HLIFT: A High-level Information Flow Tracking Method for Detecting Hardware Trojans

Chenguang Wang, Yici Cai and Qiang Zhou
Tsinghua National Laboratory for Information Science & Technology
Department of Computer Science & Technology, Tsinghua University, Beijing, China
Email: wang-cg13@mails.tsinghua.edu.cn, {caiyc, zhouqiang}@mail.tsinghua.edu.cn

Abstract— In this paper, we note that the hardware Trojans that leak information through the *unspecified output pins* are difficult to detect by functional testing or side-channel signal analysis. To solve this problem, we propose a feature matching method based on information flow tracking at high abstraction level. Experimental results show that our method can successfully identify the above-mentioned Trojans from Trust-hub, DeTrust, and OpenCores in less than 20 *ms*, showing significantly lower time complexity compared with the existing works.

I. Introduction

Hardware Trojans are malicious modifications that are inserted into the third-party intellectual property (3PIP) cores by the rogue elements. Such modifications can give rise to undesired functional behavior or provide covert channels and backdoors through which the information can be leaked. It has become a major concern of system-on-chip (SoC) designers with the globalization of hardware supply chain. In response, much work has been done to detect Trojans in pre- and post-silicon phases, e.g., functional testing, side-channel signal analysis, formal proof of security properties, and code analysis [1].

Many taxonomies for hardware Trojans have been proposed [2] [3], which usually categorize Trojans based on the insertion phase, abstraction level, trigger mechanism, and malicious effect (payload), etc. In terms of payload, most of the existing Trojans can be divided into the following categories:

1) The logic functions of design signals are modified, which violates the design specification.
2) The logic functions of design signals with unspecified behavior are modified, without violating the specification.
3) The information is leaked through specified output pins or side-channel, without modifying the logic functions.

As known to us, much work has been done to detect the Trojans in Category 1 and 3. Category 2 is introduced in [4] and several methods have been proposed to prevent and detect this Trojan type [5] [6]. In this paper, we focus on a fourth Trojan type that is less studied and more difficult to detect:

This project is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61774091.

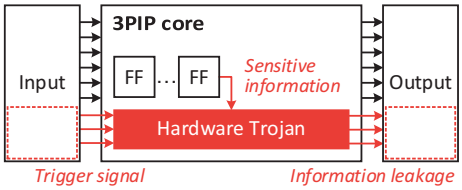

Fig. 1. The information flow model of a Trojan in Category 4. The sensitive information is stored in the flip-flop (FF) and leaked through the unspecified output pins, while the trigger signal is transmitted through the input pins.

4) The information is leaked through the unspecified output pins, without modifying the logic functions.

Fig.1 presents a sample Trojan in Category 4. The difference between Categories 3 and 4 is that the Trojans in 4 leverage the **unspecified output pins** to leak information **even without altering the monitored side-channel signals**. For example, the output of a true random number generator (TRNG) cannot be predicted and specified, thus a Trojan leaking information via the output ports in TRNG can be classified into Category 4. As a comparison, the same Trojan in a pseudo random number generator (PRNG) is classified into Category 3.

As a result, the Trojans in Category 4 evade functional testing or side-channel signal analysis. Further, they cannot be captured even if an advanced equivalence checker is used to check if the design conforms to a golden reference (if any) because the values of leaking ports are unspecified. Besides, the security properties listed in [7], used to model check the existence of Trojans in Category 3, also fail to capture the Trojans in 4 because they don't use the output ports as the Trojans in 3.

As a step toward the solution, we propose a feature matching method based on high-level information flow tracking (HLIFT) to **identify the Trojans in Categories 3 and 4**, with the basic concept borrowed from malware detection [8] in Fig.2. Information flow tracking (IFT) has been used as a powerful technique to measure and control the logical flows from Boolean gates [9]. IFT is here applied to capturing the commonality of Trojans at the register transfer level (RTL). Considering the increasing design scales, we propose a high-level format for depicting the IFT features, i.e. statement-level control and data flow graph (CDFG), to reduce the time complexity.

978-1-5090-0602-1/18/$31.00 ©2018 IEEE

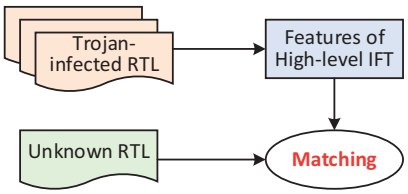

Fig. 2. The basic concept of the proposed feature matching method based on high-level information flow tracking. This concept is borrowed from pattern matching [8], which is an effective approach in malware detection area.

To validate the proposed method, we detect the benchmarks which are derived or modified from Trust-hub [10], DeTrust [11], and OpenCores [12]. The final results show that our technique is able to detect all the benchmarks with 0 false and positive negatives, and the time consumptions are significantly lower compared with the existing works.

To summarize, the contributions of this paper include:

1. We formally define a new Trojan type that leaks sensitive information through the unspecified output pins, which is stealthier and more difficult to detect.
2. To detect this Trojans type, we propose a feature matching methodology based on high-level information flow tracking techniques to capture the specific Trojan features.
3. A compact and high-level abstraction format for depicting the IFT features in RTL, i.e. the statement-level CDFG, is introduced to reduce the time complexity.

The remainder of this paper is organized as follows. Section II provides the preliminary. The HLIFT method is introduced in Section III. Section IV provides the experimental results. The conclusion and discussion are summarized in Section V.

II. Preliminary

A. Hardware Trojans Detection

The existing hardware Trojans detection techniques can be broadly classified into (a) Functional testing, (b) Side-channel signal analysis, (c) Formal proof of security properties, and (d) Code analysis based on IFT [1].

The methods in (a) may fail to detect the Trojans that do not violate the design specification, but only alter the logic functions of design signals which have unspecified behavior, e.g. the RTL don't cares [5]. Methods in (b) attempt to capture Trojans using the side-channel signals, e.g. power and path delay, because the signals can be raised by the insertion of Trojans. However, the signals are usually introduced into noise with the increasing process variation and decreasing size of Trojan circuits. Although formal methods in (c) have proven effective, the security properties are time-consuming and error-prone to write, which is a challenging task for the SoC designers who lack security expertise. Further, most formal methods do not provide clues that reveal Trojan behavior, so they are not suitable for checking an entire design.

The existing behavior and structural code analysis methods in (d) are mostly based on various levels of IFT. Hicks *et al.* propose a method named unused circuit identification (UCI) to identify suspicious circuitry by finding equivalent signal tuples [13]. Besides, Zhang *et al.* propose the VeriTrust to detect Trojans activated by complex patterns [14]. FANCI is introduced by Waksman *et al.* based on static Boolean functional analysis [15]. Fern *et al.* propose a method to detect the Trojans hidden in RTL don't cares [5]. Further, Fern *et al.* propose to precisely define "suspicious" unspecified functionality and the formulate the detection as a satisfiability problem [6]. A feature analysis method on the flip-flop level CDFG (FASTrust) is introduced for identifying the malicious circuits [16]. Hu *et al.* propose a gate-level IFT based method (GLIFT) to measure all logical flows from Boolean gates and then verify that an information flow adheres to security properties related to the confidentiality and integrity [17].

Besides, there exist methods for detecting Trojans which tag assets such as cryptographic key bits in the design, then use formal methods, e.g., equivalence checking [6], model checking [7] [18], and theorem proving [19] [20], to analyze how these bits propagate through the design. These methods are based on Category (c) and (d) in a combinational manner.

To summarize, the closest works in spirit are FASTrust [16] and GLIFT [17], whereas the key difference is that we take the stealthy Trojans in Category 4 that evade the existing techniques into consideration. Further, these methods are applied to the gate-level netlist and thus harder to eliminate the Trojans compared with HLIFT that checks the RTL code.

B. Motivation

It has been noted that the Trojans in Category 4 can never be detected by functional simulation or side-channel signal analysis, and formal methods also have limitations to capture the security properties with which the Trojans can be identified.

Since the concept of hardware Trojans is borrowed from software Trojans, it is worth investigating the methods in malware detection. The basic approach in virus scanners is pattern matching, i.e., a program is declared as malware if it contains a sequence of instructions that is matched by a regular expression [8]. For each instance of the virus, a unique pattern is extracted. Then, the program is checked if there is a sequence of data exactly matches the regular expressions.

Feature matching, which has proven effective for detecting malware, is expected to counter the hardware Trojans. For various Trojan types, we can extract unique features based on the information flow tracking techniques, which can be used to distinguish them from the genuine designs.

On the other hand, the ideal strategy of Trojans detection is to catch potential vulnerabilities that could signal a Trojan during the early design phase, when they are much easier to eliminate or mitigate [17], thus we address the Trojans in RTL code. Besides, with the increasing scales of SoC designs, the proposed method is expected to have a trade-off between time consumption and the detection ability for large designs by the introduced high-level abstraction techniques.

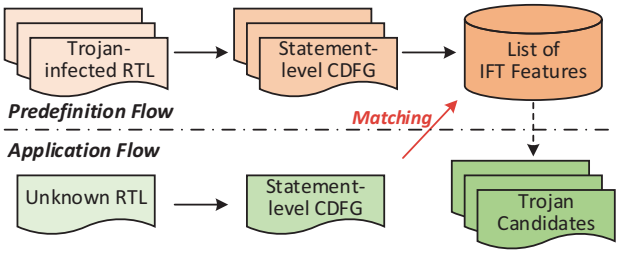

Fig. 3. The flow chart of HLIFT.

C. Threat Model

The threat model follows the commonly used one [1]. The rogue designer in a 3PIP vendor is the attacker, whose purpose is to insert Trojans into the 3PIP cores. The 3PIP consumer is the defender, whose objective is to detect the Trojans (if any) in the 3PIP cores. We assume that the 3PIP cores under detection are delivered from the attacker to defender as the RTL code in VHDL or Verilog HDL formats without trustworthiness. Besides, we assume that both the detection procedure conducted by the defender and the used commercial tools are trustworthy.

III. Methodology of HLIFT

In this section, the proposed methodology HLIFT is introduced in detail. After an overview, the building of statement-level CDFG is presented. Then, the features of IFT at the statement-level CDFG and the corresponding feature analysis are introduced. Finally, the entire HLIFT flows are presented by a case study on the information-leaking Trojan in an AES-128 encryption engine.

A. Overview

Fig.2 provides a high-level overview of the proposed framework, and the detailed one is presented in Fig.3. HLIFT consists of two consecutive steps, i.e., the **Predefinition** and **Application** flows. First, the statement-level CDFG corresponding to each Trojan-infected RTL design is built. Then, the features of IFT depicted in the statement-level CDFG formats are listed. In the Application flow, the statement-level CDFG extracted from unknown RTL is checked whether any samples match the predefined IFT features. If any, the Trojans can be identified and located in the RTL.

B. High-level Information Flow Tracking

B.1 Statement-level Control and Data Flow Graph

To reduce the computational complexity, we propose a high-level and compact abstraction of RTL netlist, i.e. statement-level CDFG. In the CDFG, **the low-level information is unnecessary for depicting the IFT features and thus removed**, while the statement- and higher-level information is reserved. As a result, the behavior space of RTL can be significantly reduced, with the time consumption also reduced.

In general, the information flow of RTL can be tracked by CDFG, which is commonly used as an internal representation in high-level synthesis. However, some of the information flow is unnecessary for depicting the Trojan features. For example, for a single statement, it is difficult to distinguish the Trojan-infected from the genuine because they share a similar set of operations, e.g. add and shift. However, for a set of statements, we can identify the Trojan-infected ones by some features, e.g., the trigger has lower transition than genuine signals. Further, it has been noted that a high-level CDFG is able to capture the Trojans features on information flow [16].

Based on the observation, we analyze the information flows at the level of RTL modules and basic blocks, including procedure, if/case block, loop block, and statement. It can be noted that the statement-level CDFG is more compact than CDFG for tracking the information flow. Formally, the statement-level CDFG is a directed acyclic graph as $G = (V, E)$, where V is the set of nodes which can be either module nodes or basic block nodes, and $E \subseteq V * V$ is the set of edges representing the transfer of either data or control from one node to another, which is presented with a sample in Fig.4.

B.2 IFT Features by Statement-level CDFG

In this section, we investigate and summarize the features of the Trojans in Categories 3 and 4 by information flow tracking techniques performed in the format of statement-level CDFG. The used terminologies are defined as follows:

- *Genuine Nodes* (N_g) are where the normal function is performed, e.g. the encryption modules in AES.
- *Genuine Data* (D_g) is used to perform the normal function of N_g, e.g. the encryption key (KEY).
- *Trojan Nodes* (N_t) are the payload and trigger of Trojans, e.g. the key-leaking modules in AES.
- *Trojan Data* (D_t) is used to perform the malicious effects of N_t, e.g. the trigger signal (COUNTER).

For the data flow, D_g can be used by both N_g and N_t as input/output because the system signals and sensitive data are also required for the malicious effects, e.g. KEY. On the contrary, D_t is only used by N_t because the Trojan-infected data, e.g. COUNTER, is not required for the normal functions. To hide the malicious operations, D_g is hardly altered for it can be read but not written by N_t. The above features can be formally expressed as (1a) - (1f), where I/O are respectively the set of input/output patterns, and D'/D'' are respectively the value of D (D_g or D_t) before/after being called by the blocks:

$$\exists D_g \in I \cup O \subset N_g \models D_g \in I \cup O \subset N_t \quad (1a)$$
$$\nexists D_t \in I \cup O \subset N_t \models D_t \in I \cup O \subset N_g \quad (1b)$$
$$\exists D_g \in I \cup O \subset N_t \models D_g'' \neq D_g' \quad (1c)$$
$$\forall D_t \in I \cup O \subset N_t \models D_t'' = D_t' \quad (1d)$$
$$\exists D_g \in I \subset N_g \models D_g \in I \subset N_t \quad (1e)$$
$$\exists D_g \in O \subset N_g \models D_g \in O \subset N_t \quad (1f)$$

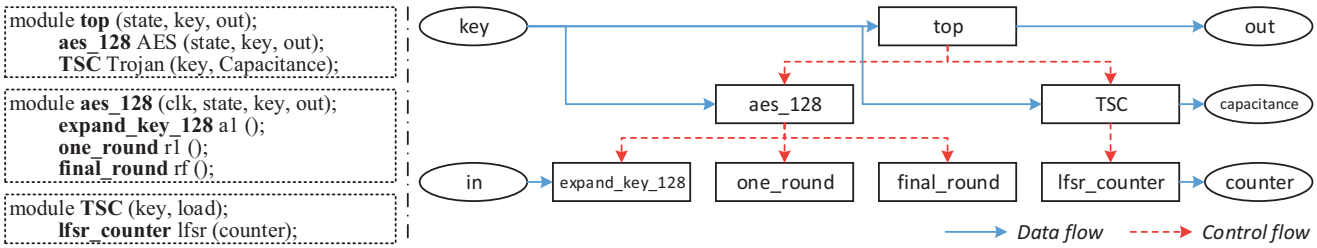

Fig. 4. Sample RTL of Trojan-infected AES-T100 from Trust-hub [10] and the corresponding statement-level CDFG, where the node (N_g or N_t) is represented as a rectangle and the data (D_g or D_t) is represented as an ellipse.

For the control flow, it can be noted that there exists no N_t which is called by N_g because the control flow is based on the data flow and there exists no D_t used by N_g as input or output. Likewise, there exists N_g which is called by N_t. To summarize, the normal function can be performed in both N_g and N_t, while the malicious effect is performed only in N_t. The above features can be formally expressed as follows:

$$\forall N_g \nexists N_t \models N_t \; is \; called \; by \; N_g \tag{2a}$$

$$\exists N_t \; \exists N_g \models N_g \; is \; called \; by \; N_t \tag{2b}$$

Equations (1a) - (1f) and (2a) - (2b) present the theoretical foundations of our proposed methodology HLIFT, i.e. the formulated **features of Category 3 & 4**. Nest, the key issue to be considered is that, how can we identify and locate N_g, D_g, N_t, and D_t in a statement-level CDFG of Category 3 & 4?

To systemically solve this issue, we summarize the features by the trigger types as follows. I) The *always-on* Trojans function without a trigger. II) The *immediate-on* Trojans are activated when observing a predefined event in one clock cycle, e.g. a special input. III) The *sequential-on* Trojans are triggered when a sequence of predefined events in multiple clock cycles occur, e.g., when the adding operation has been executed for predefined times.

- **Feature I**: The control path (N_t) has exclusive data (D_t) as only output but not input.

We note ***control path*** as a sequence of nodes from one to leaf and ***exclusive data*** as I/O that is only used by the nodes in control path. By (2a) - (2b), there must exist a path consisting of only N_t. By (1a), D_g may be input of both N_g and N_t. Besides, by (1d) there is no trigger but only payload, and D_t is used for the payload. Feature I is shown in Fig.5.

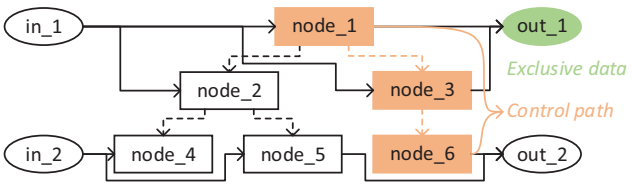

Fig. 5. Statement-level CDFG of Feature I, where the Trojan-infected node is represented as a red rectangle and Trojan data as a red ellipse.

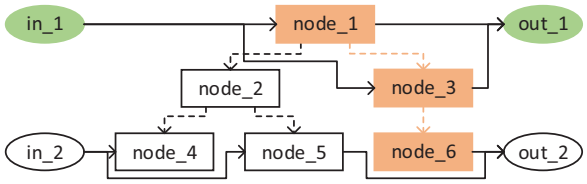

Fig. 6. Statement-level CDFG of Feature II.

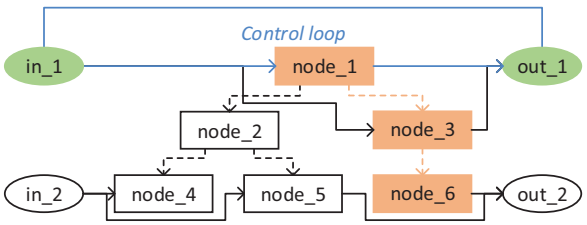

Fig. 7. Statement-level CDFG of Feature III.

- **Feature II**: The control path (N_t) has exclusive data (D_t) as both input and output.

The payload of immediate-on Trojans is activated by a trigger signal which is immediately enabled when the predefined event is observed. As a result, the outputs of N_t are the same with Feature I by (1b) - (1c). On the other hand, there must exist trigger data which may be altered in N_t according to (1d), and this trigger signal is noted as the *exclusive **input** data*. Feature II is illustrated in Fig.6.

- **Feature III**: The control loop (N_t) has exclusive data (both D_g and D_t) as either input or output.

We note the ***control loop*** as a combinational loop of nodes and data in the statement-level CDFG.

For the sequential-on Trojans type, it can be noted that the state signal will be frequently altered by N_t because the payload is activated by a state signal which is enabled when the predefined sequence of events occurs. By (1c) and (1d), there exists D_t which is both input and output of N_t. Besides, there exists a signal D_g which acts as input of both N_g and N_t by (1e) and (1f), generating a control loop consisting of N_g, N_t and D_g. This signal D_g is also noted as the *exclusive **input** data*. Feature III is shown in Fig.7.

TABLE I
FEATURES OF TROJANS IN CATEGORIES 3 & 4

#	Feature	Identification Method
I	A path in control flow has exclusive data as output	N_t: the nodes in control path or loop that have exclusive output N_g: the other nodes except for N_t D_t: the exclusive I/O data of N_t D_g: the other data except for D_t
II	A path in control flow has exclusive data as input/output	
III	A loop in control flow has exclusive data as input/output	

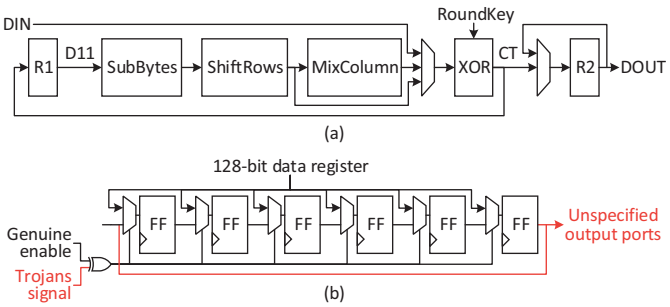

Fig. 8. (a) Data path of AES encryption. (b) AES-T100* Trojan that leaks the data stored in the register through the unspecified output ports.

Table I concludes the information-flow features of the Trojans in Categories 3 & 4 based on the trigger types. Given a statement-level CDFG of unknown RTL, N_g, D_g, N_t, and D_t can be identified by the methods in Table I.

C. Case Study: Implementation and Identification of an AES Trojan in Category 4

By modifying the information-leaking path of AES-T100 from Trust-hub [10], we update an AES engine (AES-T100*) that contains a Trojan in Category 4. The data stored in the register are transmitted through the output ports of the module table_lookup for MixColumn operation, which have unspecified values when table_lookup is not called by module one_round for Round operation. The data path of AES encryption and the implemented Trojan are presented in Fig.8.

In the Predefinition step, a list of IFT features is built. Next, in the Application step, AES-T100* is extracted as a statement-level CDFG to check if any samples match the predefined IFT features. As is expected, it is successfully detected by HLIFT, with the final results presented in Section IV.

IV. EXPERIMENTAL RESULTS

To verify the effectiveness and efficiency of the HLIFT technique, we check both Trojan-infected (Category 3) and genuine benchmarks from Trust-hub [10] and some other Trojan-infected designs proposed by DeTrust [11]. We also update the genuine RTL from OpenCores [12] with the strategy of Trust-hub in mind. For Category 4, there are no off-the-shelf benchmarks available, so we design several by the aforementioned methods (labeled with "*" in Table II). HLIFT is implemented in C++ language and executed on a Linux server with Intel Xeon E5604 CPU @ 2.67 GHz and 64GB RAM.

TABLE II
DETECTION CAPABILITY OF HLIFT ON THE BENCHMARKS FROM TRUST-HUB [10], DETRUST [11], AND OPENCORES [12]

Benchmark[a]	Category (IFT Feature)	# Trojan Nodes (FP)	# Genuine Nodes (FN)	Runtime (ms)
AES-T100	Category 3 (I)	12 (0)	19 (0)	13.871
AES-T200		13 (0)	20 (0)	13.262
OR1200_ALU[11]		13 (0)	24 (0)	2.149
AES-T400	Category 3 (II)	17 (0)	25 (0)	13.786
AES-T600		12 (0)	18 (0)	13.221
AES-T1300		11 (0)	23 (0)	13.664
BasicRSA-T100		11 (0)	23 (0)	3.709
OR1200_WBMUX[11]		13 (0)	27 (0)	2.846
UART[12]		12 (0)	31 (0)	16.258
AES-T800	Category 3 (III)	14 (0)	22 (0)	13.690
AES-T1600		17 (0)	27 (0)	13.437
BasicRSA-T300		12 (0)	20 (0)	3.686
PIC16F84		12 (0)	19 (0)	2.285
OR1200_CTRL[11]		11 (0)	27 (0)	2.061
RS232[12]		14 (0)	21 (0)	3.692
AES-T100*	**Category 4**[b] (I, II, and III)	15 (0)	19 (0)	13.935
AES-T600*		14 (0)	18 (0)	13.371
BasicRSA-T100*		14 (0)	23 (0)	3.806
BasicRSA-T300*		15 (0)	20 (0)	3.957
UART*		14 (0)	31 (0)	16.693
OR1200_CTRL*		13 (0)	27 (0)	2.267

[a] The benchmarks in Table II are all Trojan-infected. The same experiment on the genuine ones shows that HLIFT can detect the RTL with 0 false positives.
[b] The Trojans in Category 4 are implemented by modifying the leaking ports of the corresponding Trojans in Category 3, which is introduced in *Case Study*.

A. Detection Ability

Table II provides the results, where the first column lists the name of benchmarks, and the second column indicates the Trojan Category with IFT features. The next two columns present the number of statement-level CDFG nodes identified as Trojan and genuine respectively, and the last presents the runtime. Further, false positive (FP), the number of nodes wrongly identified as Trojan nodes, and false negative (FN), the number of nodes wrongly identified as genuine nodes, are reported.

While the benchmarks have various sizes and contain Trojans with different functions and features, all Trojans in Categories 3 & 4 can be successfully detected with 0 false negatives. Moreover, the same experiments on the corresponding

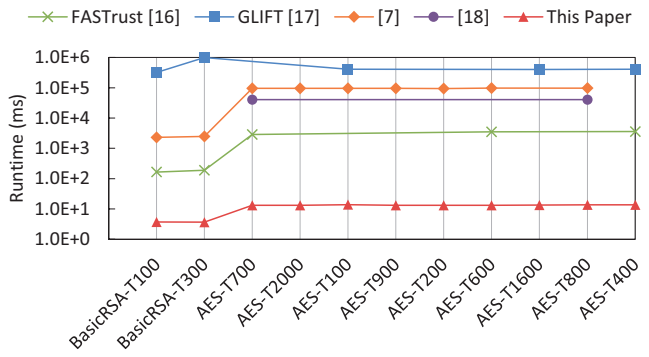

Fig. 9. Comparison of the runtime of different Trojan detection techniques.

genuine benchmarks show 0 false positives. Besides, as we can see, HLIFT has taken less than 20 *ms* in the worst case, which validates the low computational complexity of HLIFT.

B. Comparison with Existing Works

For the Trojans in Category 4, as analyzed above, they evade the existing techniques because the Trojans exploit the unspecified output ports to leak information, which cannot be captured even with advanced sequential equivalence checkers. The results demonstrate the significant advantages of HLIFT.

For Trojans in Category 3, we perform a comparison of time complexity with the closest works, FASTrust [16] and GLIFT [17], as well as the techniques which are based on the listed properties of IFT [7] [18], as shown in Fig.9. As we can see, the runtime of HLIFT is at least 2 orders of magnitude lower than the existing works, which can be contributed to the proposed high-level formats for tracking the information flow.

V. Conclusion and Discussion

In this paper, we formally define a new Trojan type that leaks information through unspecified output pins, which are stealthier and more difficult to capture. To counter it, we propose a feature matching method based on information flow tracking. A compact abstraction format for depicting the IFT features is proposed to remove the redundant information in CDFG. The HLIFT technique has the advantages as follows. (1) It is the first to provide a detection mechanism for the Trojans in Category 4 that evade the existing techniques. (2) It has at least 2 orders of magnitude lower runtime, which can be contributed to the proposed statement-level CDFG. (3) To the best of our knowledge, this is the first work using IFT techniques to check RTL, making it easier to eliminate and mitigate the Trojan circuitry compared with the gate-level countermeasures.

In the future, we plan to investigate the trade-off between the degree of information flows preserved in the high-level format and the protection level achieved to enhance the information flow model by improving its accuracy. Another future research would be to theoretically quantify the capability that an IFT feature can distinguish Trojan with genuine samples to provide an accurate score-based trust evaluation of 3PIP cores.

References

[1] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Design Autom. Electr. Syst.*, vol. 22, no. 1, pp. 6:1–6:23, 2016.

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

[3] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[4] C. Dunbar and G. Qu, "Designing trusted embedded systems from finite state machines," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 5s, pp. 153:1–153:20, 2014.

[5] N. Fern, S. Kulkarni, and K. T. Cheng, "Hardware trojans hidden in RTL don't cares - automated insertion and prevention methodologies," in *IEEE International Test Conference, ITC*, 2015, pp. 1–8.

[6] N. Fern, I. San, and K. T. Cheng, "Detecting hardware trojans in unspecified functionality through solving satisfiability problems," in *Asia and South Pacific Design Automation Conference, ASP-DAC*, 2017, pp. 598–504.

[7] J. Rajendran, A. M. Dhandayuthapany, V. Vedula, and R. Karri, "Formal security verification of third party intellectual property cores for information leakage," in *International Conference on VLSI Design, VLSID*, 2016, pp. 547–552.

[8] M. Christodorescu, S. Jha, S. A. Seshia, D. X. Song, and R. E. Bryant, "Semantics-aware malware detection," in *IEEE Symposium on Security and Privacy, S&P*, 2005, pp. 32–46.

[9] M. Tiwari, H. M. G. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*, 2009, pp. 109–120.

[10] Trust-hub. http://www.trust-hub.org.

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

[12] OpenCores. http://www.opencores.org/.

[13] 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 *IEEE Symposium on Security and Privacy, S&P*, 2010, pp. 159–172.

[14] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for hardware trust," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, 2015.

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

[16] S. Yao, X. Chen, J. Zhang, Q. Liu, J. Wang, Q. Xu, Y. Wang, and H. Yang, "FASTrust: Feature analysis for third-party IP trust verification," in *IEEE International Test Conference, ITC*, 2015, pp. 1–10.

[17] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting hardware trojans with gate-level information-flow tracking," *IEEE Computer*, vol. 49, no. 8, pp. 44–52, 2016.

[18] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Design Automation Conference, DAC*, 2015, pp. 112:1–112:6.

[19] Y. Jin and Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *IEEE VLSI Test Symposium, VTS*, 2012, pp. 252–257.

[20] Y. Jin, X. Guo, R. G. Dutta, M. M. Bidmeshki, and Y. Makris, "Data secrecy protection through information flow tracking in proof-carrying hardware IP part I: framework fundamentals," *IEEE Trans. Information Forensics and Security*, vol. PP, no. 99, pp. 1–1, 2017.