Trojan-feature Extraction at Gate-level Netlists and Its Application to Hardware-Trojan Detection Using Random Forest Classifier

Kento Hasegawa†, Masao Yanagisawa†, Nozomu Togawa†
†Department of Computer Science and Communications Engineering, Waseda University
Email: {kento.hasegawa, togawa}@togawa.cs.waseda.ac.jp

***Abstract*—Recently, due to the increase of outsourcing in IC design, it has been reported that malicious third-party vendors often insert hardware Trojans into their ICs. How to detect them is a strong concern in IC design process. The features of hardware-Trojan infected nets (or Trojan nets) in ICs often differ from those of normal nets. To classify all the nets in netlists designed by third-party vendors into Trojan ones and normal ones, we have to extract effective Trojan features from Trojan nets. In this paper, we first propose 51 Trojan features which describe Trojan nets from netlists. Based on the importance values obtained from the random forest classifier, we extract the best set of 11 Trojan features out of the 51 features which can effectively detect Trojan nets, maximizing the F-measures. By using the 11 Trojan features extracted, the machine-learning based hardware Trojan 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 74.6%, which realizes the *best values* among existing machine-learning-based hardware-Trojan detection methods.**

***Index Terms*—hardware Trojan, gate-level netlist, machine learning, random forest, F-measure**

I. Introduction

As the demand for high-functioning hardware devices continuously increases, hardware vendors tend to produce their IC products inexpensively and easily and they often outsource their IC designs to third-party vendors, where malicious third-party vendors can have a chance to insert hardware Trojans into ICs.

Hardware Trojans are malicious functions inserted in ICs in their design step and/or manufacturing step. Especially in design step, we can easily insert hardware Trojans into ICs because malicious vendors just modify electrical design files through networks. It is strongly required to detect hardware Trojans in IC design step. In this paper, we focus on hardware Trojans inserted at gate-level netlists in IC design step.

A. Hardware-Trojan detection in IC design step

Several hardware-Trojan detection methods in IC design step have been proposed.

UCI [5] detects hardware Trojans based on electrical hardware design information and the logical simulation of circuits. In UCI, hardware Trojans must be really activated. However, since some hardware Trojans are activated in very rare conditions, detecting them using only UCI is difficult.

FANCI [12] detects hardware Trojans using truth tables obtained by the hardware design information and gives suspicious flags to rare-transition nets. Since FANCI focuses on truth tables, it is difficult to detect hardware Trojans using sequential circuits.

A score-based hardware-Trojan detection method [9] gives Trojan scores to all the nets based on the features of Trojan nets extracted beforehand. This method focuses on the net features frequently used in hardware Trojans as well as their signal transitions.

The hardware Trojans which defeat these hardware-Trojan detection methods have also been developed. For example, DeTrust [13], which defeats FANCI, has been proposed. Such a vicious circle clearly exists in hardware-Trojan detection.

To improve the current situation, a machine-learning-based hardware-Trojan detection method has been proposed very recently [4]. By using machine learning, we can update the hardware-Trojan database for new types of hardware Trojans and detect them with the updated classifier. In [4], they use the Trojan features proposed in [9] as frequently-used features in Trojan nets. However, there are many features which describe the structure of netlists and we do not know which ones to be best used for machine learning. The performance of machine-learning-based hardware-Trojan detection will be much improved if effective Trojan features are extracted from a given netlist.

Several related works which use machine learning for hardware-Trojan detection have been proposed [6], [8]. In [6], they detect hardware Trojans by analyzing power consumption using machine learning. [8] is a real-time hardware-Trojan detection method for many-core platform. However, both of them just focus on run-time hardware-Trojan detection and none of them focus on Trojan feature extraction for hardware-Trojan detection in IC design step.

In this paper, we classify a set of nets into Trojan ones and normal ones using Trojan features from a given netlist which are extracted using the random forest classifier [2].

B. Summary of the proposal

In this paper, we focus on hardware-Trojan detection using machine learning in IC design step. Firstly, we propose 51 gate-level Trojan features from a given gate-level netlist which describe Trojan nets very well. At that time, we utilize the random forest classifier [2], one of the strong machine-learning methods. Since the random forest gives the *importance value* for every Trojan feature, we can effectively know which one contributes more than the others to detect hardware Trojans. Thus we select the most effective 11 Trojan features which maximize the average F-measures.

Experimental results demonstrate that machine-learning-based hardware Trojan detection using these 11 Trojan features achieves 100% true positive rate and 100% true negative rate in some cases and the best F-measures compared to an existing machine-learning-based hardware Trojan detection method.

C. Contributions of this paper

The contributions of this paper are summarized as follows: 1) we propose the 51 gate-level Trojan features which describe Trojan nets from a given netlist, and then select the best 11 Trojan features to maximize F-measures for hardware Trojan classification; 2) we have obtained 100% true positive rate and 100% true negative rate in several Trust-HUB benchmarks [14], and obtained the *best values* in average F-measures compared to a conventional method.

II. Trojan Features and Hardware-Trojan Classification

In machine learning, we have to extract the *appropriate* number of Trojan features. 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

978-1-4673-6853-7/17/$31.00 ©2017 IEEE

TABLE I
THE EXAMPLES OF TROJAN FEATURES.

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

classification and much time to learn. We cannot even classify the given nets correctly using all of these large Trojan features, which is known as *the curse of dimensionality* [1]. In this section, using a motivational example, we demonstrate that we should extract the appropriate number of Trojan features from a given netlist.

Table I shows several sample Trojan features from benchmarks in Trust-HUB. Let us focus on a net n in a given netlist of Trust-HUB. The feature "fan_in_5" (f_1) shows the number of fanins up to five-level away from the net n. The feature "out_nearest_pout" (f_2) shows the minimum logic level from n to any primary output. The feature "in_nearest_flipflop" (f_3) shows the minimum logic level to any flip-flop from the input side of the net n. In Table I, the nets of (1)–(3) are Trojan nets and (4) and (5) are normal nets. Now we classify the nets (1)–(5) in Table I into Trojan nets and normal nets. For simplicity, we give a single threshold value to every Trojan feature.

Firstly, 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 I shows $f_1 > 20$. In this case, the net (4) is identified to be a Trojan net and this is an incorrect classification. Therefore we cannot classify the nets correctly using f_1 only.

Secondly, we assume that we classify the nets using f_1 and f_2. When we set the threshold value 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 I shows $f_2 < 7$. In this case, we classify all the nets (1)–(5) correctly into Trojan ones ((1)–(3)) and normal ones ((4) and (5)).

Lastly, 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 I shows $f_3 < 10$. In this case, the net (2) is identified to be a normal net and this is an incorrect classification. We cannot classify the nets correctly using all of f_1, f_2 and f_3.

This simple example shows that we should use only f_1 and f_2 as Trojan feature values in case of Table I, not using all the Trojan features f_1, f_2, and f_3. We should extract the appropriate number of Trojan features for effective Trojan classification.

III. NETLIST FEATURES TO DETECT HARDWARE TROJANS

In this section, we refer to the first 12 benchmarks listed in Table II which is published at Trust-HUB [14] for training sets. 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 net. Then we propose several gate-level Trojan features which are likely to be related to Trojan nets.

A. 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.

Especially in 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 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 up to x-level away from every net n (fan_in_x) as a Trojan feature. In this paper, 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.

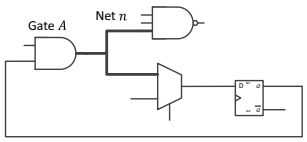

Fig. 1. The example of a loop in a netlist (in_loop_3 for the net n).

B. Flip-flops

As in RS232-T1200, some hardware Trojans have sequential-trigger circuits. Since the hardware Trojans circuits are always so small, they are placed very locally. The level of flip-flops for sequential-trigger circuits 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 levels 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 features. In this paper, we set $x = 1, 2, 3, 4, 5$ in the same way.

C. Multiplexers

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 features. In this paper, we set $x = 1, 2, 3, 4, 5$ in the same way.

D. Loops in a netlist

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 the m-level away from the net n. A loop can be found not only in sequential-trigger circuits but also in ring-oscillator Trojans.

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

E. Constants

In [9], 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 nets including constants become the efficient flags 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 features. In this paper, we consider $x = 1, 2, 3, 4, 5$ in the same way.

F. Levels to primary input and output

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.

TABLE II
THE TRUST-HUB BENCHMARKS USED IN THE EXPERIMENTS.

Data name	Number of normal nets	Number of Trojan nets	Data name	Number of normal nets	Number of Trojan nets
RS232-T1000	283	36	s38417-T100	5,798	12
RS232-T1200	289	34	s38417-T200	5,798	15
RS232-T1300	287	29	s38417-T300	5,801	44
RS232-T1400	273	45	s38584-T100	7,343	19
RS232-T1500	283	39	RS232-T1100	284	36
s15850-T100	2,429	27	RS232-T1600	292	29
s35932-T100	6,407	15	s35932-T200	6,405	12
s35932-T300	6,405	37	s35932-free	6,405	0

TABLE III
THE EXTRACTED FEATURES FROM A NETLIST ($1 \leq x \leq 5$).

Trojan feature	Description
fan_in_x	The number of logic-gate fanins up to 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.
out_loop_x	The number of up to x-level loops.
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.

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 features.

G. Trojan features extracted from a netlist

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

In Section IV, we carefully select the small number of the Trojan features out of these 51 features to effectively detect hardware Trojans.

IV. SELECTING TROJAN FEATURES USING RANDOM FOREST TO EFFECTIVELY DETECT HARDWARE TROJANS

In this section, we carefully select the Trojan features to detect hardware Trojans from the 51 Trojan features extracted in Section III using some of the Trust-HUB benchmarks [14] for training sets and the random forest classifier [2].

In Section III, we have extracted the 51 features which must be related to hardware Trojans. However, as discussed in Section II, it must be impractical to use all the 51 Trojan features for hardware Trojan classification. We have to select the best set of Trojan features which most effectively detect hardware Trojans among the 51 Trojan features. In this paper, we apply the random forest classifier to select the best set of Trojan features. The random forest is one of the machine-learning algorithms and we can obtain the *importance value* for the Trojan feature used which describe 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 features to effectively classify the nets into Trojan nets and normal nets.

As in Section III, we also use the first 12 Trust-HUB benchmarks in Table II for training sets in this section.

A. Step 1: Extract Trojan feature values from a netlist

In Step 1, we first extract the Trojan feature values from every netlist. We use the 51 Trojan features here as listed in Table III.

B. Step 2: Select the best set of Trojan features

In Step 2, we select the best set of Trojan features based on the extracted feature values in Step 1 using the random forest classifier.

1) F-measure: 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.

TABLE IV
SELECTING THE BEST SET OF FEATURES (FIRST STEP).

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

TABLE V
SELECTING THE BEST SET OF FEATURES (SECOND STEP).

# of features	F-measure
12	75.9%
11	77.8%
10	74.0%

Based on the values of TN, FP, TP and FN, we can have four more values to evaluate the classification results: the true positive rate (TPR), the true negative rate (TNR), the precision, and the F-measure; 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 TP/(FP+TP); The F-measure, F, is the harmonic mean of the precision and the recall, and represented by $F = 2PR/(P+R)$.

All of the values above are important and, specifically, the F-measure is the best to measure the classification results very well [3]. In this paper, we focus on the F-measure to evaluate the classification results for machine-learning-based hardware-Trojan detection.

2) Hardware-Trojan detection using the random forest: We classify all the nets in a given netlist into Trojan ones and normal ones based on the 51 features listed in Table III using the random forest classifier. We use the leave-one-out cross-validation method [7] to validate machine-learning-based hardware-Trojan detection, 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), and then have a classification result for each benchmark.

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-measures for all of the first 12 benchmarks listed in Table II. We consider the average F-measure values over these 12 benchmarks as the measure of the performance for classification.

3) Selecting the best set of Trojan features:

1) (First step) First, we repeatedly halves the number of Trojan features. At first, as a result of the random forest classification above, it gives the importance value to each one of the 51 Trojan features. Then we select the first half of Trojan features whose importance values are higher than those of the second half. Then, we also classify the given netlists in Table II using the selected Trojan features and obtain the average F-measure value. We repeatedly perform this process until the average F-measure value decreases the previous one.

   Table IV shows the average F-measure values when we decrease the number of Trojan features from 51 to 25, 12 and 6. From this table, the 12 Trojan features give the best value in terms of F-measure.
2) (Second step) Next, we carefully examine the 12 Trojan features selected above. We pick up one of the 12 Trojan features and discard it. We classify the given netlists using the remaining Trojan features and obtain the average F-measure value. We try this process for each of the 12 Trojan 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 V shows the results. When we select the 11 Trojan features, we can have the *best* average F-measure value.

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

V. THE RESULTS OF MACHINE-LEARNING-BASED HARDWARE-TROJAN CLASSIFICATION

In this section, we demonstrate the results of machine-learning-based hardware-Trojan classification using the 11 gate-level Trojan features obtained in Section IV. We use the random forest classifier implemented in Python using the scikit-learn [10] machine-learning library.

TABLE VI
THE BEST SET OF 11 TROJAN FEATURES AND THEIR IMPORTANCE VALUES.

No.	Trojan feature	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 VII
THE RESULTS OF MACHINE-LEARNING-BASED CLASSIFICATION.

Test data	TN	FP	FN	TP	TPR	TNR	Precision	F-measure
RS232-T1000	280	3	0	36	100.0%	98.9%	92.3%	96.0%
RS232-T1200	289	0	4	30	88.2%	100.0%	100.0%	93.8%
RS232-T1300	287	0	0	29	100.0%	100.0%	100.0%	100.0%
RS232-T1400	273	0	1	44	97.8%	100.0%	100.0%	98.9%
RS232-T1500	282	1	2	37	94.9%	99.6%	97.4%	96.1%
s15850-T100	2,418	1	6	21	77.8%	100.0%	95.5%	85.7%
s35932-T100	6,407	0	4	11	73.3%	100.0%	100.0%	84.6%
s35932-T300	6,404	1	7	30	81.1%	100.0%	96.8%	88.2%
s38417-T100	5,798	0	8	4	33.3%	100.0%	100.0%	50.0%
s38417-T200	5,798	0	8	7	46.7%	100.0%	100.0%	63.6%
s38417-T300	5,801	0	11	33	75.0%	100.0%	100.0%	85.7%
s38584-T100	7,341	2	18	1	5.3%	100.0%	33.3%	9.1%
RS232-T1100	279	5	18	18	50.0%	98.2%	78.3%	61.0%
RS232-T1600	289	3	2	27	93.1%	99.0%	90.0%	91.5%
s35932-T200	6,405	0	11	1	8.3%	100.0%	100.0%	15.4%
s35932-free	6,405	0	0	0	100.0%	100.0%	100.0%	100.0%

We use all the 16 netlists in Table II for the experiments, where the last four netlists are newly added to evaluate whether the selected 11 Trojan features can be effectively applied to them. Especially, the last netlist is a Trojan-free netlist. We also use the leave-one-out cross-validation method [7] to evaluate Trojan-included netlists. In evaluating the Trojan-free netlist, we consider the Trojan-included netlists to be training sets and the Trojan-free netlist to be a test set and then we evaluate the classification result in the Trojan-free netlist.

A. The results of machine-learning-based classification

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

Since we have obtained 100% precisions 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 the large number of nets. Especially, as the FP values in almost all the netlists become less than three, almost all the normal nets in these benchmarks are also correctly identified to be normal nets.

Note that Table VII demonstrate that we find out at least one Trojan net in Trojan-included netlists while we find out no Trojan nets in the Trojan-free netlist. These results simply imply that we can successfully classify all the given netlists into Trojan-included netlists and Trojan-free netlists.

B. Comparison to an existing method

As far as we know, there are no machine-learning-based hardware-Trojan detection methods for gate-level netlists other than the one proposed in [4], where a support vector machine is used to classify the nets in a given netlist.

Table VIII summarizes the comparison results in terms of the precision and the F-measure compared to [4]. As in Table VIII, both the precision and the F-measure using our proposed Trojan features become much larger than those of the existing method. Even in the last three netlists, our results outperform the existing method. Overall, the proposed 11 Trojan features give one of the best sets of Trojan features to detect hardware Trojans. Note that the F-measure of 74.6% is a quite good value in machine learning [11].

VI. CONCLUSIONS

In this paper, we have first proposed the 51 features which describe Trojan nets from netlists. Next, we have selected the best set of 11 Trojan features which maximize the average F-measures using the random forest classifier. The experimental results show that the maximum TPR and TNR realize 100% in several benchmarks and the F-measure becomes much larger than an existing state-of-the-art method.

In the future, we will improve the TPR and TNR of individual benchmarks, not only the average F-measure values.

TABLE VIII
THE COMPARISON TO AN EXISTING METHOD.

	Precision		F-measure	
Test data	[4]	Ours	[4]	Ours
RS232-T1000	11.5%	92.3%	19.0%	96.0%
RS232-T1200	3.4%	100.0%	6.5%	93.8%
RS232-T1300	3.5%	100.0%	6.7%	100.0%
RS232-T1400	4.1%	100.0%	7.8%	98.9%
RS232-T1500	4.1%	97.4%	7.9%	96.1%
s15850-T100	2.9%	95.5%	5.7%	85.7%
s35932-T100	0.5%	100.0%	1.1%	84.6%
s35932-T300	0.4%	96.8%	0.7%	88.2%
s38417-T100	0.8%	100.0%	1.7%	50.0%
s38417-T200	0.8%	100.0%	1.5%	63.6%
s38417-T300	2.6%	100.0%	5.1%	85.7%
s38584-T100	0.3%	33.3%	0.6%	9.1%
RS232-T1100	3.1%	78.3%	5.9%	61.0%
RS232-T1600	3.5%	90.0%	6.7%	91.5%
s35932-T200	0.6%	100.0%	1.2%	15.4%
Average	2.8%	92.2%	5.2%	74.6%

REFERENCES

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

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

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

[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)*, pp. 203–206, 2016.

[5] 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)*, pp. 159–172, 2010.

[6] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, "Detection technique for hardware Trojans using machine learning in frequency domain," in *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*, pp. 185–186, 2015.

[7] 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, pp. 1137–1143, 1995.

[8] A. Kulkarni, Y. Pino, and T. Mohsenin, "SVM-based real-time hardware Trojan detection for many-core platform," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*, pp. 362–367, 2016.

[9] 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)*, pp. 465–470, 2015.

[10] 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.

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

[12] 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)*, pp. 697–708, 2013.

[13] 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)*, pp. 153–166, 2014.

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