Using Scan Side Channel to Detect IP Theft

Leonid Azriel, *Student Member, IEEE*, Ran Ginosar, *Senior Member, IEEE*, Shay Gueron, and Avi Mendelson, *Senior Member, IEEE*

***Abstract*— In the growing heterogeneous Internet of Things market, which embraces a plurality of vendors and service providers, IP protection plays a central role. This paper proposes a process for the detection of IP theft in VLSI devices that exploits the internal test scan chains, designed for production test automation. The scan chains supply direct access to the internal registers in the device, enabling combinational analysis of the device logic. By using Boolean function learning methods, the learner creates a partial dependence graph of the internal flip-flops. The graph is further partitioned using the shared nearest neighbors graph clustering method, and individual blocks of combinational logic are isolated. These blocks can be matched with known building blocks that compose the original function. This enables reconstruction of the function implementation to the level of pipeline structure. The IP owner can compare the resulting structure with his own implementation to confirm whether an IP violation has occurred. We demonstrate the power of the presented approach with a test case of an open source Bitcoin SHA-256 accelerator, containing more than 80 000 registers. With the presented method, we discover the microarchitecture of the module, locate all the main components of the SHA-256 algorithm, and learn the module's flow control. In addition to the direct recognition of the IP content, we also demonstrate a combination of reverse engineering and watermark methods. We define a new watermark structure—pipeline-associated watermark (PAW), combined with pipeline stages that can be detected with the scan-based reverse engineering method.**

***Index Terms*— Hardware security, intellectual property protection, reverse engineering, scan side channel, side channel attacks.**

I. Introduction

IN THE highly distributed horizontal model of semiconductor development, which involves multiple parties all over the globe, IP piracy has become a significant concern [1]. Various legal means are available for the IP owner's use. They include patents, copyrights, contracts, trademarks, and trade secrets [2]. Legal means are accompanied by technical methods of securing the hardware design IP. Vast research has been devoted to finding an efficient method for IP protection, either by preventing the theft or by detecting IP violation.

In prevention, various obfuscation methods hinder IP piracy. Among them are encryption and scrambling of the design data at all the development stages, including RTL, gate level netlist, or physical layout. Unfortunately, with sufficient effort and motivation, obfuscation methods can be defeated. Griffin *et al.* [3] propose an IC locking methodology that utilizes process variations and insert sensors during synthesis to generate unique keys that will be used for the activation of the device. This and similar active protection methods that involve activation usually require special implementation flow; they also put additional burden on vendor and user. Moreover, they do not address RTL level IP theft.

In detection, the IC metering technique, in which every IC instance is uniquely marked [4], [5], has been proposed. Among other detection methods, watermarking techniques have been extensively studied. Watermarks alter the design of the device without harming its main function. For example, constraint-based watermarks [6], [7] introduce additional constraints to the design, such as specific encoding of state machine states, adding hidden transitions to the state machines [8], or assigning specific values to the *don't care* conditions. In [7], watermark techniques based on physical design topology have been proposed. The watermarking techniques equip the IP owner with evidence that will serve her if the theft case goes to court.

Constraint-based watermarking is probabilistic [2], [9]. Namely, there is some probability that even without explicit setting of the watermark-specific constraints, the original design will comply with them. For example, random selection of the state encoding may accidentally match the encoding selected by the watermark. Also, the aforementioned methods require notable effort at the design stage. In addition, the detection of the watermarks or similar structures remains a challenge [10]. In any case, the target of IP protection and the watermark are two distinct objects; therefore, the detection of a watermark does not automatically infer IP violation. It is on the IP owner to prove the connection.

Fan [11] proposes to combine watermarks and testing techniques by embedding watermarks in test paths, including scan test. The proposed technique is invoked as a post-RTL process and involves additional back-end stages. As such, it can be bypassed by avoiding these additional stages for a soft macro or making small incremental changes to a hard macro IP. In general, inserting watermarks at the earliest possible stage provides better protection against their removal by tweaking the design implementation process.

In contrast to the watermarking methods, forensic techniques aim to detect the IP violation directly by detecting the specific constructs in the target designs. Wong *et al.* [12] discuss algorithms that potentially detect the synthesis tool used to generate the design by property matching of functions.

Manuscript received January 22, 2017; revised April 19, 2017; accepted May 25, 2017. *(Corresponding author: Leonid Azriel.)*

L. Azriel, R. Ginosar, and A. Mendelson are with the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: leonida@tx.technion.ac.il; ran@ee.technion.ac.il; avi.mendelson@technion.ac.il).

S. Gueron is with the Department of Mathematics, University of Haifa, Haifa 3498838, Israel, and also with Amazon Web Services, Seattle 98108, WA, USA (e-mail: shay@math.haifa.ac.il).

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/TVLSI.2017.2715188

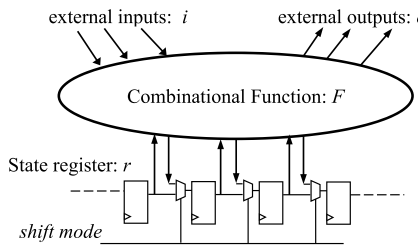


Fig. 1. Scan Design. Scan insertion adds multiplexers to each flip-flop in the chip, so that when shift mode is on, all the flip-flops are accessible from off-chip via a serial channel. The combinational function F represents the collection of combinational logic cones connecting flip-flops to one another and with primary inputs/outputs.

A similar approach can also be used to match the RTL level by comparing relevant properties.

In this paper, we propose a forensic method for IP theft detection effective in both finding watermarks and direct detection of the theft. The proposed method is based on full reverse engineering that enables extracting special structures or patented implementation details from the target design. The new technique belongs to the class of side channel analysis for reverse engineering that has been discussed in several publications [13]–[15], where power analysis was used. We demonstrate how reverse engineering is possible thanks to the test scan chains embedded in digital VLSI devices.

Scan insertion is a well-known design-for-test (DFT) technique that allows for the automatic generation of test vectors for production test of a VLSI device. Thanks to its efficiency and ability to achieve high coverage, it has become a de facto standard for testing digital circuits. However, this technique also introduces a security breach. This security breach, usually called a scan side channel, has been investigated by several research groups [16]–[21]. So far, most of the attacks that exploit the scan side channel target cryptographic keys or other secrets held in the device. Recently, an additional threat was reported: the possibility of reverse engineering using the scan side channel [22]–[24]. In this attack mode, the entire device logic can be discovered with the help of the test scan interface.

Azriel *et al.* [22] give a detailed description of how to perform reverse engineering with scan. We summarize it here for completeness. The scan insertion algorithm runs at the design stage and adds to the circuit a special shift mode, which arranges all the internal registers in one or a few shift registers, called scan chains (see Fig. 1), and connects both sides of the chain to the chip interface. During manufacturing, the production tester switches the chip to the shift mode and uses the scan chains both to place the chip in the desired state (*ShiftIn* operation) and to sample its current state (*ShiftOut* operation). The *ShiftIn* and *ShiftOut* operations can be combined with a single functional (*Capture*) cycle to learn (*Probe*) the output of the combinational function F for a given input. The function F aggregates all the combinational logic of the chip. It receives the circuit's primary inputs and register outputs as an input vector, and it returns the primary outputs and register inputs as an output vector. Heuristic algorithms can then be used to find a good approximation of the function F, from which the learner can conjecture the circuit functionality.

While scan-based side channel can be considered as a threat [22], the same method can also serve constructive purposes [25]. For example, if the learner has a reference model of the design, she can compare the learned structure with the model to find discrepancies that may lead to the detection of maliciously implanted or Trojan hardware. Alternatively, the learner can use the reverse engineering for matching with the model to discover IP protection violations.

In this paper, we make the following assumptions.

1) The learner has access to the scan interface of the integrated circuit and is capable of operating it. Security-oriented devices usually contain protection of the test interfaces. In the absence of security concerns, such measures are not common. We discuss the scan access and protection methods more comprehensively in Section II.
2) The method obtains logical functionality of combinational blocks in the circuit. Hence, the target of the IP protection should be visible at this level. Particular implementations of the same combinational function are out of the scope of this paper.
3) The objective of the learner is to find a particular IP within a bigger SoC context. This is different from reverse engineering of the entire SoC. This paper covers only the context of the IP, while methods for isolating the IP from the rest of the SoC are out of the scope of this paper.

In this paper, we present a case study, in which the SHA-256 [26] accelerator implementation details are revealed with the help of a scan-based reverse engineering technique. The case study makes use of Boolean function analysis techniques, in particular junta learning, to build a partial dependence graph of the function, graph clustering algorithms to isolate individual blocks, and property-based sorting to complete the dependence graph. Finally, it uses hypotheses to fully reconstruct the function.

We also propose a new watermark type, the pipelined-associated watermark (PAW), with three unique properties: 1) it changes the logical function, and therefore can be used to detect RTL level theft; 2) it changes the logical function in a nonobvious way, so its removal requires substantial effort; and 3) it is detectable by the scan-based reverse engineering method. The PAW watermark alters logic functions of some pipeline stages and reconstructs the original functionality in the subsequent stages.

In summary, the main contributions of this paper are as follows.

1) Proposal of a new forensic method that exploits embedded test scan chains to detect IP violation.
2) Combining Boolean functional analysis and graph clustering methods to learn combinational logic structures and reconstruct digital circuits by exploiting prior knowledge of the building blocks of the design.
3) Proposal of a new type of watermark (PAW) inserted at the RTL stage that can be detected with the aforementioned forensic method.

The remainder of this paper is organized as follows. Section II discusses assumptions and countermeasures against scan side channel. Section III presents the details of the SHA-256 algorithm implementation. Section IV introduces the learning flow. Section V defines the watermark designed for detection by the scan-based reverse engineering method. Section VI presents the results of the test case evaluation. Finally, Section VII summarizes and discusses directions for future work.

II. Assumptions and Countermeasures

We assume, in this paper, that the scan test interface is present and accessible in the target device. This assumption is reasonable for a typical device that does not target security applications. Vendors of secure VLSI devices often protect their scan interface with authentication, obfuscation, or other mechanisms. This publication may also motivate the IP violators to employ protection to conceal the event of theft. The scan-based reverse engineering method may overcome some protection mechanisms, especially when combined with other methods. The violator may also decide to exclude the entire IP from scan. The fact of exclusion may raise suspicion that the IP has been intentionally stolen. Moreover, leaving large modules disconnected from the scan may lead to unacceptable reduction in production test coverage and therefore product quality.

A number of papers discuss countermeasures against the scan-based side channel analysis. Da Rolt *et al.* [27] present their comprehensive classification, which divides the countermeasures to the following groups: advanced DFT structures, built-in self-test (BIST), secure wrappers, unbounding, scrambling, access restrictions, secret-free test, modified scan chain, and countermeasures against microprobing. In the following, we use Da Rolt's classification to show that some of the countermeasures are ineffective against the scan-based reverse engineering attack. For example, Da Rolt shows that the advanced DFT structure is insecure against scan side channel attack in general. Another countermeasure, the BIST structure resists any scan-based attack, including ours, by disabling external access to the scan chains. For this, BIST should not provide any bypass modes, which complicates debug and field failure diagnostics. In addition, fault coverage of BIST is typically insufficient. Hence, pure BIST solutions are rarely selected by vendors. The scrambling and modified scan chain types present an additional countermeasure, but they are inefficient against the scan-based reverse engineering technique. The reason is that the reverse engineering is still possible in the presence of the aforementioned structures, but the output of the reverse engineering process will include both the functional and the protection circuits. At the next stage, the learner can separate between the two. Finally, the secret-free test solution prevents volatile secret data from being exposed. However, it does not prevent reverse engineering of the design data. Solutions of the types access restriction and secure wrapper prevent unauthorized access to scan chains. Therefore, they are also efficient against reverse engineering. Nevertheless, combined attacks are still possible. These attacks may target the authorization mechanism at the first step, for example using differential power analysis [28]. Often, such mechanisms use global secrets; hence, it is sufficient to break a single unit to gain access to all the other units of the same product.

Previous work usually relies on the circuit's ability to switch dynamically from mission mode to scan mode and back to retrieve runtime information. As a result, one of the popular countermeasures against scan-based attacks is enforcing reset when switching between the modes [21]. The reverse engineering method that we propose retrieves static design information from the scan chains, and, therefore, is immune to this protection mechanism.

III. SHA-256 Algorithm

SHA-2 is a widely used family of cryptographic hash functions. The family comprises six members distinguished by the size of the hash value. In this paper, we examine one member of the family, namely, SHA-256. The SHA-256 algorithm receives a message of an arbitrary length and produces a 256-bit-long digest [Fig. 2(a)]. At the first stage, the original message is padded, which makes its length an integer number of 512-bit chunks. The subsequent processing runs for each chunk sequentially. The processing comprises a message schedule and 64 stages, called compression stages. The message schedule takes the 512-bit input and prepares 64 32-bit words, one for every compression stage. The first 16 words are a copy of the input chunk, and for the remaining 48 words, the schedule operation involves bit permutations, XOR operations, and a four-input 32-bit adder. The compression stage receives an 8×32 bit hash value and produces an input to the next stage, in which six out of the eight words are a mere permutation of the input, and the remaining two words are the result of a five-element and a seven-element adder, respectively. The inputs to the adder are the words from the input of the stage, while some of them pass additional transformations, which include permutations, XOR, selectors, and a majority function.

For the case of IP theft detection, we assume that the exact function of the circuit is known, and the target is to discover the implementation details. Namely, the majority of combinational building blocks are known, and the objective is to learn how they are combined, what the structure of the pipeline is, and what the differences from the original function are, if they exist. Hence, the learning method is built around recognition of the known structures.

The SHA-256 algorithm can be seen as an acyclic data flow graph with many repetitive stages along the way. The implementer can decide on a number of pipeline stages by dividing the stages of the algorithm. Fig. 2(b) shows two stages of the SHA-256 inner loop. If the implementation dedicates one pipeline stage for one compression stage, the combinational logic between the corresponding flip-flops will include six 32-bit pass-through connections, and two 32-bit arithmetic sums: one of seven and the other of five elements. However, if two compression stages of the algorithm comprise one pipeline stage, the combinational logic for one pipeline stage will include four 32-bit pass-through paths, and four 32-bit arithmetic sums: of 5, 7, 11, and 17 elements. Alternatively, if the main constraint is power or silicon real estate, even a single compression stage can be divided, and the same

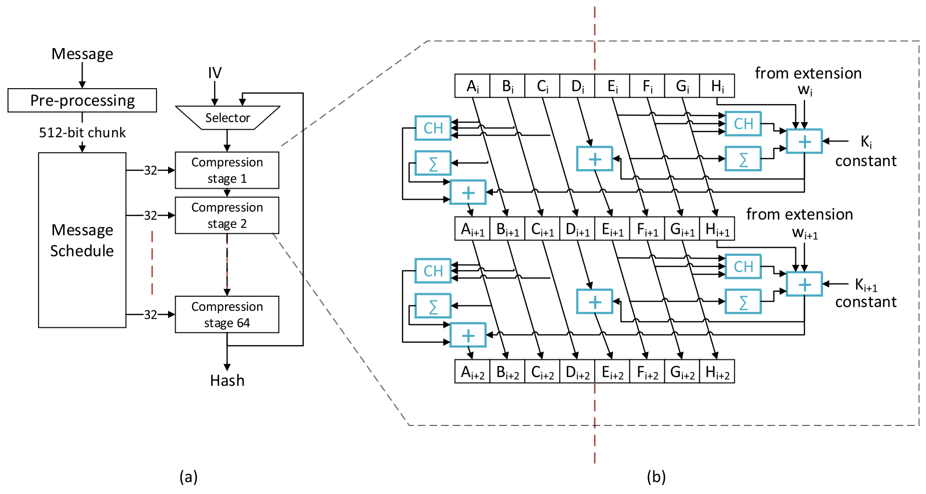


Fig. 2. SHA-256 algorithm block diagram. (a) SHA-256 execution flow, including preprocessing stage, message schedule, which outputs 64 × 32 bit words, and 64 compression stages. (b) Detailed diagram of two 256-bit-wide compression stages. Each includes 6 × 32 pass-through connections, two 32-bit adders, one five-element, and one seven-element. In addition, the compression stage includes permutations, selectors, and majority functions.

Algorithm 1 Probe(Circuit S and Vector v)

```
1: $r \| i := v$            ▷ Set registers and inputs state to $v$
2: $o_{t-1} := o$           ▷ Sample outputs of $S$
3: Capture
4: **return** $r \| o_{t-1}$   ▷ New register values and outputs
```

adder reused several times during calculation of this stage. Performance-hungry applications will use deep pipelines, and latency-oriented designs will strive to combine as many calculations as possible in a single pipeline stage.

Despite the countless configurations, clearly distinguishable structures can be found in most of them. For example, even without knowing the exact configuration, such as the number of inputs or additional logic, multiple bit adder structures have a distinct pattern of dependencies between input and result bits (as we show in detail in Section IV-A). Adders constitute the majority of SHA-256 complex building blocks; therefore, detecting adder-like structures is helpful for both partitioning the data into hierarchical structures and learning the exact function of these blocks.

IV. Learning Flow for IP Theft Detection

Discovering IP theft means detecting patterns in the target design that match elements of the owner's IP. The instrument available to the learner is the operation $Probe(S, v)$ over circuit S, defined in Algorithm 1. In the description of the algorithm, r designates the circuit's internal register vector, i the input vector, and o the circuit's output vector. If we view the circuit as a state machine, then the probe operation receives the current state of the circuit and returns its next state. The constant n designates the length of the register vector, or, in other words, the number of registers in the circuit S. For convenience, we assume that the number of registers is significantly greater than the number of circuit inputs or outputs. A rule of thumb says that the number of pins of an integrated circuit can be approximated as the square root of the number of cells. Thus, for large integrated circuits, the number of pins is relatively negligible. We will, therefore, use the notation n also to indicate the length of concatenated vectors $r \| i$ and $r \| o$.

Obviously, running probes for all possible values of v gives an accurate description of the circuit. Since the number of values is exponential, this method is not practical. Thus, the objective of the learner is to find the minimal set of probes that supply maximum information about the design. The learner possesses *a priori* knowledge about the overall function, and hence about its specific components. For example, an advance encryption standard implementation will have specific SBoxes or a 32-bit CPU will contain a 32-bit adder for address calculation. This knowledge helps to detect the sought elements from a partial set of probes.

The Boolean function analysis field [29] studies algorithms for learning Boolean functions that belong to certain classes. In particular, the junta learning method [30], [31] works for functions with the number of inputs limited by some constant K. This paper introduces an innovative way of applying Boolean function analysis techniques to learn digital circuits with the goal of IP theft detection. We employ the junta method to find the partial dependence graph, which is further processed to identify the required structures. Following are the steps of the learning flow.

1) Find the partial dependence graph using probes and the junta algorithm.
2) Partition the graph using the shared nearest neighbors (SNNs) clustering algorithm.
3) Find missing dependence links with the help of the algorithm $VertexSort$.
4) Reconstruct functions within the clusters and beyond.
5) Return to sequential circuit representation by folding the graph.

A. Creating a Dependence Graph With k-Junta Learning

The probe operation abstracts away the stateful behavior of the circuit and represents it as a combinational multibit

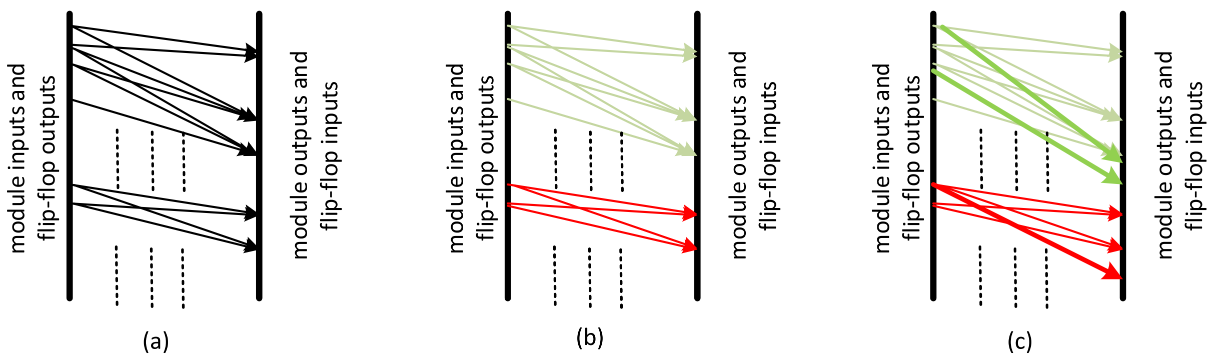

module inputs and flip-flop outputs
module outputs and flip-flop inputs
(a)
(b)
(c)

Fig. 3. Directed bipartite partial dependence graph. (a) After the run of the junta algorithm. (b) After SNN clustering (colors designate clusters). (c) After completing missing dependencies (bold lines indicate newly added edges).

Boolean function F. As shown in Fig. 1, primary inputs and register outputs of the circuit serve as inputs to F, and primary outputs with register inputs of the circuit serve as outputs of F. We can depict this Boolean function as a directed bipartite graph (Fig. 3), where the edges between the nodes designate dependence relations between input and output bits. For an output bit $y_j = f(x_1, x_2, \ldots, x_m)$, input node x_i and output node y_j are connected if and only if there exists an input vector x^0 such that $f(x^0|x_i = 0) \neq f(x^0|x_i = 1)$.

The k-junta learning flow starts with a dependence graph that contains no edges. The input nodes are located at the left side, and the output nodes at the right side of the graph. The algorithm described in the following adds edges to the graph by finding input–output dependence relations of the function.

Junta Algorithm: In computational learning theory, a function f: $\{0, 1\}^n \rightarrow \{0, 1\}$ is called a k-$junta$ for $k \in N$ if it depends on at most k of its input coordinates; that is, $f(x) = g(x_{i_1}, \ldots, x_{i_k})$ for some g: $\{0, 1\}^n \rightarrow \{0, 1\}$ and $i_1, \ldots, i_k \in [n]$ [32]. Hence, algorithms for learning junta functions from queries can be leveraged for reconstructing combinational circuits (or logic cones) with a transitive fan-in bounded by a constant k. We take the adaptive algorithm from [33]. We use the first stage of the algorithm, the stage that finds dependencies. It is formally defined in Algorithm 2. The algorithm runs separately for every output bit. It starts by generating a set of random input vectors. For each input vector in the set, it stores the corresponding probe results of the function F. These results are used to find input bits that affect the output (relevant variables or RV) by comparing input vectors that generate different probe results. For intuition, consider two input vectors υ and $\hat{\upsilon}$ from the random vector set, such that $Probe(S, \upsilon) \neq Probe(S, \hat{\upsilon})$. The vectors υ and $\hat{\upsilon}$ differ in some bits. If we start swapping the discriminating bits in υ one by one, repeating the $Probe$ operation after each swap, there will be a step at which the probe result will change. The variable that caused this change is a relevant variable. This operation is repeated for all the vectors in the set in order to find more relevant variables, while the variables already known as relevant do not swap. In our implementation, we set one of the vectors ($\hat{\upsilon}$) to constant 0 (all the bits except those already proven as relevant) and change only one vector (υ). Namely, the algorithm "walks" from υ to $\hat{\upsilon}$ by setting (swapping)

Algorithm 2 Junta Learning(Circuit S, k)

```
1: init RV[$i$] = $\emptyset$ for all $i$ from 1 to $n$
2: Stage 1: Generate random probes
3: loop $k \cdot 2^k$ times
4:     $v$ := random(1,...,$2^n - 1$)
5:     $P$ := Probe($S$,$v$)
6:     add(Probes, $\langle v,P \rangle$)
7: end loop
8: Stage 2: Find dependencies
9: for $i$ from 1 to $n$ do                    ▷ Each output bit separately
10:     for all $\langle v,P \rangle$ in Probes do
11:         $\hat{v}$ := $\{\hat{v}_1, \ldots, \hat{v}_N\} = v$
12:         For all j from 1 to n: $\hat{v}_j$ = 0 if ($v_j \notin$RV[$i$])
13:         $\hat{P}$ := Probe($S$,$\hat{v}$)
14:         if $P_i \neq \hat{P}_i$ then
15:             Binary search for relevant variable
16:             $pos := n/2$
17:             for $step = n/4$ ; $step \geq 1$ ; $step = step/2$ do
18:                 $u := v$
19:                 Set bits $u_0$ to $u_{pos}$ excluding bits in RV[$i$] to 0
20:                 if Probe$(S,u)_i \neq P_i$ then
21:                     $pos := pos - step$
22:                 else
23:                     $pos := pos + step$
24:                 end if
25:             end for
26:             add(RV[$i$], RV)
27:         end if
28:     end for
29: end for
```

bits in υ to 0. In addition, the variable swapping is done in groups using the binary search principle. During the binary search, the vector υ is recursively halved and one of the halves filled with 0s until a single variable that makes the difference between P_i and $\hat{P}_i$ is found. This results in a logarithmic search time in contrast to the linear time for the one by one swapping example.

To evaluate the time complexity of the algorithm, we count the probe operations. The probes are "expensive," since

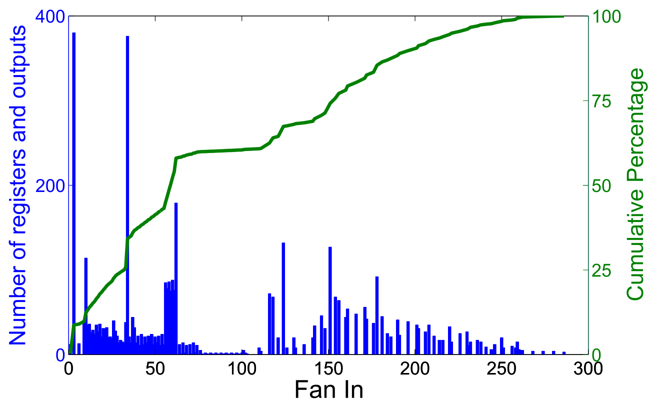


Fig. 4. Transitive fan-in histogram for flip-flop inputs and primary outputs in the ITC'99 benchmark circuits. About 50% of them have fan-in of 50 or less. Peaks can be seen at fan-ins that are multiples of 32, hinting about processor registers and arithmetic circuits.

their runtime depends on n, the size of the register vector. Moreover, the probes run serially. The processing, in contrast, can run in parallel to the probes, and also can be parallelized. Hence, the number of probes determines the algorithm time complexity.

At the first stage, the k-junta algorithm generates $k \cdot 2^k$ probes. At the next stage, the binary search invokes $\log n$ probes once for every relevant variable. Hence, the number of probes required for a single output bit is limited by $(k \cdot 2^k + k \cdot \log n)$. Taking into account that the time complexity of the probe operation itself is $O(n)$, the cumulative time complexity of k-junta with scan for all the outputs can be written as

$$\text{Junta learning time complexity} = O(n \cdot k \cdot (n \cdot \log n + 2^k)). \tag{1}$$

The nonadaptive version of the junta algorithm provides additional speedup by using the set of probes from stage 1 also for the search stage [33]. Hence, it can be used for very large-scale designs, in which the impact of $n \log n$ is dominant. We chose the adaptive version for its simplicity.

Typically, due to the exponential growth, dependence on the transitive fan-in is dominant. Fig. 4 shows a histogram of transitive fan-ins of the logic cones from the circuits in the ITC'99 benchmark set [34]. About half of the nodes have fan-in of 32 and lower, which can still be feasibly handled by the k-junta algorithm. The functionality of these nodes can be found directly by exhaustive search. Clearly, the algorithm will not be able to handle all the higher fan-in nodes within feasible time. Since finding all the dependencies directly with k-junta for high fan-out nodes is not practical, we take a heuristic approach, the key to which is in the *a priori* knowledge of the components of the target circuit. Thus, at the beginning, we choose a value of k that will make the algorithm complete in a feasible time. If this value is lower than the highest transitive fan-in in the circuit, the algorithm will discover only a subset of the dependencies. The chance to discover dependence on a certain variable depends on how strongly this variable affects the output. To formalize this phenomenon, we use the notation of *Influence*, which measures the extent to which certain input affects the function [29]. Namely, the influence of variable x_i on function $f\colon \{0,1\}^n \rightarrow \{0,1\}$ is defined to be a probability that for a random input x, inverting the variable x_i changes the output of the function

$$\mathrm{Inf}_i[f] = \Pr_{x\sim[0,1]^n}(f(x_1,\ldots,x_i,\ldots,x_n) \neq f(x_1,\ldots,\neg x_i,\ldots,x_n)). \tag{2}$$

The influence of the variable determines the probability to find the corresponding link by the k-junta algorithm. The worst case influence of a variable on a function with support size (number of variables affecting the function) of k is $1/2^k$. The k-junta algorithm finds all the dependencies for this function with high probability. For a function with support size greater than k, the k-junta algorithm will find relevant variables with influence of $1/2^k$ or higher with high probability. Consider an example of a 32-bit adder of two elements. The influence of the i'th bit of the result is $1/2^i$, and its support size is $2 \cdot i$. Hence, choosing $k = i$ for k-junta should be sufficient for finding all the relevant variables. There is a practical limitation on the size of the parameter k. Therefore, for higher order bits, only some of the relevant variables will be found. For example, the probability of finding within reasonable run time that bit 0 of the adder input affects bit 31 of its output is very low. The k-junta learning stage thus creates a partial dependence graph, which consists mainly of links with influence of $1/2^k$ and higher [Fig. 3(a)].

B. Partitioning by SNN Clustering

Learning for the purpose of IP theft detection assumes that the building blocks, such as the 32-bit adders for SHA-256, are approximately known. The dependence graph includes subgraphs that represent these building blocks, and the learner's goal is to find the matching function. As the first step, it is essential to isolate subgraphs that include nodes potentially belonging to the same building block (such as a pipeline stage or an arithmetic function). The criterion we use for partitioning the graph is edge density. This is the guiding criterion of certain graph clustering algorithms.

In particular, the adder structure has a distinct dependence pattern, where bit i of the result depends on bits 0 to i of the input operands. The same input bits will also affect all the higher order bits of the result. In practice, the dependence graph received by the k-junta algorithm run reveals only a partial set of dependencies. Since the influence of input operand's bit j on the result bit i decreases exponentially with the distance $i - j$ (see Section IV-A), the majority of edges entering the result bit i in the dependence graph will originate from input bits i to $i - l$, where l is a function of k in k-junta, and it is bounded from below by k divided by the number of operands. Hence, result bits i and $i + 1$ will share on average $min(l, i) \cdot d$ neighbors in the graph,[1] where d is the number of the operands of the adder [Fig. 3(a)]. Thus, the adder structure can be isolated using the SNNs algorithm [35]. The SNN

[1]In the SHA-256 implementation, adders are combined with more functions, and hence, the number of edges will be slightly different.

algorithm assigns any two nodes that have some predefined number of shared neighbors to the same cluster.

Our clustering groups only right-side vertices of the bipartite graph [Fig. 3(b)] according to the following principle: two vertices belong to the same cluster if and only if the number of neighbors they share is greater than the threshold t. The choice of t is important, and may vary for different designs. A value that is too high will cause underfitting, i.e., some of the relevant vertices will not be part of the cluster, while a value that is too low may group in the same cluster loosely connected vertices (for example, vertices that share some global control signals). Different values can be tried for t until a satisfactory partition is found.

The clustering serves two purposes: 1) it allows the hierarchical structure of the circuit to be seen at an early stage, before logical functionality is discovered and 2) it groups together nodes from the same building block, thus enabling a hypothesis to be made on the basis of the projections between different members of the cluster.

C. Completing the Graph With Missing Dependencies

At this stage, we assume that the clustering has successfully isolated individual building blocks, such as adder-like circuits in SHA-256. To complete the picture, we need to reveal the dependencies that the k-junta algorithm failed to discover. Considering the adder example, we take advantage again of its distinctive structure: the transitive fan-in numbers of the adder output bits grow with the position of the bit in the vector. For example, the least significant bit (bit 0) of the adder output depends only on the least significant bits of the operands. Hence, its fan-in is equal to the number of operands. The fan-in of the next bit (bit 1) is equal to twice the number of operands. This arithmetic progression continues until the MSB (see an example in Fig. 7). This allows us to sort the cluster members according to their estimated bit position in the output vector of the adder. The transitive fan-in parameter (equivalent to the number of incoming edges) serves as a classifier.

As discussed in Section IV-B, the number of bits for which the k-junta algorithm finds all the dependencies is limited by l, where l is derived fom k in k-junta. Therefore, only bits 0 to $l-1$ can be sorted directly based on the partial dependence graph. The remaining bits will have an approximately equal number of edges in the graph. Bit l of the output is not guaranteed to have all the dependencies on operand bits 0 to l. However, if we remove the bits of the operands that affect lower bits of the output, the only bits affecting output bit l that will remain are bits l of the operands. Bit $l+1$, for example, depends on bits l and $l+1$ of the operands, bit $l+2$ also on bits $l+2$, and so on. Therefore, output bit l has the smallest number of dependencies after removing operand bits 0 to $l+1$. After recording bit l of the result, we can remove bits l of the operand and proceed to bit $l+1$. This way, we can sort all the bits of the adder output. Algorithm 3 estimates the order of all the variables in the cluster. It does this recursively, at every stage, removing a vertex with the lowest number of edges and removing all the left-side vertices connected to this vertex. The order in which the vertices are removed is the final sorted order.

Algorithm 3 ClusterSort(Cluster {Vertices V, Edges E})

```
1: $I$ = Left side vertices connected to the edges in $E$
2: $\hat{V} = V$
3: $\hat{I} = I$
4: **repeat**
5:     $v_i$ = vertex with lowest number of edges connected to $\hat{I}$
6:     $I_i$ = Left side vertices connected to the edges leading to $v_i$
7:     $\hat{I} = \hat{I}$ - $I_i$
8:     $\hat{V} = \hat{V}$ - $v_i$
9: **until** $\hat{V} = \emptyset$
```

After the sorting, the missing links are added by connecting every vertex to all the left-side vertices connected to lower bits in the sorted list [Fig. 3(c)].

Although the $ClusterSort$ algorithm is explained here in the context of the adder example, it fits many other arithmetic functions with carry propagation, such as multiplier, subtractor, as well as more complex arithmetic and logic unit modules. The same algorithm may also work when an arithmetic function is combined with other logic, as in the case of the SHA-256 compression stage. In general, functions with high transitive fan-in on the one hand and complexity low enough to be implementable on an SoC on the other hand must have some regular pattern. Otherwise, due to the Shannon effect, the function's complexity will be too high to be realizable on an integrated circuit [22]. This regularity can be potentially exploited for ordering within a cluster, similar to Algorithm 3.

D. Function Reconstruction

The next stage after completing the dependence graph for nodes allocated to clusters is finding the logic function for each right-side vertex. Due to the high transitive fan-in of the adder result bits, a brute-force approach with exhaustive search is possible only for a few lower bits. Hence, we start by finding the exact function of the lower bits in the list. The resulting function is then matched against known functions from the building blocks of the original function. If a match is found, a hypothesis will be made for the whole cluster. For example, if the lower bits of the cluster match the lower bits of a seven-element adder, we try to extrapolate this finding to the higher bits of the cluster. The higher bits will be verified for compliance with the hypothesis. For this purpose, all the lower bit dependencies will be assigned values that should affect the higher bits in a certain way. This is done instead of checking all their value combinations. Taking again the adder example, the impact of operand bits 0 to $i-1$ on bit i of the result is expressed in the carry-in value. Therefore, only two assignments of these bits will be made, one yielding a carry-in of 0 and the other yielding 1. The verification of the hypothesis is statistical, based on random queries. Formal proof of the hypothesis is computationally hard. Algorithm 4 shows the required steps for arithmetic functions. The algorithm works on a graph sorted by the previous stage.

Algorithm 4 ClusterRecon(Cluster {Sorted Right-Side Vertices V, Left-Side Vertices I, Edges E})

```
1:  $G$ = Candidate functions from the IP building blocks based on the cluster size and structure with bits sorted by their fan-in
2:  $V_1 : V_l = l$ bottom vertices from $V$
3:  $T(V_1 : V_l)$ = Truth tables of the bits that correspond to $V_l$
4:  for function $g \in G$ do
5:      $g_1 : g_l = l$ bottom bits in $g$
6:      if $T(V_1 : V_l) \equiv g_1 : g_l$ then
7:          $H = g$
8:          break
9:      end if
10: end for
11: for $v_i : i$ from $l+1$ to $|V|$ do
12:     $E_i$ = edges of $v_i-$ edges of $v_1 : v_{i-1}$
13:     $I_i$ = Left side vertices connected to $E_i$
14:     $I_{icarry}$ = Remaining left side vertices connected to edges of $v_i$
15:     Represent bit $H_i$ as $f(I_i, f_{carry}(I_{icarry}))$
16:     for $c$ : all possible values of $f_{icarry}$ do
17:         Find assignment $A_c$ on $I_{icarry}$ s.t. $f_{carry}(A_c) = c$
18:         for $A - i$ : all assignemnts on $I - i$ do
19:             if query on $v_i$ with $A_i, A_c \neq g_i(A_i, A_c)$ then
20:                 go to 4
21:             end if
22:         end for
23:     end for
24: end for
25: return H
```

In addition to the nodes that belong to clusters, there are stand-alone nodes. The majority of these nodes have fan-in small enough so that their function can be learned exactly by exhaustive search. Nodes with high fan-in that have sparse shared connectivity are unlikely (see Section IV-C).

E. Returning to Circuit Representation

The first stage of the learning flow unfolded the circuit to turn it into a Boolean function, which we presented as a bipartite graph, shown in Fig. 3. The last stage performs a reverse transformation by merging back pairs of nodes that correspond to the same register. This effectively gives a sequential circuit representation, similar to the one shown in Fig. 2(b). The resulting picture gives additional information about the structure of the circuit and may give answers about the parts that were not fully understood at the preceding stages. For example, with the circuit representation, the primary inputs of the module can be traced through the pipeline stages in order to understand the data flow.

V. Pipeline-Associated Watermark

The advantage of scan-based learning presented in this paper is in its ability to detect the original IP content. As such, it presents an alternative approach to passive protection solutions, such as watermarks. From the legal side of IP protection,

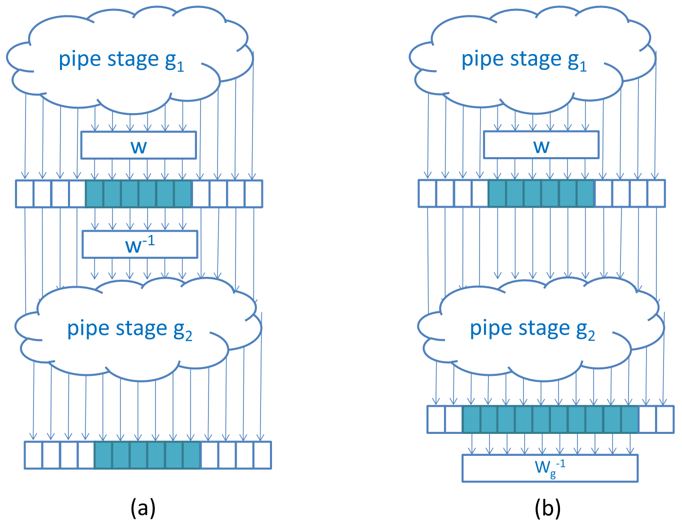


Fig. 5. Embedding the PAW watermark in pipeline stages. (a) First-order watermark comprises the encoder w, which encodes the pipe stage register inputs and the decoder w^{-1}, which is attached to the pipe register output and implements the inverse transformation. (b) Second-order watermark in which the decoder is attached to the next pipeline stage. The decoder performs a more complex transformation that includes the function of the pipeline stage.

the IP owner's objective is to provide convincing evidence of the theft. The violator's defense line could be that of a reasonable coincidence. In this case, a watermark can provide stronger evidence. Fortunately, it is possible to combine the watermarks with the scan-based methods. The IP owner may choose to amend the design with a watermark detectable by scan. Such watermarks must possess the following properties.

1) The watermark must be inserted at a front end stage, such as algorithmic or RTL coding.
2) The watermark must affect a logical function of at least one of the pipeline registers.
3) The watermark must keep the overall function unaltered.

The scan side channel exposes terminal points (inputs and outputs) of combinational logical cones of the integrated circuit. Hence, we look for watermark structures that either add dedicated terminal points or alter a logical function of the existing points. Fan [11] proposes adding dedicated registers to the scan path. These registers contain a direct representation of the watermark value. Hence, the scan side channel can be used for reading out the values. Fan's method uses post-RTL processing to insert the watermark structures and combines them with the regular scan chains inserted by electronic design automation tools. Therefore, the RTL is not protected, which means that an adversary, who possesses the RTL, may bypass the protection mechanism. Moreover, since the special structures are separate from the functional design, it is relatively easy to remove or modify them.

We introduce the pipeline-associated watermark (PAW), a new watermark structure that modifies the logic of individual pipeline stages, but keeps the functionality of the module unaltered. Fig. 5(a) shows a first-order PAW structure. The watermark is applied to a subset of the pipeline stage registers. It comprises of two parts: the encoding transformation w, which applies to the inputs of the registers, and the inverse decoding transformation w^{-1}, which applies to the outputs of the registers. The transformation w has the following properties that ensure correctness and quality.

1) w is invertible, i.e., there exists a transformation w^{-1} such that $\forall x : y = w(x) \rightarrow x = w^{-1}(y)$.
2) w is not trivial so that its identification and removal by an adversary will require substantial effort.
3) Synthesis tools cannot accidentally implement the transformation w.
4) If $y = w(x)$, the number of bits in y (and, therefore, the number of registers after applying the watermark) is greater or equal to the number of bits in x.

Property 2 is rather qualitative. It reflects the quality of the watermark with respect to the ability of the adversary to remove it. The simplest transformation that can be applied is an XOR function with a constant; then, $w = w^{-1}$. However, this watermark is not efficient. Since it is just a selective inversion of individual variables, it can be easily removed. Moreover, some scan insertion tools may use direct or inverted outputs of the registers alternatingly when connecting them to the scan chains; hence, the XOR transformation contradicts Property 3. Also, in this case, user inserted inversions are not detectable. Good candidates for the function w are arithmetic functions. On the one hand, simple arithmetic functions (such as increment by 1) can be selected to minimize the overhead. On the other hand, the arithmetic functions produce complex dependence maps, and thus, their removal from a synthesized gate level netlist is not trivial. However, an adversary, who has access to the RTL representation, even not knowledgeable in the field of the IP, may detect and remove the first-order watermarks thanks to their simple structure.

Higher order watermarks provide further obfuscation by increasing the distance between the inverse transformation w^{-1} and the direct transformation w. In the second-order watermarks, the inverse transformation ${w_g}^{-1}$ connects to the outputs of registers of the next pipeline stage [see Fig. 5(b)]. Higher order watermarks require stronger constraints on the inverse function. The inverse transformation for the second-order PAW watermark is

$$z = g(w(x_w), (x \setminus x_w)) \rightarrow {w_g}^{-1}(z) = g(x). \qquad (3)$$

Here, x is a pipeline stage register, x_w is part of x, on which w is applied, and g is the pipeline stage logical function.

In a third-order PAW watermark, direct and inverse transformations will be distanced by two stages, in a fourth order by three stages, and so on. The higher the order of the watermark, the less trivial the connection between the two transformations. Therefore, removing these watermarks from RTL will require from the adversary deep understanding of the relevant technology.

VI. Test Case: Bitcoin SHA-256 Accelerator

The Bitcoin bookkeeping system requires a heavy mining process [36], which involves numerous SHA-256 hash operations. To make this process energy efficient and economical, specialized hardware was developed. For example, the Bitcoin SHA-256 accelerator design from the OpenCores repository [37] allows for high throughput mining work. To achieve this, the design incorporates deep pipeline, thus reaching a decent size, with more than 80 000 flip-flops. This example presents an interesting test case for testing the capability of the learning flow when dealing with large-scale designs.

A. Experimental Setup

To test the flow, we built a software simulator that models the functionality of the digital circuits under test with the *Probe* operation. The simulator abstracts away the underlying scan protocol that implements the probe (Algorithm 1). The RTL of the target circuit is synthesized using the Synopsys Design Compiler tool. An automatic tool then converts the gate level netlist to a C++ function, which emulates the probe operation by removing all the flip-flops and returning the aggregate combinational logic function. This function receives the flip-flop outputs and primary inputs and returns flip-flop inputs and primary outputs. The function is then plugged into the simulator, which implements the learning algorithms, in particular, k-junta learning and SNN clustering. The platform we used for the simulator is a high performance server with four Intel Xeon E5-2690 four-core processors running at 2.90 GHz. The simulation used 32 threads, each handling one node (function output) at a time. The following stages of the algorithm are performed manually by visually inspecting the results and analyzing their distribution.

B. Results

The Bitcoin SHA-256 accelerator design was synthesized and translated to C++. The partial dependence graph was obtained by a k-junta run with $k = 8$ in the simulator. A higher k value will give higher accuracy (more discovered dependencies); however, the number of required probes will be unacceptable. The subsequent steps of the flow come to compensate for the inaccuracy caused by the insufficient value of k. With the setup outlined in Section VI-A, the k-junta run, the longest step of the flow, takes approximately 2 h.

With a physical device, the first step of the learning process is obtaining access to the scan and counting the number of flip-flops in each chain. The latter can be done by driving some pattern to the scan chain input and counting clock cycles until this pattern appears at the scan chain output. In the simulation environment, this stage is omitted, and we assume that all the flip-flops can be accessed at once. However, the time complexity of the simulated probe operation is comparable to the complexity of the real-life probe operation, that is $O(n)$, where n is the number of registers. Hence, the simulation provides a good indication of the time required to analyze a physical device.

SNN clustering is the next stage of the flow. We tried this stage with different threshold criteria and obtained a cluster distribution histogram for each of them. In the histogram, the clusters were grouped based on their size measured in the number of vertices. Eventually, we selected the threshold that gives the sharpest histogram, which has the smallest number of cluster groups and largest group sizes. This was

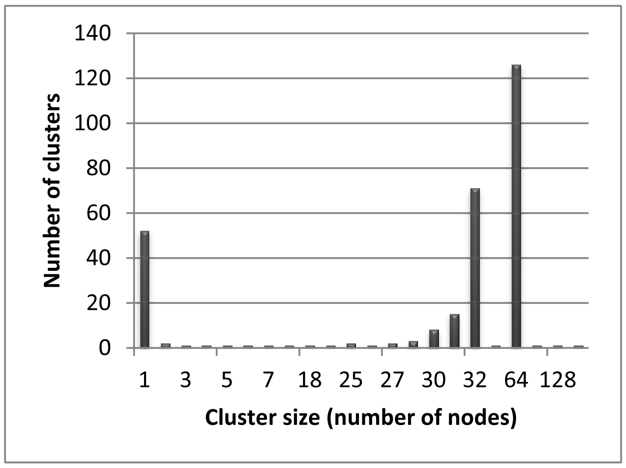


Fig. 6. Histogram of cluster sizes in the Bitcoin SHA-256 accelerator. The biggest group includes clusters with 64 nodes; hence, it matches the compression stages. The second biggest group includes clusters with 32 nodes; hence, it matches the message schedule stages.

achieved with the threshold of 5. The resulting histogram is shown in Fig. 6. There are more than 70 000 vertices with the number of incoming edges (or transitive fan-in) smaller than the threshold. These vertices do not belong to any cluster, and they are not shown in the histogram. A cluster of size 1 contains vertices with transitive fan-in greater than the threshold, which the SNN algorithm did not combine with other vertices. Besides this, two cluster groups stand out: a group of 64-sized clusters containing 126 members, and a group of 32-sized clusters containing 71 members. Having prior knowledge of the function components and the sizes of the clusters, we can hypothesize that the 64-sized clusters correspond to two 32-bit adders and the 32-sized clusters correspond to one 32-bit adder. This implies that: 1) the 64-sized clusters correspond to the compression stage and 2) the 32-sized clusters correspond to the message schedule stage. The number 126 then corresponds to 126 compression stages. A reasonable assumption is that their number is in fact 128, and the remaining two stages have either been split or merged with other vertices due to underfitting or overfitting. The message schedule contains 64 stages, only 48 of which contain adders. Therefore, our hypothesis is that the actual number of adders in the message schedule is 96, which also matches the number of compression stages. To check our hypotheses, we proceed to the next stage—completing missing dependencies.

This stage works separately with every cluster. First, the vertices in the cluster are sorted on the basis of their detected fan-in. Fig. 7 shows the fan-in map of a sample 64-sized cluster. In the same chart, a fan-in map of the original SHA-256 compression stage is shown. For lower fan-in numbers, the detected fan-in curve follows the reference curve, and then saturates at some point. This is the expected behavior for an adder, as explained in Section IV-C. Note that the knee in the curve appears, because two adders (one five-element and the other seven-element) compose the cluster. We then apply the *ClusterSort* algorithm to guess the correct bit order.

We start the function reconstruction in the cluster from the node with the lowest fan-in. For the 64-sized clusters, since we

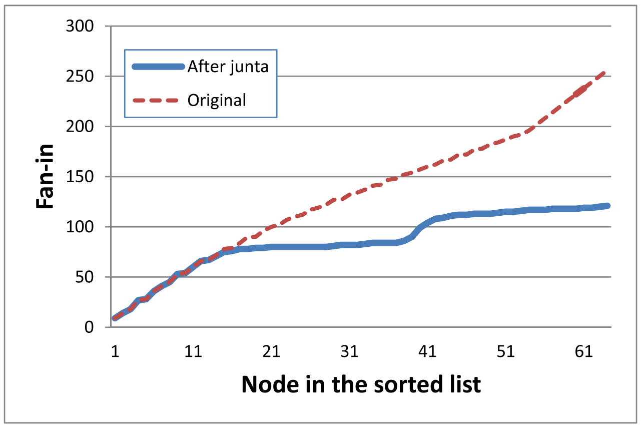


Fig. 7. Sample cluster fan-in map. The nodes in the cluster were sorted by fan-in. The lower curve is the result of dependence finding by junta. The upper curve is the calculated fan-in map from the SHA-256 compression stage.

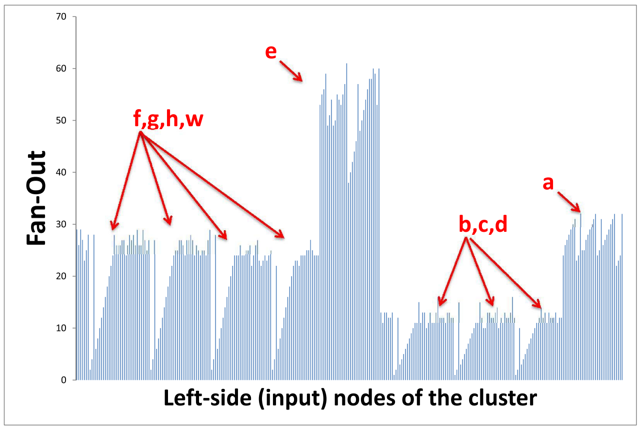


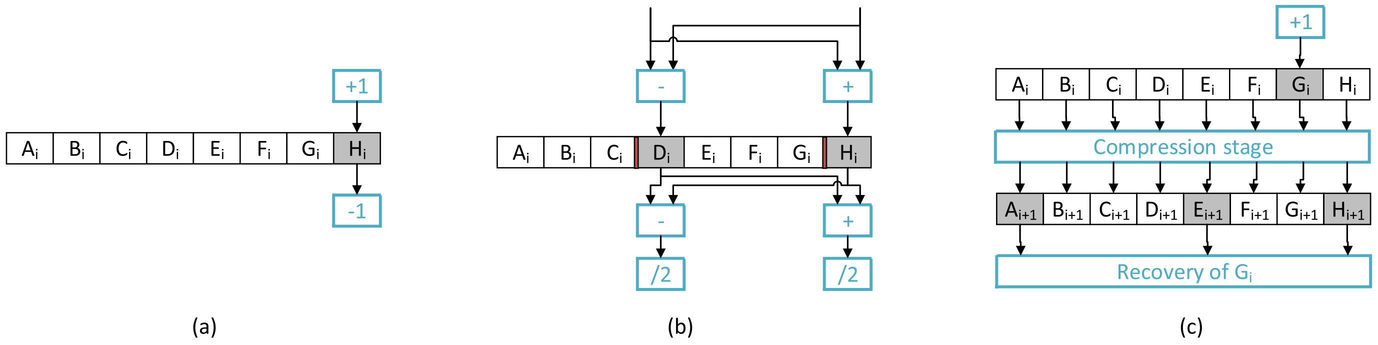
Fig. 8. Histogram of fan-outs for the left-side nodes connecting to the sample cluster of the right-side nodes. Eight groups, corresponding to the eight 32-bit stage input words, can be clearly seen. The fan-out can be used to associate the groups with specific words.

estimate that this cluster contains the logic of the compression stage, we match this bit with bit 0 of the output word e [Fig. 2(b)]. For the compression stage t, bit $e_{0,t}$ is a result of a nine-way XOR function

$$e_{0,t} = \oplus[d_0, h_0, e_6, e_{11}, e_{25}, k_0, w_0, (e_0 \wedge f_0), (\neg e_0 \wedge g_0)]_{t-1}. \quad (4)$$

The fan-in of e_0 is 9, assuming that k_0 is a hardwired constant. This number matches the fan-in of the node with the lowest fan-in in the cluster. Thus, we hypothesized that this node corresponds to bit e_0 and verified the hypothesis. The stage index, and, therefore, the constant k_0, are not known at this stage. Thus, first we checked the value of k_0 by testing the function with a 0 vector. Matching two Boolean functions, though an NP-hard problem, in general, can be done for a small number of variables. Note that the function is invariant to permutations of 6 out of 9 variables. The variables e_0, f_0, and g_0 were identified by measuring influence (2). The influence for these three variables is 1/2, while for all the others it is equal to 1.

To extrapolate to higher bits of the cluster, we reduced the learning problem to one similar to (4) by collapsing all



+1
-1
-
+
/2
Compression stage
Recovery of G_i
A_i B_i C_i D_i E_i F_i G_i H_i
A_{i+1} B_{i+1} C_{i+1} D_{i+1} E_{i+1} F_{i+1} G_{i+1} H_{i+1}
(a)
(b)
(c)

Fig. 9. PAW structures in the SHA-256 implementation. (a) Simple first-order watermark constructed by incrementing word H by 1 followed by subtraction. (b) Compound first-order watermark constructed by linear combination of words D and H (this watermark adds bits in the pipeline stage for the carry-out bit). (c) Second-order watermark constructed by incrementing word H by 1, letting the change propagate through one pipeline stage and removing the effect of the change after that (the change in a single word affects three words in the next stage).

the lower bits of the operands into the carry-in indicator. For example, to learn the function of the bit $e_{t,1}$, we select assignment $\{b_0, d_0, e_0, e_6, e_{11}, e_{25}, f_0, g_0, w_0\} = \{0, 0, 0, 0, 0, 0, 0\}$ to test the bit function with the carry-in of 0 and assignment $\{b_0, d_0, e_0, e_6, e_{11}, e_{25}, f_0, g_0, w_0\} = \{1, 1, 0, 0, 0, 0, 0\}$ to test it with the carry-in of 1. To eliminate contentions between input assignments for the carry-in and assignments for the inputs of the relevant bit, we had to identify the vectors (a and e) that enter into the adder more than once. This was done by checking the fan-out map of the cluster and comparing with the expected fan-outs of the inputs to the compression stage. Fig. 8 shows the fan-out map of all the left-side nodes connected to the nodes in the cluster. Eight groups, suggesting eight 32-bit words, can be clearly seen on the map. The group with the highest fan-out presumably contains the bits from the word e. The group with the second highest fan-out presumably contains the bits of a. Using this iterative process, we were able to reconstruct the entire adder structure.

Finally, after reconstruction of the big structures, we returned to the sequential circuit representation, where the architecture with 32 pipeline stages and two message schedules is identified. All in all, we were able to learn the following details about the given implementation of SHA-256: 1) the module contains two SHA-256 function instances, as follows from the number of stages; 2) the module has a deep pipeline: one pipeline stage per compression stage, which means it is capable of calculating one hash function per cycle; and 3) the pipeline has no flow control, which means that the calculation never stops. Additional details may be extracted in accordance with the objective of the learner.

C. Embedding the PAW Structure

In this experiment, we embedded watermarks according to the PAW methodology, as defined in Section V, into the BitCoin SHA-256 accelerator. The following configurations were tested.

1) Simple first-order watermark [Fig. 9(a)]: Constructed by incrementing the word h (Fig. 2) by 1 before sampling into the pipe registers and subtracting 1 immediately after the sampling.

TABLE I

COMPARISON BETWEEN DIFFERENT TYPES OF PAW WATERMARKS APPLIED TO THE BITCOIN SHA-256 ACCELERATOR

PAW type	Area overhead	Complexity of removal
Simple first-order	3%	easy
Compound first-order	10%	moderate
Second-order	16%	hard

2) Compound first-order watermark [Fig. 9(b)]: Constructed by a linear combination of words d and h and inverse combination immediately after the sampling.
3) Second-order watermark [Fig. 9(c)]: Similar to configuration 1, the word h is incremented by 1 before sampling into the pipe registers. The inverse transformation is placed two stages after. Hence, the inverse transformation must subtract the nonlinear function CH and add it back with the corrected value of h.

All three configurations were synthesized using the Synopsys Design Compiler. Table I shows the area overhead for each of the configurations.

The learning strategy of the PAW watermarks is similar to the strategy for learning the original functionality presented earlier. The building blocks of the original function are amended with the watermark function. For watermark 1, the incrementer circuit is added up to the two existing adders in the compression stage, thus increasing the size of the compression stage cluster to 96. The subtraction by 1 does not affect the cluster distribution, but slightly changes the dependence map within the clusters. In watermark 2, two more adders join the compression stage, making the corresponding cluster grow to the size of 128. Watermark 3, similar to 1, includes the incrementer, which increases the size of the cluster by 32. However, in this case, the recovery function adds much complexity and increases the size of the cluster further by an additional 64 members. In all the cases, the learning stages of dependence graph creation, clustering, and completing the dependence graph are identical. The final stages of the function discovery depend on the exact function searched by the learner. Rejection of watermark presence can be done at earlier stages, for example, by getting a distribution of clusters that does not match the watermark function.

VII. Conclusion and Future Work

This paper introduced a novel method for detecting IP theft. It exploits the embedded test scan chains and combines Boolean function learning methods with graph-based algorithms. The learning algorithm detects structures in the design by taking advantage of prior knowledge of the design components. This prior knowledge allows for accurate reconstruction of the design implementation details, which may supply sufficient evidence of the IP violation event. The comparison is done at the logic level at the boundaries of logic cones between sequential elements. Hence, our method works for both soft and hard IPs. The detectability of the IP theft depends on the ability to observe IP-specific elements at the logic cone boundaries. Notably, since the technique detects inherent elements of the implemented function, it is powerful enough to detect deviations from the original design aimed to obfuscate the theft. The method was built specifically to analyze SHA-256 implementations. With small modifications, the method can be used to analyze other components containing arithmetic functions, and with additional target specific modifications to handle other components containing regular structures.

We demonstrated the power of this approach by using the learning algorithm to reconstruct the design of a Bitcoin SHA-256 accelerator, a module with more than 80 000 internal registers containing complex combinational structures. We were able to obtain the module's internal pipeline structure and locate all the main components of the SHA-256 algorithm implementation.

We introduced a new watermark structure, pipeline-associated watermark (PAW), designed for detection by the scan-based reverse engineering method. Thus, the proposed IP theft detection technique can be combined with watermarks for stronger protection. We believe that these techniques can significantly contribute to protecting IP owners against violation of their rights, by easing the process of violation detection in general and in the Internet of Things device market in particular.

Future research related to this paper can study harnessing the flow for additional applications. One important application that we are exploring is the detection of deviation of the design from the original function, which may indicate the presence of Trojan hardware.

References

[1] M. Pecht and S. Tiku, "Bogus!" *IEEE Spectrum*, vol. 43, no. 5, pp. 37–46, May 2006.

[2] G. Qu and M. Potkonjak, *Intellectual Property Protection in VLSI Designs*. Boston, MA, USA: Kluwer, 2004.

[3] W. P. Griffin, A. Raghunathan, and K. Roy, "CLIP: Circuit level IC protection through direct injection of process variations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 5, pp. 791–803, May 2012.

[4] F. Koushanfar and G. Qu, "Hardware metering," in *Proc. Design Autom. Conf.*, 2001, pp. 490–493.

[5] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 51–63, Feb. 2012.

[6] T. Guneysu, B. Moller, and C. Paar, "New protection mechanisms for intellectual property in reconfigurable logic," in *Proc. 15th Annu. IEEE Symp. Field-Programm. Custom Comput. Mach. (FCCM)*, Apr. 2007, pp. 287–288.

[7] I. Torunoglu and E. Charbon, "Watermarking-based copyright protection of sequential functions," *IEEE J. Solid-State Circuits*, vol. 35, no. 3, pp. 434–440, Mar. 2000.

[8] M. Lewandowski, R. Meana, M. Morrison, and S. Katkoori, "A novel method for watermarking sequential circuits," in *Proc. IEEE Int. Symp. Hardware-Oriented Security Trust*, Jun. 2012, pp. 21–24.

[9] E. Charbon, "Hierarchical watermarking in IC design," in *Proc. IEEE Custom Integr. Circuits Conf.*, May 1998, pp. 295–298.

[10] G. T. Becker, M. Kasper, A. Moradi, and C. Paar, "Side-channel based watermarks for integrated circuits," in *Proc. IEEE Int. Symp. Hardware-Oriented Security Trust (HOST)*, Jun. 2010, pp. 30–35.

[11] Y.-C. Fan, "Testing-based watermarking techniques for intellectual-property identification in SoC design," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 3, pp. 467–479, Mar. 2008.

[12] J. L. Wong, D. Kirovski, and M. Potkonjak, "Computational forensic techniques for intellectual property protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 6, pp. 987–994, Jun. 2004.

[13] S. Guilley, L. Sauvage, J. Micolod, D. Réal, and F. Valette, "Defeating any secret cryptography with SCARE attacks," in *Progress in Cryptology—LATINCRYPT*. Berlin, Germany: Springer, 2010, pp. 273–293.

[14] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Mar. 1997.

[15] X. Wang, S. Narasimhan, A. Krishna, and S. Bhunia, "SCARE: Side-channel analysis based reverse engineering for post-silicon validation," in *Proc. 25th Int. Conf. VLSI Design*, Jan. 2012, pp. 304–309.

[16] D. Hely, K. Rosenfeld, and R. Karri, "Security challenges during VLSI test," in *Proc. IEEE 9th Int. New Circuits Syst. Conf.*, Jun. 2011, pp. 486–489.

[17] J. D. Rolt, G. D. Natale, M.-L. Flottes, and B. Rouzeyre, "New security threats against chips containing scan chain structures," in *Proc. IEEE Int. Symp. Hardware-Oriented Security Trust*, Jun. 2011, p. 110.

[18] J. D. Rolt, G. D. 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.

[19] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *IEEE Trans. Depend. Sec. Comput.*, vol. 4, no. 4, pp. 325–336, Oct. 2007.

[20] A. Das, B. Ege, S. Ghosh, L. Batina, and I. Verbauwhede, "Security analysis of industrial test compression schemes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 12, pp. 1966–1977, Dec. 2013.

[21] D. Hely, F. Bancel, M. L. Flottes, and B. Rouzeyre, "Test control for secure scan designs," in *Proc. Eur. Test Symp. (ETS)*, May 2005, pp. 190–195.

[22] L. Azriel, R. Ginosar, and A. Mendelson, "Exploiting the scan side channel for reverse engineering of a VLSI device," Techn., Israel Inst. Technol., Haifa, Israel, Tech. Rep. CCIT #897, 2016.

[23] L. Azriel, R. Ginosar, and A. Mendelson, "Revealing on-chip proprietary security functions with scan side channel based reverse engineering," in *Proc. 27th Great Lakes Symp. (VLSI)*, 2017, pp. 233–238.

[24] D. G. Saab, V. Nagubadi, F. Kocan, and J. Abraham, "Extraction based verification method for off the shelf integrated circuits," in *Proc. 1st Asia Symp. Quality Electron. Design*, Jul. 2009, pp. 396–400.

[25] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, "Reverse engineering digital circuits using functional analysis," in *Proc. Des., Auto. Test Eur. Conf. Exhibit. (DATE)*, 2013, pp. 1277–1280.

[26] F. Pub, "Secure hash standard," Public Law, Tech. Rep. 100, 1995, p. 235.

[27] J. da Rolt, A. Das, G. Di Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede, "Test versus security: Past and present," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 1, pp. 50–62, Mar. 2014.

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

[29] I. Wegener, *The Complexity of Boolean Functions*. Hoboken, NJ, USA: Wiley, 1987.

[30] P. Damaschke, "On parallel attribute-efficient learning," *J. Comput. Syst. Sci.*, vol. 67, no. 1, pp. 46–62, Aug. 2003.

[31] E. Mossel, R. O'Donnell, and R. P. Servedio, "Learning juntas," in *Proc. 34th ACM Symp. Theory Comput. (STOC)*, New York, New York, USA, Jun. 2003, pp. 206–212.

[32] R. O'Donnell, *Analysis of Boolean Functions*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[33] P. Damaschke, "Adaptive versus nonadaptive attribute-efficient learning," *Mach. Learn.*, vol. 41, no. 2, pp. 197–215, 2000.

[34] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test Comput.*, vol. 17, no. 3, pp. 44–53, Jul./Sep. 2000.

[35] R. A. Jarvis and E. A. Patrick, "Clustering using a similarity measure based on shared near neighbors," *IEEE Trans. Comput.*, vol. C-22, no. 11, pp. 1025–1034, Nov. 1973.

[36] (Mar. 2016). *Developer Guide—Bitcoin*. [Online]. Available: https://bitcoin.org/en/developer-guide#mining

[37] Y. Peng. (Mar. 2016). *Bitcoin Double SHA256 Project*. [Online]. Available: http://opencores.org/project,btc_dsha256

Leonid Azriel (S'17) received the B.Sc. and M.Sc. degrees in electrical engineering from Technion—Israel Institute of Technology, Haifa, Israel, where he is currently pursuing the Ph.D. degree in electrical engineering.

He served in different technical and managerial positions with National Semiconductor, Winbond Electronics, and Nuvoton Technologies companies, Herzlia, Israel, where he was involved in the development of the trusted platform module.

Ran Ginosar (S'79–M'82–SM'08) received the B.Sc. (*summa cum laude*) degree in electrical and computer engineering from Technion—Israel Institute of Technology, Haifa, Israel, in 1978, and the Ph.D. degree in electrical and computer engineering from Princeton University, Princeton, NJ, USA, in 1982.

He was with the AT&T Bell Laboratories from 1982 to 1983. He was a Visiting Associate Professor with The University of Utah, Salt Lake City, UT, USA, from 1989 to 1990, and a Visiting Faculty with the Intel Research Labs from 1997 to 1999. He is currently a Professor of Electrical Engineering and the Head of the VLSI Systems Research Center, Technion—Israel Institute of Technology. He has co-founded several companies in various areas of VLSI systems. His current research interests include VLSI architecture, manycore computers, asynchronous logic and synchronization, networks on chip, and biologic implant chips.

Shay Gueron was a Senior Principal Engineer with Intel, Santa Clara, CA, USA, serving as an Intel's Senior Cryptographer, until 2017. He was one of the Intel Software Guard Extensions technology architects in charge of its cryptographic definition and implementation. He is responsible for some of Intel processors' instructions, such as AESNI, PCLMULQDQ, and coming VPMADD52, and for various microarchitectural features that speed up cryptographic algorithms. He contributed software to open source libraries (OpenSSL, NSS), with significant performance gains for symmetric encryption, public key algorithms, and hashing. He is the Inventor of the Memory Encryption Engine. He is currently an Associate Professor of Mathematics with the University of Haifa, Haifa, Israel. He is also a Senior Principal Engineer with Amazon Web Services, Seattle, WA, USA, serving as a Senior Cryptographer. His current research interests include cryptography, security, and algorithms.

Avi Mendelson (M'10–SM'16) received the Ph.D. degree in computer engineering from the University of Massachusetts at Amherst, Amherst, MA, USA, in 1990.

He has a blend of industrial and academic experience in several different areas, such as computer architecture, operating systems, power management, reliability, and high-performance computing. Among his industrial jobs, he was a Senior Researcher and a Principle Engineer with Intel, Santa Clara, CA, USA, for 11 years. Among his achievements at Intel, he was the Chief Architect of the CMP (multicore-on-chip) feature of the first dual core processors Intel developed. He is currently a Professor with Technion—Israel Institute of Technology, Haifa, Israel. He has authored or co-authored over 130 papers in refereed journals, conferences, and workshops.

Dr. Mendelson served as a Program Chair of different major conferences and as the General Chair of the International Symposium on Computer Architecture in 2013. He completed a full term as an Associate Editor of the IEEE COMPUTER ARCHITECTURE LETTERS. He is currently serving as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS.