IC Protection Against JTAG-Based Attacks

Xuanle Ren, *Student Member, IEEE*, Francisco Pimentel Torres, *Student Member, IEEE*, R. D. Blanton, *Fellow, IEEE*, and Vítor Grade Tavares, *Member, IEEE*

***Abstract*—Security is now becoming a well-established challenge for integrated circuits (ICs). Various types of IC attacks have been reported, including reverse engineering IPs, dumping on-chip data, and controlling/modifying IC operation. IEEE 1149.1, commonly known as Joint Test Action Group (JTAG), is a standard for providing test access to an IC. JTAG is primarily used for IC manufacturing test, but also for in-field debugging and failure analysis since it gives access to internal subsystems of the IC. Because the JTAG needs to be left intact and operational after fabrication, it inevitably provides a "backdoor" that can be exploited outside its intended use. This paper proposes machine learning-based approaches to detect illegitimate use of the JTAG. Specifically, JTAG operation is characterized using various features that are then classified as either legitimate or attack. Experiments using the OpenSPARC T2 platform demonstrate that the proposed approaches can classify legitimate JTAG operation and known attacks with significantly high accuracy. Experiments also demonstrate that unknown and disguised attacks can be detected with high accuracy as well (99% and 94%, respectively).**

***Index Terms*—Hardware security, Joint Test Action Group (JTAG), machine learning, reverse engineering.**

I. Introduction

Hardware security is becoming a significant challenge in the design and fabrication of integrated circuits (ICs). Several integrated systems, including a field-programmable gate array (FPGA) for military use, have been attacked successfully [1]–[3]. The adversaries, including clever outsiders, knowledgeable insiders, and funded organizations, may employ different attack methods for a variety of purposes [4]. An outsider may control inputs and observe outputs of an IC in order to reverse engineer the design noninvasively. A knowledgeable insider may observe side-channel signals (via the test interface [5]–[8], perform power or magnetic analyses [9]–[11]) and disable/modify the communication within the IC [12]. A funded organization may even influence the IC fabrication process, invasively reverse engineer the IC, and inject Trojans [13].

Among the attack methods, scan-based attacks are performed through the standard test access port (TAP) and the boundary-scan architecture, which is defined by the IEEE 1149.1 standard [or named Joint Test Action Group (JTAG)] [14]. Although some attackers have greater levels of access, it is reasonable to assume that the JTAG is a preferred conduit for improperly accessing the IC, mainly because of two reasons. First, the JTAG is a widely used interface for manufacturing testing and in-field debugging of modern ICs. Second, the JTAG typically provides powerful features for accessing on-chip data and circuitry. The OpenSPARC T2 [15], for example, has more than ten debugging functions within the JTAG, such as L2 cache r/w, memory built-in self-test (MBIST), and direct memory access.

A. Motivation

The controllability and observability provided by undocumented (private) JTAG functions inevitably undermine the security of ICs. On one hand, scan chains can be used as a side channel to dump on-chip data. One example involves cryptographic keys [5]–[8], [16]–[20]. Specifically, hardware encryption primitives use a block cipher to encrypt a plain text, but the intermediate results, usually located in scan chains, can be shifted out, through which the encryption key can be derived. On the other hand, various private JTAG functions used for in-field debugging, diagnosis and firmware upgrade are also vulnerable to scan-based attacks. Such weakness has allowed attackers to successfully derive data from on-chip memory [1], update firmware [2], control chip operation [3], and uncover the JTAG architecture [21].

In this paper, three detectors are proposed for identifying JTAG attacks through monitoring the JTAG port. Two of them employ machine learning algorithms for classification,[1]

Manuscript received September 4, 2017; revised December 6, 2017; accepted January 10, 2018. Date of publication February 5, 2018; date of current version December 19, 2018. This work was supported in part by the Project "NanoSTIMA: Macro-to-Nano Human Sensing: Towards Integrated Multimodal Health Monitoring and Analytics/NORTE-01-0145-FEDER-000016", which is financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF). This work was also supported in part by the Portuguese Foundation for Science and Technology (FCT), under Scholarship SFRH/BD/52166/2013, through the Carnegie Mellon | Portugal Program. The authors would also like to thank Prof. João Canas Ferreira for the help with the FPGA. This work was presented in part at the Design, Automation and Test in Europe in Mar. 2015 and the Computer Society Annual Symposium on VLSI in Jul. 2016. This paper was recommended by Associate Editor X. Wen. *(Corresponding author: Xuanle Ren.)*

X. Ren is with the Advanced Chip Test Laboratory of the Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA, also with the Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal, and also with INESC TEC, 4200-465 Porto, Portugal (e-mail: xuanler@ece.cmu.edu).

F. P. Torres is with Chip Design Team, Qualcomm, Cambridge CB4 0WZ, U.K.

R. D. Blanton is with the Advanced Chip Test Laboratory of the Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

V. G. Tavares is with the Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal, and also with INESC TEC, Porto, Portugal.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2802866

[1]In a binary classification problem, given a set of samples, each belonging to one of two classes, a classifier based on these samples (called training) is constructed and later used for predicting which class a new sample belongs.

and the third involves an anomaly detection approach based on representative JTAG operations. All detectors rely on the assumption that an attacker only has access to the JTAG port and is, at least initially, unaware of all private JTAG functions. The first detector characterizes the JTAG operation using a set of features derived from the test port inputs; the features are then fed to a trained forest of decision trees [22] that categorizes the operation as either legitimate or an attack. The second detector uses a support vector machine (SVM) [23] to examine the JTAG operation over time; this detector is capable of detecting attacks not used in the training of the SVM. It is worth noting that the capability of detecting new attacks is difficult to verify comprehensively since different types of attacks are always arising. The experiments treat an existing attack as unknown by excluding it from classifier training, thus providing a preliminary evaluation for the capability of detecting unknown JTAG attacks. The third detector attempts to categorize JTAG attacks based on representative JTAG sequences. Different from traditional anomaly detection using manually designed state machines, the representative sequences can be derived automatically. Besides, the probabilistic nature of the derivation incurs fewer false alerts than state machines. An IC designer is recommended to employ one of them based on the tradeoff of overhead and performance.

In addition to the detectors, a JTAG protection approach is also proposed. Specifically, upon detection of an attack, access to data registers (DRs) through the JTAG is modified to disable controllability and observability provided by the JTAG.

B. Prior Work

To protect the JTAG from attacks, various techniques have been proposed [8], [24]–[29]. One technique involves disabling the JTAG through antifuse after manufacturing test; however, this also disables in-field testing and debugging [30]. Access to the JTAG can also be obfuscated by distributing its ports across the chip (or board), but the ports can still be found using exhaustive search [2]. Third, on-chip compression/compaction, originally aimed at expediting manufacturing test, also protects on-chip data from being directly observed through obfuscating test responses [25]. However, it only protects the scan chains rather than private JTAG functions [8]. Another technique involves password-based authentication that requires a correct password to gain access to the JTAG [26]–[28]. Nevertheless, the plain-text password might be eavesdropped; this shortcoming is exacerbated if all fabricated instances share the same password. The risk of plain-text passwords can be mitigated using more complex encryption algorithms, such as DES [5], AES [16], RSA [7], and ECC [17]. Nonetheless, all of these encryption algorithms require a trusted server to manage the multistage authentication between the user and the device, which relies on the availability of a network.

Most of the aforementioned approaches mainly target scan chains, while protecting private JTAG functions has gained little attention so far. Further, because modern ICs are implemented using more and more debugging functions, the security of private JTAG functions is also becoming more important.

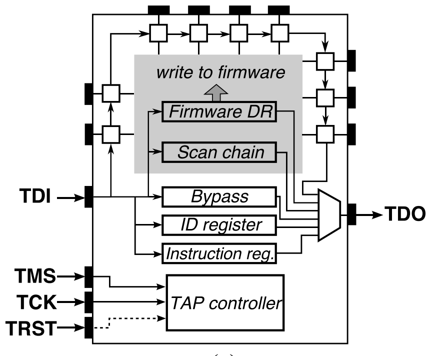
Concurrently, intrusion detection based on JTAG input analysis is an approach that can add complementary security to encryption techniques [31]–[35]. One example involves signature detection that characterizes known JTAG misuses using a database of signatures. An observed JTAG operation that matches any signature is labeled as an attack. However, the effectiveness of a signature detector strongly relies on the comprehensiveness of the signature database. To overcome this limitation, anomaly detection creates models of legitimate JTAG operation and identifies operations that do not conform. An anomaly detector can be achieved using either a state machine [32] or a Markov chain [33]. The former uses strict control flows for modeling legitimate JTAG operation and makes a binary decision about whether a given JTAG operation is legitimate, while the latter makes a probabilistic decision. However, for both models, variances of legitimate JTAG operation can incur significant inaccuracy.

In addition to detecting attacks, various architectures have also been proposed to secure the IC after an attack is detected. One technique involves locking the JTAG and restricting access [27]–[29], [36], [37]. JTAG access is permitted after verification via the authentication infrastructure. A locked JTAG behaves like a bypass register or a disconnection between the input and the output of the test interface. However, this approach suffers from the same risks as the password-based authentication. Another technique is to scramble the data at the scan output in an unpredictable order [36], [38]–[40]. The key idea is to divide a register into multiple subregisters and scramble the order in how they appear at the output. Output scrambling is deactivated if authentication is successful. Nevertheless, register data, even though scrambled, is still observable.

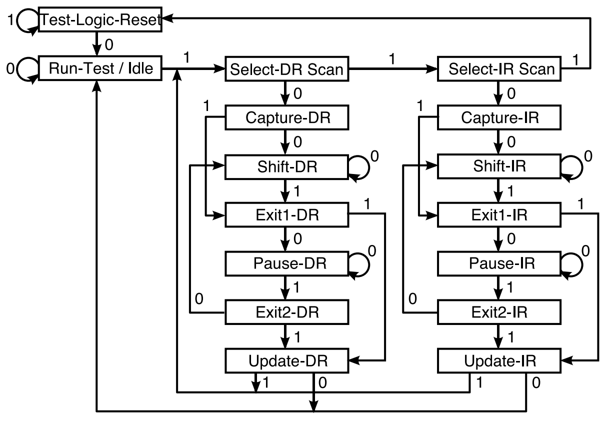
The learning-based detectors proposed here have several potential benefits compared to prior work. First, the detection is stand-alone and does not rely on an encryption scheme, thus avoiding problems related with password management and leakage. Second, learning-based detectors not only protect scan chains but also the private JTAG functions that are becoming increasingly more complex. Third, using both legitimate JTAG operations and attacks for classifier training is an improvement over an incomplete signature database, and also incurs fewer false positives than anomaly detectors. Finally, the proposed secure JTAG architecture differs from data scramble because an attacker can neither observe the actual register data nor be notified whether the JTAG enters into a protection mode.

II. Reverse Engineering Through JTAG

As shown in Fig. 1(a), the JTAG TAP consists of four (or five) pins, including the test data input (TDI), the test data output (TDO), the test mode select (TMS), the test clock, and the optional test reset (TRST). A boundary scan chain, usually located at the periphery of a chip not only allows test stimuli to be supplied serially to the chip pins via the TDI but also allows the test response to be observed serially via the TDO. In addition to the boundary scan chain, other DRs, some through private JTAG functions, can also be accessed



(a)



(b)

Fig. 1. (a) Block diagram of a typical JTAG architecture. (b) JTAG TAP controller is a one-input FSM.

using the TDI and TDO pins. Fig. 1(a) shows two example DRs, one for updating the firmware and the other for capturing/updating scan values. Access to DRs is operated by the JTAG TAP controller that implements the state diagram shown in Fig. 1(b). The JTAG TAP controller is a finite state machine (FSM) controlled by a single input, the TMS. Every DR has one or more affiliated instructions that provide access, configuration, or control. A user can select a DR by loading the appropriate instruction opcode into the instruction register (IR) via the TDI. A selected DR can be operated using three fundamental register operations, namely capture, shift, and update. For example, for the OpenSPARC T2, selecting the boundary-scan chain initially requires an opcode 00 (in hexadecimal format). The user then loads the input stimuli via the TDI by repeatedly entering the shift-DR state. Finally, the update-DR state applies the stimuli as input to the chip logic. After the operation of the chip logic is completed, the response is captured by the scan chain within the capture-DR state, which can then be shifted out via the TDO by repeatedly entering the shift-DR state. Another example involves the MBIST operation. Different from the boundary scan, the MBIST operation of the OpenSPARC T2 is executed using a sequence of instructions with a predefined order, that is, typically, selecting the on-chip modules to be tested (instruction MBIST_BYPASS), setting the MBIST mode (MBIST_MODE), initiating the MBIST process (MBIST_START), and checking the result (MBIST_RESULT).

TABLE I
JTAG Attack Strategies Are Organized As Three Main Steps and Nine Substeps

Steps	Sub-steps	
Locate test ports	S_1	Identify the location of the JTAG ports on chip/board
Find DR property	S_2	Identify the length of the IR
	S_3	Identify the length of each DR
	S_4	Find the opcodes that do not correspond to any DR
	S_5	Check if each DR can be captured/updated
	S_6	Identify internal scan chains
Interrogate private functions	S_7	Separate opcodes into bundles based on their associated functions
	S_8	Determine the nature (control vs. data) of each DR
	S_9	Investigate adjacent instruction opcodes for characterizing any interactions between them

A JTAG attack normally begins with reverse engineering the private JTAG functions. Table I lists typical attack strategies that can be categorized in three main steps and nine substeps [1], [3], [21], [41]. The first step is to identify the JTAG ports physically. This step is relatively easy because the test ports are usually clustered together someplace on the chip. The second step is to explore the property of the IR and the DRs. The length of the IR and the DRs can be easily extracted by shifting in a sufficiently long sequence of ones followed by a single zero. The opcodes that are not associated with any DR usually behave as a one-bit bypass register or a disconnection. These opcodes are important because they likely lie between valid opcodes corresponding to different JTAG functions. For example, in the OpenSPARC T2, the instructions for control-register configuration and MBIST have opcodes that are separated by unused opcodes 11 and 12 (in hexadecimal format). Besides checking the length of a DR, it is also beneficial to learn if a DR can be captured and updated. An attacker may dump on-chip data from a DR that can be captured, and write data into the chip through a DR that can be updated. Further, if a DR can be updated and another DR with neighboring opcode can be captured, then an attacker can make a reasonable guess that these two DRs are used for reading some memory. In other words, the first DR sends an address and the second DR captures the corresponding data. By similar means, an attacker can uncover additional JTAG functions by investigating the interactions between adjacent instruction opcodes.

III. Detectors for Identifying JTAG Attacks

In this section, three detectors are described for identifying JTAG attacks. The first and the second are based on a random forest and an SVM, respectively, and the third employs an anomaly detection approach based on representative JTAG operations.

A. Random Forest

The first detector employs a random-forest model [22] involving two phases, namely offline learning and online prediction, as shown in Fig. 2. For the offline learning, an ensemble of decision-tree classifiers are trained using features extracted from already-collected JTAG operations. For

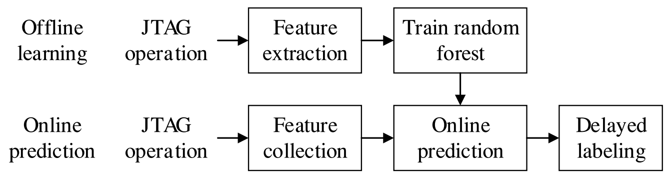


Fig. 2. Random forest detector consists of two phases: offline learning and online prediction.

the online prediction, the features are monitored in real-time and classified using the trained classifier.

1) Features: A set of specifically defined features (named FEATURE-SET-1) as shown in Table II are used to characterize JTAG operation in two aspects: 1) intrainstruction statistics and 2) interinstruction transition.

Intrainstruction statistics (F_1–F_7) describe the patterns executing a single JTAG instruction. For example, once a DR is selected, the number of shifting cycles (F_3) should be equal to the length of the DR for either a read or write operation. However, the DR may be shifted for more or fewer cycles if the attacker does not know the length of the DR. The second pattern involves the number of TMS transitions (F_6), either from zero to one or from one to zero. The TMS transition is relevant in two circumstances: 1) reset-type instructions require fewer TMS transitions because they do not select any DR and 2) entering the pause-DR state, which is necessary if shifting a DR needs to be temporarily suspended, leads to more TMS transitions. Another relevant pattern is whether the opcode loaded into the IR corresponds to a valid JTAG function (F_7). Although an r-bit IR can have at most 2^r valid opcodes, typically only a subset of all opcodes correspond to valid JTAG functions while others remain unused.

Interinstruction transition, named TYPC_TRANSIT (F_8), indicates if the transition from one instruction to the next is typical. Although there are no strict rules for JTAG instruction transitions, some transitions are regarded typical if they appear with high frequency during legitimate use of the JTAG. These typical transitions are stored in a look-up table (LUT) that is later used for online prediction.

2) Random Forest Classification: A decision tree is a supervised learning model[2] used for classification. Each node of a decision tree represents a "test" on a feature, each branch represents the outcome of the test, and each leaf node represents a class label. A classification starts from the root node, and then descends to branches until it reaches a leaf node. A random forest is an ensemble of trees and its prediction is based on majority voting of labels provided by individual trees. Each tree in the ensemble is trained using a random subset of the data (called bootstrap aggregating or bagging) [22]. Bagging is an ensemble technique that can improve accuracy and avoid overfitting.

JTAG operation is monitored by collecting the eight features per instruction and supplying them to an on-chip random-forest classifier at the clock cycle when the IR is updated with an instruction. Each tree in the forest provides a prediction that is then aggregated for the final prediction.

[2]Supervised learning is a machine learning task, where each training example consists of an input vector and an output label.

TABLE II
JTAG OPERATION IS CHARACTERIZED USING EIGHT FEATURES (FEATURE-SET-1)

No.	Feature	Description
F_1	INSTR_MSB	Higher $\lfloor r/2 \rfloor$ bits of the opcode (assume r-bit IR)
F_2	INSTR_LSB	Lower $\lceil r/2 \rceil$ bits of the opcode (assume r-bit IR)
F_3	CYC_SHIFT_DR	No. of clock cycles within the shift-DR state
F_4	CYC_RTI	No. of clock cycles within the run-test/idle state
F_5	CYC_TLR	No. of clock cycles within the test-logic-reset state
F_6	TMS_TRANSIT	No. of TMS transitions
F_7	LEGAL_OPCODE	Opcode corresponds to a valid JTAG function?
F_8	TYPC_TRANSIT	Transition from one instruction to the next is typical?

3) Delayed Labeling: As just described, the random forest makes per-instruction predictions. However, the decision of whether the JTAG is being operated legitimately is not made until sufficient evidence is collected. Delayed labeling of the JTAG operation is prudent because there is likely some natural variance in legitimate JTAG operation. Specifically, only when p predictions within a period of q instructions ($p \leqslant q$) indicate the presence of an attacker, will the JTAG be placed into a protection mode (details in Section IV). Values for both p and q are determined empirically from experiments.

Although mitigated by delayed labeling, false positives and false negatives may still occur. More details about handling false positives will be discussed in Section VII.

B. SVM Detector

The SVM detector uses a different set of features, named FEATURE-SET-2, which consists of a sequence of JTAG instruction opcodes, emphasizing more the sequential characteristics of JTAG operation. The single sequential feature in FEATURE-SET-1 (i.e., TYPC_TRANSIT) captures the pairwise transition of JTAG instructions that sometimes falls short of comprehensively capturing JTAG operation. For example, the MBIST operation described in Section II requires a sequence of at least four instructions, revealing that a sequence of n opcodes ($n > 2$) can better characterize JTAG operation.

Before the SVM classification, a fundamental check of the JTAG operation is performed to detect the use of an illegal opcode. Illegal opcode (corresponding to F_7 in Table II) refers to the bits loaded into the IR that do not correspond to any valid JTAG function. This two-tiered approach is justified since an attacker with minimal knowledge of the JTAG functions will make basic mistakes attempting to operate the JTAG.

If inputs to the JTAG TAP controller pass the fundamental check, then overlapping, sequential instructions of length n are supplied to an SVM classifier for identification of JTAG misuses. The length of the opcode sequence, n, is determined empirically from experiments.

Similar to the random forest, the SVM detection also consists of offline learning and online prediction. For the offline learning, the opcode sequences extracted from JTAG operation are used for training an SVM classifier. For the online prediction, the JTAG opcode sequences are collected in real-time in an overlapping manner using a fixed-size sliding window,

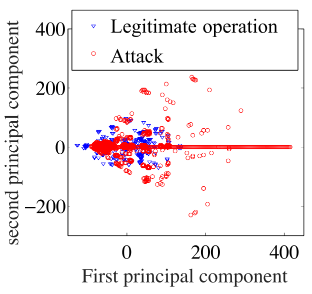


Fig. 3. Opcode sequences ($n = 4$) extracted from the OpenSPARC T2 are visualized using PCA.

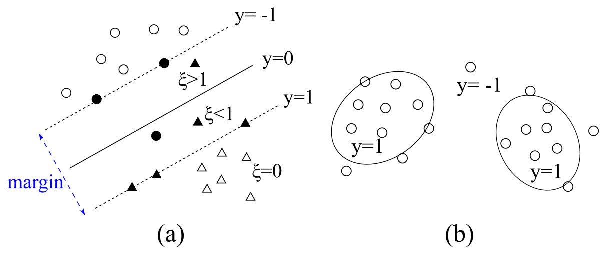


Fig. 4. (a) SVM model separates two classes of samples by finding a decision boundary and maximizing the margin. (b) One-class SVM is trained based solely on normal samples and finds regions, where samples form high-density clusters.

and classified by the SVM classifier. The prediction is based on the opcodes of both the current instruction and the prior $(n-1)$ instructions. In addition, the delayed labeling described in Section III-A3 is also used.

Opcode sequences of length $n = 4$ are visualized using principal component analysis (PCA). Specifically, the opcode sequences, which are 4-D data samples, are "rescaled" based on a set of linearly uncorrelated variables (called principal components). Fig. 3 shows the rescaled opcode sequences in terms of the first (x-axis) and the second (y-axis) principal components, respectively. It demonstrates that attacks and legitimate operations not only share a nonlinear boundary but also have overlapping regions.

The SVM detection starts with building a training set from JTAG opcode sequences, each of which is labeled as either legitimate or attack. The training of an SVM classifier involves finding a decision boundary that separates the two classes of samples such that the smallest distance between the decision boundary and any of the samples is maximized as Fig. 4(a) illustrates. The decision boundary is modeled as

$$y(x) = w^T\phi(x) + b \tag{1}$$

where x is a sample, $y(x)$ is the prediction of x, and $\phi(x)$ represents some feature-space transformation. The decision boundary, characterized by w and b, is determined by solving

$$\begin{aligned}&\underset{w,\xi}{\operatorname{argmin}} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i\\ &\text{s.t.} \ \ \left\|t_i\left(w^T\phi(x_i)\right) + b\right\| \geq 1 - \xi_i, \quad i = 1, \ldots, N\end{aligned} \tag{2}$$

where x_i is the i-th training sample, t_i is the true label (either 1 or -1) of x_i, ξ_i is a slack variable that allows samples to

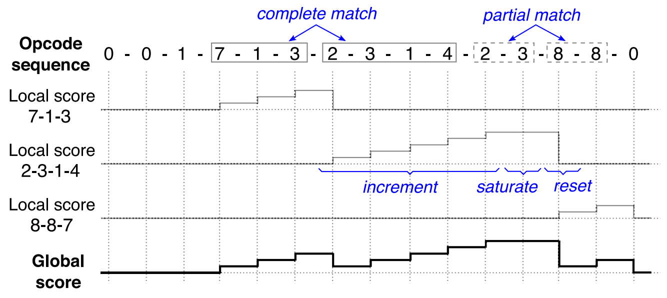


Fig. 5. Opcode sequence collected in real-time is compared with three representative sequences, namely 7-1-3, 2-3-1-4, and 8-8-7, from the first opcode to the last. Each representative sequence is associated with a local score that may increment, saturate (saturation score $s = 5$ in this example) or reset, according to the matching result. A global score, equal to the maximum local score, serves as an indicator of the security status of the JTAG.

be misclassified with a penalty, and $C > 0$ controls the trade-off between the slack variable penalty and the margin. This optimization problem is convex so the solution can be derived from its dual representation. More details concerning solving a convex problem can be found in [42]. The decision boundary is formulated as

$$y(x) = \operatorname{sgn}\left(\sum_{i\in S} a_i t_i k(x, x_i) + b\right) \tag{3}$$

and

$$b = \frac{1}{N_S}\sum_{i\in S}\left(t_i - \sum_{j\in S} a_j t_j k\left(x_i, x_j\right)\right) \tag{4}$$

where S denotes the set of support vectors (SVs), N_S represents the number of SVs, and $k(x, x') = \phi(x)\phi(x')$ denotes the kernel function in terms of x and x'. In this paper, the radial basis function (RBF) is used as the kernel function.

To detect unknown attacks, a one-class SVM is also evaluated because it is commonly used for novelty detection [43]. The goal behind the one-class SVM is then to capture regions in the input space where samples form high-density clusters, as shown in Fig. 4(b).

C. Representative-Based Anomaly Detector

The prior detectors are trained using both legitimate JTAG operations and attacks. However, collecting all types of attacks is not a trivial task because new attacks are likely arising. We propose an anomaly detector that uses a set of typical JTAG sequences as representatives of legitimate JTAG operation, and any deviating operation will be regarded as an attack.

1) Derivation of Representative Sequences: A set of instruction sequences are derived to represent legitimate JTAG operation. Initially, all instruction sequences with length three to six are derived from already-collected legitimate JTAG operations (analysis of legitimate uses of the JTAG reveals that most operations for the OpenSPARC T2 contain three to six instructions). The representation sequences are derived as follows. First, the frequency of appearance for each sequence is counted, with sequences of frequency $<t$ discarded. Then, redundant sequences are identified through checking if one

TABLE III
FOR EACH TYPE OF DR, CONTROLLABILITY AND OBSERVABILITY ARE PROHIBITED THROUGH SPECIFIC ACTIONS

Type of DR	Disable controllability	Disable observability
Capture-only	N/A	Read from decoy shadow register
Update-only	Disable update operation	N/A
Capture/update	Only write to decoy shadow register	Read from decoy shadow register
Scan-accessible	Disable shift operation	Shift out LFSR for cycles equal to DR length, then shift out decoy DR data

sequence is contained in another with similar frequency. For such cases, the shorter subsumed sequence is discarded. Finally, redundancy is further reduced through greedy search, meaning that, the sequence with the least impact on the detection accuracy is discarded in each iteration. The removal proceeds until the detection accuracy shows a significant drop.

2) Attack Detection: The security status of the JTAG is measured using a score that indicates whether the JTAG operation corresponds to any representative sequence. Specifically, each representative sequence is associated with a score that measures its matching degree with the JTAG operation, with the maximum score selected as a global one. Fig. 5 illustrates how the global score is calculated. The score associated with each representative sequence (named a local score) is initialized to zero when the chip is powered on. Then opcode sequences are collected in real-time and compared with each representative sequence from the first opcode to the last. Every time an opcode match is observed, the local score increments (or remains the same if it has saturated); otherwise, it is reset to zero. The saturation value for a local score, s, is predefined and determined empirically from experiments.

Due to the variance that naturally occurs in legitimate JTAG operation, delayed labeling described in Section III-A3 is employed. Specifically, operation of the JTAG is labeled as an attack only if the global score stays saturated for no fewer than p instructions out of q consecutive instructions.

IV. JTAG PROTECTION

Upon detection of an attack, the JTAG needs to be protected, to prevent further attacks. However, if the attacker knows which operation triggers the protection, then he/she may avoid that operation in future attacks (to other chips). Thus, that particular triggering operation needs to be obfuscated. In order to achieve this objective, access to DRs is modified as shown in Fig. 6. It prevents the chip from being controlled/modified, and also prevents on-chip data from being observed.

Fig. 6 shows four types of DRs, i.e., capture-only DR, update-only DR, capture/update DR, and scan-accessible DR. A capture-only DR is typically connected to a shadow register and data is read in parallel from the shadow register to the scan register within the capture-DR state. An update-only DR allows its data to be updated from the scan register to the shadow register within the update-DR state. A capture/update DR allows data transferring in both directions. A scan-accessible DR is not connected to any shadow register, so the operations of capture and update are not needed.

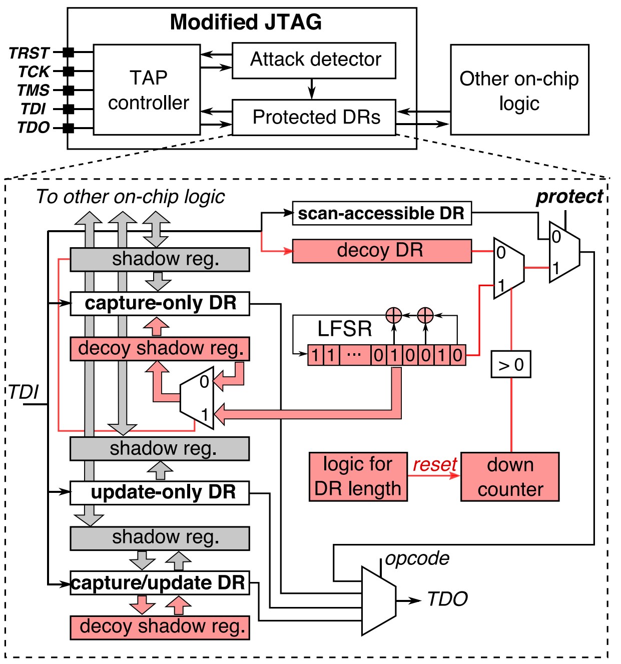


Fig. 6. To protect the JTAG, access to DRs is modified as shown in red.

Table III lists how controllability and observability are prohibited for each type of DR. For a capture-only DR, a decoy shadow register and a linear-feedback shift register (LFSR) are employed to disable observability. More precisely, upon detection of an attack, connection to the actual shadow register is disabled; instead, the DR captures data from the decoy shadow register. Further, the data in this decoy shadow register is provided by the LFSR. An LFSR is commonly used as a pseudo-random number generator for stream ciphers due to its low hardware overhead, long period, and uniformly distributed output stream [44]. For example, the period of a 32-bit LFSR can be over 10^9. As shown in Fig. 6, the hardware of an LFSR is a shift register whose leftmost bit is driven by the XOR of some bits of the shift register. The data in the decoy shadow register is updated by the LFSR only when the data in the actual shadow register is updated. The attacker may find that the data supplied via the TDO is random, but it is still hard to tell when the data became random because the attacker does not know what to expect. Thus, the LFSR is effective in preventing the attacker from knowing which operation triggers the protection. For an update-only DR, controllability is disabled through disabling the update operation. For a capture/update DR, a decoy shadow register is employed. Upon detection of an attack, connection to the actual shadow register is disabled; instead, the decoy register responds to the operations of capture and update. Because the decoy shadow register is stand-alone, it cannot control or observe any on-chip data. For a scan-accessible DR, because data should neither be shifted in nor out, the shift operation is disabled. A decoy DR and the LFSR are used instead to support the shift operation and supply data to the TDO. The decoy DR is a shift register that imitates the shift operation of the normal DRs. Once the JTAG enters the protection mode, the decoy DR and the LFSR,

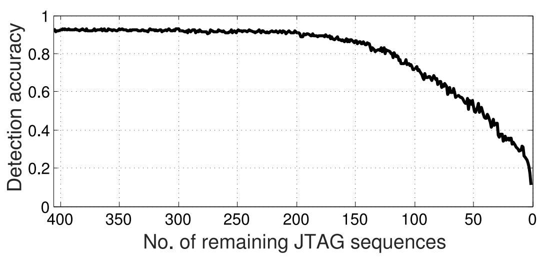


Fig. 7. Detection accuracy is evaluated iteratively, and in each iteration, the sequence with the least impact on the detection accuracy is discarded.

instead of the normal DR, are used for shifting data in and out. The selection of either the decoy DR or the LFSR relies on the comparison between the number of shifting clock cycles and the length (L) of the DR. The LFSR supplies bits to the TDO for the first L clock cycles, while the decoy DR provides bits after L clock cycles, allowing the serial input supplied by the user (via the TDI) to be observed at the TDO after an expected delay (L clock cycles). To achieve this function, the decoy DR needs to be as long as the longest scan-accessible DR and its length should always correspond to the DR that is being selected in real-time.

V. Experiments

Three sets of experiments are performed to evaluate the capability of detecting known, unknown, and disguised attacks, respectively. The experiments are performed using the OpenSPARC T2 microprocessor[3] [15]. A set of legitimate JTAG operations (including 767 programs that contain 125 017 instructions) are generated based on the official documentation, while a variety of attacks (including 1092 programs that contain 154 980 instructions) are created based on the strategies described in Table I.

It is worth noting that, for the representative-based anomaly detector, the representative sequences need to be created using the process described in Section III-C. The result shows that after discarding sequences of frequency <40 and removing redundancy through similarity check, only 407 (<10%) sequences are left. These sequences are further examined through greedy search, as shown in Fig. 7. More precisely, the performance of the remaining sequences is evaluated iteratively, and in each iteration, the sequence with the least impact on the detection accuracy is discarded. The result demonstrates that 200 representative sequences provide similar accuracy as the original, larger set of sequences.

A. Detecting Known JTAG Attacks

In this set of experiments, the capability of detecting known JTAG attacks is evaluated. In particular, for the random forest and the SVM, ten-fold cross-validation is performed (i.e., all data is partitioned into ten subsets of the same size; nine subsets are used for training the classifier, and the tenth for evaluating the accuracy of the resulting classifier; each subset is used once for testing, with the final accuracy being averaged).

[3]The OpenSPARC T2 platform is used for experiments because of its well-documented JTAG functions. Nonetheless, the detectors proposed in this paper are generic, not specific for the OpenSPARC T2 platform.

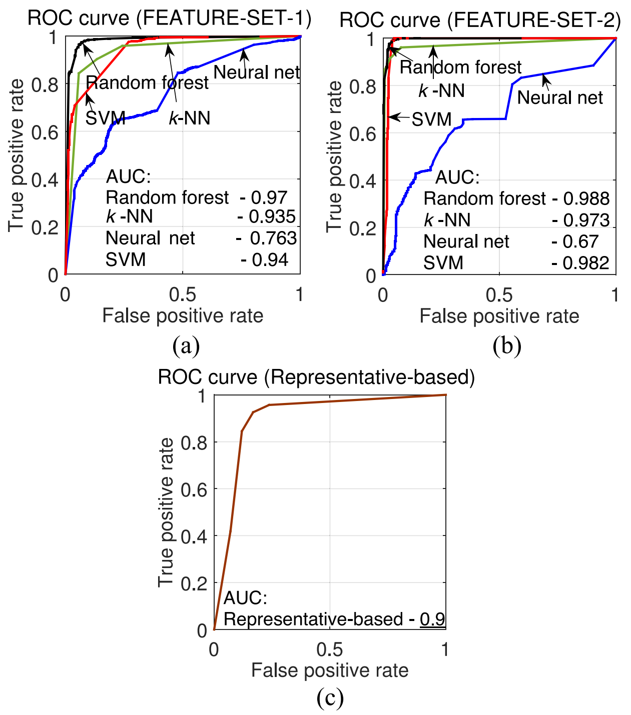


Fig. 8. Performance of detecting known attacks is evaluated for various algorithms using ten-fold cross-validation, with the results plotted as ROC curves. JTAG operation is characterized using (a) FEATURE-SET-1, (b) FEATURE-SET-2, and (c) representative sequences, respectively. The AUC measurements show that the random forest outperforms the other algorithms.

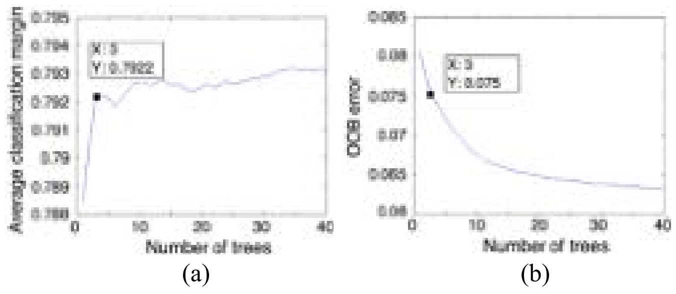


Fig. 9. Average classification margin and OOB error are measured to determine the number of trees. According to the results, three trees demonstrate a significant performance gain at the cost of a modest hardware overhead.

The receive operating characteristic (ROC) curve [45], plotted as the true positive rate (TPR) in terms of the false positive rate (FPR) at various threshold settings, is used for evaluating the performance of making per-instruction prediction (Fig. 8). The ROC curves are plotted for different algorithms, namely the random forest (three bagged trees in the forest), SVM (with RBF kernel), neural network (one hidden layer consisting of ten neurons), k-nearest-neighbor (k-NN, $k = 3$) and the representative-based detector, using both FEATURE-SET-1 and FEATURE-SET-2. The ROC curves in Fig. 8(a) and (b) are plotted using MATLAB built-in functions (i.e., through varying the number of trees that determines the voting result for the random forest, varying the classification threshold from -1 to 1

TABLE IV
JTAG ATTACKING PROGRAMS ARE DIVIDED INTO EIGHT CATEGORIES, EACH ONE TARGETING A DIFFERENT IC COMPONENT

Targeted components			
C_1	Check basic profile	C_5	L2 cache access
C_2	Clock control	C_6	Logic BIST
C_3	Control register	C_7	Memory BIST
C_4	Electronic fuse	C_8	Shadow scan

for the SVM and the neural network, and varying the value of k for k-NN), while the ROC curve for the representative-based detector in Fig. 8(c) is plotted through varying the saturation score s from 1 to 20. Additionally, for FEATURE-SET-2, the optimal value of opcode sequence length, through simulation, is four. In order to compare the performance of the different algorithms, the area under the curve (AUC) is used. AUC measures the capability of a binary classifier to make a correct prediction for a random sample. The results indicate that the random forest outperforms the other algorithms using both FEATURE-SET-1 and FEATURE-SET-2.

For FEATURE-SET-1, a possible explanation for the random forest outperforming the other algorithms resides in the fact that the features in FEATURE-SET-1 are weakly correlated to each other. A random forest can then easily distinguish an attack from legitimate operations by dividing the hyperspace into subspaces using parallel-to-axis boundaries. To demonstrate this, the linear pairwise correlations of the features (measured by Pearson correlation coefficient [46]) in both FEATURE-SET-1 and FEATURE-SET-2 were calculated, revealing that the average pairwise correlation of the features in FEATURE-SET-1 (0.081) is significantly lower than FEATURE-SET-2 (0.442). Another explanation for the superior performance of the random forest is that the random forest, consisting of an ensemble of trees, benefits from the use of bootstrap aggregation.

The number of trees in the forest is decided by the tradeoff between performance gain and hardware overhead. Fig. 9 shows the performance gain of adding more trees to the forest using two metrics, namely average classification margin [47] and out-of-bag (OOB) error [48]. For a binary classification problem, classification margin measures the difference between the probabilities of labeling a sample correctly and incorrectly, while OOB error is the mean prediction error on each training sample x_i using only the trees trained without x_i. Here, the number of trees is set to three, which reduces the OOB error from 8.2% (obtained with a single tree) to 6.7%.

B. Detecting Unknown JTAG Attacks

In this set of experiments, the performance of detecting unknown JTAG attacks is evaluated. An unknown JTAG attack is one that targets a different IC component or exploits a scan-based strategy not used in the training phase. In the first group of experiments, the attacking programs are divided into eight categories based on the components they target (as listed in Table IV), while in the second group, the programs are divided into nine categories based on the strategies they exploit (as listed in Table I).

For the first group of experiments, the capability of detecting attacks that target a new IC component is evaluated using different algorithms (see Table V). The extracted opcode sequences ($n = 4$) are processed by each algorithm, and two metrics, namely accuracy of identifying legitimate operation (A_L) and accuracy of identifying unknown attacks (A_U), are used for evaluating the performance. Here, accuracy is defined as the percentage of correct predictions out of all evaluated opcode sequences. The first set of algorithms involve two-class learning algorithms, namely SVM, neural network (one hidden layer consisting of ten neurons), k-NN ($k = 3$), and the random-forest detector based on FEATURE-SET-1. For these algorithms, seven out of eight categories of attacks in Table IV, and all cases of legitimate operations, are simulated using tenfold cross-validation. The second set of algorithms involve one-class learning algorithms, including the representative-based detector, a one-class SVM ($\nu = 0.1$), and a one-class k-NN (radius $= 2$). A one-class k-NN identifies an opcode sequence by searching for the neighboring area (radius $= 2$) centered by the opcode sequence; if at least one training sample resides within the area, the opcode sequence is then labeled legitimate. For these algorithms, 90% of the legitimate opcode sequences are used for training, while the remaining 10% are used for evaluating A_L. To calculate A_U, each category of the component attack is evaluated as an unknown attack. The next approach, named "match", simply uses the instruction sequences described in the OpenSPARC T2 documentation as legal ones and rejects any sequence that does not conform.

As demonstrated in Table V, the SVM and the representative-based anomaly detector have more balanced performance in identifying legitimate operation and unknown attacks, revealing that they can not only tolerate the variances occurring during legitimate JTAG operation but also detect unknown attacks. A neural network and k-NN are capable of identifying either legitimate operation or an attack, but not both. The overall performance of a one-class SVM and a one-class k-NN is worse than an SVM, especially for detection of unknown attacks. A likely explanation for this outcome is that having some general knowledge of attacks is quite beneficial for detecting new types of attacks. The match approach fails to identify variances exhibited by legitimate operation because the matching rule, by definition, simply does not allow for any variance. The shortcoming of the random-forest detector based on FEATURE-SET-1 resides in the poor accuracy of detecting unknown attacks; however, it performs better than the SVM detector for identifying attacks targeting C_1. A likely explanation is that checking-the-basic-profile of the JTAG may result in frequent easy-to-detect mistakes because the attacker is assumed to have no prior knowledge of the private JTAG functions. The random-forest detector performs better in that situation because various features, other than solely the sequential order of JTAG instructions, are used for characterizing legitimate JTAG operation.

For the second group of experiments, the capability of detecting attacks that exploit a different strategy is evaluated in a similar manner. Table V shows again that the SVM and the representative-based detector have better overall performance.

TABLE V

ACCURACY OF DETECTING LEGITIMATE OPERATION (A_L) AND UNKNOWN ATTACKS (A_U) THAT TARGET DIFFERENT IC COMPONENTS (C_1–C_8) AND EXPLOIT DIFFERENT STRATEGIES (S_4–S_9) ARE EVALUATED FOR VARIOUS ALGORITHMS (OPCODE SEQUENCE LENGTH $n = 4$). (EACH COLUMN LABELED C_i HAS TARGET COMPONENT C_i OF TABLE IV AS THE UNKNOWN ATTACK, THAT IS, NOT USED DURING TRAINING. EACH COLUMN LABELED S_j EXPLOITS STRATEGY S_j OF TABLE I AS THE UNKNOWN ATTACK.) THE HIGHEST A_L AND A_U IS SET IN BOLD FOR EACH C_i AND S_j. ACCORDING TO THE RESULTS, THE SVM AND THE REPRESENTATIVE-BASED ANOMALY DETECTOR SHOW MORE BALANCED PERFORMANCE

Metric	**Algorithm**	**IC component targeted by attacks**								**Strategy exploited by attacks**					
		C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	S_4	S_5	S_6	S_7	S_8	S_9
A_L	Random forest	0.944	0.95	0.953	**0.966**	0.943	0.944	0.957	0.937	0.967	0.949	0.956	0.945	0.955	0.958
	SVM	0.876	0.885	0.876	0.874	0.896	0.876	0.878	0.875	0.876	0.903	0.868	0.886	0.879	0.869
	k-NN	0.954	0.944	**0.97**	0.956	0.949	**0.968**	0.945	0.944	**0.971**	0.956	0.928	0.935	0.92	0.924
	Neural net	0.608	0.562	0.473	0.62	0.308	0.604	0.431	0.547	0.753	0.888	0.198	0.528	0.57	0.504
	Representative-based	**0.96**	**0.96**	0.96	0.96	**0.96**	0.96	**0.96**	**0.96**	0.96	**0.96**	**0.96**	**0.96**	**0.96**	**0.96**
	Match	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71
	1-class SVM	0.676	0.676	0.676	0.676	0.676	0.676	0.676	0.676	0.651	0.651	0.651	0.651	0.651	0.651
	1-class k-NN	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.946	0.946	0.946	0.946	0.946	0.946
A_U	Random forest	0.909	0.354	0.687	0.443	0.403	0.304	0.367	0.943	0.432	0.45	0.578	**0.915**	0.601	0.659
	SVM	0.77	0.855	**0.985**	**0.97**	**1**	0.907	0.845	**1**	0.877	**1**	**1**	0.843	**0.994**	**0.986**
	k-NN	0.028	0.09	0.379	0.056	0.304	0.161	0.25	0.95	0.521	0.303	0.515	0.689	0.573	0.852
	Neural net	**0.934**	**0.876**	0.977	0.954	0.982	0.91	0.829	0.999	0.801	0.988	0.998	0.659	0.85	0.918
	Representative-base	0.596	0.758	0.969	0.93	0.993	0.803	0.8	**1**	**1**	0.992	0.621	0.783	0.989	0.949
	Match	0.538	0.751	0.964	0.916	0.993	0.79	0.772	**1**	0.599	0.992	**1**	0.762	0.973	0.942
	1-class SVM	0.574	0.504	0.685	0.437	0.936	0.514	**0.858**	**1**	0.447	0.929	**1**	0.376	0.833	0.704
	1-class k-NN	0.825	0.626	0.947	0.807	0.358	**0.975**	0.833	**1**	0.349	0.945	**1**	0.398	0.789	0.854

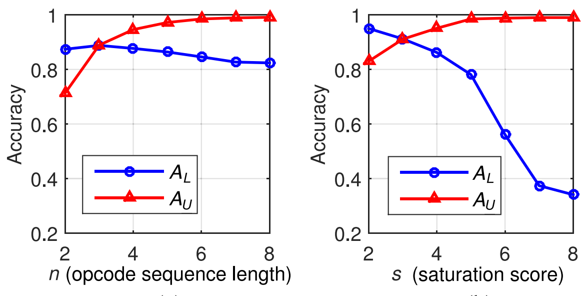


Fig. 10. (a) For the SVM detector, the detection accuracy (y-axis), as a function of opcode sequence length (x-axis), is evaluated for legitimate operation (A_L) and unknown attacks (A_U). (b) For the representative-based anomaly detector, the detection accuracy (y-axis), as a function of s (x-axis), is evaluated for legitimate operation and unknown attacks.

Fig. 10(a) demonstrates that, for the SVM detector, the accuracy of identifying legitimate operation and unknown attacks trends in opposite directions when the opcode sequence length n increases. The accuracy is an average of each category of attacks weighted by its population size. Fig. 10(a) reveals that a larger n is preferable for detecting unknown attacks, but at the cost of more false positives. It also reveals that the length of most legitimate JTAG operations is naturally less than seven or eight. The selection of the value for n depends on the cost ratio of a false positive and a false negative. If a false positive and a false negative have identical cost, then the optimal value for n is four or five.

For the representative-based anomaly detector, the accuracy of identifying legitimate operation and unknown attacks is affected by the saturation score s in a similar manner as shown in Fig. 10(b). Fig. 10(b) demonstrates that a higher s is preferable for detecting unknown attacks, but at a significant cost of false positives. If a false positive and a false negative have equal cost, then the optimal value for s is three or four.

TABLE VI

TO EVALUATE THE DELAYED LABELING, THE ACCURACY OF CLASSIFYING INSTRUCTION WINDOWS (LABELED AS "DELAYED") IS COMPARED WITH THE ACCURACY OF PER-INSTRUCTION CLASSIFICATION (LABELED AS "PER-INSTR") FOR THE THREE DETECTORS. THE VALUES FOR q AND p SHOWN IN THE TABLE ACHIEVE BETTER ACCURACY THAN OTHER VALUES ACCORDING TO SIMULATION RESULTS. THE DELAYED LABELING DEMONSTRATES AN IMPROVEMENT IN CLASSIFICATION ACCURACY

Detector	(p, q)	**Legitimate**		**Unknown attack**	
		per-instr	delayed	per-instr	delayed
Random forest	(4, 5)	0.932	0.962	0.944	0.979
SVM	(3, 5)	0.891	0.939	0.962	0.993
Representative-based	(5, 5)	0.856	0.911	0.947	0.965

C. Delayed Labeling and Disguised Attacks

This set of experiments aim to evaluate the effectiveness of delayed labeling, i.e., the labeling of JTAG operation as legitimate or as an attack based on the predictions of q consecutive instructions. Similar to Section V-B, one out of eight categories of attacks in Table IV is treated as unknown attack, while the remaining seven categories are treated as known attacks. In the experiments, seven known categories of attacks, together with a half of randomly chosen legitimate JTAG programs (described in the beginning of Section V), are used for training the classifier. The unknown category of attacks and the other half legitimate JTAG programs are used for evaluation, in which the prediction is made for every nonoverlapping window of q instructions. More precisely, if a window of q instructions contains no fewer than p positive per-instruction predictions, then the corresponding window is labeled as an attack; otherwise, it is labeled as legitimate. Accuracy is measured using the percentage of correctly classified windows, which is then compared with the accuracy of per-instruction classification (see Table VI). The comparison shows that delayed labeling improves detection accuracy for all detectors.

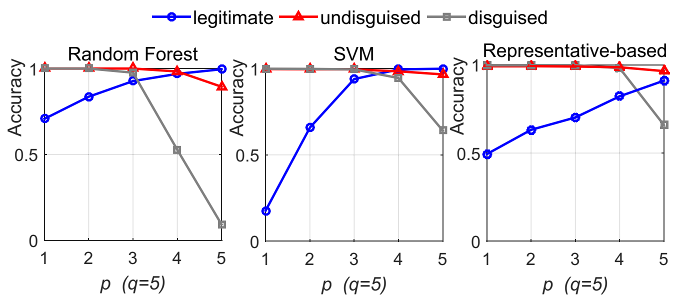
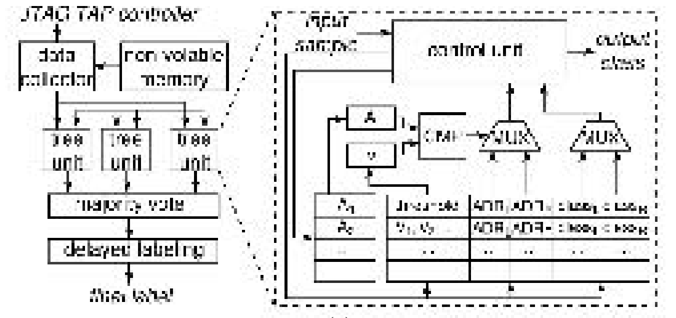
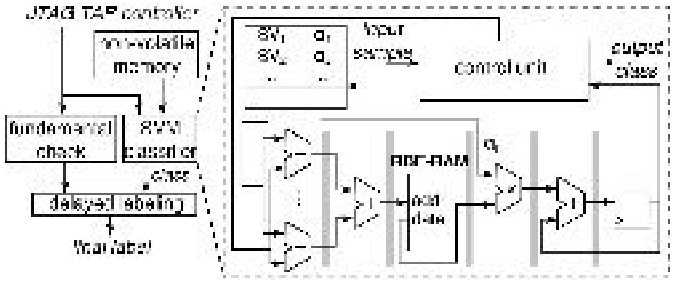


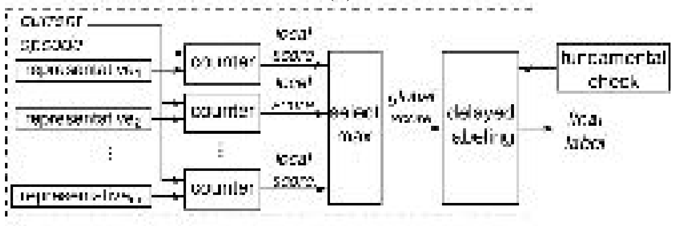
Fig. 11. Accuracy of identifying disguised attacks is compared with the accuracy of identifying legitimate operation and undisguised attacks for the random-forest detector, the SVM detector, and the representative-based anomaly detector. The results show that a larger window increases the risk of a disguised attack.



(a)



(b)



(c)

Fig. 12. Architecture of the (a) random-forest detector, (b) SVM detector, and (c) representative-based anomaly detector.

The delayed window, however, may be used by the attacker to insert malicious instruction sequences. If an attacker knows how to operate a portion of the private JTAG functions but not all. Then, to discover the unknown functions, the attacker will attempt to disguise attacks through interleaving them with already-known legitimate sequences. For example, if an attacker loads one unknown instruction into the JTAG after every nine legitimate ones, then it may escape the detection because 90% of the instructions seem "legitimate". To verify this assertion, an experiment is performed as described next. An attacker is assumed to know how to operate control registers (shown as C_3 in Table IV). To disguise the attacks, a legitimate operation of the control registers (with opcode sequence length = 9) is inserted into attacking programs after every segment of l instructions, where l varies from one to three randomly. These disguised attacking programs are then cut into sequences of N instructions. Here, N is set to 14 so that each N-instruction sequence contains at least one complete attacking segment. The accuracy of identifying disguised attacks is compared with the accuracy of identifying legitimate operation and undisguised attacks (Fig. 11). As p increases, the detection becomes more conservative, which means that more JTAG operations are labeled as legitimate and fewer as attacks. The results show that the SVM detector has the most balanced performance in identifying legitimate operation, disguised attacks, and undisguised attacks. Besides, the setting of ($p = 4$, $q = 5$) achieves better accuracy in detecting disguised attacks (94%) than other settings. A possible reason for the SVM detector being able to identify disguised attacks can be described as follows. SVM classification is based on a sequence of n instructions, so each per-instruction prediction considers $(n - 1)$ prior instruction transitions. A back-and-forth jump between the unknown functions and the known ones may cause at least n positive per-instruction predictions that are easy to be detected by delayed labeling.

VI. Hardware Implementation of Detectors

The proposed detectors are implemented within the OpenSPARC T2 design in register-transfer level (RTL). Further, the detectors are implemented using the *Xilinx Zynq7000 ZC706* FPGA platform.

A. RTL Design Within OpenSPARC T2

Fig. 12(a) shows the architecture of the random-forest detector. The parameters of the learned forest are loaded into a RAM when the chip is powered on. Since the access to non-volatile memory only occurs at the power-on stage, the online classification latency is not impacted. A tree unit employs the architecture of a universal tree node described in [49]. During the classification, one node is processed per clock cycle and all trees are processed in parallel, so the required number of clock cycles is equal to the depth of the deepest tree. It is noted that a continuous feature is compared with a threshold, while a discrete feature is compared with a set of values (i.e., v_1, $v_2, \ldots$). The size of the RAM storing the learned forest relies on the number of trees and the size of each tree. As explained in Section V-B, the number of trees is set to three. In this paper, the size of the RAM is 3 KB assuming at most 256 nodes per tree, and the latency for classifying a JTAG instruction is eight clock cycles.

Fig. 12(b) shows the architecture of the SVM detector. When the chip is powered on, the parameters of the learned SVM (i.e., each SV$_i$ and each corresponding weight α_i) are loaded into a RAM (named SVM-RAM). The operation of the JTAG TAP controller is monitored by both a fundamental-check module and the SVM classifier. The SVM classifier employs a fully pipelined architecture that consists of distance

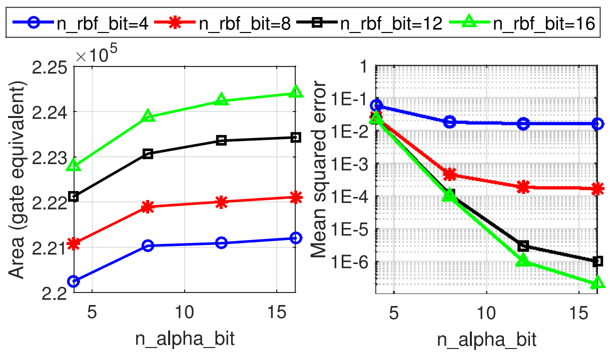


Fig. 13. Area of the SVM detector and the MSE of the classification results (compared to software) are impacted by the number of bits for α_i (n_alpha_bit) and the RBF values (n_rbf_bit).

TABLE VII
SYNTHESIS RESULTS ARE COMPARED FOR THE THREE DETECTORS

Detector	Random forest	SVM	Representative-based
RAM (KB)	4	2.94	-
Area (gate equivalents)	371,013	224,688	32,445
% of OpenSPARC T2 area	1.81%	1.10%	0.16%
Area compared to JTAG	57×	35×	5×
Latency (clock cycles)	8	520	6
Dynamic power (mW)	15.50	9.45	0.74

computation, an RBF kernel computation, a multiplication, and an addition. The distance computation employs L1-norm and the computation of the RBF kernel employs an LUT (loaded in RBF-RAM) that is built in advance by calculating all possible values. It is noted that the SVM classification results computed by hardware may differ from software due to limited precision for α_i and the RBF values. Fig. 13 shows that a higher precision reduces the mean squared error (MSE) for the classification results (compared to software), but at the cost of more chip area. In the implementation, both α_i and the RBF value are represented using 12 bits, because the 12-bit representation reduces the MSE significantly to a low level (3×10^{-6}) when compared to the representations with fewer bits.

The chip area required by an SVM classifier depends on n (i.e., opcode sequence length) in two ways. First, n affects the width of the pipeline linearly since it represents the dimension of data. Second, the size of SVM-RAM relies on n which affects both the number of SVs and the size of each SV. A larger n produces more SVs, that is, for example, the numbers of SVs produced by $n = 3, 4, 5$, and 6 are 763, 844, 901, and 962, respectively. However, because these numbers are between 512 and 1024, the number of rows remains 1024 (each row stores an SV). Consequently, the area for the SVM-RAM is linearly dependent on n. The size of the RBF-RAM also depends on n linearly. To summarize, the relative area overhead in terms of n is: 0.75 ($n = 3$), 1 ($n = 4$), 1.25 ($n = 5$), and 1.5 ($n = 6$).

Fig. 12(c) shows the architecture of the representative-based anomaly detector. Each representative JTAG sequence is associated with a counter that records a local matching score. A global score is then computed by selecting the maximum local score. The structure of the select-max function utilizes

TABLE VIII
FPGA RESOURCE UTILIZATION AND POWER CONSUMPTION ARE EVALUATED FOR THE THREE DETECTORS

Detector	Random forest	SVM	Representative-based
LUT	399 (0.18%)	402 (0.18%)	2702 (1.24%)
LUTRAM	624 (0.89%)	385 (0.55%)	1 (<0.01%)
FF	252 (0.06%)	236 (0.05%)	1736 (0.4%)
Power (100 MHz)	346 mW	335 mW	342 mW

a multistage pipeline due to a large number of representative sequences. The size of the design depends on the number of representative sequences (200 representatives are used in this paper as elaborated in the beginning of Section V). The latency for identifying a JTAG instruction is six clock cycles assuming that the select-max function has four stages.

Table VII shows the synthesis results using Synopsys Design Compiler with a 0.18 μm library. The random-forest detector consumes the largest chip area, while the SVM detector requires the most clock cycles for making a prediction. The area of the detectors are acceptable compared to the whole design considering that the OpenSPARC T2, due to its complex testing/debugging functions, is likely to require a complex detector. However, when compared to the JTAG, the detectors are much larger, namely 57, 35, and 5 times, respectively. This is as expected because a classical JTAG, typically consisting of a TAP controller and DRs (excluding scan chains), is small.

B. FPGA Implementation

The detectors are also evaluated using the *Xilinx Zynq7000 ZC706* FPGA board. The on-board ARM processor is used for operating the detector. Specifically, the processor communicates with the detector through an AMBA AXI protocol [50] included in the *Vivado Design Suite* [51]. The communication interface contains a set of 32-bit registers; a user can operate the detector (or collect results from the detector) through writing to (or reading from) the registers using a C program. More, the processor and the detector are synchronized through handshaking.

To support simulation, in addition to the detector, the JTAG TAP controller of the OpenSPARC T2 and three RAMs storing JTAG inputs, namely TDI, TMS, and TRST, are also implemented in the FPGA. Two clocks with different frequencies are also used. The detector operates with a faster clock than the JTAG TAP controller such that the classification for a JTAG instruction can be completed before the next instruction is supplied.

Table VIII shows the FPGA resource utilization and the power consumption estimated by the *Vivado Design Suite* through post-implementation analysis. The resource utilization is evaluated using LUTs, LUTRAM, and flip-flops (FFs), all of which are based on configurable logic blocks and, therefore, are not reflected in Fig. 12. The results show that the random forest and the SVM detectors utilize a similar amount of LUTs and FFs, but the random forest detector requires more RAM. The results also show that the representative-based detector consumes more LUTs and FFs but less RAM. This is because 1) the representative sequences are stored in shift registers instead of an RAM and 2) each representative sequence requires independent logic for sequence comparison.

VII. Discussion

Other issues, including false positives, scalability of the detectors, collection of new attacks, and possible attacks to the detectors, are discussed in this section.

A. False Positives

A false positive occurs when a legitimate user is classified as an attacker. It causes inconvenience for legitimate users because the JTAG will be permanently protected in the case of false positives. Fig. 8 shows a tradeoff between the FPR and the TPR, meaning that a reduction of the FPR leads to a higher probability that an attacker escapes detection. The delayed labeling is an effective method for mitigating false positives. For example, in Fig. 8(b), an FPR of 3.2% is observed for the SVM if the TPR is 96.2%, but a delayed window with four instructions can reduce the FPR to 2.1% and raises the TPR to 98.1%. Considering that an FPR of 2.1% is small, and that in-field uses of JTAG involve only a few scenarios (e.g., flash programming and software debugging) and are typically not frequent [52], [53], the JTAG is not likely to enter the protection mode. In other words, if used legitimately, the JTAG is expected to be accessible for a long enough period.

B. Scalability

As new types of attacks emerge, the performance and the size of the detectors may be impacted. We conduct the following experiment as a preliminary evaluation. First, attacks S_4–S_9 in Table I are gradually added to the training set for both the random forest and the SVM. Next, the performance of both detectors are evaluated using ten-fold cross-validation. The size of the random forest is evaluated using the average number of nodes per tree, while the SVM size is evaluated using the number of SVs. The results in Fig. 14 demonstrate an overall increase of size and a decrease of performance as more types of attacks are used in classifier training. However, the impact of different attacks varies. The size and the accuracy change dramatically for the first two types of attacks (i.e., S_4 and S_5); then they become more stable as the variety of attacks increases. S_6 is more likely a hard-to-detect attack because it causes a significant degradation in accuracy.

C. Update of Detectors

Although the detectors demonstrate a potential of detecting unknown attacks, it still does not guarantee that the detectors can identify all types of attacks not included in training. Collecting JTAG attacks is an important but difficult process because: 1) hardware attackers typically do not publish their attacks and 2) unknown attacks always exist. Nevertheless, hardware companies should still be aware of potential attacks not only for mitigating the current vulnerability but also for improving security of future designs. We suggest that hardware companies build a database of JTAG attacks and collect

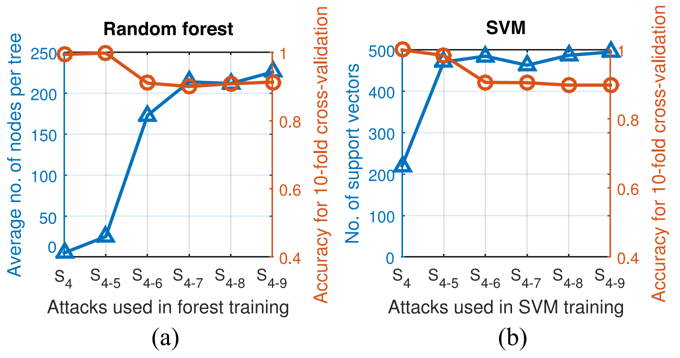


Fig. 14. Performance and the size of the (a) random forest and (b) SVM are impacted by the variety of attacks used in classifier training.

potential attacks using similar strategies as anti-virus software, such as analyzing user report and periodic security examination. The operation that triggers the protection mode can also be saved in on-chip memory and then collected via software or scan chains for further analyses.

When new attacks are collected, it needs to be checked if they can be effectively detected by the existing detector with high confidence. If not, the detector is retrained and updated to the on-chip memory. This can be achieved by incorporating the parameters of the detector into firmware that is typically upgraded via USB [54] or JTAG [55]. To ensure security of firmware upgrade, two issues need to be considered. First, only authorized firmware can be upgraded. This can be realized using firmware signature [56], i.e., generating a unique hash code for each authorized copy and appending it to the firmware when distributed. The other issue involves leakage of firmware contents. This can be mitigated by encrypting the firmware using cryptographic blocks [57]. The decryption of firmware needs to be done on chip, which however requires additional chip area. Since firmware, as well as its encryption/decryption, is ubiquitous for electronic systems, the overhead caused by incorporating detector parameters into firmware can be amortized.

D. Possible Attacks to Detectors

The proposed detectors identify JTAG attacks through examining JTAG operation over time. However, their effectiveness can be circumvented in following scenarios. The first scenario involves an attacker that has full knowledge of the private JTAG functions of what they are and how they should be operated. In this scenario, the detectors have no effect to the attacker at all. The second scenario involves an attacker that can perform attacks on multiple chips and is therefore more likely to escape detection. Nevertheless, it is still challenging and costly for the attacker to recognize that the JTAG has entered into the protection mode. The third scenario assumes the attacker has obtained the parameters of the detector (i.e., the tree nodes of the forest, the SVs and their α for the SVM, or the representative instruction sequences). In this scenario, the attacker can escape detection easily. However, if the parameters are not leaked, then the attacker can only reverse engineer them using brute-force guessing. Hence, it is critical to avoid

TABLE IX
PROPOSED DETECTORS ARE COMPARED WITH OTHER JTAG PROTECTION TECHNIQUES. IN THE COLUMNS OF SECURITY AND OVERHEAD, MORE +'S INDICATE BETTER SECURITY AND MORE OVERHEAD, RESPECTIVELY

Technique	Security	Overhead	Protected function	Drawback	Vulnerability
Disable JTAG [30]	+++++	+	All	Disable in-field debugging	Invasive attack
On-chip compression/compaction [25]	+	++	Boundary scan	Not protect private functions	Differential attack
Encryption [26]–[29], [58]–[60]	++++	+++	All	Key management	Key leakage
Signature-based detector [31]	++	+++	Private	False negatives	Unknown attack
FSM-based anomaly detector [32]	+++	+++	Private	False positives	Disguised attack
Random-forest detector	+++	++++	Private	Assume attacker does not know private functions	Unknown attack, disguised attack
SVM detector	++++	+++++			
Representative-based anomaly detector	+++	+++			

leakage of the data stored in the NVM and the logic of the detector.

The effectiveness of the JTAG protection architecture may also be circumvented in two scenarios. For the first scenario, a read-after-write operation may exhibit inconsistent results, which can be noticed by an attacker. The second scenario involves the random bit stream supplied by the LFSR. However, these two scenarios do not necessarily reveal an evidence of the JTAG under protection because an attacker is assumed to have no prior knowledge of the private JTAG functions. Under this assumption, an attacker neither recognizes an accidentally executed read-after-write operation nor knows that a selected DR should provide certain data rather than the observed bits.

E. Comparison to Other Techniques

The learning-based detectors proposed in this paper are compared with other JTAG protection techniques as shown in Table IX. It is noted that the proposed detectors are orthogonal with some other techniques (such as JTAG encryption), and thus they can be combined to achieve complementing protection for the JTAG.

VIII. CONCLUSION

Security is becoming an important concern for integrated systems because attacks via the JTAG reveal possibilities of improperly acquiring on-chip data and IP design. In this paper, three detectors examining JTAG operation and a secure JTAG architecture are proposed to protect the JTAG from attacks, assuming that an attacker only has access to JTAG port and is unaware of the private JTAG functions. An IC designer can select any of the detectors based on the requirements for security and overhead. The detector is general for all JTAGs, but the parameters are not (due to different debugging functions and instruction opcodes), meaning that for a different JTAG design, a new set of parameters need be derived.

REFERENCES

[1] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2012, pp. 23–40.

[2] Z. Basnight, J. Butts, J. Lopez, and T. Dube, "Firmware modification attacks on programmable logic controllers," *Int. J. Crit. Infrastruct. Protect.*, vol. 6, no. 2, pp. 76–84, 2013.

[3] I. M. F. Breeuwsma, "Forensic imaging of embedded systems using JTAG (boundary-scan)," *Digit. Invest.*, vol. 3, no. 1, pp. 32–42, 2006.

[4] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens, "Transaction security system," *IBM Syst. J.*, vol. 30, no. 2, pp. 206–229, 1991.

[5] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *Proc. Int. Test Conf.*, 2004, pp. 339–344.

[6] Y. Liu, K. Wu, and R. Karri, "Scan-based attacks on linear feedback shift register based stream ciphers," *ACM Trans. Design Autom. Electron. Syst.*, vol. 16, no. 2, pp. 1–15, 2011.

[7] R. Nara, K. Satoh, M. Yanagisawa, T. Ohtsuki, and N. Togawa, "Scan-based side-channel attack against RSA cryptosystems using scan signatures," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 93, no. 12, pp. 2481–2489, 2010.

[8] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Are advanced DfT structures sufficient for preventing scan-attacks?" in *Proc. VLSI Test Symp.*, 2012, pp. 246–251.

[9] O. Choudary, *Breaking Smartcards Using Power Analysis*, University of Cambridge, Cambridge, U.K., 2005.

[10] V. Banciu, E. Oswald, and C. Whitnall, "Reliable information extraction for single trace attacks," in *Proc. Design Autom. Test Europe*, 2015, pp. 133–138.

[11] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "SCANDALee: A side-channel-based disassembler using local electromagnetic emanations," in *Proc. Design Autom. Test Europe*, 2015, pp. 139–144.

[12] R. Kumar, P. Jovanovic, W. Burleson, and I. Polian, "Parametric Trojans for fault-injection attacks on cryptographic hardware," in *Proc. Fault Diagnosis Tolerance Cryptography*, 2014, pp. 18–28.

[13] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. New York, NY, USA: Springer, 2012.

[14] C. Maunder, "The joint test action group," *Comput.-Aided Eng. J.*, vol. 3, no. 4, pp. 121–122, Aug. 1986.

[15] Oracle. *OpenSPARC T2*. Accessed: Oct. 1, 2013. [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/

[16] B. Yang, K. Wu, and R. Karri, "Secure scan: A design-for-test architecture for crypto chips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2287–2293, Oct. 2006.

[17] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Scan-based attack against elliptic curve cryptosystems," in *Proc. Asia South Pac. Design Autom. Conf.*, 2010, pp. 407–412.

[18] J. Da Rolt, A. Das, and G. Di Natale, "A new scan attack on RSA in presence of industrial countermeasures," in *Proc. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, 2012, pp. 89–104.

[19] J. Da Rolt *et al.*, "A scan-based attack on elliptic curve cryptosystems in presence of industrial design-for-testability structures," in *Proc. VLSI Nanotechnol. Syst.*, 2012, pp. 43–48.

[20] S. S. Ali, O. Sinanoglu, S. M. Saeed, and R. Karri, "New scan-based attack using only the test mode," in *Proc. Int. Conf. Very Large Scale Integr.*, 2013, pp. 234–239.

[21] F. Domke, "Blackbox JTAG reverse engineering," in *Proc. Chaos Commun. Congr.*, 2009, pp. 1–5.

[22] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[23] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[24] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Thwarting scan-based attacks on secure-ICs with on-chip comparison," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 4, pp. 947–951, Apr. 2014.

[25] *High Quality Test Solutions for Secure Applications*, Mentor Graph., Wilsonville, OR, USA, 2010.
[26] A. Das, U. Kocabaş, A.-R. Sadeghi, and I. Verbauwhede, “PUF-based Secure test wrapper design for cryptographic SoC testing,” in *Proc. Design Autom. Test Europe*, 2012, pp. 866–869.
[27] S. Paul, “VIm-Scan: A low overhead scan design approach for protection of secret key in scan-based secure chips,” in *Proc. VLSI Test Symp.*, 2007, pp. 455–460.
[28] F. Novak and A. Biasizzo, “Security extension for IEEE Std 1149.1,” *J. Electron. Test.*, vol. 22, no. 3, pp. 301–303, 2006.
[29] J. Dworak and A. Crouch, “Don’t forget to lock your SIB: Hiding instruments using P16871,” in *Proc. Int. Test Conf.*, 2013, pp. 1–10.
[30] *Design Security in Nonvolatile Flash and Antifuse FPGAs*, Actel, San Jose, CA, USA, 2002.
[31] D. E. Denning, “An intrusion-detection model,” *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987.
[32] K. Ilgun, R. A. Kemmerer, and P. A. Porras, “State transition analysis: A rule-based intrusion detection approach,” *IEEE Trans. Softw. Eng.*, vol. 21, no. 3, pp. 181–199, Mar. 1995.
[33] J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo, “Stochastic protocol modeling for anomaly based network intrusion detection,” in *Proc. Int. Workshop Inf. Assurance*, 2003, pp. 3–12.
[34] X. Ren, V. G. Tavares, and R. D. Blanton, “Detection of illegitimate access to JTAG via statistical learning in chip,” in *Proc. Design Autom. Test Europe*, 2015, pp. 109–114.
[35] X. Ren, R. D. Blanton, and V. G. Tavares, “A learning-based approach to secure JTAG against unseen scan-based attacks,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2016, pp. 541–546.
[36] G.-M. Chiu and J. C.-M. Li, “A secure test wrapper design against internal and boundary scan attacks for embedded cores,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 126–134, Jan. 2012.
[37] J. Lee, M. Tebranipoor, and J. Plusquellic, “A low-cost solution for protecting IPs against scan-based side-channel attacks,” in *Proc. 24th IEEE VLSI Test Symp.*, 2006, pp. 94–99.
[38] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, “Securing designs against scan-based side-channel attacks,” *IEEE Trans. Depend. Secure Comput.*, vol. 4, no. 4, pp. 325–336, Oct./Dec. 2007.
[39] D. Hely *et al.*, “Scan design and secure chip [secure IC testing],” in *Proc. Int. On-Line Test. Symp.*, vol. 4, 2004, pp. 219–224.
[40] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, “Securing scan design using lock and key technique,” in *Proc. IEEE Defect Fault Tolerance VLSI Syst.*, 2005, pp. 51–62.
[41] I. Slochinsky, “Introduction to embedded reverse engineering for PC reversers,” in *Proc. REcon Conf.*, 2010.
[42] C. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.
[43] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, “Support vector method for novelty detection,” in *Proc. Neural Inf. Process. Syst.*, 1999, pp. 582–588.
[44] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. New York, NY, USA: Springer, 2000.
[45] *ROC Curve*. Accessed: Aug. 1, 2014. [Online]. Available: http://www.mathworks.com/help/stats/perfcurve.html
[46] J. Biesiada and W. Duch, “Feature selection for high-dimensional data—A Pearson redundancy based filter,” in *Computer Recognition Systems 2*. Heidelberg, Germany: Springer, 2007, pp. 242–249.
[47] Classification Margin. Accessed: Mar. 1, 2015. [Online]. Available: http://www.mathworks.com/help/stats/compactclassificationdiscriminant.margin.html
[48] L. Breiman, “Out-of-bag estimation,” Citeseer, Rep., 1996.
[49] J. R. Struharik, “Implementing decision trees in hardware,” in *Proc. Int. Symp. Intell. Syst. Informat.*, 2011, pp. 41–46.
[50] *AMBA AXI and ACE Protocol Specification*, ARM, Cambridge, U.K., 2011.
[51] *Vivado Design Suite*. Accessed: Mar. 1, 2015. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html
[52] A. Dushistova, *Debugging With JTAG*, MontaVista Softw. Inc., San Jose, CA, USA, 2009.
[53] A. Sirotkin. (2010). *Debugging the Linux Kernel With JTAG*. Accessed: Nov. 1, 2017. [Online]. Available: https://www.embedded.com/design/operating-systems/4207333/Debugging-the-Linux-kernel-with-JTAG
[54] J. Kaye, *BL600 Firmware Upgrade Over JTAG*, Laird Technol., Chesterfield, MO, USA, 2015.
[55] *SIS3300/3301 JTAG Firmware Upgrade Instructions*, Struck Innov. Syst., Hamburg, Germany, 2005.
[56] Hewlett Packard Enterprise. *HPE Digitally Signed Firmware*. Accessed: Dec. 1, 2017. [Online]. Available: https://hpe.techdata.ch/fileadmin/user_upload/Infos/Produkt/SSD/HPE_Digitally_Signed_Firmware_Disks.pdf
[57] *AN0060: Bootloader With AES Encryption*, Silicon Labs, Austin, TX, USA, 2016.
[58] R. F. Buskey and B. B. Frosik, “Protected JTAG,” in *Proc. Int. Conf. Parallel Process. Workshops*, 2006, p. 8.
[59] A. Das *et al.*, “Secure JTAG implementation using Schnorr protocol,” *J. Electron. Test.*, vol. 29, no. 2, pp. 193–209, 2013.
[60] K.-Y. Park, S.-G. Yoo, and J.-H. Kim, “Debug port protection mechanism for secure embedded devices,” *J. Semicond. Technol. Sci.*, vol. 12, no. 2, pp. 240–253, 2012.

Xuanle Ren (S’15) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2012. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA, and the University of Porto, Porto, Portugal, supported by the Carnegie Mellon Portugal Program.

He is a member of the Advanced Chip Testing Laboratory, Carnegie Mellon University and a Researcher with INESC TEC, Porto, Portugal. His current research interests include security of integrated system and network, and machine learning in chip.

Francisco Pimentel Torres (S’10) received the B.S. degree in electrical and computer engineering and the M.S. degree in electrical and computer engineering (microelectronics and embedded systems) from the University of Porto, Porto, Portugal, in 2013 and 2015, respectively.

He did research internships with the Technical University of Munich, Munich, Germany, in 2013, and Carnegie Mellon University, Pittsburgh, PA, USA, in 2015. He is currently a Digital Design Engineer with Qualcomm, Cambridge, U.K. His current research interests include integrated circuit design and testing, machine learning, and computer architecture.

R. D. (Shawn) Blanton (S’93–M’95–SM’03–F’09) received the B.S. degree in engineering from Calvin College, Grand Rapids, MI, USA, in 1987, the M.S. degree in electrical engineering from the University of Arizona, Tucson, AZ, USA, in 1989, and the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA, in 1995.

He is currently a Professor of electrical and computer engineering with Carnegie Mellon University, Pittsburgh, PA, USA. He also serves as the Associate Director of the SYSU-CMU Joint Institute Engineering, and as the Founder and the Leader of the Advanced Chip Testing Laboratory. His current research interest includes test and diagnosis of integrated heterogeneous systems.

Vítor Grade Tavares (M’03) received the Licenciatura and M.S. degrees in electrical engineering from the University of Aveiro, Aveiro, Portugal, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering from the Computational NeuroEngineering Laboratory, University of Florida, Gainesville, FL, USA, in 2001.

In 1999, he joined the University of Porto, Porto, Portugal, as an Invited Assistant, where he has been an Assistant Professor since 2002. In 2010, he was a Visiting Professor with Carnegie Mellon University, Pittsburgh, PA, USA. He is also a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência, Porto. His current research interests include low-power, mixed-signal and neuromorphic integrated-chip design and biomimetic computing, CMOS RF integrated circuit design for wireless sensor networks, and transparent electronics.