DOI:10.1587/transinf.2024EDL8057

Publicized:2024/10/01

LETTER

FP-GNN: A Graph Neural Network for Hardware Trojan Detection in Gate-Level Netlist

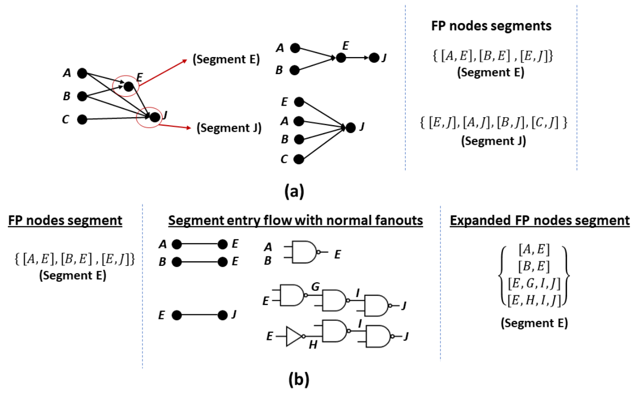
Ann Jelyn TIEMPO[†a)], ***Member and*** **Yong-Jin JEONG**[†], ***Nonmember***

SUMMARY Feature-based detection method has been widely used for hardware trojan detection where hardware trojan features are explicitly extracted and trained a classifier or build an algorithm that can differentiate hardware trojan signals from normal ones. This method shows a good performance in the available benchmark circuits. However, when tested on a randomly generated circuit, that contains different structural features of benchmark circuit, it started to fail detecting the stealthy signals, because of the constraints when the features are explicitly extracted. To overcome this situation, in this paper, the authors aim to explore the benefit of graph neural network (GNN) to enhance the hardware trojan detection performance in both benchmark circuits and randomly generated circuits. More specifically, finding the appropriate representation of the digital circuit that is suitable for the GNN model and that can learn hidden feature and relationship between the signals. Experiments show satisfactory result on benchmark circuit with an average accuracy of 95.34%. Further, tests done on randomly generated circuits gives positive result with an average accuracy of 90.82%.
key words: *hardware security, hardware trojan detection, GNN, gate-level netlist, segmentation*

1. Introduction

Hardware security has been a hot topic as a large number of participants is involving and coordinating during the design and manufacturing process of integrated circuits (ICs) which can result to possible insertion of hardware trojan on the design that can compromise the functionality and reliability of ICs. In the past years, several hardware trojan detection methods have been proposed [1][2], aiming to detect malicious circuits at different stages of the design flow. These methods can be classified into four types: (1) reverse engineering, (2) side channel analysis, (3) logic testing, and (4) circuit feature analysis. Among these methods, circuit feature analysis is more advantageous because it can do comprehensive analysis without the need to simulate the circuit.

Features are extracted by observing different characteristics of hardware trojan signals and heuristically tests and tune in the available benchmark [3][4], which demonstrates good performance. However, these features are constraints on specifics values, for examples, it assumes that the depth of hardware trojan circuit is between x-level where $x \in [1, 5]$ or hardware trojans are closer to flip-flops and multiplexers. With this, when it is tested on different circuit, that is outside the bounds of defined constraints, it missed to detect the hardware trojan signals. Further, selecting the limits should be optimal in all the data samples.


(Segment E)
(Segment J)
FP nodes segments
{ [A, E], [B, E] , [E, J]}
(Segment E)
{ [E, J], [A, J], [B, J], [C, J] }
(Segment J)
(a)
FP nodes segment
{ [A, E], [B, E] , [E, J]}
(Segment E)
Segment entry flow with normal fanouts
Expanded FP nodes segment
[A, E]
[B, E]
[E, G, I, J]
[E, H, I, J]
(Segment E)
(b)
Fig. 1 (a) Adapted FP-based segmentation. (b) Expanded FP nodes segment including the original flow of outgoing edge.

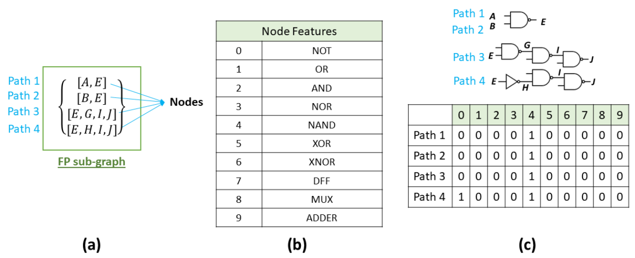
On the other hand, representing digital circuit as a graph is suitable because of its inherently unstructured format where signals can be represented as nodes and the interconnection can be represented as edges. In this paper, the authors intend to utilize the advantage of GNN to learn hidden features directly from the digital circuit graph and improve the detection performance in both benchmark circuits and randomly generated circuits. In particular, segmenting the digital circuit to FP sub-graphs and then defining the nodes and edges features as well as the link pairs. Then, train a GNN model to generate graph-level embeddings which then can be used to train a classifier to identify the hardware trojan signals from normal signals. Experiment demonstrates a positive result in both benchmark circuit and randomly generated circuits with average accuracy of 95.34% and 90.82%, respectively.

2. Methodology

2.1 Fanout Point Sub-Graph

In [5], FP-based segmentation is introduced where fanout points (FPs) are identified in the circuit and transform the circuit into FP-nodes' structure by considering the fanout points, primary inputs and primary outputs as nodes and there exists an edge if a node is reachable from another node. Then, the FP-nodes' structure is split into segments according to nodes that have head ends. In this paper, the concept of FP-based segmentation is adapted to consider both the

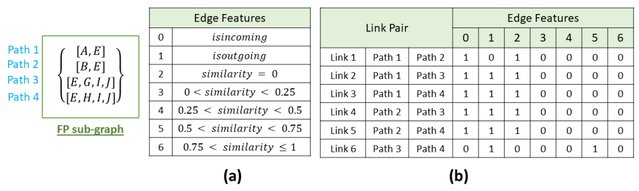
[†]The authors are with the Dept. of Electronics and Communications Engineering, Kwangwoon University, Seoul, South Korea.
a) E-mail: annjelyntiempo@yahoo.com



Node Features	
0	NOT
1	OR
2	AND
3	NOR
4	NAND
5	XOR
6	XNOR
7	DFF
8	MUX
9	ADDER

	0	1	2	3	4	5	6	7	8	9
Path 1	0	0	0	0	1	0	0	0	0	0
Path 2	0	0	0	0	1	0	0	0	0	0
Path 3	0	0	0	0	1	0	0	0	0	0
Path 4	1	0	0	0	1	0	0	0	0	0

Fig. 2 (a). Node definition. (b) Node features. (c) Node representation.


Path 1
Path 2
Path 3
Path 4
$[A,E]$
$[B,E]$
$[E,G,I,J]$
$[E,H,I,J]$
FP sub-graph
(a)
(b)

Edge Features	
0	*isincoming*
1	*isoutgoing*
2	$similarity = 0$
3	$0 < similarity < 0.25$
4	$0.25 < similarity < 0.5$
5	$0.5 < similarity < 0.75$
6	$0.75 < similarity \leq 1$

Link Pair			Edge Features						
			0	1	2	3	4	5	6
Link 1	Path 1	Path 2	1	0	1	0	0	0	0
Link 2	Path 1	Path 3	1	1	1	0	0	0	0
Link 3	Path 1	Path 4	1	1	1	0	0	0	0
Link 4	Path 2	Path 3	1	1	1	0	0	0	0
Link 5	Path 2	Path 4	1	1	1	0	0	0	0
Link 6	Path 3	Path 4	0	1	0	0	0	1	0

Fig. 3 (a) Edge features. (b) Edge representation.

incoming and outgoing signals at the head end nodes. Thus, the outgoing edges at the head ends are included in each segment as shown in Fig. 1(a). Consequently, the original flow of outgoing edges at the head end will also be expanded in the expanded FP nodes segment as shown in Fig. 1(b). These expanded segments will be treated as the FP sub-graphs and will be further represented in terms of node features, edge features and link pairs which are appropriate inputs to the GNN model.

2.2 Input Representation

The input to the GNN model is a set of node feature vectors and an adjacency matrix with vector valued entries indicating the similarities of signal paths as well as the indicator if the path is incoming or outgoing to and from head ends. In this paper, a node is defined as the signal path in each expanded FP nodes segment and each signal path within the segment is linked with each other because of the common fanout point.

In previous studies, number of features are proposed which describe the different structural features of hardware trojan, however, these features are based on a viewpoint of individual signals. Whereas in our method, the node is defined as the signal path, thus the node feature is represented in terms of discrete logic type used within the path. Ten (10) basic logic block types are used, as shown in Fig. 2, and represented by 1 if specific logic type is observed within the signal path, for example, node 4 has node representation of 1000100000 because NAND and NOT gates are present within the path.

The adjacency matrix entries are described in terms of structural similarity of linked signal paths. Structural similarity can capture how correlated are signal path to another and how a signal can affect in the signal path. In this paper, we adopt the definition of [6] of structural similarity which can be computed by looking on the common signal of two paths and is normalized by the geometric mean of the signal paths' size. Since matrix multiply message function assumes discrete edge types, the structural similarity values are one-hot encoded into five (5) value ranges as described in Fig. 3. Further, directed paths are explicitly represented by encoding if the path is an incoming or outgoing to and from head end. If each path of link pair is incoming and outgoing respectively, features 0 and 1 are set to 1, otherwise, only either will be applied.

2.3 Graph Neural Network Model

Various GNN architectures are proposed [7][8][9] to handle different graph problems. In this paper, the architecture described in [8] is adopted but using a transformer encoder and average pooling as the readout function. The architecture is consisting of three stages: (1) message passing, (2) readout and (3) classification network, as shown in Fig. 4, where message passing stage is the propagation step that computes the node representation of each node while the readout stage maps the node representations to graph-level embeddings.

The message passing stage is consists of edge network and Gated Recurrent Units (GRU) which used message function $M(h_v, h_w, e_{vw}) = A(e_{vw})h_w$ where $A(e_{vw})$ is a neural network that maps the edge vector e_{vw} to a dxd matrix. d is denoted as the dimension of the internal hidden representation at each node. This step has iterative procedure to propagate the representation where in initial state, the input node feature will be the first components of the hidden state and pads the rest with zeros. Then, in each step, it passes the information, based on the edge features, between the neighboring nodes and use the GRU to update each node's hidden state by incorporating the information from neighboring nodes and from previous timesteps. Further, weight tying is used, so the same update function is used at each timestep.

The readout model is defined per node and is a differentiable function that maps to an output but it operates on the set of nodes states instead of individual nodes. Thus, the k-step-aggregated states are partitions according to the FP sub-graph and are padded to match the FP sub-graph with the highest number of nodes and stacked together. Though some of the FP sub-graph are padded, it also masked to make sure that the paddings do not interfere with training. Then, it is passed to the transformer encoder followed by average pooling to reduced to graph-level embeddings. Finally, a two-layer classification is added to make predictions if the FP sub-graph contains hardware trojan signals or not.

3. Experimental Results

3.1 Setup

Our experiment used the available gate-level netlist provided on Trust-HUB benchmark [10] and follow a leave-one-out cross-validation [11] where the circuit under test is not included in the training sets. In addition, randomly generated

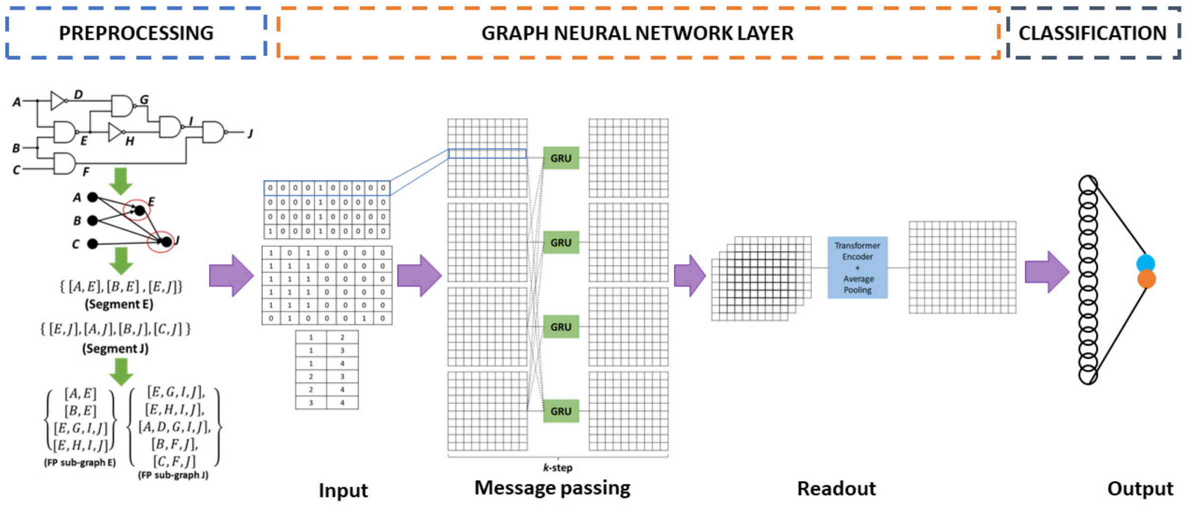

PREPROCESSING
GRAPH NEURAL NETWORK LAYER
CLASSIFICATION
{ [A, E], [B, E] , [E, J]}
(Segment E)
{ [E, J], [A, J], [B, J], [C, J] }
(Segment J)
(FP sub-graph E)
(FP sub-graph J)
GRU
Transformer Encoder + Average Pooling
k-step
Input
Message passing
Readout
Output

Fig. 4 FP-GNN model.

Table 1 Result of TrustHUB benchmark circuit on proposed detection method.

Benchmark	TP	TN	FP	FN	TPR (%)	TNR (%)	Accuracy (%)
RS232-T1000	3	202	40	0	100	83.47	91.74
RS232-T1100	12	203	39	0	100	83.88	91.94
RS232-T1200	13	208	35	0	100	85.60	92.80
RS232-T1300	9	217	25	0	100	89.67	94.83
RS232-T1400	13	204	37	0	100	84.65	92.32
RS232-T1500	14	204	39	0	100	83.95	91.98
RS232-T1600	12	212	30	0	100	87.60	93.80
s15850-T100	27	2373	33	0	100	98.63	99.31
s35932-T100	15	5557	830	0	100	87.00	93.50
s35932-T200	16	5508	975	0	100	84.96	92.48
s35932-T300	36	5391	996	0	100	84.41	92.20
s38417-T100	12	5763	26	0	100	99.55	99.78
s38417-T200	15	5732	57	0	100	99.02	99.51
s38417-T300	44	5773	16	0	100	99.72	99.86
s38584-T100	19	7180	94	0	100	98.71	99.35
s38584-T200	124	7275	2	0	100	99.97	99.99
Average					**100**	**90.67**	**95.34**

circuits are generated using the tool described in [12]. In this tool, the user can set the circuit depth, the number of gates and the number of primary inputs and outputs and will then generate a circuit that is different from the reference circuits. When generating the circuits, the reference circuits are the hardware trojan describe in the benchmark circuit. Further, randomly generated circuits are not used as training set but exclusively for test set only. With regards with data labelling, each expanded FP nodes segment is labelled as 1 if it contains hardware trojan signal, otherwise 0. Moreover, the result is evaluated in terms of True Positive Rate (TPR), True Negative Rate (TNR) and Balanced Accuracy. Lastly, the parameters such as number of batches, number of feature dimensions, number of k-steps, number of heads and dense unit are set as 32, 64, 4, 8 and 512, respectively.

3.2 Result

Table 1 shows the detection results of proposed method on benchmark circuits where average TPR is 100%, a TNR average of 90.67% and balanced accuracy of 95.34%. In Table 2, the comparison between our proposed method and previous papers is shown, where proposed method has promising result. Further, the proposed method and the previous methods are tested on randomly generated circuit where the structure of the hardware trojan is different on the benchmark circuits in terms of the circuit depth, the number of inputs to the hardware trojan circuit, the number of gates and the connectivity of signals within the hardware trojan circuit. The proposed method gives a better result than on the previous methods even without using explicitly extracted features which validate that it can learn features and relationships by representing the circuit as a graph.

Further, GNN has been explored as a detection method in recent studies [14][15][16] where hardware trojan detection is treated as a node-level problem, which means that each signal is defined as a node. On the other hand, the proposed detection method treated the hardware trojan detection problem as a graph-level task where a gate-level netlist is transformed into sub-graphs. Table 3 shows the comparison of the proposed method with the previous GNN-based detection method on benchmark circuits. The proposed detection method shows better performance in terms of TPR, or detection rate, but falls behind in terms of TNR. Since the hardware trojan detection is treated as a graph-level task, the signals within the sub-graph include hardware trojan signals with some genuine signals, however, these genuine signals are directly incorporated with the hardware trojan circuit, such as the genuine signals used as inputs to the hardware trojan circuit. Comparing with a node-level task, each node is equivalent to one signal in the design, making it independent from other signals, resulting in no direct connection

Table 2 Comparison on existing methods and performance on unknown HTs.

Detection Method	TPR(%)		TNR(%)		Balanced Accuracy(%)	
	Benchmark Circuit	Modified Circuit	Benchmark Circuit	Modified Circuit	Benchmark Circuit	Modified Circuit
[13] Unsupervised ML	42.42	0	97.25	94.41	69.83	47.21
[4] Supervised ML	64.22	19.40	99.94	100	82.08	59.70
[5] Split and Eliminate	100	26.43	98.62	99.16	99.31	62.80
FP-GNN (Proposed)	100	**87.72**	90.67	**93.92**	95.34	**90.82**

Table 3 Average performance comparison with previous GNN-based detection method on benchmark circuits.

Benchmark	TPR (%)	TNR (%)	Balanced Accuracy (%)
[16] NHTD-GL	89.04	99.95	94.50
[15] Unioned GNN	93.4	99.9	96.65
FP-GNN (Proposed)	100	90.67	95.34

of the detected signals. In the context of the ASIC design process, the detection of possible hardware trojans is further analyzed to ensure the functionality of the design, thus, analyzing a segment of connected signals gives better insights than individual nodes. Moreover, representing the graph in terms of its logic types as the node features and the structural similarities as the edge features improves the detection rate since the features are not constrained to specific values, unlike previous studies methods of representing the features. Removing the constraints on the features, the proposed detection method demonstrates good results in both benchmark circuits and randomly generated circuits.

4. Conclusion

In this paper, we proposed a HT detection method that explores the benefit of graph neural network to improve the detection performance in both benchmark circuit and randomly generated circuit where digital circuit is represented as a graph. The circuit is segmented first according to fanout points and defined the paths within the segment as the node and paths within the nodes are link with each other which then treated as link pairs. The node features are represented by the basic logic gate types used within the path and edge features are represented by the structural similarity. Experimental result shows a satisfactory performance with an average accuracy of 95.34% on benchmark circuits and an average accuracy of 90.82% on randomly generated circuit. In conclusion, GNN can be employed to learn graph embedding and used this embedding to train a classifier to detect hardware trojan which is more efficient and generalized method.

Acknowledgments

This work was supported by Kwangwoon University and by the MISP Korea, under the National Program for Excellence in SW (2017-0-00096) supervised by IITP.

References

[1] K. Liakos, G. Georgakilas, S. Moustakidis, N. Sklavos, F. Plessas, “Conventional and machine learning approaches as countermeasures against hardware trojan attacks,” Microprocess Microsys, 79 (2020).

[2] Z. Huang, Q. Wang, Y. Chen and X. Jiang, “A Survey on Machine Learning Against Hardware Trojan Attacks: Recent Advances and Challenges,” in IEEE Access, vol. 8, pp. 10796-10826, 2020, doi:10.1109/ACCESS.2020.2965016.

[3] K. Hasegawa, M. Yanagisawa and N. Togawa, ”Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier,” 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050827.

[4] T. Kurihara and N. Togawa, ”Hardware-Trojan Detection Based on the Structural Features of Trojan Circuits Using Random Forests,” IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, vol. 105-A, no. 7, pp. 1049-1060, Jul. 2022. DOI: 10.1587/transfun.2021EAP1091.

[5] A. Tiempo, Y. Jeong, “Split and Eliminate: A Region-Based Segmentation for Hardware Trojan Detection”, in IEICE Transactions on Information and Systems, vol. E106-D, no. 3, pp. 349-356, Mar. 2023. DOI: 10.1587/transinf.2022EDP7169.

[6] X. Xu, N. Yuruk, Z. Feng, T.A.J. Schweiger, “SCAN: a structural clustering algorithm for networks,” in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2007, pp. 824–833, https://doi.org/10.1145/1281192.1281280.

[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in International Conference on Learning Representations, 2019.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in Proceedings of the 34th International Conference on Machine Learning - Volume 70, 2017, p. 1263–1272

[9] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and ‘Y. Bengio, “Graph attention networks,” in International Conference on Learning Representations, ICLR, 2018.

[10] M. Tehranipoor, D. Forte, Trust-Hub, Aug. 2016, [online] Available: https://www.trust-hub.org/benchmarks.php.

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

[12] S. Amir and D. Forte, ”EigenCircuit: Divergent Synthetic Benchmark Generation for Hardware Security Using PCA and Linear Programming,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 12, pp. 5207-5219, Dec. 2022, doi: 10.1109/TCAD.2022.3166675.

[13] C. Dong, Y. Liu, J. Chen, X. Liu, W. Guo and Y. Chen, “An Unsupervised Detection Approach for Hardware Trojans,” in IEEE Access, vol. 8, pp. 158169-158183, 2020, doi: 10.1109/ACCESS.2020.3001239.

[14] R. Yasaei, L. Chen, S. -Y. Yu, and M. A. A. Faruque, ”Hardware Trojan Detection using Graph Neural Networks,” IEEE Trans.

Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/TCAD.2022.3178355.

[15] W. Pan, M. Dong, C. Wen, H. Liu, S. Zhang, B. Shi, Z. Di, Z. Qiu, Y. Gao, and L. Zheng, ”A unioned graph neural network based hardware Trojan node detection,” IEICE Electronics Express, vol. 20, no. 13, pp. x-x, 2023. doi: 10.1587/elex.20.20230204.

[16] K. Hasegawa, K. Yamashita, S. Hidano, K. Fukushima, K. Hashimoto, and N. Togawa, ”Node-wise Hardware Trojan Detection Based on Graph Learning,” IEEE Trans. Computers, doi: 10.1109/TC.2023.3280134.