Efficient Triggering of Trojan Hardware Logic

Artemios G. Voyiatzis*†, Kyriakos G. Stefanidis†, Paris Kitsos‡†
* SBA Research, Vienna, Austria
avoyiatzis@sba-research.org
† Industrial Systems Institute, "Athena" RIC, Platani Patras, Greece
stefanidis@isi.gr
‡ Computer Informatics and Engineering Department, TEI of Western Greece, Antirrion, Greece
pkitsos@ieee.org

***Abstract*—The detection of malicious hardware logic (hardware Trojan) requires test patterns that succeed in exciting the malicious logic part. Testing of all possible input patterns is often prohibitively expensive. As an alternative, we explored previously the applicability of the combinatorial testing principles.**

In this paper, we turn our focus on the efficiency of this approach for triggering the hidden malicious logic. We present a series of experiments with Trojan designs of various activation patterns and lengths that target a cryptographic module performing AES cryptography. Our findings indicate that the available test suites succeed in triggering the malicious logic in all cases requiring only a very small number of test vectors. Thus, it is an efficient means for detecting malicious hardware logic.

I. Introduction

The reliance on digital supply chains of unprecedented depth and spread around the globe raises concerns on the trustworthiness of complex (or even simple) embedded computing systems [1]. The supply chain threat is already recognized in the USA [2], [3]. The European Union, a pioneer in embedded systems, is now also heavily depending on (possibly untrusted) foreign ICT components and parts [4].

The hardware development lifecycle, from requirements analysis to fabrication of the integrated circuits or configuration of the FPGAs, involves many stages that are performed by different parties. Thus, it is possible that a malevolent actor (insider or outsider) injects malicious functionality at some point. The terms "malicious hardware logic" and "hardware Trojan horses" (or simply "Trojans") are commonly used to describe such functionality implemented in hardware [5].

The malicious functionality must be detected and removed at the earliest possible. A process of secure development and rigorous testing at each production stage, such as the one proposed in [6], can reveal manipulations before fabrication.

If the malicious logic becomes part of the fabricated circuit or the configured FPGA, the detection of a hardware Trojan requires testing methods that go beyond the established practice [7]. A black-box approach should be employed, as one must assume and test for the existence of some Trojan functionality hidden in the circuit. The reduction of the number of performed tests is an important parameter to consider when designing a test strategy for hardware Trojan detection.

Combinatorial testing (CT) is an efficient means for functional testing of software components where their internals are considered as a black box [8]. CT-based approaches utilize theoretical results from combinatorics, can compact significantly the test suite size (i.e., the number of different test inputs applied) under specific assumptions, and (in contrast to random-based tests) provide mathematical guarantees for the coverage of the assumed search space.

We explored the use of combinatorial testing for the case of hardware Trojan detection by drawing the analogy with black-box software testing in [9]. There, we demonstrated the *applicability* of the technique for only one specific hardware Trojan case. Here, we explore further this approach and study its *efficiency* when applied for a wide range of Trojans against a hardware implementation of the AES cryptographic algorithm and in comparison with random-based testing.

The rest of the paper is organized as follows. Section II provides background information on hardware Trojan detection techniques, including the applicability of the combinatorial testing. Section III describes the set of the experiments we designed. Section IV discusses the results of the experiments and the detection efficiency. Finally, Section V concludes our paper and presents some future directions of work.

II. Hardware Trojan detection

A. Trojan components and operation

A hardware Trojan comprises, in general, two parts: the *trigger* and the *payload*. The trigger part is often considered to be an always-on circuitry. It monitors the operation of the contaminated system for the proper activation sequence to occur. This can be a physical event (e.g., a switch press or a temperature increase) or a digital one (e.g., a specific bit pattern on selected inputs). Once the trigger logic detects the correct sequence, it activates the payload part. The payload part delivers the malicious functionality (e.g., injecting faults in cryptographic computations aiding fault-based cryptanalysis [10]).

It is important to highlight the difference between Trojan *triggering* and Trojan *excitation*. The latter refers to the operation of the contaminated part of a hardware design. If the trigger part is not always on or if the triggering logic is complex enough, some inputs might partially or fully *excite* the Trojan circuitry albeit without triggering the Trojan payload (Trojan activation). Based on this observation, defenses based on run-time monitoring, such as the ones described in [11], can further assist into detecting the presence of a Trojan without activating its payload.

B. Attack model

It is desirable, from an attacker point of view, for the Trojan to be *controllable* and *observable*. The former relates to the ability to control the activation of the Trojan through some external input. The latter relates to the ability to observe the outcome of the Trojan activation at the output.

There is an upper limit of triggering complexity that an attacker can implement. This relates to the number of gates used, the degree of changes in the physical characteristics of the circuit, and the number of input signals that can be combined for realizing the triggering logic. Beyond this limit, the conventional design automation tools and established testing practices can reveal easily the presence of additional logic [12]. Below this, we need improved testing approaches.

We assume in our attack model that the attackers opt for a rather rare combination of a small number of input signal values (pattern) for exciting and activating their Trojans, while the triggering logic is realized using a handful of gates in order to remain as stealthy as possible. This assumption is based on the observation that the models used by the design automation tools do not capture the behavior of a Trojan and cannot exhaustively test all possible inputs in a realistic timeframe. Yet, a less rare pattern is more probable to be applied during the analysis and thus, reveal the presence of the Trojan.

C. Testing for Trojan presence based on signal rareness

The assumption of rareness is explored in many Trojan detection proposals. The objective is to reduce the size of the set of different inputs applied (i.e., the size of the *test suite* comprising *test vectors*, while increasing the confidence level that the system under test (SUT) is Trojan-free. This is beneficial as it reduces the test time per SUT (faster batch testing) and the stress of continuous operation (aging effect avoidance).

Automatic Test Pattern Generation (ATPG) could be an initial approach; however it tends to produce a large number of test vectors [13], [14]. Some initial approaches (e.g., MERO [15]) focus on analyzing the hardware design and guiding the test vector generation towards rare patterns, assuming that the attacker would do the same [16], [17], [18].

Attempts to introduce testability signals in the designs may result in Trojan designs that incorporate these signals as a notification for remaining silent when tested [18], [19]. This is an established technique used by malicious software authors to defend against malware analysis platforms [20], [21]. As design analysis becomes more and more complex and time consuming, probabilistic generation of test vectors is also proposed [22].

D. Testing based on search space coverage

The assumption of rareness is challenged for practical reasons (time complexity to derive an appropriate test suite, as discussed in [22]) as well as because rare triggering conditions can be constructed by *combining* conditions that are not so rare [23]. It is beneficial to shift the focus towards providing metrics of the coverage of the input vector space.

TABLE I
TEST SUITE STRENGTH AND SEARCH SPACE REDUCTION

Strength	Suite size	Covered patterns
$t = 2$	11	32,512
$t = 3$	37	2,731,008
$t = 4$	112	170,688,000
$t = 5$	252	8,466,124,800
$t = 6$	720	347,111,116,800
$t = 7$	2,462	12,099,301,785,600
$t = 8$	17,544	366,003,879,014,400

The attacker needs to be able to control the Trojan operation and remain stealthy, as explained in Section II-B. The new assumption is that the attacker is self-limited to use only k of the available n input signals, where $k << n$, as to define the activation sequence. In this case, a test suite comprising all the $2^k \times \binom{n}{k}$ possible input signal combinations will reveal the presence of the Trojan. A first approach towards generating test suites that cover all the k subspaces is provided in [13]. However, the approach generates rather large test suites.

E. Combinatorial testing

We proposed a better approach in [9]. The approach is based on the combinatorial testing principles and utilizes the mathematical construct of covering arrays [8].

Assume that a Trojan targetting a cryptographic algorithm implementation is controlled using as input signals some $k = 2$ out of the $n = 128$ possible plaintext bits. There are $32,512$ possible combinations (patterns) to check. In this case, the CT-based approach succeeded in compacting this search space into just 11 vectors (i.e., test vectors). Thus, by applying the test suite comprising these 11 test vectors, one is assured that all possible activation patterns of length $k = 2$ were applied. This is denoted as the test suite having a strength $t = 2$. Similar test suite size reductions were reported for k up to 8 and are summarized in Table I for convenience. The test suites are available online[1].

The *applicability* of the CT-based approach was demonstrated only for one specific case of Trojan with $k = 8$ and for an all-ones activation pattern. In the following, we explore this promising approach further by validating the initial findings and by studying its *efficiency* against multiple Trojan variants of different sizes and triggering patterns.

III. EXPERIMENTAL SETUP

Hardware implementations of cryptographic algorithms are an attractive target for Trojan insertion due to the inherent complexity of their designs and the sensitive information that they process. We opted to study the efficiency of the CT-based approach using the AES symmetric-key encryption algorithm as a case study. We describe in the next paragraphs the AES cryptographic module, the Trojan variants that we implanted

[1]http://www.tamps.cinvestav.mx/~jtj/HMD/index.html

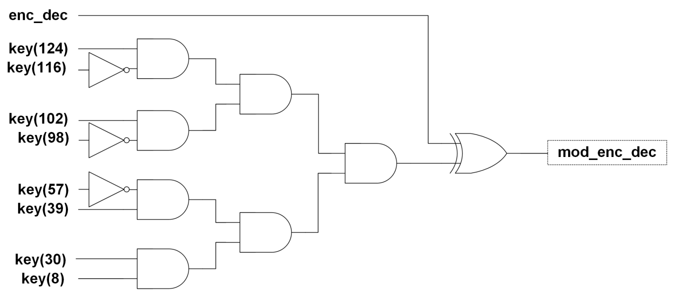

Fig. 1. An example Trojan design.

in the design of the module, and the set and aims of the performed experiments.

A. AES cryptographic module

We opted for the AES implementation that is provided with the SAKURA-G board[2] for our experiments. Its hardware architecture is discussed in detail in [24].

The AES module accepts as inputs a 128-bit `key` quantity and a 128-bit `plaintext` (ciphertext) quantity and produces as an output a 128-bit `ciphertext` (plaintext). The module implements the ECB mode of AES. It can be used as a building block for implementing other modes of AES, such as CBC or OFB, using additional logic for combining and reusing its output. The module can be controlled to perform either the encryption or the decryption operation with one control signal (herein, `enc_dec`). In our experiments, we consider the internals of the AES module as a black box.

B. Trojan contamination

We assume that an informed attacker is able to inject malicious functionality in the aforementioned AES module. Figure 1 depicts an example Trojan design with $k = 8$. The Trojan monitors eight inputs of the key material of the AES module (namely: 8,30,39,57,98,102,116, and 124) for the specific pattern "`11100101`" to occur. The payload part of the Trojan comprises one XOR gate that inverts the value of the `enc_dec` signal. This results in a denial-of-service attack, as the module will not perform the correct operation and the output will be incomprehensible. Observe that the attack is very hard to cope with. There are about 366 *trillion* combinations ($2^8 \times \binom{128}{8}$) for an attacker to choose from and only a handful of gates are needed to realize the whole Trojan.

Simulating the attacker behavior, we contaminated the AES module with 112 variations of the same Trojan of length $k = 8$. The variants are classified in two groups. In the first group, namely A, we use "`11111111`" (eight ones) as a trigger pattern and we vary the monitored positions (e.g., one Trojan triggers when the provided key contains ones in the positions {1, 6, 27, 54, 58, 116, 120, 122} while another one triggers when an all-ones pattern occurs in the positions {22, 37, 38, 64, 75, 99, 109, 118}.

In the second group, namely B, we keep the monitored positions fixed and we vary the trigger pattern (e.g., one Trojan triggers when the pattern "`01100101`" occurs, while another one triggers when the pattern "`11010100`" occurs.

C. Trojan detection

We consider the scenario where someone (a defender) receives a batch of fabricated AES modules and suspects that they are contaminated with a Trojan. The aim of the defender is to reduce the test time per module while retaining a high confidence that the module is Trojan-free. The defender can check the output of the AES module against the one of a trusted implementation of the algorithm (e.g., a software version from a trusted source). If the two outputs differ, then the Trojan is assumed to be present.

We assess the effectiveness of the test suites reported in Table I on exciting the various Trojans described in the previous section. In total, we report and discuss the results of five sets of experiments:

- We run each of the seven test suites once against a trusted implementation of the AES module. The results serve as the baseline (sanity check) for the comparison.
- We run each of the seven test suites once against a specific contaminated AES module.
- We run each of the seven test suites once against eight variants of contaminated AES modules. In all cases, the Trojan uses a triggering pattern of eight ones but the eight monitored positions are different.
- We run each of the seven test suites once against eight more variants of contaminated AES modules. In all cases, the Trojan uses the same eight monitored positions but uses a triggering pattern with varying number of ones (The Hamming weight of the pattern varies between 1 and 8).
- We run each of the seven test suites against contaminated AES modules and we compare their efficiency in contrast with test suites comprising randomly-selected test vectors.

We used the Modelsim tool with appropriate scripting (`do` files and shell scripts) in all of the experiments for automating the execution, collection, and comparison of the outputs. The approach can be easily extended to a hardware co-simulation using the Xilinx ISim HW Co-Simulation[3] environment.

IV. Results and discussion

A. Applicability validation

The first set of experiments studies the efficiency of the test suites of Table I on detecting a Trojan that monitors $k = 8$ inputs for an all-ones pattern. Table II summarizes the results and compares with the ones reported in [9].

We confirm that test suites of strength t that is smaller than the length of the Trojan k are still capable to trigger the Trojan. Furthermore, in the case of $t = k$, the Trojan payload is activated hundreds of times.

[2]http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html

[3]http://www.xilinx.com/tools/feature/14_1_isim_hw_cosim_qrg.pdf

TABLE II
NUMBER OF ALL-ONES $k = 8$ TROJAN PAYLOAD ACTIVATIONS PER TEST SUITE

Strength	[9]	Ours
$t = 2$	0	0
$t = 3$	0	1
$t = 4$	1	1
$t = 5$	1	2
$t = 6$	6	4
$t = 7$	10	11
$t = 8$	178	272

There are some slight deviations in the number of the Trojan payload activations between the two works. This can be explained by the fact that the underlying covering array constructs used for deriving the test suites guarantee the presence of each pattern *at least once*. However, they do not guarantee a fixed number of occurrences for each possible of the possible patterns.

B. Effect of inputs selection

The second set of experiments studies the efficiency of the test suites on detecting a Trojan that monitors $k = 8$ inputs of varying patterns. We used eight different patterns and Table III summarizes the results.

The lower-strength test suites ($t < 6$) succeed in triggering the Trojan in many cases. The higher-strength test suites ($t > 5$) always succeed in activating the Trojans. The test suite matching the length of the Trojan ($t = k = 8$) succeeds *hundreds* of times. As before, there is a variation in the exact number of activations, which depends on the structure of the covering array construct and the specific pattern.

C. Effect of triggering pattern

The third set of experiments studies the efficiency of the test suites on detecting a Trojan that monitors for a specific (not all-ones) pattern to appear on a specific combination of inputs. For the sake of comparison and reference, we opted for the same combination of inputs as the ones reported in Table II, namely `8-30-39-57-98-102-116-124`. Table IV summarizes the results.

We can group the results of this set in three classes. The very-low-strength test suites ($t = 2$ and $t = 3$) almost never succeed in triggering the $k = 8$ Trojan. The low-strength test suites ($t = 4$ and $t = 5$) sporadically succeed in triggering the Trojan. The middle-strength test suites($t = 6$ and $t = 7$) always succeed in triggering the Trojan. The test suite with strength that matches the Trojan length ($t = k = 8$) always succeeds in activating the Trojan; this happens not only once but dozens of times per examined Trojan.

D. Effect of Trojan length

The fourth set of experiments studies the efficiency of the test suites on detecting a Trojan with variable length k. We vary the length of the Trojan, k, the monitored positions, and the triggering pattern. Based on the indications of Table IV, we favored patterns with a balanced number of ones and zeros. Such patterns are expected to be covered less times in the test suites, given the combinatorial origin of the latter. Table V summarizes the results.

The test suites always succeed in activating the Trojan, provided that $t \geq k$ i.e., the test suite strength is equal or greater than the length of the Trojan. Testing with a test suite of strength greater than the Trojan's length requires more test vectors to be applied (cf. Table I) and thus, more testing time. Yet, it results in increased activations, providing clearer evidence of the Trojan's presence.

There is no case where a lower-strength test suite succeeds in activating a Trojan while a higher-strength test suite fails to do so. This is an expected outcome for these test suites: by their definition, any higher-strength test suite already includes the lower-strength combinations too.

The above analysis indicates that there is no apparent reason to run lower-strength test suites before or on top of the higher-strength ones. From a testing budget perspective, one should apply directly the highest-strength test suite they can afford.

TABLE VII
SEARCH SPACE COVERAGE PER TEST SUITE

Length	Total patterns	Our	Random	Missing
$k = 2$	32,512	100%	$94,92\%$	1,649
$k = 3$	2,731,008	100%	$99,20\%$	21,718
$k = 4$	170,688,000	100%	$99,92\%$	129,882
$k = 5$	8,466,124,800	100%	$99,96\%$	3,295,565
$k = 6$	347,111,116,800	100%	$99,998\%$	4,268,479

E. Effect of test suite construction method

The fifth set of experiments studies the efficiency of the test suites in comparison with a test suite consisting of randomly-selected vectors on detecting a Trojan. Table VI summarizes our experiments.

For each Trojan length k, we apply the two test suites of equal size and compare the number of activations achieved. It appears that the two suites have more or less the same performance for the specific combination of monitored positions and activation pattern.

One can argue that test suites comprising random vectors are as good as the ones of Table I. However, the former do not provide a full coverage of the search space and do not provide a minimum guarantee of coverage percentage. To confirm this, we enumerated using a software program the total number of patterns contained in generated test suites comprising random vectors. We report in Table VII the total patterns per case, the percentage of those covered by our test suites (always 100%), the coverage of the random ones, and the number of missing patterns. It was beyond our capacity to enumerate the patterns beyond $k = 6$.

It is evident that the suites comprising randomly-selected vectors fail to cover the whole search space albeit they exhibit a good coverage percentage. Since the test suites of Table I are readily available and provide full coverage, there is no reason to opt for the random ones and risk the possibility to use a contaminated implementation.

TABLE III
ACTIVATIONS OF ALL-ONES TROJAN PER MONITORED INPUT POSITIONS AND TEST SUITE

Monitored positions ($k = 8$)	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$
0-21-32-53-79-100-104-126	1	1	1	0	3	11	159
0-33-45-79-82-94-120-124	0	2	1	0	4	10	168
1-6-27-54-58-116-120-122	0	1	1	1	6	9	193
17-29-44-67-80-98-114-119	0	1	2	2	3	4	208
22-37-38-64-75-99-109-118	0	0	1	1	1	6	182
29-35-47-62-77-107-112-117	0	1	1	1	5	9	201
30-34-37-40-110-115-125-127	1	1	1	1	4	7	202
9-15-37-60-71-97-110-120	0	0	2	0	5	10	216

TABLE IV
ACTIVATIONS OF PATTERN TROJAN PER PATTERN AND TEST SUITE

Trigger pattern	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$
`11111111`	0	1	1	2	4	11	272
`11111101`	0	0	0	0	8	10	63
`10110111`	0	0	0	1	5	10	43
`01011110`	0	0	1	1	5	12	52
`11100100`	0	0	1	0	8	12	66
`01010010`	0	0	0	1	4	11	61
`00100010`	0	0	1	1	3	5	64
`00100000`	0	0	2	0	5	7	142

TABLE V
ACTIVATIONS OF VARIABLE-LENGTH TROJANS PER LENGTH k AND TEST SUITE

Length	Positions	Pattern	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$
$k = 2$	21-114	`01`	3	10	30	63	308	616	4,125
$k = 3$	21-79-114	`101`	1	4	15	32	156	308	2,063
$k = 4$	21-79-97-119	`0101`	0	3	7	17	82	160	987
$k = 5$	3-23-89-107-124	`10100`	0	2	7	9	38	80	532
$k = 6$	3-23-89-95-117-124	`001101`	0	0	2	6	21	44	200
$k = 7$	3-23-63-90-96-118-122	`1010110`	0	0	2	3	12	23	107
$k = 8$	3-23-63-79-90-96-118-122	`01100100`	0	0	0	1	7	10	67

TABLE VI
ACTIVATIONS PER TROJAN LENGTH k AND TEST SUITE CONSTRUCTION METHOD

Length	Monitored positions	Pattern	Size	Our	Random
$k = 2$	21-114	`01`	11	3	4
$k = 3$	21-79-114	`101`	37	4	9
$k = 4$	21-79-97-119	`0101`	112	7	13
$k = 5$	3-23-89-107-124	`10100`	252	9	9
$k = 6$	3-23-89-95-117-124	`001101`	720	21	11
$k = 7$	3-23-63-90-96-118-122	`1010110`	2,462	23	19
$k = 8$	3-23-63-79-90-96-118-122	`01100100`	17,544	67	71
$k = 8$	8-30-39-57-98-102-116-124	`10110111`	17,544	43	75
$k = 8$	8-30-39-57-98-102-116-124	`11100100`	17,544	66	82
$k = 8$	8-30-39-57-98-102-116-124	`00100000`	17,544	142	62

V. CONCLUSIONS AND FUTURE WORK

In this paper, we studied the *efficiency* of test suites comprising covering arrays constructs for hardware Trojan horse detection. The results indicate that these test suites are powerful into uncovering the presence of a Trojan. This power originates from the mathematical guarantees that they cover the whole input space of combinations for a specific number t (the test suite strength).

Our work indicates that combinatorial testing, an approach mainly practiced by the software testing community, offers techniques that are useful for integrating in the hardware development lifecycles. As such, a closer collaboration between researchers and practitioners of the two fields may turn rather beneficial for cross-fertilization and advances in both fields.

From a hardware Trojan detection perspective, it is beneficial to have constructs of as high strength t as possible. This would allow for detecting Trojans of greater length k. A defender can define an "undetectability level" (u) i.e., the number of inputs beyond which they can detect a manipulation in the design using established hardware testing techniques. As such, test suites with strength u will be sufficient. If $u = t = 8$ is sufficient remains to be assessed by the research and industry community.

We further observe that a test suite of strength u covers all the cases up to and including u. Thus, there is no need to apply the test suites of smaller strength. If the design is contaminated with a Trojan of smaller length, applying a test suite of greater length will result in numerous activations.

Finally, the size of the test suite is an important aim for optimization. The mathematical guarantees of the search space coverage can be utilized once the test suite size fits the time budget constraints for testing. At the one extreme, there is exhaustive testing; this cannot be applicable in most cases, and definitely not in the case of cryptographic modules with hundreds of inputs for key and data material. At the other extreme of testing lies testing with randomly-generated vectors. While this can quickly reach high coverage rates, it takes enormous more vectors to provide a 100% coverage. Currently, just 11 vectors are enough for activating any Trojan that uses two inputs. This number reaches 17,544 in the case of eight inputs, covering 366 trillion possible combinations. The number is pretty small. Yet, any further size reductions result in less time needed for testing and this would be more than welcomed.

We envision three directions for future work. The first direction relates to the application of combinatorial testing techniques against combinational Trojans targeting modules that offer other functionality rather than cryptographic operations. The second direction relates to the study of advanced Trojans that incorporate state elements in their design. A repeated application of the available test suites may reveal the presence of sequential Trojans hence, any further size reduction of the test suites is rather beneficial. Finally, a third direction relates to studying the applicability and the efficiency of such test suites as signal amplifiers against Trojans that do not directly affect the output but rather leak information through a side channel.

VI. Acknowledgments

This work was partially supported by the COMET K1 program by the Austrian Research Funding Agency (FFG) and the GSRT Action "KRIPIS" with funds by the EU and the Greek State in the context of the research project "ISRTDI".

References

[1] R. George, "Why we should worry about the supply chain," *International Jornal of Critical Infrastructure Protection*, vol. 11, pp. 22–23, 2015.

[2] M. Rogers and C. D. Ruppersberger, *Investigative Report on the US National Security Issues Posed by Chinese Telecommunications Companies Huawei and ZTE: A Report*. US House of Representatives, 2012.

[3] J. Boyens, C. Paulsen, R. Moorthy, N. Bartol, and S. A. Shankles, "Supply chain risk management practices for federal information systems and organizations," *NIST Special Publication*, vol. 800, no. 161, p. 1, 2014.

[4] European Network and Information Security (NIS) Platform, "Cybersecurity Strategic Research Agenda - SRA," August 2015, final version v0.96.

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

[6] A. Dabrowski, H. Hobel, J. Ullrich, K. Krombholz, and E. Weippl, "Towards a hardware Trojan detection cycle," in *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, Sept 2014, pp. 287–294.

[7] P. Kitsos and A. Voyiatzis, "Towards a hardware Trojan detection methodology," in *2nd EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems (ECYPS 2014)*, Budva, Montenegro, Jun. 2014.

[8] D. Kuhn, R. Bryce, F. Duan, L. Ghandehari, Y. Lei, and R. Kacker, "Combinatorial testing: Theory and practice," *Advances in Computers*, 2015.

[9] P. Kitsos, D. Simos, J. Torres-Jimenez, and A. Voyiatzis, "Exciting FPGA cryptographic Trojans using combinatorial testing," in *26th IEEE International Symposium on Software Reliability Engineering*, ser. ISSRE 2015. Gaithersburg, MD, USA, November 2-5, 2015: IEEE Computer Society, 2015, pp. 69–76.

[10] S. Bhasin, J.-L. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, "Hardware Trojan horses in cryptographic IP cores," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*. IEEE, 2013, pp. 15–29.

[11] P. Kitsos and A. Voyiatzis, "FPGA Trojan detection using length-optimized ring oscillators," in *17th EUROMICRO Conference on Digital System Design (DSD 2014)*. Verona, Italy: IEEE CPS, Aug. 2014.

[12] K. S. Kumar, R. Chanamala, S. R. Sahoo, and K. Mahapatra, "An improved AES hardware Trojan benchmark to validate Trojan detection schemes in an ASIC design flow," in *VLSI Design and Test (VDAT), 2015 19th International Symposium on*. IEEE, 2015, pp. 1–6.

[13] N. Lesperance, S. Kulkarni, and K.-T. T. Cheng, "Hardware Trojan detection using exhaustive testing of k-bit subspaces," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. Tokyo, Japan: IEEE, Jan. 2015, pp. 755–760.

[14] M.-L. Flottes, S. Dupuis, P.-S. Ba, and B. Rouzeyre, "On the limitations of logic testing for detecting hardware Trojans horses," in *Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2015 10th IEEE International Conference On*. IEEE, 2015.

[15] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware Trojan detection," in *Cryptographic Hardware and Embedded Systems (CHES 2009)*. Springer, 2009, pp. 396–410.

[16] L. Fand, L. Li, and Z. Li, "A practical test patterns generation technique for hardware Trojan detection," *ELEKTROTEHNIKI VESTNIK*, vol. 80, no. 5, pp. 266–270, 2013.

[17] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware Trojan detection and reducing Trojan activation time," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 1, pp. 112–125, 2012.

[18] A. Sreedhar, S. Kundu, and I. Koren, "On reliability Trojan injection and detection," *Journal of Low Power Electronics*, vol. 8, no. 5, pp. 674–683, 2012.

[19] S. Ray, J. Yang, A. Basak, and S. Bhunia, "Correctness and security at odds: Post-silicon validation of modern SoC designs," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 146:1–146:6.

[20] T. Vidas and N. Christin, "Evading Android runtime analysis via sandbox detection," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014, pp. 447–458.

[21] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti, "Detecting environment-sensitive malware," in *Recent Advances in Intrusion Detection*. Springer, 2011, pp. 338–357.

[22] X. Mingfu, H. Aiqun, H. Yi, and L. Guyue, "Monte Carlo based test pattern generation for hardware Trojan detection," in *Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11th International Conference on*. IEEE, 2013, pp. 131–136.

[23] S. Dupuis, P.-S. Ba, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "New testing procedure for finding insertion sites of stealthy hardware Trojans," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. EDA Consortium, 2015, pp. 776–781.

[24] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Advances in CryptologyASIACRYPT 2001*. Springer, 2001, pp. 239–254.