IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 19, 2022, accepted February 6, 2022, date of publication February 14, 2022, date of current version February 22, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3151378

Locating Hardware Trojans Using Combinatorial Testing for Cryptographic Circuits

LUDWIG KAMPEL[1], PARIS KITSOS[2], (Senior Member, IEEE), AND DIMITRIS E. SIMOS[1], (Member, IEEE)

[1]SBA Research, 1040 Vienna, Austria
[2]Electrical and Computer Engineering Department, University of the Peloponnese, 26334 Patras, Greece

Corresponding author: Ludwig Kampel (lkampel@sba-research.org)

This work was supported in part by the Bundesministerium für Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie (BMK); in part by the Bundesministerium für Digitalisierung und Wirtschaftsstandort (BMDW); in part by the Federal State of Vienna; in part by the European Union and Greek National Funds; and in part by the Regional Operational Program "Western Greece 2014–2020," under the Call "Financial Strengthening Research Development and Innovation Projects in the Priority Area of RIS3-ICT" through the Project entitled Integrated Information and Communications Technology (ICT)-Based Active Living Support System 'MeACT' under Grant 5038641.

ABSTRACT This paper presents a novel method for locating combinational hardware Trojans (HT) based on fault location approaches used in combinatorial testing. This method relies exclusively on the combinatorial properties of the executed test vectors and the results of test execution. Under specific assumptions, the method is guaranteed to locate all combinational HTs with trigger patterns of length ℓ or less, with the location process itself consuming negligible time. We give a description of our method by devising suitable algorithms and provide the links to combinatorial fault location. Furthermore, we demonstrate our approach in a concrete case study where we locate HTs embedded in a circuit that implements the AES symmetric-key encryption algorithm with 128 bits key length. In these experiments, we demonstrate how any HT that is activated by a trigger pattern of length $\ell \leq 8$ can be located in an effective way. Our method compares particularly well against randomized approaches. Although instantiated for a specific circuit in our case study, the proposed approach is generic, due to its algorithmic description, and can be applied for testing other (cryptographic) circuits. We believe that our work presents an important first step in the development of more general logic testing methodologies for HT location using combinatorial testing methods.

INDEX TERMS Circuit testing, combinatorial testing, detection techniques, hardware trojans, fault location analysis.

I. INTRODUCTION

The security of information and communication technologies and electronic systems in general is often solely related to the security of its software part, leaving hardware security out. However, when treating the security of an electronic system holistically hardware security must be addressed as well. A reliable and secure piece of hardware is expected to implement and execute only what it is designed to and nothing else, even in the presence of an intentional attack. In modern society, the globalization of the semiconductor industry raises additional concerns regarding the authenticity and security of fabricated integrated circuits (ICs). IC design and manufacturing may involve multiple fabricators and circuits have to run through multiple stages until a final product reaches its customer. In each individual production stage, there is potential for malicious manipulation of an IC. This threat is recognized in the US not only by intelligence service agents [1], but also in reports of government institutions [2]. It has been the subject of scientific discussion and investigation for several [3]. For instance, rumors exist that in 2007, a Syrian radar failed to warn of an incoming air strike due to a backdoor in the system's chips [3]. Similarly, a German missile system located at the Turkish-Syrian border may have carried out "unexplained commands" in 2015, with rumors suggesting that the system had been hacked and tampered hardware might have been used as an entry point [4]. More concrete documentation of attacks based on HTs are difficult to find, possibly due to concerns regarding the impact on security, economy, and society. Regardless of whether these rumors are true or not, there exists scientific evidence that cybersecurity attacks based on vulnerable hardware are possible [5]. Thus, establishing a trustworthy supply chain for information technology equipment is of

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

interest for government institutions [6] as well as researchers, see [7] or [8].

Accounting for the manifold opportunities for threats, it becomes very hard to test if a downstream provider has installed an undesired functionality or if they are fully trustworthy. One of the most severe and threatening attacks to an IC is the integration of a hardware Trojan (abbreviated as HT or simply *Trojan* for short), a malicious modification to field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), microprocessors or IoT devices [9], [10]. Such modifications can change the functionality of the hardware, e.g. downgrade its performance or provide a backdoor through which sensitive information can be leaked. A valuable survey analyzing the threat posed by hardware Trojans is provided by [11]. To give an example, the practical feasibility of an S-Box substitution attack for AES on FPGA designs has already been demonstrated in [12]. More recently, in [13] a tampering attack on AES ICs is presented that is designed to recover the secret key and thus fully undermine security of the encryption module. The described attack makes use of an HT that consumes (plaintext) input signals.

The motivation for our work lies in providing an efficient and effective method for the location of "small" HTs that are integrated in the IC at the manufacturing stage, i.e. the malicious components are added after the design phase and are not represented in the netlist or register transfer level. The approach shall be non-invasive, treat the circuit under test as a black box and allow for automation. Further, we want to investigate and demonstrate the suitability of combinatorial testing methods to hardware testing and to advance HT location. We believe that particularly the covering and separating properties of detecting arrays (considered in combinatorial testing) will positively influence the future development of HT location techniques.

A. HARDWARE TROJANS: COMPONENTS, TYPES AND COUNTER MEASURES

While there are implementations of HTs that do not require the addition of any gates, see e.g. [14], in this paper we consider HTs that are realized as additional logic circuits. Such HTs generally consist of two parts: the *trigger* and the *payload*. The trigger circuit is always active. Once it recognizes the activating input, it activates the payload circuit that executes the malicious function of the HT, i.e. the HT is *triggered*. We distinguish HTs according to their trigger condition into analog and digital. The latter can be further split into combinational and sequential HTs according to the type of their circuit logic. See [15] or [16] for a comprehensive HT taxonomy. An example of a combinational HT circuit is depicted in Fig. 1. This HT is comprised by a trigger circuit consisting of seven AND-gates and three NOT-gates, while the payload circuit consists of only one XOR-gate that changes the encryption-decryption mode of the underlying AES module. The trigger circuit consumes the gates that process the input bits of the key or plaintext at

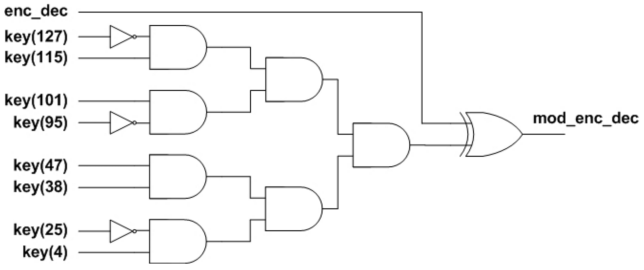

FIGURE 1. An example design of an HT of *length* eight, e.g., for an AES module, that consumes the gates in positions `4–25–38–47–95–101–115–127` corresponding to the key or plaintext input and that is activated when the signals 10110110 occur in these positions.

positions `4–25–38–47–95–101–115–127` and activates the payload circuit if the signals 1, 0, 1, 1, 0, 1, 1 and 0 occur in the respective bits. We call the position of the consumed input bits together with their respective values the *trigger pattern* of the HT and the number of consumed input gates the *length* of the HT. Thus, in the following we often identify HTs with their trigger patterns and do not distinguish between them, e.g. we interchangeably use expressions like *length of the trigger pattern* and *length of the HT*.

Researchers have explored various approaches for HT detection. The acquired techniques can be classified as *destructive* methods, which permanently destroy the IC, and *non-destructive*. Non-destructive methods can be further divided into *invasive* techniques, where the layout of the circuit is modified (e.g. runtime methods) and *non-invasive* techniques where the circuit design is unaffected (e.g. logic testing or side channel analysis methods). More details are presented in surveys of these topics, such as [17] or [18]. Additionally, we want to mention the recent survey [19] on physical and logic testing techniques for HT detection as well as the work given in [20], which reviews HT threats and existing detection and prevention methods from a system-on-a-chip life cycle perspective with a focus on the advancement of machine learning in these domains.

B. THREAT MODEL

We assume that the malicious modification of the design happens via an untrusted electronic design automation (EDA) tool, or in the manufacturing phase, by an untrusted employee at the foundry. Building upon the works of [21] and [22], we consider the same concrete threat model for our case study: We consider an implementation of the AES cryptographic algorithm, with a key length of 128 bits, in the form of hardware IP cores where an attacker integrated an HT triggered by an ℓ bit pattern in the plaintext or key input. The attacker can control the plaintext or the key input and can observe the ciphertext output. We further assume that the attacker combines only a few input signals for the activation, e.g. using a combinational logic relying on AND-gates and NOT-gates for the trigger circuitry, in order to remain undetected. For the payload circuit we assume that the attacker changes the encryption mode of the circuit, switching it

from encryption to decryption and vice versa. Under these assumptions, we can recognize misbehavior of the circuit by comparing its output with the output of a *golden chip*, which can either be a trusted hardware or software implementation of the AES algorithm.

Note that these assumptions are made primarily for the sake of clarity and simplicity of the experimental setup. Our proposed testing method does not depend on *how* the HT manipulates the logic of the hardware, the essential precondition needed for our methodology to work is rather the existence of a testing oracle, i.e. a way to recognize *that* the HT is activated.

According to the comprehensive list of threat models presented in [15], we can categorize the described threat model as model B (untrusted foundry) or model C (untrusted EDA tool). In this sense, we can also categorize the HT design considered in this work according to the hardware Trojan taxonomy presented in [15], which is also underlying the benchmark library of [23] and is used in several scientific works (in slightly adapted versions), as in [19]. Adopting this terminology, we consider HTs that are

- inserted in the fabrication phase,
- on the gate-level,
- activated by a combinational pattern in the user input,
- located at the input, and
- changing the functionality of the circuit.

Regarding the physical characteristics of such HTs, we want to point out that if the targeted circuit is an ASIC, the layout will change in many cases. However, it does not change in case the targeted circuit is an FPGA and the integration of the malicious logic can be realized with available space in already used lookup tables, in which case the HT is also not visible in the netlist. Further, the size of a HT is a relevant physical characteristic, as a small and compact design is better from an attacker's point of view, warding off detection through e.g. optical inspection.

C. CONTRIBUTION

In this paper, we introduce a novel logic testing methodology based on combinatorial fault location methods that can excite and *locate* hardware Trojans that are triggered by certain combinational ℓ bit patterns in the primary input using combinatorial test sets. We instantiate this method by applying it to tampered AES cryptographic circuits. Being non-invasive in nature, our methodology relies solely on the results of the executed test vectors and their combinatorial properties. Further, our method does not rely on a physical realization of a golden chip, i.e. does not require a fully trusted *hardware* realization of the circuit under test. The set of test vectors is optimized aiming for as few vectors as possible while allowing for an efficient method for trigger pattern identification. The proposed work is in line with those presented in [21] and [22], considerably extending their results from *mere detection of the presence of a combinational HT to the precise identification of the consumed input gates* and the respective values that trigger a potential HT – hereafter referred to as *HT location* or HT trigger pattern location.

The devised methodology relies on results coming from combinatorial testing and combinatorial fault location. We revisit these results and show how they can be applied in the domain of hardware Trojan detection and location. Further, we devise novel algorithms in order to describe the developed HT location method. An extensive experimental evaluation demonstrates the functionality of the theoretical results applied to an AES module. Although exemplified by means of this application, due to its algorithmic nature, the proposed method can also be applied for HT location in more general setups. In the experiments, we perform HT location for circuits that implement the AES symmetric-key encryption algorithm in ECB mode for 128 bit key length that have been maliciously modified with HTs of length up to eight. In these experiments, we realize the concept of a golden chip in the form of a software implementation of the aforementioned AES algorithm. We also compare our HT location method against a random approach, showcasing the completeness and efficiency of our technique.

Our testing methodology does not rely on knowledge of the internals of the circuit under test and can be considered as a black box testing approach. We firmly believe that this is a strong advantage, as under realistic circumstances, we cannot assume to have knowledge of the internals of the hardware design where the HT is already inserted.

Thus, the main contributions of this work are as follows:

- To the best of our knowledge, we introduce the first "pure" logic testing method for HT trigger pattern identification, i.e. we present a non-invasive logic testing method that relies solely on the results of the executed test vectors and their combinatorial properties to identify combinational patterns in the input that trigger HTs.
- The proposed method treats the circuit under test as a black-box and is independent from the gate-level netlist or any side-channel analysis.
- It does not require a hardware golden chip.
- We map concepts of combinatorial testing to HT testing and realize combinatorial fault location methods for logic testing.
- We describe the developed HT location method by means of novel algorithms which allow for fast HT trigger identification.
- We conduct an extensive experimental evaluation, locating HTs of length up to eight in FPGAs realizing the AES symmetric-key encryption algorithm in ECB mode for 128 bit key length.
- We further compare our method against a random approach that highlights the completeness and efficiency of our technique.

D. STRUCTURE OF THE PAPER

In Section II we briefly review related work on logic testing and HT location. Thereafter, in Section III we give a brief introduction to combinatorial testing, give some preliminaries

regarding combinatorial fault location and further outline how the concepts of combinatorial testing can be linked to HT location. Subsequently, in Section IV, we describe three different algorithms for HT location using annotated test sets, along with a running example that illustrates the essential workings of these algorithms. In Section V we describe the experimental set-up of our case study and in Section VI we document our experiments regarding HT location based on combinatorial and randomly generated test sets. Section VII contains a brief discussion of practical limitations of the presented work, while Section VIII summarizes the paper and outlines potential impacts. Finally in Section IX we outline several directions of future work based on the methods presented herein.

II. RELATED WORK

This paper presents a combinatorial testing method for locating hardware Trojans in cryptographic circuits. We thus discuss related work pertaining to logic testing methods as well as to HT location approaches.

A. LOGIC TESTING: TEST SET GENERATION

Logic testing approaches generally rely on the execution of test vectors while observing the responses of the circuit under test. Any deviation from the expected result (which can be provided, amongst others, by a golden chip or a simulation) reveals the presence of an HT. Logic testing is thus primarily suited for the detection of HTs that modify the IC's functionality, but it can also be used to enhance side channel analysis, as it was done in [24]–[26] and [27]. Here, the goal of logic testing methods is to activate potential HTs with a reasonable number of test vectors. For a survey dedicated to logic testing methods as a countermeasure to HT insertion, see also [28].

Generally, an attacker will try to design an HT to be stealthy under ''normal conditions'' in order to make it hard to detect. The trigger condition of an HT can thus be assumed to be a rare signal in the IC. Exhaustive testing, however, is usually infeasible due to the number of combinations of inputs and internal states being not tractable. The objective of testing is thus to activate potential HTs within a reasonable test time, aiming for a small number of test vectors, see [29] and [30]. There exist different strategies for test vector generation in order to trigger HTs, where some methods connect the rarity of HT activation to gates with rare values and design vectors in order to activate these signals in the IC, see e.g. [31]. Other methods assume that an attacker has no access to internal gates of the circuit and use combinatorial objects called covering arrays[1] to cover all possible trigger patterns up to a certain length in the primary input of the IC, see e.g. [21] and [22].

In [31] a random sampling approach (called MERO) for HT detection is presented, which is based on multiple excitation of low-probability conditions at the internal nodes of a circuit. MERO works by first identifying *rare nodes* with their associated *rare values*, followed by statistical sampling and execution of test vectors until all rare nodes have been triggered a certain number of times (similar to N-detect tests). Experimental evaluation shows that MERO improves over a random approach by achieving comparable HT detection capabilities while reducing the number of test vectors by 85%.

The work in [32] presents a test generation method based on a genetic algorithm for logic testing of circuits. The key point of this work is to define the fitness function that guides the genetic algorithm based on switching probabilities, controllability and observability parameters. A test vector that activates more rare nodes gets a higher score from the fitness function. A genetic algorithm is used to optimize an initially random set of test vectors until 95% of rare nodes are activated. The authors argue that not covering all of the rare nodes is the main reason why the test generation is faster than MERO [31]. The generated test vectors achieve very competitive results, but ultimately cannot compete with MERO in terms of trigger coverage.

Contrary to these approaches, logic testing based on combinatorial testing treats the circuit under test as a black box. Test vectors are generated such that all input combinations of a fixed number of input gates are *covered* at least a pre defined number of times in order to guarantee the excitement of all possible combinational HTs up to a certain length. In [21], the applicability of combinatorial testing to HT testing is demonstrated by means of testing a hardware implementation of the AES cryptographic algorithm. Since there are no constraints amongst the 128 bit vectors that represent the primary inputs to an AES module, the authors use methods of unconstrained combinatorial interaction testing to generate the test set. Their work highlights the efficiency of combinatorial testing, as it provides test sets that are smaller in size by several orders of magnitude when compared to other approaches while guaranteeing the excitement of specific combinational HTs up to a certain length. Testing focuses exclusively on the input gates of the circuit under test, as they represent the primary interface for testing and triggering a potential attack.

This line of research was continued in [22], where the *completeness* of combinatorial testing in terms of excitement of combinational HT up to a certain length was compared against random testing. A series of experiments with different HT designs underpins the efficiency of combinational Trojan detection through combinatorial testing. A similar study on the effectiveness of combinatorial testing this task is presented in [33].

In addition to the ''pure'' logic testing methods mentioned above, we want to highlight the hybrid approach to HT detection presented in [26], which proposes a side-channel-aware test generation paradigm. The authors introduce the MERS (Multiple Excitation of Rare Switching) algorithm - an evolution of the MERO approach [31] - for test set generation, which takes as input a list of previously identified rare nodes and a set of random vectors.

[1] A brief introduction to these concepts and combinatorial testing is given in Section III.

The set of vectors is modified until each rare node is switched (i.e. changes from its non-rare to its rare value) at least a given required number of times. The generated test vectors are then reordered with the goal of minimizing the total switching in the circuit while maintaining or improving the switching in the rare nodes. The aim of this optimization is to improve the side-channel sensitivity of the approach. Two methods for test reordering are proposed, the first is a heuristic based on minimizing the hamming distance of consecutive test vectors, the second is simulation based and reorders the test set based on information of the switching activity in the circuit obtained from iterated simulations.

The logic testing methods reviewed above share one communality, which is the lack of means for HT location. To the best of our knowledge, most existing logic testing approaches do not offer the generation of test sets that are capable of HT trigger identification at a post silicon stage.

Most recently, another method combining power-based side-channel analysis with logic testing allows to fully isolate Trojan signals in some cases [27]. In this work, a three-phase method based on adaptive logic testing is proposed. In the first phase, an N-detect test set for transition delay fault testing is deployed with the goal of exciting any signal from the Trojan circuitry to produce an initial suspicious power signal. The second phase aims to magnify the suspicious signal by modifying test vectors based on a heuristic that changes small groups of input bits and evaluates the newly generated test vector for its relative power difference value. The process returns the test vector with the highest relative power difference found during the process. The third phase uses adaptive test vector superposition, aiming for test vectors that have a large overlap and cancel out their common effects, in order to detect the presence of a Trojan even under extreme process variation domains. In two documented cases, it was possible to fully isolate the Trojan signal. This method achieves Trojan signal magnification increase by orders of magnitude when compared to ATPG, a significant increase when compared to adaptive ATPG, and is capable of fully isolating the Trojan signals for two instances. The authors of [27] require to apply many modifications to the test vectors in order to find a pair suitable for superposition. To a certain extent, the combinatorial methods for HT location presented in this work can be considered a superposition method that incorporates information of multiple test vectors.

B. LOCATION OF HARDWARE TROJANS

The work in [34] presents the *COTD* technique for Trojan detection and identification that makes it possible to fully identify an inserted HT by isolating its trigger and payload circuit. COTD takes the gate-level netlist as input and combines controllability and observability analysis with unsupervised machine learning to distinguish Trojan gates from genuine gates. A clear advantage of this approach is its independence from a *golden chip* and any test pattern application for partial of full Trojan activation. However, the knowledge of the complete gate-level netlist is required as input, with the Trojan logic inserted, which may not always be available. Similarly, the authors of [35] propose a reference-free scheme for HT trigger location by identifying their rare trigger signals based on the gate-level netlist. They make use of the hypothesis that nodes with rare values (low probability signals) are nodes with an imbalance in 0/1-controllability, which can be calculated by the Synopsys EDA tool TetraMAX. Based on the differences of the 0/1-controllability values 3-means clustering is applied in order to obtain lists of suspicious signals, which can then be refined by a dynamic probability analysis. The authors report that their method can achieve zero false negatives while improving the number of false positives in numerous benchmarks and being very competitive otherwise. Again, this method requires the gate-level netlist with the inserted HT.

The authors of [36] use social network analysis of register transfer level designs for HT trigger and payload location in the design. They assume that the circuit under test, with the potentially inserted HT, is represented as an edge-labelled directed acyclic graph, for which they compute several attributes, such as different centrality measures for vertices or the density of subgraphs. Based on these attributes some of the vertices are marked as possible HT trigger or HT payload nodes, not requiring any simulations or side-channel analysis.

The methods outlined above, in one way or another, assume the knowledge of the Trojan infected design of the circuit, which might not be available, especially if the malicious modification of the circuit happens at the foundry in the silicon stage.

The work in [37] could serve as an important initial step to our work, as it presents a method to identify circuit cites where a potential HT trigger may be inserted. It proposes to first identify nodes with a low controllability based on probability analysis and then to consider those nodes where an insertion of additional HT gates would not result in a significant delay, by considering the nodes with a positive slack. Additionally, the nodes' physical placement in the circuit is taken into account, as a potential HT trigger needs some space in the layout in order to be integrated. Finally, the work proposes to consider subsets of nodes that fulfill the above three criteria, taking into account their physical closeness in the circuit's layout. The authors further propose to generate test vectors designed to trigger the potential HT. However, this step is not carried out, leaving open the problem of (partial) HT triggering and location. Nevertheless, this work treats an important preprocessing step, when one is interested in physically locating an HT. We mention it here, because we believe that such methods can benefit, when supplemented with the hereafter proposed logic testing based on combinatorial fault location.

III. PRELIMINARIES

In the following subsections, we summarize the main aspects of combinatorial testing and combinatorial fault location. We also bridge concepts used in HT detection and location with the aforementioned topics.

A. COMBINATORIAL TESTING

Combinatorial testing is a black box methodology for testing a system under test (SUT) against undesired interactions of its input parameters. The requirements for the application of combinatorial testing are twofold: First, an input parameter model (IPM) [38], that models the SUT by means of input parameters, and their respective values. Second, a testing *oracle*, which conceptually is some method by which the tester can recognize faulty behavior of the SUT.

Combinatorial testing can model dependencies of $t \geq 2$ parameters and allows for testing of *higher-order interaction faults* between input parameters. Combinatorial t-wise testing (combinatorial testing in short) has been applied in several domains in recent years [39] and has been shown to be a valuable and efficient testing methodology, see e.g. [40]–[42] and [43]. For a thorough introduction to combinatorial testing, we direct the reader towards [44].

The theoretical backbone of combinatorial testing is provided by combinatorial objects, such as covering arrays, which can be defined as follows (see also [45]).

Definition 1: A (uniform) *covering array*[2] $\mathsf{CA}(N; t, k, v)$ is an $N \times k$ array over a v-ary alphabet Σ that has the property that in any $N \times t$ sub-array, comprised of any t different columns of the covering array, every t-tuple in Σ^t appears at least once as a row. The parameter t is called the *strength* of the covering array.

Simply put, a covering array of strength t provides a combinatorial test set for the SUT that ensures testing of all possible configurations of any t input parameters. An example of a $\mathsf{CA}(12; 3, 11, 2)$ is given in Table 1. This array has the property that when selecting any three different columns b_i, b_j, b_k with pairwise different $i, j, k \in \{1, \ldots, 11\}$, every binary 3-tuple (word of length three) appears at least once in the array (b_i, b_j, b_k) comprised of these three columns. Further, we know (e.g. from [48]) that there exists no array with less rows that also has this property, i.e. there exists no $\mathsf{CA}(11; 3, 11, 2)$. Covering arrays having the smallest number of rows possible are called *optimal*. The generation of covering arrays with small values of N (which translates to small test sets) is a challenging problem that is subject to current research, and the computational complexity of optimal covering array generation is still unknown [49].

The concept of interactions that are covered by the rows of covering arrays can be formally defined as follows.

Definition 2: For positive integers t, k and v, a *t-way interaction* is a set of t pairs $\{(p_1, v_1), \ldots, (p_t, v_t)\}$ with the property that $v_i \in \{0, \ldots, v-1\}$, $\forall i \in \{1, \ldots, t\}$ and $1 \leq p_1 < \ldots < p_t \leq k$. The values for k and v are usually clear from the context and hence omitted from the notation. We say a t-way interaction $\{(p_1, v_1), \ldots, (p_t, v_t)\}$ is *covered* by a vector $u = (u_i)_{i=1}^k$ of length k, iff $u_{p_j} = v_j$, $\forall j = 1, \ldots, t$. We further define a $\leq t$-way interaction as a set of $s \leq t$ pairs $\{(p_1, v_1), \ldots, (p_s, v_s)\}$ with the property that $v_i \in \{0, \ldots, v-1\}$, $\forall i \in \{1, \ldots, s\}$ and $1 \leq p_1 < \ldots < p_s \leq k$. The pairs that constitute a t-way interaction represent parameter-value assignments in combinatorial testing. To give examples, for $k = 11$ and $v = 2$ a 3-way interaction is might be $\{(3, 1), (4, 0), (11, 1)\}$ and a $\leq$3-way interaction could be the 2-way interaction $\{(2, 1), (5, 1)\}$. The above 3-way interaction is covered by the first and fourth row of the $\mathsf{CA}(12; 3, 11, 2)$ in Table 1.

One of the main advantages of combinatorial testing is that it can be automated, which can help to reduce costs and resource consumption of testing. Based on the IPM of the SUT, a combinatorial test set of *strength t* can be generated with one of the existing tools, e.g. [50], which then can be readily applied for testing the SUT.

TABLE 1. An example of an (optimal) CA(12; 3, 11, 2) that is also a (1, 2)-detecting array.

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}
1	0	1	0	0	1	0	1	1	0	1
1	1	0	0	1	0	1	1	0	0	1
1	0	0	0	1	1	1	0	1	1	0
0	1	1	0	0	0	1	0	1	1	1
1	1	1	1	0	1	1	0	0	0	0
1	0	1	1	1	0	0	0	0	1	1
1	1	0	1	0	0	0	1	1	1	0
0	0	1	1	1	0	1	1	1	0	0
0	1	1	0	1	1	0	1	0	1	0
0	1	0	1	1	1	0	0	1	0	1
0	0	0	1	0	1	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0

B. COMBINATORIAL FAULT LOCATION

Combinatorial fault location is an aspect of combinatorial testing that aims to *identify failure inducing t-way interactions* (FITs). Instead of mere detection, i.e. verification of the presence, of misbehavior caused by t-way interactions, we are further interested in identifying *which* t-way interactions triggered this misbehavior. It is worthwhile to mention again that the SUT is considered a black box and combinatorial fault location can thus not tell us *why* a specific t-way interaction is causing the misbehavior, but only that it does so. Thus, in case the SUT is a software system, combinatorial fault location differs from traditional software fault location - which is part of the debugging process - where we are interested in locating a fault in the *source code*. However, knowledge about FITs can facilitate the location of a fault in the source code [51].

Combinatorial fault location methods can be divided into adaptive and non-adaptive approaches. Adaptive approaches, such as [52] and [53] rely on an online communication with the SUT during testing. The test set is generated during test execution, where results of earlier tests influence the generation of later tests. Alternating test execution and test generation, the set of potential failure inducing t-way interactions is reduced and concertized iteratively.

In contrast, non-adaptive approaches do not require such online communication with the SUT. Here, fault location

[2]In the literature there also exist generalizations, such as *mixed-level covering arrays* [46] and *variable strength covering arrays* [47]. These notions, however, are not relevant for the contribution of this paper.

solely relies on properties of the underlying combinatorial object that gave rise to the test set and the results of test execution in the form of a pass/fail assignment of the tests, obtained from a testing oracle. Combinatorial objects with these desired properties can be regarded as covering arrays having additional properties that allow to locate failure inducing t-way interaction, see e.g. [54] or [55] for definitions an discussions of such structures. For example, *locating arrays* and *detecting arrays* as introduced in [54] are covering arrays with the additional property that a set of t-way interactions is uniquely identifiable via the set of rows where at least one element of the set is covered. In terms of testing, this means that the tests are structured in such a way that the failure inducing t-way interactions can be reconstructed and revealed from the oracle assignment. More precisely, as introduced in [54], in this work we consider (d, t)-detecting arrays, which are covering arrays A of strength t with the additional property that any set $\mathcal{T}$ of d different t-way interactions can be distinguished from any other t-way interaction $\tau \notin \mathcal{T}$ only by the rows of A. This means that any such τ must be covered in at least one row of A that does not cover any element of $\mathcal{T}$. Such a row can be considered a "witness" for τ not being an element of $\mathcal{T}$. Similarly, $(d, \bar{t})$-detecting arrays can be defined, which have analogue properties when quantifying over $_{\leq}t$-way interactions, i.e. interactions of strength *up to t* (recall Definition 2). For the formal definition of these combinatorial structures, the interested reader is referred to [54]. Finally we note that the name *detecting arrays* may be unfortunate in some way, as detecting arrays can be used for FIT *location* and not only FIT detection. An example of a $(1, 2)$-detecting array is given in Table 1.

When used for combinatorial testing, detecting arrays allow for the following fault location procedure (see also [54]): iterate over all passing tests, mark all t-way interactions covered by passing tests as non-failure triggering; the set of all failure inducing t-way interactions is retrieved as the remaining un-marked set. It must be noted that this location procedure is guaranteed to work only if a (d, t)-detecting array is applied for the location of exactly d different failure inducing t-way interactions. However, there exist similar structures, $(\bar{d}, \bar{t})$-detecting arrays, that can be used for the location of *up to d* failure inducing $_{\leq}t$-way interactions.

In the following we make use of a theoretical result presented in [54, Th. 8.5], which states:

A $\mathsf{CA}(N; t + d, k, v)$ *with* $d < v$ *is a* (d, t)*-detecting array.*

Remark 1: Since every $\mathsf{CA}(N; t + d, k, v)$ is also a $\mathsf{CA}(N; (t - 1) + d, k, v)$, we immediately get that for $d < v$ every $\mathsf{CA}(N; t + d, k, v)$ is a (d, s)-detecting array for every $s \leq t$, and hence for $d = 1$ we get further that every $\mathsf{CA}(N; t + 1, k, v)$ is a $(1, \bar{t})$-detecting array. In other words, this shows that every $\mathsf{CA}(N; t + 1, k, v)$ can be used for the location of one failure inducing t-way interaction.

In this work, we aim to apply (non-adaptive) combinatorial fault location for locating HTs (after these have been successfully detected).

C. LINKAGE BETWEEN HARDWARE TROJAN LOCATION AND COMBINATORIAL TESTING

We denote with k the total number of the input signals available to the attacker and with ℓ the *length of the HT*, i.e. the number of input signals of the Trojan's trigger circuit.

In combinatorial testing terminology the 128 bits of the plaintext yield an IPM consisting of 128 binary parameters. As we assume no input dependencies (i.e., the value of one bit does not depend on the value of any of the other input bits), there are no constraints in our model. According to the threat model, the attacker is using some ℓ bit pattern in the plaintext to activate the HT. In this setting, the natural choice is to map the HT's trigger pattern of length ℓ to the concept of the failure inducing t-way interaction, the FIT we are interested in locating in combinatorial fault location. In this setting, the natural choice is to map the concept of the length of trigger patterns of HTs to the concept of the strength of failure inducing interactions. Hence, in our work we can identify trigger patterns of length ℓ as failure inducing t-way interactions. In order to locate the HT's trigger pattern we thus have to construct a binary detecting array which has 128 columns and is capable of locating a single ℓ-way interaction, i.e. a $(1, \ell)$-detecting array or a $(1, \bar{\ell})$-detecting array for 128 binary columns. Since ℓ is unknown to the tester, the key issue is the selection of an appropriate strength t to guide the combinatorial test set generation, allowing us to capture the length ℓ of the integrated HT (note that $t \neq \ell$ in general, as we shall see in the next sections). While this problem is not unknown to general combinatorial testing applications, there is currently no golden rule for selecting the correct value of t. The selection of t is rather a trade-off between detection capabilities and availability of resources. In some cases, there exists empirical evidence that can guide the selection of the interaction strength for the combinatorial test set generation. For example, for some applications within the domain of software testing, security testing as well as testing of medical devices there is empirical evidence that suggests that an interaction strength of $t = 4$ to $t = 6$, depending on the use case, is sufficient for testing, in the sense that all previously documented bugs can be triggered by interactions of these strengths, see [56] and [57]. We will demonstrate, however, that based on the arguments given in Remark 1, we are able to locate HTs of length ℓ with $(1, t)$-detecting arrays, as long as $\ell \leq t$.

Justified by the identification of trigger patterns with failure inducing t-way interactions, we use the same terminology for t-way interactions also for patterns in plaintext inputs. For example, if a plaintext consisting of 128 binary bits contains a certain binary sub-pattern, we also say that the pattern is *covered* by the test vector, just as we do for t-way interactions and binary vectors of length k in general.

Table 2 summarizes how concepts and terms from combinatorial testing can be translated to the domain of hardware testing (with a focus on cryptographics Trojans) and vice versa.

TABLE 2. A summary of the mappings between equivalent concepts and notions in HT location and combinatorial testing.

Hardware Testing	Combinatorial Testing
Plaintext/key length	Number of input parameters k
Plaintext input/key	Test
Test vector	Row of a covering array
Pattern	t-way interaction
Trigger pattern	Failure inducing t-way interaction (FIT)
(Comparison to) golden chip	(Call of the) testing oracle
Correct ciphertext	Passing test
Incorrect ciphertext	Failing test

IV. LOCATION ALGORITHMS

Our objective is to first design a testing method that can excite and *locate* an HT using an optimized test set, and second to manifest an efficient procedure to retrieve the trigger pattern from an annotated set of test vectors. In the following we assume that the AES keys or plaintext vectors applied for testing are given as the rows of an array A, and the result of the testing against the golden chip is given as an *oracle assignment*, that is a column o of pass/fail assignments to the test vectors. In other words, we assume we have an annotated test set (A, o). A naive approach would be to generate and test all possible input vectors and to use simple enumeration methods to locate the HT's trigger pattern. However, this would be practically infeasible due to the size of the input space being exponential in the number of input bits.

A. LOCATION VIA FULL ENUMERATION

The first approach towards locating trigger patterns can be described as follows. Whenever the HT is triggered we observe a discrepancy in the ciphertext output by comparing to the ciphertext output of the golden chip. On the contrary, for any plaintext that represents a passing test (the ciphertext output is identical with that of the golden chip), we know that the trigger pattern cannot be covered in the plaintext and thus that each pattern in this plaintext cannot be the trigger pattern of the HT (i.e. not the FIT in terms of combinatorial testing).

Therefore, a straightforward approach for identifying the trigger pattern of length ℓ is to iterate over all passing tests and all patterns of length ℓ covered by the individual passing tests to mark them as non-trigger patterns. Algorithm 1 represents a pseudocode for such a procedure. This location algorithm for detecting arrays was originally mentioned in [54] directly after the introduction of detecting arrays.

If a trigger pattern exists, the set $\mathcal{T}$ returned by Algorithm 1 contains the trigger pattern or a set of potential trigger patterns, depending on the quality of the test set. For example, if we use a test set that was randomly generated, we generally cannot expect the returned set $\mathcal{T}$ by Algorithm 1 to contain exactly the trigger pattern, but rather a set of potential trigger patterns. This is because there is no guarantee that each non-trigger pattern is covered in a passing test. In contrast, if the test set is deduced from a $(1, \ell)$-detecting array, in case of the presence of an HT of length ℓ, we have the guarantee that there is exactly one remaining pattern in $\mathcal{T}$ due to the combinatorial properties of detecting arrays: each non-trigger pattern must appear in at least one passing test. Thus, we can formulate the following lemma:

Algorithm 1 SlowPatternLocation(A, o, ℓ)

```
INPUT: Test set A, oracle assignment o, length ℓ
P ← passing tests(A, o)        ▷ Extract passing tests from annotated test set
T ← set of all patterns of length ℓ
for p ∈ P do
    for all patterns τ covered by p do
        T ← T \ {τ}            ▷ Mark τ as non-trigger pattern
    end for
end for
return T                       ▷ Set of unmarked patterns
```

Lemma 1: If the test set A is deduced from a $(1, \ell)$-detecting array and the oracle column o is the pass/fail assignment retrieved from testing a modified AES module with an integrated HT of length ℓ, then Algorithm 1 returns the trigger pattern of the HT.

Example 1: We illustrate the location method set out by Algorithm 1 by means of the following example. We use the $(1, 2)$-detecting array given in Table 1 as an example test set A. As this array has 11 columns, it is only suitable for testing an SUT that consists of 11 binary parameters; however, the same concept applies to testing the AES module presented later in this paper. We assume that the occurrence of the values 01 in the positions $1 - 2$ trigger a failure in the SUT; in other words, we assume that there is one FIT, which is $\{(1, 0), (2, 1)\}$ denoted as 2-way interaction. Note that the positions of the trigger pattern do not have to be adjacent for our method to work. We assume adjacent positions solely for the sake of better readability. When we use the array given in Table 1 for testing, the oracle returns the pass/fail assignment $o = (0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0)^T$, where 1 denotes a failing test and 0 denotes a passing test. Henceforth we assume we are only given the annotated test set (A, o) and that we have no information about the trigger pattern, except that its length is two.

From Algorithm 1, we first extract the passing tests $\mathcal{P}$ from (A, o), which yields the following array:

$$\mathcal{P} = \begin{pmatrix} 1&0&1&0&0&1&0&1&1&0&1 \\ 1&1&0&0&1&0&1&1&0&0&1 \\ 1&0&0&0&1&1&1&0&1&1&0 \\ 1&1&1&1&0&1&1&0&0&0&0 \\ 1&0&1&1&1&0&0&0&0&1&1 \\ 1&1&0&1&0&0&0&1&1&1&0 \\ 0&0&1&1&1&0&1&1&1&0&0 \\ 0&0&0&1&0&1&1&1&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix} \tag{1}$$

The set $\mathcal{T}$ is initialised as the set of all $2^2\binom{11}{2}$ binary patterns of length 2, i.e. binary 2-way interactions for $k = 11$:

$$\mathcal{T} = \{\{(1, 0), (2, 0)\}, \{\{(\mathbf{1}, \mathbf{0}), (\mathbf{2}, \mathbf{1})\}\}, \{(1, 1), (2, 0)\},$$

$$\{(1,1),(2,1)\},\{(1,0),(3,0)\},\{(1,0),(3,1)\},$$
$$\{(1,1),(3,0)\},\{(1,1),(3,1)\},\ldots,$$
$$\{(10,0),(11,0)\},\{(10,0),(11,1)\},$$
$$\{(10,1),(11,0)\},\{(10,1),(11,1)\}\}.$$

While this is somewhat tedious to manually verify, the test vectors defined by the rows of $\mathcal{P}$ cover all patterns except 01 in the positions $1-2$, which is the only remaining pattern in the set $\mathcal{T}$ at the end of Algorithm 1. This means that we have successfully recovered the trigger pattern from the annotated test set (A, o) and $\ell = 2$.

Example 2: To compare with HT trigger pattern location when the test set is *randomly generated*, we consider the same preconditions as in Example 1, but now assume we have tested the SUT with the test set A_{rand} that was randomly generated:

$$A_{rand} = \begin{pmatrix} 1&0&0&1&0&1&1&1&1&1&0\\ 1&1&0&0&0&1&1&0&0&1&0\\ 1&0&1&1&0&0&1&1&0&1&0\\ 0&1&1&1&1&1&1&1&0&0&1&1\\ 1&1&0&0&1&0&0&0&1&0&0\\ 0&0&1&1&0&0&0&0&1&1&1\\ 1&0&1&0&1&1&1&0&1&1&0\\ 0&1&1&0&1&1&0&0&0&0&0\\ 1&0&0&0&1&1&1&1&1&1&0\\ 0&0&0&1&1&0&1&1&0&1&1\\ 0&0&0&0&0&1&0&0&0&0&0\\ 0&0&1&0&0&1&0&1&1&1&1 \end{pmatrix}. \quad (2)$$

It is of the same size as the $(1, 2)$-detecting array used in Example 1, but each entry has been selected uniformly at random from the set $\{0, 1\}$. The oracle assignment to the test set A_{rand} is $o_{rand} = (0,0,0,1,0,0,0,1,0,0,0,0)^T$. Following Algorithm 1 on the input $(A_{rand}, o_{rand}, 2)$, we first extract the passing tests $\mathcal{P}_{rand}$ from (A_{rand}, o_{rand}), which yields the array

$$\mathcal{P}_{rand} = \begin{pmatrix} 1&0&0&1&0&1&1&1&1&1&0\\ 1&1&0&0&0&1&1&0&0&1&0\\ 1&0&1&1&0&0&1&1&0&1&0\\ 1&1&0&0&1&0&0&0&1&0&0\\ 0&0&1&1&0&0&0&0&1&1&1\\ 1&0&1&0&1&1&1&0&1&1&0\\ 1&0&0&0&1&1&1&1&1&1&0\\ 0&0&0&1&1&0&1&1&0&1&1\\ 0&0&0&0&0&1&0&0&0&0&0\\ 0&0&1&0&0&1&0&1&1&1&1 \end{pmatrix}.$$

Again, we can iterate over all patterns of length two that are covered by the rows of $\mathcal{P}_{rand}$ and mark them as non-trigger patterns, respectively remove them from $\mathcal{T}$. Doing so, we will find a total of 11 patterns of length two that are not covered by the rows of $\mathcal{P}$; for example, the patterns 01 in positions $1-2$, 11 in positions $2-4$ and 01 in positions $10-11$ to name a view. In this case, we are not able to locate, i.e. uniquely identify, the trigger pattern.

B. LOCATION VIA SEMI-FULL ENUMERATION

A second, slightly different algorithm, also based on the enumeration of patterns, can be described as follows.

This algorithm first splits the executed test vectors into two sets: the failing test vectors $\mathcal{F}$ and the passing test vectors $\mathcal{P}$. As each failing test must cover the trigger pattern, we can select a failing test $\bar{f} \in \mathcal{F}$ at random and we are guaranteed that it covers the trigger pattern. We can now iterate over all patterns $\mathcal{T}_{\bar{f}}$ covered by the failing test $\bar{f}$, check which patterns appear in a passing test and thus mark it as non-trigger pattern. Finally, $\mathcal{T}_{\bar{f}}$ is reduced to a set of (potential) trigger patterns. Algorithm 2 gives a pseudocode of such an algorithmic procedure.

Let us again consider the case where the test set is deduced from a $(1, \ell)$-detecting array for the location of an HT trigger pattern of length ℓ. Again, as for Algorithm 1, we are guaranteed that the returned set $\mathcal{T}_{\bar{f}}$ contains exactly the trigger pattern. This guarantee comes from the property of the $(1, \ell)$-detecting array that each non-trigger pattern must appear in at least one passing test. We can now formulate the following lemma:

Lemma 2: If the test set A is deduced from a $(1, \ell)$-detecting array and the oracle column o is the pass/fail assignment retrieved from testing a modified AES module with an integrated HT of length ℓ, then Algorithm 2 returns the trigger pattern of the HT.

Remark 2: While Algorithm 1 has a runtime in $\Theta(\binom{k}{\ell}|\mathcal{P}|)$, as it iterates over all patterns in all passing tests, Algorithm 2 has a runtime in $O(\binom{k}{\ell}|\mathcal{P}|)$, with the potential of a reduced average runtime, since we expect to find a $\tau \in T_{\bar{f}}$ that is a non-trigger pattern covered by a test vector in $\mathcal{P}$ in $|\mathcal{P}|/2$ steps.

Also, for randomly generated test vectors, we expect Algorithm 2 to be better than Algorithm 1 when applied to the same input (A, o, ℓ), in the sense that the result is more precise. Since we initialize the set of potential trigger patterns only with those of length ℓ that are covered by the failing test $\bar{f}$ instead of all possible trigger patterns of length ℓ, the base set from which we remove the non-trigger patterns is smaller. As both algorithms remove all patterns covered by some passing test, the set returned by Algorithm 2 must be a subset of the one returned by Algorithm 1. In other words, let $patterns_\ell(\mathcal{P}) := \{\tau | \exists p \in \mathcal{P} : p \text{ covers } \tau \text{ and the length of } \tau \text{ is } \ell\}$, then:

$$\mathcal{T}_{\bar{f}} \subseteq \mathcal{T} \Rightarrow T_{\bar{f}} \setminus patterns_\ell(\mathcal{P}) \subseteq T \setminus patterns_\ell(\mathcal{P}).$$

Example 1: (Continued): We locate the failure inducing pattern of length two, this time using Algorithm 2. We extract the passing tests $\mathcal{P}$ from (A, o) (see Equation (1)), and randomly select one of the failing tests $\bar{f} \in \mathcal{F}$, say

$$\bar{f} = (0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0).$$

The set $\mathcal{T}_{\bar{f}}$ of all patterns of length two covered by $\bar{f}$ is then

$$\mathcal{T}_{\bar{f}} = \{\{(\mathbf{1},\mathbf{0}),(\mathbf{2},\mathbf{1})\},\{(1,0),(3,1)\},\{(1,0),(4,0)\},$$

Algorithm 2 SlowPatternLocationVariant(A, o, ℓ)

```
INPUT: Test set A, oracle assignment o, length ℓ
P ← passing tests(A, o)        ▷ Extract passing tests from annotated test set
F ← failing tests(A, o)        ▷ Extract failing tests from annotated test set
Select f̄ ∈ F randomly
T_f̄ ← set of all patterns of length ℓ covered by f̄
for τ ∈ T_f̄ do
    for p ∈ P do
        if p covers τ then
            T ← T \ {τ}                ▷ mark τ as non-FIT
            go to next τ ∈ T_f̄
        end if
    end for
end for
return T_f̄                     ▷ set of unmarked t-way interactions
```

$$\ldots, \{(1,0),(11,0)\}, \{(2,1),(3,1)\}, \{(2,1),(4,0)\},$$
$$\{(2,1),(5,1)\}, \ldots, \{(2,1),(11,0)\}, \ldots \{(8,1),(11,0)\},$$
$$\{(9,0),(10,1)\}, \{(9,0),(11,0)\}, \{(10,1),(11,0)\}.\}.$$

It is once more a tedious to verify manually, but we find every pattern in $\mathcal{T}_{\bar{f}}$, except for $\{(\mathbf{1},\mathbf{0}),(\mathbf{2},\mathbf{1})\}$ covered by some of the passing tests in $\mathcal{P}$. Again, the failure inducing pattern 01 in positions $1-2$ is successfully recovered from the annotated test set (A, o) and $\ell = 2$.

Example 2: (Continued): To compare against random testing, we follow the same procedure where the test set is A_{rand}, as given in Equation (2). We randomly select one of the failing tests, say $\bar{f}_{rand} = (0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0)$. Then, from all patterns of length two $\mathcal{T}_{\bar{f}_{rand}}$ covered by $\bar{f}_{rand}$, we remove those patterns that are covered by the rows of $\mathcal{P}_{rand}$. This yields the returned set

$$\mathcal{T}_{\bar{f}_{rand}} = \{\{(1,0),(2,1)\}, \{(2,1),(3,1)\}, \{(3,1),(10,0)\}\},$$

which contains 3 patterns of length two. This is a reduction compared to the 11 patterns returned from Algorithm 1 applied to the same test set, but it also does not locate the trigger pattern precisely.

C. LOCATION VIA IDENTIFICATION OF SHARED PATTERNS

The third algorithm we introduce allows for efficient HT location when the test set is deduced from a detecting array. We present the procedure in Algorithm 3 using HT location terminology (see Table 2). The underlying idea of this location algorithm is as follows: The trigger pattern must be covered in all failing test sets, hence all failing test vectors must be identical in the positions of the trigger pattern.

In case the test set is derived from a detecting array with the desired properties (i.e. sufficient strength), we can further conclude that for each position $i \in \{1, \ldots, 128\}$ that is not involved in the trigger pattern, there are at least two failing tests $f, f' \in \mathcal{F}$ that disagree in position i: $f_i \neq f'_i$. Thus, we can characterize the positions that are part of the trigger pattern exactly as those where all failing tests agree. Once the positions of the trigger pattern are determined, i.e. the gates that are consumed by the HT have been identified, we can simply obtain the values of the trigger pattern by looking them up in one of the failing test vectors. In Algorithm 3 we present a pseudocode of this algorithmic procedure.

Algorithm 3 FastPatternLocation(A,o)

```
INPUT: Test set A, oracle assignment o
F ← failing tests(A, o)        ▷ Extract failing tests from annotated test set
τ ← ∅                          ▷ Initialize HT trigger pattern τ as empty
Select f̄ ∈ F randomly
for i ∈ {1, ..., 128} do
    if all f ∈ F agree in position i then
        τ = τ ∪ {(i, f̄(i))}
    end if
end for
return τ
```

Lemma 3: If the test set A is deduced from a $(1,\ell)$-detecting array that is a covering array of strength $(\ell+1)$ and the oracle column o is the pass/fail assignment retrieved from testing a modified AES module with an integrated HT of length at most ℓ, then Algorithm 3 returns the trigger pattern of the HT.

For a randomly generated test set A, we cannot expect that the τ returned by Algorithm 3 contains the trigger pattern. First, there is no guarantee that the trigger pattern is actually covered by one of the tests in A. Second, in case it is covered, the property that all failing tests agree exclusively on the trigger pattern is not guaranteed to be satisfied.

Example 1 (Continued): We locate the trigger pattern of length two, this time with Algorithm 3. We first extract all failing tests from (A, o):

$$\mathcal{F} = \begin{pmatrix} 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1 \end{pmatrix}.$$

The idea of Algorithm 3 is to find the trigger pattern as the common pattern of length 2 of all failing tests. It is easy to see that the pattern 01 in positions $1-2$ is exactly this common pattern of length 2.

Example 2 (Continued): A third time, we want to compare against random testing. We follow the same algorithmic procedure with the randomly generated test set A_{rand} given in Equation (2) and the pass/fail assignment given by $o_{rand} = (0,0,0,1,0,0,0,1,0,0,0,0)^T$. We extract the failing test cases $\mathcal{F}_{rand}$ from (A_{rand}, o_{rand}):

$$\mathcal{F}_{rand} = \begin{pmatrix} 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1 \\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0 \end{pmatrix}.$$

The pattern τ returned by Algorithm 3 is 0111100 in positions $1-2-3-5-6-8-9$, or, written as a 7-way interaction, $\tau = \{(1,0),(2,1),(3,1),(5,1),(6,1),(8,0),(9,0)\}$.

However, this 7-way interaction contains 21 different 2-way interactions: $\{(1, 0), (2, 1)\}, \{(1, 0), (3, 1)\}, \ldots, \{(2, 1), (8, 0)\}, \ldots, \{(6, 1), (8, 0)\}$ and $\{(6, 1), (9, 0)\}$. The failure inducing 2-way interaction is thus not uniquely identifiable.

Remark 3: The time complexity of Algorithm 3, is greatly reduced compared to that of Algorithms 1 and 2, as it only iterates once over the k bits of the test vectors, searching for common entries. Algorithms 1 and 2 both rely on enumeration of length ℓ sub-patterns of test vectors, introducing a factor of $\binom{128}{\ell}$ to the runtime complexity in our case, or more general a factor of $\binom{k}{\ell}$ when there are k gates accessible to an attacker. The reduction in runtime becomes apparent when we consider the location of an HT of length $\ell = 8$; in this case, we would have to iterate over $\binom{128}{8}$ (more than 10^{12}) patterns to locate the HT via Algorithms 1 or 2. In contrast to this the runtime of Algorithm 3 is largely *independent* from the HTs length ℓ.

V. CASE STUDY

We demonstrate the efficiency of the proposed HT location based on combinatorial testing in a case study, applying it to an FPGA implementing the AES symmetric-key encryption that got tampered with a combinational HT. Even though we consider a concrete case, we highlight once more that the proposed method is explained by means - but not limited to - the described AES module. In the next paragraphs, we describe the used AES cryptographic module's, the HT variants that were integrated in the modules design, as well as the setup for the conducted experiments.

We consider the same scenario as in [21] and [22], where a tester receives a batch of fabricated AES modules. The testers suspect that (some of) the modules are contaminated with a combinational HT that consumes primary inputs. Their goal is to locate the HT, with the additional aim of reducing the test time per module while attaining a high confidence that the module is HT free.

For our experiments, we opted for the Verilog code of the AES implementation that is provided by the SAKURA-G board 2, which hardware architecture is discussed in detail in [58]. The AES module accepts as input a 128-bit key and a 128-bit plaintext (respectively ciphertext) and produces 128-bit ciphertext (plaintext) as output. The module implements the ECB mode of AES, which 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 via a control signal to switch between the encryption and the decryption operation. *In our experiments, we consider the internals of the AES module as a black box.*

As mentioned in Section I-B the output of the AES module (Verilog code simulation) can be checked against the output of a trusted implementation of the algorithm (e.g., a software version from a trusted source), which serves as a testing oracle. If the two outputs differ, then the HT is assumed to be activated, i.e. its trigger pattern must be present in the input.

We used the ModelSim[3] tool with appropriate scripting (do files and shell scripts) in all of our experiments for automating the execution, collection, and comparison of the outputs. The approach can be easily extended to a hardware co-simulation using hardware co-simulation with Vivado.[4] The processing of the test vectors and the testing results was performed using a Matlab implementation of Algorithm 3. The presented time measurements are based on Matlab implementations of the described procedures and were conducted on a machine with an Intel i9-9900 CPU clocked at 3.60 GHz with 64GB of RAM.

VI. EXPERIMENTS

In this section we demonstrate the capabilities of the proposed HT location using combinatorial fault location techniques. To this end, we adopt the conceptualization of the experiments performed in [22], i.e. consider cases where AES modules have been contaminated with HTs of different lengths and different trigger patterns. Thereby, we shift the focus from mere HT detection as presented in previous works [21] and [22] to HT location, i.e. the exact identification of the trigger pattern via Algorithm 3, if not stated differently. Further, we analyze the capabilities of randomly generated test sets (respectively arrays), similar to the analogous process in [22], but now performing HT location instead of HT detection. Therefore, as part of our experiments, we consider the following test sets (which are provided online under [59]):

- CTℓ: refers to a combinatorial test set derived from a $(1, \bar{\ell})$-detecting array for 128 binary parameters, suited for HT location of length up to ℓ. The number of rows (i.e. test vectors) is reported in Table 3.
- *rand*$(CT\ell)$: refers to a randomly generated test set derived from a random array (each entry chosen uniformly at random from $\{0, 1\}$) that is of the same size as the $(1, \bar{\ell})$-detecting array underlying the CTℓ array.
- $rand_N(CT\ell)$: refers to a randomly generated test set derived from a random array with N rows (each entry chosen uniformly at random from $\{0, 1\}$). The chosen size N is identical to that of the smallest $(1, \bar{\ell})$-detecting array for 128 binary parameters currently known.

In our experiments we used combinatorial test sets CTℓ that allow for HT location based on Algorithm 3, i.e. we generated $(1, \ell)$-detecting arrays that are CAs of strength $\ell+1$ for the location of HTs with trigger patterns of length up to ℓ. In comparison to other logic testing approaches these test sets demonstrate the efficiency of combinatorial testing in terms of generating small size test sets. Table 3 shows a comparison to other (state of the art) test sets for hardware testing for $k = 128$ input bits. The column headed by "ℓ" denotes the length of the HT for which the respective test set is designed; the column "Lesperance *et al.*" reports the test set sizes given in [60]; "CWV" contains the analogous sizes given in [61]; "CTdetect" shows the sizes from [22];

[3]https://www.mentor.com/products/fv/modelsim/

[4]https://www.xilinx.com/video/hardware/hardware-co-simulation-vivado-system-generator-for-dsp.html

TABLE 3. Comparison of sizes of test sets coming from CT methods (CTdetect & CTlocate) against other state of the art logic testing techniques for combinational HT detection.

k	ℓ	Lesperance et al.	CWV	CTdetect	CTlocate (CTℓ)
128	2	2^7	129	11	54
128	3	-	256	37	135
128	4	2^{13}	8,256	112	346
128	5	-	16,256	252	5,921
128	6	-	349,504	720	29,830
128	7	-	682,752	2,462	103,691
128	8	2^{23}	11,009,376	17,544	595,979

and "CTlocate" reports the sizes of the test sets CTℓ for $\ell \in \{1, \ldots, 8\}$ used in the following experiments. We can see that the test sets used for HT location (CTlocate) are considerably larger than those for HT detection (CTdetect). However, these are still smaller by orders of magnitude compared to the test sets of [60] and [61] that provide full coverage of all length ℓ patterns and are designed merely for HT detection. The increased size of the test sets for HT location is caused due to additional structure and tests that are required in order to locate all HTs of a specific length. As the test sets can still be processed very fast in terms of test execution, we do believe that the capability of locating all HTs of length ℓ justifies the increased size.

Further note, that the test sets we used are not necessarily optimal, i.e. it is possible to construct combinatorial test sets with less test vectors. However, the number of test vectors in the combinatorial test set is not relevant for the location capabilities of our approach, as long as the necessary combinatorial properties, i.e. being a $(1, \bar{\ell})$-detecting array for 128 binary parameters, are guaranteed. This becomes more clear when we consider Algorithm 3 and the associated Lemma 3, which do not depend on the test set size, but only on its combinatorial properties.

In the following sections, we report and discuss the results of a set of experiments, where in each case the test sets are applied to a trusted implementation of the AES algorithm and to a contaminated AES module where an HT is triggered by a pattern unknown to the tester:

(A) In Section VI-A, we focus on locating Trojans of length ℓ using CTℓ arrays:
- We run eight combinatorial test sets against eight contaminated AES modules that differ in the length of the inserted HT.
- We run eight combinatorial test sets against eight different versions of contaminated AES modules. In each case, the HT is triggered by the signals 11111111, but the eight gates monitored by the HT differ.
- We run eight combinatorial test sets against eight different versions of contaminated AES modules. In each case, the HT monitors the same positions/input gates, but the values of the pattern differ, having a varying number of ones.

(B) In Section VI-B, we focus on analyzing Trojans of length ℓ using random arrays:
- We run eight random test sets against eight contaminated AES modules that differ in the length of the inserted HT.
- We measure the location capabilities of the random test sets for HTs of length $\ell = 1, 2, 3, 4$ when used in conjunction with Algorithm 3.

(C) In Section VI-C, we focus on fast pattern location with random arrays that have the same number of rows as the smallest $(1, \bar{\ell})$-detecting array for 128 binary parameters currently known:
- We measure the location capabilities of a second set of random test sets of reduced size for HTs of length $\ell = 1, 2, 3, 4$ when used in conjunction with Algorithm 3.

(D) In Section VI-D, we focus on slow pattern location properties of random arrays:
- We measure the location capabilities of the first set of random tests for HTs of length $\ell = 1, 2, 3, 4$ when used in conjunction with Algorithm 1.

(E) In Section VI-F, we summarize the experimental evaluation as a whole.

A. LOCATING LENGTH ℓ TROJANS WITH CTℓ ARRAYS

In the first set of experiments, we demonstrate the HT location capabilities of the combinatorial test sets CTt for $t = 1, \ldots, 8$, in conjunction with Algorithm 3. The results of our experiments with HTs of different length are documented in Table 4. The columns contain the following values:
- "ℓ" shows the length of the inserted HT,
- "positions" shows the number of the respective gates that the HT trigger circuit is consuming,
- "pattern" shows the signals that need to appear at these gates to trigger the HT,
- "CTt" shows, for $t = 1, \ldots, 8$, how often the specific HT was triggered (# trig) by the test set CTt and whether the HT was located (loc) using Algorithm 3.

When we are able to locate the HT, i.e. precisely retrieve positions and pattern, we mark this with a ✓ in the corresponding column headed by "loc"; otherwise we denote it as ✗. We can see that all HTs of length up to ℓ can be located with the test set CTℓ, which is expected since CTℓ represents a $(1, \bar{\ell})$-detecting array. In some cases, we can also locate HTs of length ℓ with test sets CTt where $t \leq \ell$. While there is no theoretical guarantee for this capability, it is possible that some trigger patterns can be located using these concrete test sets in combination with Algorithm 3.

In the second set of experiments, we use the same combinatorial test sets CTt for $t = 1, \ldots, 8$ in conjunction with Algorithm 3 in order to locate HTs of length eight that have the trigger pattern 11111111 in different positions. Table 5 documents the results of our experiments, with column headings corresponding to those of Table 4.

In the third set of experiments we fixed the input gates that are consumed by the HT and vary the values in the activation

TABLE 4. The given patterns were used in the specified positions as HT trigger patterns. The remaining columns detail the trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 3 for HT location.

			CT1		CT2		CT3		CT4		CT5		CT6		CT7		CT8	
ℓ	pattern	positions	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
1	0	13	8	✓	28	✓	69	✓	175	✓	2968	✓	14955	✓	51964	✓	298460	✓
2	11	70-120	4	✗	13	✓	33	✓	82	✓	1 482	✓	7 455	✓	25 971	✓	143 654	✓
3	101	5-26-111	1	✗	7	✗	14	✓	45	✓	754	✓	3 823	✓	13 291	✓	72 937	✓
4	1100	0-30-110-127	0	✗	3	✗	5	✗	23	✓	407	✓	1 902	✓	6 546	✓	48 160	✓
5	01100	0-20-70-126-127	2	✗	2	✗	6	✗	10	✓	219	✓	1 045	✓	3 527	✓	17 112	✓
6	001111	1-21-79-100-101-123	1	✗	1	✗	3	✗	2	✗	115	✓	505	✓	1 735	✓	16 703	✓
7	1101010	1-21-50-82-101-111-123	0	✗	0	✗	0	✗	4	✗	48	✓	230	✓	804	✓	5 746	✓
8	10011100	9-10-30-40-89-90-100-125	0	✗	0	✗	0	✗	2	✗	18	✓	83	✓	330	✓	3 026	✓

TABLE 5. The values 11111111 were used in the positions specified in the first column as HT trigger pattern. The remaining columns detail the trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 3 for HT location.

	CT1		CT2		CT3		CT4		CT5		CT6		CT7		CT8	
positions	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
1-23-27-48-74-95-106-127	1	✗	0	✗	0	✗	1	✗	46	✗	250	✓	759	✓	3 084	✓
3-7-33-45-48-82-94-127	2	✗	2	✗	0	✗	1	✗	86	✓	313	✓	945	✓	7 248	✓
5-7-11-69-73-100-121-126	1	✗	1	✗	1	✗	1	✗	74	✓	369	✓	1 009	✓	6 426	✓
8-13-29-47-60-83-98-110	1	✗	1	✗	2	✗	2	✗	46	✗	222	✓	670	✓	4 125	✓
9-18-28-52-63-89-90-105	1	✗	1	✗	1	✗	1	✗	44	✗	172	✓	554	✓	1 330	✓
10-15-20-50-65-80-92-98	1	✗	1	✗	1	✗	1	✗	26	✓	145	✓	462	✓	5 654	✓
0-2-12-17-87-90-93-97	0	✗	0	✗	0	✗	2	✗	37	✓	185	✓	526	✓	7 073	✓
7-17-30-56-67-90-112-118	1	✗	1	✗	1	✗	2	✗	34	✗	186	✓	537	✓	3 609	✓

pattern (with varying Hamming weight, i.e. different numbers of ones). Table 6 shows the results of these experiments; again, the column headings are identical to those of Table 4. Reflecting on the experiments documented in Table 5 and 6, we can see that some test sets CTt with $t < 8$ can locate the HTs of length eight. More detailed, we see that based on the test sets CT1, CT2, CT3 and CT4 we cannot locate any of the HTs of length eight, which is expected as they are designed for locating lower length HTs. In several cases these test sets also fail to trigger the HTs in the first place. The test set CT5 can trigger all examined HTs of length eight, and it is possible to also locate them in half of the cases. Based on the results of the test sets CT6, CT7 and CT8 it is possible to locate all examined HTs of length eight. In general it is more likely being able to locate a HT when using CTt test sets for larger t, because the test sets are larger. However, there is no guarantee that all HTs of length eight can be located with test sets CTt for $t \leq 7$. Nonetheless, we are ensured to locate all HTs of length eight when using the test set CT8.

We wish to highlight that it is always possible to locate HTs of length ℓ, not just those using the examined trigger patterns, when a combinatorial test set CTt with $\ell \leq t$ is used in conjunction with Algorithm 3. The trigger patterns used in these experiments were arbitrarily selected and can certainly not *guarantee* the effectiveness of our proposed approach, but only serve as a means to exemplify it. However, thanks to the combinatorial properties of detecting arrays and the arguments of Lemma 3 we can safely argue that the previously mentioned location capabilities always hold.

B. ANALYZING LENGTH ℓ TROJANS WITH RANDOM ARRAYS

To further illustrate that some trigger patterns can also be located by arbitrary arrays, we conduct the same experiments documented in Table 4 with the randomly generated test sets $rand(CT\ell)$ and try to locate the trigger patterns via Algorithm 3. The results of these experiments are given in Table 7. We can see that the randomly generated arrays tend to perform similarly to the combinatorial test sets CTℓ, for $\ell = 1, \ldots, 8$, exhibiting comparable trigger and location capabilities.

These experiments raise the question, how well random arrays are suited for HT location in general. As these illustrative experiments rely on some exemplary random samples and do not provide a comprehensive assessment, we measure how many patterns of a given length can be located by Algorithm 3 when using randomly generated test sets, to address this question.

For this purpose, for the arrays $rand(CT\ell)$ (with $\ell = 1, \ldots, 4$), we measure how many patterns are locatable via Algorithm 3. To this end, we first generate the respective oracle column o for each pattern by checking which test vectors of the random array $rand(CT\ell)$ cover the specific pattern. Second, we call Algorithm 3 on input $(rand(CT\ell), o)$ and check if the returned pattern τ equals the specific pattern at hand. We implemented this procedure in Matlab and conduct these experiments for all patterns of length $\ell = 1, 2, 3, 4$ and measure how many can be located with each random test set, where the focus of interest rests on the number of located patterns of length ℓ when using the $rand(CT\ell)$ test set. The results of these measurements are given in Table 8. The diagonal entries of Table 8, i.e. the values of the percentage and total number of patterns of length ℓ that were *not* correctly located by $rand(CT\ell)$, are visualized in Fig. 2 and Fig. 3. We can see that the randomly generated arrays $rand(CT\ell)$ are not capable of locating all patterns of length ℓ, while the equally sized combinatorial test sets CTℓ are capable of doing so. Further, we see that for increasing length of the pattern ℓ,

TABLE 6. The positions 3-11-25-29-70-88-97-119 are fixed and were used in combination with the patterns specified in the first column as HT trigger pattern. The remaining columns detail trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 3 for HT location.

	CT1		CT2		CT3		CT4		CT5		CT6		CT7		CT8	
pattern	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
11111111	1	✗	1	✗	3	✗	0	✗	52	✓	258	✓	812	✓	4 646	✓
10111111	0	✗	0	✗	2	✗	1	✗	10	✗	47	✓	227	✓	1 458	✓
11101101	0	✗	1	✗	1	✗	1	✗	17	✓	65	✓	272	✓	2 587	✓
01111010	0	✗	0	✗	2	✗	2	✗	8	✗	65	✓	252	✓	1 897	✓
00100111	0	✗	1	✗	0	✗	2	✗	8	✗	122	✓	388	✓	1 411	✓
01001010	0	✗	0	✗	1	✗	1	✗	11	✗	61	✓	246	✓	1 446	✓
01000100	0	✗	0	✗	0	✗	0	✗	15	✓	83	✓	307	✓	788	✓
00000100	0	✗	1	✗	0	✗	0	✗	46	✓	135	✓	494	✓	3 168	✓

TABLE 7. The given patterns were used in the specified positions as HT trigger patterns. The remaining columns detail the trigger and location capabilities of the *rand*(*CT*ℓ) arrays for $\ell = 1, \ldots, 8$, using Algorithm 3 for HT location.

			$rand(CT1)$		$rand(CT2)$		$rand(CT3)$		$rand(CT4)$		$rand(CT5)$		$rand(CT6)$		$rand(CT7)$		$rand(CT8)$	
ℓ	pattern	positions	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
1	0	13	7	✗	40	✓	66	✓	161	✓	2 977	✓	14 914	✓	51 906	✓	297 355	✓
2	11	70-120	4	✗	21	✓	31	✓	84	✓	1 459	✓	7 508	✓	25 858	✓	148 628	✓
3	101	5-26-111	3	✗	5	✗	14	✓	32	✓	722	✓	3 835	✓	13 123	✓	13 123	✓
4	1100	0-30-110-127	0	✗	2	✗	10	✓	15	✓	361	✓	1 861	✓	6 492	✓	37 294	✓
5	01100	0-20-70-126-127	1	✗	2	✗	8	✓	7	✗	196	✓	903	✓	3 223	✓	18 606	✓
6	001111	1-21-79-100-101-123	0	✗	1	✗	5	✗	4	✗	90	✓	500	✓	1 584	✓	9 368	✓
7	1101010	1-21-50-82-101-111-123	1	✗	3	✗	4	✗	4	✗	42	✓	252	✓	817	✓	4 630	✓
8	10011100	9-10-30-40-89-90-100-125	0	✗	0	✗	0	✗	1	✗	29	✓	111	✓	470	✓	2 398	✓

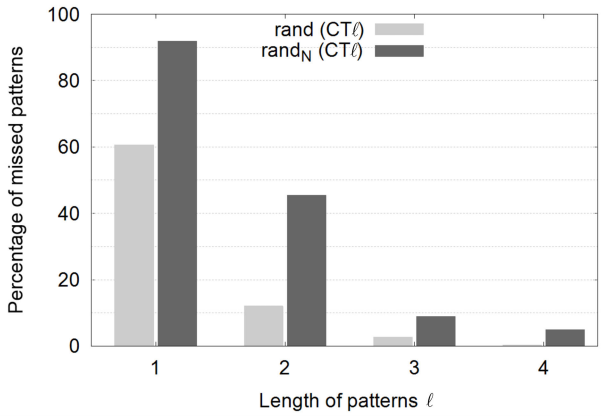

FIGURE 2. The percentage of missed patterns when using randomly generated arrays (*rand*(*CT*ℓ) and *rand*$_N$(*CT*ℓ)) on the vertical axis, for the values of $\ell = 1, 2, 3, 4$ on the horizontal axis.

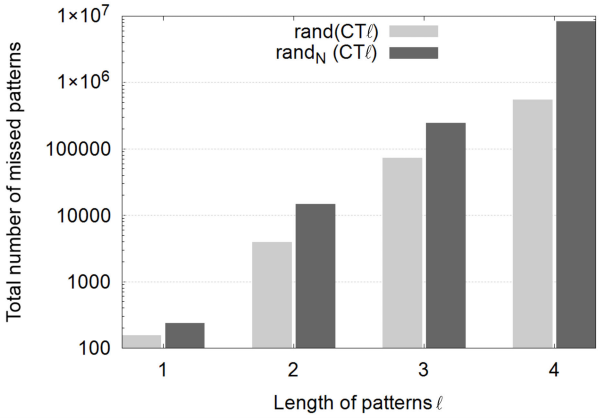

FIGURE 3. The total number of missed patterns when using randomly generated arrays (*rand*(*CT*ℓ) and *rand*$_N$(*CT*ℓ)) on the vertical axis as base-10 logarithms, for the values of $\ell = 1, 2, 3, 4$ on the horizontal axis.

the randomly generated test sets $rand(CT\ell)$ seem to increase their performance, in terms of locating a higher percentage of HTs via Algorithm 3.

C. FAST PATTERN LOCATION WITH RANDOM ARRAYS SIZED AS OPTIMAL COVERING ARRAYS

As the combinatorial test sets utilized in our experiments are not optimal, we conduct the same measurements as reported in Table 8 with randomly generated test sets $rand_N(CT\ell)$, that are of the same size as (currently) best known approximates to optimal $(1, \bar{\ell})$-detecting arrays for 128 binary parameters. The number of tests for these sets can be found at [62]. However, as the actual test sets are not provided, we could not use them for our experiments documented in Section VI-A. The results of our measurements are presented in Table 9. Due to the decreased number of tests, the location capabilities of the $rand_N(CT\ell)$ test sets are further reduced when compared to those of the $rand(CT\ell)$ - we can see that the numbers of HT trigger patterns that are not correctly located increase compared to Table 8. Again, we visualize the diagonal entries of Table 9, i.e. the results regarding the location of length ℓ HTs with $rand_N(CT\ell)$ for $\ell = 1, 2, 3, 4$ in Fig. 2 and Fig. 3. These measurements highlight even more the advantage of fault location based on combinatorial testing compared to random testing approaches.

D. SLOW PATTERN LOCATION PROPERTIES OF RANDOM ARRAYS

For the sake of completeness, we also conduct experiments regarding the location of HTs when the randomly generated test sets $rand(CT\ell)$ are used in conjunction with Algorithm 1. Due to the time complexity of the location via full enumeration used in Algorithm 1 (see Remark 2 and Remark 3), it is not feasible to run a *full enumeration* version for all

TABLE 8. Measurements regarding the fast pattern location properties (i.e. number of patterns that can be located via Algorithm 3) of the *rand*($CT\ell$) arrays. The column 'loc' denotes whether all patterns can be located based on the respective array; the column "miss %" gives the percentage of patterns of length ℓ that can *not* be successfully located - "miss #" the total number; the column "time" denotes the time consumed for the measurement.

	$rand(CT1)$				$rand(CT2)$				$rand(CT3)$				$rand(CT4)$			
ℓ	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time
1	✗	60.54%	155	0 s	✓	0	0	0 s	✓	0	0	0 s	✓	0	0	0 s
2	✗	97.94%	31 844	3 s	✗	12.03%	3 914	5 s	✓	0	0	7 s	✓	0	0	15 s
3	✗	99.94%	2 729 530	315 s	✗	76.64%	2 093 121	372 s	✗	2.65%	72 462	575 s	✓	0	0	1 103 s
4	✗	99.99%	170 686 267	22 097 s	✗	98.53%	168 175 085	23 065 s	✗	54.43%	92 904 935	33 770 s	✗	0.32%	546 726	67 780 s

TABLE 9. Measurements regarding the fast pattern location properties (i.e. number of patterns that can be located via Algorithm 3) of the $rand_N(CT\ell)$ arrays. The column "loc" denotes whether all patterns can be located based on the respective array; the column "miss %" gives the percentage of patterns of length ℓ that can *not* be successfully located - "miss #" the total number; the column "time" denotes the time consumed for the measurement.

	$rand_{11}(CT1)$				$rand_{37}(CT2)$				$rand_{112}(CT3)$				$rand_{252}(CT4)$			
ℓ	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time
1	✗	91.79%	235	0 s	✓	0	0	0 s	✓	0	0	0 s	✓	0	0	0 s
2	✗	99.83%	32 457	4 s	✗	45.46%	14 779	4 s	✗	0.009%	3	7 s	✓	0	0	12 s
3	✗	99.99%	2 730 926	345 s	✗	94.30%	2 575 258	334 s	✗	8.94 %	244 281	516 s	✗	0.002%	60	862 s
4	✗	99.99 %	170 687 887	20 985 s	✗	99.81 %	170 355 238	25 096 s	✗	71.95 %	122 803 555	30 315 s	✗	4.90 %	8 372 513	51 522 s

possible HT trigger patterns, but rather a *decision* version. This means that for given $\ell \in \{1, 2, 3, 4\}$, we iterate over all possible HT trigger patterns of length ℓ, generate the oracle column o for the respective test set A_{rand} and check if Algorithm 1 successfully reconstructs the trigger pattern on input A_{rand}, o and ℓ. If a trigger pattern is reconstructed, we proceed with the next trigger pattern of length ℓ; if the pattern is not reconstructed we abort the search and record that the used test set A_{rand} cannot locate all HTs of length ℓ based on Algorithm 1.

The results of our experiments are documented in Table 11. If a pattern exists that is not located correctly, we represent this with ✗, otherwise we use ✓. If we were unable to perform a computation due to restrictions of our existing computing infrastructure, we denote it with **?**. As a result of the decision version of the measurements, we cannot provide the number of HTs of length ℓ that were not located correctly. We can see that the test sets *rand*($CT\ell$) are capable of locating all HT trigger patterns of length ℓ via Algorithm 1. Compared to the location via Algorithm 3, when using Algorithm 1 the test sets *rand*($CT\ell$) can locate all trigger patterns that use one additional gate. This can be explained by the more general location method used in Algorithm 1 that is more suited for pattern location using arbitrary test sets. However, note that conducting such measurements for higher strengths with the aim of obtaining guarantees for the location of HTs is infeasible in practice. For example, to verify that an arbitrary array is capable of locating all HTs of length five, using the technique described above would require processing a multiple of $\binom{128}{5}^2 > 6 \cdot 10^{16}$ patterns of length 5, a requirement that cannot be satisfied using current computing infrastructure.

To showcase this argument and to highlight the advantage of Algorithm 3 in conjunction with combinatorial test sets, we compare the runtimes of Algorithm 3 and Algorithm 1 when used to locate an HT of length $\ell = 1, 2, 3, 4$. As the time needed for test execution does not influence this comparison, we do not include it in the measurement and instead only compare the runtimes of Algorithm 3 over Algorithm 1, provided that the testing oracle o is already known. To represent 100 randomly generated HTs of lengths

TABLE 10. Comparison of the runtimes of Algorithm 1 and 3 when used to locate an HT of length ℓ based on testing with the combinatorial test set CTℓ and the respective testing oracle. The given runtimes are averaged over 100 runs, where the HTs were placed randomly.

HT length \ Algorithm	Algorithm 1	Algorithm 3
$\ell = 1$	$278 \cdot 10^{-4}$ s	$1.0 \cdot 10^{-4}$ s
$\ell = 2$	2.0138 s	$1.1 \cdot 10^{-4}$ s
$\ell = 3$	139 s	$1.2 \cdot 10^{-4}$ s
$\ell = 4$	5 503 s	$1.7 \cdot 10^{-4}$ s

$\ell = 1, 2, 3$ and 4, we select 100 trigger patterns out of the $2^\ell \binom{128}{\ell}$ total possible trigger patterns of length ℓ uniformly at random. For each of these HTs, we generate the respective testing oracles o and run Algorithms 1 and 3 on the input $(CT\ell, o, \ell)$ and $(CT\ell, o)$, respectively. The two Algorithms are implemented in Matlab and the experiments are run on a machine with an Intel i9-9900 CPU clocked at 3.60 GHz with 64GB of RAM. The results of these measurements can be found in Table 10. The table shows that the runtime of Algorithm 3 remains under one millisecond in all cases and appears to grow only linearly in ℓ. This is due to its runtime being primarily influenced by the number of failing tests, which grows slowly with increased ℓ and numbers of test vectors in CTℓ (see, for example, Table 4). This observation is further justified by conducting HT location for HTs of length $\ell = 8$ with Algorithm 3 and the CT8 test set, which on average (again for 100 randomly selected HT activation patterns) needs only 0.2260 seconds for locating the HT. Compared to that, the runtime of Algorithm 1 seems to follow an exponential growth in ℓ, which is explained by the factor of $\Theta(\binom{k}{\ell})$ being present in the Algorithm's runtime, see also Remark 3. These large runtimes for the location of an HT of length $\ell \geq 5$ are the reason why Tables 8 and 9 are only provided for $\ell \leq 4$.

E. DISCUSSION ABOUT OTHER TECHNIQUES

The proposed method can be easily combined with other complementary techniques. First, it can be used in combination with on-chip sensors based on ring oscillators in order

TABLE 11. Measurements regarding the slow pattern location properties (i.e. patterns that can be located via Algorithm 1) of the *rand*(*CT*ℓ) arrays. The column 'loc' denotes whether all patterns can be located based on the respective array.

	$rand(T1)$	$rand(T2)$	$rand(T3)$	$rand(T4)$
ℓ	loc	loc	loc	loc
1	✗	✓	✓	✓
2	✗	✗	✓	✓
3	✗	✗	✗	✓
4	✗	✗	✗	?

to detect IC modifications that might indicate that the HT is activated. This would allow testers to locate HTs where the activation is not necessarily propagated to the primary outputs. This can be done by using the proposed test vectors in order to activate HTs that are placed close to an on-chip sensor [24]. The ring-oscillator's frequency depends, amongst others, on process variation, the local temperature, and the voltage. Thus, the oscillator frequency is changed when the operation around a sensor is altered. In particular, if the CTℓ test vectors [59] are selected to constitute the test vectors in the first step of the method presented in [24], any combinational HT with a trigger pattern of length up to ℓ can be located via Algorithm 3 based on the on-chip sensor measurements.

In addition, when a HT is activated its signal switching activity is increased, leading to more power consumption and increased electromagnetic emission. A side channel method using power consumption or electromagnetic emission side channel techniques can be performed to detect differences due to the presence of a HT [25]. Again, combining the proposed combinatorial test vectors with such a more sensitive method for HT activation allows us to locate the trigger patterns of more general HT designs.

F. SUMMARY OF THE EXPERIMENTAL EVALUATION

We first want to note that the experiments and measurements of randomly generated test sets have been conducted only for one specific random test set for each instance. It would have been desirable to conduct the experiments documented in Tables 8, 9 and 11, for all lengths of HTs and for several test sets in order to present results regarding the average and expected performance of random test sets for all $\ell = 1, \ldots, 8$. However, this would involve computationally expensive and unaffordable tasks. In this sense, the experiments conducted for random test sets should be understood as experiments with *arbitrary* test sets with a given number of tests.

So far, we have demonstrated that combinatorial testing can provide the mathematical guarantees to locate HTs. In our experiments, this was exemplified for HTs up to length $\ell = 8$. *Although in some cases random arrays appear to have similar capabilities to the ones derived from combinatorial testing they do not provide the guarantees provided by combinatorial test sets.* Moreover, the run-time comparison of Algorithms 1 and 3 highlights that Algorithm 3 is faster by several orders or magnitude.

To summarize, the experiments demonstrate that the combinatorial test sets CTℓ can locate HTs of length up to ℓ using the efficient location procedure of Algorithm 3. Further, by theory (see Remark 1 and Lemma 3) we are guaranteed that this holds for *all* HTs of length up to ℓ.

VII. THREATS TO VALIDITY

The presented combinatorial methods for HT trigger pattern identification rely on some information about the length ℓ of the pattern that shall be located. We are aware that a testers generally do not know the length of the trigger pattern that they want to identify. However, we demonstrated that an *upper bound* on the length of the inserted HT is sufficient to precisely identify it with the proposed combinatorial methods (combinatorial test sets in combination with Algorithm 3). A potential attacker is always faced with a trade-off: On one hand longer trigger patterns are more rare and thus harder to detect and locate via logic testing; on the other hand they also necessitate larger HT trigger circuits, thus consuming more area and power, making detection by physical inspection or side-channel analysis more likely. Hence, we can assume that an attacker will not use a full 128-bit pattern, but rather some pattern of length ℓ, where $\ell \ll 128$. A Tester, on the other side, is faced with the inverse problem: The tester has to select a strength t for the combinatorial test set that is high enough to locate a potential HT while avoiding excessive resource consumption. The tester's selection of the chosen strength t is likely influenced by the circuit under test and other HT detection techniques involved in the testing process. Aside from this, available resources may determine the efforts affordable for testing, which is common to all testing problems in general. For example, a tester can select the strength t based on the test suite size and the available time budget. If combinatorial test sets are precomputed and readily available, a resource consumption assessment is rather easy. Finally, experience and domain knowledge of the tester may influence the choice of t.

The work conducted in this paper shows how HTs with trigger patterns of length up to eight can be located precisely.

The threat model of this work assumes that an attacker designs his HT so that it only consumes primary inputs. This is obviously a drastic restriction, as other works [36] notably exclude primary inputs (and outputs) from their investigations. However, once more, we made these assumptions for the sake of simplicity and clarity of the experimental evaluation and in order to present the proposed methodology in a concise manner. The method proposed in this paper can also be applied more generally to any set of gates of the circuit, as long as their input can be actively controlled in order to apply combinatorial testing methods. To this extent the proposed method can be applied to sets of suspicious gates of a circuit that were previously identified using alternative approaches, such as probability analysis or the topology of the circuits layout as proposed in [37].

We want to mention that our approach to HT location scales well with the number k of gates that are modelled as subject to a potential attack and are thus represented as parameters in the combinatorial model. The number of test vectors grows logarithmically in the number of modelled gates k, which is due to the number of rows of detecting arrays growing logarithmically in the number of parameters, which was shown in [54, Th. 8.6].

For some applications, the assumption that an attacker can combine *any* ℓ of the modelled k gates as input to the trigger circuit might be too general. For example, a plausible assumption might be that the input gates consumed by the trigger circuit are within an interval of some r consecutive gates, so that the Trojan design is kept compact. In such a case, variable strength covering arrays (VCAs) [47] may be well-suited to be used as combinatorial test sets, since the notion of VCAs allows to specify more freely which interactions of parameters should be *covered*. For example, the columns of a $\mathsf{CA}(N; t, r, v)$ can be used to constitute a VCA that covers all t-way interactions of any r consecutive parameters of total k parameters. The number of rows of such a combinatorial test set is in $O(\log r)$ and hence independent from the total number of parameters k.

Nonetheless, in this work, we focus on the unrestricted case where an attacker can freely combine any ℓ input gates, as the restricted threat model described above would constitute a threat to the generality of our proposed approach. We also do not present additional experiments specific to this restricted threat model, based on test sets coming from VCAs, as this would require us to use and review further notions from combinatorial testing as well as minor changes of Algorithm 3. The authors believe that this would make it impossible to keep the paper at reasonable length and therefore defer further studies incorporating VCAs to future work.

VIII. SUMMARY

In this work, we introduced a method for identifying trigger patterns of hardware Trojans, which are triggered by combinational ℓ bit patterns (e.g. trigger circuits composed of AND-gates and NOT-gates) in the primary input. Using concepts from combinatorial testing and combinatorial fault location, our method relies only on the applied set of test vectors and the testing results in order to locate the HT trigger pattern, while the circuit under test is considered a black box. We demonstrated the effectiveness of our approach in a concrete case study, utilizing it to locate HTs with trigger patterns of length up to eight embedded in a circuit that implements the AES symmetric-key encryption algorithm with 128 bits key length. Our results show that our testing methodology can perform trigger pattern identification in a negligible amount of time while providing the guarantee of locating any HT with trigger pattern of length up to eight.

IX. OUTLOOK AND FUTURE WORK

The *location* of HTs is an important step in HT testing, as it can enable several additional options depending on the scenario. This includes restricted usage of the infected circuit (e.g. with a reduced key space) in case the identified HT is known to be the only malicious component. Knowledge about the operation of the HT may even be used against the attacker, e.g. by leaking information on purpose. In general, the *location* of HTs can be very valuable, as it allows for understanding the purpose of an attack and thus can help to gain insights about the attacker's intention and capabilities. In a different scenario, the identification of combinational HTs might enhance the analysis of the infected circuit, especially when logic testing is used in combination with other testing techniques as in [25], since the identified HT can be excited or completely avoided on purpose. For example, avoiding an already identified combinational HT may be helpful when subsequent testing for sequential HTs is conducted.

We believe that combinatorial methods like those presented in this paper can give rise to techniques and tools for HT testing that are complementary to existing approaches. In general, the hybridization of the proposed combinatorial method with existing and established HT location methods (such as side-channel analysis and inspection) is considered future work. In this context, it is of special interest to investigate and overcome potential gaps in the literature regarding the location of HTs of specific length. Such gaps may emerge when combinatorial methods can no longer locate HTs due to the trigger pattern being too long, and inspection methods fail to locate HTs due to the trigger circuit (and the trigger pattern) being too small.

As future work, one might seek lift to lift the restriction of locating HTs with trigger circuits consisting exclusively of AND-gates and NOT-gates. If we apply combinatorial methods for the location of multiple FITs, e.g. when using (d, t)-detecting arrays with $d \geq 2$ for the testing of integrated circuits, we can also capture HTs that additionally use OR-gates in their trigger circuit. For example, an HT with a trigger circuit that combines two of the trigger circuits as considered in this paper (see e.g. Fig. 1) via an OR-gate (on the highest level of its concrete syntax tree) can be located with a $(2, t)$-detecting array of appropriate strength t. Consider an HT with a trigger logic of the form $(\neg g_{25} \wedge g_{38}) \vee (\neg g_{95} \wedge g_{115} \wedge \neg g_{127})$ (see Fig. 4). This Trojan can be located applying combinatorial testing based on a $(2, \bar{3})$-detecting array: the HT is triggered, when the pattern 01 or 010 appears in the positions 25 - 38 or 95 - 115 - 127 respectively. In terms of combinatorial testing, this means that we have to locate the failure inducing $\leq$3-way interactions $\{(25, 0), (38, 1)\}$ and $\{(95, 0), (115, 1), (127, 0)\}$ - which we are guaranteed to find when testing with a binary $(2, \bar{3})$-detecting array. The logic of any combinational circuit is (logically) equivalent to a circuit that uses exclusively OR-gates, AND-gates and NOT-gates, which is provided by considering the disjunctive normal form (DNF) of the trigger logic. Since our proposed method does not rely on the actual implementation of the HT trigger, but merely on its logic, this means that the work presented in this paper can be generalized for combinational HTs with arbitrary trigger circuit logic. As straightforward

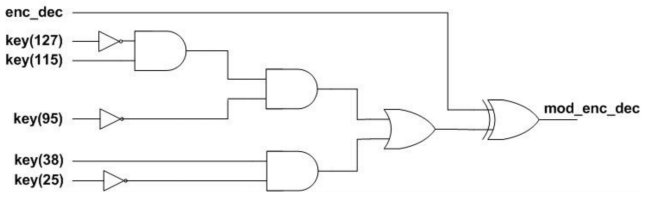

FIGURE 4. An example of an HT with a generalized trigger logic, activated by the signals 01 in position 25-38 or by the signals 010 in positions 95 - 115 - 127.

as the combinatorial modelling of such HTs might seem, the challenges arising in this context is caused by the fact that the existing work on the efficient generation of combinatorial test sets enhancing non-adaptive fault location for multiple failure inducing t-way interactions is rather limited [63]. Only a few algorithms capable of generating such combinatorial test sets in a near-optimal manner are known. Additionally, these algorithms have only been investigated for a small number of failure inducing t-way interactions of small strength [64].

Another important aspect of our future work is the investigation of combinatorial testing methods for the detection and location of sequential HTs [16]. Addressing this issue likely requires to enlarge the combinatorial testing toolbox from CAs and detecting arrays to approaches used in combinatorial event sequence testing, including sequence covering arrays [65] and algorithms for test sequence generation [66]. Adequate adapted translations of notions appearing in HT testing to the notions of sequential combinatorial testing should make it possible to derive suitable combinatorial test sets for testing against HTs with sequential trigger patterns (e.g. counter-based time bombs [30], or finite-state machine based triggers [67]). For the purpose of testing for HTs that combine combinational with sequential trigger logic, e.g. counter-based time bombs with trigger circuits that consume multiple gates, the notion of CAs of higher index λ might prove useful. These are CAs that guarantee that any t-way interaction is covered at least $\lambda \geq 1$ times; see [45] for their definition and the tool presented in [68] for their generation.

Finally, as another avenue of future work, we could integrate our methods into a comprehensive framework. We have already automated the individual steps of test set generation, test execution, and the analysis of the testing results for the work in this paper, leaving the interfaces between these steps as the main tasks to be automated. We envision this framework as a means to facilitate our and others future research endeavors regarding combinatorial methods for HT testing.

ACKNOWLEDGMENT

The authors are thankful to the anonymous reviewers for their constructive comments that improved the quality and presentation of the paper. They are especially thankful to Manuel Leithner for proofreading and resolving several grammatical issues. SBA Research (SBA-K1) is a COMET Centre within the framework of COMET - Competence Centers for Excellent Technologies Programme. The COMET Programme is managed by FFG.

REFERENCES

[1] R. George, "Why we should worry about the supply chain," *Int. J. Crit. Infrastructure Protection*, vol. 11, pp. 22–23, Dec. 2015.
[2] C. Ruppersberger and M. Rogers. (Oct. 2012). *Investigative report on the U.S. national security issues posed by chinese telecommunications companies Huawei and ZTE. U.S. House of Representatives, 112th Congress, 2nd Session*. Accessed: Aug. 8, 2021. [Online]. Available: https://intelligence.house.gov/sites/democrats.intelligence.house.gov/files/huawei-zte%20investigative%20report%20(final)_0.pdf
[3] S. Mitra, H.-S. P. Wong, and S. Wong, "The Trojan-proof chip," *IEEE Spectrum*, vol. 52, no. 2, pp. 46–51, Feb. 2015.
[4] Z. Abbany. (Jul. 8, 2015). *Has Germany's Patriot missile system been hacked?*. Accessed: Aug. 8, 2021. [Online]. Available: https://p.dw.com/p/1FvEy
[5] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Cryptographic Hardware and Embedded Systems*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer, 2012, pp. 23–40.
[6] C. T. Lopez. (May 19, 2020). *DOD Adopts 'Zero Trust' Approach to Buying Microelectronics*. U.S. Department of Defense. Accessed: Aug. 8, 2021. [Online]. Available: https://www.defense.gov/News/News-Stories/Article/Article/2192120/dod-adopts-zero-trust-approach-to-buying-microelectronics/
[7] S. Garg, "Inspiring trust in outsourced integrated circuit fabrication," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1128–1228.
[8] Y. Liu, C. Bao, Y. Xie, and A. Srivastava, "Introducing TFUE: The trusted foundry and untrusted employee model in IC supply chain security," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[9] J. Dofe, J. Frey, and Q. Yu, "Hardware security assurance in emerging iot applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 2050–2053.
[10] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and internet-of-things systems," *Proc. IEEE*, vol. 106, no. 1, pp. 9–20, Jan. 2018.
[11] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions," in *Proc. IEEE Int. High Level Design Validation Test Workshop*, Nov. 2009, pp. 166–171.
[12] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "FPGA trojans through detecting and weakening of cryptographic primitives," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 34, no. 8, pp. 1236–1249, Aug. 2015.
[13] A. Jain and U. Guin, "A novel tampering attack on AES cores with hardware trojans," in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, Sep. 2020, pp. 77–82.
[14] S. Ghandali, T. Moos, A. Moradi, and C. Paar, "Side-channel hardware trojan for provably-secure SCA-protected implementations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 6, pp. 1435–1448, Jun. 2020.
[15] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *J. Hardw. Syst. Secur.*, vol. 1, no. 1, pp. 85–102, Mar. 2017, doi: 10.1007/s41635-017-0001-6.
[16] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan. 2010.
[17] H. Salmani, "Hardware trojan attacks and countermeasures," in *Fundamentals IP SoC Security: Design, Verification, Debug*, S. Bhunia, S. Ray, and S. Sur-Kolay, Eds. Cham, Switzerland: Springer, 2017, pp. 247–276, doi: 10.1007/978-3-319-50057-7_10.
[18] H. Li, Q. Liu, and J. Zhang, "A survey of hardware trojan threat and defense," *Integration*, vol. 55, pp. 426–437, Sep. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926016000067
[19] S. R. Rajendran, R. Mukherjee, and R. S. Chakraborty, "SoK: Physical and logic testing techniques for hardware trojan detection," in *Proc. 4th ACM Workshop Attacks Solutions Hardw. Secur.*, New York, NY, USA, Nov. 2020, pp. 103–116, doi: 10.1145/3411504.3421211.
[20] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10796–10826, 2020.
[21] P. Kitsos, D. E. Simos, J. Torres-Jimenez, and A. G. Voyiatzis, "Exciting FPGA cryptographic trojans using combinatorial testing," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 69–76.

[22] A. G. Voyiatzis, K. G. Stefanidis, and P. Kitsos, "Efficient triggering of trojan hardware logic," in *Proc. IEEE 19th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2016, pp. 1–6.

[23] Trust-HUB.org. *Chip-level Trojan Benchmarks*. Accessed: Aug. 8, 2021. [Online]. Available: https://www.trust-hub.org/#/benchmarks/chip-level-trojan

[24] L. Pyrgas and P. Kitsos, "A hybrid FPGA trojan detection technique based-on combinatorial testing and on-chip sensing," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, and P. C. Diniz, Eds. Cham, Switzerland: Springer, 2018, pp. 294–303.

[25] A. P. Fournaris, L. Pyrgas, and P. Kitsos, "An efficient multi-parameter approach for FPGA hardware trojan detection," *Microprocessors Microsyst.*, vol. 71, Nov. 2019, Art. no. 102863. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933118305106

[26] Y. Huang, S. Bhunia, and P. Mishra, "MERS: Statistical test generation for side-channel analysis based trojan detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, Oct. 2016, pp. 130–141, doi: 10.1145/2976749.2978396.

[27] C. Nigh and A. Orailoglu, "AdaTrust: Combinational hardware trojan detection through adaptive test pattern construction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 544–557, Mar. 2021.

[28] S. Dupuis, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "Protection against hardware trojans with logic testing: Proposed solutions and challenges ahead," *IEEE Design Test*, vol. 35, no. 2, pp. 73–90, Apr. 2018.

[29] M.-L. Flottes, S. Dupuis, P.-S. Ba, and B. Rouzeyre, "On the limitations of logic testing for detecting hardware trojans horses," in *Proc. 10th Int. Conf. Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Apr. 2015, pp. 1–5.

[30] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards trojan-free trusted ICs: Problem analysis and detection scheme," in *Proc. Design, Autom. Test Eur.*, Mar. 2008, pp. 1362–1365.

[31] 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*, C. Clavier and K. Gaj, Eds. Berlin, Heidelberg: Springer, 2009, pp. 396–410.

[32] M. A. Nourian, M. Fazeli, and D. Hely, "Hardware trojan detection using an advised genetic algorithm based logic testing," *J. Electron. Test.*, vol. 34, no. 4, pp. 461–470, Aug. 2018, doi: 10.1007/s10836-018-5739-4.

[33] X. Chuan, Y. Yan, and Y. Zhang, "An efficient triggering method of hardware trojan in AES cryptographic circuit," in *Proc. 2nd IEEE Int. Conf. Integr. Circuits Microsyst. (ICICM)*, Nov. 2017, pp. 91–95.

[34] H. Salmani, "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 338–350, Feb. 2017.

[35] K. Huang and Y. He, "Trigger identification using difference-amplified controllability and dynamic transition probability for hardware trojan detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3387–3400, 2020.

[36] S. A. Islam, F. Islam Mime, S. M. Asaduzzaman, and F. Islam, "Socio-network analysis of RTL designs for hardware trojan localization," in *Proc. 22nd Int. Conf. Comput. Inf. Technol. (ICCIT)*, Dec. 2019, pp. 1–6.

[37] S. Dupuis, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Identification of Hardware Trojans triggering signals," in *Proc. First Workshop Trustworthy Manuf. Utilization Secure Devices*, Avignon, France, May 2013. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00991360

[38] M. Grindal and J. Offutt, "Input parameter modeling for combination strategies," in *Proc. 25th Conf. IASTED Int. Multi-Conf., Softw. Eng.*, Anaheim, CA, USA, 2007, pp. 255–260.

[39] L. Hu, W. E. Wong, D. R. Kuhn, and R. N. Kacker, "How does combinatorial testing perform in the real world: An empirical study," *Empirical Softw. Eng.*, vol. 25, no. 4, pp. 2661–2693, Jul. 2020, doi: 10.1007/s10664-019-09799-2.

[40] L. S. G. Ghandehari, M. N. Bourazjany, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Applying combinatorial testing to the Siemens suite," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2013, pp. 362–371.

[41] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Trans. Softw. Eng.*, vol. 41, no. 9, pp. 901–924, Sep. 2015.

[42] M. Bures and B. S. Ahmed, "On the effectiveness of combinatorial interaction testing: A case study," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Jul. 2017, pp. 69–76.

[43] D. Jarman, R. Smith, G. Gosney, L. Kampel, M. Leithner, D. Simos, R. Kacker, and R. Kuhn, "Applying combinatorial testing to large-scale data processing at adobe," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2019, pp. 190–193.

[44] D. Kuhn, R. Kacker, and Y. Lei, *Introduction to Combinatorial Testing* (Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series). Boca Raton, FL, USA: CRC Press, 2013.

[45] C. J. Colbourn and J. H. Dinitz, *Handbook of Combinatorial Designs* (Discrete Mathematics and its Applications). 2nd ed. Boca Raton, FL, USA: CRC Press, 2007.

[46] L. Moura, J. Stardom, B. Stevens, and A. Williams, "Covering arrays with mixed alphabet sizes," *J. Combinat. Designs*, vol. 11, no. 6, pp. 413–432, 2003, doi: 10.1002/jcd.10059.

[47] S. Raaphorst, L. Moura, and B. Stevens, "Variable strength covering arrays," *J. Combinat. Designs*, vol. 26, no. 9, pp. 417–438, Sep. 2018, doi: 10.1002/jcd.21602.

[48] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith, "Constraint models for the covering test problem," *Constraints*, vol. 11, nos. 2–3, pp. 199–219, Jul. 2006.

[49] L. Kampel and D. E. Simos, "A survey on the state of the art of complexity problems for covering arrays," *Theor. Comput. Sci.*, vol. 800, pp. 107–124, Dec. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397519306486

[50] MaTRIS. *CAgen*. Accessed: Sep. 18, 2021. [Online]. Available: https://matris.sba-research.org/tools/cagen

[51] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.

[52] L. S. Ghandehari, J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn, "BEN: A combinatorial testing-based fault localization tool," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2015, pp. 1–4.

[53] X. Niu, C. Nie, H. Leung, Y. Lei, X. Wang, J. Xu, and Y. Wang, "An interleaving approach to combinatorial testing and failure-inducing interaction identification," *IEEE Trans. Softw. Eng.*, vol. 46, no. 6, pp. 584–615, Jun. 2020.

[54] C. J. Colbourn and D. W. McClary, "Locating and detecting arrays for interaction faults," *J. Combinat. Optim.*, vol. 15, no. 1, pp. 17–48, Jan. 2008, doi: 10.1007/s10878-007-9082-4.

[55] C. Martínez, L. Moura, D. Panario, and B. Stevens, "Locating errors using ELAs, covering arrays, and adaptive testing algorithms," *SIAM J. Discrete Math.*, vol. 23, no. 4, pp. 1776–1799, Jan. 2010, doi: 10.1137/080730706.

[56] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *IT Prof.*, vol. 10, no. 3, pp. 19–23, May 2008.

[57] D. E. Simos, R. Kuhn, A. G. Voyiatzis, and R. Kacker, "Combinatorial methods in security testing," *Computer*, vol. 49, no. 10, pp. 80–83, Oct. 2016.

[58] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Advances in Cryptology*, C. Boyd, Ed. Berlin, Heidelberg: Springer, 2001, pp. 239–254.

[59] MaTRIS. *HT Location Arrays*. Accessed: Sep. 18, 2021. [Online]. Available: https://matris.sba-research.org/data/HTlocation/

[60] N. Lesperance, S. Kulkarni, and K.-T. Cheng, "Hardware trojan detection using exhaustive testing of k-bit subspaces," in *Proc. 20th Asia South Pacific Design Automat. Conf.*, Jan. 2015, pp. 755–760.

[61] Tang and Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Trans. Comput.*, vols. C–32, no. 12, pp. 1145–1150, Dec. 1983.

[62] C. J. Colbourn. *Covering Array Tables for t=2,3,4,5,6*. Accessed: Aug. 8, 2021. [Online]. Available: http://www.public.asu.edu/~ccolbou/src/tabby/catable.html

[63] C. J. Colbourn and V. R. Syrotiuk, "There must be fifty ways to miss a cover," in *50 years of Combinatorics, Graph Theory, and Computing*, 1st ed. Boca Raton, FL, USA: CRC Press, 2019, pp. 319–333, ch. 18.

[64] C. J. Colbourn and V. R. Syrotiuk, "Coverage, location, detection, and measurement," in *Proc. IEEE 9th Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2016, pp. 19–25.

[65] Y. M. Chee, C. J. Colbourn, D. Horsley, and J. Zhou, "Sequence covering arrays," *SIAM J. Discrete Math.*, vol. 27, no. 4, pp. 1844–1861, Jan. 2013, doi: 10.1137/120894099.

[66] L. Yu, Y. Lei, R. N. Kacker, D. R. Kuhn, and J. Lawrence, "Efficient algorithms for t-way test sequence generation," in *Proc. IEEE 17th Int. Conf. Eng. Complex Comput. Syst.*, 2012, pp. 220–229.

[67] S. Yu, W. Liu, and M. O'Neill, "An improved automatic hardware trojan generation platform," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 302–307.
[68] M. Wagner, K. Kleine, D. E. Simos, R. Kuhn, and R. Kacker, "CAGEN: A fast combinatorial test generation tool with support for constraints and higher-index arrays," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Oct. 2020, pp. 191–200.

LUDWIG KAMPEL received the master's degree in technical mathematics with focus on discrete mathematics from the Technical University of Vienna, where he is currently pursuing the Ph.D. degree in technical informatics. He is currently a Senior Researcher with the Mathematics for Testing Reliability and Information Security (MATRIS) Group at SBA Research. His research interests include discrete mathematics, with an emphasis on combinatorial designs and error-correcting codes. His work has a strong focus on the application of results in these fields to practical problems of information security, such as to software testing.

PARIS KITSOS (Senior Member, IEEE) received the B.Sc. degree in physics and the Ph.D. degree in electrical and computer engineering from the University of Patras, in 1999 and 2004, respectively. From 2014 to 2019, he was an Assistant Professor and an Associate Professor with the TEI of Western Greece. He is currently an Associate Professor with the Electrical and Computer Engineering Department, University of the Peloponnese, and the Head of the Electronics Circuits, Systems and Applications Laboratory (ECSA Laboratory, https://sites.google.com/view/ecsalab) and a Collaborating Academic Faculty (since 2014) with the Industrial Systems Institute (ISI, www.isi.gr). He has participated in many EU and national research programs projects both as a Senior Researcher and as a WP Leader in the areas of VLSI design, secure hardware design, and embedded systems. He has up to 110 research publications, in international journals, conferences, book chapters, and books in the above-mentioned research areas and he has received more than 1500 non-self citations. Finally, he has organized numerous special issues and special sessions in international journals and conferences in the above-mentioned areas. His research interests include FPGA and ASIC design, microprocessor and microcontroller system design, system-on-chip design, digital IC design for security, digital signal processing, and ML systems.

DIMITRIS E. SIMOS (Member, IEEE) is currently a Key Researcher for the applied discrete mathematics for information security research area with SBA Research, Vienna, Austria, and leads its Mathematics for Testing, Reliability and Information Security (MATRIS) Research Group, since 2017. He is also an Associate Professor (Non-Tenured Track, Venia Docendi for Applied Computer Science) with the Graz University of Technology as of 2021. He further holds a Guest Researcher appointment with the U.S. National Institute of Standards and Technology (NIST), where he is also a Research Member of its Working Group on "Automated Combinatorial Testing for Software" (ACTS). During his career, he has (co)authored over 100 articles in discrete mathematics and their applications to computer science. His research interests include combinatorial designs and their applications to software testing, combinatorial testing in particular, symbolic computation and optimization algorithms, and all aspects of information security. He is a member of the Editorial Board of two Springer journals (*MCS* and *ORFO*). He has been awarded the rank of fellow of the Institute of Combinatorics and its Applications (FTICA), in 2012. He (co)-organized or (co)-chaired many international scientific conferences, such as MACIS, ACA, LION, CAI, and QRS, and workshops, such as IWCT and MoCrySEn.