathbf{y})$ defined as follows. Note that $\mathbf{x}$ and $\mathbf{y}$ are two vectors in the original input space F.

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x})\cdot\Phi(\mathbf{y})) \tag{10}$$

So instead of mapping our data via Φ and computing the dot products in a high dimensional space, we do it in one step, leaving the mapping completely implicit. In fact in the end we do not even need to know Φ, all we need to know is how to compute the dot products via a kernel function.
In [16], the author proposes four kernel functions such as gaussian and polynomial. Our results show that polynomial kernel function gives the best result. Thus we will use polynomial kernel function of the following form:

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = (\gamma\mathbf{x}\cdot\mathbf{y})^d \tag{11}$$

where γ is a parameter whose value can be decided by cross validation. In this paper, we use the default value in LIBSVM where $\gamma = 1/\#features$. d is the degree of polynomial. Too large d will cause the problem of overfitting while too small d will cause large training error. In practice, we found that $d = 5$ kept a good balance between training error and real error.

C. Final Classification

The final classification for each chip is decided by looking at the number of grids classified as Trojan-Inserted (TI) and their location. We do not classify a chip as TI if it has only a few sparse TI grids. Rather, we assume that in order for the chip to be classified at TI there must be at least n neighboring TI classified grids in the chip. Neighboring grids can be defined within layers (horizontally adjacent grids) and between neighboring IC layers (vertically adjacent grids). This is based on the observation that malicious modifications tend to be continuous. They are either connected in some way on one layer or connected to some other malicious modifications through neighboring layers. The maximum number of connected negative grids gives us a notion of how large deviations from the golden layout there are in the chip under test.

D. Merits of the Proposed Approach

Our proposed approach is noteworthy for several reasons:

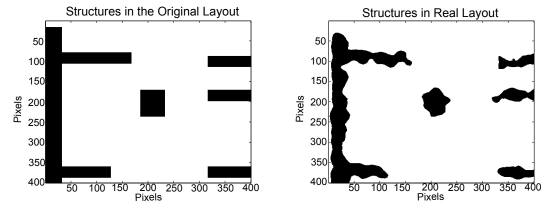
- In contrast to the naive approach (Section III), we eliminate RE steps 4 and 5 which are not only very excessive, but also may require manual effort. In contrast, our approach is fully automated and considerably less expensive since we do not need to obtain an entire netlist for each chip. Furthermore, since our approach doesn't use the netlists for detection, we may also be able to detect parametric Trojans.
- Another interesting feature of our approach is how we divide the classification into smaller sub-problems. This allows parallelizable feature extraction and SVM classification of grids for very fast/distributed computations.
- Finally, our approach accounts for noise in the RE ICs in the SVM training phase. The noise margins we employ in creating the features allows us to intelligently distinguish between differences in the ICs caused by fabrication and RE variations vs. systematic changes caused by an HT. Furthermore, ν-SVM makes our approach tunable (via ν parameter) to outliers that might exist in the training samples.

On one final note, we have only proposed using this our approach to build a golden model for other HT detection approaches because reverse-engineering is destructive. However, in the future, if less destructive methods are discovered to obtain images for one or more IC layers, our approach could also be applied to perform Trojan detection for malicious additions, deletions, and parametric changes. Our SVM-based approach would provide better coverage of HTs than other existing test-time methods (functional cannot detect parametric changes while side-channel analysis cannot detect small Trojans).

V. Experiments and Results

A. Experiment Setup

Benchmarks. We tested our approach on 6 publicly available benchmarks(from ISCAS89 and ITC99). They were all synthesized using Cadence RTL Compiler with Synopsys 90nm Generic Library and placed and routed with Cadence Encounter. Reports from Cadence tool showed that these benchmarks have a gate count ranging from 57 gates for s27 to 122,559 for b18 (see Table I).



(a) Golden layout (b) Real layout

Fig. 7: Simulation of process variations in reverse-engineering process

Golden layout extraction. The golden layout was then obtained by GDSII file generated by Cadence Encounter tool. This file was later parsed by a Matlab script and used to generate a binary image for each layer. Each pixel in the image represents 5nm in reality. '1' in the image denotes there are structures at that position in the golden layout while '0' denotes that there is nothing in the golden layout.

Trojan insertion. For each benchmark, we implemented three kinds of malicious modifications. We randomly duplicated one component, deleted one component and selected one gate and doubled its width. We will refer to these three modifications as Trojan addition(TA), Trojan deletion(TD) and Trojan parametric(TP). We will refer to original Trojan-free design as TF. In our experiments, we simulated fabrication and reverse-engineering noise (see below) for 10 "sample" chips of each type (TF, TA, etc.). Note that in the modifications made to 10 sample chips of each type, the same gate was added, deleted or altered but the noise was different.

ICUT generation and noise. Using the same method as golden layout extraction, we generated binary images for these modified designs. To simulate process variations in fabrication and reverse-engineering, we added some random noise to these binary images. Figure 7 shows the noise. Those images with noise would be used as SEM images of ICUT.

SVM training. In our experiments, we use the LIBSVM [17] as the tool to solve ν-SVM optimization problem (Equation (8)).

Model and algorithm parameters. We used polynomial kernel function defined by (11) and chose degree $d = 5$. We used the default in LIBSVM $\gamma = 0.2$. We varied ν according to the number of grids per layer in the design so that $\nu \times \#$ grids is around 10. We set the grid size discussed in Section IV to be 160×160 pixels (800×800 nm in reality), which is $20\lambda \times 20\lambda$. We chose the threshold on number of connected negative grids as $n = 2$. We set $d_{nm} = 8$ (note that by examining the golden layout images and ones with noise, we could estimate that the value of noise margin d_{nm} defined in Section IV-A is around 10 pixels). These parameters were chosen because they gave the best results according to our previous experiments.

Experiments and results recorded. We conducted four experiments. The first one was to test our method's ability to detect HTs using the above parameters. We selected 1~20 chips (shown in Table I) randomly from TF, TA, TD and TP chips and used them as training samples. The test samples consisted of 10 TF, 10 TA, 10 TD and 10 TP chips. Training phase was then conducted and test samples were classified as Trojan-free or Trojan-infested. To get a notion of the runtime of the algorithm, we record the training time for one case of each benchmark. Accuracy rate defined in Equation (12) was recorded for each case. This training and classification process was repeated for 10 trials and accuracy rate was averaged as shown in Equation (12). The difference between ten trials was samples chosen for training.

$$r_{acc} = \frac{\#\text{mislabeled chips}}{\#\text{chips under test}} \tag{12}$$

In the second, third and fourth experiments, we varied three different parameters, namely ν, grid size g, and noise margin d_{nm} for s298 benchmark while keeping everything else the same as the first experiment. In addition to accuracy rate, we also recorded the false negative/positive rate of classification of all the grids given by:

$$f_- = \frac{\#\text{false negative grids}}{\#\text{true positive grids} + \#\text{false negative grids}} \tag{13}$$

$$f_+ = \frac{\#\text{false positive grids}}{\#\text{true negative grids} + \#\text{false positive grids}} \tag{14}$$

B. Results and Discussion

Results of the first experiment are listed in Table II. They demonstrate that our method can detect three kinds of malicious modifications, including malicious addition, deletion and parametric changes, with high accuracy rate while recognizing Trojan-free chips reliably.

Results of the second experiment are shown in Table III. Note that s298 has 1936 grids per layer for grid size $20\lambda \times 20\lambda$. The results show that when ν is very small ($\nu \times \#\text{grids} < 5$), we cannot detect Trojans very well because # of outliers identified in the training samples is small. Therefore most of outliers in training samples will be mislabeled as positive. In the classification phase, many negative samples whose feature vectors are similar to positive samples will be mislabeled, resulting in failure to detect Trojans. Detection of parametric Trojans is extremely hard because their modifications are very small and their feature vectors are very similar to Trojan-free grids. On the contrary, when ν is too large, the decision boundary surrounds those true positive training samples tightly and even some positive samples may be mislabeled. This leads to detection of all the Trojans but also causes false alarms (which is indicated by a failure to detect all Trojan-free chips reliably). The results suggest that $\nu \times \#grids$ should be around 10 to 15 to get the best result.

Results of the third experiment are shown in Table IV. The results reveal that as grid size increased from $10\lambda \times 10\lambda$ to $30\lambda \times 30\lambda$, the false negative rate decreased. This is because when the grid size is small, the denominator in Equation (1), (2) and (3) tend to be small, causing it to be more sensitive to noise. Furthermore, when the grid size gets smaller, total number of grids will be larger but ν remains the same, so there may be more grids mislabeled as outliers. When grid size is equal to $10\lambda \times 10\lambda$, more false negative grids existed and were connected, giving rise to false alarm in 4 out of 10 TF chips. On the other hand, when grid size increased, the false positive rate increased. This is because when grid size is large, the method is less sensitive to noise and small modifications within a grid. In the worst case, there may be so many false positive grids that negative grids are not connected, leading failure to detect Trojans. Moreover, when the grid size gets larger, the number of grids that contain malicious modifications gets smaller. When this number is equal to one, we cannot detect the Trojan at all. Thus, large grid size does not work for small Trojans. Therefore, we concluded from the

TABLE I: Benchmarks used in the experiments.

Benchmarks	#gates	Source	#training chips	Training Time(s)
s27	57	ISCAS89	20	121
s298	283	ISCAS89	5	175
s5378	3455	ISCAS89	1	351
s15850	10984	ISCAS89	1	1032
s38417	30347	ISCAS89	1	3055
b18	122559	ITC99	1	13054

TABLE II: Chip classification accuracy rate averaged over ten trials.

Benchmark	TF	TA	TD	TP
s27	100%	100%	100%	100%
s298	100%	100%	100%	100%
s280	100%	100%	100%	100%
s15850	100%	100%	100%	100%
s38417	100%	100%	100%	100%
b18	100%	100%	100%	100%

TABLE III: Chip classification accuracy rate for ten s298 chips with varying ν.

ν	TF	TA	TD	TP
5e-4	100%	9%	6%	0%
2e-3	100%	98%	94%	82%
7e-3	100%	100%	100%	100%
2e-2	54%	100%	100%	100%
5e-2	0%	100%	100%	100%

TABLE IV: Average false negative rate f_-, false positive rate f_+ and chip classification accuracy r_{acc} for ten s298 chips of four kinds with varying grid size.

Gridsize	TF			TA			TD			TP		
	f_-	f_+	r_{acc}	f_-	f_+	r_{acc}	f_-	f_+	r_{acc}	f_-	f_+	r_{acc}
$10\lambda \times 10\lambda$	0.043%	-	78%	0.041%	5.8%	100%	0.039%	9.8%	100%	0.037%	1.7%	100%
$20\lambda \times 20\lambda$	0.037%	-	100%	0.041%	6.5%	100%	0.034%	14.5%	100%	0.035%	10.3%	100%
$30\lambda \times 30\lambda$	0.031%	-	100%	0.023%	5.2%	100%	0.027%	36.5%	100%	0.028%	19.7%	100%

TABLE V: Average false negative rate f_-, false positive rate f_+ and chip classification accuracy r_{acc} for ten s298 chips of four kinds with varying noise margin values.

Noise Margin (Pixels)	TF			TA			TD			TP		
	f_-	f_+	r_{acc}	f_-	f_+	r_{acc}	f_-	f_+	r_{acc}	f_-	f_+	r_{acc}
0	0.42%	-	84%	0.31%	8.18%	100%	0.42%	82.8%	92%	0.33%	21.8%	100%
8	0.037%	-	100%	0.042%	7.52%	100%	0.037%	13.6%	100%	0.036%	10.6%	100%
16	0.036%	-	100%	0.071%	3.41%	100%	0.043%	16.9%	100%	0.071%	3.19%	100%
24	0.30%	-	96%	0.18%	6.65%	100%	0.28%	24.9%	96%	0.16%	11.6%	100%
30	0.81%	-	63%	0.28%	6.81%	100%	0.45%	83.7%	54%	0.36%	87.8%	18%

results that grid size could neither be too small or too large.

Results of the fourth experiment are listed in Table V. Results showed that our approach worked the best when the value of noise margin was close to its true value(when it was 8). However, when the value of noise margin deviated from its true value slightly (when $d_{nm} = 16$, 24), the results were only slightly worse. This is actually good because we do not need to accurately estimate the true value of noise margin, which is almost impossible to be done precisely. We just need to pick a reasonable guess and our approach would work.

In summary, the results above show that our proposed method can detect HTs with high accuracy rate if parameters are chosen correctly. The SVM modeling parameter ν should be decided according to the number of grids per layer. When the grid size $g = 20\lambda \times 20\lambda$ and noise margin d_{nm} is close to its true value, results are the best. However, if d_{nm} deviates a little from its true value, results are only slightly worse.

VI. Conclusion

In this paper, we propose a novel reverse-engineering based approach without generating a gate or transistor netlist. Our approach adapts one-class ν-SVM to the HT detection problem. The experimental results on several publicly available benchmarks show that our method can achieve very high Trojan detection accuracy. We also investigated the impact of some modeling and algorithm parameters on the accuracy rate. Our method is efficient in storage space, does not require the existence of golden chip and is robust to variations in fabrication and reverse-engineering process. In future work, we may also investigate our SVM approach for non-invasive and non-destructive hardware Trojan detection.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. 1223233

References

[1] R. Karri, J. Rajendran, and K. Rosenfeld, "Trojan taxonomy," in *Introduction to Hardware Security and Trust*, M. Tehranipoor and C. Wang, Eds. Springer New York, 2012, pp. 325–338.

[2] Xuehui Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital IP cores," May, pp. 67–70.

[3] F. Wolff, C. Papachristou, S. Bhunia, and R. Chakraborty, "Towards trojan-free trusted ICs: problem analysis and detection scheme," Oct., pp. 1362–1365.

[4] R. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: a statistical approach for hardware trojan detection," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds. Springer Berlin Heidelberg, 2009, vol. 5747, pp. 396–410.

[5] Y. Jin and Y. Makris, "Hardware trojans in wireless cryptographic integrated circuits," *Design & Test, IEEE*, vol. PP, no. 99, pp. 1–1, 2013.

[6] S. Narasimhan and S. Bhunia, "Hardware trojan detection," in *Introduction to Hardware Security and Trust*. Springer, 2012, pp. 339–364.

[7] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," May, pp. 333–338.

[8] Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, and W. Clay, "Process reliability based trojans through nbti and hci effects," in *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*. IEEE, 2010, pp. 215–222.

[9] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno, "A case study in hardware trojan design and implementation," *International Journal of Information Security*, vol. 10, no. 1, pp. 1–14, 2011.

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

[11] S. Narasimhan, X. Wang, D. Du, R. Chakraborty, and S. Bhunia, "TeSR: a robust temporal self-referencing approach for hardware trojan detection," in *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, Jun. 2011, pp. 71 –74.

[12] S. Narasimhan, Dongdong Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Multiple-parameter side-channel analysis: A non-invasive hardware trojan detection approach," pp. 13–18.

[13] V. Vapnik, *The nature of statistical learning theory*. springer, 2000.

[14] B. Schlkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, p. 14431471, 2001.

[15] N. Shahid, I. H. Naqvi, and S. B. Qaisar, "One-class support vector machines: analysis of outlier detection for wireless sensor networks in harsh environments," *Artificial Intelligence Review*, pp. 1–49, 2013.

[16] D. Tax, "One-class classification: concept-learning in the absence of counter-examples," Ph.D. dissertation, [Sl: sn], 2001.

[17] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, p. 27:127:27, 2011, software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.