Hardware-Trojan Detection Methods Utilizing Machine Learning Based on Hardware-Specific Features

ハードウェア固有の特徴にもとづく機械学習を利用したハードウェアトロイ検出

February 2020

Kento HASEGAWA

長谷川 健人

Hardware-Trojan Detection Methods Utilizing Machine Learning Based on Hardware-Specific Features

ハードウェア固有の特徴にもとづく機械学習を利用したハードウェアトロイ検出

February 2020

Waseda University

Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering,

Research on Information System Design

Kento HASEGAWA

長谷川 健人

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Hardware devices have widely been used in our daily lives. For example, most of the people have smart phones which contain high-performance processors and various types of sensors. Motor vehicles are equipped with numerous numbers of hardware devices to perform advanced driver-assistance system such as lane centering and parking assistance. Regarding the industry field, the foundries producing semiconductors are highly automated with industrial robots. To describe the highly-automated industries by smart devices, the term 'Industry 4.0' is introduced by German government [6]. Companies all over the world have now aimed to realize the 'Industry 4.0'. Japanese government promotes 'Society 5.0' where people resolve various society challenges by incorporating the technological innovations of the fourth industrial innovation [7]. As exemplified above, people no longer lead highly convenient lives without hardware devices.

As the prevalence of Internet of Things (IoTs), which are the hardware devices connected to the Internet, the number of hardware deices is rapidly increased. According to the survey by Cisco, 28.5 billions of IoT devices will be connected to the Internet, which is drastically increased from 18 billion devices in 2017 [8]. The number of hardware devices is much larger than the population all over the world. Therefore, it is difficult to strictly manage all the hardware devices by their appropriate owners. In addition, novel technologies have been developed by hardware vendors. For example, ZYNQ SoC by Xilinx, Inc. is a low-cost and highly-integrated SoC product that integrates software programmability of an ARM-based processor with hardware programmability of an FPGA core[1]. Therefore, we can develop high-functioning hardware devices more

[1]The author and his colleagues have developed ZYNQ-based puzzle solver systems for the "Algorithm Design Contest" as in ⟨22⟩, ⟨28⟩, ⟨33⟩, and ⟨36⟩, and won the prizes as in ⟨41⟩, ⟨42⟩, ⟨46⟩, ⟨51⟩, and ⟨52⟩.

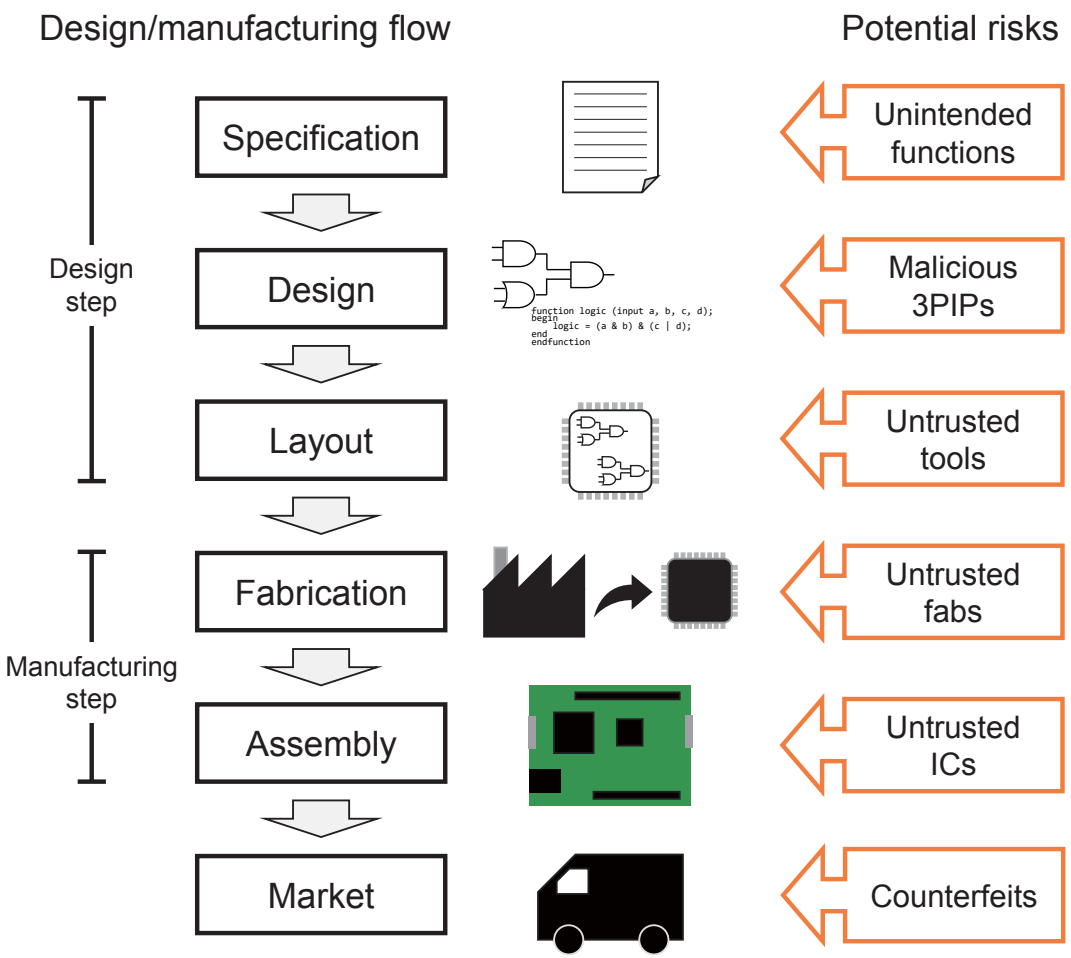


Figure 1.1: The overview of a typical hardware supply chain.

easily than ever.

Under the circumstances, security threats at hardware devices have been pointed out. An article [9] reports that a military weapon is maliciously modified by an attacker. If an attacker would successfully modified a military weapon, it might hurt or kill people. The threats on hardware devices are now raising and becoming reality. How to tackle the problem is a major concern in the IoT era.

There exist potential risks at any steps on hardware design and manufacturing. Figure 1.1 shows the overview of hardware design and manufacturing steps. The production process of hardware devices is roughly divided into two steps: the design step and manufacturing step. Hardware specification and circuit design are determined at the design step. Due to the rapid and low-cost production, third-party intellectual properties (3PIPs) are often integrated to products. Since recent integrated circuits (ICs) are highly integrated and contain millions of gates, attackers can easily modify the circuits with malicious intent. If 3PIP providers insert malicious functions into their products,

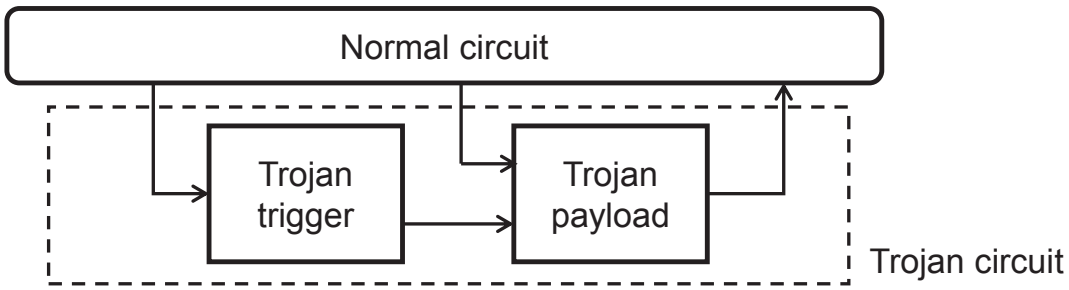


Figure 1.2: The example of a typical hardware Trojan circuit.

their user vendors would integrate them to their hardware products. As for the manufacturing step, circuit designs are often fabricated and assembled at the overseas fabrications because the hardware market becomes globalized. As discussed above, there exist potential risks at any steps on hardware design and manufacturing because of globalization and low-cost production. A malicious function inserted at a hardware circuit is often call as a 'hardware Trojan (HT).' How to detect hardware Trojans is a serious concern. Note that, existing hardware design and manufacturing processes typically include test process, but the test processes just check the functionality and validity of the products. Since typical test processes do not take the threats of hardware Trojans into account, we have to develop a hardware-Trojan detection scheme. In this dissertation, we aim to find out how to tackle the threats of hardware Trojans. The most important goal of this dissertation is **to establish effective hardware-Trojan detection methods** which will protect hardware products from the threats of hardware Trojans.

Figure 1.2 illustrates the structure of a typical hardware Trojan circuit. A hardware Trojan circuit is inserted into a normal circuit. The hardware Trojan circuit is composed of two components: a trigger circuit and a payload circuit. In order to conceal hardware Trojan circuits, attackers often set trigger conditions to their hardware Trojans. Note that, some of the hardware Trojans have no trigger conditions (such hardware Trojans are called as 'always-on hardware Trojans'), and these hardware Trojans should be detected at existing test processes because any Trojan payloads must affects the original functionality and electrical specification. In this dissertation, we assume that a hardware Trojan has a certain trigger conditions. The trigger circuit receives several signals from the normal circuit as a trigger condition. When the values of the received signals satisfies the conditions determined by an attacker beforehand, the trigger circuit outputs an enable signal to the payload circuit. When the payload circuit receives the enable signal, the payload circuit performs malicious functions such as leakage of internal information of the normal circuit and/or denial of service. Since attackers set rare conditions to their hardware Trojans, hardware vendors cannot catch the hardware Trojan circuits in a test process.

The methodologies defeating hardware Trojans have been studied recent years [10, 11]. The methodologies can be classified into two categories: a prevention methodology and a detection methodology. With the prevention methodology, hardware designs are altered to be difficult for third parties to insert or further modify the circuit. Hardware logic encryption approaches thwart insertion of hardware Trojans by encrypting hardware designs [12, 13]. Physical unclonable functions (PUF) [14, 15] are often used to generate secret keys for logic encryption. Camouflaging (or obfuscating) approaches [16, 17] are also applied to protect hardware designs. The prevention methodology is effective to protect hardware designs so as not to be unintentionally modified by third parties. On the other hand, the detection methodology aims to catch hardware Trojan circuits. The detection approaches are further classified into two categories: a destructive approach and a non-destructive approach. The destructive approach generally adopts destructive reverse-engineering techniques to depackage an IC and performs optical analysis [18]. Though this approach is useful to physically analyze the manufactured ICs, the tested ICs cannot be shipped anymore. Meanwhile, the non-destructive approach does not destruct ICs. Since the non-destructive approach just analyzes the design or manufactured ICs itself without destruction, this approach can be easily integrated to existing design and manufacturing process. In this dissertation, we focus on the non-destructive approaches to defeat hardware Trojans.

As discussed above, we aim to defeat hardware Trojans adopting non-destructive approaches. Several non-destructive methods can be taken on the design or manufacturing step. The non-destructive methods on the design step analyze hardware designs including 3PIPs. Formal verification and code analysis are often taken. The non-destructive methods on the manufacturing step analyze manufactured hardware products. Functional tests and side-channel analysis are often taken. Most of the existing methods take model-based approaches, and therefore detectability of unknown threats has to be discussed. Moreover, due to the rapid development in hardware industries, a large variety of hardware devices are developed. In order to follow the rapid development, we must detect hardware Trojans effectively.

Recently, machine learning has attracted the interest of researchers as a breakthrough in data mining, and it is expected to overcome security-related challenges. Machine learning helps us to analyze a number of datasets and to observe the trend of the datasets. Machine learning can be used to find out malicious behaviors such as in [19, 20]. [19] focuses on run-time malicious behavior detection for the multi-core platforms. [20] focuses on run-time malicious behavior detection by analyzing the power consumption of the hardware products. However, as far as we know, there have been no hardware-Trojan detection methods using machine learning at gate-level netlists focusing on design step of IC production process proposed so far. Developing a sophisticated machine-learning-based hardware-Trojan detection method is a challenging problem, but also a promising

technology to realize highly automated society. The major problems to leverage machine learning for hardware Trojan detection are how to extract effective features to identify Trojan nets (the nets that consist of a hardware Trojan circuit) and normal nets, and how to detect hardware Trojans without Golden models.

In this dissertation, we leverage machine learning algorithms to the non-destructive hardware-Trojan detection methods.

First, we aim to detect hardware Trojans at gate-level netlists in the design step. As discussed above, we take the non-destructive approach to detect hardware Trojans in a design information which is going to be a product. The major concern on the non-destructive approach in the design step is to follow the rapid development of hardware production. In order to rapidly update the hardware Trojan database and to effectively apply hardware Trojan detection to a number of hardware products, machine-learning-based approaches are quite useful. To realize the methodology above, our objective is to explore how to apply machine learning algorithms to hardware Trojan detection in the design step. In Chapter 2, we utilize machine learning algorithms such as support vector machine (SVM), random forest, and neural networks (NNs) for hardware Trojan detection with extracting effective features from a net in gate-level netlists. This study breaks new ground in the field of hardware Trojan detection in the design step. The main contributions of Chapter 2 are; to extract effective features from gate-level netlists for machine learning algorithms, and to implement machine-learning-based hardware-Trojan detection methods. In addition to the first study on machine-learning-based hardware-Trojan detection in the design step, we further discuss their applications in Chapter 3. The main contribution of Chapter 3 is to propose applications utilizing machine-learning-based hardware-Trojan detection methods in the design step.

Next, we aim to detect malicious behaviors based on power analysis utilizing one of the unsupervised machine learning algorithms after the manufacturing step or when using a hardware product. As discussed above, side-channel analysis is often taken in the manufacturing step. How to detect abnormal behavior without the Golden model is a major concern for the hardware Trojan detection based on side-channel analysis. In this point, our objective is to develop abnormal behavior detection method without the Golden model. Unsupervised machine learning algorithms can be applied to this scenario. In Chapter 4, we first obtain power profile identifying between active and sleep modes of a target micro-controller, and then extract feature values. Based on the extracted feature values, we apply an abnormal detection method which is one of the unsupervised machine learning algorithms. The concept of referencing the target micro-controller itself has not been discussed well. Our proposed method establishes a new methodology to detect abnormal behavior based on power analysis. The main contributions of Chapter 4 are; to extract feature values based on power profile of a hardware device using a micro-controller, and to apply an abnormal detection algorithm

to detect malicious behaviors.

Throughout this dissertation, we aim to overcome the hardware-security issues by utilizing machine learning with hardware-specific features.

1.2 Dissertation Overview

In this dissertation, we propose machine-learning-based hardware-Trojan detection methods based on effective feature values. This dissertation is organized according to the following chapters.

Chapter 2 [Hardware Trojan Classification Utilizing Machine Learning] proposes a hardware-Trojan classification method at gate-level netlists to identify hardware-Trojan infected nets (or Trojan nets). In this chapter, we have a preliminary discussion on how to apply machine learning to hardware Trojan detection, and then we evaluate the effective feature values for hardware Trojan detection. As a preliminary discussion on the hardware Trojan detection at gate-level netlist, we extract the *five hardware-Trojan features* from each net in a netlist. These feature values are complicated so that we cannot give the simple and fixed threshold values to them. Hence, we secondly represent them to be a *five-dimensional vector* and *learn* them by using SVM or NN. Finally, we can successfully classify all the nets in an unknown netlist into Trojan ones and normal ones based on the learned classifiers. The experimental results with Trust-HUB benchmarks demonstrate that our method increases the true positive rate compared to the existing state-of-the-art results in most of the cases. Based on the preliminary discussion, we propose effective Trojan-net features for supervised machine-learning-based hardware-Trojan detection and their application to a random forest classifier. We propose 51 Trojan-net features which describe well Trojan nets. After that, we pick up random forest as one of the best candidates for machine learning and optimize it to apply to hardware-Trojan detection. Based on the importance values obtained from the optimized random forest classifier, we extract the best set of 11 Trojan-net features out of the 51 features which can effectively classify the nets into Trojan ones and normal ones, maximizing the F-measures. By using the 11 Trojan-net features extracted, our optimized random forest classifier has achieved at most 100% true positive rate as well as 100% true negative rate in several Trust-HUB benchmarks and obtained the average F-measure of 79.3% and the accuracy of 99.2%, which realize the best values among existing machine-learning-based hardware-Trojan detection methods.

Chapter 3 [Application of the Hardware-Trojan Detection Utilizing Machine Learning] proposes three applications of machine-learning-based hardware-Trojan detection. First, we propose a machine-learning-based hardware-Trojan detection method for gate-level netlists using multi-layer neural networks. We classify the nets in an un-

known netlist into a set of Trojan nets and that of normal nets using multi-layer neural networks based on 11 Trojan-net features proposed in Chapter 2. By experimentally optimizing the structure of multi-layer neural networks, we can obtain an average of 84.8% true positive rate and an average of 70.1% true negative rate while we can obtain 100% true positive rate in some of the benchmarks, which outperforms the existing methods in most of the cases. Second, we propose a Trojan-invalidating circuit, and implement it on an FPGA board. The implementation results demonstrate that the implemented Trojan-invalidating circuit successfully prevent from activating a hardware Trojan. Third, we propose a reinforcement of the hardware-Trojan detection utilizing machine learning. Since existing machine-learning-based hardware-Trojan detection methods are performed in the feature spaces, the proposed method considers boundary net structures between normal nets and Trojan nets and compensates the first machine-learning-based detection results based on them. The experimental results demonstrate that our proposed method successfully improve the detection results compared to the existing method.

Chapter 4 [Malicious Behavior Detection Based on Power Analysis] proposes an anomaly behavior detection method utilizing power analysis for low-cost micro-controllers. Our method accurately measures power consumption of the target device, and then classifies its waveform into the sleep-mode part, in which a micro-controller saves power, and into the active-mode part, in which a micro-controller works in a normal operation. After that, we obtain the duration time and consumed power from each active-mode period as feature values. Finally, we detect abnormal behavior based on the obtained feature values utilizing an outlier detection method. In our experiments, we empirically evaluate the proposed method utilizing two types of micro-controllers, and the experimental results demonstrate that our proposed method successfully detects abnormal behaviors.

Chapter 5 [Conclusion] summarizes this dissertation and gives several future directions on machine-learning-based hardware-Trojan detection.

Chapter 2

Hardware Trojan Classification Utilizing Machine Learning[1]

2.1 Introduction

As the semiconductor processes technology has continuously advanced over the years, there have been introduced low-cost and high-performance embedded hardware products in our daily lives. In order to meet such hardware product demands, IC vendors often outsource their products to third-party IC vendors. However, third-party IC vendors are not always reliable and some of them may embed or insert hardware Trojans into their IC products intentionally.

Hardware Trojans are malicious functions inserted to IC products, which is unintended for IC vendors. Hardware Trojans may cause malfunctions, destroy IC products, and/or leak secret information. The manufacturing process of IC products comprises many steps and every step has the risks that malicious vendors may insert hardware Trojans into IC products [10]. In particular, hardware Trojans are most likely inserted in the design step since inserting hardware Trojans is very easy in this step. In the design step, IC designers use electrical files which describe IC designs and malicious vendors just need to modify these files electrically to insert hardware Trojans. Even these files can be modified remotely via the networks. In this chapter, we focus on hardware Trojan inserted in the design step of the IC production process.

Several hardware Trojan detection methods have been proposed so far, but malicious vendors can develop new hardware Trojans which defeat existing hardware Trojan detection methods as described in [21]. A new approach to tackle hardware Trojans is highly expected.

Here we clarify our goal for hardware-Trojan detection at the design step. There are

[1]Technical contents in this chapter have been presented in the publications ⟨2⟩, ⟨3⟩, ⟨16⟩, and ⟨17⟩.

roughly two goals for hardware-Trojan detection:

- **Goal 1 [Partial detection]**: We detect any part of hardware Trojans in a chip and we no longer use the entire chip. In this case, we have to detect any part of hardware Trojans [2, 3, 22];

- **Goal 2 [Complete detection]**: We detect all parts of hardware Trojans in a chip and eliminate them. We can use the chip after eliminating hardware Trojans. In this case, we have to detect all the hardware Trojans in the chip somehow [23, 24].

Since the Goal 2 includes the Goal 1, we focus on Goal 2 in this chapter.

Even if new hardware Trojan detection methods are proposed, new types of hardware Trojans can be developed very soon. For example, hardware Trojans called DeTrust [21] are proposed, just after the static hardware Trojan detection method FANCI [3] is developed. As one of the effective solutions to solve this problem, we can utilize machine learning and detect unknown hardware Trojans based on the learned classifier.

Machine learning can be used to find out malicious behaviors such as in [19, 20]. [19] focuses on run-time malicious behavior detection for the multi-core platforms. [20] focuses on run-time malicious behavior detection by analyzing the power consumption of the hardware products. However, as far as we know, there have been no hardware-Trojan detection methods using machine learning at gate-level netlists focusing on design step of IC production process proposed so far.

Based on the discussions above, we propose a static machine-learning-based hardware-Trojan classification method at gate-level netlists. The proposed method classifies a set of the nets in a given unknown netlist into Trojan nets and normal nets without using logic simulations nor functional simulations.

The Goal 2 discussed above can be achieved by the following two phases: In Phase 1, we first try to detect all possible Trojan nets by using our method. After that, in Phase 2, we apply any existing hardware-Trojan detection methods such as [3, 22] to the nets identified to be Trojans by Phase 1 and refine the classification results even though they include false positives. Among them, Phase 1 is the most important since it is the first step to classify between Trojan nets and normal nets, where the number of Trojan nets identified mistakenly to be normal nets should be as small as possible, in other words, the false negative value should be as small as possible. Our method is targeted to Phase 1 of this approach.

In order to achieve that, this chapter discusses how to apply machine learning to hardware Trojan detection. This is the first work to apply machine learning algorithms to hardware Trojan detection and to extract effective features from Trojan nets.

To start with, we preliminary discuss how to effectively apply machine learning to hardware Trojan detection. We first discuss the Trojan net feature values that frequently

appear in the Trojan nets, and extract the five hardware-Trojan features, or *Trojan features*, of each net based on the several known hardware-Trojan infected netlists. Then, we apply *machine learning* to the extracted features of every net in netlists. We consider the five Trojan features to be a *five-dimensional vector* and learn many five-dimensional vectors with machine learning. In this step, we use a support vector machine (SVM) or a neural network (NN). Finally, we can successfully classify a set of nets in a given unknown netlist into Trojan one and normal one by using the learned classifier. Machine learning enables us to classify hardware Trojans automatically without simulating a given circuit nor actually running it.

After the preliminary discussion on how to apply machine learning to hardware Trojan detection, we propose effective feature values that can be best applied to machine-learning-based hardware-Trojan detection. First, we propose 51 gate-level Trojan-net features from given gate-level netlists which describe well Trojan nets very well. At that time, we utilize the random forest classifier [25], one of the strong machine-learning methods, and optimize it to apply to hardware-Trojan detection. Since random forest gives the *importance value* for every Trojan-net feature, we can effectively know which one contributes more than the others to detect hardware Trojans. We select the most effective 11 Trojan-net features which maximize the average F-measures.

The contributions of this chapter are summarized as follows:

1. First, we propose a static machine-learning-based hardware-Trojan classification method by learning the five Trojan features to classify a set of unknown nets into Trojan nets and normal nets. As far as we know, this is the world-first approach which successfully applies machine learning to detect hardware Trojans at gate-level netlists;

2. When applying our machine-learning-based hardware-Trojan classification method to Trust-HUB benchmarks, our method can much increase the true positive rate compared to the existing state-of-the-art results in most of the cases even though our method is completely static;

3. After that, we propose 51 gate-level Trojan-net features which describe Trojan nets from many sets of known netlists, and then select the best 11 Trojan-net features to maximize F-measures for hardware Trojan classification;

4. By using the optimized random forest classifier based on the 11 Trojan-net features, we have obtained 100% true positive rate and 100% true negative rate in several Trust-HUB benchmarks [1] and obtained the *best values* in average F-measure compared to conventional methods. Moreover, we achieve classification results of 94.9% average precision and 99.2% average accuracy.

2.2 Related Works

Now we classify hardware-Trojan detection methods into the two types: the dynamic detection methods and the static detection methods.

The dynamic detection methods detect hardware Trojans based on simulating the circuits and/or actually running them including hardware Trojans. In these methods, it is necessary to find out the trigger states to active hardware Trojans and see their behaviors. Since we require too much time to know the trigger states, we have to find out beforehand which part of the circuit includes hardware Trojans and set up test patterns very carefully [23, 24]. Hence the hardware-Trojan detection results must be much dependent on input patterns and/or simulation results. Very recently, a pattern-matching-based dynamic method [22] and a clustering-based dynamic method [2] have been proposed.

The static detection methods check whether a given circuit includes hardware Trojans or not, without simulating the circuits nor actually running them, but by just using hardware-Trojan related information. Since these methods do not actually run the circuits nor simulate them, we do not have to generate input test patterns and thus the detection results are not dependent on simulation results. Recently, a static hardware Trojan detection method [3] and a statistical technique for foundry identification [26] have been proposed. However, they are hard to apply to sequential circuits and large systems. For example, FANCI [3] first calculates truth tables and gives the suspicious flags to the gates whose output is rare to transit. After that, hardware-Trojan infected nets (or *Trojan nets*) are identified using these suspicious flags. Since this detection method uses truth tables, it can be applied only to combinational-triggered hardware Trojans. In other words, detecting sequential-triggered hardware Trojans is difficult by using this method.

Overall, static approaches are robust and effective to hardware Trojans, if we can successfully develop these approaches.

2.3 Preliminary Evaluation on Machine-Learning-Based Hardware-Trojan Detection

2.3.1 Feature Extraction

As in [27], assume that hardware Trojans have several trigger conditions and they are activated when primary inputs and/or internal states of a given circuit satisfy the trigger conditions. Now we try to classify a set of nets in a given netlist into a set of Trojan nets and a set of normal nets. We focus on a static hardware-Trojan detection approach not using circuit simulation.

In this section, we first pick up hardware-Trojan infected gate-level netlists from

Table 2.1: Five gate-level netlists from Trust-HUB.

Netlist Name	Number of all the nets	Number of Trojan nets
RS232-T1300	307	9
RS232-T1500	314	12
s35932-T200	6435	16
s38584-T100	7399	9
s38584-T300	9110	1730

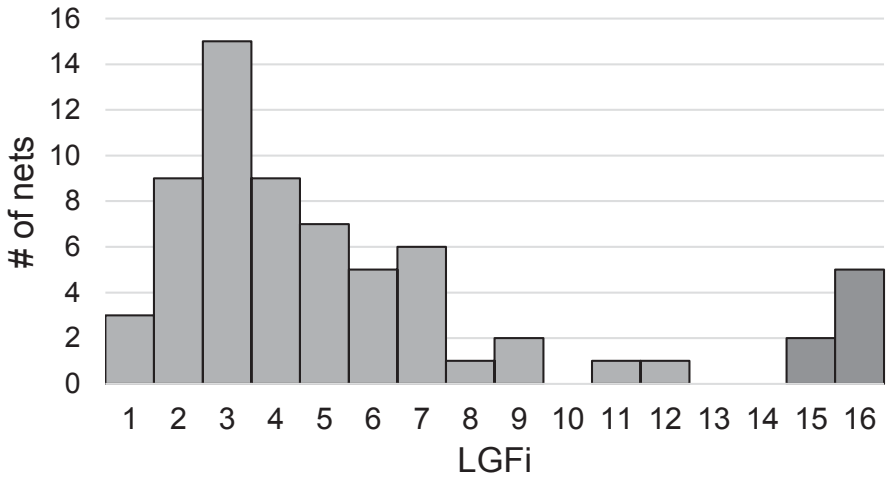


Figure 2.1: The histogram of LGFi for Trojan nets.

Trust-HUB benchmark suites [1] and find out several features which must be strongly related to hardware Trojans. Here we randomly pick up five hardware-Trojan infected netlists as listed in Table 2.1. Note that, the original circuits in the Trust-HUB benchmark suites are often used to evaluate circuit designing in research context and Trust-HUB gives which net is a Trojan net and which net is a normal net in every netlist benchmark.

Now we focus on a target net n in a given netlist and extract its several Trojan features.

Logic-Gate Fanins (LGFi)

Since hardware Trojans are activated only when the primary inputs and/or internal states of the given circuit satisfy the trigger conditions, we expect that the transitive fanins of Trojan nets become large enough.

Figures 2.1 and 2.2 summarize the breakdown of the fanin counts (logic-gate fanins, LGFi in short) in Trojan nets and normal nets in the five netlists of Table 2.1, where LGFi here means the number of the inputs of the logic gates two-level away from the target net

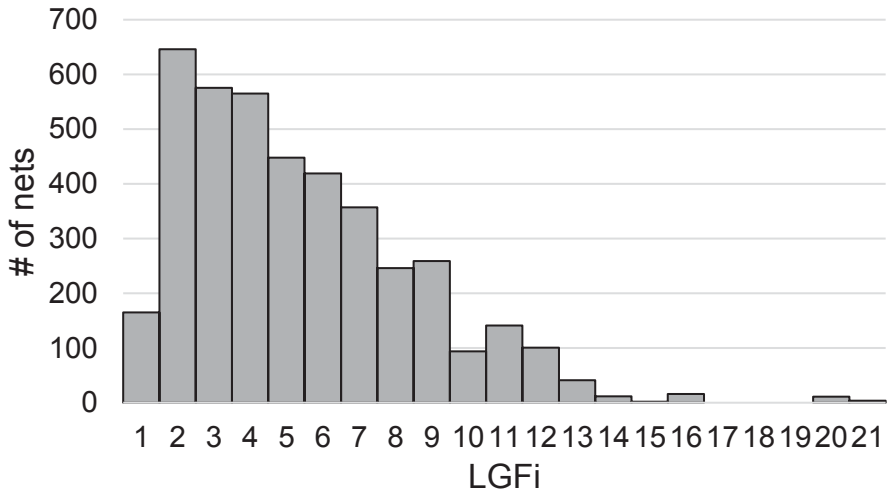


Figure 2.2: The histogram of LGFi for normal nets.

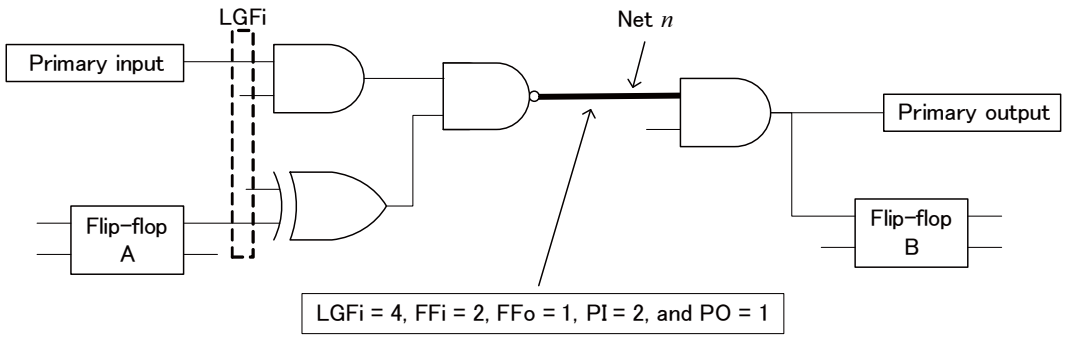


Figure 2.3: The example of the Trojan feature values extracted from a netlist.

n (see LGFi in Figure 2.3).[2] As in Figure 2.1, some of the Trojan nets have very large fanin counts compared to other Trojan nets. Those nets having large fanin counts are very likely to be Trojan nets and LGFi can be a large hint to classify between Trojan nets and normal nets.

However, some normal nets also have large fanin counts as in Figure 2.2. The number of fanins itself is not enough to classify between Trojan nets and normal nets.

FFi, FFo, PI, and PO

Since some hardware Trojans have to memorize the internal states of a circuit to trigger Trojans, flip-flops are expected to be located near Trojan nets. Then we define FFi to be

[2]In designing LGFi, how many logic-levels away from the target net *n* we should see becomes a problem. Here we simply define LGFi to be the number of the inputs of the logic gates *two-level* away from *n* because we want to see the transitive fanins of *n* which are not so far from *n* and not so close to *n*.

Table 2.2: Average values of FFi, FFo, PI, and PO.

Trojan/Normal	FFi	FFo	PI	PO
Trojan nets	0.98	0.98	3.36	4.26
Normal nets	1.67	1.45	4.45	4.41

the minimum gate level to any flip-flop inputs from the target net n. We can also define FFo to be the minimum gate level from any flip-flop outputs to the target net n.

Some hardware Trojans give particular outputs from the chip and then the primary outputs are expected to be located near Trojan nets. Also, some hardware Trojans use primary inputs to activate hardware Trojans and then the primary inputs are expected to be located near Trojan nets. We define PI to be the minimum gate level from any primary input to the target net n. We can also define PO to be the minimum gate level to any primary output from the target net n.

In order to confirm that the assumptions above are true, Table 2.2 summarizes the average values of FFi, FFo, PI, and PO in the five netlists of Table 2.1. As in Table 2.2, those values in Trojan nets are definitely smaller than those in normal nets. We expect that we can effectively classify between Trojan nets and normal nets by using FFi, FFo, PI, and PO values.

Trojan Features

Overall, we can consider the five Trojan feature values below for every target net n in a netlist to classify between Trojan nets and normal nets:

1. **LGFi (Logic Gate Fan-ins)**: The number of inputs of the logic gates two-level away from the net n.

2. **FFi (FlipFlop Input)**: The number of logic levels to the nearest flip-flop input from the net n.

3. **FFo (FlipFlop Output)**: The number of logic levels to the nearest flip-flop output from the net n.

4. **PI (Primary Input)**: The minimum logic level from any primary input to the net n.

5. **PO (Primary Output)**: The minimum logic level to any primary output from the net n.

Table 2.3: Examples of the five feature values.

LGFi	FFi	FFo	PI	PO	Trojan/Normal
3	4	1	2	3	Trojan net
7	0	2	5	7	Trojan net
9	0	2	5	9	Trojan net
16	0	0	3	5	Trojan net
6	0	2	5	6	Trojan net
8	2	4	5	3	Normal net
3	2	1	2	3	Normal net
6	1	0	7	4	Normal net
2	3	0	2	7	Normal net
5	1	5	7	5	Normal net

Figure 2.3 shows the example of the five feature values above extracted from a netlist. In this figure, we focus on the bold net n. Since n has the four transitive fanins two-level gates away from n as depicted in dotted lines in the figure, LGFi = 4. Since the logic level from the flip-flop A to the net n is two, then FFi = 2. Since the logic level from the net n to the flip-flop B is one, then FFo = 1. In the same way, we have PI = 2 and PO = 1.

The Trojan features extracted above must be strong clues to detect hardware Trojans. In the next section, we discuss how to classify between Trojan nets and normal nets using these Trojan feature values extracted from an unknown netlist.

2.3.2 Hardware Trojan Classification Utilizing Machine Learning

In Section 2.3.1, we extract the five Trojan feature values for every net n in a given netlist, which must be strongly related to hardware Trojans. However, we cannot set up simple and fixed threshold values for them to classify between Trojan nets and normal nets.

For example, Table 2.3 shows the five Trojan feature values of several Trojan nets and normal nets from Trust-HUB benchmarks. As seen in this table, some Trojan nets have very large LGFi values but some of the normal nets have also large LGFi values. FFi, FFo, PI, and PO values in Trojan nets tend to become small but we cannot set up the particular threshold values to classify between Trojan nets and normal nets, even in this table. In addition to that, we have to manually update these threshold values even though we can successfully set up the threshold values, every time new hardware Trojans are developed.

Based on the discussion above, we propose a machine-learning-based hardware-

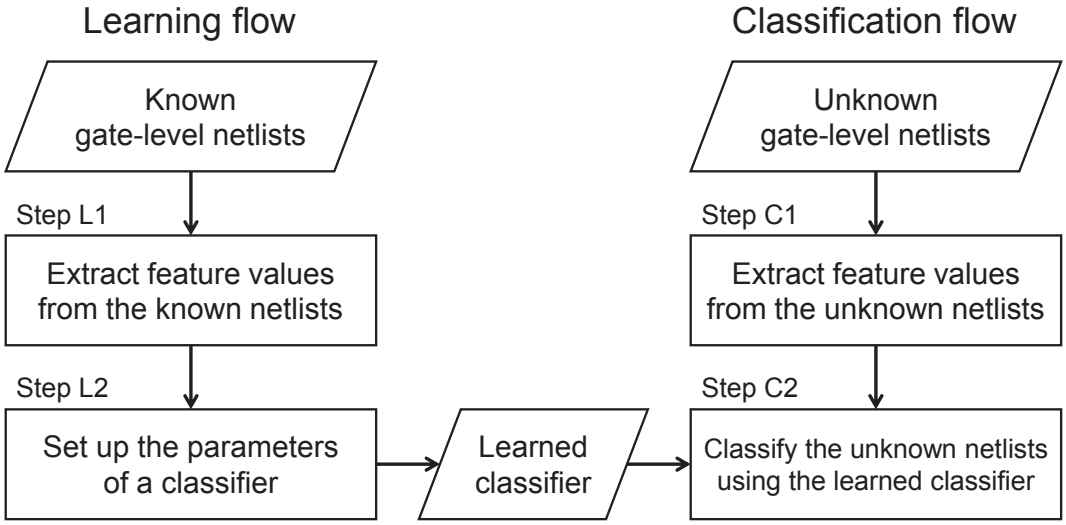


Figure 2.4: The flowchart of learning and classification.

Trojan classification method, where it automatically learns Trojan nets and normal nets using the five Trojan feature values and classifies an unknown netlist into a set of Trojan nets and a set of normal nets by using the learned classifier. We utilize the two major classifiers: a support vector machine (SVM) and a neural network (NN).

As discussed before, new types of hardware Trojans can be developed very soon even if new types of hardware-Trojan detection methods are proposed. We expect that a machine-learning-based approach can detect even unknown hardware Trojans based on the learned classifier.

The Flow of the Proposed Method

Figure 2.4 shows the flow of the proposed method. The proposed method is composed of the two parts: the learning flow and the classification flow.

In the learning flow, we learn many known Trojan nets and normal nets by using the five Trojan feature values. The five Trojan feature values can be considered to be a *five-dimensional feature vector* $\boldsymbol{x}_n$ for every net n. We first extract the five-dimensional feature vector for every net in known netlists (Step L1). After that, we learn the extracted five-dimensional feature vectors (Step L2). In Step L2, we set up the parameters of the classifier so that the true positive rate (TPR) is maximized where TPR is defined by the ratio of the true positives (TP) over the number of all the Trojan nets.

In order to realize Phase 1 of the Goal 2 as discussed in Section 2.1, TPR is the most important because identifying *all* the Trojan nets in a given unknown netlist is the most important. If a large number of Trojan nets remain unidentified, they may much affect the manufactured products using them. However, if we can identify almost all the real

Trojan nets to be Trojan nets, we just need to examine the identified ones carefully, even though some of the normal nets are identified mistakenly to be Trojan nets. Overall, we set our goal to maximize TPR firstly and we optimize our classifier using the TPR value.[3]

In the classification flow, we classify a given unknown netlist into a set of Trojan nets and a set of normal nets using the learned classifier. We first extract the five-dimensional feature vector for each net from the unknown netlist (Step C1). After that, we identify each net in the unknown netlist to be a Trojan net or not by using the learned classifier (Step C2).

We consider SVM and NN as classifiers, and describe how to learn them in the rest of this section.

Classification Algorithms

Classification using SVM First, we use SVM [29] to learn Trojan nets and normal nets. For each net n, we give the label y_n which shows a net n is a normal net or Trojan net as an integer. If a net n is a normal net, $y_n = -1$, and if a net n is a Trojan net, $y_n = 1$.

Between a linear SVM and a non-linear SVM, we utilize a non-linear one because it has obtained a better result through preliminary experiments.[4] In a non-linear SVM, a net k is classified into a normal net or a Trojan net by Eq. (2.1):

$$f(\boldsymbol{x}_k) = \mathrm{sign}\left(\sum_{i=1}^{N} y_i \lambda_i K(\boldsymbol{x}_i, \boldsymbol{x}_k) + b\right) \tag{2.1}$$

where N is the number of learned nets in known netlists, λ_i is the Lagrange multiplier, $K(\boldsymbol{u}, \boldsymbol{v})$ is the kernel function for vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, b is a bias term, and $\mathrm{sign}(u)$ is the sign function which returns 1 when $u \geq 0$, and returns -1 when $u < 0$. $f(\boldsymbol{x}_k) = 1$ means that the net k is identified to be a Trojan net, and $f(\boldsymbol{x}_k) = -1$ means that the net k is identified to be a normal net.

Now we briefly describe how to set up the parameters in Eq. (2.1) according to [29]. In Eq. (2.1), b is a bias term, which is given by Eq. (2.2) using any one of the net i

[3] In machine learning, there are many other metrics such as recall and precision [28]. For example, if we achieve 100% precision, it means that the nets identified to be Trojans are truly Trojan nets but it does not mean that all of Trojan nets in a netlist are detected. If some Trojan nets are not detected, we have to examine again all the nets identified to be normal nets. Since a target gate-level netlist usually includes a few Trojan nets and thus includes so many normal nets, it must be very hard to do that. Hence, we try to maximize TPR firstly in SVM and NN, while maximizing TNR secondly in NN.

[4] In fact, we have tried a linear SVM and a non-linear SVM as preliminary experiments and obtained good results when we have used a non-linear SVM. For example, TPR becomes 88% when we use a linear SVM with dynamic weighting for s35932-T200 but it improves to 100% when we use a non-linear SVM with dynamic weighting.

satisfying $\lambda_i > 0$:

$$b = y_i - \sum_{j=1}^{N} \lambda_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.2}$$

As in [30], we use the radial basis function (RBF) kernel for the SVM classifier. The RBF kernel is given by $K(\boldsymbol{u}, \boldsymbol{v}) = e^{-\gamma|\boldsymbol{u}-\boldsymbol{v}|^2}$ where γ is a parameter of the SVM classifier.

To obtain λ_i for Eq. (2.1), we maximize $L(\boldsymbol{\lambda})$ by Eq. (2.3) where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N)$:

$$L(\boldsymbol{\lambda}) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.3}$$

Eq. (2.3) is constrained by $\sum_{i=1}^{N} \lambda_i y_i = 0$ $(0 \leq \lambda_i \leq C)$ where C is a parameter of the SVM classifier.

Overall, the parameter values γ and C determine the complete form of Eq. (2.1) and we decide them using learning data so that the TPR value over all the learned netlists is maximized.

In Step L2, we use random search and grid search to determine the γ and C values [30]. Firstly, we give the rough range $0.0001 \leq \gamma \leq 0.01$ and $1 \leq C \leq 1000$ for the parameters γ and C. Then we execute machine leaning using the randomly selected γ and C values 100 times. At that time, we randomly divide a set of all the nets over all the known netlists into the two sets: we assume that the first half set is known data and the other half set is unknown data and calculate the TPR value. As a result, we obtain the best values γ_b and C_b which maximize the TPR value. Now we pick up the parameter range $\gamma_b - 0.001 \leq \gamma \leq \gamma_b + 0.001$ and $C_b - 10 \leq C \leq C_b + 10$ and perform more detailed search in this region by using the grid search. Finally, we can obtain the parameter values γ_{opt} and C_{opt}.

In Step C2, we classify the unknown netlist into a set of Trojan nets and a set of normal nets using the parameters γ_{opt} and C_{opt}.

Classification using NN A neural network [31] is composed of an input layer, middle layers and an output layer. The input layer receives a vector input to NN. The middle layers are composed of one or more layers and compute internal vectors between the input layer and the output layer. The output layer sends out the calculated data as an output of NN. Each layer is further composed of units which receive the output values as inputs from the units in the previous layer and compute an output value using these inputs.

Every unit has a type of calculation model. Since the sigmoid function is very widely used as in [32, 33], we also use this function in our NN. Let us focus on a particular unit.

Let o be an output of the unit, K be a set of units in the previous layer, x_k be an input from a unit $k \in K$, and w_k be a weight for each input value x_k. Then we define $z = \sum_{k \in K} w_k x_k + b$ where b is a bias term. Then the sigmoid function is given by $\sigma(z) = 1/(1 + e^{-z})$.

NN Structure Now we consider how to construct the NN structure for hardware Trojan detection. Firstly, it is natural that we set the number of units in the first layer to be five, each of which corresponds to each of the five Trojan feature values. Next, we set the number of units in the output layer to be one, which just outputs a value ranging from 0 to 1. If the output is larger than or equal to 0.5[5] , its input five-dimensional vector $\boldsymbol{x}_n$ is identified to be Trojan. Otherwise, it is identified to be normal.

The problem here is how to construct the middle layers in NN. The middle layers have the two important parameters: the number of layers and the number of the units in each layer.

The number of layers in the middle layers: We use a simple single-layered middle layer, since our proposed method requires to classify the five-dimension feature vectors into a set of Trojan nets and a set of of normal nets. This classification is not so complicated and we just use a single-layered middle layer for simplicity.

The number of units in the single-layered middle layer: Let N_{mid} be the number of units in the single-layered middle layer. We perform several experiments and determine the best N_{mid} value. First we set $N_{mid} = 5$, which is the same number of units in the input layer. We increase the N_{mid} gradually and find out that, the average TPR value over all the known netlists is continuously increased until $N_{mid} = 7$. When $N_{mid} = 8$, the average TPR value is not increased but the average true negative rate (TNR) defined by Section 4 is dramatically decreased. Then we finally set $N_{mid} = 7$.[6]

Figure 2.5 shows the NN structure used in our proposed method, and Figure 2.6 shows the j-th unit in the middle layer where w_{jk} is the weight for the k-th input x_k and b_j is the bias term.

How to Perform (Step L2) and (Step C2) According to [32], we briefly describe how to set up the NN classifiers' parameters: the weight value w_k and bias value b in every unit in NN.

[5]We set the threshold value to be 0.5 based on [3].

[6]When we set N_{mid} to be 8 or larger, TNR continues to decrease but TPR is not improved and hence we consider that our learning process is adequately converged when we use seven units in the middle layer.

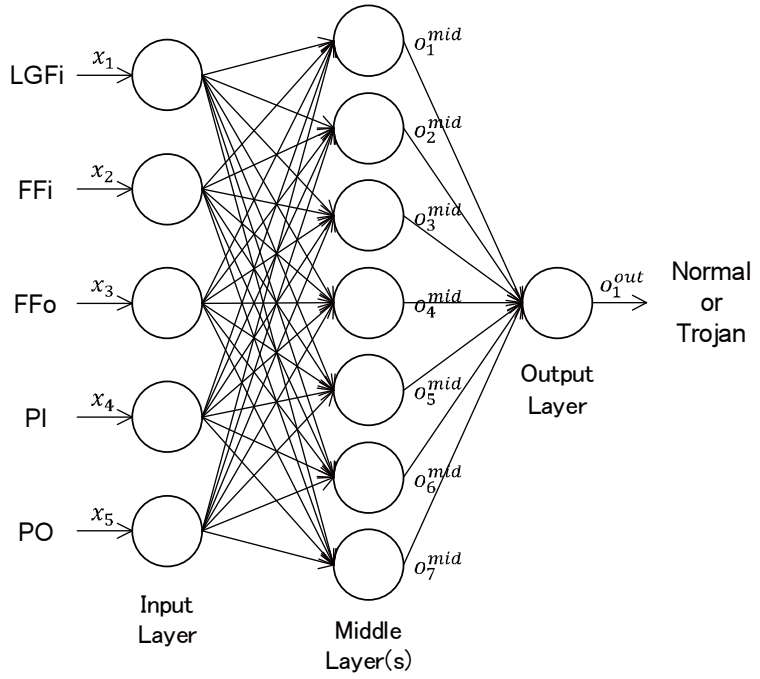


Figure 2.5: The proposed NN structure.

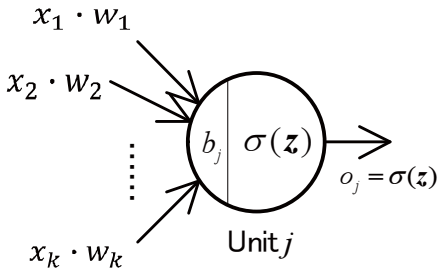


Figure 2.6: The unit in each layer in NN.

Firstly, we set $w_k = 1$ and $b = 0$ in all the units in the input layer, since they just receive the input five-dimensional vector. Next, we initially set w_k and b to be the random values (following the normal distribution, where its mean μ is 0 and its standard deviation σ is 1) in all the units in the middle and output layers of Figure 2.5 and update them as follows:

We give the five-dimensional vector $\boldsymbol{x}_n$ of a known net n to NN. Every net n has the true value y_n showing whether n is a normal net or Trojan net. If a net n is a normal net, $y_n = 0$, and if a net n is a Trojan net, $y_n = 1$. Then we calculate the error E^{out} between the NN output o_1^{out} and the true value y_n of the net n:

$$E^{out} = \frac{1}{2}(y_n - o_1^{out})^2 \tag{2.4}$$

Based on [32], the weight w_k^{out} $(k = 1, 2, \cdots, 7)$ of the output unit is updated as follows:

$$w_k^{out} \leftarrow w_k^{out} - \eta \cdot \delta \cdot o_1^{out} \cdot (1 - o_1^{out}) \cdot o_k^{mid} \tag{2.5}$$

where $\delta = \partial E^{out} / \partial o_1^{out}$, η is the learning rate, which is the parameter of NN, and o_k^{mid} is the input value of the output unit (see Figures 2.5 and 2.6). In this chapter, we set $\eta = 0.01$ which is the default value in PyBrain [34].

In the same way, the bias value b^{out} of the output unit is updated as follows:

$$b^{out} \leftarrow b^{out} - \eta \cdot \delta \cdot o_1^{out} \cdot (1 - o_1^{out}) \tag{2.6}$$

After all the parameters in the output unit are updated, those in every unit in the middle layer will be updated in the same way.

In Step L2, every time we input a five-dimensional vector $\boldsymbol{x}_n$ of the known net n into NN, the parameters in all the units in NN are updated. We can finally have a learned NN classifier when we give all the known nets.

In Step C2, we classify the unknown netlist into a set of Trojan nets and a set of normal nets using the learned NN classifier. If the NN output is larger than 0.5, the input five feature values are identified to be a Trojan net. Otherwise, the input five feature values are identified to be a normal net.

2.3.3 Experimental Results

In this section, we apply our proposed method to the several gate-level netlists in Trust-HUB benchmark suite [1]. We use an Intel Xeon E7-4870 computer environment. In our method, Step L1 and Step C1 are written in the C language and Step L2 and Step C2 are written in Python. We use Python machine learning library scikit-learn [35] for the SVM classifier and PyBrain [34] for the NN classifier.

Table 2.4 summarizes the 17 gate-level netlists from Trust-HUB. Note that these data include very small number of Trojan nets and detecting them must be very difficult.

Classification by Support Vector Machine

We classify a netlist listed in Table 2.4 into a set of Trojan nets and a set of normal nets using SVM. Since we have 17 benchmarks that is too small to split into the train, validation and test datasets, the machine learning is performed with the leave-one-out cross validation [36], where one of the netlists in Table 2.4 is considered to be an unknown netlist and the others are considered to be known netlists. For example, when RS232-T1000 is considered to be an unknown netlist, all the other 16 netlists are considered to be known netlists. After learning known netlists and setting up the learned SVM classifier, we classify an unknown netlist into a set of Trojan nets and a set of normal nets.

Table 2.4: The Trust-HUB benchmarks [1] used in the experiments.

Netlist Name	Number of normal nets	Number of Trojan nets	Netlist Name	Number of normal nets	Number of Trojan nets
RS232-T1000	266	45	s35932-T200	6,419	16
RS232-T1100	300	12	s35932-T300	6,423	37
RS232-T1200	305	10	s38417-T100	5,807	12
RS232-T1300	298	9	s38417-T200	5,807	15
RS232-T1400	299	12	s38417-T300	5,807	44
RS232-T1500	302	12	s38584-T100	7,390	9
RS232-T1600	298	9	s38584-T200	7,380	200
s15850-T100	2,429	27	s38584-T300	7,380	1,730
s35932-T100	6,423	15			

The number of Trojan nets is relatively small compared to the total number of nets in a given netlist. For example, the number of Trojan nets is just 0.1%–19% in the netlists as listed in Table 2.4. This is because malicious third-party vendors tend to hide their presence in IC and try to pass the IC tests. Learning data for Trojan nets in our SVM-based hardware-Trojan classification method may be much smaller than those for normal nets. It is very important to balance the learning data between Trojan nets and normal nets.

Based on this discussion, we have performed the three types of experiments as follows:

1. **No weighting:**
   We have just used original data for learning. For example, if some known netlist has N_n normal nets and N_t Trojan nets, SVM has learned N_n normal nets and N_t Trojan nets.

2. **Static weighting ($W = 20$):**
   Every Trojan net is learned by SVM W times. For example, if some known netlist has N_n normal nets and N_t Trojan nets, SVM has learned N_n normal nets and $W \times N_t$ Trojan nets.

   The ratio of the number of Trojan nets and that of normal nets is different in the benchmarks used for machine learning. In this chapter, we use (Trojan nets/normal nets) ratio for RS232 benchmarks (RS232-T1000 – RS232-T1600). For RS232 benchmarks, the total number of normal nets is 2,068 and that of Trojan nets is 109. Therefore the ratio of the number of Trojan nets and that of normal nets becomes $2,068/109 \approx 18.97$. Based on this value, we set the static weight $W = 20$. This is just an example of static weighting.

Table 2.5: Learned normal nets and Trojan nets.

Unknown Data	No weighting		Static weighting		Dynamic weighting	
	Learned normal nets	Learned Trojan nets	Learned normal nets	Learned Trojan nets	Learned normal nets	Learned Trojan nets
RS232-T1000	63,067	2,169	63,067	43,380	7,225	7,175
RS232-T1100	63,033	2,202	63,033	44,040	7,191	7,038
RS232-T1200	63,028	2,204	63,028	44,080	7,204	7,153
RS232-T1300	63,035	2,205	63,035	44,100	7,202	7,130
RS232-T1400	63,034	2,202	63,034	44,040	7,218	7,130
RS232-T1500	63,031	2,202	63,031	44,040	7,215	7,176
RS232-T1600	63,035	2,205	63,035	44,100	7,205	7,130
s15850-T100	60,904	2,187	60,904	43,740	6,664	6,490
s35932-T100	56,910	2,199	56,910	43,980	7,155	6,969
s35932-T200	56,914	2,198	56,914	43,960	7,219	6,923
s35932-T300	56,910	2,177	56,910	43,540	7,215	6,975
s38417-T100	57,526	2,202	57,526	44,040	7,209	6,992
s38417-T200	57,526	2,199	57,526	43,980	7,213	6,946
s38417-T300	57,526	2,170	57,526	43,400	7,204	7,072
s38584-T100	55,943	2,205	55,943	44,100	6,663	6,426
s38584-T200	55,953	2,014	55,953	40,280	7,225	7,200
s38584-T300	55,953	484	55,953	9,680	7,225	7,068

3. **Dynamic weighting:**
   We balance the number of learned normal nets and the number of learned Trojan nets in every benchmark netlist. At first, we find out the nets which have the identical feature vector in normal nets, leave one of them and delete the rest of them. We also find out the nets which have the identical feature vector in Trojan nets, leave one of them and delete the rest of them. For example, if the feature vector $(1,2,1,2,2)$ appears three times in normal nets, we delete two of them and leave one of them.

   After that, we balance the number of learned normal nets and the number of learned Trojan nets as follows: Assume that we now have N'_n normal nets and N'_t Trojan nets. Then SVM has learned every normal nets once but every Trojan net (N'_n/N'_t) times. Totally, SVM has learned N'_n normal nets and N'_n Trojan nets.

Table 2.5 summarizes the number of learned normal nets ("learned normal nets") and the number of learned Trojan nets ("learned Trojan nets") in each experiment type when classifying every unknown data. For example, in case of RS232-T1000 at "No weighting", the SVM classifier learns 63,067 normal nets and 2,169 Trojan nets from the other 16 netlists. Since "Dynamic weighting" deletes several feature vectors, the number of learned normal nets and the number of learned Trojan nets are reduced compared to

the other two experiments but, as shown below, the accuracy of the SVM classifier in this case is much increased by balancing normal nets and Trojan nets and deleting ambiguous Trojan feature vectors.

Table 2.6: Experimental results of the SVM classifier.

	No weighting							Static weighting							Dynamic weighting						
Netlist name	C	γ	TN	FP	FN	TP	TPR	TNR	C	γ	TN	FP	FN	TP	TPR	TNR	C	γ	TN	FP	FN	TP	TPR	TNR
RS232-T1000	1	0.001	266	0	45	0	0%	100%	1	0.001	151	115	41	4	9%	57%	1	0.001	82	184	21	24	53%	31%
RS232-T1100	1	0.001	300	0	12	0	0%	100%	1	0.001	188	112	9	3	25%	63%	1	0.001	81	219	5	7	58%	27%
RS232-T1200	1	0.001	305	0	10	0	0%	100%	1	0.001	192	113	8	2	20%	63%	1	0.001	78	227	2	8	80%	26%
RS232-T1300	1	0.001	298	0	9	0	0%	100%	1	0.001	182	116	8	1	11%	61%	1	0.001	77	221	1	8	89%	26%
RS232-T1400	1	0.001	299	0	12	0	0%	100%	1	0.001	176	123	10	2	17%	59%	1	0.001	66	233	2	10	83%	22%
RS232-T1500	1	0.001	302	0	12	0	0%	100%	1	0.001	184	118	10	2	17%	61%	1	0.001	71	231	2	10	83%	24%
RS232-T1600	1	0.001	298	0	9	0	0%	100%	1	0.001	181	117	9	0	0%	61%	1	0.001	77	221	1	8	89%	26%
s15850-T100	1	0.001	2,429	0	27	0	0%	100%	1	0.001	2,164	265	24	3	11%	89%	1	0.001	1,597	832	2	25	93%	66%
s35932-T100	1	0.001	6,423	0	15	0	0%	100%	1	0.001	5,878	545	11	4	27%	92%	1	0.001	3,839	2,584	1	14	93%	60%
s35932-T200	1	0.001	6,419	0	16	0	0%	100%	1	0.001	5,870	549	13	3	19%	91%	1	0.001	3,799	2,620	0	16	100%	59%
s35932-T300	1	0.001	6,423	0	37	0	0%	100%	1	0.001	5,863	560	17	20	54%	91%	16	0.001	3,725	2,698	27	10	27%	58%
s38417-T100	1	0.001	5,807	0	12	0	0%	100%	1	0.001	5,358	449	11	1	8%	92%	1	0.001	4,393	1,414	0	12	100%	76%
s38417-T200	1	0.001	5,807	0	15	0	0%	100%	1	0.001	5,349	458	15	0	0%	92%	1	0.001	4,399	1,408	4	11	73%	76%
s38417-T300	1	0.001	5,807	0	44	0	0%	100%	1	0.001	5,362	445	0	44	100%	92%	11	0.001	4,158	1,649	0	44	100%	72%
s38584-T100	1	0.001	7,390	0	9	0	0%	100%	1	0.001	6,706	684	4	5	56%	91%	1	0.001	4,568	2,822	0	9	100%	62%
s38584-T200	1	0.001	7,380	0	200	0	0%	100%	1	0.001	6,771	609	44	156	78%	92%	1	0.001	4,695	2,685	12	188	94%	64%
s38584-T300	1	0.001	7,380	0	1,730	0	0%	100%	1	0.001	7,235	145	618	1,112	64%	98%	1	0.001	4,907	2,473	185	1,545	89%	66%

Table 2.6 shows the classification results. In this table, TN shows the number of normal nets identified to be normal nets correctly. FP shows the number of normal nets identified to be Trojan nets mistakenly. FN shows the number of Trojan nets identified to be normal nets mistakenly. TP shows the number of Trojan nets identified to be Trojan nets. TPR shows the true positive rate. TNR shows the true negative rate which is defined by the the number of the true negatives over the number of total normal nets, where the true negatives here mean the normal nets which are identified to be the normal nets.

As in the table, if we just use original data by giving no weights ("No weights"), TPR becomes 0 which shows that all the Trojan nets are identified to be normal nets mistakenly. By giving a static weight ($W = 20$) to every Trojan net ("Static weight"), TPR values are improved. By giving a dynamic weight and balancing the learned normal nets and Trojan nets ("Dynamic weight"), we can have the best TPR values. By using our method with dynamic weighting, we can achieve 80% or more TPR values in most of the cases. In some cases, we can achieve 100% TPR values. As discussed in Section 2.3.2, it is important to maximize TPR. The results clearly demonstrate that our method with dynamic weighting can successfully find out almost all the Trojan nets in a given unknown netlist by just using learning data. TNR in dynamic weighting is relatively small compared to static weighting. However, we believe that identifying Trojan nets to be Trojan nets is the most important in Trojan detection, since we can examine all the Trojan nets and gates around the identified Trojan nets.

The parameter values γ and C are changed depending on the learned known netlists but their values are almost the same in our experiments, which means that we can use almost the same SVM classifier in all the cases. Even if we are given a new unknown netlist, we can just re-use the same SVM classifier and classify it into Trojan nets and normal nets.

In all the cases, Step L1 and Step L2 require three to ten hours and Step C1 and Step C2 require one to two hours.

Classification by Neural Networks

As in the discussion in the previous subsection, dynamic weighting is efficient for hardware Trojan detection. Hence we have experimented the NN-based hardware-Trojan classification method using dynamic weighting.

We have used 17 benchmarks summarized in Table 2.4. Since we have 17 benchmarks that is too small to split into the train, validation and test datasets, the experiments are performed with the leave-one-out cross validation. Table 2.7 shows the classification results by NN. Table 2.7 shows that the six benchmarks obtained the TPR value of 100%. This means that the NN classifier has successfully identified all the Trojan nets for these benchmarks.

Table 2.7: Experimental results of the NN classifier.

Netlist name	TN	FP	FN	TP	TPR	TNR
RS232-T1000	178	88	26	19	42%	67%
RS232-T1100	187	113	0	12	100%	62%
RS232-T1200	159	146	3	7	70%	52%
RS232-T1300	218	80	7	2	22%	73%
RS232-T1400	153	146	0	12	100%	51%
RS232-T1500	199	103	4	8	67%	66%
RS232-T1600	190	108	2	7	78%	64%
s15850-T100	1,835	594	3	24	89%	76%
s35932-T100	5,431	992	0	15	100%	85%
s35932-T200	5,577	842	2	14	88%	87%
s35932-T300	3,763	2,660	0	37	100%	59%
s38417-T100	4,194	1,613	0	12	100%	72%
s38417-T200	3,893	1,914	4	11	73%	67%
s38417-T300	4,419	1,388	11	33	75%	76%
s38584-T100	4,125	3,265	0	9	100%	56%
s38584-T200	6,103	1,277	15	185	93%	83%
s38584-T300	5,630	1,750	194	1,536	89%	76%

Table 2.8 shows the examples of parameter values of the output unit in our NN classifier. As shown in Table 2.8, these values are dependent on learned data and we cannot set up the identical NN classifier which can be applied to all the netlists. We have to set up the optimized NN classifier in each of the cases.

Table 2.9 shows the comparison between the results of the SVM classifier and those of the NN classifier. The bottom row in this table shows the average value of each column in this table. The average TPR value of the SVM classifier is 2% higher than that of the NN classifier. On the other hand, the average TNR value of the NN classifier is 20% higher than that of the SVM classifier. We have to select an appropriate classifier depending on the situations. How to develop an *identical* and *best* classifier is one of our important future works.

In all the cases, Step L1 and Step L2 require three to ten hours and Step C1 and Step C2 require one to two hours.

Table 2.8: Examples of the parameter values of the output unit in our NN classifier.

Netlist name	w_1^{out}	w_2^{out}	w_3^{out}	w_4^{out}	w_5^{out}	w_6^{out}	w_7^{out}
RS232-T1000	-0.79	0.89	-0.91	0.12	0.98	0.33	-0.64
RS232-T1100	-0.27	-0.67	0.12	0.26	0.89	0.33	1.13
RS232-T1200	1.09	-0.76	-0.24	1.28	-0.29	-0.84	-0.97
RS232-T1300	-0.47	0.19	1.07	-0.26	-0.76	0.54	0.83
RS232-T1400	1.44	0.41	-1.52	0.99	-0.66	1.09	-0.87
RS232-T1500	0.67	0.63	0.73	-0.75	0.92	0.68	0.95
RS232-T1600	-0.89	-1.05	-0.61	-0.23	0.28	-1.02	1.13
s15850-T100	0.33	0.63	-0.45	0.48	-1.04	-0.17	-0.31
s35932-T100	0.38	-0.27	0.99	0.83	-0.80	-0.31	0.78
s35932-T200	-0.83	0.85	1.19	0.84	1.37	0.91	0.22
s35932-T300	0.90	1.19	-0.75	-0.37	-0.34	-1.06	-1.43
s38417-T100	0.43	0.29	0.79	1.01	0.50	-0.55	0.81
s38417-T200	1.27	0.57	-1.35	-0.49	0.95	1.39	1.56
s38417-T300	0.18	0.92	-0.94	-0.83	-0.01	-1.02	1.14
s38584-T100	-0.69	-0.86	0.56	-0.39	-0.84	-0.87	0.42
s38584-T200	-1.03	-0.66	-0.41	0.81	-0.54	0.76	-0.77
s38584-T300	-1.00	0.96	-0.53	-0.93	0.62	-0.98	0.91

Comparison to the Existing Methods

As far as we know, [3] is the only method for static hardware-Trojan detection. However, TPR values are not shown there and hence we cannot compare our TPR values directly to them. In [3], truth tables are calculated for a given gate-level netlist and the suspicious flags are given to the gates whose output is rare to transit. Since this method is based on truth tables, it can be applied to combinational-triggered hardware Trojans and it is very hard to apply to sequential-triggered hardware Trojans. On the other hand, our proposed method only focuses on gate-level net features and hence it can be applied to both combinational-triggered and sequential-triggered hardware Trojans. In fact, RS232-T1200 includes sequential-triggered hardware Trojans [1] and our method realizes TPR of 80% as in Table 2.6 with dynamic weighting. In this point, we consider that our method is superior to [3]. In [26], a statistical technique for foundry identification is proposed, which is a static approach but not a hardware-Trojan detection method. We cannot compare our proposed method to [26] directly.

Now we pick up the method in [2], which gives one of the most recent results, for comparison purpose. Table 2.10 shows the comparison results between the method

Table 2.9: Comparison between SVM classifier and NN classifier.

	TPR		TNR	
Netlist name	SVM	NN	SVM	NN
RS232-T1000	53%	42%	31%	67%
RS232-T1100	58%	100%	27%	62%
RS232-T1200	80%	70%	26%	52%
RS232-T1300	89%	22%	26%	73%
RS232-T1400	83%	100%	22%	51%
RS232-T1500	83%	67%	24%	66%
RS232-T1600	89%	78%	26%	64%
s15850-T100	93%	89%	66%	76%
s35932-T100	93%	100%	60%	85%
s35932-T200	100%	88%	59%	87%
s35932-T300	27%	100%	58%	59%
s38417-T100	100%	100%	76%	72%
s38417-T200	73%	73%	76%	67%
s38417-T300	100%	75%	72%	76%
s38584-T100	100%	100%	62%	56%
s38584-T200	94%	93%	64%	83%
s38584-T300	89%	89%	66%	76%
Average	83%	81%	49%	69%

proposed in [2] and our proposed method with dynamic weighting. We pick up the results in [2] since they give most recent results in hardware-Trojan detection in gate-level netlists, which are just cited from [2]. The method in [2] uses signal correlation between Trojan nets and normal circuits and identify whether each net in an unknown netlist is Trojan or not, which is one of the most strong hardware-Trojan detection methods proposed so far. Table 2.10 shows that our method with dynamic weighting outperforms the method in [2] in most of the cases in terms of the TPR.

We further compare our proposed method to [2] using the detailed experimental results. In [2], the detailed experimental results for s35932-T200 are shown varying the threshold value. In [2], TPR becomes 27% and TNR becomes 99% as shown in Table 2.10 when its threshold value is fixed to 1.0 according to [2]. However, TPR becomes 100% and TNR becomes approximately 75% by changing the threshold value appropriately in [2], which is one of the best possible results. Our method using NN also uses the threshold value, which is fixed to 0.5, as described in Section 2.3.2. If the NN output value is equal to or larger than 0.5, the net is identified to be a Trojan net.

Table 2.10: Comparison between [2] and our proposed methods (with dynamic weighting).

	TPR			TNR		
Netlist name	[2]	Ours (SVM)	Ours (NN)	[2]	Ours (SVM)	Ours (NN)
s15850-T100	61%	93%	89%	99%	66%	76%
s35932-T200	27%	100%	88%	99%	59%	87%
s38417-T100	100%	100%	100%	99%	76%	72%
s38584-T200	99%	94%	93%	98%	64%	83%

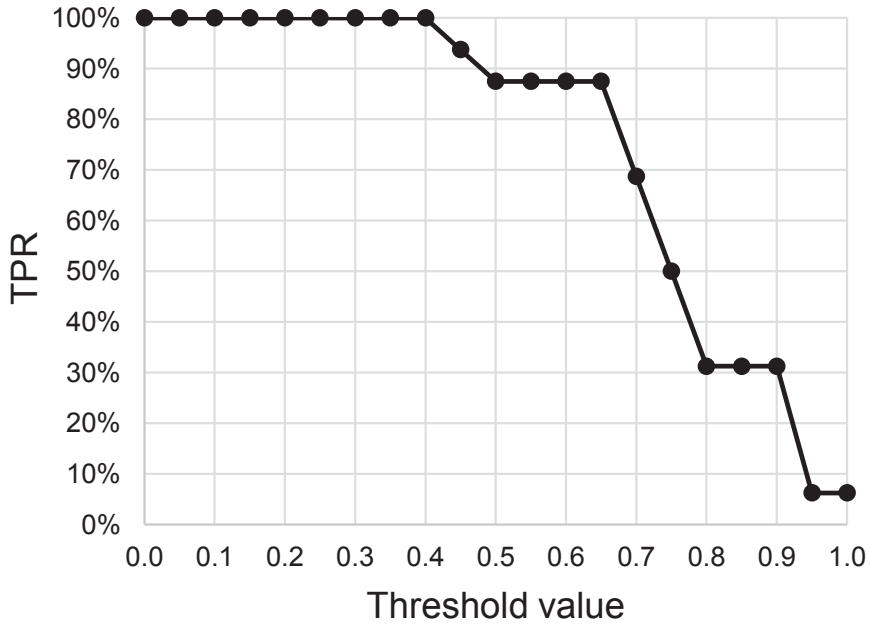
If the NN output value is smaller than 0.5, the net is identified to be a normal net. If this threshold value can be changed, the TPR-TNR trade-off curves vary in our method depending on the threshold value.

We have performed the experiments and obtained the detailed results for s35932-T200 by our method using NN varying the threshold value from 0.0 to 1.0. Figure 2.7 shows the detailed results. In Figure 2.7a, the x-axis shows the threshold value to identify the net to be a Trojan net and the y-axis shows the TPR value[7]. In Figure 2.7b, the x-axis shows the threshold value to identify the net to be a normal net and the y-axis shows the TNR value. For example, when the threshold value is set to be 0.4 in our method using NN, TPR becomes 100% and TNR becomes approximately 80%. These results outperform the results of [2], where TPR = 100% and TNR $\approx$ 75% when its threshold value is set to be 0.3 for s35932-T200 in [2]. As shown above, the values of TPR and TNR vary by changing the threshold value and our results can outperform the results of [2] by appropriately setting the threshold value.

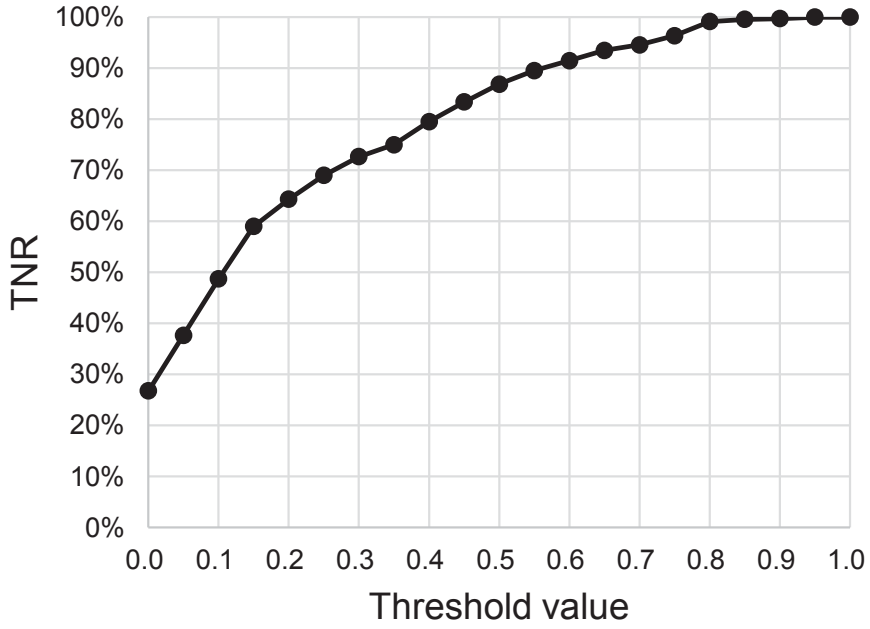
In a practical use, it is not useful to change the threshold value for an unknown test set, since we do not know its correct classification result beforehand. Furthermore, [2] uses another parameter which must be optimized for every benchmark circuit. On the other hand, our proposed method uses the static threshold value which is fixed to 0.5. Our method using NN does not use any other parameters and hence our method is strong in this point.

Furthermore, since the method in [2] is based on functional simulation, its TPR and TNR values can be much dependent on the functional simulation as well as input patterns. Particularly in large netlists, it is almost impossible to run functional simulation by giving all the possible input patterns. On the other hand, our proposed method does not require functional simulation nor logic simulation. The results obtained by our method are very static and thus, even if a large circuit is given, our method can identify a Trojan net just

[7]Since an NN output value sometimes becomes more than 1.0 or less than 0.0, TPR does not reach 0% even if the threshold value is 1.0. In the same way, TNR does not reach 0% even if the threshold value is 0.0.



(a) TPR for s35932-T200.



(b) TNR for s35932-T200.

Figure 2.7: TPR and TNR for s35932-T200 by our method using NN varying the threshold value.

Table 2.11: Classification matrix.

		NN	
		Normal	Trojan
SVM	Normal	3,706	93
	Trojan	1,873	763

based on the learned SVM or NN classifier.

2.3.4 Discussion

As discussed in Section 2.1, our final goal for hardware-Trojan detection is that we detect all parts of hardware Trojans in a chip and eliminate them.

For a practical use, we need to combine our method with the other existing methods. For example, s35932-T100 has $6,438$ ($= 6,423 + 15$) nets and we have to examine all of them to detect hardware Trojans originally. After our proposed method using NN is applied to them, the number of suspicious Trojan nets is decreased to $1,007$ ($= 992 + 15$) as in Table 2.7. Since TPR = 100% is realized for s35932-T100 by our method using NN, we have to only examine these 1,007 nets by using other existing methods. At that time, our proposed method takes up to several tens of minutes to classify the gate-level netlist using the learned classifier. We believe that decreasing the number of examined nets to just 16% ($1,007/6,438 \approx 0.156$) must be effective in complete hardware-Trojan detection. In this sense, our method must be utilized in a practical use.

Since TPR does not always become 100% in our method, the example above is an ideal case. Improving TPR values furthermore is an important future work.

The results of SVM and NN are not so similar to each other. For example, Table 2.11 shows the classification results of our method using SVM and NN for s35932-T200. This table shows that 1,873 nets are identified to be Trojan nets by SVM, but identified to be normal nets by NN. On the other hand, 93 nets are identified to be normal nets by SVM, but identified to be Trojan nets by NN. The other nets are classified into Trojan nets or normal nets by both of SVM and NN.

Now we combine the results. For example, assume that we identify the nets to be Trojan nets when both SVM and NN identify the nets to be Trojan nets, and we identify the other nets to be normal nets. In this case, the classification results are shown in Tables 2.12 and 2.13. Table 2.12 shows how true Trojan nets are classified by SVM and NN, where s35932-T200 includes 16 true Trojan nets. Table 2.13 shows how true normal nets are classified by SVM and NN, where s35932-T200 includes 6,419 true normal nets. Table 2.14 shows the results when the results of Table 2.12 and Table 2.13 are combined

Table 2.12: The number of classified nets as Trojan nets (s35932-T200).

		NN	
		Normal	Trojan
SVM	Normal	0	0
	Trojan	2	14

Table 2.13: The number of classified nets as normal nets (s35932-T200).

		NN	
		Normal	Trojan
SVM	Normal	3,706	93
	Trojan	1,871	749

Table 2.14: Prediction and answer (s35932-T200).

		Prediction	
		Normal	Trojan
Answer	Normal	5,610	749
	Trojan	2	14

under the assumption above.

As shown in Table 2.14, we have TN=5,610, FP=749, FN=2, and TP=14. In this case, TPR becomes 88% and TNR becomes 88%. The original results of our method using NN are: TN=5,577; FP=842; FN=2; and TP=14, whose TPR and TNR are 88% and 87%, respectively. If the results are combined, TNR is increased but TPR is not increased.

Tables 2.15–2.17 show the results of RS232-T1000. As shown in Table 2.17, we have TN=180, FP=86, FN=34, and TP=11. In this case, TPR becomes 24% and TNR becomes 68%. The original results of our method using NN are: TN=178; FP=88; FN=26; and TP=19, whose TPR and TNR are 42% and 67%, respectively. If the results are combined, TNR is increased but TPR is decreased.

As shown in the examples above, we may have better TNR values, when we identify the nets to be Trojan nets only if both SVM and NN identify the nets to be Trojan nets. But TPR value may be worsened. Just combining the results does not directly lead to better results. How to effectively combine the results must be one of the future works.

In the section, we have proposed a machine-learning-based hardware-Trojan classification method for gate-level netlists based on Trojan features. The experimental results demonstrate that the true positive rate of the proposed method is increased to up to 100%. Even if the proposed method is completely static not using any logic/functional simulations, the results are better than those obtained by the existing state-of-the-art dynamic detection method in terms of TPR in most cases.

As discussed in Section 2.1, we focus on maximizing TPR in this chapter. However, there is still room for improvement in terms of TNR and other machine learning metrics.

Table 2.15: The number of classified nets as Trojans nets (RS232-T1000).

		NN	
		Normal	Trojan
SVM	Normal	13	8
	Trojan	13	11

Table 2.16: The number of classified nets as normal nets (RS232-T1000).

		NN	
		Normal	Trojan
SVM	Normal	80	2
	Trojan	98	86

Table 2.17: Prediction and answer (RS232-T1000).

		Prediction	
		Normal	Trojan
Answer	Normal	180	86
	Trojan	34	11

For practical use, we can further apply existing hardware-Trojan detection methods after utilizing our proposed method. Since the TPR values of our proposed method are high compared to the existing method, our proposed method successfully pick up suspicious nets. In this sense, even if the TNR is low compared to existing methods, our proposed method can be utilized to screen test nets. In the next section, we discuss on the features that can be best applied to the machine-learning-based hardware-Trojan detection methods utilizing random forests.

2.4 Feature Extraction for Machine-Learning-Based Hardware Trojan Detection

2.4.1 Backgrounds

The purpose of hardware-Trojan detection is to prevent hardware-Trojan infected devices from being released to users. As the aforementioned discussion in Section 2.1, there are two goals for hardware-Trojan detection in IC design step. Since Goal 2 above tries to detect all parts of hardware Trojans, it includes Goal 1 and hence we focus on Goal 2, i.e., our goal in this chapter is that, given a netlist, we try to detect all the Trojan nets in it.

Supervised machine-learning-based hardware-Trojan detection is one the best ways to realize Goal 2 and solves the vicious circle problem in hardware-Trojan detection. By effectively using machine learning, we can update the hardware-Trojan database for new

types of hardware Trojans and detect them with the updated classifier. As discussed in Section 2.3.2, a supervised machine-learning-based hardware-Trojan detection method is proposed to realize Goal 2, in which the Trojan-net features proposed in [22] are used. However, there must exist many other possible features which describe Trojan nets and we do not know which ones are effective for machine learning. Machine-learning-based hardware-Trojan detection will be much improved if effective Trojan-net features are extracted from many sets of known netlists.

In this section, we extract effective Trojan-net features for supervised machine-learning-based hardware-Trojan detection and apply them to a random forest classifier. Then we classify a set of nets from a given gate-level netlist into Trojan nets and normal nets using the optimized random forest classifier.

Here we focus on hardware-Trojan detection using supervised machine learning in IC design step. First, we propose 51 gate-level Trojan-net features from given gate-level netlists which describe well Trojan nets very well. At that time, we utilize the random forest classifier [25], one of the strong machine-learning methods, and optimize it to apply to hardware-Trojan detection. Since random forest gives the *importance value* for every Trojan-net feature, we can effectively know which one contributes more than the others to detect hardware Trojans. We select the most effective 11 Trojan-net features which maximize the average F-measures.

The optimized random forest classifier using these 11 Trojan-net features achieves 100% true positive rate and 100% true negative rate in some cases and the best F-measures compared to existing machine-learning-based hardware Trojan detection methods.

Related Works

In machine learning, we have to extract the *appropriate* set of features appeared in Trojan nets. If it is too small, we cannot classify the nets in a given netlist correctly. If it is too large, we require many data for classification and much time to learn them. We cannot even classify the given nets correctly using all of these large number of Trojan-net features, which is known as *the curse of dimensionality* [37]. In this section, using a motivational example, we demonstrate that we should extract an appropriate set of Trojan-net features from a given netlist.

Table 2.18 shows several sample Trojan-net features from benchmarks in Trust-HUB. Let us focus on a net n in a given netlist from Trust-HUB. The feature "fan_in_5" (f_1) shows the number of fanins five-level away from the net n. The feature "out_nearest_pout" (f_2) shows the minimum level from n to any primary output. The feature "in_nearest_flipflop" (f_3) shows the minimum level to any flip-flop from the input side of the net n. In Table 2.18, the nets of (A)–(C) are Trojan nets and (D) and (E) are normal nets. Now we try to classify the nets (A)–(E) in Table 2.18 into Trojan nets and normal nets. For

Table 2.18: The examples of Trojan-net features.

#	Benchmark	Net name	fan_in_5 (f_1)	out_nearest_pout (f_2)	in_nearest_flipflop (f_3)	Trojan / Normal
(A)	RS232-T1000	iXMIT_state_1	59	2	4	Trojan
(B)	s35932-T100	Tj_OUT1	24	6	29	Trojan
(C)	s35932-T100	Trojan_SE	28	3	1	Trojan
(D)	RS232-T1600	n84	36	8	2	Normal
(E)	s35932-T100	n8403	7	6	38	Normal

simplicity, we give a single threshold value to every Trojan-net feature.

First, we assume that we classify the nets using f_1 only. When we set the threshold value of f_1 to 20, we consider that the nets of $f_1 > 20$ are Trojan nets and the others are normal nets. The underlined part _ in Table 2.18 shows $f_1 > 20$. In this case, the net (D) is identified to be a Trojan net and this is an incorrect classification. Therefore we cannot classify the nets correctly using f_1 only.

Next, we assume that we classify the nets using f_1 and f_2. When we set the threshold values of f_1 to 20 and that of f_2 to 7, we consider that the nets satisfying $f_1 > 20$ and $f_2 < 7$ are Trojan nets, and the others are normal nets. The underlined part $\underline{=}$ in Table 2.18 shows $f_2 < 7$. In this case, we classify all the nets (A)–(E) correctly into Trojan ones ((A)–(C)) and normal ones ((D) and (E)).

Finally, we assume that we classify the nets using f_1, f_2, and f_3. When we set the threshold values of f_1, f_2, and f_3 to 20, 7, and 10, respectively, we consider that the nets satisfying $f_1 > 20$, $f_2 < 7$, and $f_3 < 10$ are Trojan nets, and the others are normal nets. The underlined part $\underset{\sim}{\ }$ in Table 2.18 shows $f_3 < 10$. In this case, the net (B) is identified to be a normal net and this is an incorrect classification. We cannot classify the nets correctly if we set the threshold values such that $f_1 > 20$, $f_2 < 7$, and $f_3 < 10$. Even though we set another threshold value to f_3, we cannot correctly classify them using all of f_1, f_2, and f_3, either.

This simple example shows that it is better to use only f_1 and f_2 as Trojan-net feature values in case of Table 2.18, not to use all the Trojan-net features f_1, f_2, and f_3. We should extract an appropriate set of Trojan-net features for effective Trojan classification.

2.4.2 Trojan-Net Extraction for Hardware Trojan Detection

In this section, we refer to the first 12 benchmarks listed in Table 2.19 which is published at Trust-HUB for training sets and propose effective Trojan-net features to detect hardware Trojans. The benchmarks are gate-level netlists written in Verilog-HDL. In these benchmarks, we know beforehand which net is a Trojan net and which net is a normal

Table 2.19: The Trust-HUB benchmarks [1] used in the experiments.

Data name	Number of normal nets	Number of Trojan nets
RS232-T1000	283	36
RS232-T1200	289	34
RS232-T1300	287	29
RS232-T1400	273	45
RS232-T1500	283	39
s15850-T100	2,419	27
s35932-T100	6,407	15
s35932-T300	6,405	37
s38417-T100	5,798	12
s38417-T200	5,798	15
s38417-T300	5,801	44
s38584-T100	7343	19
RS232-T1100	284	36
RS232-T1600	292	29
s35932-T200	6,405	12
s38484-T200	7,343	127
s38484-T300	7,344	1,144
s35932-free	6,405	0
s38417-free	5,798	0
s38584-free	7,343	0

net. Then we propose several gate-level Trojan-net features which are likely to be related to Trojan nets.

Logic-gate fanins

As in RS232-T1000, some hardware Trojans are activated only when trigger conditions are satisfied. This is because hardware Trojans should be non-activated during IC test and/or normal conditions, where neither hardware vendors nor users can know that there are hardware Trojans in their ICs.

In the case of combinational-circuit triggers, trigger circuits require multiple logic gates since they have to implement complex trigger conditions. If the trigger has a very rare condition, the number of logic-gate fanins tends to become large. Since hardware Trojans tend to have very rare trigger conditions, the number of logic-gate fanins in

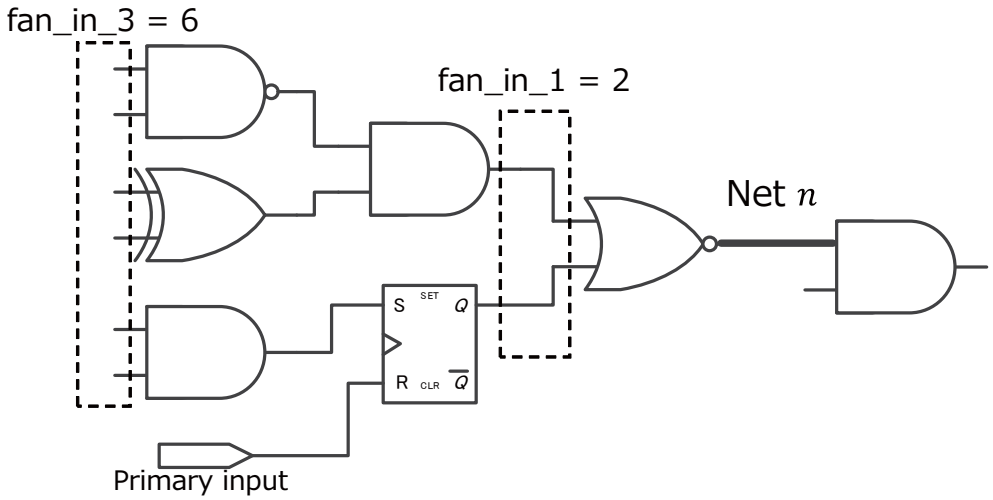


Figure 2.8: The example of Trojan-net features for logic-gate fanins.

Trojan nets must be large compared to that of normal nets.

Let n be a net in a given netlist throughout this section. From the discussion above, we extract the number of logic-gate fanins x-level away from every net n (fan_in_x) as a Trojan-net feature. In this chapter, we set $x = 1, 2, 3, 4, 5$, since the fanins which are 6 or more-level away from the net n are too far from n and they become less related to n.

For example, when we focus on the net n in Figure 2.8, its fan_in_1 becomes 2 and fan_in_3 becomes 6.

Flip-flops

As in RS232-T1200, some hardware Trojans have a sequential-trigger circuit. Since the hardware Trojans circuit is always so small, it is placed very locally. Hence the logic levels to/from flip-flops in the sequential-trigger circuit must be small enough.

From the discussion above, we extract the numbers of flip-flops up to x-level away from the input side and output side of the net n (in_flipflop_x and out_flipflop_x, respectively), and the level of the nearest flip-flops from the input side and output side of the net n (in_nearest_flipflop and out_nearest_flipflop, respectively) as Trojan-net features. In this chapter, we set $x = 1, 2, 3, 4, 5$ in the same way.

For example, when we focus on the net n in Figure 2.9, its in_flipflop_2 becomes 1 and out_nearest_flipflop becomes 3.

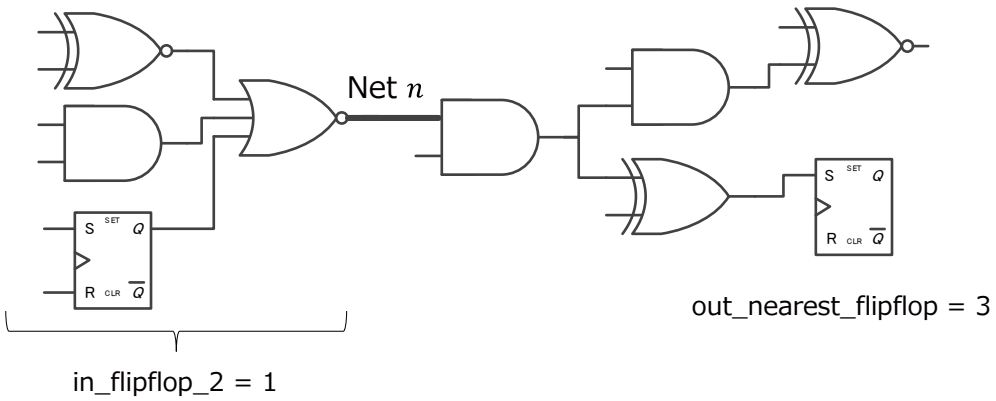


Figure 2.9: The example of Trojan-net features for flip-flops.

Multiplexers

As in s15850-T100, some hardware Trojans have multiplexers which receive trigger signals from trigger circuits and switch output signals to activate malfunctions. For example, secret internal signals are leaked through primary outputs by hardware Trojans. For this type of hardware Trojans, multiplexers connect internal signals to primary outputs when malfunctions are activated.

As discussed above, we extract the number of multiplexers up to x-level away from the input side and output side of the net n (in_multiplexer_x and out_multiplexer_x, respectively), and the level of the nearest multiplexers from the input side and output side of the net n (in_nearest_multiplexer and out_nearest_multiplexer respectively) as Trojan-net features. In this chapter, we set $x = 1, 2, 3, 4, 5$ in the same way.

For example, when we focus on the net n in Figure 2.10, its in_multiplexer_2 becomes 1 and out_nearest_multiplexer becomes 2.

Loops in a netlist

As in RS232-T1200, some hardware Trojans have sequential circuits for triggers. Sequential-trigger circuits often use looped flip-flops. We define an *m-level loop* as follows: the gate A connected to the input side or output side of the net n appears again at m-level away from the net n. A loop can be found not only in sequential-trigger circuits but also in ring-oscillator-type hardware Trojans.

Figure 2.11 shows an example of a loop in a netlist. The gate A is directly connected to the net n. At the same time, the gate A is also reachable from n when we focus on the bottom path in this figure, where the gate A is three-level away from n. We can find out a three-level loop ($m = 3$) for n in this example.

From the discussion above, we extract the numbers of up to x-level loops for the input

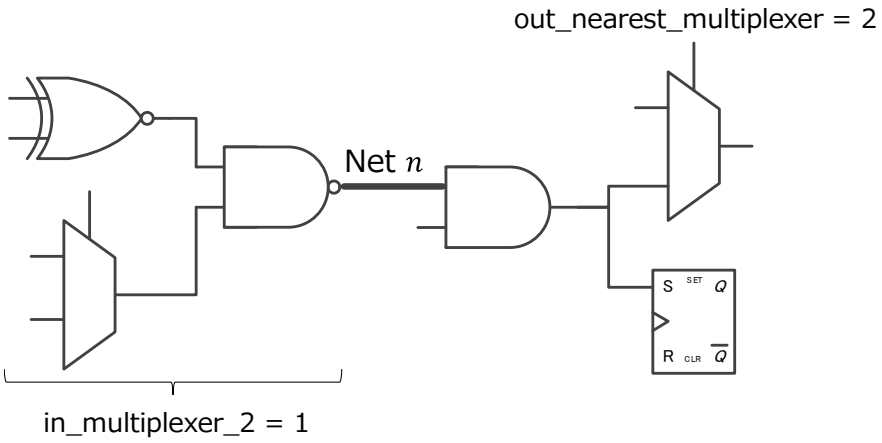


Figure 2.10: The example of Trojan-net features for multiplexers.

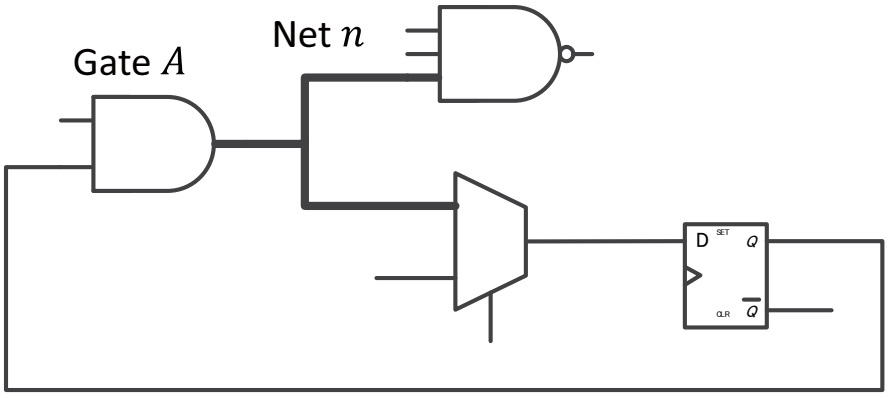


Figure 2.11: The example of a loop in a netlist (in_loop_3 for the net n).

side and output side of the net n (in_loop_x and out_loop_x, respectively) as Trojan-net features. In this chapter, we consider $x = 1, 2, 3, 4, 5$ in the same way.

Constants

In [22], it is reported that the net connecting a constant (i.e., one side of a net is fixed at 0 or 1) to the input of a flip-flop is likely to be a Trojan net. The net including constants becomes the efficient flag to detect hardware Trojans.

From the discussion above, we extract the numbers of constants up to x-level away from the input side and output side of the net n (in_const_x and out_const_x, respectively) as Trojan-net features. In this chapter, we consider $x = 1, 2, 3, 4, 5$ in the same way.

For example, when we focus on the net n in Figure 2.12, its in_const_2 becomes 1.

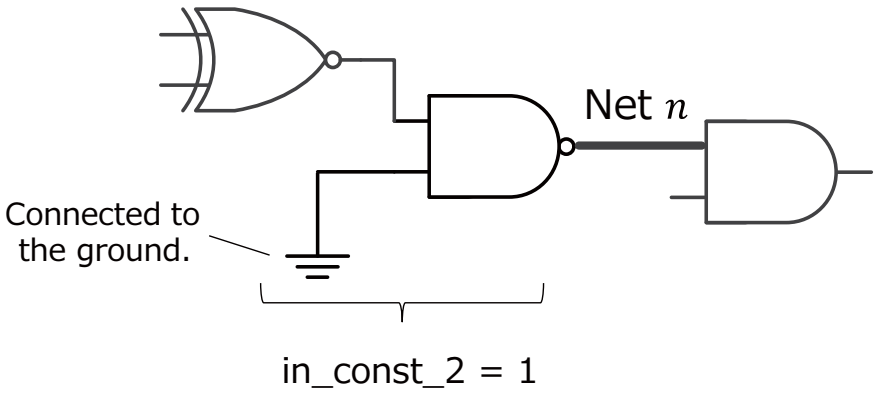


Figure 2.12: The example of Trojan-net features for constants.

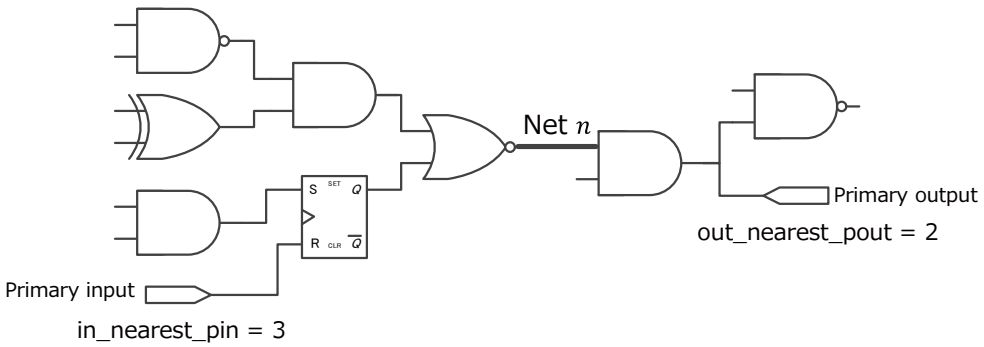


Figure 2.13: The example of Trojan-net features for primary inputs and outputs.

Levels to primary input and output

As in RS232-T1200, the primary inputs of ICs are often used as a trigger of hardware Trojans. Hardware Trojans are likely to be placed close to the primary inputs. The primary outputs of ICs are often used as the output ports of internal signals for malfunctions. Trojan nets must be connected close to the primary outputs to leak these internal signals.

From the discussion above, we extract the minimum levels from the net n to any primary input and output (in_nearest_pin and out_nearest_pout, respectively) as Trojan-net features.

For example, when we focus on the net n in Figure 2.13, its in_nearest_pin becomes 3 and out_nearest_pout becomes 2.

Trojan-net features extracted from a netlist

Based on all the discussions in this section, we finally propose the 51 gate-level Trojan-net features which must be related to hardware Trojans, as summarized in Table 2.20.

In Section 2.4.3, we carefully select a small set of the Trojan-net features out of these 51 features to effectively detect hardware Trojans.

Table 2.20: The extracted features from a netlist ($1 \leq x \leq 5$).

Trojan-net feature candidate	Description
fan_in_x	The number of logic-gate fanins x-level away from the net n.
in_flipflop_x	The number of flip-flops up to x-level away from the input side of the net n.
out_flipflop_x	The number of flip-flops up to x-level away from the output side of the net n.
in_multiplexer_x	The number of multiplexers up to x-level away from the input side of the net n.
out_multiplexer_x	The number of multiplexers up to x-level away from the output side of the net n.
in_loop_x	The number of up to x-level loops from the input side of the net n.
out_loop_x	The number of up to x-level loops from the output side of the net n.
in_const_x	The number of constants up to x-level away from the input side of the net n.
out_const_x	The number of constants up to x-level away from the output side of the net n.
in_nearest_pin	The minimum level to the primary input from the net n.
out_nearest_pout	The minimum level to the primary output from the net n.
{in, out}_nearest_flipflop	The minimum level to any flip-flop from the input or output side of the net n.
{in, out}_nearest_multiplexer	The minimum level to any multiplexer from the input or output side of the net n.

2.4.3 Feature Selection Utilizing Random Forest

In this section, we firstly summarize several supervised machine-learning approaches and find out that random forest [25] is best applied to hardware-Trojan detection. After that, we carefully select the Trojan-net features to detect hardware Trojans from the 51 Trojan-net features extracted in Section 2.4.2 using some of the Trust-HUB benchmarks [1] for training sets and the optimized random forest classifier.

Machine Learning Approach Best Applied to Hardware-Trojan Detection

Firstly, we pick up several supervised machine-learning approaches and discuss which one is the best to apply to hardware-Trojan detection.

Support vector machine (SVM) [29] is one of the supervised machine-learning approaches which classifies input data into two groups. In [4], an SVM-based hardware-Trojan classification method is proposed which successfully detects Trojan nets from an unknown netlist in some cases. However, their results show that the method sometimes has many false positives and hence true negative rate does not become so high. As in the discussion in Section 2.4.2 as well as [22, 38], Trojan nets may have many features which are independent of each other and these features are distributed intricately. Hence SVM cannot separate them accurately in some cases.

Neural networks [31] are also one of the supervised machine-learning approaches. Recently, this approach has been applied to image recognition [39] and natural language processing [40] and leads to very good results. However, in hardware-Trojan detection, we have to detect a set of Trojan nets whose number is much smaller than the total number of normal nets as shown in Table 2.19. In this sense, hardware-Trojan detection is completely different from image recognition and natural language processing.

Random forest [25] is one of the supervised machine-learning method using several *decision trees*. Each decision tree uses a subset of features randomly sampled from a set of entire input features. As described above, we have to deal with many independent Trojan-net features effectively in hardware-Trojan detection and every decision tree in the random forest can be applied to an effective subset of those of many Trojan-net features. Even though a given netlist includes a very small number of Trojan nets, classification based on decision trees can lead to good results.

While there are other supervised machine-learning approaches, we believe that no approaches other than random forest can lead to good results since a given unknown netlist has a very small number of Trojan nets and these Trojan nets have many independent features.

Based on this discussion, we pick up random forest and apply it to hardware-Trojan detection. Moreover, random forest can calculate the importance of every feature and we can know which ones contribute to lead good results the best. By using random

forest, we expect that we select the best set of Trojan-net features among many features extracted in Section 2.4.2.

Selection of the Best Possible Trojan-Net Features

Although we have extracted the 51 features which must be related to hardware Trojans, It must be impractical to use all the 51 Trojan-net features for hardware-Trojan classification, as discussed in Section 2.4.1. We have to select the best set of Trojan-net features which most effectively detects hardware Trojans among the 51 Trojan-net features. Based on the discussion in Section 2.4.3, we apply a random forest classifier to select the best set of Trojan-net features. In random forest, we can obtain the *importance value* [25] of every Trojan-net feature used which describes how much important every feature is to classify the nets into Trojan ones and normal ones. We use these importance values to select the best set of Trojan-net features to effectively classify the nets into Trojan ones and normal ones.

We employ the two-step approach as follows: In Step 1, we first extract the Trojan-net feature values from given netlists; In Step 2, we select the best set of Trojan-net features based on the importance values obtained by the optimized random forest classifier.

Step 1: Extract Trojan-net feature values from a netlist In Step 1, we first extract the Trojan-net feature values of each net n from every given netlist. As in Section 2.4.2, we also use the first 12 Trust-HUB benchmarks in Table 2.19 as given netlists. We use the 51 Trojan-net features here as listed in Table 2.20.

Step 2: Select the best set of Trojan-net features In Step 2, we select the best set of Trojan-net features based on the extracted feature values in Step 1 using the random forest classifier optimized to hardware-Trojan detection.

Evaluation indexes of machine learning for hardware-Trojan detection

In case of binary classification, there are four values to evaluate the classification results: the true negative value (TN), the false positive value (FP), the true positive value (TP) and the false negative value (FN); TN shows the number of normal nets identified to be normal nets; FP shows the number of normal nets identified to be Trojan nets mistakenly; TP shows the number of Trojan nets identified to be Trojan nets; FN shows the number of Trojan nets identified to be normal nets mistakenly.

Based on the values of TN, FP, TP and FN, we can have five more values to evaluate the classification results: the true positive rate (TPR), the true negative rate (TNR), the precision, the F-measure, and the accuracy; TPR is defined by TP/(TP+FN), and

also known to be the *recall* (R); TNR is defined by TN/(TN+FP); The precision, P, is defined by $P = \mathrm{TP}/(\mathrm{FP} + \mathrm{TP})$; The F-measure, F, is the harmonic mean of the precision and the recall, and represented by $F = 2PR/(P + R)$; The accuracy is defined by (TP+TN)/(TP+TN+FP+FN).

All of the values above are important but, specifically, the F-measure is the best to measure the classification results well [41]. In this chapter, we focus on the F-measure to evaluate the classification results for machine learning.

Optimizing a random forest classifier applied to hardware-Trojan detection

In order to construct random forest, we have to optimize several parameters best applied to hardware Trojan detection. Random forest has the three parameters: 1) the number of maximum features in each decision tree, 2) the depth of each decision tree, and 3) the number of decision trees used in a random forest classifier.

1. **The number of maximum features in each decision tree:** It is generally set to be $\sqrt{n}$, where n is the number of all the features used [42]. Firstly, we set the number of maximum features to $\sqrt{n}$ as well.

2. **The depth of each decision tree:** We set no specific depth for each decision tree. Every decision tree grows until all its leaf nodes reach the identical classification result and we may have the best classification result in this sense. Note that, as the experiments in Section 2.4.4 demonstrates, the learning process just takes approximately several minutes even if we do not set a specific depth value and hence we consider that setting no specific depth in hardware-Trojan detection is reasonable.

3. **The number of decision trees:** We have determined how many decision trees are constructed in a random forest classifier by performing the preliminary experiments.

   We have set it to 2, 5, and 8. All the 51 Trojan-net features have been used here. The experimental setup was the same as the one described below. Table 2.21 shows the results of the preliminary experiments. As shown in Table 2.21, we can obtain the best F-measure value when we construct five decision trees and hence we set the number of decision trees to five.

Based on the discussion above, we can appropriately set the parameters in random forest for hardware-Trojan detection.

Hardware-Trojan detection using the optimized random forest classifier

Table 2.21: The number of trees and F-measures

# of trees	F-measure
2	60.0%
5	66.9%
8	64.7%

Now we classify all the nets in a given netlist from the first 12 Trust-HUB benchmarks in Table 2.19 into Trojan ones and normal ones based on the 51 features listed in Table 2.20 using the optimized random forest classifier. We use the leave-one-out cross-validation method [36] to validate classification results, where each one of the benchmarks is considered to be a test set (unknown data set) and the others are considered to be training sets (known data sets).

Since the number of Trojan nets (N_t) is much smaller than that of normal nets (N_n) in hardware-Trojan classification, we learn Trojan nets (N_n/N_t) times. As a result, we can obtain the F-measure for each of the benchmarks. We calculate the average F-measure values over these 12 benchmarks as the measure of the classification performance.

Selecting the best set of Trojan-net features

Step 2-1: First, we repeatedly halves the number of Trojan-net features. At first, as a learning result of the random forest classification above, it gives the importance value to each one of the 51 Trojan-net features. For every benchmark, we have 51 importance values and then we can have an average importance value over 12 benchmarks for each of 51 Trojan-net features. Then we select the first half of Trojan-net features whose average importance values are higher than those of the second half. Then, we also classify the given netlists using the selected Trojan-net features and obtain the average F-measure value. We repeatedly perform this process until the average F-measure value decreases the previous one.

Table 2.22 shows the average F-measure values when we gradually decrease the number of Trojan-net features from 51 to 6. From this table, the 12 Trojan-net features give the best value in terms of F-measure.

Step 2-2: Next, we carefully examine the 12 Trojan-net features selected above. We pick up one of the 12 Trojan-net features and discard it. We classify the given netlists using the remaining Trojan-net features and obtain the average F-measure value. We try this process for each of the 12 Trojan-net features and really discard the one when the best average F-measure value is obtained. We repeat this process until no further improvement is seen.

Table 2.22: Selecting the best set of features (Step 2-1).

# of features	F-measure
51	66.9%
25	75.7%
12	75.9%
6	57.2%

Table 2.23: Selecting the best set of features (Step 2-2).

# of features	F-measure
12	75.9%
11	79.3%
10	74.0%

Table 2.24: The best set of 11 Trojan-net features and their importance values.

No.	Trojan-net feature	Average importance value
1	fan_in_4	0.056
2	fan_in_5	0.070
3	in_flipflop_4	0.084
4	out_flipflop_3	0.115
5	out_flipflop_4	0.070
6	in_loop_4	0.056
7	out_loop_5	0.133
8	in_nearest_pin	0.043
9	out_nearest_pout	0.200
10	out_nearest_flipflop	0.124
11	out_nearest_multiplexer	0.048

Table 2.23 shows the results. When we select the 11 Trojan-net features, we can have the *best* average F-measure value.

Table 2.24 summarizes the resultant best set of the 11 gate-level Trojan-net features and their average importance values.

From these results, the best set of Trojan-net features is composed of the number of fanins, the number of flip-flops, the number of loops, the minimum level to primary inputs and outputs, the minimum level to flip-flops from output side, and the minimum level to multiplexers from output side.

2.4.4 Experimental Results

In this section, we demonstrate the results of hardware-Trojan classification using the 11 gate-level Trojan-net features obtained in Section 2.4.3. We use the random forest

Table 2.25: The classification results utilizing the extracted 11 features.

Test data	TN	FP	FN	TP	TPR	TNR	Precision	F-measure	Accuracy
RS232-T1000	278	5	0	36	100.0%	98.2%	87.8%	93.5%	98.4%
RS232-T1200	289	0	2	32	94.1%	100.0%	100.0%	97.0%	99.4%
RS232-T1300	287	0	0	29	100.0%	100.0%	100.0%	100.0%	100.0%
RS232-T1400	273	0	0	45	100.0%	100.0%	100.0%	100.0%	100.0%
RS232-T1500	282	1	1	38	97.4%	99.6%	97.4%	97.4%	99.4%
s15850-T100	2,418	1	3	24	88.9%	100.0%	96.0%	92.3%	99.8%
s35932-T100	6,407	0	4	11	73.3%	100.0%	100.0%	84.6%	99.9%
s35932-T300	6,405	0	2	35	94.6%	100.0%	100.0%	97.2%	100.0%
s38417-T100	5,798	0	7	5	41.7%	100.0%	100.0%	58.8%	99.9%
s38417-T200	5,798	0	9	6	40.0%	100.0%	100.0%	57.1%	99.8%
s38417-T300	5,801	0	1	43	97.7%	100.0%	100.0%	98.9%	100.0%
s38584-T100	7,342	1	16	3	15.8%	100.0%	75.0%	26.1%	99.8%
RS232-T1100	280	4	17	19	52.8%	98.6%	82.6%	64.4%	93.4%
RS232-T1600	288	4	1	28	96.6%	98.6%	87.5%	91.8%	98.4%
s35932-T200	6,405	0	11	1	8.3%	100.0%	100.0%	15.4%	99.8%
s38584-T200	7,332	11	45	82	64.6%	99.9%	88.2%	74.5%	99.3%
s38584-T300	7,335	9	29	1,115	97.5%	99.9%	99.2%	98.3%	99.6%
RS232-free	298	8	0	0	100.0%	97.4%	0.0%	0.0%	97.4%
s15850-free	2,417	2	0	0	100.0%	99.9%	0.0%	0.0%	99.9%
s35932-free	6,405	0	0	0	100.0%	100.0%	100.0%	100.0%	100.0%
s38417-free	5,798	0	0	0	100.0%	100.0%	100.0%	100.0%	100.0%
s38584-free	7,343	0	0	0	100.0%	100.0%	100.0%	100.0%	100.0%

classifier optimized as in the previous section implemented in Python using the scikit-learn machine-learning library [35]. The experiments are performed using an Intel Xeon E7-4870 computer environment with 300GB memory.

We use all the 22 netlists in Table 2.19 for the experiments, where the last ten netlists are newly added to evaluate whether the selected 11 Trojan-net features can be effectively applied to them. The last five netlists are Trojan-free netlists. Since the datasets are too small to split into train, validation, and test datasets, we apply the leave-one-out cross-validation method [36] to evaluate Trojan-included netlists, where each one of the Trojan-included netlists is considered to be a test set (unknown data set) and the other 16 Trojan-included netlists are considered to be training sets (known data sets). In evaluating the Trojan-free netlist, we consider the 17 Trojan-included netlists to be training sets and the Trojan-free netlists to be test sets and then we evaluate the classification result in the Trojan-free netlists.

Classification Results

Table 2.25 shows the classification results. We have obtained 100% TPR in RS232-T1000, RS232-T1300, and RS232-T1400, i.e., all the Trojan nets in these netlists are correctly identified to be Trojan nets in these cases. All the benchmarks realized 98% or more TNR values.

Since we have obtained 100% precision in more than half of benchmarks, all the nets identified to be Trojan nets are actually Trojan nets in these benchmarks, even if they have a large number of nets. As the FP values in almost all the netlists become less than five, the normal nets in these benchmarks are also correctly identified to be normal nets.

In terms of accuracy, most benchmarks realized more than 99% or more. This means that most nets are correctly classified into a set of Trojan nets and a set of normal nets. Goal 2 described in Section 2.1 is mostly realized, although we still have several false positives and false negatives.

Note that, Table 2.25 demonstrates that we find out at least one Trojan net in Trojan-included netlists while we find out no Trojan nets in the three Trojan-free netlists. These results simply imply that we can successfully classify almost the given netlists into Trojan-included netlists and Trojan-free netlists, i.e., Goal 1 described in Section 2.1 is also mostly realized.

However, we obtained several FP nets[8] in RS232-free and s15850-free. In RS232-free, all of the nets identified to be Trojan nets mistakenly are around the border between normal nets and Trojan nets in other Trojan-inserted netlists RS232-T1000–RS232-T1600. The nets around the border between normal nets and Trojan nets have similar feature values and thus the random forest classifier identifies these nets to be Trojan nets mistakenly. In s15850-free, two nets are identified to be Trojan nets mistakenly. For s15850 series, we learn only one benchmark s15850-T100 and thus we obtained several FP nets in s15850-free. Actually two FP nets have similar feature values to Trojan nets in s15850-T100. Since the ratios of FP nets over all the nets are less than 3% in RS232-free and s15850-free, we can re-inspect the FP nets in them again and see if they are really Trojan nets or normal nets carefully. However, it is better for us to correctly identify the Trojan-free netlists to be Trojan free for the practical use of machine-learning-based hardware-Trojan detection. This is our important future work.

In the experiments, it takes 10 minutes to learn 16 netlists with random forest and it takes 10 seconds to classify an unknown netlist.

[8]FP nets refer to the normal nets identified to be Trojan nets mistakenly. Similarly, we can define TN nets, FN nets and TP nets.

Comparison to Existing Methods

Comparison to machine-learning-based hardware-Trojan detection methods An SVM-based hardware-Trojan detection method is proposed in [4], which classifies a set of nets in a gate-level netlist into normal nets and Trojan nets. In [4], the five Trojan-net features are used. For comparison purpose, we gave the proposed 11 Trojan-net features to [4] and obtained the classification results ("[4] with 11 features" in Table 2.26).

Table 2.26 summarizes the comparison results in terms of precision, F-measure, and accuracy compared to [4] and "[4] with 11 features". As in Table 2.26, the average precision, F-measure, and accuracy using our proposed Trojan-net features become larger than those of the existing methods. Even if the proposed 11 Trojan-net features are given to [4], its classification results do not outperform our results. This is because the 11 Trojan-net features from given netlists are independent of each other and these features are distributed intricately, as discussed in Section 2.4.3. SVM-based approach cannot separate them accurately.

Overall, the proposed 11 Trojan-net features give one of the best sets of Trojan-net features to detect hardware Trojans. Note that F-measure of 79.3% is a quite good value in machine learning [43]. The average accuracy of 99.2% is also a quite good value in hardware-Trojan classification.

As discussed in Section 2.4.3, a neural network is another machine-learning-based approach. Neural networks can also be applied to hardware-Trojan detection. An example structure of the neural networks appears in [44] which uses the same 11 features in a given netlist as our proposed method.

However, as shown in [44], when we apply the neural networks to hardware-Trojan detection, the average TPR value becomes 85% while the average TNR value becomes 70%, and thus the average F-measure value becomes only 25%. In Table 2.26, the average F-measure of [4] with 11 features is 49.4% and that of our proposed method is 79.3%, and therefore both of them are larger than that of [44]. In this sense, the random forest classifier is superior to the neural networks. Neural networks can be very effectively applied to image recognition and natural language processing as discussed in Section 2.4.3, but they cannot be well applied in a simple way to hardware-Trojan detection compared to the optimized random forest classifier in terms of F-measures.

Table 2.26: The comparison to existing methods.

	Precision			F-measure			Accuracy		
Test data	[4]	[4] with 11 features	Ours	[4]	[4] with 11 features	Ours	[4]	[4] with 11 features	Ours
RS232-T1000	11.5%	92.3%	87.8%	19.0%	96.0%	93.5%	34.1%	99.1%	98.4%
RS232-T1200	3.4%	100.0%	100.0%	6.5%	90.3%	97.0%	27.3%	98.1%	99.4%
RS232-T1300	3.5%	100.0%	100.0%	6.7%	98.2%	100.0%	27.7%	99.7%	100.0%
RS232-T1400	4.1%	97.6%	100.0%	7.8%	94.3%	100.0%	24.4%	98.4%	100.0%
RS232-T1500	4.1%	97.4%	97.4%	7.9%	96.1%	97.4%	25.8%	99.1%	99.4%
s15850-T100	2.9%	80.0%	96.0%	5.7%	25.0%	92.3%	66.0%	99.0%	99.8%
s35932-T100	0.5%	80.0%	100.0%	1.1%	40.0%	84.6%	59.8%	99.8%	99.9%
s35932-T300	0.4%	45.8%	100.0%	0.7%	36.1%	97.2%	57.8%	99.4%	100.0%
s38417-T100	0.8%	100.0%	100.0%	1.7%	15.4%	58.8%	75.7%	99.8%	99.9%
s38417-T200	0.8%	100.0%	100.0%	1.5%	23.5%	57.1%	75.7%	99.8%	99.8%
s38417-T300	2.6%	57.1%	100.0%	5.1%	27.6%	98.9%	71.8%	99.3%	100.0%
s38584-T100	0.3%	0.0%	75.0%	0.6%	0.0%	26.1%	61.9%	99.7%	99.8%
RS232-T1100	3.1%	92.1%	82.6%	5.9%	94.6%	64.4%	28.2%	98.8%	93.4%
RS232-T1600	3.5%	91.7%	87.5%	6.7%	83.0%	91.8%	27.7%	97.2%	98.4%
s35932-T200	0.6%	50.0%	100.0%	1.2%	14.3%	15.4%	59.3%	99.8%	99.8%
s38584-T200	6.5%	0.0%	88.2%	12.2%	0.0%	74.5%	64.4%	98.1%	99.3%
s38584-T300	38.5%	100.0%	99.2%	53.8%	4.8%	98.3%	70.8%	86.9%	99.6%
Average	5.1%	75.5%	94.9%	8.5%	49.4%	79.3%	50.5%	98.3%	99.2%

Table 2.27: False positive rates in cited from [3].

Benchmark categories	[3]
RS232	about 8%
s15850	less than 1%
s35932	less than 1%
s38417	less than 1%

Table 2.28: False positive rates in ours.

Benchmark categories	Ours
RS232	0.70%
s15850	0.04%
s35932	0.00%
s38417	0.00%

Comparison to a static hardware-Trojan detection method Our proposed method focuses on *static* hardware-Trojan detection which just uses hardware-Trojan related information without simulating the circuits nor actually running them. Though many hardware-Trojan detection methods have been proposed, most of them are *dynamic* hardware-Trojan detection methods such as in [24] and [21] which require to simulate the circuits or actually run them. One of the most effective static method for hardware-Trojan detection is FANCI [3] and thus we have compared our method to it. Table 2.27 shows the results in [3] from the viewpoint of false positive rate (FPR) which is defined by the number of FP nets over the number of normal nets.

Since the benchmarks used are not explicitly specified in [3], we cannot compare our results to [3] on a one-to-one basis. Now we compare our results to [3] in an equivalent way.

In [3], the benchmarks used are not specified but the benchmarks RS232 series, s15850 series, s35932 series, and s38417 series are grouped into RS232, s15850, s35932, and s38417, respectively, and Table 2.27 summarizes the results. In the same way, we group our benchmarks and summarize the results in Table 2.28. For example, when we focus on RS232 series in Table 2.25, the total number of normal nets in RS232-T1000–RS232-T1600 becomes 1,991 while the total FP becomes 14, then the FPR can be calculated by $14/1991 \simeq 0.70\%$. As shown in Tables 2.27 and 2.28, our method outperforms [3] from the view point of FPR in all benchmark categories.

Note that, as far as we know, since none of other static hardware-Trojan detection methods describes the statistical data such as the number of nets, TN, FP, FN and TP, we cannot compare our method to them directly.

Discussion

Analysis of FN nets To realize complete detection discussed in Section 2.1, we have to minimize FN and FP values. FN shows the number of Trojan nets identified to be normal nets mistakenly and thus we should minimize FN with the highest priority. Now we focus on the ratio of FN over TP which is defined by FN/TP in Table 2.25. The netlists s35932-T200 and s38584-T100 have the first and second largest FN/TP values and therefore we focus on them. Additionally, we focus on FN especially in s38584-T200, which obtained the largest FN in Table 2.25. We also focus on FN in RS232-T1100, which obtained the largest FN other than s38584 series.

Each Trojan circuit in s35932-T200 and s38584-T100 is composed of a trigger circuit and a payload circuit. The trigger circuit is a comparator and the payload circuit changes the functionality of the original circuit [1]. All FN nets are included in the trigger circuit. Since the trigger circuits in s35932-T200 and s38584-T100 have small number of logic gates and levels, the nets in the trigger circuits have similar feature values to those of normal nets. In this sense, the random forest classifier identified Trojan nets to be normal nets mistakenly.

The Trojan circuit in s38584-T200 is composed of a trigger circuit and a payload circuit. The trigger circuit is a counter of a rare-vector and the payload circuit leaks the value of one internal signal [1]. Most FN nets are included in the trigger circuit, especially in the counter circuit. A counter circuit is a generally used in a normal circuit and thus the random forest classifier identified these nets to be normal nets mistakenly.

The Trojan circuit in RS232-T1100 is also composed of a trigger circuit and a payload circuit. The trigger circuit is a sequential comparator and the payload circuit gains control over one internal signal [1]. All of the FN nets are included in the trigger circuit, especially near the border between Trojan nets and normal nets. Other RS232 benchmarks also obtained FN nets around the border between Trojan nets and normal nets. Thus the random forest classifier identified these nets to be normal nets mistakenly.

As discussed above, FN nets occur (1) when the trigger circuits have small number of gates and their logic-levels are small, (2) when the nets are frequently used in normal nets, and (3) when the nets are around the border between Trojan nets and normal nets. To detect these FN nets, we can apply other hardware-Trojan detection method such as [3] and/or [23]. As shown in Table 2.25, when we obtain Trojan-identified nets, we can further examine the nets themselves and their nearby nets using [3] and/or [23]. However, our goal is to realize complete detection with a machine-learning-based hardware-Trojan detection method and thus optimizing the classifier furthermore is our important future work.

Low precision, F-measure, and accuracy results In Table 2.26, RS232-T1000 and RS232-T1100 obtained low precision, low F-measure, and low accuracy compared to "[4] with 11 features". Table 2.29 shows the comparison results of RS232-T1000 and RS232-T1100 between our proposed method and "[4] with 11 features".

Table 2.29: Comparison of RS232-T1000 and RS232-T1100 between our proposed method and [4] with 11 features.

Test data	Method	TN	FP	FN	TP	TPR	TNR	Precision	F-measure	Accuracy
RS232-T1000	Ours	278	5	0	36	100.0%	98.2%	87.8%	93.5%	98.4%
	[4] with 11 features	280	3	0	36	100.0%	98.9%	92.3%	96.0%	99.1%
RS232-T1100	Ours	280	4	17	19	52.8%	98.6%	82.6%	64.4%	93.4%
	[4] with 11 features	281	3	1	35	97.2%	98.9%	92.1%	94.6%	98.8%

In RS232-T1000, "[4] with 11 features" obtained 3 FP nets while the proposed method obtained 5 FP nets. In RS232-T1100,"[4] with 11 features" obtained only one FN net while the proposed method obtained 17 FN nets.

According to Table 2.26, "[4] with 11 features" tends to identify normal nets to be Trojan nets and thus the average precision becomes lower than our proposed method. On the other hand, our proposed method tends to identify Trojan nets to be normal nets and thus FN is larger than FP in most of the netlists listed in Table 2.25. Mistakenly-identified nets in our method are placed near the border between Trojan nets and normal nets, and thus the features are similar to each other. Whether the net near the border between Trojan nets and normal nets is identified to be a normal net or Trojan net depends on the classifier applied. The property of the classifier that we apply to hardware-Trojan detection decides whether the nets with ambiguous feature values are classified into Trojan nets or normal nets.

In RS232-T1000, the FP in our proposed method is larger than that in "[4] with 11 features". Three FP nets in both of the methods are the same nets. The other two FP nets in our proposed method can be mistakenly identified to be Trojan nets due to the randomness of random forest classifier. This is because these nets are also placed near the border between Trojan nets and normal nets. Note that in order to demonstrate the randomness of the random forest classifier, we have applied our method to RS232-T1000 with another random seed. The experimental results show that we have also obtained three FP nets in RS232-T1000, which are the same as the ones in "[4] with 11 features".

In RS232-T1100, "[4] with 11 features" obtained only one FN net while the proposed method obtained 17 FN nets. This is because the classifier in our method identifies Trojan nets around the border to be normal nets mistakenly. In RS232-T1100, 4 FP nets are also obtained in the proposed method while "[4] with 11 features" obtained 3 FP nets. Because of the randomness of the random forest classifier, one extra FP net is obtained in the proposed method. Note that, we have applied our method to RS232-T1100 with another random seed and the experimental results show that we have also obtained three FP nets in RS232-T1100, which are the same as the ones in "[4] with 11 features".

In the future, we work for maximizing TPR and TNR values in individual benchmarks by considering around the Trojan-identified nets.

2.5 Conclusion

In the chapter, we have proposed a machine-learning-based hardware-Trojan classification method for gate-level netlists based on Trojan features.

In order to realize it, we first discuss on how to apply machine learning to hardware Trojan detection (Section 2.3). The experimental results demonstrate that the true positive

rate of the proposed method is increased to up to 100%. Even if the proposed method is completely static not using any logic/functional simulations, the results are better than those obtained by the existing state-of-the-art dynamic detection method in terms of TPR in most cases. As discussed in Section 2.1, we focus on maximizing TPR in this chapter. This approach has not been proposed as far as we know, hence these results provide us the meaningful suggestion that machine learning can be applied to hardware Trojan detection.

After the discussion on how to apply machine learning to hardware Trojan detection, we discuss on the Trojan features that can be best applied to machine-learning-based hardware-Trojan detection. In Section 2.4, we have first proposed the 51 Trojan-net features which well describe Trojan nets from netlists. Next, we have selected the best set of 11 Trojan-net features which maximize the average F-measures using the optimized random forest classifier. The experimental results show that the maximum TPR and TNR realize 100% in several benchmarks and the F-measure becomes larger than existing state-of-the-art methods.

As summarized above, this chapter shows that machine learning will effectively detect hardware Trojans. In particular, the extracted feature values suggest that typical Trojan nets have identical features, for example of a large number of fan-ins because of a rare-trigger circuit. These features can be important clues to detect hardware Trojans. However, there still remains to be improved in terms of TPR and TNR. The next chapter proposes several approaches to enhance the machine-learning-based hardware-Trojan detection methods and to leverage their results to the hardware products.

Chapter 3

Application of the Hardware-Trojan Detection Utilizing Machine Learning[1]

3.1 Introduction

The purpose of hardware-Trojan detection is to prevent hardware-Trojan infected netlists from being released to users. As discussed in Section 2.1, there are the two goals for hardware-Trojan detection for gate-level netlists: partial detection (Goal 1) and complete detection (Goal 2). In this chapter, we focus on the Goal 2.

In [38], a hardware-Trojan detection method for gate-level netlists is proposed targeting the Goal 1 above. In [38], hardware-Trojan-specific netlist features are extracted manually from the netlists in [1] and hardware-Trojan infected netlists are distinguished from hardware-Trojan free netlists. Since this method is based on manually extracting hardware-Trojan-specific netlist features, it must take much time. Further, it is impractical to manually extract Trojan-net features whenever new types of hardware Trojans are found.

Chapter 2 proposes the first method to apply machine learning to hardware Trojan detection at gate-level netlists, and extract feature values best applied to the machine-learning-based hardware-Trojan detection.

In this chapter, we propose three applications of machine-learning-based hardware-Trojan detection. Based on Charpter 2, we further discuss how to enhance the performance and how to utilize their results.

First, we propose a hardware-Trojan detection method based on multi-layer neural networks for gate-level netlists through experimental evaluations. Our proposed method classifies the nets in a netlist into a set of Trojan nets and that of normal nets. First,

[1]Technical contents in this chapter have been presented in the publications ⟨1⟩, ⟨10⟩, ⟨12⟩, ⟨13⟩, and ⟨15⟩.

we extract several Trojan-net feature values from each net in a netlist. Here we use the 11 Trojan-net features proposed in Chapter 2. Next, we construct a set of training data which includes the 11 Trojan-net feature values and the label which denotes whether the net is a Trojan one or a normal one for each net. After that, we learn the set of training data using *multi-layer neural networks*. Finally, we classify the unknown nets into a set of Trojan ones and that of normal ones using the learned multi-layer neural networks.

Second, we propose a Trojan-invalidating circuit that prevents a hardware Trojan circuit from working. The implementation evaluation on an FPGA board demonstrates that the implemented Trojan-invalidating successfully disable a hardware Trojan circuit inserted into a cryptographic circuit.

Third, we discuss on refinement of classification performance and invalidation of the detected hardware Trojan circuit. Since existing machine-learning-based hardware-Trojan detection methods take care of the features extracted from the nets in a netlist, they does not consider the nearby nets from a net. Therefore, even if the nearby nets are all Trojan nets, a machine learning algorithm may misclassify a Trojan net as a normal net whose feature is like a normal net. In particular, the boundary nets between Trojan nets and normal nets is likely to be misclassified. This method considers the boundary net structures, and correct the misclassified results.

The contributions of this chapter are summarized as follows:

1. We propose an effective multi-layer neural networks for hardware-Trojan detection. By experimentally optimizing the structure of multi-layer neural networks, we can obtain an average of 84.8% true positive rate and an average of 70.1% true negative rate while we can obtain 100% true positive rate in some of the benchmarks, which outperforms the existing methods in most of the cases.

2. We propose a Trojan-invalidating circuit that can be inserted at the nets based on the results of a machine-learning-based hardware-Trojan detection method;

3. Based on a machine-learning-based hardware-Trojan detection results, we further improve the results by considering the features of nearby nets.

3.2 Hardware Trojan Classification Utilizing Multi-Layer Neural Networks

3.2.1 Related Works

In this section, we show the related works of machine-learning-based hardware-Trojan detection and clarify their problems.

Machine-Learning-Based Hardware-Trojan Detection Method Using Support Vector Machines

In [4], a machine-learning-based hardware-Trojan detection method using support vector machines (SVMs) is proposed. The *five* Trojan-net features regarding the primary input/output and fan-in counts are used for hardware-Trojan detection, and they are firstly extracted from nets in a netlist based on the several known hardware-Trojan infected netlists. Then the five feature values are considered to be a five-dimensional vector and it is learned by an SVM classifier. Finally, an unknown netlist is classified into a Trojan one and a normal one.

This method has obtained at most 100% true positive rate (TPR), which is the ratio of detected Trojan nets divided by the number of true Trojan nets. However, the true negative rate (TNR), which is the ratio of detected normal nets divided by the number of true normal nets, is not always high enough in this method.

Trojan-Feature Extraction for Machine-Learning-Based Hardware-Trojan Detection

In [5], the 51 Trojan-net features are firstly extracted from known netlists published at Trust-HUB [1]. After that, the 11 effective Trojan-net features are extracted for hardware-Trojan detection using random forest, which is one of the supervised machine-learning algorithms. Table 3.1 shows the extracted 11 Trojan-net features in [5]. Figure 3.1 depicts the examples of Trojan-net features for a small circuit. Now we show several examples for the Trojan-net feature values using the small circuit in Figure 3.1. First, we extract the feature #1 'fan_in_4' in Table 3.1 which shows the number of logic-gate fanins 4-level away from the net n. In Figure 3.1, the number of logic-gate fanins 4-level away from the net n, which is shown in the dotted rectangle, becomes 6, and therefore we obtain fan_in_4 = 6. Similarly, we can obtain the feature #2 'fan_in_5'. As a next example, we extract the feature #8 'in_nearest_pin' in Table 3.1 which shows the minimum level to the primary input from the net n. In Figure 3.1, there are two primary inputs "Primary input A" and "Primary input B" in the input side of the net n. "Primary input B" is the nearest from the net n and this is 2-level away from the net n, and therefore we obtain in_nearest_pin = 2. We extract the 11 Trojan-net feature values in Table 3.1 from the structure of nets around each net n in the same way as above. See [5] in detail.

This method has obtained 100% TPR in several benchmarks and their TNRs are quite good in most benchmarks. However, the average TPR value becomes smaller than that obtained by [4].

Table 3.1: The best set of 11 Trojan features and their descriptions extracted in [5].

#	Feature	Description
1	fan_in_4	The number of logic-gate fanins 4-level away from the net *n*.
2	fan_in_5	The number of logic-gate fanins 5-level away from the net *n*.
3	in_flipflop_4	The number of flip-flops up to 4-level away from the input side of the net *n*.
4	out_flipflop_3	The number of flip-flops up to 3-level away from the output side of the net *n*.
5	out_flipflop_4	The number of flip-flops up to 4-level away from the output side of the net *n*.
6	in_loop_4	The number of up to 4-level loops from the input side of the net *n*.
7	out_loop_5	The number of up to 5-level loops from the output side of the net *n*.
8	in_nearest_pin	The minimum level to the primary input from the net *n*.
9	out_nearest_pout	The minimum level to the primary output from the net *n*.
10	out_nearest_flipflop	The minimum level to any flip-flop from the output side of the net *n*.
11	out_nearest_multiplexer	The minimum level to any multiplexer from the output side of the net *n*.

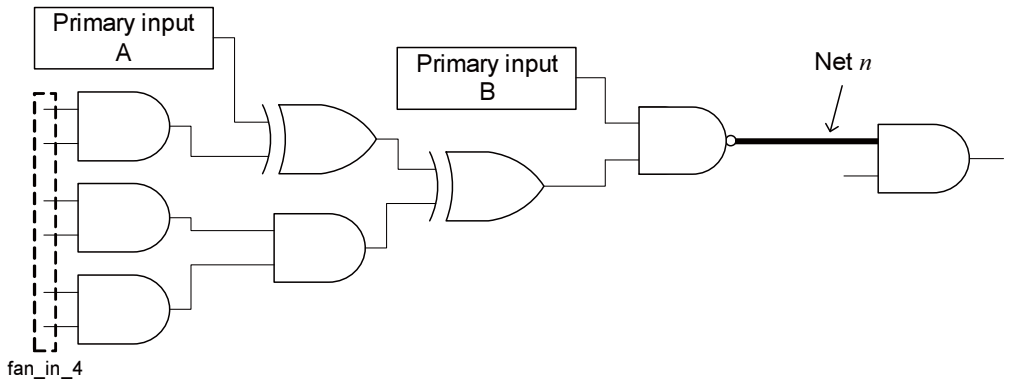


Figure 3.1: The examples of Trojan-net features.

Points of Our Proposed Method

In machine learning, there is a trade-off between TPR and TNR and how to maximize both of TPR and TNR is a serious concern. In order to realize the goal 2) discussed in Section 3.1, we focus on the following points:

1. Multi-layer neural networks: A *multi-layer neural network* is one of the supervised machine-learning algorithms, which is very effectively used in image recognition [39] and natural language processing [40]. We utilize multi-layer neural networks for hardware-Trojan detection;

2. 11 Trojan-net features: In [5], the 11 Trojan-net features are proposed which are expected to effectively classify the nets into Trojan ones and normal ones using machine learning. We expect that we can effectively use them in multi-layer neural networks;

3. Evaluation criteria: We consider both the TPR and TNR as the criteria to evaluate classification results.

3.2.2 Algorithm for Hardware Trojan Classification Utilizing Multi-Layer Neural Networks

We propose a hardware-Trojan detection method using multi-layer neural networks in this section. In this chapter, "multi-layer neural networks" refer to the neural networks which have one or more middle layers.

The proposed method works as follows:

In the learning phase, (Step L1) we firstly extract the 11 Trojan-net feature values in Table 3.1 from every net in a known netlist. Note that we also know whether each net is Trojan or not in the learning phase; (Step L2) After that, we learn the extracted 11 features using multi-layer neural networks and construct a learned multi-layer neural network.

In the classification phase, (Step C1) we firstly extract the 11 Trojan-net feature values in Table 3.1 from every net in an unknown netlist; (Step C2) After that, we classify every net into a Trojan net or a normal net using a learned multi-layer neural network.

In the proposed method, how to design a multi-layer neural network structure is a major concern and then we propose it hereafter.

Structures of Neural Networks

For multi-layer neural networks, how to decide their parameters is a serious concern. There are several parameters in multi-layer neural networks such as the number of middle layers and the number of units in each layer. The classification results depend on the structure of the networks and thus we have to decide these parameters carefully.

In this subsection, we propose the structure of 1) the input and output layer and 2) the middle layers.

Structure of the input and output layer The number of units in the input layer and that of the output layer are dependent on the structure of input data and the type of output, respectively.

The number of units in the input layer is determined by the number of features in input data. In this chapter, we use the 11 Trojan-net features and thus we use 11 units for it. To learn or classify the input data, we input each feature to the corresponding unit in the input layer.

The number of units in the output layer is one or two for the binary classification. When we use one unit in the output layer, we set a threshold value to the output value. For the binary classification, the output value ranges 0.0 to 1.0 and 0.5 is frequently used as a threshold value [33]. For example, we set the threshold value to 0.5, and then if the output value is larger than 0.5, we classify the net to be a Trojan net. Otherwise

we classify the net to be a normal net. When we use two units in the output layer, we compare the output values of the two units. One unit corresponds to the normal net, and the other corresponds to the Trojan net. When the output value of the first unit is larger than that of the second one, we identify the net to be a normal net. Otherwise, we identify the net to be a Trojan net.

When we use one output unit, we can obtain only one output value and how to set its threshold value becomes another serious concern. On the other hand, when we use two output units, we require no threshold value but we just compare the two output values relatively. Based on this discussion, we use two output units in our proposed method.

In summary, Figure 3.2 shows the basic structure of the proposed neural networks. The input layer has 11 units, and the output layer has two units. The structure of middle layers is experimentally determined as described later.

The number of middle layers and the number of units Generally, it is known that the number of middle layers and the number of units in each middle layer affect the classification results. It is also known that we can flexibly learn datasets when the number of middle layers and the number of units in each middle layer are large. On the other hand, a vanishing gradient problem [45] may occur when we use a large number of middle layers. However, the structure of middle layers cannot be determined theoretically [46].

In this chapter, we try the various numbers of middle layers and the various numbers of units in each layer experimentally as follows and optimize the structure of middle layers based on these results (the optimization results are shown in Section 3.2.3):

1. We set the number of middle layers to one and we set the number of the units in the layer to 10, 12, 15, 20, 50, 100, 200, or 500.

2. We set the number of middle layers to two. For the first layer, we set the number of the units to 20, 50, 100, or 200. For the second layer, we set the number of the units to 10, 12, 15, 20, 50, 100, 200, or 500.

3. We set the number of middle layers to three. For the first and third layer, we set the number of the units to 50, 100, or 200. For the second layer, we set the number of the units to 20, 50, 100, or 200.

4. We set the number of middle layers to four. For each layer, we set the number of the units to 50, 100, or 200.

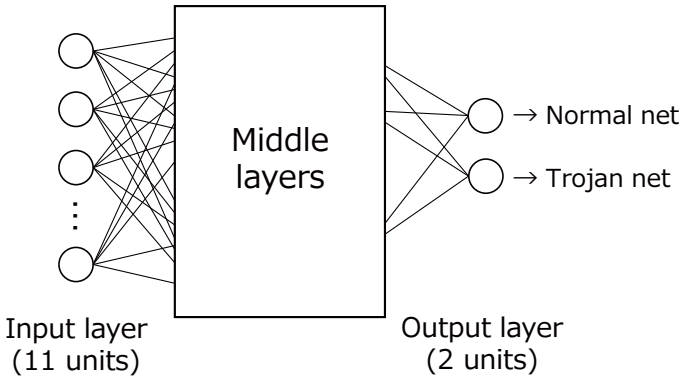


Figure 3.2: The structure of multi-layer neural networks.

Weighting using a loss function The number of Trojan nets is much smaller than that of normal nets as summarized in Table 3.2. As shown in Table 3.2, RS232-T1000 has 283 normal nets and 36 Trojan nets. Generally, a training dataset should be balanced for machine learning.

In our proposed method, we apply weighting to the loss function used in neural networks. We apply a weighted-cross-entropy loss function to neural networks [47]. According to [47], the weighted-cross-entropy loss function E is expressed as $E = -\sum_{k=1}^{2} w_k \cdot t_k \ln y_k$, where t_k is the output value of k-th unit in neural networks, y_k is the answer value, and w_k is the weight value.

In our proposed method, we set the weighting vector $\boldsymbol{w} = (w_1, w_2)$ to the ratio of the number of normal nets to the number of the Trojan nets. For example, the ratio for RS232-T1000 is $1 : 7.86$ and thus we set (w_1, w_2) to $(1, 7.86)$.

Evaluation on the Structure of Multi-Layer Neural Networks

In this subsection, we discuss how to evaluate the classification results of neural networks.

For the evaluations, we use the 17 Trust-HUB benchmarks [1] listed in Table 3.2. In the benchmarks, Trojan nets and normal nets are clearly stated and then we know which nets are Trojan ones and which nets are normal ones beforehand. We apply a leave-one-out cross-validation [36] to the benchmarks in the experiments, where each one of the netlists is considered to be a test set (unknown data set) and the other 16 netlists are considered to be training sets (known data set), and obtain classification results.

In the machine learning, there are several evaluation indices. In case of binary classification, there are four values to evaluate the classification results: the true negative value (TN); the false positive value (FP); the true positive value (TP); and the false negative value (FN). TN shows the number of normal nets identified to be normal nets;

Table 3.2: The Trust-HUB benchmarks [1] used in the experiments.

Data name	# of normal nets	# of Trojan nets
RS232-T1000	283	36
RS232-T1100	284	36
RS232-T1200	289	34
RS232-T1300	287	29
RS232-T1400	273	45
RS232-T1500	283	39
RS232-T1600	292	29
s15850-T100	2,429	27
s35932-T100	6,407	15
s35932-T200	6,405	12
s35932-T300	6,405	37
s38417-T100	5,798	12
s38417-T200	5,798	15
s38417-T300	5,801	44
s38584-T100	7,343	19
s38584-T200	7,373	97
s38584-T300	7,614	874

FP shows the number of normal nets identified to be Trojan nets mistakenly; TP shows the number of Trojan nets identified to be Trojan nets; FN shows the number of Trojan nets identified to be normal nets mistakenly. Based on the values of TN, FP, TP and FN, we can define the true positive rate (TPR) and the true negative rate (TNR), where TPR is defined by TP/(TP+FN) and TNR is defined by TN/(TN+FP).

In order to realize the goal 2) in Section 3.1, maximizing TPR must be the first priority. However, only maximizing TPR may be insufficient. For example, if we identify all the nets in a netlist to be Trojan nets, the TPR becomes 100%, although they definitely include many normal nets. Thus we also focus on maximizing TNR next.

3.2.3 Experimental Results

In this section, we show the optimized neural network structure through experimental evaluations and its comparison to existing methods. In the experiments, we use 17 Trust-hub [1] benchmarks. Table 3.2 shows the benchmarks used as training data and test data in the experiments. We applied leave-one-out cross-validation where one benchmark is used as a test data and the other benchmarks are used as training data to the experiments.

We use an Intel Xeon E5-2695 computer environment with a 256GB memory. The program is written in Python3 using Chainer [48] as a library for machine learning.

Classification Results

Experiment 1: One middle layer Table 3.3 shows the results when we used one-middle-layer neural networks. In Table 3.3, we show the average TPR and TNR values where the numbers of units listed in the left column are applied to the neural network. The underlined parts are the maximum TPR and TNR values.

We obtained the maximum TPR when we applied 20 units, and the maximum TNR when we applied 50 units. When the number of units in the first middle layer is 20, 200, and 500, TPR becomes more than 86%. When the number of units in the first middle layer is 50, 100, and 200, TNR becomes more than 60%. When the number of units is 500, neither the TPR nor the TNR is maximized. Just increasing the number of units does not always lead to improving the classification results.

Experiment 2: Two middle layers Table 3.4 shows the results when we used two-middle-layer neural networks. We set the number of the units in the second middle layer to 10, 12, 15, 20, 50, 100, 200, and 500. In Table 3.4, we show the average TPR and TNR values where the numbers of units listed in the columns of "# of units" are applied to the neural network. The underlined parts are the maximum TPR and TNR values.

In this experiment, we obtained the maximum average TNR when we used the 200-50 neural network, where we use 200 units for the first middle layer and 50 units for the second middle layer. From the viewpoint of average TPRs, they are decreased compared to single-middle-layer neural networks. However, the neural networks whose first middle layer has 200 units obtained 63.1% of average TNR and this result is better than the average TNR of single-middle-layer neural networks.

Experiment 3: Three middle layers Table 3.5 shows the results when we used three-middle-layer neural networks. We set the number of the units in the first middle layer to 50, 100, and 200, the number of the units in the second one to 20, 50, 100, and 200, and the number of the third one to 50, 100, and 200. In Table 3.5, we show the average TPR and TNR values where the numbers of units listed in the columns of "# of units" are applied to the neural network. The underlined parts are the maximum TPR and TNR values.

In this experiment, we obtained the maximum average TPR when we used the 100-20-50 neural network where we use 100, 20, and 50 units in the first, second, and third middle layer, respectively, and the maximum average TNR when we used the 200-200-100 neural network. The maximum average TPR value becomes 86.2%, where its average TNR

Table 3.3: Experimental results (One middle layer).

# of units	Average values		# of units	Average values	
1st layer	TPR	TNR	1st layer	TPR	TNR
10	85.9%	59.7%	50	84.2%	62.9%
12	85.7%	58.5%	100	83.4%	60.1%
15	82.7%	59.5%	200	86.8%	62.3%
20	87.4%	59.4%	500	86.7%	53.3%

Table 3.4: Experimental results (Two middle layers).

# of units		Average values		# of units		Average values	
1st layer	2nd layer	TPR	TNR	1st layer	2nd layer	TPR	TNR
20	10	87.4%	58.4%	100	10	79.1%	61.4%
	12	83.5%	61.2%		12	85.8%	57.3%
	15	84.7%	56.4%		15	82.1%	61.4%
	20	88.0%	59.6%		20	87.7%	58.6%
	50	83.7%	59.9%		50	81.9%	61.1%
	100	83.9%	57.5%		100	83.7%	59.8%
	200	82.5%	65.7%		200	79.1%	66.6%
	500	77.9%	59.9%		500	85.4%	60.6%
50	10	84.1%	59.5%	200	10	85.7%	61.8%
	12	83.3%	57.2%		12	82.6%	58.6%
	15	85.0%	62.0%		15	77.1%	62.4%
	20	83.9%	59.9%		20	79.3%	66.7%
	50	86.0%	60.2%		50	84.7%	66.8%
	100	88.9%	59.2%		100	84.8%	60.2%
	200	84.2%	57.6%		200	78.2%	62.3%
	500	82.2%	58.6%		500	83.6%	66.0%

value becomes 60.4% but this value seems to be small. As in the prior hardware-Trojan classification method using single-middle-layer neural networks [49], the average TNR becomes 68.7%. Then, from the viewpoint of average TNR in Table 3.5, more than 70% of average TNR value is expected. Consequently, the 200-100-50 neural network seems to be the most effective structure for hardware-Trojan classification, where average TPR becomes 84.8% and TNR becomes 70.1% (the wave-underlined ones in Table 3.5).

Table 3.5: Experimental results (Three middle layers).

# of units			Average values		# of units			Average values		# of units			Average values	
1st layer	2nd layer	3rd layer	TPR	TNR	1st layer	2nd layer	3rd layer	TPR	TNR	1st layer	2nd layer	3rd layer	TPR	TNR
50	20	50	82.2%	58.4%	100	20	50	86.2%	60.4%	200	20	50	82.7%	65.3%
		100	81.9%	60.4%			100	85.2%	64.4%			100	80.7%	68.8%
		200	83.7%	59.9%			200	79.6%	63.4%			200	80.5%	67.9%
	50	50	78.2%	61.9%		50	50	83.2%	59.9%		50	50	80.6%	75.4%
		100	81.5%	55.7%			100	81.8%	63.3%			100	74.3%	61.0%
		200	82.9%	60.1%			200	78.1%	61.8%			200	83.8%	64.8%
	100	50	77.5%	67.1%		100	50	79.7%	60.0%		100	50	84.8%	70.1%
		100	80.6%	60.8%			100	77.3%	59.7%			100	76.6%	70.0%
		200	75.6%	61.4%			200	78.9%	67.2%			200	80.2%	69.4%
	200	50	79.2%	64.9%		200	50	79.1%	60.8%		200	50	77.3%	70.3%
		100	76.8%	65.7%			100	83.9%	66.7%			100	81.0%	75.7%
		200	77.8%	67.0%			200	78.4%	63.5%			200	79.9%	60.5%

Experiment 4: Four middle layers Table 3.6 shows the results when we used four-middle-layer neural networks. We set the number of the units in each middle layer to 50, 100, and 200. In Table 3.6, we show the average TPR and TNR values where the numbers of units listed in the columns of "# of units" are applied to the neural network. The underlined parts are the maximum TPR and TNR values.

In this experiment, we obtained the maximum average TPR when we used the 100-200-200-200 neural network where we use 100, 200, 200, and 200 units in the first, second, third, and fourth middle layer, respectively, and the maximum average TNR when we used the 200-50-100-200 neural network.

Table 3.6: Experimental results (Four middle layers).

# of units in 1st layer = 50					# of units in 1st layer = 100					# of units in 1st layer = 200				
# of units			Average values		# of units			Average values		# of units			Average values	
2nd layer	3rd layer	4th layer	TPR	TNR	2nd layer	3rd layer	4th layer	TPR	TNR	2nd layer	3rd layer	4th layer	TPR	TNR
50	50	50	77.9%	66.3%	50	50	50	84.6%	63.8%	50	50	50	78.6%	68.0%
		100	74.3%	68.1%			100	81.2%	62.3%			100	80.9%	71.8%
		200	81.2%	59.2%			200	76.2%	61.9%			200	75.7%	71.6%
	100	50	77.9%	61.0%		100	50	82.7%	66.4%		100	50	83.6%	71.4%
		100	84.3%	61.3%			100	83.2%	64.6%			100	82.0%	69.4%
		200	78.2%	58.7%			200	81.2%	61.9%			200	75.5%	79.2%
	200	50	78.0%	59.3%		200	50	82.2%	61.2%		200	50	82.9%	62.6%
		100	81.4%	65.5%			100	73.3%	74.2%			100	77.0%	66.4%
		200	80.4%	64.1%			200	83.7%	64.2%			200	81.4%	65.5%
100	50	50	76.6%	65.5%	100	50	50	85.4%	64.7%	100	50	50	69.2%	70.6%
		100	76.4%	60.5%			100	82.2%	64.0%			100	81.5%	66.7%
		200	76.5%	63.3%			200	81.7%	69.4%			200	83.5%	66.9%
	100	50	76.6%	60.6%		100	50	78.4%	65.0%		100	50	71.4%	68.3%
		100	82.6%	68.1%			100	80.6%	65.1%			100	76.0%	71.9%
		200	81.3%	54.6%			200	76.7%	64.1%			200	71.9%	68.7%
	200	50	77.0%	64.9%		200	50	80.1%	67.8%		200	50	76.2%	65.8%
		100	75.4%	62.9%			100	80.9%	63.1%			100	74.1%	72.0%
		200	80.0%	62.9%			200	83.7%	64.8%			200	74.9%	79.2%
200	50	50	76.1%	65.1%	200	50	50	79.9%	54.1%	200	50	50	79.0%	77.1%
		100	75.0%	73.2%			100	82.0%	60.1%			100	81.3%	78.5%
		200	84.4%	57.3%			200	84.0%	63.7%			200	81.8%	63.0%
	100	50	81.3%	68.0%		100	50	81.0%	68.1%		100	50	73.8%	71.9%
		100	77.2%	74.6%			100	79.6%	68.0%			100	66.7%	78.5%
		200	79.5%	64.1%			200	78.0%	60.0%			200	78.4%	72.7%
	200	50	74.4%	71.4%		200	50	69.5%	76.2%		200	50	71.1%	78.5%
		100	82.9%	64.7%			100	79.5%	66.4%			100	74.1%	75.6%
		200	79.6%	67.4%			200	86.1%	67.3%			200	84.4%	71.3%

Table 3.7: Total average TPR and TNR values of the experimental results.

# of middle layers	Total average values	
	TPR	TNR
1	85.7%	59.6%
2	83.6%	61.2%
3	80.3%	64.3%
4	78.9%	66.8%

Overview of the experimental results Table 3.7 shows the total average TPR and TNR values for Experiments 1–4 above. As shown in Table 3.7, the one-middle-layer neural networks obtained the largest total average TPR value in the experiments, where the total average TNR value is the smallest. On the other hand, the four-middle-layer neural networks obtained the largest total average TNR value in the experiments. These results suggest that the neural networks with many layers and units do not always improve the classification results in terms of TPR. Considering the balance between TPR and TNR, three-or-less-layer neural networks are appropriate for hardware-Trojan classification.

In all the average TPR and TNR values obtained in the experiments, their third quartiles become 83.5% and 68.0%, respectively. Only the three neural networks, the 200-100-50, the 200-50-100-50, and the 200-200-200-200 neural networks, obtained larger than 83.5% average TPR value and 68.0% average TNR value. Among the three neural networks, the 200-100-50 neural network obtained the largest average TPR value, and therefore, as mentioned in Experiment 3, the 200-100-50 neural network seems to be the most effective structure for hardware-Trojan classification.

Comparison to Existing Methods

We compare the results to existing methods. For the comparison, we use the 200-100-50 neural network, which is one of the best results in the experiments above. In [4], a machine-learning-based hardware-Trojan classification method using an SVM is proposed. In [2], a clustering-based hardware-Trojan classification method using signal correlations is proposed, which is one of the most state-of-the-art methods proposed recently. We have compared our experimental results to these existing methods from the viewpoints of TPR and TNR.

Table 3.8 shows the comparison to [4] and Table 3.9 shows the comparison to [2]. From the viewpoint of average TPR, our method outperforms both of existing methods. Though [4] is also one of the hardware-Trojan detection methods using supervised machine learning, the average results using neural networks are quite good in terms of TPR and TNR. In the comparison to [2], our results have a better TPR value in the two

Table 3.8: Comparison to the existing method [4].

	TPR		TNR	
Test data	[4]	Ours	[4]	Ours
RS232-T1000	53%	100%	31%	24%
RS232-T1100	58%	78%	27%	25%
RS232-T1200	80%	91%	26%	55%
RS232-T1300	89%	86%	26%	65%
RS232-T1400	83%	100%	22%	15%
RS232-T1500	83%	82%	24%	47%
RS232-T1600	89%	97%	26%	28%
s15850-T100	93%	81%	66%	96%
s35932-T100	93%	80%	60%	99%
s35932-T200	100%	67%	59%	88%
s35932-T300	27%	100%	58%	97%
s38417-T100	100%	83%	76%	98%
s38417-T200	73%	93%	76%	74%
s38417-T300	100%	100%	72%	94%
s38584-T100	100%	16%	62%	99%
s38584-T200	94%	91%	64%	93%
s38584-T300	89%	97%	66%	93%
Average	83%	85%	49%	70%

benchmarks out of four benchmarks outperform from and our average TPR value is better than [2], though average TNR is not improved compared to [2].

Discussions on Experimental Results

The relationship between the number of middle layers and the results In this chapter, we examined our method using one, two, three, and four middle layers for neural networks. Since there is no limit for the number of middle layers, we can set more than four middle layers for neural networks. However, four or more middle layers are not suitable for the hardware-Trojan detection. As shown in Table 3.7, the average TPR values tend to be decreased when the number of middle layers is increased. On the other hand, the average TNR values are increased when the number of middle layers is increased. Though increasing TNR values improves the accuracy of neural networks, decreasing TPR too much is not appropriate for hardware-Trojan detection because we have to avoid mistakenly identifying Trojan nets to be normal nets.

Table 3.9: Comparison to the existing method [2].

	TPR		TNR	
Test data	[2]	Ours	[2]	Ours
s15850-T100	61%	81%	99%	96%
s35932-T200	27%	67%	99%	88%
s38417-T100	100%	83%	99%	98%
s38584-T200	99%	91%	98%	93%
Average	72%	81%	99%	94%

Comparison results In Section 3.2.3, we compared our results to [2]. As shown in Table 3.9, our method outperforms [2] in two benchmarks from the viewpoint of TPR and our average TPR value is better than [2]. To realize the Goal 2 mentioned in Section 2.1, maximizing TPR is the most important and we believe that it is meaningful to outperform an existing method from the viewpoint of TPR.

Although our TPR values of two benchmarks are better than those of [2], our TPR values of the other two benchmarks and the TNR values should be further improved. This is an important future work.

3.3 Refinement of Classification Results Based on Boundary Net Structures

3.3.1 Related Works

Based on Trojan features, several machine-learning-based hardware-Trojan detection methods have been proposed [5, 50]. These methods can identify whether the net in a netlist is Trojan net or normal net. In [50], the five Trojan features are extracted from each net in a gate-level netlist. After that we learn the extracted features using a classifier such as a support vector machine (SVM) or a neural network (NN). Then, we extract the features from an unknown netlist and classify them using the learned classifier. The experimental results show that the classifier identify most of the Trojan nets to be Trojan nets correctly. In [5], the 51 Trojan features are extracted from each net in a gate-level netlist at first. After that the eleven Trojan features are selected using a random forest classifier with maximizing F-measures. Table 3.10 shows the results of the previous works [5, 50] in terms of the average true positive rate (TPR)[2], the average true negative

[2]Trojan nets identified to be Trojan nets correctly is called true positives. TP shows the number of true positives. Normal nets identified to be Trojan nets mistakenly is called false negatives. FN shows

Table 3.10: The classification results in the previous works using machine-learning-based approaches.

Approach	Ave. TPR	Ave. TNR	Ave. Accuracy
SVM [50]	83%	49%	51%
NN [50]	81%	69%	70%
Random forest [5]	68%	99.7%	99%

rate (TNR), and the average accuracy. In Table 3.10, we use the leave-one-out evaluation to obtain each of the results. In Table 3.10, the SVM-based approach in [50] obtained the maximum average TPR in the three approaches listed. On the other hand, the average TNR becomes less than 50% and the average accuracy becomes only 51% in the SVM-based approach. In the NN-based approach, the average TPR is the second largest in the three approaches, but the average TNR becomes 69%. The average accuracy of the NN-based approach becomes 70%. Finally in the random-forest-based approach, though the average TPR is low, the average TNR becomes almost 100% and the average accuracy becomes 99%.

In hardware-Trojan detection, it is important to identify Trojan nets to be Trojan nets correctly. We have to avoid identifying Trojan nets to be normal nets mistakenly (they are called false negatives. FN shows the number of false negatives). Therefore, it is important for us to minimize FN and maximize TPR at first, and secondly to maximize TNR. Based on the priority discussed above, the SVM-based approach is the best for hardware-Trojan detection in Table 3.10. However, it is impractical because the average accuracy is only 51%, which means that about half of the nets are identified to be Trojan nets or normal nets mistakenly. If we try to refine the classification results, we have to re-investigate many nets identified to be Trojan nets because the set of Trojan nets includes many *truely* normal nets. In order to increase the TPR and TNR, it is efficient to re-investigate the results of the random-forest-based approach because the average accuracy becomes 99%, which means that almost all of the nets are classified into a set of Trojan nets and that of normal nets correctly. In this chapter, we focuses on the results obtained from the random-forest-based approach proposed in [5], and try to refine the classification results.

the number of false negatives. TN and FP are defined similarly. Based on the definitions above, the true positive rate (TPR) is defined by TP/(TP + FN), and the true negative rate (TNR) is defined by TN/(TN + FP).

Table 3.11: The classification results using the random-forest-based hardware-Trojan detection method in [5].

Benchmark	TN	FP	FN	TP	TPR	TNR	Accuracy
RS232-T1000	280	3	0	36	100.0%	98.9%	99.1%
RS232-T1100	279	5	18	18	50.0%	98.2%	92.8%
RS232-T1200	289	0	4	30	88.2%	100.0%	98.8%
RS232-T1300	287	0	0	29	100.0%	100.0%	100.0%
RS232-T1400	273	0	1	44	97.8%	100.0%	99.7%
RS232-T1500	282	1	2	37	94.9%	99.6%	99.1%
RS232-T1600	289	3	2	27	93.1%	99.0%	98.4%
s15850-T100	2,418	1	6	21	77.8%	100.0%	99.7%
s35932-T100	6,407	0	4	11	73.3%	100.0%	99.9%
s35932-T200	6,405	0	11	1	8.3%	100.0%	99.8%
s35932-T300	6,404	1	7	30	81.1%	100.0%	99.9%
s38417-T100	5,798	0	8	4	33.3%	100.0%	99.9%
s38417-T200	5,798	0	8	7	46.7%	100.0%	99.9%
s38417-T300	5,801	0	11	33	75.0%	100.0%	99.8%
s38584-T100	7,341	2	18	1	5.3%	100.0%	99.7%

The hardware-Trojan detection based on the random forest classifier and its limitation

Based on the discussion above, the random-forest-based approach is applicable for hardware-Trojan detection. Table 3.11 demonstrates the classification results based on the random-forest-based approach [5]. In Table 3.11, both of FP and FN in all of the benchmarks are less than 20, and this number is much smaller than the number of normal nets (TN + TP). Especially in RS232-T1300, since both FP and FN are 0, the classifier completely classify all of the nets in the netlist into a set of normal nets and Trojan nets. On the other hand, in RS232-T1200, several Trojan nets are mistakenly identified to be normal nets while all of the normal nets are identified to be normal nets correctly.

As discussed above, we have to maximize TPR at first, and secondly we have to maximize TNR as high as possible. In this chapter, we focus on RS232 series benchmarks (RS232-T1000–RS322-T1600) to carefully investigate the netlists. To begin with, we investigate RS232-T1100, RS232-T1200, RS232-T1400, RS232-T1500, and RS232-T1600 benchmarks since their FNs are one or more. After that we investigate RS232-T1000, RS232-T1100, RS232-T1500, and RS232-T1600 since their FPs are one or more.

The limitation of feature-based machine-learning method

In the machine-learning-based hardware-Trojan detection methods such as [50, 5], the learning procedure is shown as below:

1. We extract several Trojan feature values from each net in a given netlist, and we make a Trojan feature vector for each net.
2. We give a label, which shows whether the net is a Trojan net or a normal net, to its Trojan feature vector.
3. We learn the Trojan feature vectors and their labels using a machine learning algorithm.

When we learn the Trojan feature vectors of nets, we do not know where the net is placed or which nets are connected with. However, since Trojan nets are placed close to each other in most cases, investigating the nearby nets around Trojan-identified nets must be good hints to detect all the Trojan nets and thus increases TPR. In this chapter, by considering the boundary nets of Trojan-identified nets, we increase the TPRs of the machine-learning-based hardware-Trojan detection method.

3.3.2 Analysis of Mistakenly-Identified Nets

In this section, we analyze the classification results obtained in [5], and extract the features of mistakenly-identified nets and their nearby nets. We have focused on the RS232 series benchmarks which can be obtained in [1], listed in the top seven benchmarks of Table 3.11. To begin with, we investigate the Trojan nets identified to be normal nets mistakenly (we define the net as "FN net"). After that, we investigate the normal nets identified to be Trojan nets mistakenly (we define the net as "FP net").

The FN nets and their nearby nets

Table 3.12 shows the details of the 27 FN nets in [5], and the number of Trojan nets and normal nets connected to/from up to 2-level away from the net. In Table 3.12, "Trojan net" means the net identified to be Trojan net by the classifier, and "normal net" means the net identified to be normal net by the classifier. As shown in Table 3.12, most of the nets have no Trojan nets 2-level away to them. This means that their nets are placed near the boundary between Trojan nets and normal nets.

Table 3.12: The FN nets and their profiles.

Benchmark	Name	Two-level away (Input)		One-level away (Input)		One-level away (Output)		Two-level away (Output)	
		Trojan	Normal	Trojan	Normal	Trojan	Normal	Trojan	Normal
RS232-T1100	iRECEIVER_bitCell_CTRL	0	10	0	4	1	0	1	0
RS232-T1100	iRECEIVER_N22	0	4	0	2	1	2	1	6
RS232-T1100	iRECEIVER_N20	0	7	0	2	1	2	1	7
RS232-T1100	iRECEIVER_N21	0	4	0	2	1	2	1	7
RS232-T1100	iRECEIVER_N27	0	7	0	2	1	1	1	1
RS232-T1100	iRECEIVER_N28	0	8	0	2	1	1	1	1
RS232-T1100	iRECEIVER_N19	1	7	0	2	2	0	2	2
RS232-T1100	iRECEIVER_N18	0	7	0	2	0	2	2	2
RS232-T1100	iRECEIVER_N17	0	8	0	2	0	2	2	2
RS232-T1100	iRECEIVER_state_2_	1	5	2	4	2	2	2	6
RS232-T1100	iRECEIVER_bitCell_cntrH_1_	0	6	0	6	0	6	1	8
RS232-T1100	iRECEIVER_N26	0	8	0	2	1	1	1	1
RS232-T1100	iXMIT_N27	2	3	0	2	2	0	1	6
RS232-T1100	iXMIT_N26	0	8	0	2	2	1	2	4
RS232-T1100	iXMIT_N25	0	8	0	2	1	1	1	1
RS232-T1100	iXMIT_N24	2	5	0	2	1	1	2	3
RS232-T1100	iXMIT_N28	0	5	0	2	1	3	2	4
RS232-T1100	iRECEIVER_bitCell_cntrH_0_	0	3	0	6	0	5	1	7
RS232-T1200	iXMIT_next_state_2_	0	4	0	1	2	1	1	12
RS232-T1200	iRECEIVER_rec_datH	0	7	1	5	1	4	1	6
RS232-T1200	iXMIT_state_1_	0	5	0	6	2	3	1	12
RS232-T1200	iXMIT_state_0_	0	6	0	6	1	6	2	10
RS232-T1400	iRECEIVER_rec_datSyncH	0	3	0	6	0	4	1	9
RS232-T1500	iXMIT_bitCell_cntrH_1_	2	5	1	5	0	7	3	4
RS232-T1500	xmit_doneH	3	5	2	0	0	0	0	0
RS232-T1600	iRECEIVER_bitCell_CTRL	3	7	0	2	0	0	0	0
RS232-T1600	iRECEIVER_N_CTRL_1_	0	0	0	0	1	0	1	0

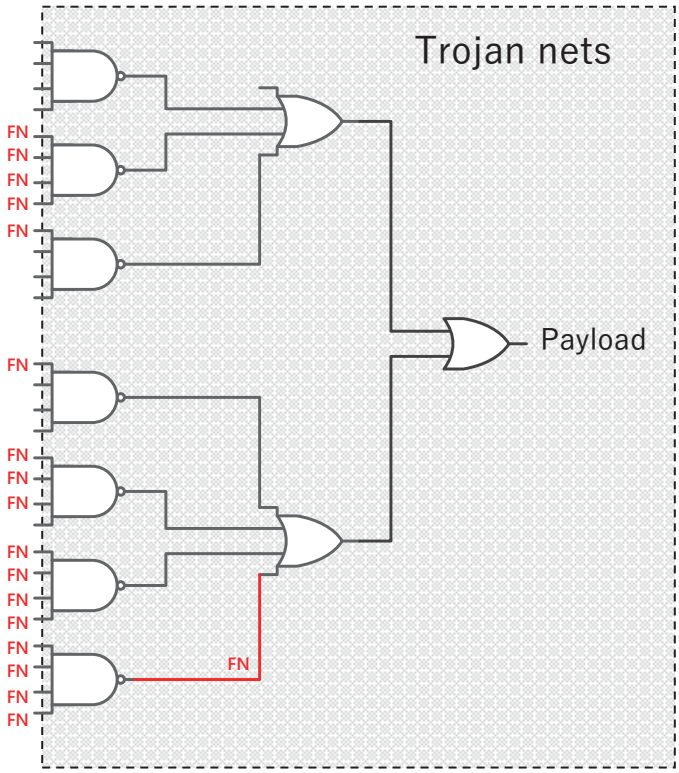


Figure 3.3: The FN nets obtained in RS232-T1100.

In order to simplify the analysis, we pick up the netlists whose FNs are the first and second largest in Table 3.12: RS232-T1100 and RS232-T1200.

Figure 3.3 shows the FN nets in RS232-T1100. The hardware Trojan inserted into RS232-T1100 has a three-logic-level combinational circuit. In the classification result, the classifier could not identify the nets, which are around the boundary between Trojan nets and normal nets, to be Trojan nets. However, their succeeding nets are identified to be Trojan nets correctly. We can identify the FN nets to be Trojan nets correctly if we investigate their succeeding nets.

Figure 3.4 shows the FN nets in RS232-T1200. As shown in Figure 3.4, all the FN nets in RS232-T1200 are placed around the boundary between Trojan nets and normal nets. The three FN nets are connected to the trigger circuit of the hardware-Trojan circuit. The other one FN net is connected to a scan-input signal of the flip-flop in the Trojan nets. The nets connected to/from the flip-flop, which is connected from the net *iRECEIVER_rec_datH*, are identified to be Trojan nets except for the net *iRECEIVER_rec_datH*, and thus we can guess that the net *iRECEIVER_rec_datH* is a Trojan net. The NAND gate connected to the nets *iXMIT_next_state_2_*, *iXMIT_state_1_*, *iXMIT_state_0_* has a fan-out which is identified to be a Trojan net, and it has four fan-ins, and therefore we can guess that the nets *iXMIT_next_state_2_*, *iXMIT_state_1_*,

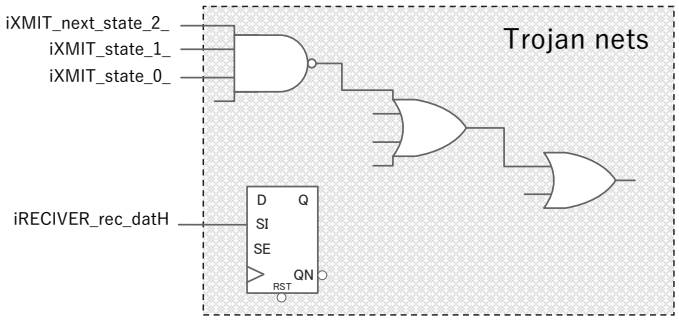


Figure 3.4: The FN nets obtained in RS232-T1200.

*iXMIT_state_0_* are the trigger-condition signals for the hardware Trojan. These features can be found in the net *iRECEIVER_rec_datSyncH* of RS232-T1400 and the net *iXMIT_bitCell_cntrH_1_* of RS232-T1500. The above discussion can be applied to other FN nets in RS232-T1500 and RS232-T1600.

From the discussion above, the FN nets are placed around the boundary between Trojan nets and normal nets.

The FP nets and their nearby nets

Table 3.13 shows the details of the 12 FP nets in [5], and the number of Trojan nets and normal nets connected to/from up to 2-level away from the net. As shown in Table 3.13, each FP nets is connected to several Trojan nets up to 2-level away to/from it.

Table 3.13: The FP nets and their profiles.

Benchmark	Name	Two-level away (Input)		One-level away (Input)		One-level away (Output)		Two-level away (Output)	
		Trojan	Normal	Trojan	Normal	Trojan	Normal	Trojan	Normal
RS232-T1000	xmit_doneH_temp	1	5	1	5	1	1	0	1
RS232-T1000	xmit_doneH	3	5	2	0	0	0	0	0
RS232-T1000	IntConst_U296	0	0	0	0	1	0	1	0
RS232-T1100	iXMIT_bitCell_cntrH_1_	0	7	1	5	0	6	1	6
RS232-T1100	iXMIT_xmit_ShiftRegH_4_	0	7	0	6	1	2	2	4
RS232-T1100	xmit_doneH	1	5	1	5	0	1	0	1
RS232-T1100	IntConst_U296	0	0	0	0	1	0	1	0
RS232-T1100	iXMIT_bitCell_cntrH_0_	2	4	0	6	1	3	0	10
RS232-T1500	IntConst_U296	0	0	0	0	1	0	1	0
RS232-T1600	iXMIT_next_state_2_	1	3	0	1	1	1	0	12
RS232-T1600	iXMIT_state_1_	0	5	1	5	1	3	0	12
RS232-T1600	iXMIT_state_0_	0	6	0	6	1	5	2	9

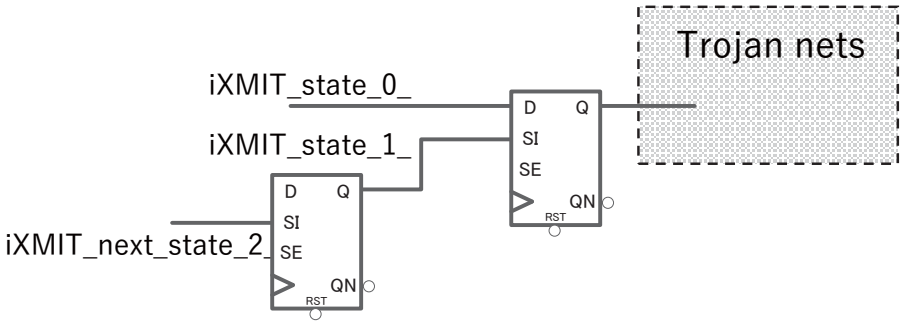


Figure 3.5: The FP nets obtained in RS232-T1600.

Now we focus on RS232-T1600. Figure 3.5 shows the FP nets in RS232-T1600. As shown in Figure 3.5, all the three FP nets in RS232-T1600 are connected to the filp-flops which are placed near the Trojan nets. The net *iXMIT_state_1_* is used as a part of Trojan nets, and thus the classifier identified the net in RS232-T1600 to be Trojan net mistakenly.

3.3.3 Proposed Method and Experimental Results

The boundary net structures

Based on the classification results obtained from the method in [5], we add the following procedures to the machine-learning-based hardware-Trojan detection:

1. **Detect a trigger circuit**
   If the target normal net n is connected to a logic gate g which has at least one Trojan net as a fan-in or fan-out, we identify the normal net n to be a Trojan net. For example, in Figure 3.6, the net n will be identified to be a Trojan net even if n is mistakenly identified to be a normal net by [5].

2. **Detect a flip-flop near the Trojan net**
   If the target normal net n is connected to a flip-flop f, and the flip-flop is also connected only to a logic gate g, which has four or more fan-ins and its fan-out is a Trojan net, we identify the normal net n to be a Trojan net. For example, in Figure 3.7, the net n will be identified to be a Trojan net even if n is mistakenly identified to be a normal net by [5].

The experimental results

Table 3.14 shows the experimental results of our proposed method. The experimental results demonstrate that four benchmarks obtained 100% TPRs. All of the accuracies

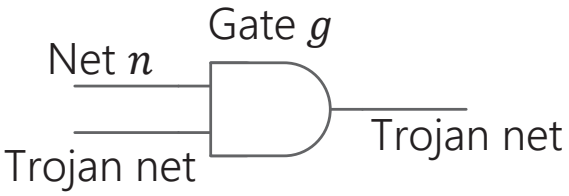


Figure 3.6: An example of a trigger circuit.

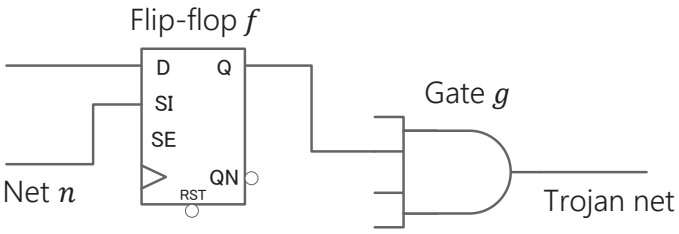


Figure 3.7: An example of a circuit with a flip-flop.

become more than 93%. As discussed in Section 3.3.1, it is important to maximize TPR in hardware-Trojan detection. In this sense, our method increases the TPR, which definitely leads to a good result.

Table 3.15 shows the comparisons between our proposed method and [5]. Our proposed method decrease FNs, and thus the average TPR is increased. These results mean that our proposed method can effectively identify most of the Trojan nets to be Trojan nets correctly. On the other hand, the average TNR is decreased, which means that several benchmarks obtained more FPs in our method than [5]. The reasons why the average TNR is decreased are shown below:

1. The normal nets between Trojan nets and normal nets are identified to be Trojan nets.

2. In several benchmarks, several nets are used as trigger signals of the hardware Trojan, while in the other benchmarks they are used as genuine normal nets. Then, some nets are defined as Trojan nets in a netlist, while the same nets are defined as normal nets in another netlist. This definition causes the confusion in the classification results.

Though several normal nets are identified to be Trojan nets mistakenly, all of the FPs are less than 12. If we carefully investigate the nets identified to be Trojan nets using existing method such as [3] and [24], we can classify the nets more accurately without taking much time. However, combing with other method is impractical. To solve the problems is our future work.

Table 3.14: The experimental results of our method.

Benchmark	TN	FP	FN	TP	TPR	TNR	Accuracy
RS232-T1000	278	5	0	36	100.0%	98.2%	98.4%
RS232-T1100	275	9	11	25	69.4%	96.8%	93.8%
RS232-T1200	277	12	0	34	100.0%	95.8%	96.3%
RS232-T1300	286	1	0	29	100.0%	99.7%	99.7%
RS232-T1400	265	8	0	45	100.0%	97.1%	97.5%
RS232-T1500	276	7	1	38	97.4%	97.5%	97.5%
RS232-T1600	287	5	1	28	96.6%	98.3%	98.1%

Table 3.15: The comparison between our method and [5].

Method	Ave. TPR	Ave. TNR	Ave. Accuracy
[5]	89.1%	99.4%	98.3%
Ours	94.8%	97.6%	97.3%

3.4 Trojan-Net Invalidation Based on Classification Results

3.4.1 Backgrounds

We have discussed on how to detect hardware Trojans at a gate-level netlist. However, even if we can establish a machine-learning-based hardware-Trojan detection scheme, some hardware Trojans may avoid the detection. Typically, detecting sequential-triggered hardware Trojans where the trigger circuits contain sequential circuits is difficult because the trigger condition is more complicated than combinational-triggered hardware Trojans whose trigger are combinational circuits. In order to defeat hardware Trojans after the design step, defeating hardware Trojans during normal operation is an important issue.

In addition to the problems above, reconfigurable devices such as FPGAs and CPLDs (Complex Programmable Logic Devices) have been widely used in final productions. These devices can be easily reconfigured after the hardware products are released. In [51] and [52], the risks of hardware Trojans inserted into FPGA devices have been reported. We have to take the Trojan-infected FPGA issues into account.

Hardware Trojans are often composed of two parts: a trigger part and a payload part. The trigger part checks whether the trigger condition is met or not. The trigger condition often uses the internal signals and/or the internal states of the chip. When the trigger condition is met, the trigger signal becomes 1 and the signal propagates to the payload part. The payload part is for the malfunction of the hardware Trojan. There are several types of malfunctions such as leakage of internal information and disabling the entire

chip. When the trigger signal of the trigger part becomes 1, the malfunction works. In this subsection, we assume that the hardware Trojan has a trigger condition and the malfunction works only when the trigger condition is met. Considering the features of hardware Trojans, a net which holds the same value for a long time should be a Trojan net.

Several on-time hardware-Trojan detection methods have also been proposed in recent years. In [53], a hardware-Trojan detection method focusing on the supply voltage has been proposed. When a hardware Trojan works, the distribution of the supply voltage in the chip slightly change. By checking the change using on-chip voltage sensors which compares the voltage distribution to that of a golden chip (the chip guaranteed to be Trojan-free), this method successfully observed the activation of the hardware Trojan. However, this method needs a golden chip though it is difficult to guarantee a chip to be Trojan-free. In [54], an authentication method for suspicious Trojan nets in a circuit has been proposed. This method monitors suspicious Trojan nets which are extracted by existing hardware-Trojan detection method. When the suspicious Trojan net behaves like a normal net, the net is authenticated and the signal will pass through the gate. When the suspicious Trojan net behaves like a Trojan net, the net is not authenticated and the signal will be blocked by the gate. This method is applied to an AES encryption module and correctly blocks the Trojan nets.

Though on-chip hardware-Trojan detection is effective since it overcomes the difficulties of the design-time detection, there are several problems in on-chip hardware-Trojan detection.

In [54], an authentication circuit monitors a net in a chip, and authenticate the net if the signal transition count is large enough. In this method, a chip has two modes: the authentication mode and the normal mode. The authentication mode and the normal mode work as follows:

1. To begin with, the authentication circuit monitors a suspicious Trojan net in the authentication mode. If the signal of the net transits few times, the net will be deactivated and its value will be fixed to 0.

2. After that, the chip works normally in the normal mode.

Figure 3.8 shows the signal transitions of the net *Signal A* and a clock signal. In Figure 3.8, the transition count of *Signal A* becomes 4 before the 8th clock. When we consider that the transition count of 4 out of 8 clock cycles is large enough, we regard the net to be a normal net and authenticate it. Then, at the 8th clock, the authentication circuit authenticates *Signal A*. Figure 3.9 shows the signal transitions of the net (Signal B) and a clock signal. In Figure 3.9, the transition count of *Signal B* becomes 2, which is too few to regard the net to be a normal net. Therefore, at the 8th clock, the authentication circuit does not authenticate *Signal B*.

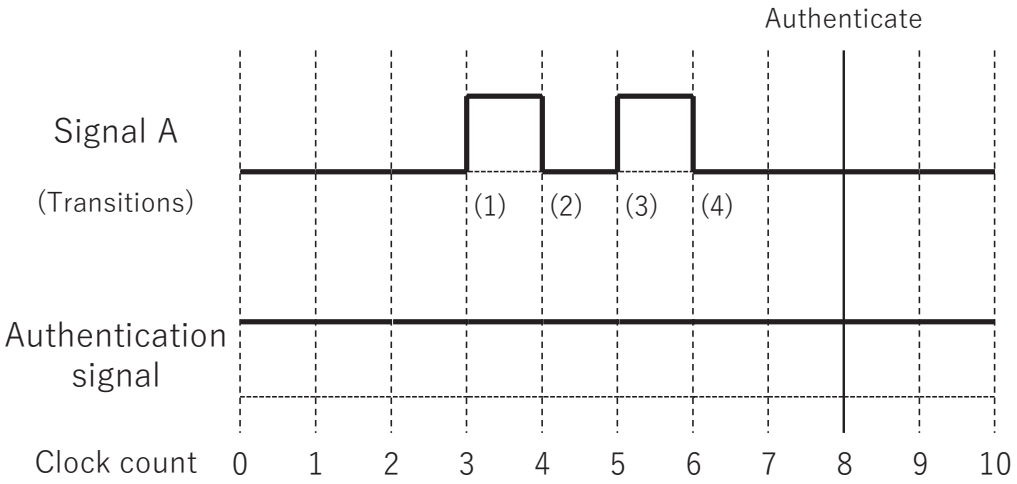


Figure 3.8: The signal transition of the monitored net (normal net).

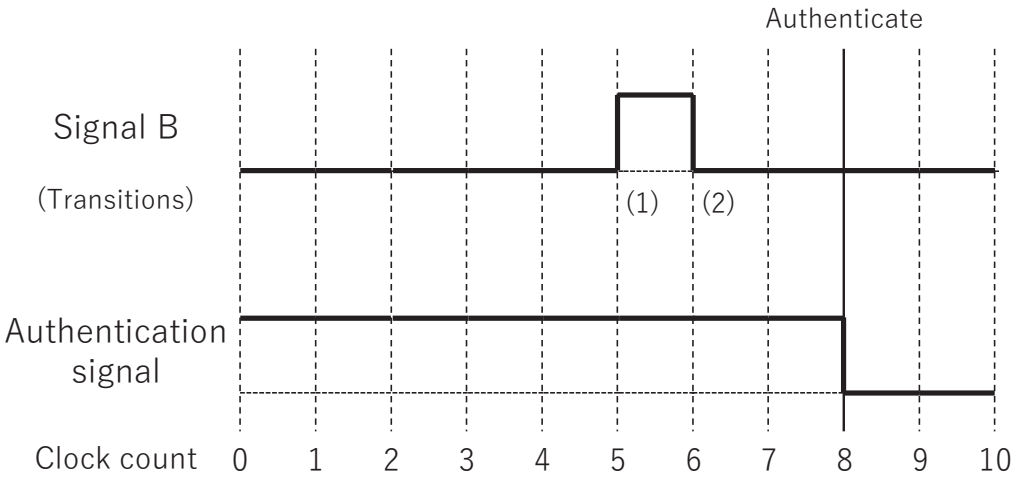


Figure 3.9: The signal transition of the monitored net (Trojan net).

In this subsection, we design a Trojan-invalidating circuit which includes an authentication circuit based on [54].

3.4.2 Designs of Trojan-Infected Cryptographic Circuit and Trojan-Invalidating Circuit

In this section, we design three parts of circuits: (i) the cryptographic part, (ii) the hardware Trojan part, and (iii) the Trojan-invalidating part.

We apply the AES algorithm to the cryptographic circuit. Firstly, we design an AES cryptographic circuit (i), which is called [a] in this subsection. Secondly, we design a hardware Trojan circuit (ii) and insert it into [a], which is called [b]. Finally, we design

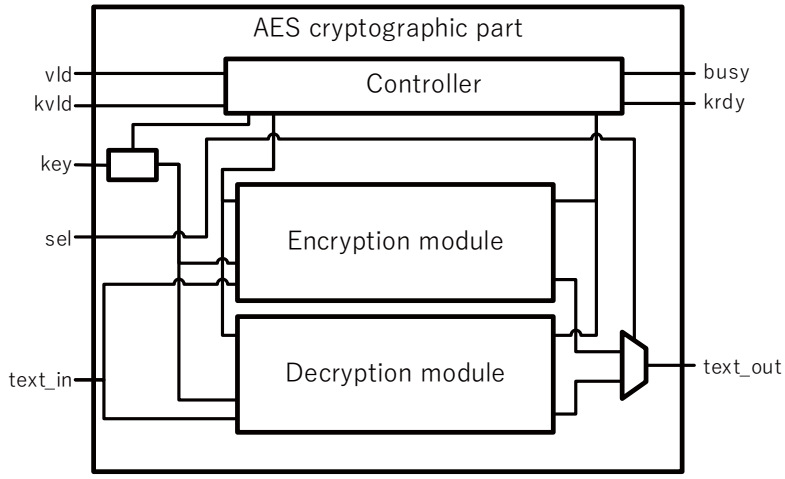


Figure 3.10: The block diagram of the AES cryptographic circuit.

Table 3.16: Input/output ports of the AES cryptographic circuit.

Port name	Direction	Width	Description
clk	In	1	System clock.
rstn	In	1	Reset signal. When the signal is 0, reset the internal state and registers.
vld	In	1	When 1, the circuit starts to encrypt or decrypt data.
kvld	In	1	When 1, the circuit obtain the cipher key to the internal register.
sel	In	1	When 0, the circuit encrypts data. When 1, the circuit decrypts data.
busy	Out	1	Output 1 while the circuit works.
krdy	Out	1	Output 1 when the cipher key is set.
key	In	128	128-bit cipher key.
text_in	In	128	128-bit data (plain text for encryption, coded text for decryption).
text_out	Out	128	128-bit output data (coded text for encryption, plain text for decryption).

a Trojan-invalidating circuit (iii) and insert it into [b], which is called [c].

We design the parts (i)–(iii) as the following subsections.

(i) The cryptographic part The AES is a symmetric block cipher algorithm. The AES is standardized by the National Institute of Standards and Technology (NIST) and it has been widely used recently. The length of a cryptographic key is 128, 192 or 256 bits. The length of encryption and decryption data block is 128 bits [55].

Figure 3.10 shows the block diagram of the AES cryptographic circuit. Table 3.16 shows the input/output ports of the AES cryptographic circuit. Input ports are 'clk', 'rstn', 'vld', 'kvld', 'sel', 'key', and 'text_in'. The signal 'clk' is a system clock and the signal 'rstn' is a reset signal. The signal 'vld' starts the process of encryption or decryption. When it become 0, a plain text or a coded text is obtained from the port

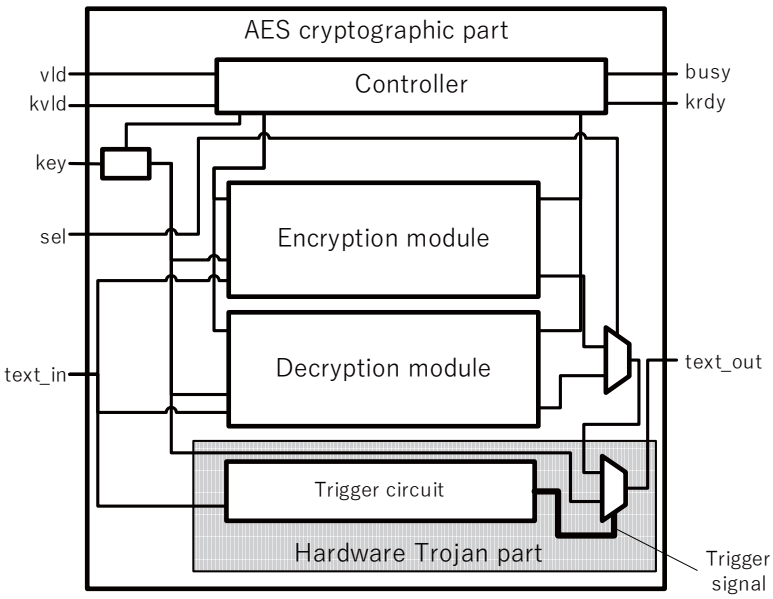


Figure 3.11: The block diagram of the Trojan-infected AES cryptographic circuit.

'text_in' and the encryption or decryption starts. The signal 'kvld' starts to obtain the cipher key from the port 'key'. The signal 'sel' selects encryption or decryption. The port 'key' obtains the 128-bit cipher key. The port 'text_in' obtains the 128-bit data to encrypt or decrypt. Output ports are 'busy', 'krdy', and 'text_out'. The signal 'busy' becomes 1 while the circuit is processing. The signal 'krdy' becomes 1 when the cipher key is set from 'key'. The port 'text_out' outputs the result of encryption or decryption.

(ii) The hardware Trojan part In this subsection, we design a hardware Trojan which leaks the cipher key set in the register of the AES cryptographic circuit. The designed hardware Trojan is composed of two parts: the trigger part and the payload part. The trigger part checks whether the trigger condition is satisfied. In this subsection, we set the trigger condition as that the input data 'text_it' is equal to the specific data. The payload part leaks the cipher key to the output port 'text_out' when the trigger condition is satisfied. After we design the hardware Trojan part, we insert the part into the AES cryptographic circuit and make a Trojan-infected AES cryptographic circuit.

Fig 3.11 shows the block diagram of the Trojan-infected AES cryptographic circuit. The shaded area in Fig 3.11 is the hardware Trojan part.

(iii) The Trojan-invalidating part In this section, we design a Trojan-invalidating circuit. The Trojan-invalidating circuit is composed of two parts: the authentication circuit and the gating circuit. The authentication circuit monitors a suspicious Trojan net

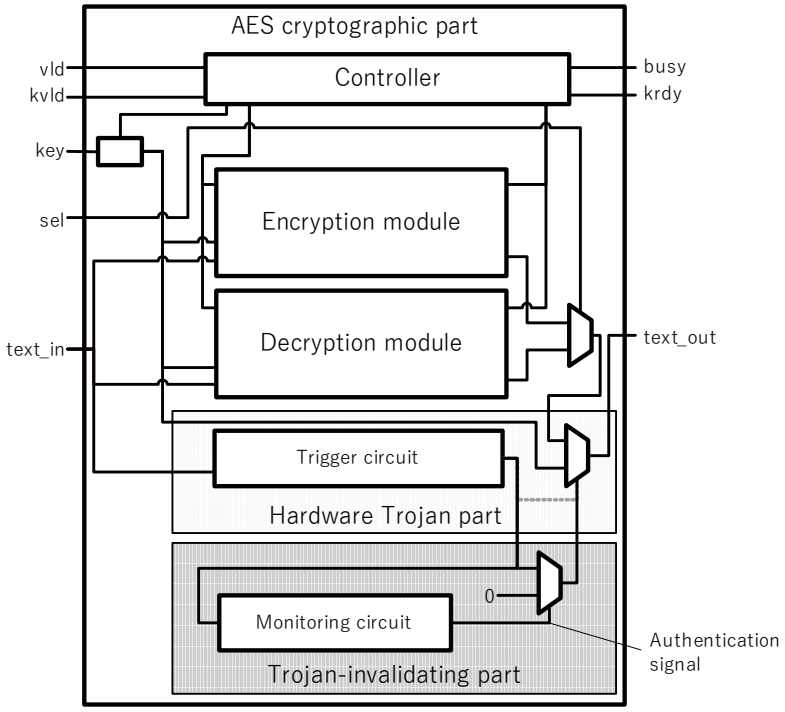


Figure 3.12: The block diagram of the Trojan-infected AES cryptographic circuit with a Trojan-invalidating circuit.

which is extracted by the existing design-time hardware-Trojan detection method such as [3] or [22]. In this subsection, we assume that the trigger signal in Figure 3.11 is picked up as a suspicious Trojan net by an existing design-time hardware-Trojan detection method. If the authentication circuit judges that the net should be Trojan net, the gating circuit blocks the signal of the net.

After we design the Trojan-invalidating circuit, we insert the circuit into the Trojan-infected AES cryptographic circuit. Figure 3.12 shows the whole circuit. The shaded area in Figure 3.12 is the Trojan-invalidating circuit. The Trojan-invalidating circuit works in the authentication mode at first (e.g. for the test in the production). After that, it works in the normal mode. The authentication mode and the normal mode work as follows:

- **Authentication mode**: The Trojan-inserted AES cryptographic circuit with the Trojan-invalidating circuit runs for a long time. At first, the monitoring circuit monitors the transition of a net, while the gating circuit does not work. If the transition count becomes more than 1 in each 8 clock cycles, the authentication circuit authorizes the net and the authentication signal becomes 1. If the transition count becomes less than or equal to 1 in each 8 clock cycles, the authentication

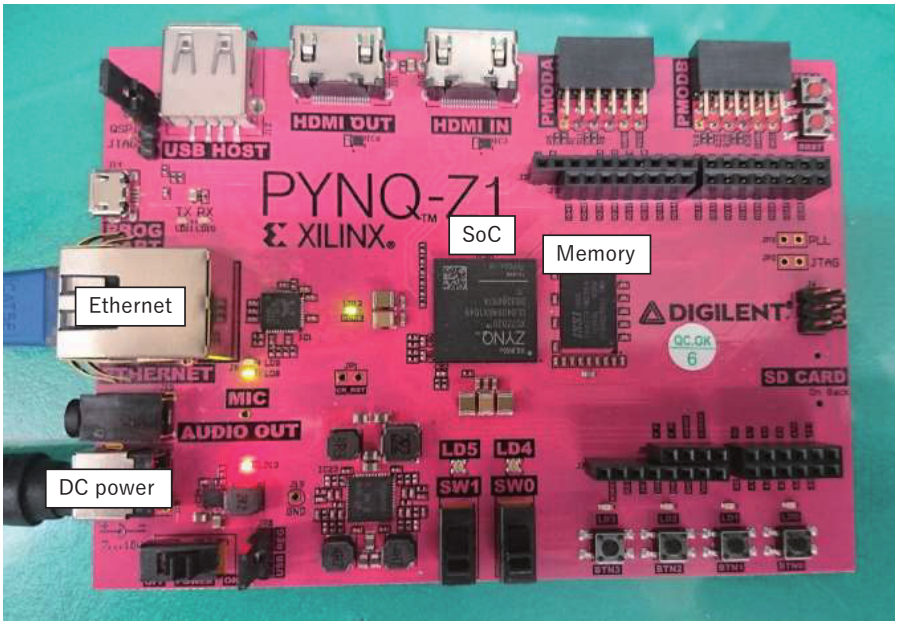


Figure 3.13: The overview of the PYNQ-Z1 board.

circuit considers the net as a Trojan net and the authentication signal becomes 0.

- **Normal mode**: The circuit works depending on the value of the authentication signal. When the value of the authentication signal is 0, the net is blocked. When not, the signal of the net propagates to the subsequent nets. If we can invalidate the trigger signal of the hardware Trojan, we can invalidate the whole hardware Trojan and we can run the chip normally.

3.4.3 Implementation and Evaluation

In this section, we implement the designs [a]–[c] described in Section 3.4.2 into an FPGA device.

FPGA board and environments

We use a Xilinx PYNQ-Z1 board as an FPGA device. Figure 3.13 shows the overview of the PYNQ-Z1 board. The PYNQ-Z1 board has a ZYNQ XC7Z020-1CLG400C SoC. This SoC has two parts: the Processing System (PS) and the Programmable Logic (PL). PS and PL are connected by a shared memory and bus lines, and we can exchange any data between PS and PL. PS has an ARM Cortex-A9 processor. In this experiment, Linux OS, Ubuntu 15.10 runs on PS. PL is an FPGA and we can program any logic circuit.

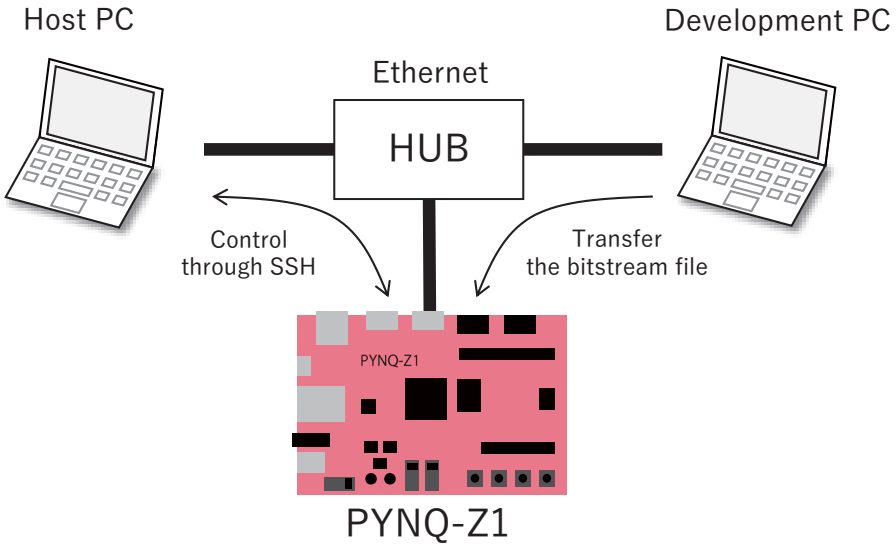


Figure 3.14: Experimental environments.

Table 3.17: FPGA implementation results.

Circuit	# of LUTs	# of FFs	# of BRAMs	WNS[ns]
[a]	2010 (3.78%)	2260 (2.12%)	12 (8.57%)	1.422
[b]	2058 (3.87%)	2259 (2.12%)	12 (8.57%)	1.363
[c]	2062 (3.88%)	2268 (2.13%)	12 (8.57%)	1.523

Figure 3.14 shows the experimental environment. We connect the PYNQ-Z1 board, a development PC, and a host PC through the network. The bitstream to configure to the PL is compiled on the development PC, and it is transferred from the development PC to PL on the PYNQ-Z1 board through the network. The host PC control the PYNQ-Z1 board through ssh connection.

FPGA implementation results

We used the development PC with Xilinx Vivado 2016.1. We applied default settings for the optimization options of the synthesis and the implementation. We utilized a OpenCores [56] project for the AES cryptographic circuit design. The PL clock frequency is set to 100MHz, which is the default setting of the PYNQ-Z1 board. Table 3.17 shows the number of LUTs, FFs, and BRAMs, and WNS (worst negative slack) for the implemented designs. The numbers in parentheses are the resource usage rates.

In Table 3.17, [b] uses 48 more LUTs than [a]. A hardware Trojan circuit is inserted into [b], and thus the increased LUTs are used in the hardware Trojan circuit. The ratio of increased LUTs is $48/2010 \simeq 2.4\%$ and thus the designed hardware Trojan is small.

[c] uses 4 more LUTs and 9 more FFs than [b]. A Trojan-invalidating circuit is inserted in [c], and thus the increased LUTs and FFs are used for the Trojan-invalidating circuit. Since this overhead is small, the Trojan-invalidating circuit is small compared to [a].

Experimental results

In this section, we demonstrate the experimental results using the implemented designs. We control the circuit from the host PC. In this experiment, we give a cipher key to the AES cryptographic circuit at first. After that, we give two plain text and encrypt them using the given cipher key. Figure 3.15, 3.16, and 3.17 shows the results of the experiments using the designs of [a], [b], and [c], respectively.

In Figure 3.15, the 32 characters in the first line are the cipher key in hexadecimal. 'Test 1' block shows the first encryption. 'Plain text' shows the plain text to encrypt, and 'Result' shows the coded text using the cipher key. 'Test 2' block shows the second encryption. 'Plain text' is different from 'Test 1' and thus the result is also different from 'Test 1'. In Figure 3.15, the AES encryption is successfully done in both of 'Test 1' and 'Test 2'.

In Figure 3.16, the result in 'Test 2' is different from that of Figure 3.15. Since [b] is infected by the hardware Trojan, the result of 'Test 2' is changed by the hardware Trojan, and therefore the cipher key is leaked.

In Figure 3.17, both the hardware Trojan and the Trojan-invalidating circuit work. To begin with, the circuit works in the authentication mode and the Trojan invalidating circuit monitors the behavior of the trigger net in the hardware Trojan. As a result, the trigger net is successfully invalidated. After that, the circuit works in the normal mode. As a result, the AES encryption successfully works and the result of 'Test 2' in Figure 3.17 is correct. Note that we assumed that the trigger net is picked up by an existing design-time hardware-Trojan detection method such as [4] or [22].

3.5 Conclusion

In this chapter, we proposed three applications of machine-learning-based hardware Trojan detection.

First, we propose a hardware-Trojan classification method using multi-layer neural networks. The experimental results demonstrate that our proposed method outperforms several existing methods.

Second, we propose a Trojan-invalidating circuit that prevents a hardware Trojan circuit from activating. The implementation evaluation on an FPGA board demonstrate that the implemented circuit successfully disable the hardware Trojan circuit.

Third, we propose an approach to correct detection results by a machine-learning-based hardware-Trojan detection method. The experimental results demonstrate that our proposed method successfully correct mistakenly identified Trojan nets to be Trojan nets.

In Chapters 2 and 3, machine-learning-based hardware-Trojan detection methods and their applications are proposed. By leveraging the proposed methods, we can effectively detect hardware Trojans at gate-level netlists. Although the proposed methods still have remaining problems in terms of TPR and TNR, these methods provide us with a great meaningful clues from the viewpoint of the detection of hardware Trojans at the design step.

In the future, we can further develop the extended-version of machine-learning-based hardware-Trojan detection. Identifying the types of hardware Trojans by utilizing multi-class classification and targeting obfuscated designs are our future works.

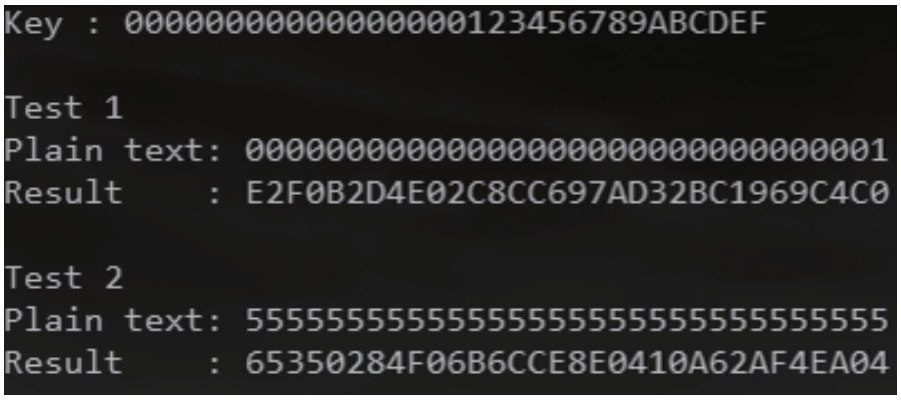


Figure 3.15: Output results of [a].

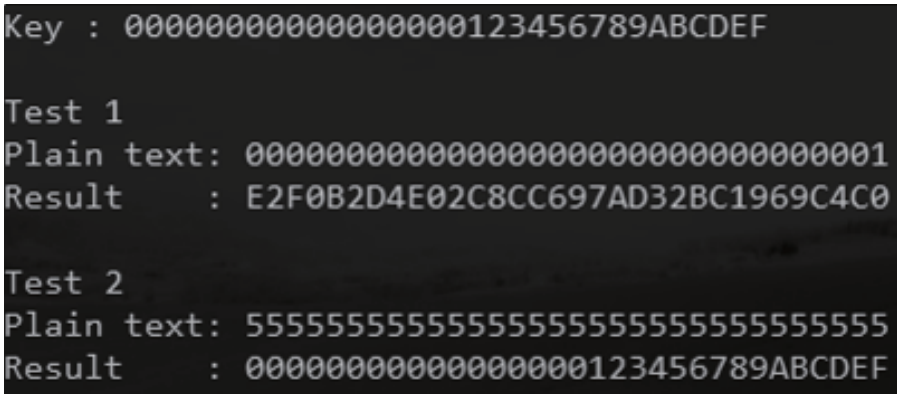


Figure 3.16: Output results of [b].

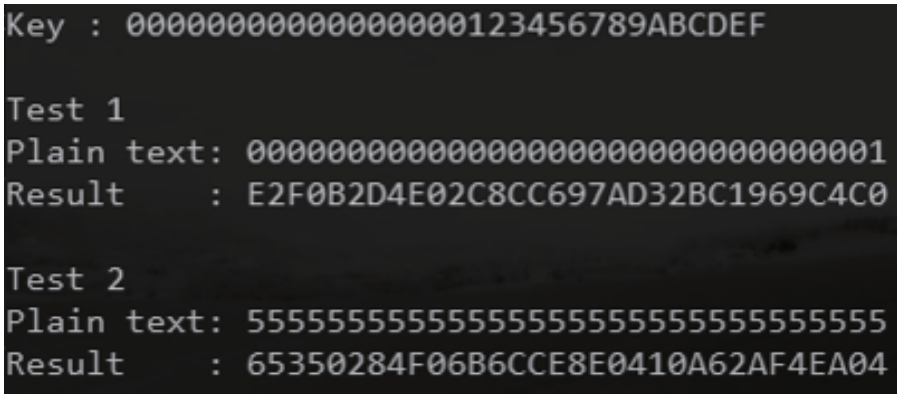


Figure 3.17: Output results of [c].

Chapter 4

Malicious Behavior Detection Based on Power Analysis[1]

4.1 Introduction

The more critical threats have been reported as the more high-functioning IoT devices have been embedded into the heart of mission-critical devices connected to the Internet. Since hardware/software vendors produce their IoT products easily and inexpensively, they often outsource their designs to third-party vendors where malicious third-party vendors can have a chance to insert software Trojans as well as "hardware Trojans" into their IoT devices [10]. Here we focus on hardware-oriented security issues embedded into IoT devices.

Chapters 2–4 focus on hardware Trojan detection at the design step. However, hardware Trojans inserted at the manufacturing step have also threatened hardware production. In this chapter, we focus on the threats after the design step.

Monitoring completely a large amount of main-channel data from/to a large amount of IoT devices in a "cloud level" is impractical. In contrast, edge-level monitoring is the idea to monitor the individual IoT device, or edge device. Once this method is established, almost all the malicious behaviors in IoT devices can be detected locally, leading to reduce the burden to monitor the remaining main-channel data.

Since every IoT device has its unique embedded micro-controllers, operating system, and embedded software and hardware, it is very difficult to establish the general edge monitoring method. This could be solved simply by not directly monitoring the main-channel communications but effectively utilizing side-channel signals, such as voltages, currents, and power signals of the target IoT device.

In [57], a side-channel-based hardware-Trojan detection method has been proposed.

[1]Technical contents in this chapter have been presented in the publications ⟨7⟩ and ⟨11⟩.

Its detection method effectively utilizes side-channel information in the experiment, but this method requires the 'Golden' circuit where no hardware Trojan is inserted. Therefore, in this chapter, we learn the normal behavior of the target device based on side-channel signals and detect abnormal behavior. In [58], based on the assumption that a malicious function is rarely triggered, power waveform in a normal process is learned beforehand and the abnormal behavior is detected. If we can extend this approach and empirically demonstrate its effectiveness utilizing more high-functioning micro-controllers, this approach can be applied to a wide range of IoT devices to detect their abnormal behaviors

In this chapter, we propose an anomaly behavior detection method in a low-cost micro-controller by effectively utilizing an outlier detection algorithm based on accurate power analysis. We learn the normal behaviors as well as several abnormal behaviors of a target IoT device based on the abnormal scenarios prepared beforehand. Based on the learned classifier, we can effectively distinguish between normal and abnormal behaviors in micro-controllers.

The contributions of this chapter are summarized as follows:

1. We propose an anomaly behavior detection method for low-cost micro-controllers utilizing accurate power analysis based on an outlier detection method. Our proposed method does not require Golden model or detailed preliminary analysis.

2. We empirically evaluate the proposed method and successfully detect the abnormal behaviors inserted into micro-controllers.

4.2 Related Works

This section introduces several existing works on hardware-Trojan detection based on side-chanenel analysis, and clarify the problems of them.

Hardware devices contain many parts, and especially ASIC, FPGA, and microcontroller handle complicated processes. ASICs, FPGAs, and microcontrollers are attractive targets for malicious vendors to insert malfunctions because circuits are packed in packages where attackers can hide malicious circuits. For ASICs and FPGAs, how to deal with the threat is widely discussed by researchers. The malfunctions inserted into ASICs and/or FPGAs are called 'hardware Trojan' and their detection methods have been proposed in recent years. An ASIC has a circuit for a specific application, and an FPGA has a programmable circuit which can be configured after manufacturing. Both an ASIC and an FPGA have a common point that they have a physical circuit for a specific application. A path-delay based method [59] for these ASIC devices and a feature-based method [22] for the hardware design have been proposed, and these studies demonstrate good results. On the other hand, malfunction detection for microcontrollers are not discussed

Table 4.1: Flexibility and performance of ASIC, FPGA, and Microcontroller.

Platform	Flexibility	Performance
ASIC	Poor	Excellent
FPGA	Good	Good
Microcontroller	Excellent	Poor

enough as far as we know. They are widely used in small systems like the controller of a game console and the touch pad of a laptop computer, and therefore this issue must be considered. As shown in Table 4.1, microcontrollers have more flexibility than ASICs and FPGAs. Unlike ASICs and FPGAs, microcontrollers interpret binary commands in program memories which do not use specific circuits. Thus the existing hardware-Trojan detection methods for ASICs and FPGAs cannot be applied to microcontrollers. The programs in microcontrollers are written in general programming languages such as C and C++ languages, but expert knowledges are required to implement programs on microcontrollers. Since typical small systems have no operating systems, programmers required to directly access memories and peripherals. Because of these points, anti-virus softwares cannot be applied to the programs on microcontrollers. Detecting malfunctions inserted into microcontrollers requires a different approach from hardware-Trojan detection methods and anti-virus softwares.

Several methods for detecting malicious behaviors in integrated circuits (ICs) utilizing power analysis have recently been proposed. For example, [57] and [60] are the malicious behavior detection methods based on power analysis. According to them, these methods have successfully detected malicious behaviors in field-programmable gate arrays (FPGAs). However, the key idea of these methods is to compare a device under test (DUT) to its Golden model and analyze difference between them. In order to effectively employ these methods, how to prepare a Golden model is a critical problem. Considering the practical supply chain, preparing a Golden model is difficult because we have no methods to check whether an IC is Golden or not.

Other approaches targeting a micro-controller such as in [61] and [62] have also been proposed. These approaches analyze the relationship between power consumption and instruction sets. Applying these approaches to a micro-controller, we can predict what instruction is executed there. Based on the analysis, we can detect whether the behavior is malicious or not. However, these methods need to preliminarily analyze the instruction sets of the target device in detail. Since the number of IoT devices is rapidly increasing, we have no time for the detailed preliminary analysis.

In order to develop a malicious behavior detection method, the following two points are required: to use no Golden model, and not to require preliminary analysis. Therefore, we develop a malicious function detection method based on unsupervised machine

learning utilizing accurate power analysis.

In this chapter, we try to detect the existence of malfunctions for microcontrollers using power analysis. Power analysis has been utilized for hardware-Trojan detection [57], however, this method requires the golden circuit which is guaranteed to be Trojan-free. Here we focus on a tiny IoT system like a sensor logger. Such a tiny IoT system is required to save power because it is driven by a battery. The tiny system does not always work, but it intermittently works and sleeps. When the system works normally (*active mode*), it consumes several mW power. When the system sleeps (*sleep mode*), it consumes little power (approximately several μW). Our proposed method distinguishes the active mode and the sleep mode using power analysis. Calculating the total power consumption and the total time of the active mode, we utilize unsupervised machine learning and identify whether a malfunction expresses or not.

Several side-channel-based methods such as [61] and [63] have also been proposed very recently, but they are too specific to the target micro-controllers. Moreover, our goal is not to disassemble the executing code but to detect abnormal behavior of the target micro-controller utilizing side-channel signals. Therefore, in this chapter, we learn the normal behavior of the target device based on side-channel signals and detect abnormal behavior.

4.3 Threat Model

This section elaborates the threat model of micro-controllers and the characteristics of them in terms of operation modes.

4.3.1 Malicious Function Inserted into a Micro-Controller

Since a micro-controller runs an application on its central processing unit (CPU), the application can be configured to the micro-controller by using a computer through wired or wireless communication connection. In addition, recent IoT devices utilizing micro-controllers have been connected to the Internet and they automatically check the update by themselves. When they find the available update application, they download and apply it themselves. Though the automatic update is useful, there is a risk that attackers may replace genuine binary codes with malicious ones abusing this automatic updating system. The malicious function inserted into the target micro-controller will leak its internal information such as a secret key or stored data, or disable its function. Detecting malicious functions inserted into micro-controllers must require a different approach from hardware-Trojan detection methods and anti-virus softwares.

In order to conceal the malicious function, attackers often set a trigger condition to

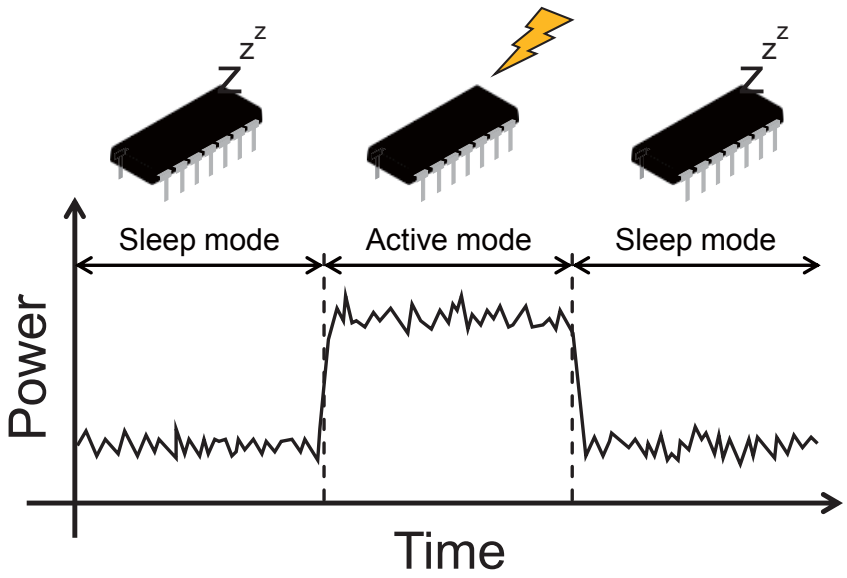


Figure 4.1: The model of consumed power in the sleep mode and the active mode.

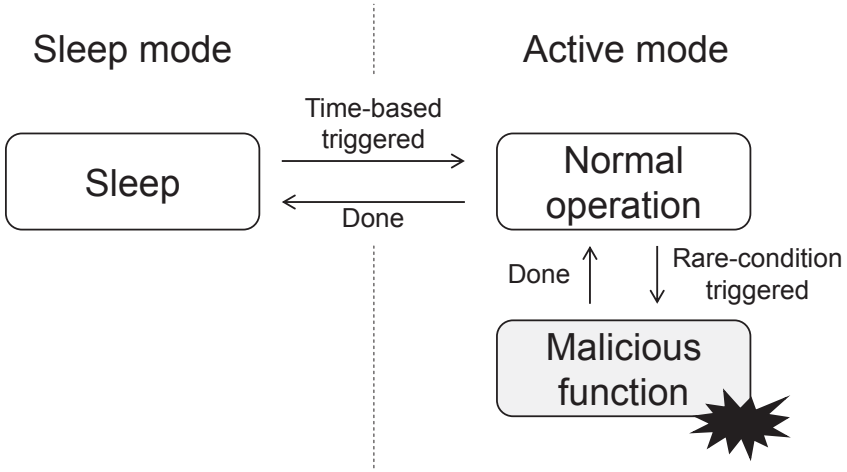


Figure 4.2: The operation modes in our threat model.

activate the malicious function. In this chapter, we focus on the triggered-type malicious functions, and we learn the behaviors including several abnormal behaviors of an IoT device based on the abnormal scenarios prepared beforehand.

4.3.2 Operation Modes of Micro-Controllers

In this chapter, we assume that a tiny IoT system utilizes a low-cost micro-controller. Such a low-cost micro-controller often has a power-saving function because a tiny IoT system is typically required to save power to run for a long time with a small-capacity

battery. In this section, we call the power-saving mode as 'sleep mode.' In contrast, we call the normal mode as 'active mode.' Figure 4.1 depicts the consumed power of the microcontroller. The consumed power slightly changes every moment, but the difference between the consumed power in the active mode and that in the sleep mode should be distinguishable. In the active mode, the micro-controller runs normally. For an instance of a sensor-logging system, a micro-controller obtains sensor values from the connected sensors, and transmits the obtained values to a host computer. This operation consumes several mW power. On the other hand, in the sleep mode, the micro-controller saves power. Several components in the micro-controller are powered off by power gating except for the minimally required components. The consumed power in the sleep mode is quite less compared to in the active mode. The consumed power slightly changes every moment, but the consumed power in the active mode and in the sleep mode should be distinguishable. This chapter utilizes its difference to classify the waveform into active-mode parts and sleep-mode parts, and we focus on the active-mode parts.

Now we consider that a micro-controller is infected with a malicious function. Now we consider that a malicious function is inserted into a micro-controller by an attacker. Figure 4.2 shows the assumed operation of the target micro-controller. In the sleep mode, a micro-controller cannot do anything because only the minimal components in the micro-controller work and the others are powered off. In the active mode, a micro-controller works normally. However, under the rare condition during the active mode, a malicious function is activated. When the malicious function is activated, side-channel signals must reflect its existence.

In this chapter, we focus on power consumption and detect abnormal behavior based on a outlier detection method. According to [58], an anomaly behavior detection method for a micro-controller has been proposed, but it focuses on a low-cost micro-controller and just one experiment is carried out. In this chapter, we propose an anomaly behavior detection method utilizing accurate power analysis.

4.4 Malicious Behavior Detection Algorithm Based on Power Analysis

In this section, we propose an anomaly behavior detection method utilizing accurate power analysis. The procedure of the proposed anomaly behavior detection method is summarized as follows:

1. **Power measurement**: Measure consumed power from a target device.

2. **Waveform smoothing**: Smooth the waveform of the measured power.

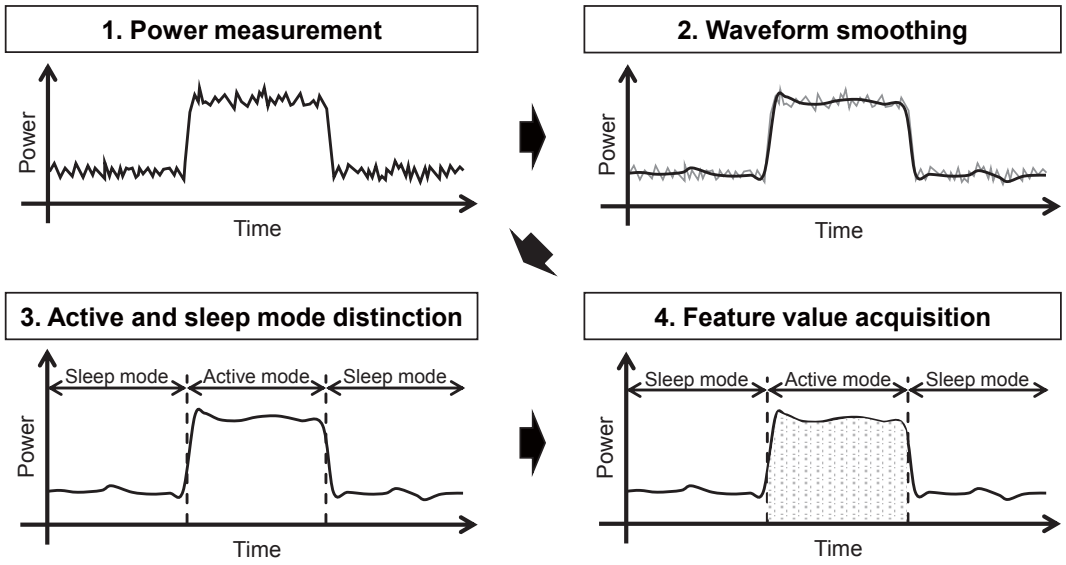


Figure 4.3: The procedure of our proposed method.

3. **Active and sleep mode distinction**: Distinguish between the active mode and the sleep mode using an unsupervised machine learning algorithm.

4. **Feature value acquisition**: Acquire feature values from the waveform in the active-mode.

5. **Anomaly behavior detection**: Detect abnormal behavior from the acquired feature values using an anomaly detection algorithm.

Figure 4.3 illustrates how to process measured waveform form 'power measurement' to 'feature value acquisition'. The details in each step are elaborated in the following subsections.

4.4.1 Power Measurement

First, we measure time-domain consumed power waveform for a target IoT device, separately measuring the voltage V and current I, and then obtaining $P = V \times I$. In general, a clamp-on type AC/DC current probe may be used for this purpose. However, the performance is not enough in repeatability, sensitivity, and dynamic range and it is not always suitable for measuring the power rail current in IoT devices [64]. Furthermore, the clock frequency of a low-cost micro-controller sometimes exceeds 100MHz these days, therefore we have to prepare an instrument having precise current sensing capability as well as enough bandwidth. Based on the discussion above, we leverage a current sensor which accurately measures current flow.

4.4.2 Waveform Smoothing

The measured power itself is noisy, and we have to smooth it before analyzing it. A simple moving average, which is frequently used to find the trend of time-series data, is used in our proposed method. Let x be the time-series data, and let $x[n]$ be the value at time n. N-data moving average at time n is represented as $y[n]$ and it is expressed by $y[n] = \frac{1}{N}\sum_{i=0}^{N-1} x[n-i]$. We set N to be 5 in this chapter.

4.4.3 Active and Sleep Mode Distinction

In this step, we classify the waveform into a part of active mode and a part of sleep mode. An unsupervised clustering method, k-means method is applied to this step. k-means in one of the unsupervised clustering algorithm which is a simple unsupervised clustering algorithm [65].

In our proposed method, we set the number of clusters to two because we classify the waveform into a part of active mode and a part of sleep mode. Since k-means method requires no threshold values, parameter tuning for numerous types of IoT devices is unnecessary.

4.4.4 Feature Value Acquisition

Now we focus on the active-mode part because malicious behavior should appear in the active mode. In this step, we extract feature values from the waveform in the active mode. As discussed in Section 4.3, a malicious behavior should appear in the active mode under a rare condition. When a malicious behavior such as denial-of-service or leakage of internal information appears, the duration time and consumed power in the active-mode period must be different from ones in the normal situation. Based on the discussion above, we extract the two following feature values:

1. the duration time of the active-mode period;

2. the consumed energy over the active-mode period.

In the following anomaly detection step, we give these feature values to the anomaly detection method and detect the abnormal behavior.

4.4.5 Malicious Behavior Detection

In order to detect abnormal behaviors based on the given feature values, the local outlier factor (LOF) [66] is adopted in our proposed method. The LOF method regards the given samples which have low density compared to their neighbor samples as outliers.

The duration time and consumed power in the active mode will differ from time to time in the anomaly behavior detection. However, if a malicious function works, the feature values will quite different from those in normal behavior. The LOF method is expected to effectively detect such a abnormal behavior.

Before we apply the LOF method to the obtained feature values in the anomaly detection step, we must standardize the data. Since the LOF method is based on density of feature values in the feature space, the scale of each axis affects its result. In order to standardize the data, we first calculate the average and variance values for each features. Assume that we have N two-dimensional data $(x_i, y_i) \in \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$. Let μ_x and μ_y be the average values for x and y, respectively. In the same way, let σ_x and σ_y be the variance values for x and y, respectively. The standardized feature value for x, x'_i becomes $(x_i - \mu_x)/\sigma_x$. Similarly, the standardized feature value for y, y'_i becomes $(y_i - \mu_y)/\sigma_y$. The standardization is effective when the scale of features are quite different. In this chapter, since we utilize the duration time and consumed power whose scales are quite different, we apply standardization to the obtained data before the anomaly detection step.

After standardization, we apply the LOF method to the standardized feature values of the active-mode periods. Finally, we obtain the LOF values for each active-mode period. When a LOF value is clearly different from the others, we identify the period to be the abnormal behavior, that is, a malicious function appears in that period.

4.5 Experimental Results

In this section, we demonstrate the empirical evaluation and its results of the proposed method.

4.5.1 Experimental Setup

Figure 4.4 illustrates the connection diagram of the measurement devices and Figure 4.5 shows the measurement setup. Keithley 2280S-32-6 [67] is used as a power supply device, and it supplies power to the target device. In the experiment, the output voltage is set to 5V and the maximum current is set to 0.4A. Keysight CX3324A [68] is used to measure the voltage and current flow of the target device. Keysight CX3324A [69] is used to measure the voltage and current flow of the target device. The voltage is measured by a general passive probe and the current flow is measured using a current sensor Keysight CX1101A.

As described in [64], a clamp-on current probe's minimum measurable current is 1-3mA. On the contrary, CX1101A covers up to 1A and has lower minimum measurable

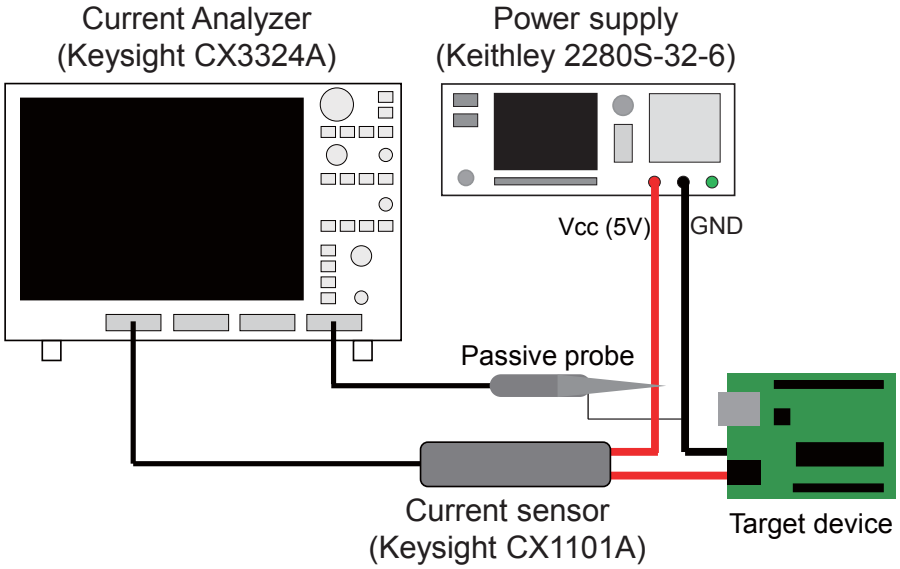


Figure 4.4: The connection diagram.

current as small as 3uA with bandwidth of 100 MHz at small input insertion resistance of 0.41Ω [69]. This low level measurement at high bandwidth is made possible utilizing new current sensing scheme combining sense resistor and current transformer described in [70]. Due to the small insertion resistance, we can avoid large voltage drop in power-rail and brown-out of target devices even when large current spike is observed. Therefore, with using this type of current sensors, we can precisely capture dynamic current flow in power rail of the target device repeating steep active and sleep transition.

After the acquisition of the power waveform using the measurement devices above, we save the data in comma-separated values (CSV) format and analyze it using a computer. The analysis procedure is written in Python 3 language and used scikit-learn [35], a data-mining library, for k-means classification and LOF detection. Unless otherwise noted, the default parameters of scikit-learn is applied to k-means classification and LOF calculation.

4.5.2 Target Devices

In this experiments, we utilized two types of micro-controller development boards, Arduino UNO [71] and Nucleo L476RG [72]. They are widely used in research fields and several low-cost IoT products use them.

Arduino UNO: Arduino UNO uses a Microchip ATmega328P chip, which is an AVR 8-bit RISC micro-controller. The operation frequency of ATmega328P can be set up to 20MHz, but the default setting for Arduino UNO is 16MHz. In this experiment, we implemented the application described below into the micro-controller with Arduino

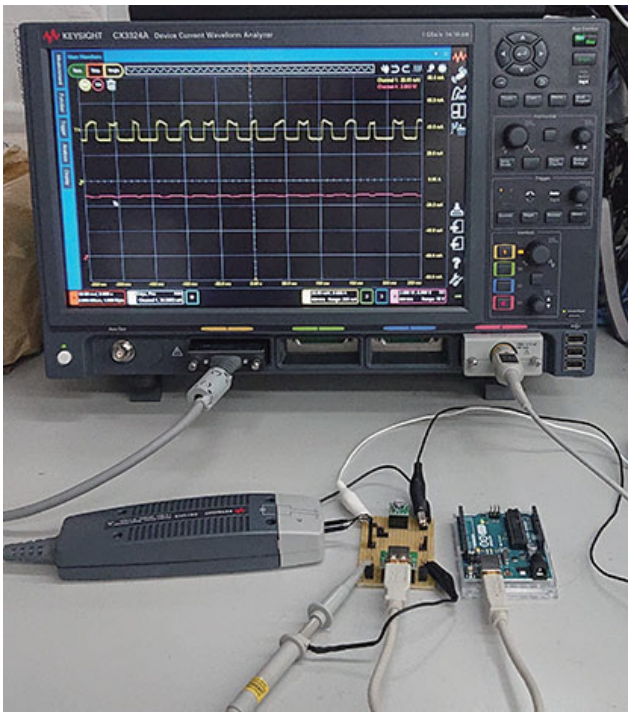


Figure 4.5: Measurement setup.

libraries, and the micro-controller is clocked at 16MHz. We set the trigger duration time to wake the micro-controller up from the sleep mode to 32ms utilizing the watch dog timer.

Nucleo L476RG: Nucleo L476RG uses a STMicroellectronics STM32L476RG chip, which is an ARM Cortex-M4 32-bit micro-controller. The operation frequency of STM32L476RG can be set up to 80MHz, and therefore the micro-controller is clocked at 80MHz in this experiment. Nucleo L476RG is more high-performance compared to Arduino UNO. We set the trigger duration time to wake the micro-controller up from the sleep mode to 50ms utilizing an internal timer.

4.5.3 Target Application

For the experiments, we implemented a sensor-logging application. The overview of the application is illustrated in Figure 4.6. First, the micro-controller reads the voltage of an analog input port, and convert the voltage to a digital value at the analog-to-digital (A/D) conversion step. Second, the micro-controller encrypts the obtained data using Advanced Encryption Standard (AES) at the AES encryption step. For the implementation of the AES encryption, we used an AES encryption library [73]. After that, the micro-controller transmits the encrypted data to a host computer through a serial interface at the serial

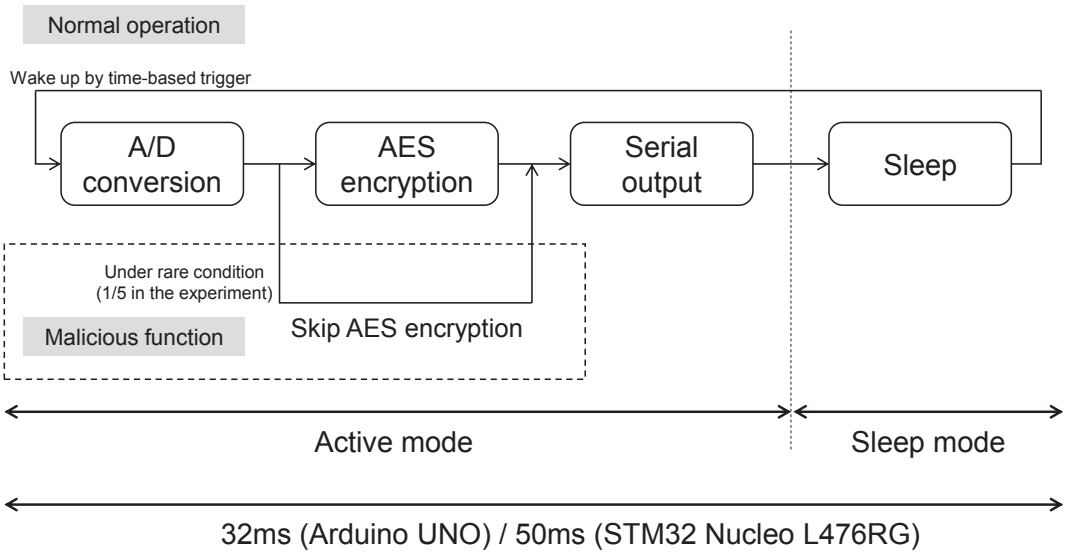


Figure 4.6: The overview of the sensor-logging application and the malicious function implemented into the target devices.

output step. The three steps are processed as an active mode. After the active-mode process is completed, the micro-controller goes into the sleep mode to save power. This process is repeated every 32ms on Arduino UNO and every 50ms on Nucleo L476RG.

In order to implement a malicious behavior, we disable the AES encryption of the A/D conversion results once every five times. When the malicious function is triggered, the AES encryption is disabled and the A/D conversion results are transmitted without encryption.

4.5.4 Malicious Behavior Detection Results

Experimental Results for Arduino UNO

Figure 4.7 shows the obtained power consumption of Arduino UNO. Figure 4.8 and Table 4.2 show the results of anomaly behavior detection for Arduino UNO. In Figure 4.8, the x-axis shows the duration time and the y-axis shows the consumed energy in each active-mode period. The background shade in the plot shows the degree of LOF. The deep shade shows the low LOF value and the light shade shows the high LOF value as shown in the right side of the plot. As shown in Figure 4.8, there are two clusters in the top right side and bottom left side. The samples in the cluster of the top right side show that their duration time is longer and their consumed energy is higher than the other cluster. The cluster of the top right side is identified to be normal by the proposed method. On the other hand, the cluster of the bottom left side is identified to be abnormal because they are quite different in feature values from the normal cluster and the number

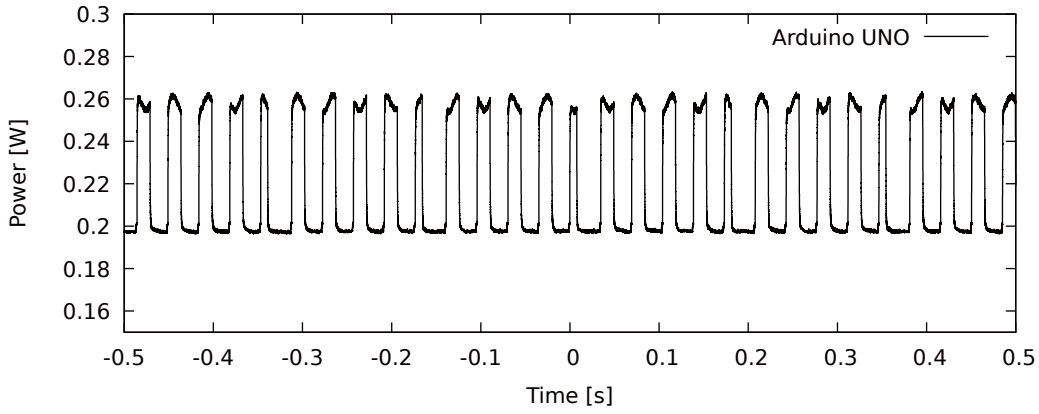


Figure 4.7: Measured power consumption for Arduino UNO.

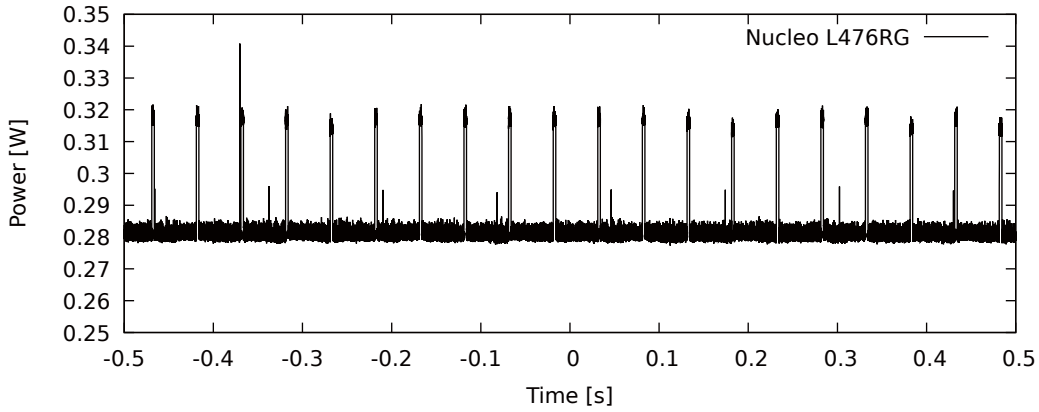


Figure 4.9: Measured power consumption for Nucleo L476RG.

of samples in this cluster is less than in the normal cluster. In Table 4.2, each row shows the feature values (duration time and consumed energy) and LOF value in each active-mode period. The underlined values in the LOF column show that they are identified to be abnormal behaviors. As shown in Table 4.2, the LOF values of normal and abnormal behaviors are clearly different. Based on the results, we can set a LOF threshold value to -100 to identify whether the period is normal or abnormal. By utilizing the threshold value, we can completely detect all abnormal behavior periods out of 29 periods in this experiment. Because the operation frequency of Arduino UNO is low, we can easily observe the behavior through power analysis in this case. Arduino UNO employs low operation frequency and the power rail current noise due to the CMOS switching noise is low, these make the power analysis easy in this case.

Experimental Results for Nucleo L476RG

Figure 4.9 shows the obtained power consumption of Nucleo L476RG. As shown in Figure 4.9, the duration time of the active mode is quite short compared to the duration time shown in Figure 4.7. Figure 4.10 and Table 4.3 show the results of abnormal behavior detection for Nucleo L476RG. In Figure 4.10, the x-axis, y-axis, and the background shade show the duration time, consumed energy, and degree of LOF, respectively in the same way as in Figure 4.8. In Table 4.3, the underlined values in the LOF column show that they are identified to be abnormal behaviors. As shown in Table 4.3, we can set the LOF threshold value to -1 to identify whether the period is normal or abnormal. In this case, the difference of LOF values between normal and abnormal is slight compared to the results of Arduino UNO. Moreover, we cannot detect abnormal behavior just utilizing the energy consumption. For example, the consumed energy values of the fifth and seventh active-mode part in Table 4.3 are almost the same, but the fifth period is normal and the seventh one is abnormal because the duration time is short in the seventh period. By leveraging our proposed method, we can completely detect all abnormal behavior periods out of 20 periods in this experiment even if the difference between normal and abnormal periods is slight. Therefore, combining duration time with consumed energy obtained by accurate power analysis enables us to successfully identify whether the behavior is normal or abnormal, even if the micro-controller is clocked at 80MHz which is a fast clock speed for a low-cost micro-controller.

The difference in consumed energy between normal and abnormal in Table 4.3 is quite slight compared to that in Table 4.2. Because of the slight difference in consumed energy, we cannot detect abnormal behavior in Nucleo L476RG by just analyzing energy consumption. Our proposed method with LOF effectively detects abnormal behavior utilizing accurate power analysis.

4.6 Conclusion

In this chapter, we propose an anomaly behavior detection method utilizing accurate power analysis. In our experiment, we implemented a sensor-logging application into two types of micro-controllers, and empirically evaluate the anomaly behavior detection method utilizing accurate power analysis. The experimental results demonstrate that the proposed method successfully detects the abnormal behavior of low-cost micro-controllers which are clocked at up to 80MHz utilizing accurate power analysis. Even if the difference of the duration time and consumed power between normal behavior and abnormal behavior is quite slight, our proposed method successfully detect abnormal behaviors.

Although the proposed method is successfully applied to the model application such

as a sensor logger or a frequently-driven device in this chapter, the application to the real-world devices is still remained to be studied. In the future, we will target high-functioning micro-controllers and complicated applications which are more difficult to obtain power profile.

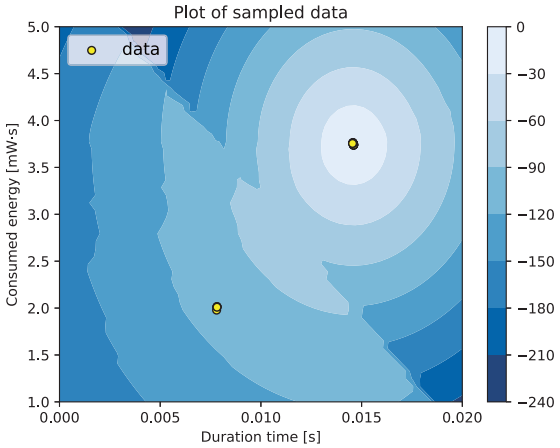


Figure 4.8: The plot of the obtained samples for Arduino UNO.

Table 4.2: Experimental results for Arduino UNO.

Active mode period	Duration [s]	Consumed energy [mW · s]	LOF
1	0.01461	3.74	-1.01
2	0.01457	3.76	-1.02
3	0.01457	3.76	-1.00
4	0.01459	3.74	-1.10
5	0.00782	2.02	-109.57
6	0.01457	3.77	-1.07
7	0.01458	3.75	-0.94
8	0.01461	3.74	-1.09
9	0.01458	3.75	-0.94
10	0.00782	2.01	-109.67
11	0.01459	3.75	-0.98
12	0.01461	3.74	-1.06
13	0.01458	3.75	-0.94
14	0.01457	3.76	-1.01
15	0.00780	1.98	-110.45
16	0.01460	3.74	-0.99
17	0.01457	3.76	-1.02
18	0.01457	3.76	-0.99
19	0.01460	3.74	-1.10
20	0.00782	2.02	-109.57
21	0.01456	3.76	-1.02
22	0.01459	3.75	-0.90
23	0.01460	3.74	-1.14
24	0.01459	3.76	-0.94
25	0.00782	2.01	-109.69
26	0.01459	3.75	-0.98
27	0.01460	3.74	-1.09
28	0.01458	3.76	-0.99
29	0.01455	3.76	-1.01

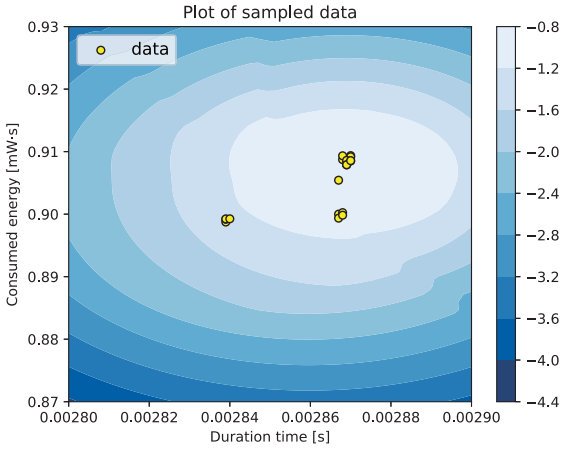


Figure 4.10: The plot of the obtained samples for Nucleo L476RG.

Table 4.3: Experimental results for Nucleo L476RG.

Active mode period	Duration [s]	Consumed energy [mW · s]	LOF
1	0.002868	0.9087	-0.99
2	0.002839	0.8987	-1.35
3	0.002870	0.9094	-0.99
4	0.002869	0.9086	-0.99
5	0.002867	0.9000	-0.99
6	0.002868	0.9093	-0.99
7	0.002839	0.8992	-1.34
8	0.002869	0.9080	-0.99
9	0.002870	0.9092	-0.99
10	0.002869	0.9086	-0.99
11	0.002870	0.9085	-0.99
12	0.002839	0.8992	-1.34
13	0.002867	0.9054	-0.99
14	0.002867	0.8994	-0.99
15	0.002869	0.9079	-0.99
16	0.002869	0.9079	-0.99
17	0.002840	0.8993	-1.31
18	0.002868	0.9002	-0.99
19	0.002870	0.9086	-0.99
20	0.002868	0.8998	-0.99

Chapter 5

Conclusion

In this dissertation, we have proposed machine-learning-based hardware-Trojan detection methods based on effective feature values. The most important goal of this dissertation is **to establish effective hardware-Trojan detection methods** which will protect hardware products from the threats of hardware Trojans. Overall, this dissertation proposes machine-learning-based hardware-Trojan detection methods which adequately detect hardware Trojans in the experiments. In this point, this dissertation has successfully achieved the goal.

As discussed in Section 1, hardware devices have widely been used in our daily lives, and security threats at hardware devices have been pointed out. The threats of hardware Trojans have risen and researchers have studied very recently. Machine learning is one of the powerful strategies to tackle the problem. However, how to apply machine learning to hardware Trojan detection has not been studied so far. This dissertation proposes effective feature values to apply machine learning to hardware Trojan detection, and empirically demonstrates machine-learning-based hardware-Trojan detection methods. The proposed methods are divided into two types in terms of hardware production steps: the design step and the production step.

First, we focus on the hardware-Trojan detection method in the design step. Most of the existing methods take model-based approaches, and therefore detectability of unknown threats has to be discussed. Moreover, due to the rapid development in hardware industries, a large variety of hardware devices are developed. Machine learning is an effective approach to tackle the above problem. However, how to apply a machine leaning algorithm to hardware-Trojan detection has not been discussed yet. Our proposed methods in Chapter 2 extract effective feature values from hardware design information written in hardware description language, and learn the extracted feature values utilizing machine learning algorithms. Furthermore, by proposing application methods in Chapter 3, we enhance the machine-learning-based hardware-Trojan detection methods and bring them closer to practical use. This dissertation hereby breaks new ground in the

field of hardware Trojan detection in the design step from the viewpoint of proposing a machine-learning-based methodology and its applications.

Second, we focus on the hardware-Trojan detection method in the manufacturing step. Existing hardware-Trojan detection methods utilizing side-channel information often refer to the Golden model. However, preparing the Golden model is unrealistic in the real-world hardware production process. Our proposed method in Chapter 4 detects abnormal behavior on a micro-controller based on power analysis without the Golden model. In Chapter 4, we extract feature values based on power profile of a hardware device using a micro-controller, and successfully apply an abnormal detection algorithm to detect malicious behaviors. This dissertation here establishes a new methodology to detect abnormal behavior based on power analysis.

Throughout this dissertation, we have overcome the hardware-security issues by utilizing machine learning with hardware-specific features.

The chapters in this dissertation are summarized as follows.

In **Chapter 2**, a hardware-Trojan classification method at gate-level netlists utilizing a support vector machine (SVM) or a neural network (NN) is proposed. First, the *five hardware-Trojan features* are extracted from each net in a netlist. These feature values are complicated so that we cannot give the simple and fixed threshold values to them. Hence we secondly represent them to be a *five-dimensional vector* and *learn* them by using SVM or NN. Finally, we can successfully classify all the nets in an unknown netlist into Trojan ones and normal ones based on the learned classifiers. We have applied our machine-learning-based hardware-Trojan classification method to Trust-HUB benchmarks. The results demonstrate that our method increases the true positive rate compared to the existing state-of-the-art results in most of the cases. In some cases, our method can achieve the true positive rate of 100%, which shows that all the Trojan nets in an unknown netlist are completely detected by our method. After that, effective Trojan-net features for supervised machine-learning-based hardware-Trojan detection and their application to a random forest classifier are proposed. We first propose 51 Trojan-net features which describe well Trojan nets. After that, we pick up random forest as one of the best candidates for machine learning and optimize it to apply to hardware-Trojan detection. Based on the importance values obtained from the optimized random forest classifier, we extract the best set of 11 Trojan-net features out of the 51 features which can effectively classify the nets into Trojan ones and normal ones, maximizing the F-measures. By using the 11 Trojan-net features extracted, our optimized random forest classifier has achieved at most 100% true positive rate as well as 100% true negative rate in several Trust-HUB benchmarks and obtained the average F-measure of 79.3% and the accuracy of 99.2%, which realize the *best values* among existing machine-learning-based hardware-Trojan detection methods.

In **Chapter 3**, three applications of machine-learning-based hardware-Trojan detec-

tion are proposed. First, we propose a machine-learning-based hardware-Trojan detection method for gate-level netlists using multi-layer neural networks. We classify the nets in an unknown netlist into a set of Trojan nets and that of normal nets using multi-layer neural networks based on 11 Trojan-net features proposed in Chapter 2. By experimentally optimizing the structure of multi-layer neural networks, we can obtain an average of 84.8% true positive rate and an average of 70.1% true negative rate while we can obtain 100% true positive rate in some of the benchmarks, which outperforms the existing methods in most of the cases. Second, we propose a Trojan-invalidating circuit, and implement it on an FPGA board. The implementation results demonstrate that the implemented Trojan-invalidating circuit successfully prevent from activating a hardware Trojan. Third, we propose a reinforcement of the hardware-Trojan detection utilizing machine learning. Since existing machine-learning-based hardware-Trojan detection methods are performed in the feature spaces, the proposed method considers boundary net structures between normal nets and Trojan nets and compensates the first machine-learning-based detection results based on them. The experimental results demonstrate that our proposed method successfully improve the detection results compared to the existing method.

In **Chapter 4**, an anomaly behavior detection method utilizing power analysis for low-cost micro-controllers is proposed. Our method accurately measures power consumption of the target device, and then classifies its waveform into the sleep-mode part, in which a micro-controller saves power, and into the active-mode part, in which a micro-controller works in a normal operation. After that, we obtain the duration time and consumed power from each active-mode period as feature values. Finally, we detect abnormal behavior based on the obtained feature values utilizing an outlier detection method. In our experiments, we empirically evaluate the proposed method utilizing two types of micro-controllers, and the experimental results demonstrate that our proposed method successfully detects abnormal behaviors.

In conclusion, we find out that hardware Trojan detection utilizing machine learning based on hardware-specific features has future prospect. However, there still remain several tasks to be done. Our future works are summarized as follows:

- Enhance the classification performance of hardware-Trojan detection methods utilizing machine learning algorithms.
- Evaluate the proposed methods utilizing real-world devices.
- Put the proposed methods into practical use.

First, we aim to enhance the classification performance of hardware Trojan detection so that the TPR and TNR are sufficiently increased. Second, we aim to utilize the real-world datasets to evaluate the machine-learning-based hardware-Trojan detection and to implement the detection to the real-world IC design and production steps. Finally, we would like to work on putting the proposed methods into practical use.

Acknowledgment

First and foremost, I would like to express the deepest appreciation and heartfelt thanks to Prof. Nozomu Togawa (戸川望教授) at the Department of Computer Science and Communications Engineering of Waseda University for his throughout supports on my research work for almost five years. He has taught me not only my research field but also how to handle a various of jobs flawlessly.

I am deeply grateful to Prof. Kazuo Hashimoto (橋本和夫教授) at the Research Innovation Center of Waseda University, Prof. Masao Yanagisawa (柳澤政生教授) at the Department of Electronic and Physical Systems of Waseda University, and Prof. Tatsuya Mori (森達哉教授) at the Department of Computer Science and Communications Engineering of Waseda University for their strong support and valuable advice.

I have received valuable comments from Prof. Youhua Shi (史又華教授) at the Department of Electronic and Physical Systems of Waseda University. I would like to offer my special thanks to Dr. Masashi Tawada (多和田雅師講師) and Dr. Kazushi Kawamura (川村一志講師) at the Department of Communications Engineering of Waseda University for their helpful technical supports. Dr. Masaru Oya (大屋優氏) at Waseda University provided me with the opportunity to start my research life and technical supports on hardware Trojan detection.

I would like to offer my special thanks to Ms. Shuko Watanabe (渡部周子氏) for the support of my research life. I have had fulfilling time at Togawa Laboratory with my colleagues Ms. Siya Bao (鮑思雅氏), Mr. Daisuke Oku (於久太祐氏), and Mr. Ryota Ishikawa (石川遼太氏). I also thank all the students in the Togawa Laboratory and the Information System Laboratory.

Dr. Shinsaku Kiyomoto (清本晋作氏) and Dr. Seira Hidano (披田野清良氏) at KDDI Research Inc. have provided me with highly sophisticated advice on machine learning.

Mr. Ryoichi Kida (木田良一氏) at LAC Co., Ltd. has also instructed me on power analysis for IoT devices.

Mr. Kiyoshi Chikamatsu (近松聖氏) and Mr. Naoki Kobayashi (小林直紀氏) at Keysight Technologies Japan G.K. have instructed me on power analysis utilizing measurement instruments.

Mr. Kuniaki Ito (伊藤晋朗氏), Dr. Chikaaki Kodama (児玉親亮氏), Mr. Masaomi

Teranishi (寺西正臣氏), and Mr. Ryo Yonezawa (米澤遼氏) all at KIOXIA Corporation have supported me on technical challenges.

I have had a fantastic time with my colleagues, especially Mr. Ryota Iwanaji (岩名地良太氏) at Denso Corporation, Mr. Tomoaki Ogiyama (扇山友彰氏) at System Integrator Corporation, Mr. Keisuke Kono (河野圭亮氏) at Panasonic Corporation, Mr. Hiroyuki Kakinuma (柿沼博幸氏) at Panasonic Corporation, Mr. Tomoki Manabe (真鍋知樹氏) at PwC Consulting LLC, Mr. Yoshiyuki Muramatsu (村松和侑氏) at JR East Information Systems Company, and Mr. Ryoya Yano (矢野椋也氏) at NEC Corporation.

I have had a great time with Mr. Hirotaka Tamura (田村裕高氏) at Yahoo Japan Corporation and Mr. Tadahiro Noguchi (野口直寛氏) at Abema TV, Inc.

My parents always support my daily life, and my grandparents encourage me.

Finally, this dissertation has been supported partially by JSPS KAKENHI Grant-in-Aid for JSPS Fellows, MIC/SCOPE #171503005, and the cooperation of organization between Waseda University and KIOXIA Corporation (former Toshiba Memory Corporation).

References

[1] "Trust-HUB." http://www.trust-hub.org

[2] B. Cakir and S. Malik, "Hardware Trojan detection for gate-level ICs using signal correlation based clustering," in *Proc. Design, Automation and Test in Europe (DATE)*, 2015, pp. 471–476.

[3] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: identification of stealthy malicious logic using boolean functional analysis," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (ACM-CCS)*, 2013, pp. 697–708.

[4] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists based on machine learning," in *Proc. IEEE Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016, pp. 203–206.

[5] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier," in *Proc. International Symposium on Circuits and Systems*, 2017, pp. 2154–2157.

[6] "Industrie 4.0." https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html

[7] "Society 5.0." https://www.gov-online.go.jp/cam/s5/eng/

[8] "Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper." https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[9] S. Adee, "The hunt for the kill switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[10] J. Francq and F. Frick, "Introduction to hardware Trojan detection methods," in *Proc. Design, Automation and Test in Europe (DATE)*, 2015, pp. 770–775.

[11] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, pp. 1–23, 2016.

[12] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans," in *Proc. International On-Line Testing Symposium (IOLTS)*, 2014, pp. 49–54.

[13] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. International Symposium on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.

[14] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proc. ACM conference on Computer and communications security (CCS)*. ACM Press, 2002, p. 148.

[15] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2007, p. 9.

[16] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 270–271.

[17] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu, "Physical Design Obfuscation of Hardware: A Comprehensive Investigation of Device and Logic-Level Techniques," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 64–77, 2017.

[18] C. Bao, D. Forte, and A. Srivastava, "On reverse engineering-based hardware Trojan detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 49–57, 2016.

[19] A. Kulkarni, Y. Pino, and T. Mohsenin, "SVM-based real-time hardware Trojan detection for many-core platform," in *Proc. International Symposium on Quality Electronic Design (ISQED)*, 2016, pp. 362–367.

[20] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, "Detection technique for hardware Trojans using machine learning in frequency domain," in *Proc. Global Conference on Consumer Electronics (GCCE)*, 2015, pp. 185–186.

[21] J. Zhang, F. Yuan, and Q. Xu, "DeTrust: defeating hardware trust verification with stealthy implicitly-triggered hardware Trojans," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (ACM-CCS)*, 2014, pp. 153–166.

[22] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa, "A score-based classification method for identifying hardware-trojans at gate-level netlists," in *Proc. Design, Automation and Test in Europe (DATE)*, 2015, pp. 465–470.

[23] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware Trojans in third-party digital IP cores," in *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2011, pp. 67–70.

[24] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: detecting and removing malicious hardware automatically," in *Proc. Symposium on Security and Privacy (SP)*, 2010, pp. 159–172.

[25] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] J. B. Wendt, F. Koushanfar, and M. Potkonjak, "Techniques for foundry identification," in *Proc. Design Automation Conference (DAC)*, 2014, pp. 1–6.

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

[28] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *27th European Conference on IR Research, ECIR 2005*, vol. 3408, 2005, pp. 345–359.

[29] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[30] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," 2003. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[31] D. E. Rumelhart, J. L. McClelland, and P. R. Group, *Parallel distributed processing*. The MIT Press, 1986, vol. 1.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[33] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: A methodology review," *Journal of Biomedical Informatics*, vol. 35, no. 5-6, pp. 352–359, 2002.

[34] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "PyBrain," *The Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. International Joint Conference on Artificial Intelligence*, vol. 2, 1995, pp. 1137–1143.

[37] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[38] M. Oya, N. Yamashita, T. Okamura, Y. Tsunoo, M. Yanagisawa, and N. Togawa, "Hardware-Trojans rank: quantitative evaluation of security threats at gate-level netlists by pattern matching," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99.A, no. 12, pp. 2335–2347, 2016.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1–9, 2012.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, pp. 1–9, 2013.

[41] S. Ding, "Feature selection based F-score and ACO algorithm in support vector machine," in *Proc. International Symposium on Knowledge Acquisition and Modeling*, vol. 1, 2009, pp. 19–23.

[42] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer New York, 2013, vol. 103.

[43] E. Pitler and A. Nenkova, "Using syntax to disambiguate explicit discourse connectives in text," in *ACL-IJCNLP*, 2009, pp. 13–16.

[44] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists using multi-layer neural networks," in *Proc. IEEE Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017, pp. 227–232.

[45] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[46] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[47] T. Zahavy, A. Magnani, A. Krishnan, and S. Mannor, "Is a picture worth a thousand words? A deep multi-modal fusion architecture for product classification in e-commerce," pp. 1–9, 2016. http://arxiv.org/abs/1611.09534

[48] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: a next-generation open source framework for deep learning," in *Proc. Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[49] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojan identification based on netlist features using neural networks," *IEICE Technical Report*, vol. 116, no. 93, CAS2016-1, pp. 1–6, 2016.

[50] K. Hasegawa, M. Yanagisawa, and N. Togawa, "A hardware-Trojan classification method using machine learning at gate-level netlists based on Trojan features," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100-A, no. 7, pp. 1427–1438, 2017.

[51] S. Mal-Sarkar, A. Krishna, A. Ghosh, and S. Bhunia, "Hardware Trojan attacks in FPGA devices," in *Proc. Great Lakes Symposium on VLSI (GLSVLSI)*, 2014, pp. 287–292.

[52] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "FPGA Trojans through detecting and weakening of cryptographic primitives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1236–1249, 2015.

[53] M. Lecomte, J. Fournier, and P. Maurine, "An on-chip technique to detect hardware Trojans and assist counterfeit identification," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–14, 2016.

[54] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa, "In-situ Trojan authentication for invalidating hardware-Trojan functions," in *Proc. International Symposium on Quality Electronic Design (ISQED)*, 2016, pp. 152–157.

[55] National Institute of Standards and Technology (NIST), "Announcing the advanced encryption standard (AES)," Federal information Processing Standards Publication 197, 2001.

[56] "OpenCores," http://opencores.org/.

[57] R. Shende and D. D. Ambawade, "A side channel based power analysis technique for hardware trojan detection using statistical learning approach," in *Proc. International Conference on Wireless and Optical Communications Networks (WOCN)*, 2016, pp. 1–4.

[58] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Detecting the existence of malfunctions in microcontrollers utilizing power analysis," in *Proc. IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2018, pp. 97–102.

[59] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *Proc. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57.

[60] S. Wang, X. Dong, K. Sun, Q. Cui, D. Li, and C. He, "Hardware Trojan detection based on ELM neural network," in *Proc. IEEE International Conference on Computer Communication and the Internet (ICCCI)*, 2016, pp. 400–403.

[61] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a side channel based disassembler," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6340, no. PART 1, pp. 78–99.

[62] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1019–1031.

[63] P. Saravanan, N. Rajadurai, and P. Kalpana, "Power analysis attack on 8051 microcontrollers," in *Proc. IEEE International Conference on Computational Intelligence and Computing Research*, 2014, pp. 1–4.

[64] Keysight Technologies, "Evaluating current probe technologies for low-power measurements." http://literature.cdn.keysight.com/litweb/pdf/5991-4375EN.pdf

[65] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *Proc. Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14, 1967, pp. 281–297.

[66] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *Proc. ACM SIGMOD International Conference on Management of Data*, 2000, pp. 93–104.

[67] Tektronix, "Keithley 2280 Precision Measurement DC Power Supplies." https://www.tek.com/tektronix-and-keithley-dc-power-supplies/keithley-2280s-series

[68] Keysight Technologies, "CX3324A Device Current Waveform Analyzer." https://www.keysight.com/en/pd-2657157-pn-CX3324A/device-current-waveform-analyzer-1-gsa-s-14-16-bit-4-channel

[69] Keysight Technologies, "CX3300 Series Device Current Waveform Analyzer Datasheet." https://literature.cdn.keysight.com/litweb/pdf/5992-1430EN.pdf?id=2727780

[70] K. Chikamatsu, "Current sensing circuit," U.S. Patent 9 689 900, 2017.

[71] "Arduino." https://www.arduino.cc/

[72] STMicroelectronics, "Nucleo-L476RG." https://www.st.com/en/evaluation-tools/nucleo-l476rg.html

[73] "AESLib." https://github.com/DavyLandman/AESLib

List of Publications

Peer review journal articles

⟨1⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "Empirical Evaluation and Optimization of Hardware-Trojan Classification for Gate-Level Netlists based on Multi-Layer Neural Networks," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science*, vol. E101-A, no. 12, pp. 2320-2326, Dec. 2018.

⟨2⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "Trojan-net Feature Extraction and Its Application to Hardware-Trojan Detection for Gate-level Netlists Using Random Forest," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science*, vol. E100-A, no. 12, pp. 2857–2868, Dec. 2017.

⟨3⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "A Hardware-Trojan Classification Method Using Machine Learning at Gate-level Netlists based on Trojan Features," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100-A, no. 7, pp. 1427–1438, Jul. 2017.

Peer review conference papers

⟨4⟩ **K. Hasegawa**, R. Ishikawa, M. Nishizawa, K. Kawamura, M. Tawada, and N. Togawa, "FPGA-based Heterogeneous Solver for Three-Dimensional Routing," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, Beijing, China, Jan. 2020.

⟨5⟩ **K. Hasegawa**, K. Takasaki, M. Nishizawa, R. Ishikawa, K. Kawamura, and N. Togawa, "Implementation of a ROS-Based Autonomous Vehicle on an FPGA Board," in *Proc. International Conference on Field-Programmable Technology (FPT)*, pp. 457–460, Tianjin, China, Dec. 2019.

⟨6⟩ K. Nozawa, **K. Hasegawa**, S. Hidano, S. Kiyomoto, K. Hashimoto, and N. Togawa, "Adversarial Examples for Hardware-Trojan Detection at Gate-Level Netlists," in

Proc. International Workshop on Attacks and Defenses for Internet-of-Things (ADIoT), Luxembourg, Luxembourg, Sep. 2019.

⟨7⟩ ○ **K. Hasegawa**, K. Chikamatsu, and N. Togawa, "Empirical Evaluation on Anomaly Behavior Detection for Low-Cost Micro-Controllers Utilizing Accurate Power Analysis," in *Proc. IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 54-57, Rhodes Island, Greece, July 2019.

⟨8⟩ M. Nishizawa, **K. Hasegawa**, and N. Togawa, "Capacitance Measurement of Running Hardware Devices and its Application to Malicious Modification Detection," in *Proc. IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 362-365, Chengdu, China, Oct. 2018.

⟨9⟩ T. Inoue, **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "Designing Subspecies of Hardware Trojans and Their Detection Using Neural Network Approach," in *Proc. IEEE International Conference on Consumer Electronics in Berlin (ICCE-Berlin)*, pp. 156-159, Berlin, Germany, Sep. 2018.

⟨10⟩ ○ (Invited talk) **K. Hasegawa**, Y. Shi, and N. Togawa, "Hardware Trojan Detection Utilizing Machine Learning Approaches," in *Proc. IEEE International Workshop on Hardware Security and Trust (HSAT)*, pp. 1891-1896, Elizabeth, NJ, USA, Aug. 2018.

⟨11⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "Detecting the Existence of Malfunctions in Microcontrollers Utilizing Power Analysis," in *Proc. IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 97-102, Platja d'Aro, Spain, July 2018.

⟨12⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "A Trojan-invalidating Circuit Based on Signal Transitions and Its FPGA Implementation," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5, Florence, Italy, May 2018.

⟨13⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "A Hardware-Trojan Classification Method Utilizing Boundary Net Structures," in *Proc. IEEE International Conference on Consumer Electronics (ICCE)*, pp. 103-106, Las Vegas, NV, USA, Jan. 2018.

⟨14⟩ T. Inoue, **K. Hasegawa**, M. Yanagisawa, and N. Togawa, "Designing Hardware Trojans and Their Detection based on a SVM-based Approach," in *Proc. IEEE International Conference on ASIC (ASICON)*, pp. 811–814, Guiyang, China, Oct. 2017.

⟨15⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, “Hardware Trojans Classification for Gate-level Netlists Using Multi-layer Neural Networks,” in *Proc. IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 227–232, Thessaloniki, Greece, July 2017.

⟨16⟩ ○ **K. Hasegawa**, M. Yanagisawa, and N. Togawa, “Trojan-feature Extraction at Gate-level Netlists and Its Application to Hardware-Trojan Detection Using Random Forest Classifier,” in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2154–2157, Baltimore, MD, USA, May 2017.

⟨17⟩ ○ **K. Hasegawa**, M. Oya, M. Yanagisawa, and N. Togawa, “Hardware Trojans Classification for Gate-level Netlists based on Machine Learning,” in *Proc. IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 203–206, Sant Feliu de Guixols, Spain, July 2016.

国内学会

⟨18⟩ 西澤 誠人, 長谷川 健人, 戸川 望, “非正規挿入デバイス検知のための電気容量監視装置とその実験的評価,” 信学技報, vol. 119, no. 260, HWS2019-66, pp. 53–58, 大阪市, Nov. 2019.

⟨19⟩ 野澤 康平, 長谷川 健人, 披田野 清良, 清本 晋作, 橋本 和夫, 戸川 望, “ニューラルネットワークを用いたハードウェアトロイ識別に対する敵対的サンプル攻撃の実証評価,” 信学技報, vol. 119, no. 260, HWS2019-64, pp. 41–46, 大阪市, Nov. 2019.

⟨20⟩ 長谷川 健人, 戸川 望, “IoT デバイス管理基盤の一考察,” 電子情報通信学会ソサイエティ大会, p. 139, 豊中市, Sep. 2019.

⟨21⟩ (査読あり) 長谷川 健人, 近松 聖, 戸川 望, “マイクロコントローラのスリープ状態に着目した消費電力にもとづく悪意のある機能の発現検知,” 情報処理学会 DA シンポジウム 2019 論文集, pp. 93-98, 加賀市, Aug. 2019.

⟨22⟩ (ポスター発表) 西澤 誠人, 石川 遼太, 長谷川 健人, 川村 一志, 多和田 雅師, 戸川 望, “配置配線のためのアンサンブルソルバシステム,” 情報処理学会 DA シンポジウム 2019 ポスター発表, 加賀市, Aug. 2019.

⟨23⟩ (招待講演) 長谷川 健人, 野澤 康平, 披田野 清良, 清本 晋作, 橋本 和夫, 戸川 望, “ハードウェアセキュリティにおける AI 活用と攻撃,” 電子情報通信学会総合大会, pp. SS-58-59, 新宿区, Mar. 2019.

⟨24⟩ 井上 智貴, 長谷川 健人, 戸川 望, “周辺ネットの特徴量を考慮した二段階のニューラルネットワークによるハードウェアトロイ検出手法,” 情報処理学会研究報告, 西之表市, Mar. 2019.

⟨25⟩ 野澤 康平, 長谷川 健人, 披田野 清良, 清本 晋作, 橋本 和夫, 戸川 望, “ニューラルネットワークを用いたハードウェアトロイ識別に対する敵対的サンプル攻撃に関する一考察,” 暗号と情報セキュリティシンポジウム予稿集, 大津市, Jan. 2019.

⟨26⟩ (査読あり) 西澤 誠人, 長谷川 健人, 柳澤 政生, 戸川 望, “低電力化電気容量検出装置を用いた動作中の不正デバイス検知,” 情報処理学会 DA シンポジウム 2018 論文集, pp. 112-117, 加賀市, Aug. 2018.

⟨27⟩ (査読あり) 長谷川 健人, 柳澤 政生, 戸川 望, “マイクロコントローラのスリープ状態に着目した消費電力にもとづく悪意のある機能の発現検知,” 情報処理学会 DA シンポジウム 2018 論文集, pp. 118-123, 加賀市, Aug. 2018.

⟨28⟩ (ポスター発表) 石川 遼太, 西澤 誠人, 長谷川 健人, 川村 一志, 多和田 雅師, 戸川 望, “ナンバーリンクソルバのための FPGA 協調システム,” 情報処理学会 DA シンポジウム 2018 ポスター発表, 加賀市, Aug. 2018.

⟨29⟩ 井上 智貴, 長谷川 健人, 柳澤 政生, 戸川 望, “亜種ハードウェアトロイの設計とそのニューラルネットワークを用いた検出,” 信学技報, vol. 118, no. 83, VLD2018-36, pp. 173-178, 札幌市, Jun. 2018.

⟨30⟩ 長谷川 健人, 柳澤 政生, 戸川 望, “暗号回路に挿入されたハードウェアトロイとその抑止回路の FPGA 実装,” 信学技報, vol. 117, no. 273, VLD2017-53, pp. 139–144, 熊本市, Nov. 2017.

⟨31⟩ 井上 智貴, 長谷川 健人, 柳澤 政生, 戸川 望, “トリガ条件の異なるハードウェアトロイの設計と SVM を用いた検出,” 信学技報, vol. 117, no. 273, VLD2017-51, pp. 133–138, 熊本市, Nov. 2017.

⟨32⟩ (査読あり) 長谷川 健人, 柳澤 政生, 戸川 望, “ネットの周辺情報を考慮した機械学習によるハードウェアトロイ識別,” 情報処理学会 DA シンポジウム 2017 論文集, pp. 127–132, 加賀市, Aug. 2017.

⟨33⟩ (ポスター発表) 長谷川 健人, 石川 遼太, 寺田 晃太朗, 川村 一志, 多和田 雅師, 戸川 望, “組込みデバイスと FPGA を用いたナンバーリンクソルバの設計と実装,” DA シンポジウム 2017, 加賀市, Aug. 2017.

⟨34⟩ 長谷川 健人, 柳澤 政生, 戸川 望, “ネットの特徴量を用いた多層ニューラルネットワークによるハードウェアトロイ識別,” 情報処理学会研究報告, Vol. 2017-SLDM-179, No. 23, pp. 1–6, 久米島町, Mar. 2017.

⟨35⟩ (査読あり) 長谷川 健人, 柳澤 政生, 戸川 望, “Random Forest を用いたネットリスト特徴選択と機械学習によるハードウェアトロイ識別,” 情報処理学会 DA シンポジウム 2016 論文集, pp. 8–13, 加賀市, Sep. 2016.

⟨36⟩ (ポスター発表) 寺田 晃太朗, 長谷川 健人, 川村 一志, 多和田 雅師, 戸川 望, “機械学習と FPGA を用いたナンバーリンクソルバ,” DA シンポジウム 2016, 加賀市, Sep. 2016.

⟨37⟩ 長谷川 健人, 柳澤 政生, 戸川 望, “ニューラルネットを利用したネットリストの特徴にもとづくハードウェアトロイ識別，” 信学技報, vol. 116, no. 94, VLD2016-7, pp. 1–6, 弘前市, Jun. 2016.

⟨38⟩ 長谷川 健人, 大屋 優, 柳澤 政生, 戸川 望, “SVM を利用したネットリストの特徴に基づくハードウェアトロイ識別,” 信学技報, vol. 115, no. 338, VLD2015-58, pp. 135–140, 長崎市, Dec. 2015.

雑誌記事

⟨39⟩ 長谷川 健人, 戸川 望, “スパイチップはあるのか - ハードウェアセキュリティの必要性,” 情報処理, vol. 60, no. 1, pp. 4–6, Jan. 2019.

⟨40⟩ 川村 一志, 長谷川 健人, 多和田 雅師, 戸川 望, “機械学習と FPGA を用いた配線問題解法への取り組み,” 情報処理, vol. 59, no. 3, pp. 228–231, Mar. 2018.

業績賞等

⟨41⟩ DA シンポジウム アルゴリズムデザインコンテスト 特別賞, DA シンポジウム 2019, Aug. 2019.

⟨42⟩ DA シンポジウム アルゴリズムデザインコンテスト 特別賞, DA シンポジウム 2018, Aug. 2018.

⟨43⟩ 第 33 回電気通信普及財団賞（テレコムシステム技術学生賞）最優秀賞, Mar. 2018.

⟨44⟩ 情報処理学会 山下記念研究賞, Mar. 2018.

⟨45⟩ IEEE CEDA All Japan Chapter Academic Research Award, Nov. 2017.

⟨46⟩ DA シンポジウム アルゴリズムデザインコンテスト 最優秀賞, DA シンポジウム 2017, Aug. 2017.

⟨47⟩ IEEE CEDA All Japan Chapter Academic Research Award, Aug. 2017.

⟨48⟩ 情報処理学会 SLDM 研究会優秀発表学生賞, 情報処理学会 ETNET 2017, Aug. 2017.

⟨49⟩ 早稲田大学基幹理工学研究科情報理工・情報通信専攻 専攻賞, Mar. 2017.

⟨50⟩ 大川功情報通信学術奨学金, Mar. 2017.

⟨51⟩ DA シンポジウム アルゴリズムデザインコンテスト 最優秀賞, DA シンポジウム 2016, Sep. 2016.

⟨52⟩ DA シンポジウム アルゴリズムデザインコンテスト 特別賞, DA シンポジウム 2016, Sep. 2016.

⟨53⟩ 情報処理学会 SLDM 研究会優秀発表学生賞, 情報処理学会 デザインガイア 2015, Sep. 2016.

⟨54⟩ 小野梓記念奨学金, July 2016.

⟨55⟩ 三菱 UFJ 信託奨学財団奨学金, Apr. 2016.

⟨56⟩ 早稲田大学基幹理工学部 学部長賞 優秀賞, Mar. 2016.

研究費・助成金

⟨57⟩ 早稲田大学理工学術院総合研究所 東芝メモリ 若手奨励研究, “FlashAir とブロックチェーンを用いたハードウェアデバイスとデータ認証,” Sep. 2018–Feb. 2020, 総額 50 万円.

⟨58⟩ 日本学術振興会 特別研究員奨励費, “IoT 時代における機械学習を用いたハードウェアトロイ識別に関する研究,” Apr. 2018–Mar. 2020, 総額 210 万円 (2018 年度: 110 万円, 2019 年度: 100 万円).

⟨59⟩ 電気通信普及財団, 平成 29 年度 6 月期海外渡航旅費援助, Jun. 2017, 総額 29 万円.

特許

⟨60⟩ (発明者) 披田野 清良, 清本 晋作, 長谷川 健人, 戸川 望, (出願人) KDDI 株式会社, 学校法人早稲田大学, “学習装置、学習方法及び学習プログラム”, 特願 2019-140107, (出願日) 2019 年 7 月 30 日.

⟨61⟩ (発明者) 戸川 望, 長谷川 健人, (出願人) 学校法人早稲田大学, “検出方法及び検出装置,” 特願 2018-113649, (出願日) 2018 年 6 月 14 日.