Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

# Generating Adversarial Examples for Hardware-Trojan Detection at Gate-Level Netlists

Kohei Nozawa[1,a) ] Kento Hasegawa[1] Seira Hidano[2] Shinsaku Kiyomoto[2]
Kazuo Hashimoto[3] Nozomu Togawa[1]

Received: June 29, 2020, Accepted: December 1, 2020

**Abstract:** Recently, the great demand for integrated circuits (ICs) drives third parties to be involved in IC design and manufacturing steps. At the same time, the threat of injecting a malicious circuit, called a hardware Trojan, by third parties has been increasing. Machine learning is one of the powerful solutions for detecting hardware Trojans. However, a weakness of such a machine-learning-based classification method against adversarial examples (AEs) has been reported, which causes misclassification by adding perturbation in input samples. This paper firstly proposes a framework generating adversarial examples for hardware-Trojan detection at gate-level netlists utilizing neural networks. The proposed framework replaces hardware Trojan circuits with logically equivalent ones, and makes it difficult to detect them. Secondly, we propose a Trojan-net concealment degree (TCD) and a modification evaluating value (MEV) as measures of the amount of modifications. Finally, based on the MEV, we pick up adversarial modification patterns to apply to the circuits against hardware-Trojan detection. The experimental results using benchmarks demonstrate that the proposed framework successfully decreases the true positive rate (TPR) by a maximum of 30.15 points.

**Keywords:** hardware Trojan, netlist, logic gate, machine learning, adversarial example

## 1. Introduction

### 1.1 Hardware Trojans and Its Detection Methods

The demand for integrated circuits (ICs) has recently been increasing due to the introduction of Internet of Things (IoT) in our daily lives. In order to effectively design and produce hardware devices at low cost, has resulted in them becoming more complicated. The hardware design and production process can be divided into two steps: the design step and the manufacturing step. In the design step, hardware vendors design IC chips according to product specifications, and describe the design in hardware description language (HDL). This step is often outsourced to third-party vendors [1]. A frequently-used circuit such as a processor or communication interface is packed as a module called an intellectual property core (IP core), and a hardware vendor often purchases it from a third-party vendor. In the manufacturing step, a circuit is manufactured at a foundry based on the information designed in the design step. As above, hardware design and manufacturing steps involve several third-party vendors.

At the same time, it is reported that an adversary including an untrusted third-party vendor may modify a hardware design or product with malicious intent in the design or manufacturing step [2], [3]. A malicious circuit inserted into a genuine product is called a "hardware Trojan" [4]. A hardware Trojan often leaks internal information, degrades performance, and/or deactivates functionalities. Most IoT devices may be infected with hardware Trojans. This is an emerging threat since IoT devices are spreading to our home.

Now we focus on the hardware design step. Detecting hardware Trojans in the design step, the first step, can be more effective. There are two reasons: Firstly, attackers are easily able to insert hardware Trojans in the design step. To insert hardware Trojans in the design step, all attackers need to do is just to manipulate design information. Design information is usually written in HDL, which is human-readable, and hence inserting hardware Trojans in this step must be very easy. By adding a few lines into HDL, attackers can insert hardware Trojans [5]. However, in the manufacturing step, attackers need to consider wiring, timing, and clock distribution to insert hardware Trojans. From the viewpoint of attackers, attackers can very easily insert hardware Trojans in the design step. At the same time, after the design step, defenders may lose sight of the hardware Trojans from the complicated circuit structures. Secondly, detecting in the manufacturing step is time-consuming and cost-consuming. If manufactured IC products turn out to be infected by hardware Trojans, vendors need to remake ICs from the very first step. Vendors also have to throw infected products away and need new resources. Therefore, detecting hardware Trojans in the design step can be more important and logical.

In the hardware design step, a hardware design is often written in gate level which describes how to connect circuit elements with wires (also called nets). A gate-level netlist, which is a list of nets and circuit elements, includes so many nets, and there-

[1] Dept. Computer Science and Communications Engineering, Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] KDDI Research, Inc., Fujimino, Saitama 356–8502, Japan
[3] Research Innovation Center, Waseda University, Shinjuku, Tokyo 169–8050, Japan
[a)] kohei.nozawa@togawa.cs.waseda.ac.jp

fore it is difficult to inspect each net in detail. How to detect a hardware Trojan from such a huge gate-level netlist is a serious concern. In Ref. [6], a hardware-Trojan detection method at gate-level netlists based on the structural features is firstly proposed and it is extended in Refs. [7], [8], [9] utilizing machine learning. In particular, a hardware-Trojan detection method utilizing neural networks is expected to detect subspecies of hardware Trojans effectively [10].

### 1.2 Attacks on Detection Methods

In the field of machine learning, an attack which makes a classifier erroneously classify a given sample has been proposed [11], [12]. According to Ref. [11], a test sample with a certain noise, which is called perturbation, will be classified into a different label from the original one. Such a test sample with perturbation is named an *adversarial example* (AE), and we call the attack utilizing it an *AE attack*. A number of AE attack schemes have been proposed [13], and mitigating the problem of AE attacks becomes a serious concern. While initial studies related to AEs started with image recognition, some recent ones focus on different use cases. For instance, there is an attempt to generate AEs for object recognition [14]. An AE attack method against a malware detection system has also been proposed [15].

However, existing hardware-Trojan detection methods utilizing neural networks [7], [16] have been designed with no consideration for advanced attacks such as AE attacks. Thus, they will also be exposed to the risk of AE attacks. An AE attack to hardware-Trojan detection will deceive a classifier and a hardware-Trojan circuit will be misclassified as normal. If the attack is realized, hardware-Trojan detection becomes more difficult. In order to put hardware-Trojan detection to practical use, it is necessary to clarify its potential risks against AE attacks. Before studying the defense against AE attacks, we discuss how to realize the AE attacks to machine-learning-based hardware-Trojan detection at gate-level netlists from the viewpoint of attackers. AEs for image recognition can be generated by calculating perturbation. For circuits, however, there are additional constraints of keeping logical equivalency, i.e., the modified circuits must be logically the same as the original ones. Circuits cannot be modified arbitrarily corresponding to calculated perturbation. Thus, a different approach from existing AE-attack methods is needed to generate AEs for hardware-Trojan detection.

There exists an engineering change order (ECO) in the last step of the design process. Some bugs or changes in circuit specification sometimes occur during the design process. There have been developed several tools [17], [18] to remove the bugs and apply the new constraints to circuits. These approaches alter synthesized circuits, and the impacts of changes in circuits can be significant. However, circuits once designed should be modified only when necessary. Therefore, the previous machine-learning-based hardware-Trojan detection studies [7], [16] utilize the gate-level netlists in Trust-HUB without any modifications for training data. Modification patterns that we introduce in this paper are out of the ordinary cases in circuit design. Thus, samples with the modification can be adversarial examples.

From the viewpoint of IC designers, to strengthen the tolerance of the classifier against AE attacks, it is important to analyze the adversaries' scenarios. We try to analyze the scenarios by introducing a new attack method which can be realized. We can specify the goal that the adversaries want to realize and the capabilities that adversaries have. By investigating the attacks, IC designers will be able take countermeasures against them.

In this paper, we propose a framework generating adversarial examples for hardware-Trojan detection at gate-level netlists utilizing neural networks [*1]. In the proposed method, we firstly propose a Trojan-net concealment degree (TCD), and secondly propose a modification evaluating value (MEV) which can be obtained from a loss function of the neural networks utilized for hardware-Trojan detection. Utilizing TCD and MEV, we generate adversarial examples which cause misclassification in hardware-Trojan detection. The proposed method enables us to generate adversarial examples against hardware-Trojan detection very effectively without exploring all of the modification patterns of a target circuit. As far as we know, this is the first study on adversarial examples against hardware-Trojan detection and its application to the benchmark dataset.

The contributions of this paper are summarized as follows:

(1) We propose a framework that generates an AE against hardware-Trojan detection.
(2) In the proposed framework, we propose TCD and MEV to evaluate the amount of modifications and we can effectively pick up effective modifications on generating an AE based on TCD and MEV.
(3) By applying our method, the true positive rate (TPR) [*2] is decreased by a maximum of 30.15 points, validating modification patterns and MEV.

The rest of this paper is organized as follows: In Section 2, we discuss related works on hardware-Trojan detection utilizing machine learning and AE. In Section 3, we propose a framework generating adversarial examples for hardware-Trojan detection for gate-level netlists utilizing neural networks. We also propose TCD and MEV for evaluating modifications on the original hardware Trojans. In Section 4, we perform experiments and demonstrate their results. In Section 5, we conclude this paper.

## 2. Related Works

In this section, we sum up the backgrounds of hardware Trojans and researches to detect them. Then, we show the fundamentals on adversarial examples.

### 2.1 Hardware Trojan and Its Detection Utilizing Neural Networks

A hardware Trojan is a malicious circuit embedded in ICs. A hardware Trojan often consists of two parts of circuits, a trigger

[*1] The preliminary version of this paper appeared in Ref. [19].
[*2] In a gate-level netlist, a circuit is represented by logic gates and nets. In particular, a circuit with hardware Trojans is composed of a Trojan-free normal circuit and a Trojan circuit. Normal nets refer to the nets in a normal circuit. Trojan nets refer to the nets in a Trojan circuit. Then, TP (true positive) shows the number of Trojan nets identified as Trojan nets by a classifier. FN (false negative) shows the number of Trojan nets identified as normal nets by a classifier. Then the true positive rate is obtained from TP/(TP + FN).

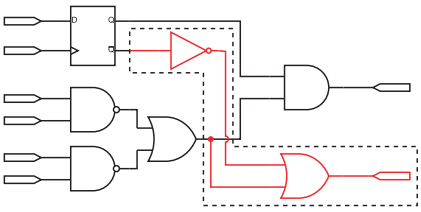

**Fig. 1** Netlist representation by a graph $G = (V, E)$ in which the hardware Trojan represented by a subgraph $G_t = (V_t, E_t)$ is surrounded by a dotted line.

**Table 1** 51 feature values of each net $e$ for hardware-Trojan detection utilizing neural network ($1 \leq x \leq 5$) [16].

| Trojan features | Description |
|---|---|
| fan_in_$x$ | The number of logic-gate fanins $x$-level away from the target net $e$. |
| in_flipflop_$x$ | The number of flip-flops up to $x$-level away from the input side of the target net $e$. |
| out_flipflop_$x$ | The number of flip-flops up to $x$-level away from the output side of the target net $e$. |
| in_multiplexer_$x$ | The number of multiplexers up to $x$-level away from the input side of the target net $e$. |
| out_multiplexer_$x$ | The number of multiplexers up to $x$-level away from the output side of the target net $e$. |
| in_loop_$x$ | The number of up to $x$-level loops from the input side of the target net $e$. |
| out_loop_$x$ | The number of up to $x$-level loops from the output side of the target net $e$. |
| in_const_$x$ | The number of constants up to $x$-level away from the input side of the target net $e$. |
| out_const_$x$ | The number of constants up to $x$-level away from the output side of the target net $e$. |
| in_nearest_pin | The minimum level to the primary input from the target net $e$. |
| out_nearest_pout | The minimum level to the primary output from the target net $e$. |
| {in, out}_nearest_flipflop | The minimum level to any flip-flop from the input/output side of the target net $e$. |
| {in, out}_nearest_multiplexer | The minimum level to any multiplexer from the input/output side of the target net $e$. |

circuit and a payload circuit [20]. A trigger circuit enables a payload circuit when a start-up condition is satisfied. For example, when primary inputs and/or internal states satisfy certain conditions, a trigger circuit wakes up. A payload circuit performs a malicious function such as leakage of information and performance degradation when the trigger condition is satisfied. If hardware Trojans are embedded in home appliances, especially in IoT devices, they can be a familiar threat to us. Since a hardware Trojan is very small compared to a genuine circuit, how to find the features of a hardware Trojan is an important issue.

In general, a gate-level netlist can be represented by a graph structure such as a Boolean network [21]. Let a graph $G = (V, E)$ be a whole gate-level netlist including a Trojan circuit. $V$ is a set of vertexes, or gates. $E$ is a set of edges, or nets. Similarly, let a graph $G_t = (V_t, E_t)$ be a Trojan circuit. $V_t$ is a set of Trojan vertexes, or Trojan gates. $E_t$ is a set of Trojan edges, or Trojan nets. $V_t$ and $E_t$ satisfy $V_t \subseteq V$ and $E_t \subseteq E$, respectively, as shown in **Fig. 1**.

Here we focus on hardware-Trojan detection utilizing machine learning. In Ref. [7], a hardware-Trojan detection method utilizing a neural network has been proposed. The method extracts 5 feature values of each net in a gate-level netlist and identifies whether the net is a Trojan net or a normal net. The subsequent research [22] firstly proposes 51 feature values extracted from gate-level netlists similarly to the method [7] and picks up 11 effective feature values from them. At first, in the learning flow, these methods extract the features from each net $e \in E$, and then obtain the vector of the feature values $\boldsymbol{x}(e) = (x_1, x_2, \ldots)$, where $x_k (1 \leq k)$ shows the feature value. **Table 1** shows the 51 feature values for a net $e$ in a gate-level netlist utilized in Ref. [22]. After that, a neural network learns the extracted feature values. As for each net $e$, we give the feature vector $\boldsymbol{x}(e) = (x_1, x_2, \ldots)$ to the neural network as an input, and then we obtain an output $\boldsymbol{z} = (z_1, z_2)$, where $z_1$ shows the possibility that $e$ is a normal net and $z_2$ shows the possibility that $e$ is a Trojan net. By comparing the output and its answer label, the parameters of the neural network is optimized. Secondly, in the classification flow, the method extracts the features from every net in an unknown netlist and classifies them with the classifier leaned at the learning flow. In the classification flow, the classifier identifies $e$ as a Trojan net when $z_2$ is larger than $z_1$, or identifies $e$ as a normal net when $z_1$ is larger than $z_2$.

Considering the changes in the circuit behavior is an important characteristic of the hardware Trojans and may be useful to detect hardware Trojans [23]. However, these approaches usually require very long-term logic tests and cannot even detect some type of hardware Trojans, particularly the ones triggered very rarely. Recently, the hardware Trojan detection methods based on the static netlist structure were proposed [6] and these methods very successfully detect many types of hardware Trojans including the ones triggered very rarely. The classification method that we use in this paper uses the latter approach. That is why we use the netlist feature approach in adversarial example generation, not taking into circuit behavior.

Since the methods judge whether every net in a circuit is a Trojan net or not, we can know that not only is the circuit infected by hardware Trojans but we can know the detailed positions where the hardware Trojans are inserted. This is one of the main reasons why the methods [7], [16], [22] are powerful at detecting hardware Trojans. There is no need to synthesize the netlist into physical mask patterns. As above, the methods [7], [16], [22] do not consider gate size as a feature value, since we cannot evaluate whether every net is Trojan or not even if the gate size is given. At the same time, though the gate count, or the number of gates included in a given circuit, may help the classifier detect whether the circuit includes hardware Trojans, we cannot know the detailed positions where the Trojans are inserted. Furthermore, we cannot know the correct gate count beforehand for unknown circuits. Note that, the experimental results of Refs. [7], [16], [22] indicate that we can detect Trojan nets accurately without considering the gate size and gate count.

On hardware-Trojan detection utilizing a neural network, it is extremely important to correctly identify Trojan nets as Trojan nets since we want to know all the Trojan nets. Now let $e_{t'}$ be a net classified as a Trojan net, and $E_{t'}$ be a set of the nets classified as Trojan nets. The goal on hardware-Trojan detection is that $e_{t'}$ involves all the Trojan nets in $E_t$. Therefore, maximizing the true positive rate (TPR), detecting as many Trojan nets as possible is the first priority in hardware-Trojan detection. In contrast, attackers aim to minimize TPR, which means detecting as few Trojan nets as possible.

Although the detection method [7] utilizes a neural network with a single middle layer, the detection method [16] utilizes a multi-layer neural network. In the experiment [16], the middle layer in the neural network consists of three layers and the neural network gives good results in hardware-Trojan detection in terms of TPR.

### 2.2 Adversarial Example

Recently, an attack scheme where a certain test sample causes

misclassification to the target classifier utilizing machine learning including a neural network has been proposed [11]. This is an attack for machine learning models which causes misclassification results by adding perturbation to test samples which are originally classified into their correct classes. These samples which cause misclassification are called adversarial examples (AEs) and the attack using AE is called an AE attack. AEs for images are well described in Ref. [12].

Given the fact of the development in machine learning utilizing neural networks, novel attack methods have been proposed. In Ref. [11], for example, a method generating test samples that cause misclassification in image recognition has been proposed. Image recognition is frequently leveraged in the physical world such as in self-driving cars and face recognition, and hence both of the attack methods and defense methods are getting much attention [14], [24], [25]. Although we cannot distinguish adversarial examples from original ones, a classifier misclassifies them. In addition to image recognition, audio recognition [26], sentence recognitio [27], [28] and graph data recognition [29], [30] become new AE targets recently.

In this paper, we propose a framework to generate AEs against hardware-Trojan detection at gate-level netlists utilizing a neural network by modifying hardware designs. In image recognition, AEs are generated with minimizing visible impacts. Likewise, in our case, we aim to generate AEs which hardly degrade circuit performance such as circuit area and path delay. The AEs cause misclassification so that Trojan nets are mistakenly classified as normal nets.

Now we focus on AE attacks on hardware-Trojan detection utilizing neural networks such as in Refs. [7], [16], [22]. In AE attacks against hardware-Trojan detection, an adversary aims to decrease $|E_{t'}|$, i.e., the number of nets classified as Trojan nets. When $|E_{t'}|$ is drastically decreased, most of the Trojan nets are mistakenly classified as normal nets. If such an AE attack is realized, it becomes hard to detect hardware Trojans in a gate-level netlist. In order to learn hardware design information utilizing machine learning methods, we represent hardware circuits as graph structures as described in Section 2.1. However, the conversion from a graph structure into a feature space [31] is one-way. Therefore, even if we add perturbations in the feature space, we cannot easily specify the corresponding changes in the graph structure. In AE attacks against hardware-Trojan detection, we need to take a completely different approach from conventional AE-attack methods.

## 3. AE Attacks on Hardware Design

### 3.1 Scenario of the AE Attacks

Let us consider AE attacks in hardware-Trojan detection utilizing neural networks for gate-level netlists. Now we assume the attack scenario as follows: Hardware Trojans are injected mainly in two different points of the designing process [32]. The first one is the RTL (register-transfer level) description design step. IP cores produced by malicious IP vendors may be embedded in this step. The second one is after logic synthesis. Malicious designing tools made overseas in many cases may inject hardware Trojans into output designing information. As mentioned above, if IP vendors or design tool developers are malicious, circuits can be infected by hardware Trojans.

In the supply chain of the IC design step, we can assume that there are at least two entities; vendor A and vendor B. In designing ICs, vendors are involved to each other [33], [34]. One vendor sometimes becomes an IP core supplier and sometimes becomes an IP core buyer. For example, vendor A designs a product V embedding IP core W designed by vendor B at some point. In this case, vendor B is considered to be a third party for vendor A. At another point, vendor B designs a product X embedding IP core Y designed by vendor A. In this case, vendor A is considered to be a third party for vendor B. In addition, we assume that there exists a standard hardware-Trojan classifier as in Refs. [7], [16] and most vendors utilize it to certify a third parties' products.

A standard hardware-Trojan classifier may be prepared by an IP core vendor as software or cloud services. A Japanese IC design vendor has just started the hardware Trojan detection service [35]. This IC design vendor becomes an IP core supplier in some cases but it becomes an IC core buyer in some cases.

Adversaries aim to design hardware Trojans which are difficult to detect by a hardware-Trojan detection system in order to insert malicious circuits into hardware products. In this section, we discuss several assumptions for adversaries and the AE attacks against hardware-Trojan detection.

#### 3.1.1 Adversary's Goal

For the purposes described below, adversaries may try to attack hardware designs.

**G1** Adversaries insert Trojan nets to hardware design information at gate level.

**G2** Adversaries make a classifier misclassify a Trojan net as a normal net by applying AEs.

**G3** Adversaries attack the classifier for hardware-Trojan detection with the constraint of minimizing the impact of modification to the circuits.

G1 is the main goal of adversaries. The adversaries try to embed hardware Trojans into an original gate-level netlist. G2 and G3 are the supplementary goals. The adversaries try to conceal the embedded hardware Trojans so as not to be detected easily.

#### 3.1.2 Adversary's Capability

Assuming the cases discussed above, vendors can access the classifier and investigate its architecture. If some vendors are malicious, they access their classifier and develop attacks based on the classifier that they own. In this paper, we assume adversaries like the malicious vendors above will attack under the conditions as follows:

**C1** Adversaries know that the hardware vendor utilizes neural networks to detect hardware Trojans.

**C2** Adversaries can access the raw output values of the neural networks.

In real case, hardware-Trojan detection tools may be provided as software or cloud services. We can assume that attackers can also access the tools anytime. If the tools are provided as software, attackers may be able to crack loss values of detection tools by disassembling the tools, which is the worst case. Since current artificial intelligence (AI) techniques require transparency and explainability [36], loss values will be provided as a confi-

dence value in the future to improve service reliability. If the tools are provided as cloud services, attackers can also obtain loss values from provided confidence values.

Note that the assumption C2 is based on a pessimistic and rigorous scenario against AE attacks to improve the tolerance of Trojan detection. In real security module evaluation, complete disclosure of raw output values may be rare. In addition, the requirement of explainable machine learning does not draw directly the consequence of complete disclosure of hardware-Trojan detection techniques against adversarial attacks.

Based on the assumptions above, adversaries slightly modify Trojan circuits, which is similar to perturbation of AEs on image recognition, and make a classifier misclassify Trojan nets as normal nets. If this attack succeeds, the Trojan nets that adversaries newly design will be classified as normal nets by the hardware-Trojan detection utilizing a neural network at hardware vendors. Thus, adversaries are able to hide the hardware Trojan into hardware design information and an AE attack is successfully realized.

### 3.2 AE Attacks on Hardware Design Information

On AE attacks to hardware design information, a small change to a circuit structure causes a significant change to the feature values of the nets in the circuit. In image recognition, perturbation can be theoretically obtained [37]. In contrast, generating adversarial examples for hardware-Trojan detection is quite different. Since the conversion from a graph structure to a feature space is one-way, we cannot take the same scheme as in image recognition. Even if we can add perturbation to feature values of nets in an original circuit and obtain the modified feature values, we cannot always generate a modified circuit which has the modified feature values. In addition, the circuit with arbitrary perturbation is not guaranteed to be logically equivalent to the original one. The modified circuit whose functionality is not logically equivalent to the original one can be easily detected by an existing test tool due to the lack of the original functionality. For example, assume that the function of trigger circuits in a hardware Trojan is destroyed and trigger conditions are satisfied in most cases. Then the hardware Trojan frequently wakes up. In this case, the malicious behavior of a hardware Trojan can be easily detected in a normal product test process [38]. Such circuits are inappropriate for hardware Trojans. Therefore, adversaries must generate logically-equivalent hardware Trojan circuits so that the generated circuit keeps the functionalities of the original circuit (Point 1 below).

We target on the circuit where a hardware Trojan has already been inserted. Generally, a hardware-Trojan circuit is inserted into a non-critical path of an original circuit and thus it may not affect the entire path delay of the original circuit, even if the path delay of the hardware-Trojan circuit is slightly increased. When adversaries modify the original normal circuit, original designers can easily detect the modification because the original normal circuit itself is elaborately designed to satisfy the requirements such as circuit size and circuit path delay. Therefore, adversaries should modify only Trojan circuits to achieve AE attacks (Point 2 below).

Furthermore, on generating AEs for hardware-Trojan detection, adversaries should not significantly decrease the performance of a circuit. For example, if we apply arbitrary modifications to hardware design, the logical equivalence may be broken and the path delay may be affected. The logical equivalence and path delay are important factors on hardware design, and thus significant change in such factors can be easily detected in the existing test process [38]. We must consider the amount of modification based on the factors such as circuit size and circuit path delay to generate an effective AE (Point 3 below).

Based on the discussion above, generating AEs against hardware-Trojan detection must satisfy the following three points:

**Point 1** Modified circuits are logically equivalent to the original ones.

**Point 2** Only Trojan circuits are modified.

**Point 3** AEs degrade classification performance and conceal Trojan nets with small modification.

In the rest of this section, we firstly propose a Trojan-net concealment degree (TCD), and also propose a modification evaluating value (MEV), which indicates how likely the Trojan-nets are classified as normal. Based on these values, we propose an AE generation method that enables us to degrade the performance of hardware-Trojan detection.

### 3.3 Trojan-net Concealment Degree and Modification Evaluating Value

An ideal adversarial example largely degrades detection performance by modifying a circuit as slightly as possible. Thus, in this subsection, we propose TCD, a degree which indicates the possibility of hiding a hardware Trojan, and MEV, a value evaluating the amount of modifications.

#### 3.3.1 Trojan-net Concealment Degree

In order to degrade classification performance, we maximize the loss function of the neural network used in the learning flow. In this case, the loss function is a cross entropy $H$ expressed as follows [16]:

$$H = -\sum_{i=1}^{K} p_i(\boldsymbol{x}(e)) \log q_i(\boldsymbol{x}(e)) \tag{1}$$

where $K$ is the number of units in the output layer ($K = 2$ in the case of Ref. [16]), $p_1(\boldsymbol{x}(e))$ and $p_2(\boldsymbol{x}(e))$ are the functions to return answer labels of $\boldsymbol{x}(e)$. When $e$ is a Trojan net, $p_1(\boldsymbol{x}(e))$ equals to 0 and $p_2(\boldsymbol{x}(e))$ equals to 1. When $e$ is a normal net, $p_1(\boldsymbol{x}(e))$ equals to 1 and $p_2(\boldsymbol{x}(e))$ equals to 0. $q_i(\boldsymbol{x}(e))$ is the function to return the prediction result by the classifier. We obtain $q_i(\boldsymbol{x}(e))$ from the output value of the unit $u_i$ in the output layer of the neural network. If we have two units $u_1$ and $u_2$ in the output layer, we obtain $q_1(\boldsymbol{x}(e))$ from the output of $u_1$ and $q_2(\boldsymbol{x}(e))$ from the output of $u_2$. The output values of $u_1$ and $u_2$ are calculated according to the structure of the neural network in Ref. [39]. $q_1(\boldsymbol{x}(e))$ indicates the possibility that the net $e$ is a normal net. $q_2(\boldsymbol{x}(e))$ indicates the possibility that the net $e$ is a Trojan net. For example, if we have $q_1(\boldsymbol{x}(e)) = -3.60$ and $q_2(\boldsymbol{x}(e)) = 3.76$ for the net $e$, the classifier determines that $e$ is a Trojan net because $q_2(\boldsymbol{x}(e))$ is larger than $q_1(\boldsymbol{x}(e))$.

To adapt Eq. (1) to the whole Trojan nets, we sum up the values for each net and propose the average as the Trojan-net concealment degree (TCD):

$$\mathrm{TCD} = -\frac{1}{|E_t|}\sum_{e_t \in E_t}\left(\sum_{i=1}^{K} p_i(\boldsymbol{x}(e_t)) \log q_i(\boldsymbol{x}(e_t))\right) \tag{2}$$

When TCD becomes large, the difference between the prediction and answer is large. Therefore, if the value is large enough, adversaries can easily achieve their purposes to conceal Trojan nets.

As discussed in Section 3.1.2, we assume that adversaries know that the hardware vendor utilizes neural networks to detect hardware Trojans and also know its output values. Then the adversaries obtain the values of $q_1(\boldsymbol{x}(e_t))$ and $q_2(\boldsymbol{x}(e_t))$ and can calculate the TCD value above, regardless of the structure of the classifier.

#### 3.3.2 Modification Evaluating Value

A modification to the target circuit affects the whole circuit. More modifications lead to larger impacts on the whole circuit. To achieve the purpose G3 in Section 3.1.1, we need to modify the hardware Trojan circuit making it as small as possible. Therefore, we focus on the amount of modifications in addition to TCD.

From the viewpoint of circuit design, there are several evaluating indicators such as (i) increment of gates and (ii) increment of logic levels. Although (i) is generally proportional to the increment of gates, (ii) is not necessarily proportional to the increment of gates. Therefore, it is not always true that minimizing the number of altered gates works well to generate AEs. It is significantly important to consider that on modifying circuits. Generalizing these features, we propose a modification evaluating value (MEV) as follows:

$$\begin{aligned}\mathrm{MEV} &= -\mathrm{TCD} + \sum_{j=1}^{N} \lambda_j m_j \\ &= \frac{1}{|E_t|}\sum_{e_t \in E_t}\left(\sum_{i=1}^{K} p_i(\boldsymbol{x}(e_t)) \log q_i(\boldsymbol{x}(e_t))\right) + \sum_{j=1}^{N} \lambda_j m_j\end{aligned} \tag{3}$$

where $m_j (1 \le j \le N)$ is one of the $N$ kinds of evaluation indicators, and $\lambda_j (1 \le j \le N)$ is the corresponding coefficient. To consider hardware-Trojan detection utilizing neural networks, we take (i) increment of gates and (ii) increment of logic levels into account as the amount of modification, i.e., $N = 2$. When the MEV value is small, this indicates that TCD becomes large with the small modifications. Note that, we can approximate path delay by simply counting logic levels. As in the discussion in Section 3.3.1, adversaries can also calculate the MEV value when Trojan circuit modifications are given.

### 3.4 An AE Generation Method

Now the AE generation problem for hardware Trojans is defined as follows: The inputs are an entire circuit given by a graph $G = (V, E)$ including a hardware Trojan circuit, an original hardware Trojan circuit given by $G_t = (V_t, E_t)$, and a learned neural network classifier $f$ given in Ref. [16]; The output is the modified hardware Trojan circuit; The objective function is to minimize MEV given by Eq. (3).

As we discussed in Section 3.1, since the conversion from a graph structure to a feature space is one-way, we cannot take the same AE generation scheme as in image recognition. Thus, we apply an approach for AE generation in hardware Trojans as follows: We firstly enumerate AE patterns applied to each net and generate an effective AE evaluating MEV.

**Algorithm 1** Generate AEs based on MEV

**Inputs:** Learned classifier $f$, the circuit infected with a hardware Trojan $G = (V, E)$, the hardware Trojan $G_t = (V_t, E_t)$, AE patterns $P$, the number of iterations $L$
**Output:** An AE circuit

```
i ⇐ 0, best_MEV ⇐ 0
now_circuit ⇐ G
next_candidate is null
while i < L do
  for all v_t ∈ V_t in now_circuit that has not yet been modified do
    for all p ∈ P that can be applied to v_t do
      Apply p to v_t and generate the modified circuit G'
      Calculate MEV of G' w.r.t. f
      if MEV < best_MEV then
        next_candidate ⇐ G'
        best_MEV ⇐ MEV
      end if
    end for
  end for
  now_circuit ⇐ next_candidate
  i ⇐ i + 1
end while
return now_circuit
```

However, it is impractical that we modify all the nets in a hardware Trojan because the number of modification patterns is exponentially increased. In order to effectively generate an AE, we have to approximate the evaluation of AEs. Therefore, based on MEV, we further propose an AE generation method that enables us to degrade the performance of hardware-Trojan detection in a practical time. Our proposed method chooses the modification whose MEV is the best at every iteration.

The proposed AE generation method is shown in Algorithm 1 above. Note that $P$ is a set of possible AE patterns by which a Trojan gate is modified to a logically equivalent circuit. Examples of such patterns are provided in Section 4.2. By using the proposed method utilizing MEV, we expect that we can obtain AE which decreases TPR without much modifying the original hardware Trojan circuit. Note that our proposed method can be applied to any hardware-Trojan detection methods utilizing neural networks because our method just deals with the feature values extracted from design information at gate-level netlists.

## 4. Experiments

In this section, we apply the proposed AE generation method in Section 3.4 to three benchmark circuits, RS232-T1000, RS232-T1200 and RS232-T1300, from Trust-HUB [5], [40], [41]. In the proposed AE generation method, we utilize the AE patterns shown in Section 4.2 under the condition described in Section 4.1.

### 4.1 Experimental Setup

The dataset that we utilize in the experiments are 15 bench-

**Table 2** Benchmarks used in our experiments.

| Benchmark [5] | # 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 | 2419 | 27 |
| s35932-T100 | 6407 | 15 |
| s35932-T200 | 6405 | 12 |
| s35932-T300 | 6405 | 37 |
| s38417-T100 | 5798 | 12 |
| s38417-T200 | 5798 | 15 |
| s38417-T300 | 5841 | 44 |
| s38584-T100 | 7343 | 19 |

marks [*3] from Trust-HUB [5]. **Table 2** shows the benchmarks from Trust-HUB that we used in the experiments. Each of these 15 benchmark circuits includes normal circuits and Trojan circuits. As an initial trial, we just randomly pick up three RS232 benchmarks: RS232-T1000, RS232-T1200, and RS232-T1300, out of 15 benchmarks and perform the AE applying experiments against them. We apply leave-one-out cross-validation. For example, if we test RS232-T1000 with AE, the classifier learns 14 benchmarks excluding RS232-T1000 and classifies the nets in RS232-T1000 with AE into a set of normal nets and a set of Trojan nets. We adopt over-sampling for imbalanced training sample distribution between positive and negative samples.

We construct the neural network utilizing Chainer library in Python. The parameters of the neural network utilized in the experiments are as follows: 51 units in the input layer; three layers (200 units, 100 units and 50 units) in the middle layers; and two units in the output layer based on Ref. [16]. We use the sigmoid function for an activation function. This classifier is utilized in both AE generation and evaluation. In fact, the method in Ref. [16] is based on the method in Ref. [22], which originally proposes 51 feature values. Thus, we utilize these 51 feature values for inputs so that we can evaluate the whole impact of adversarial examples to the classification from the viewpoint of feature values. We use an Intel Xeon Bronze 3104 computer environment with a 93 GB memory.

In our experiments, we apply the six AE patterns described in Section 4.2 to the Trojan circuits of RS232-T1000, RS232-T1200 and RS232-T1300 (shown in **Fig. 2**, **Fig. 3** and **Fig. 4**, respectively) according to the proposed AE generation method in Section 3.4 and generate AEs for hardware-Trojan detection. Note that, we set the number $L$ of iterations to three and both $\lambda_1$ and $\lambda_2$ in Eq. (3) to 1 in the experiments.

### 4.2 Six AE Patterns

The hardware-Trojan detection method in Refs. [7], [16], [22] utilizes the distance from the net to a near multiplexer, multiplexer and primary input/output and thus modifications with changing the distance must be efficient. For instance, the number of logic levels between multiplexer gates increases by replacing a four-input-gate with multiple two-input-gates. In this way, we generate a logically-equivalent circuit and alter feature values at the same time.

We generate six AE patterns shown in **Fig. 5**. The first two,

[*3] The benchmarks s38584-T200 and s38584-T300 are not included in our experiment comparing to Ref. [7]. This is because of the following reasons: Firstly, in our experiment, we utilize all the 51 feature values proposed in Ref. [22]. Secondly, in Ref. [22], the 15 benchmarks in Table 2 are utilized as learning datasets when evaluating all the 51 feature values (As mentioned in Section 2.1, after that, the 11 effective feature values are picked up from them [22] and evaluated using the 17 benchmarks [7]). Hence, we adapt the same benchmark condition.

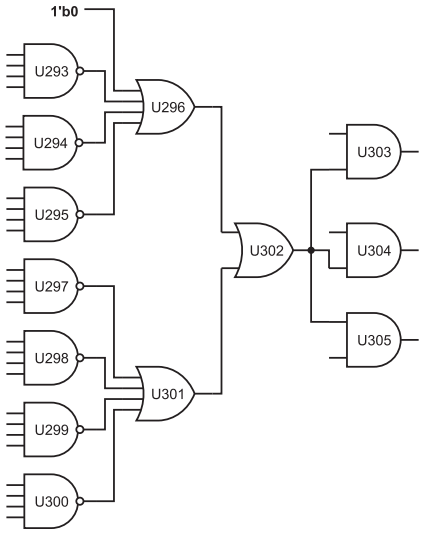


**Fig. 2** Hardware Trojan part embedded in RS232-T1000 [5].

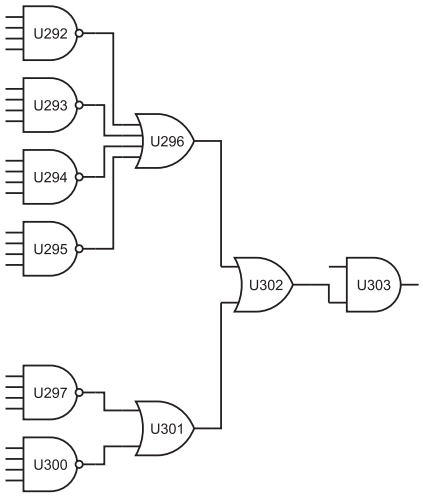


**Fig. 3** Hardware Trojan part embedded in RS232-T1200 [5].

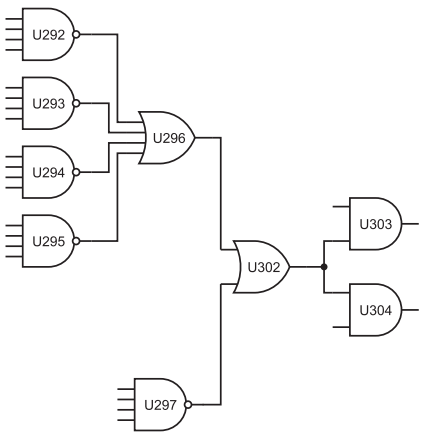


**Fig. 4** Hardware Trojan part embedded in RS232-T1300 [5].

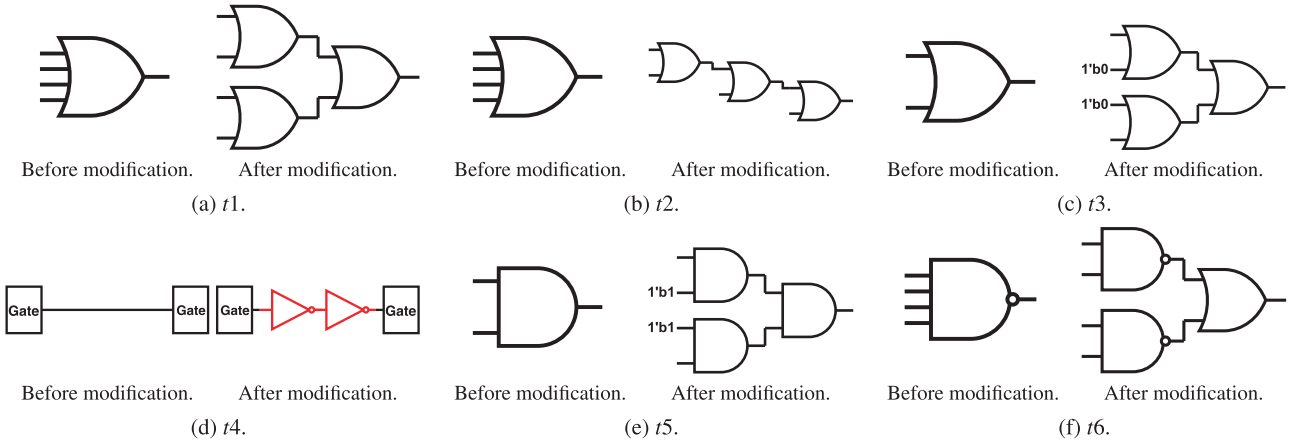


**Fig. 5** Six AE modification patterns that we utilize.

$t1$ and $t2$, replace a single four-input-OR with three two-input-ORs. $t3$ adds the constant 0 and $t5$ adds the constant 1. $t4$ injects two inverters. $t6$ replaces an a single four-input-NAND gate with logically equivalent gates. We may also consider other AE modification patterns, but we now consider these six AE modification patterns for evaluating AE generation for hardware Trojans at the first step. Note that these AE patterns clearly give the logically equivalent modifications.

### 4.3 Evaluation

**Table 3** shows the results when we apply the six AE patterns to RS232-T1000 up to three times. In the first iteration, the gate U298 is modified using the AE pattern $t6$ in Fig. 5, which gives the minimum MEV at that time. In the second iteration, the gate U299 is modified also using $t6$, which gives the minimum MEV at that time. In the third iteration, we enumerate all the modification patterns in Table 3. In Table 3, # indicates "after modification." For instance, #$t6$[U299] at the third row means that $t6$ in Fig. 5 is applied to the gate U299 in Fig. 2 at the second iteration after modification of $t6$[U298]. In the same way, ##$t1$[U296] at the forth row means that $t1$ is applied to the gate U296 in Fig. 2 at the third iteration after modification of $t6$[U298] and $t6$[U299]. Note that, $t4$[U302] means that $t4$ (injecting two inverters between gates) is applied to the output side wire of the gate U302. From the viewpoint of MEV, the most efficient modifications to induce misclassification are $t6$[U298], $t6$[U299] and $t1$[U301]. **Figure 6** shows the Trojan circuit of RS232-T1000 with these AE patterns. Modified gates are colored in red. The MEV becomes 4.29. TPR is decreased to 64.29% by 30.15 points. According to Table 3, applying $t4$[U302] at the third iteration increases six gates and three logic levels compared to the original circuit. On the other hand, applying $t1$[U301] at the third iteration increases six gates and just one logic level compared to the original circuit. $t1$[U301] has less amount of modification than $t4$[U302]. Reflecting this difference in MEV, we can pick up modifications to decrease TPR with minimizing amount of modifications to the circuits. Therefore, we can achieve the purpose G3 discussed in Section 3.1.

**Table 4** shows the results when we apply the six AE patterns to RS232-T1200 up to three times. From the viewpoint of MEV, the most efficient modifications to induce misclassification are

**Table 3** Experimental results of RS232-T1000 with AE modification three times.

| Pattern [ Gate ] | TPR | Incr. of gates | Incr. of logic levels | TCD | MEV |
|---|---|---|---|---|---|
| Original | 94.44% | 0 | 0 | 0.24 | −0.24 |
| $t6$[U298] | 86.84% | 2 | 1 | 0.94 | 2.06 |
| #$t6$[U299] | 80.00% | 4 | 1 | 1.52 | 3.48 |
| ##$t1$[U296] | 80.95% | 6 | 1 | 1.49 | 5.51 |
| ##$t1$[U301] | 64.29% | 6 | 1 | 2.71 | 4.29 |
| ##$t2$[U296] | 78.57% | 6 | 2 | 1.52 | 6.48 |
| ##$t2$[U301] | 69.05% | 6 | 2 | 2.56 | 5.44 |
| ##$t3$[U302] | 66.67% | 6 | 2 | 2.12 | 5.88 |
| ##$t4$[U302] | 54.76% | 6 | 3 | 3.24 | 5.76 |
| ##$t5$[U303] | 64.29% | 6 | 2 | 2.10 | 5.90 |
| ##$t5$[U304] | 78.57% | 6 | 2 | 1.53 | 6.47 |
| ##$t5$[U305] | 78.57% | 6 | 2 | 1.54 | 6.46 |
| ##$t6$[U293] | 80.95% | 6 | 1 | 1.45 | 5.55 |
| ##$t6$[U294] | 73.81% | 6 | 1 | 1.95 | 5.05 |
| ##$t6$[U295] | 73.81% | 6 | 1 | 1.60 | 5.40 |
| ##$t6$[U297] | 78.57% | 6 | 1 | 1.81 | 5.19 |
| ##$t6$[U300] | 73.81% | 6 | 1 | 1.88 | 5.12 |

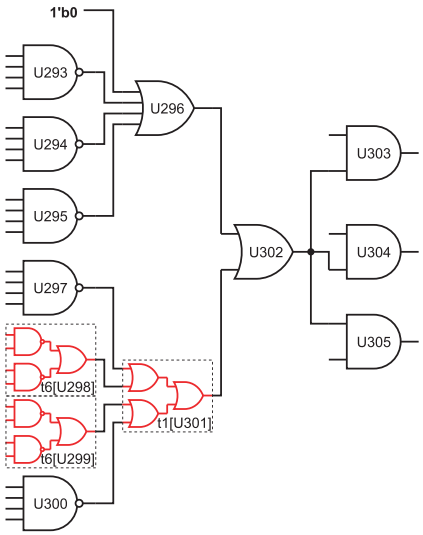


**Fig. 6** Hardware Trojan embedded in RS232-T1000 after three modifications (modified gates are colored in red).

$t6$[U292], $t6$[U293] and $t6$[U297] in Fig. 3. The MEV becomes 5.34. TPR is decreased to 75.00% by 22.06 points. The tendency on Table 4 is similar to RS232-T1000 in terms of MEV, which gets higher when more amount of modification is applied to the circuit. The most powerful AE choice is ##$t4$[U296] which has the second highest MEV values, decreasing TPR the most. Showing high MEV means giving a strong impact on performance of

**Table 4** Experimental results of RS232-T1200 with AE modification three times.

| Pattern [ Gate ] | TPR | Incr. of gates | Incr. of logic levels | TCD | MEV |
|---|---|---|---|---|---|
| Original | 97.06% | 0 | 0 | 0.28 | −0.28 |
| *t*6[U292] | 86.11% | 2 | 1 | 0.99 | 2.01 |
| #*t*6[U293] | 78.95% | 4 | 1 | 1.55 | 3.45 |
| ##*t*1[U296] | 70.00% | 6 | 2 | 2.11 | 5.89 |
| ##*t*2[U296] | 67.50% | 6 | 3 | 2.19 | 6.81 |
| ##*t*3[U301] | 80.00% | 6 | 1 | 1.49 | 5.51 |
| ##*t*3[U302] | 72.50% | 6 | 2 | 2.09 | 5.91 |
| ##*t*4[U296] | 62.50% | 6 | 3 | 2.62 | 6.38 |
| ##*t*4[U301] | 77.50% | 6 | 2 | 1.62 | 6.38 |
| ##*t*5[U303] | 72.50% | 6 | 2 | 2.09 | 5.91 |
| ##*t*6[U294] | 75.00% | 6 | 1 | 1.56 | 5.44 |
| ##*t*6[U295] | 80.00% | 6 | 1 | 1.47 | 5.53 |
| ##*t*6[U297] | 75.00% | 6 | 1 | 1.66 | 5.34 |
| ##*t*6[U300] | 80.00% | 6 | 1 | 1.47 | 5.53 |

**Table 5** Experimental results of RS232-T1300 with AE modification three times.

| Pattern [ Gate ] | TPR | Incr. of gates | Incr. of logic levels | TCD | MEV |
|---|---|---|---|---|---|
| Original | 100.00% | 0 | 0 | 0.00 | 0.00 |
| *t*4[U302] | 70.97% | 2 | 2 | 2.42 | 1.58 |
| #*t*6[U297] | 72.73% | 4 | 2 | 2.28 | 3.72 |
| ##*t*1[U296] | 74.29% | 6 | 3 | 2.80 | 6.20 |
| ##*t*2[U296] | 71.43% | 6 | 4 | 3.05 | 6.95 |
| ##*t*3[U302] | 74.29% | 6 | 3 | 2.78 | 6.22 |
| ##*t*5[U303] | 74.29% | 6 | 3 | 2.15 | 6.85 |
| ##*t*5[U304] | 74.29% | 6 | 3 | 2.15 | 6.85 |
| ##*t*6[U292] | 68.57% | 6 | 3 | 2.78 | 6.22 |
| ##*t*6[U293] | 71.43% | 6 | 3 | 2.26 | 6.74 |
| ##*t*6[U294] | 71.43% | 6 | 3 | 2.68 | 6.32 |
| ##*t*6[U295] | 65.71% | 6 | 3 | 2.49 | 6.51 |

the circuit. Selecting a modification pattern with the smallest MEV leads to selecting a modification pattern with a small impact on circuit modification.

**Table 5** shows the results when we apply the six AE patterns to RS232-T1300 up to three times. From the viewpoint of MEV, the most efficient modifications to induce misclassification are *t*4[U302], *t*6[U297] and *t*1[U296] in Fig. 4. The MEV becomes 6.20. TPR is decreased to 74.29% by 25.71 points. Results in Table 5 point out that applying modifications many times is not always effective. The lowest TPR among all the candidates in Table 5 is *t*4[U302], which is obtained when modifying just once, when we select modification candidates based on MEV.

In this experiment, we set the number of modifications to three, and we repeat applying modifications. However, there are some cases where modifying just once or twice is the best. We need to explore the optimal number of times of modifications for each circuit in the future. In addition, if we set both $\lambda_1$ and $\lambda_2$ in MEV to 0, where the MEV is equal to the negated TCD, more powerful AE modification patterns are selected as shown in Table 3, Table 4 and Table 5 (underlined parts in the TCD columns). However, these patterns have more impacts on the circuits from the viewpoint of the increment of gates and logic levels than patterns based on MEV ($\lambda = 1$). Therefore, setting $\lambda$ values properly is also important.

From the discussion above, our proposed method effectively decreases TPR by repeatedly modifying the hardware Trojan based on MEV. As an initial trial, we have just randomly picked up three RS232 benchmarks out of 15 benchmarks. The experimental results demonstrate that our proposed AE generation method effectively picks up the effective gates to apply to AE patterns and successfully decreases TPR in all these three cases. The proposed TCD well indicates the strong gate modification candidates decreasing TPR primarily. The proposed MEV indicates the ideal gate modification candidates which decrease TPR and also do not give a large impact on circuit performance as in Table 3, Table 4 and Table 5.

As the classifier that we use in this paper cannot detect the inserted hardware Trojans with AE modifications, it clearly reveals that existing Trojan detection methods are vulnerable to AE attacks. For example, by applying our proposed AE patterns, the number of levels from the nets to certain types of gates has changed. Changing the feature values by perturbation of AE modification leads Trojan nets to get closer to normal nets and it is difficult to classify these nets as Trojan nets.

Note that, since AE modifications that we apply introduce actually redundant gates for circuit design, there is a possibility of removal of these gates by optimization in logic synthesis. However, modification is available in any time in design step. For example, clock trees are designed after circuit synthesis [42]. Thus, modifications in this step are not removed by a logic synthesizer. We have to consider any threats in every step in designing circuits and try to remove them.

## 5. Conclusion

In this paper, we have proposed a framework generating adversarial examples for hardware-Trojan detection for gate-level netlists utilizing neural networks. The experimental results demonstrate that the proposed AE attack succeeds in inducing the classifier to misclassify Trojan nets. As a result, TPR decreases by 30.15 points at most. We can conclude that the proposed method gives the world-first step to AE generation framework for hardware Trojans. Our proposed method can be applied to any hardware-Trojan detection methods utilizing neural networks because our method just deals with the feature values extracted from design information at gate-level netlists.

From the viewpoint of IC designers, it is important to analyze the adversaries' techniques to enforce the tolerance of the classifier against AE attacks. We analyze the IC design scenario by introducing a new attack method which can be realized. We can utilize the information obtained by analyzing the AE attack for strengthening the defense techniques against hardware Trojans and AE attacks. By investigating the AE attacks, IC designers will be able to take countermeasures against them. In the future, in order to defeat our AE attack, we will develop a robust hardware-Trojan detection method utilizing adversarial training techniques, for example, retraining the models with circuits containing AEs.

### References

[1] Rostami, M., Koushanfar, F., Rajendran, J. and Karri, R.: Hardware security: threat models and metrics, *Proc. International Conference on Computer-Aided Design* (*ICCAD*), pp.819–823 (2013).

[2] Francq, J. and Frick, F.: Introduction to hardware Trojan detection methods, *Proc. 2015 Design, Automation & Test in Europe Conference & Exhibition* (*DATE*), *EDAA*, pp.770–775 (2015).

[3] Xiao, K., Forte, D., Jin, Y., Karri, R., Bhunia, S. and Tehranipoor, M.: Hardware trojans: lessons learned after one decade of research, *ACM Trans. Design Automation of Electronic Systems* (*TODAES*), Vol.22,

No.1, pp.1–23 (2016).
[4] Liu, B. and Qu, G.: VLSI supply chain security risks and mitigation techniques: A survey, *Integration, VLSI Journal*, Vol.55, pp.438–448 (2016).
[5] TrustHub.org: Trust-HUB, available from ⟨http://trust-hub.org/benchmarks/trojan⟩.
[6] Oya, M., Shi, Y., Yanagisawa, M. and Togawa, N.: A score-based classification method for identifying hardware-Trojans at gate-level netlists, *Proc. 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp.465–470 (2015).
[7] Hasegawa, K., Yanagisawa, M. and Togawa, N.: A hardware-Trojan classification method using machine learning at gate-level netlists based on Trojan features, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E100.A, No.7, pp.1427–1438 (online), DOI: 10.1587/transfun.E100.A.1427 (2017).
[8] Inoue, T., Hasegawa, K., Yanagisawa, M. and Togawa, N.: Designing hardware Trojans and their detection based on a SVM-based approach, *Proc. International Conference on ASIC*, pp.811–814 (2018).
[9] Dong, C., He, G., Liu, X., Yang, Y. and Guo, W.: A multi-layer hardware trojan protection framework for IoT chips, *IEEE Access*, Vol.7, pp.23628–23639 (2019).
[10] Inoue, T., Hasegawa, K., Yanagisawa, M. and Togawa, N.: Designing subspecies of hardware Trojans and their detection using neural network approach, *Proc. 2018 IEEE 8th International Conference on Consumer Electronics in Berlin* (*ICCE-Berlin*) (2018).
[11] Goodfellow, I.J., Shlens, J. and Szegedy, C.: Explaining and harnessing adversarial examples, *Proc. 2015 International Conference on Learning Representations* (*ICLR*) (2015).
[12] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R.: Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199, pp.1–10 (online), DOI: 10.1021/ct2009208 (2013).
[13] Akhtar, N. and Mian, A.: Threat of adversarial attacks on deep learning in computer vision: A survey, *IEEE Access*, pp.14410–14430 (2018).
[14] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T. and Song, D.: Robust Physical-World Attacks on Deep Learning Models, *Computing Research Repository* (*CoRR*), Vol.abs/1707.0 (online), DOI: 10.1109/CVPR.2018.00175 (2017).
[15] Grosse, K., Papernot, N., Manoharan, P., Backes, M. and McDaniel, P.: *Adversarial Examples for Malware Detection*, pp.62–79 (2017).
[16] Hasegawa, K., Yanagisawa, M. and Togawa, N.: Hardware Trojans Classification for Gate-level Netlists Using Multi-layer Neural Networks, *Proc. 2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design* (*IOLTS*), pp.227–232 (online), DOI: 10.1109/IOLTS.2017.8046227 (2017).
[17] Xiang, H., Chao, K.-Y.K. and Wong, M.D.F.: An ECO Routing Algorithm for Eliminating Coupling-Capacitance Violations, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.25, No.9, pp.1754–1762 (2006).
[18] Huang, S.-L., Lin, W.-H., Huang, P.-K. and Huang, C.-Y.R.: Match and Replace: A Functional ECO Engine for Multierror Circuit Rectification, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.32, No.3, pp.467–478 (2013).
[19] Nozawa, K., Hasegawa, K., Hidano, S., Kiyomoto, S., Hashimoto, K. and Togawa, N.: Adversarial Examples for Hardware-Trojan Detection at Gate-Level Netlists, *Computer Security*, pp.341–359, Springer International Publishing (2020).
[20] Chakraborty, R.S., Narasimhan, S. and Bhunia, S.: Hardware Trojan: Threats and emerging solutions, *Proc. International High-Level Design Validation and Test Workshop* (*HLDVT*), pp.166–171 (2009).
[21] Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets, *Journal of Theoretical Biology*, Vol.22, No.3, pp.437–467 (online), DOI: 10.1016/0022-5193(69)90015-0 (1969).
[22] Hasegawa, K., Yanagisawa, M. and Togawa, N.: Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier, *Proc. 2017 IEEE International Symposium on Circuits and Systems* (online), DOI: 10.1109/ISCAS.2017.8050827 (2017).
[23] Chakraborty, R.S., Wolff, F., Paul, S., Papachristou, C. and Bhunia, S.: MERO: A Statistical Approach for Hardware Trojan Detection, *Proc. Cryptographic Hardware and Embedded Systems* (*CHES 2009*), Clavier, C. and Gaj, K. (Eds.), pp.396–410, Springer Berlin Heidelberg (2009).
[24] Kurakin, A., Goodfellow, I.J. and Bengio, S.: Adversarial examples in the physical world, *Proc. 2017 International Conference on Learning Representations* (*ICLR*) (2017).
[25] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Tram, F., Prakash, A., Kohno, T. and Song, D.: Physical Adversarial Examples for Object Detectors, *CoRR* (2018).
[26] Carlini, N. and Wagner, D.: Audio adversarial examples: Targeted attacks on speech-to-text, *2018 IEEE Security and Privacy Workshops* (*SPW*) (2018).
[27] Jia, R. and Liang, P.: Adversarial examples for evaluating reading comprehension systems, *Proc. 2017 Conference on Empirical Methods in Natural Language Processing*, pp.2021–2031, Association for Computational Linguistics (2017).
[28] Iyyer, M., Wieting, J., Gimpel, K. and Zettlemoyer, L.: Adversarial example generation with syntactically controlled paraphrase networks, *Proc. 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol.1 (Long Papers), pp.1875–1885, Association for Computational Linguistics (2018).
[29] Zügner, D., Akbarnejad, A. and Günnemann, S.: Adversarial attacks on neural networks for graph data, *Proc. 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18*, pp.2847–2856, ACM Press (2018).
[30] Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J. and Song, L.: Adversarial attack on graph structured data, *Proc. International Conference on Machine Learning* (*ICML*) (2018).
[31] Duch, W. and Diercksen, G.H.: Feature space mapping as a universal adaptive system, *Computer Physics Communications*, Vol.87, No.3, pp.341–371 (online), DOI: 10.1016/0010-4655(95)00023-9 (1995).
[32] Baumgarten, A., Steffen, M., Clausman, M. and Zambreno, J.: A case study in hardware Trojan design and implementation, *International Journal of Information Security*, Vol.10, No.1, pp.1–14 (online), DOI: 10.1007/s10207-010-0115-0 (2011).
[33] Guin, U., Huang, K., Dimase, D., Carulli, J.M., Tehranipoor, M. and Makris, Y.: Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain, *Proc. IEEE*, Vol.102, pp.1207–1228 (online), DOI: 10.1109/JPROC.2014.2332291 (2014).
[34] EDN: Using 3rd party IP in ASIC/SoC design (2013), available from ⟨https://www.edn.com/using-3rd-party-ip-in-asic-soc-design/⟩.
[35] Toshiba Information Systems Corp.: HTfinder, available from ⟨https://www.tjsys.co.jp/lsi/htfinder/index_j.htm⟩ (in Japanese).
[36] Adadi, A. and Berrada, M.: Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI), *IEEE Access*, Vol.6, pp.52138–52160 (online), DOI: 10.1109/ACCESS.2018.2870052 (2018).
[37] Moosavi-Dezfooli, S.-M., Fawzi, A. and Frossard, P.: DeepFool: A simple and accurate method to fool deep neural networks, *IEEE Conference on Computer Vision and Pattern Recognition*, pp.2574–2582 (2016).
[38] Bhunia, S., Hsiao, M.S., Banga, M. and Narasimhan, S.: Hardware Trojan attacks: Threat analysis and countermeasures, *Proc. IEEE*, Vol.102, No.8, pp.1229–1247 (2014).
[39] Hasegawa, K., Yanagisawa, M. and Togawa, N.: Empirical Evaluation and Optimization of Hardware-Trojan Classification for Gate-Level Netlists Based on Multi-Layer Neural Networks, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E101.A, No.12, pp.2320–2326 (online), DOI: 10.1587/transfun.E101.A.2320 (2018).
[40] Shakya, B., He, T., Salmani, H., Forte, D., Bhunia, S. and Tehranipoor, M.: Benchmarking of hardware trojans and maliciously affected circuits, *Journal of Hardware and Systems Security*, Vol.1, No.1, pp.85–102 (2017).
[41] Salmani, H., Tehranipoor, M. and Karri, R.: On design vulnerability analysis and trust benchmarks development, *2013 IEEE 31st International Conference on Computer Design* (*ICCD*), pp.471–474 (2013).
[42] Tsai, J.-L., Zhang, L. and Chen, C.C.-P.: Statistical Timing Analysis Driven Post-Silicon-Tunable Clock-Tree Synthesis, *Proc. ICCAD-2005, IEEE/ACM International Conference on Computer-Aided Design*, pp.575–581, IEEE (online), DOI: 10.1109/ICCAD.2005.1560132 (2005).

**Kohei Nozawa** received his B.Eng. degree from Waseda University in 2019 in computer science and communications engineering. He is presently working toward his M.Eng. degree there. His research interests are hardware Trojans and adversarial examples.

**Kento Hasegawa** received his B.Eng., M.Eng., and Dr.Eng. degrees from Waseda University in 2016, 2017, and 2020, respectively, all in computer science and communications engineering. In 2020, he joined KDDI Corp. His research interests are secure encryption circuit design and hardware Trojans.

**Seira Hidano** received his M.E. and Ph.D. degrees in computer science and engineering from Waseda University, Japan, in 2009 and 2012, respectively. In 2010, he was a JSPS research fellow. In 2011 and 2012, he was a research assistant at Waseda University. In 2013, he joined KDDI. He is currently a research engineer of the Information Security Lab. in KDDI Research, Inc. His research interest includes trustworthy AI, information theoretic security, and privacy preservation.

**Shinsaku Kiyomoto** received his B.E. in engineering sciences and his M.E. in Material Science from Tsukuba University, Japan, in 1998 and 2000, respectively. He joined KDD (now KDDI) and has been engaged in research on stream ciphers, cryptographic protocols, and mobile security. He is currently a senior researcher at the Information Security Laboratory of KDDI Research, Inc. He was a visiting researcher of the Information Security Group, Royal Holloway University of London from 2008 to 2009. He received his doctorate in engineering from Kyushu University in 2006. He received the IEICE Young Engineer Award in 2004, Distinguished Contributions Awards in 2011, and Achievement Award in 2016. He is a member of IEICE and JPS.

**Kazuo Hashimoto** received his M.Eng. in Electronics, Tohoku University in 1979, M.Sci. degree in Computer Science, Brown University in 1986 and his Ph.D. degree in Information Science, Tohoku University in 2001. He is presently a Professor of Research Innovation Center, Waseda University. His research interests include machine learning and information security.

**Nozomu Togawa** received his B.Eng., M.Eng., and Dr.Eng. degrees from Waseda University in 1992, 1994, and 1997, respectively, all in electrical engineering. He is presently a Professor in the Department of Computer Science and Communications Engineering, Waseda University. His research interests are VLSI design, graph theory, and computational geometry. He is a member of IEEE, ACM, and IEICE.