Exciting FPGA Cryptographic Trojans using Combinatorial Testing

Paris Kitsos*[†], Dimitris E. Simos[‡], Jose Torres-Jimenez[§], Artemios G. Voyiatzis[‡][†]
* Computer and Informatics Engineering Department, TEI of Western Greece, Greece
Email: pkitsos@ieee.org
[†] Industrial Systems Institute/RC 'Athena', Greece
Email: bogart@isi.gr
[‡] SBA Research, Vienna, Austria
Email: dsimos@sba-research.org
[§] CINVESTAV-Tamaulipas, Ciudad Victoria, Tamaulipas, Mexico
Email: jtj@cinvestav.mx

***Abstract*—Contemporary hardware design shares many similarities with software development. The injection of malicious functionality (Trojans) in FPGA designs is a realistic threat. Established techniques for testing correctness do not cope well with Trojans, since Trojans are not captured in the system model. Furthermore, a well-designed Trojan activates under rare conditions and can escape detection during testing. Such conditions cannot be exhaustively searched, especially in the case of cryptographic core implementations with hundreds of inputs.**

In this paper, we explore the applicability of a prominent combinatorial strategy, namely combinatorial testing, for FPGA Trojan detection. We demonstrate that combinatorial testing provides the theoretical guarantees for exciting a Trojan of specific lengths by covering all input combinations. Our findings indicate that combinatorial testing constructs can improve the existing FPGA Trojan detection capabilities by reducing significantly the number of tests needed. Besides the foundations of our approach, we also report on first experiments that indicate its practical use.

***Keywords*—Combinatorial testing, hardware Trojan horses, detection techniques, FPGA design, cryptography engineering**

I. Introduction

Hardware design nowadays exhibits many similarities with software development. Software developers reuse existing code and libraries, often offered by third parties, in order to accelerate development and improve their offerings. Similarly, hardware vendors, especially in the embedded systems domain, reuse soft and hard intellectual property (IP) from third-party organizations and integrate such IP into their products and offerings. These might end up in even more complex designs, creating a rather deep supply chain until a final product reaches its intended customer. Accounting for this complexity, it becomes harder and harder to test if a downstream provider might have installed an undesired functionality in their offerings or they can be fully trusted.

Hardware Trojan horses (or simply "Trojans" in this paper) refer to malicious functionality inserted in a hardware circuit. A lot of concern is raised lately for the trustworthiness of integrated circuits that are manufactured in remote silicon foundries, using components from third, possible untrusted, parties [1]–[3]. In the case of computer software, such concerns are already confirmed [4]. In the case of hardware, it is an acknowledged threat in ICT supply chain for at least the USA [5].

From an attacker point of view, the two major obstacles to overcome for inserting a Trojan relate to the *moment* of the insertion during the design and fabrication workflow and to the *controlability* of the Trojan activation while remaining hidden during the various tests of the circuits.

From a defender point of view, applying a process of secure development and rigorous testing at each production stage, such as the one proposed in [6], can reveal circuit manipulations early enough so as avoid disastrous effects after fabrication.

Hardware designers can incorporate run-time monitoring in the circuits so as to detect malicious actions [7]. They can also apply testing before deployment (test-time detection). While the attackers need just one means to activate the Trojan, the defenders must test all possible means for ensuring that a circuit is Trojan-free and trustworthy.

A hardware Trojan horse comprises two parts: the trigger and the payload. The trigger part is always active so as to detect the activation sequence. Once received, the malicious functionality of the payload part is activated. There can be many types of Trojans, depending on the malicious function they implement and the point of insertion [8]. A Trojan may act as a "kill switch" disabling parts of the circuit, it may degrade the performance of the system, or it may leak information through some side channel, like carrier modulation [9]. The practical feasibility of an S-Box substitution attack for AES on FPGA designs is already demonstrated [10]. The processor design is not the only place to hide the malicious functionality; even a network card can be an attractive target for inserting a Trojan [11].

The attackers may not have control on the internal workings of a circuit once it is fabricated. Rather, they can interface with it through its inputs. Assuming that n such inputs are available to the attacker, there are 2^n possible activation vectors to choose from. In the case of a circuit that implements

Authors are listed in alphabetical order.

cryptography e.g., the AES algorithm, these values range in hundreds of inputs (e.g., 128-256 inputs for the plaintext and the key for an AES cryptographic core) and exhaustive testing becomes intractable.

These assumptions can be relaxed by observing that an attacker must choose carefully the Trojan *activation sequence* i.e., some rare patterns so as to activate the Trojan. The attacker is faced with a tradeoff: on the one hand, a longer (bitwise) activation pattern will be more rare under normal operation; on the other hand, the activation circuitry (Trojan trigger) will consume more area and power leading to detection by inspection or by side-channel analysis. Thus, we can assume that attacker will not use an exact n-bit pattern but rather some k-bit where $k << n$. Under this realistic assumption, the question for testing is formulated as *how to efficiently test all the possible k-bit input vectors for Trojan activation.*

In this paper, we focus on the efficient test vector generation. The aim is to increase the trigger signal generation probability up to 100%. We explore the applicability of *combinatorial testing* for FPGA Trojan detection and we demonstrate that combinatorial testing provides the required theoretical guarantees for exciting a Trojan of specific lengths. Our findings indicate that combinatorial testing constructs can improve the existing FPGA Trojan detection capabilities by reducing significantly the number of tests needed. Besides the foundations of our approach, we also report on first experiments that indicate its practical use.

The rest of the paper is organized as follows. Section II describes hardware Trojan horses and detection techniques. Section III introduces combinatorial testing as a novel methodology for hardware Trojan horse detection, while Section IV describes the case study and presents the experimental results showcasing its effectiveness. Finally, Section V concludes the paper and discusses future directions of work.

II. Trojan detection approaches

A. Types of Trojans

The Trojans can be classified in two main categories based on their activation logic: combinational and sequential Trojans. The combinational ones ("c-Trojans") use logic gates that combine input signals; once the correct combination is applied, the Trojan payload is activated. The sequential ones ("s-Trojans") additionally use memory and counters for states; once the correct state is reached, the Trojan payload is activated. Assuming n input bits and m flip-flops for state storage, the attackers can choose among 2^{n+m} possible states for defining a trigger signal.

The c-Trojans are always-on. Less area is required to implement them rendering harder the detection based on the variations of the physical parameters. The small-area logic does not affect significantly the power consumption, voltage drop, or thermal behavior. Moreover, it is easier to attribute such small distortions in the process variation envelope. On the other hand, the detection of a c-Trojan is easier in the sense that less tries are necessary to detect its existence. Once a correct combination is applied in the inputs of the circuit, the c-Trojan always activates its payload. In this sense, a c-Trojan is more responsive to correct excitation.

The s-Trojans are also always-on but use additional state logic to trigger the payload after the correct inputs are applied many times. A greater area is required to implement an s-Trojan rendering harder the efforts to hide its existence in the process variation. The s-Trojan is less responsive to correct input, as the payload activation logic is more complex. It can utilize less rare and more "innocent-looking" inputs so as to build rare sequences of input patterns.

Neither the trigger, nor the payload parts of a Trojan require a lot of space in the circuit. In one experiment, a c-Trojan design with a trigger monitoring 32 input bits for AES that causes a denial-of-service attack covered just 0.19% of the FPGA slices [12]. In the same experiment, an s-Trojan design incorporating a counter before activation covered just 0.36% of the FGPA slices, while the AES implementation itself covered more than 38%. In another experiment, a Trojan design used one gate and two input signals as a payload to assist fault-based cryptanalysis of the AES algorithm [13]. The erroneous ouput produced by the circuit was then used to launch a differential fault attack and recover the whole AES key.

B. Trojan detection

The reliable detection of Trojans is a very important topic. Strong evidence must be collected to undeniably prove infection or lack of it, even in courts. Criminals already accuse software Trojan infections so as to deny their actions [14]. It may be the case that in the future they blame a piece of hardware too. Thus, the need for reliable detection of a Trojan spans beyond the obvious cases of industrial espionage and national security.

Any additional circuit logic that realizes a Trojan leaves some traces, at the physical level (e.g., change in circuit layout, power leakage and consumption, emissions, or path delay execution) and the logic level (e.g., incorrect output values). Depending on the characteristics of the Trojan, it may not always be possible to detect its presence. Limitations in measurement granularity allow sophisticated Trojans to hide their existence within the process variation during manufacturing [15]. Thus, a power trace of a circuit that includes a Trojan can be indistinguishable from a Trojan-free one.

In the case the Trojan activity traces are strong enough, they can used to characterize the circuit, using techniques of "side-channel analysis". These techniques rely on the existence of a trusted, Trojan-free (also called "golden-chip") profile with which such comparisons are drawn for identifying unjustifiable variations. The availability of golden chips and the level of trust we can be put on them is debated [16]. In principle, variations can be observed in points such the circuit layout, the leakage current, the dynamic power due to switching activity, the output delays, and the electromagnetic (EM) emissions [12]. However, the efficiency of side-channel analysis for reliable detection is also questioned [17].

Hybrid detection approaches use a combination of logic and physical testing. Some Trojans may hide their existence in the output by, for example, producing new output signals that are not included in the original design. Such a Trojan may exhibit increased power consumption, heat dissipation, or EM emissions. Its detection can be simplified if the logic testing can include input vectors that increase the Trojan activity.

Monitoring and targetting both the functional and the physical behavior of a circuit may improve the detection rate of a hardware Trojan horse.

Additional on-chip logic, such as ring oscillators and ring oscillator networks (RONs), can improve the signal-to-noise ratio, provided that they are placed close to the inserted Trojan. Even with such enhancements, it is always possible that small Trojans remain mostly undetected and a large number of false positives are produced. The original proposal of RON for Trojan detection achieves less than 50% detection with 30% false positives [18]. An appropriate post-processing technique can severely improve the Trojan detection accuracy, as shown in [19]. The researchers there fabricated 33 chips in 90 nm technology. The chips contained the 8 sequential and 15 combinational on-demand activated Trojans described in [18]. In this experimental setup, the researchers propose the use of a Genetic Algorithm (GA) and a Support Vector Machine (SVM) and compare the accuracy of their proposal against the commonly used Principal Component Analysis (PCA) [9]. In their experimental setup, PCA achieved a detection accuracy of 76.8%, while a combined GA+SVM achieved 99.6% accuracy by selecting the optimal RO. The effect of RO length on detection sensitivity is also discussed in [7].

C. Test pattern generation for Trojan excitation

A common problem regarding the detection of a Trojan is the excitation of the infected part of the circuit. We can assume that the attacker has chosen a signal input pattern (signature) that seldom occurs during the normal operation of the circuit. If such a signature does not exist, the attacker could have not installed a silent Trojan in first place and thus, it would be easy to detect an infection.

The problem of excitation can be defined as the problem "*to efficiently derive a set of input vectors so that when they are applied to a system-under-test (SUT), multiple signal transitions (toggles) occur in the infected area*". This benefits both physical (side-channel) and logic testing. For the physical part, more toggles result in increased power consumption, if the infected part of the circuit is activated, indicating the infection. For the logic testing, the toggles will result in erroneous output, guiding the detection process.

An efficient derivation method should produce and use a small set of input vectors. This is necessary so as to minimize the test time per circuit, a crucial parameter when batches of circuits must be checked. The problem of generating patterns that can directly trigger Trojan's functionality is considered NP-hard [20]. Exhaustive testing is not an option in most cases, as the number and combinations of possible inputs and internal states is intractable. A reduced set of input patterns that maximizes the probability of activating the Trojan is thus desirable [21].

In order to reduce the search space into a tractable size, an assumption made in the published literature states that the attackers will choose nodes with rare signal values (e.g., a node that outputs in most cases "0") to plug their triggering logic so as to avoid detection during normal testing. Thus, a test pattern generation algorithm for Trojan detection should focus on generating inputs that produce the rare values in the internal nodes.

The automatic test pattern generation (ATPG) is an initial approach so as to increase the test *coverage* of the search space. However, it is not efficient, as a large number of vectors are generated to achieve the necessary coverage. Furthermore, such approaches rely on circuit models that do not account for the inserted logic and thus, cannot guide testing towards uncovering its presence and cannot guarantee the activation of the Trojan [21], [22]. Hence, there is a need for techniques dedicated to Trojan detection that aim at finding a set of test patterns, which maximizes the probability of triggering a Trojan while the Trojan itself is not yet introduced in the netlist to be analyzed [23].

Design-for-trust methods can assist both Trojan detection and circuit protection against Trojan insertion [24]. This is achieved by increasing the *controllability* and *observability* of internal signals [25], by obfuscating or encrypting the circuit operations [26], [27], and by changing the probabilities of signals (e.g., by inserting dummy scan flip-flops [28]).

MERO is a statistical approach for test pattern generation [29]. The MERO technique tries to maximize the probability of Trojan activation, while drastically reducing the number of vectors used. The basic concept of MERO is to detect low-probability conditions at the internal nodes of a circuit and then derive an optimal set of vectors that can trigger each of these nodes multiple times (e.g., at least N times, where N is a given parameter). Simulation results show that MERO can achieve comparable Trojan detection coverage with an 85% reduction in the set of applied vectors compared to random patterns.

MERO does not come without limitations, as it execution time for deriving the vectors can be very large [30]. Another approach is to model (approximate) transition probabilities based on the geometric distribution rather than simulating the circuit operation [28]. This can speed up significantly the vector generation process. A practical implementation of this technique along with experimental results for ASICs are described in [30].

Shreedhat *et al.* propose n-excitation as metric i.e., exciting every node at least n times during testing [31]. They suggest 0.1% as a threshold for rare nodes but they do not describe an efficient method on how to adapt testing so as to increase activity on those rare nodes. Furthermore, they discuss the dangers of incorporating testability signals - these signals can also be used to activate a Trojan. For example, a scan-enable signal that is used to activate the testing mode of a circuit, can be also used to notify the Trojan to hide its operation. We note that is a close analogy with software malware, where their authors incorporate logic to detect that the malware executes in a testing environment and adjust the behaviour of the software in order to remain undetected [32], [33].

Mingfu *et al.* propose a Monte Carlo based vector generation method that converges fast, even when the Trojan occupies less than 0.006% of the ISCAS89 benchmark suites [20]. The method uses two metrics so as to detect if an SUT is Trojan-free or not. The first is a direct comparison of outputs with those of a golden chip. The second is the percentage difference in the signal signature (power dissipation) between the SUT and the golden chip.

Dupuis *et al.* propose a testing procedure for exciting stealthy Trojans [23]. Three criteria are used to identify potential points of Trojan insertion by grouping signals based on: a) signal's controllability, b) signal's time margin (slack time), and c) available space in proximity. A commercial ATPG software is used to find an input pattern that can trigger each signal group and successful experiments are reported on two ISCAS benchmarks. Interestingly, when the signal controllability constraint is removed, it is still possible to *compose a rare triggering condition of signals that do not have individually a low controllability* (i.e., exhibit rare values). Thus, the assumption of rareness is challenged.

Lesperance *et al.* follow a different approach and they assume that the attackers cannot control or observe the circuit internals [22]. Under this assumption, the attackers cannot analyze the signal rareness at each circuity node. Thus, a test procedure should not be biased towards generating test vectors that produce rare values in the hope that these vectors will activate the Trojan. Rather, the procedure should focus on guaranteeing coverage of the input state space, independently of the individual signal rareness. A test suite generation approach is proposed for both c- and s-Trojans and coverage trade-offs are discussed for the cases when all test vectors cannot be applied due to time and budget constraints. However, the authors report only results of the mathematical analysis.

D. Threat model

We assume the implementation of cryptography in the form of hardware IP cores where the attacker has managed to implant a Trojan and can activate it using some k bits of the plaintext or key input. The hardware core may incorporate in the design hardware Trojan prevention mechanisms, such as ring oscillators [7].

The attacker can control the key or the plaintext input and can observe the ciphertext output. Further, we assume that the attacker combines only a few signals for the activation, using either combinational or sequential logic, in order to remain undetected.

III. Combinatorial Testing Methodology

We aim to design a testing workflow that can excite a Trojan using as few test vectors as possible. We denote with n the total number of the input signals (bits) available to the attackers and with k the number of bits they use to activate the Trojan. A naïve approach is to generate and test all the possible $\binom{n}{k} \times 2^k$ vectors. This number can be even greater than all the possible 2^n vectors. As a lower bound, if we assume to know which k input bits the attacker is using but not their exact values, then there are 2^k possible vectors to test.

We denote with $T(n, k)$ the set containing the minimum number of test vectors that can exhaustively cover all possible activation sequences (e.g., they cover all the $k-$ subspaces). Then, $2^k \leq |T(n, k)| \leq 2^n$. However, it is not straightforward how to determine the $T(n, k)$ or even its size [22].

A number of combinatorial strategies have been devised to help testers choose subsets of input combinations that would maximize the probability of detecting faults: random testing [34], each-choice and base choice [35], antirandom [36], and t-wise testing strategies, with pairwise testing ($t = 2$) being the most prominent among these. Pairwise testing ensures that all possible input pairs will be covered *at least once*. However, pairwise testing is not sufficient in our case since it cannot guarantee full coverage of the input space when the Trojan activation sequence depends on more than two input signals.

Combinatorial testing can model dependencies of inputs that involve more than two parameters and can produce a *test suite* for t-wise testing i.e., a set of *test cases* (input vectors) with mathematical guarantees of *input space coverage* [37]. Combinatorial testing has been successfully applied for testing (critical) software systems in large organizations [38]. It is an already proven method for security testing of large-scale software systems [39], [40].

We explore in the following the applicability of the combinatorial testing principles for detecting hardware Trojans based on the assumption that the Trojan uses a bounded number of the available circuity inputs for its trigger part. Under an appropriate model, combinatorial testing provides a) a test suite containing a significantly reduced number of input vectors *and* b) a mathematical guarantee that all possible combinations are covered by this test suite. This can increase our trust that an SUT is Trojan-free up to a specific level and with a reduced time budget for testing.

A. Combinatorial modeling of input signals

The first stage consists of test vector generation in an abstract model of circuit operation. We consider the circuit as a black box on which we can apply some inputs and can observe its output. In combinatorial testing (CT) terminology, we are interested in the t-way interactions of the n binary inputs. It is feasible to construct *covering arrays* (CAs) of *strength* t. These arrays contain n columns and can achieve 100% *configuration coverage* i.e., every possible variable-value tuple of size t appears at least in one of the rows of the array.

For the sake of clarity, we will use in the following the AES cryptographic algorithm as an example case. We will assume a circuit performing encryption and decryption using a key of 128 bits. The input model specification for the instance of the problems for Trojan detection is actually a simple case in terms of input parameter modelling [41]. In particular, the model consists of 128 binary parameters since the attacker is using the key input bits so as to activate the Trojan. Thus, we assume no input dependencies (i.e., the value of one bit be dependent on the values of other input bits).

We note that combinatorial testing can also cope with scenarios where there are input dependencies too and can avoid producing invalid test cases [42], [43]. However, the AES algorithm does not impose such dependencies for its key; this could be a case for the now outdated DES algorithm, where some input bits serve as parity checks.

Nevertheless, the key issue here is the selection of an appropriate size for k so as to guide the test suite generation. An obvious choice is to map k into t. At the moment, there is no golden rule or empirical evidence in the literature for the right value of k. Rather, the selection is more a kind of a tradeoff between detectability and availability of resources. A too small k may not be able to uncover the presence of a

Trojan that uses a larger number of inputs in its trigger logic. On the other hand, a large k may lead to long testing cycles, with unnecessary stressing of the circuits and waste of time and resources. In both cases, CT tools and algorithms can produce covering arrays with a comparatively small number of test cases (input vectors).

B. Test suite generation

A Covering Array (CA) is a mathematical object that is well-suited for hardware and software testing, since it exists for any number of columns. Optimal CAs provide minimal cardinality (i.e., minimum number of tests) with maximal coverage (i.e., all interactions of a certain size are covered at least once). Formally, a CA is defined by four positive integers: N, the number of rows or tests; t, the strength of size of interactions that are covered; n, the number of factors or variables; and v (the alphabet of the CA). A covering array is denoted by CA$(N; t, n, v)$ that is an $N \times n$ array $A = (a_{i,j})$, $0 \leq i \leq N-1$, $0 \leq j \leq n-1$, over $\mathbf{Z}_v = \{0, 1, \ldots, v-1\}$ with the property that for any t distinct columns $0 \leq c_0 < c_1 \cdots < c_{t-1} \leq n-1$ and any member $(x_0, x_1, \ldots, x_{t-1})$ of $\mathbf{Z}_v^t$, there exists at least one row r such that $x_i = a_{r,c_i}$ for all $0 \leq i \leq t-1$. For a general reference about construction methods for CAs, we refer interested readers to [44].

In the special case of $v = 2$, the CAs are denoted as *binary CAs*. In this paper, we will use binary CAs of strengths $t = 2, \ldots, 8$ with $n = 128$ factors. There are many construction methods reported in the literature; the one designed specifically for binary CAs of any strength is based in the process of juxtaposing sets of vectors with equal weight [45]. Using this procedure, the size of the required CAs is reported in Table I in the column labeled as CWV. This is also the approach used for exhaustive testing of k-bit subspaces in [22].

We derived test suites with a significantly smaller number of test vectors, as depicted in column "*ours*" of Table I. The covering array generation methods used to derive our binary CAs are as follows:

- for $t = 2$, we used the procedure defined in [46], [47] to build optimal $CA(11; 2, 128, 2)$;
- for $t = 3$, we used the procedure reported in [48] to build $CA(37; 3, 128, 2)$;
- for $t = 4$, we used the procedure reported in [49] to build $CA(112; 4, 128, 2)$;
- for $t = 5$, we used the procedure reported in [48] to build $CA(252; 5, 128, 2)$;
- for $t = 6$, we used the original idea reported in [50], implemented using the ideas of [51] to construct $CA(720; 6, 128, 2)$;
- for $t = 7$, we used the original idea reported in [50], implemented using the ideas of [51] to construct $CA(2462; 7, 128, 2)$;
- for $t = 8$, we used the original idea reported in [50], implemented using the ideas of [51] and postprocessed using the algorithm reported in [48] to construct $CA(17544; 8, 128, 2)$.

TABLE I. COMPARISON OF TEST SUITE SIZES USING THE CONSTANT WEIGHT VECTORS (CWV) PROCEDURE AND COVERING ARRAY GENERATION METHODS.

n	t	[22]	CWV [45]	ours
128	2	2^7	129	11
128	3	-	256	37
128	4	2^{13}	$8,256$	112
128	5	-	$16,256$	252
128	6	-	$349,504$	720
128	7	-	$682,752$	$2,462$
128	8	2^{23}	$11,009,376$	$17,544$

To the best of our knowledge, this is the *first time* that such large CAs are needed for combinatorial case studies, and in particular for testing with interaction strength higher than six. The rationale behind this number is provided in [52]. There, it is empirically determined that t-way testing with $t = 6$ is enough to reach 100% fault coverage in most software systems.

We emphasize that the computational difficulty of creating such arrays of minimal size should not be overlooked. This is a well-known open problem in combinatorial design literature that still requires efficient algorithms to be tackled with [53]. The detailed description of the algorithms that we developed to derive our test suites is the topic of another paper itself that is currently under preparation.

The reduction in the number of test cases we achieved is in orders of magnitude. The optimality of the test suites is an interesting question. Optimal lower bounds (mathematically or empirically) for N were defined only for small values of n and t [54], while useful theoretical upper bounds are not yet determined. Therefore, we limit the comparison of the size of our test suites with the ones provided in [22] and [45].

C. Test execution and oracle

Combinatorial testing for software defects faces the problem of test execution (i.e., the setup of an environment for translation of abstract mathematical constructs into software artifacts and for guided execution) and, more importantly, the problem of defining an oracle to decide if a defect is present or not. The case of applying combinatorial testing for hardware Trojan horse detection is much simpler. The rows of the binary CAs can be easily converted into input vectors for the hardware IP core, as it is mostly a matter of scripted conversion from one binary notation to another, without the involvement of complex grammar parsers, as for example in the case for web-based applications [40].

The hardware design process includes the production and the processing of intermediate circuit representations in different abstraction layers (HDL, netlist, gatelist, circuit layout). At each layer, different tools can be used to test a design. This is rather beneficial, especially in the case of logic testing. Even if a design manipulation cannot be seen, there are is a whole set of execution environments that can serve as oracles for the SUT, both in a software and in a hardware form. Furthermore, if a golden chip is available, it can also serve as an oracle too.

In the case of cryptographic implementations, such as the AES algorithm, there are already numerous software implementations against which hardware designers already test their implementations for correctness during development. This is a standard practice in hardware design. Thus, even

TABLE II. AES MODULE I/OS

Name	Type	Width	Description
clock	Input	1	Clock signal
reset	Input	1	Reset signal
enc_dec	Input	1	0: encrypt, 1: decrypt
key_exp	Input	1	Round key expansion
start	Input	1	Start operation
key_in	Input	128	Provided AES key
text_in	Input	128	Provided text material
busy	Output	1	AES unit is busy
key_val	Output	1	Round key is ready
text_val	Output	1	Output is ready
output	Output	128	AES output material

if the whole hardware design toolchain is not trusted, the encrypted output can be readily checked against any available software implementation of the cryptographic algorithm and detect malfunction or manipulation.

IV. CASE STUDY

The case study comprises a hardware implementation for the AES symmetric encryption algorithm. We opted for a Verilog-HDL model offered with the Sakura-G FPGA board[1]. This board is designed for research on hardware security (e.g., side-channel and fault-injection attacks) and allows for future experimentation with other forms of Trojan analysis.

The AES module can perform both encryption and decryption in ECB mode with 128-bit keys and inputs. One can reuse this module to build the more complex AES modes of operation. The module includes 7 inputs and 4 outputs of varying length, as depicted in Table II. Out the the seven inputs, the first five are for controlling the operation of the module and are not considered in the attack scenario. Conventional hardware testing should be able to detect any defects on them.

This AES module operates in two phases. In the first phase, the provided AES key (`key_in`) is expanded into the 10 round keys. In the second phase, the provided input text (`text_in`) is mixed with the round keys in order to produce the final output material (`output`). The input signal `enc_dec` controls if an encryption or a decryption operation is performed to produce the output material. Overall, the module can produce a result every 22 clock cycles (two cycles for module setup, ten cycles for completing the key expansion, and ten cycles for the transformation of the input).

We augmented this module with a malicious behaviour. The Trojan consists of two parts. The Trojan trigger part monitors the key expansion and the (internal) round number: in the tenth round, the `key_val` is set, signalling that the AES key is expanded. Once this condition is met, the Trojan trigger checks if k specific bits of `key_in` are set. All these are realized with a few additional AND gates. Once all the conditions are met, the Trojan payload part is activated. This merely consists of an XOR gate that inverses the `enc_dec` mode of operation.

Overall, the execution flow of the AES module is not changed; it's just an input signal that changes *internally* and reverses the mode of operation, resulting in getting garbage information in the `text_val` output. Neither the number of cycles is affected, nor an external signal that can be monitored.

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

TABLE III. TEST SUITE STRENGTH (t) VS. TROJAN LENGTH (k)

	Suite	Number of activations		
t	size	$k=2$	$k=4$	$k=8$
2	11	5	3	0
3	37	12	4	0
4	112	32	7	1
5	252	62	14	1
6	720	307	73	6
7	2462	615	153	10
8	17544	4246	1294	178

A. Evaluation

We held a series of experiments aiming to evaluate the capabilities of combinatorial testing for detecting the malicious functionality injected in the AES. We realized three Trojans of different lengths, monitoring $k = 2$, $k = 4$, and $k = 8$ bits of the AES key and we automated the testing procedure using the Mentor Graphic ModelSim[2] simulator for ASIC and FPGA designs.

For each of the three designs, test suites of different strengths ($t \in \{2, 3, \ldots, 8\}$) where applied. The same test suites were then applied on the Trojan-free design, which served as an oracle. Table III summarizes the executions.

It is evident that each test suite of strength t is capable of activating at least the Trojan with the respective size k and this happens numerous times. The longest the strength, the more Trojan activations of a given length are achieved. We also confirm the fundamental principle of combinatorial testing: testing with higher strength test suites will result in more activating sequences when compared to lower strength test suites.

As expected, pairwise testing ($t = 2$) fails to detect much higher-interaction Trojans (e.g., $k = 8$). Yet, even with 11 vectors only, it succeeds in the case of $k = 4$. It is interesting to note that lower strength test suites are capable of activating Trojans of higher size - the test suite with strength $t = 6$ activates multiple times the three different Trojans. This can serve as an interesting engineering trade-off between the size of test suites (and thus, the time needed for testing a circuit in a batch) and the coverage achieved for activating larger-sized Trojans (and thus, the risk taken by testing with a reduced set of vectors).

The number of test vectors used and the activations achieved is a point for further emphasis. There are $\binom{128}{8} \times 2^8$ i.e., about 366 *trillion* possible combinations for the Trojan activation; yet the whole space is covered with less than 18 *thousands* vectors and these vectors activate the Trojan *hundreds* of times.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we explored the applicability of combinatorial testing for FPGA Trojan detection. Combinatorial testing provides both the mathematical guarantees for search space coverage *and* an efficient means, in the sense of test suite size, for exciting a Trojan. We are the first to report test suites of increased strength ($t > 6$) for a large number of inputs ($n = 128$) with a specific application scenario. Our experiments confirm the superiority of combinatorial testing

[2]http://www.mentor.com/products/fv/modelsim/

in the number of tests performed and the capability to excite a Trojan.

We envision four directions for future extensions of this work. The first relates to the efficient generation of test suites with appropriate strengths. While covering arrays offer quite optimal suite size, it remains a hard problem how to generate them for arbitarily (n, k) tuples. Other classes of arrays such orthogonal arrays could offer a viable alternative, should their size remains within the time envelope of the interested party. However, since they do not exist for all number of factors up to 128 limits their potential use.

The second direction relates to the study of s-Trojans. A repeated application of the test suites for combination Trojans can be reused for this case. However, it remains an interesting problem if and how the test suite size for detecting an s-Trojan with m internal states can be reduced.

The third direction of research relates to experimenting with the intermix of combinatorial testing with side-channel analysis for Trojan detection. While the principle remains the same, the testing oracle can become quite fuzzy. How this incomplete information can be used to guide subsequent testing rounds remains an open problem.

Finally, the fourth direction relates to developing capabilities not only to detect the presence (existence) of a Trojan but also to provide guidance for identifying its location. While the exact location cannot be identified without knowledge of the circuit's internals, the information of which of all the input signals lead to Trojan payload activation can reduce significantly the debugging time. A possible workaround would be to employ Fault Location Analysis (FLA) where identification of inducing combinations and faulty statements is considered. The first component does not require any knowledge of the circuit's internals and thus makes it an interesting approach to consider in the near future for identifying the bit location of Trojan's activating sequences in FPGA cryptographic designs.

Acknowledgments

This work was partially supported by the COMET K1 program by the Austrian Research Funding Agency (FFG), the EU COST Actions IC1204 TRUDEVICE and IC1403 CRYPTACUS, the GSRT Action "KRIPIS" with national (Greece) and EU funds in the context of the research project "ISRTDI", and the projects 238469-CONACyT Métodos Exactos para Construir Covering Arrays Óptimos; 232987-CONACyT Conjuntos de Prueba Óptimos para Métodos Combinatorios (Optimal Test Sets for Combinatorial Methods).

References

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

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

[3] W. Lee and B. Rotoloni, "Emerging cyber threats report 2013," Georgia Techn Cyber Security Summit 2012, pp. 4–5, 2012.

[4] R. Boscovich, "Microsoft disrupts the emerging nitol botnet being spread through an unsecure supply chain," Available at http://blogs.microsoft.com/blog/2012/09/13/microsoft-disrupts-the-emerging-nitol-botnet-being-spread-through-an-unsecure-supply-chain/, 2012.

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

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

[7] P. Kitsos and A. Voyiatzis, "FPGA Trojan detection using length-optimized ring oscillators," in *17th EUROMICRO Conference on Digital System Design (DSD 2014)*. IEEE CPS, 2014, verona, Italy, August 27–29, 2014.

[8] ——, "Towards a hardware Trojan detection methodology," in *2nd EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems (ECYPS 2014)*, 2014, budva, Montenegro, June 15–19, 2014.

[9] Y. Jin and Y. Makris, "Hardware Trojans in wireless cryptographic ICs," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 26–35, 2010.

[10] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "FPGA Trojans through detecting and weakening of cryptographic primitives," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[11] J. Shield, B. Hopkins, M. Beaumont, and C. North, "Hardware Trojans - a systemic threat," in *Australasian Information Security Conference (ACSW-AISC) 2015*, ser. Conference in Research and Practice in Information Technology. Australian Computer Society, 2015, pp. 45–51.

[12] X.-T. Ngo, I. Exurville, S. Bhasin, J.-L. Danger, S. Guilley, Z. Najm, J.-B. Rigaud, and B. Robisson, "Hardware Trojan detection by delay and electromagnetic measurements," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. EDA Consortium, 2015, pp. 782–787.

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

[14] S. Bowles and J. Hernandez-Castro, "The first 10 years of the Trojan horse defence," *Computer Fraud & Security*, vol. 2015, no. 1, pp. 5–13, 2015.

[15] A. Nayak, K. Yen, and J. Fan, "Breaching of ring oscillator based Trojan detection and prevention in physical layer," *International Journal on Recent Trends in Engineering and Technology*, vol. 10, no. 1, 2014.

[16] J. Francq and F. Frick, "Introduction to hardware Trojan detection methods," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 770–775.

[17] G. Di Natale, S. Dupuis, and B. Rouzeyre, "Is side-channel analysis really reliable for detecting hardware Trojans?" in *DCIS'2012: XVII Conference on Design of Circuits and Integrated Systems*, 2012, pp. 238–242.

[18] A. Ferraiuolo, X. Zhang, and M. Tehranipoor, "Experimental analysis of a ring oscillator network for hardware Trojan detection in a 90nm ASIC," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '12. New York, NY, USA: ACM, 2012, pp. 37–42.

[19] N. Karimian, F. Tehranipoor, M. Rahman, and D. Forte, "Genetic algorithm for hardware Trojan detection with ring oscillator network (RON)," in *IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 2015, p. to appear.

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

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

[22] N. Lesperance, S. Kulkarni, and K.-T. T. Cheng, "Hardware Trojan detection using exhaustive testing of k-bit subspaces," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, p. to appear, tokyo, Japan, January 19–22, 2015.

[23] S. Dupuis, P.-S. Ba, M.-L. Flottes, G. Di Natale, and B. Rouzeyre,

"New testing procedure for finding insertion sites of stealthy hardware Trojans," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. EDA Consortium, 2015, pp. 776–781.

[24] J. Rajendran, O. Sinanoglu, and R. Karri, "Regaining trust in VLSI design: Design-for-trust techniques," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1266–1282, 2014.

[25] R. S. Chakraborty, S. Paul, and S. Bhunia, "On-demand transparency for improving hardware Trojan detectability," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 48–50.

[26] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 113–116.

[27] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware Trojans," in *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*. IEEE, 2014, pp. 49–54.

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

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

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

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

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

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

[34] R. Mandl, "Orthogonal Latin squares: An application of experiment design to compiler testing," *Commun. ACM*, vol. 28, no. 10, pp. 1054–1058, Oct. 1985.

[35] P. Ammann and J. Offutt, "Using formal methods to derive test frames in category-partition testing," in *Computer Assurance, 1994. COMPASS '94 Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security. Proceedings of the Ninth Annual Conference on*, Jun 1994, pp. 69–79.

[36] Y. Malaiya, "Antirandom testing: getting the most out of black-box testing," in *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, Oct 1995, pp. 86–95.

[37] D. Kuhn, R. Kacker, and Y. Lei, "Practical combinatorial testing," NIST Special Publication 800-142, 2010.

[38] J. D. Hagar, T. L. Wissink, D. Kuhn, and R. N. Kacker, "Introducing combinatorial testing in a large organization," *Computer*, vol. 48, no. 4, pp. 64–72, Apr 2015.

[39] B. Garn and D. Simos, "Eris: A tool for combinatorial testing of the Linux system call interface," in *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on*. IEEE, 2014, pp. 58–67.

[40] B. Garn, I. Kapsalis, D. Simos, and S. Winkler, "On the applicability of combinatorial testing to web application security testing: a case study," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*. ACM, 2014, pp. 16–21.

[41] M. Grindal and J. Offutt, "Input parameter modeling for combination strategies," in *Software Engineering*, Innsbruck, Austria, Feb 2007.

[42] J. Petke, S. Yoo, M. B. Cohen, and M. Harman, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in *Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, 2013, pp. 26–36.

[43] J. Bozic, B. Garn, D. E. Simos, and F. Wotawa, "Evaluation of the IPO-family algorithms for test case generation in web security testing," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, April 2015, pp. 1–10.

[44] J. Torres-Jimenez and I. Izquierdo-Marquez, "Survey of covering arrays," in *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2013)*. IEEE CPS, 2013.

[45] D. T. Tang and L. S. Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Transactions on Computers*, vol. c-32, no. 12, pp. 1145–1150, 1983.

[46] G. O. H. Katona, "Two applications (for search theory and truth functions) of Sperner type theorems," *Periodica Mathematica Hungarica*, vol. 3, no. 1–2, pp. 19–26, 1973.

[47] D. J. Kleitman and J. Spencer, "Families of k-independent sets," *Discrete Mathematics*, vol. 6, no. 3, pp. 255–262, 1973.

[48] J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," *Information Sciences*, vol. 185, no. 1, pp. 137–152, 2010.

[49] C. J. Colbourn and J. Torres-Jimenez, "Heterogeneous hash families and covering arrays," in *Error-correcting Codes, Finite Geometries and Cryptography (Contemporary Mathematics) Editors Aiden A. Bruen, and David L. Wehlau*. American Mathematical Society, 2010, pp. 3–16.

[50] C. J. Colbourn, "Covering arrays from cyclotomy," *Design Codes and Crytography*, vol. 55, no. 2–3, pp. 201–219, 2010.

[51] J. Torres-Jimenez, N. Rangel-Valdez, A. L. Gonzalez-Hernandez, and H. Avila-George, "Construction of logarithm tables for Galois fields," *International Journal of Mathematical Education in Science and Technology*, vol. 42, no. 1, pp. 91–102, 2011.

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

[53] A. Hartman and L. Raskin, "Problems and algorithms for covering arrays," *Discrete Mathematics*, vol. 284, no. 13, pp. 149–156, 2004.

[54] C. J. Colbourn, "Covering arrays," in *Handbook of Combinatorial Designs*, 2nd ed., ser. Discrete Mathematics and Its Applications, C. J. Colbourn and J. H. Dinitz, Eds. Boca Raton, Fla.: CRC Press, 2006, pp. 361–365.