Using Scan Side Channel for Detecting IP Theft

Leonid Azriel
Technion - Israel Institute of Technology
Haifa 32000 ISRAEL
leonida@tx.technion.ac.il

Ran Ginosar
Technion - Israel Institute of Technology
Haifa 32000 ISRAEL
ran@ee.technion.ac.il

Shay Gueron
University of Haifa, Israel
and
Intel Corporation
Intel Development Center,
Haifa, Israel
shay@math.haifa.ac.il

Avi Mendelson
Technion - Israel Institute of Technology
Haifa 32000 ISRAEL
avi.mendelson@technion.ac.il

ABSTRACT

We present a process for detection of IP theft in VLSI devices that exploits the internal test scan chains. The IP owner learns implementation details in the suspect device to find evidence of the theft, while the top level function is public. The scan chains supply direct access to the internal registers in the device, thus making it possible to learn the logic functions of the internal combinational logic chunks. Our work introduces an innovative way of applying Boolean function analysis techniques for learning digital circuits with the goal of IP theft detection. By using Boolean function learning methods, the learner creates a partial dependency graph of the internal flip-flops. The graph is further partitioned using the SNN 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 or refute that 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.

1. 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]. Vast research has been devoted to finding an efficient method for IP protection. One of the suggested approaches is the watermarking technique, where additional data is embedded into the design in a way that its removal is difficult. Different types have been proposed, such as constraint-based watermarking or watermarking state machines [2, 3]. The presence of this watermark in the target design may serve as a proof of theft. An additional technique for fighting IP theft is the IC metering technique, in which every IC instance is uniquely marked [4, 5]. The aforementioned methods require notable effort at the design stage. In addition, even in the presence of the watermarks or similar structures, their detection remains a challenge [6] . The power of Intellectual Property is measured by its contribution as well by the ability to legally protect it. We propose a method for detection of IP theft that is efficient both in the presence and absence of the special structures. The proposed method is based on full reverse engineering that enables extracting special structures or patented implementation details from the target design. Side Channel Analysis for Reverse Engineering (SCARE) has been discussed in several publications [7, 8, 9], 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 DFT (Design-For-Test) 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 [10, 11, 12, 13, 14, 15]. 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 [16, 17]. In this attack mode, the entire device logic can be discovered with the help of the test scan

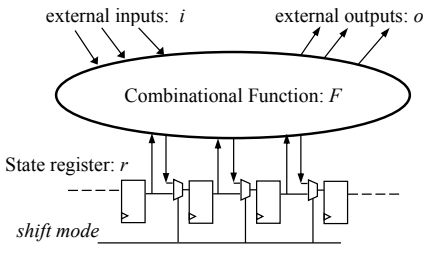

Figure 1: Scan Design

HASP 2016, June 18 2016, ,

DOI: http://dx.doi.org/10.1145/2948618.2948619

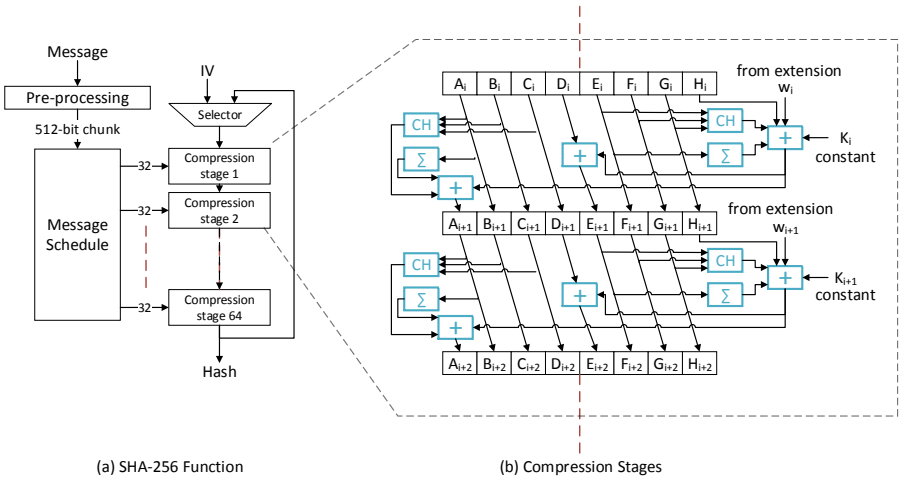

Figure 2: SHA-256 algorithm block diagram. (a) SHA-256 execution flow, including preprocessing stage, message schedule, which outputs 64x32-bit words, and 64 compression stages. (b) Detailed diagram of two 256-bit wide compression stages. Each includes 6x32 pass-through connections, two 32-bit adders, one 5-element and one 7-element. In addition, the compression stage includes permutations, selectors and majority functions.

interface.

In [16], Azriel et al. 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 Figure 1), and connects both sides of the chain to the chip interface. During manufacturing, the production tester may switch the chip to the shift mode and use 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 circuit functionality can be conjectured.

The reverse engineering is presented in [16] as a threat. However, the same method can serve constructive purposes [18]. 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 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 present a case study in which the SHA-256 [19] accelerator implementation details are revealed with the help of a scan-based reverse engineering technique.

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, as detailed in [16]. The violator may also decide to exclude the entire IP from scan. The fact of exclusion, in addition to being a quality issue, may raise suspicion that the IP has been intentionally stolen. Modern designs employ advanced DFT techniques such as scan compression to save test resources. Scan compression may add complexity to the learning. Nevertheless, it does not fully prevent it [11, 14].

The remainder of this paper is organized as follows. Section 2 presents the details of the SHA-256 algorithm implementation. Section 3 introduces the learning flow. Section 4 presents the results of the test case evaluation. Finally, Section 5 summarizes and discusses directions for future work.

2. SHA-256 ALGORITHM

SHA-2 is a widely used family of cryptographic hash functions. The family comprises 6 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 (Figure 2a). 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 4-input 32-bit adder. The compression stage receives an 8 by 32-bit hash value and produces an input to the next stage, in which 6 out of the 8 words are a mere permutation of the input, and the remaining 2 words are the result of a 5-element and a 7-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

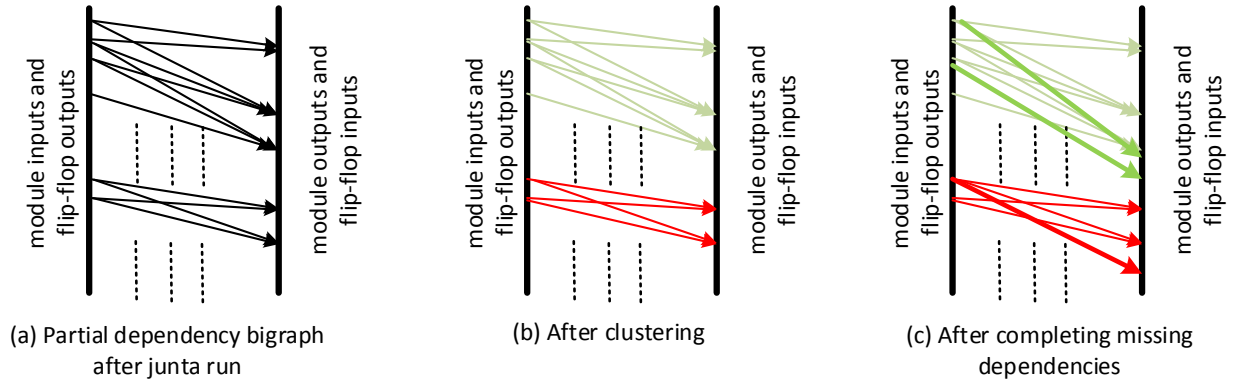

Figure 3: The directed bipartite partial dependency graph. (a) After the run of the junta algorithm. (b) After SNN clustering (colors designate clusters). (c) After completing missing dependencies (bold lines indicate newly added edges).

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 the 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 such 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 implementor can decide on a number of pipeline stages by dividing the stages of the algorithm. Figure 2b 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 adder reused several times during calculation of this stage. Performance-hungry applications will use deep pipeline, 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 3.1). Adders constitute the majority of SHA-256 complex building blocks; therefore, detecting adder-like structures is helpful both for partitioning the data into hierarchical structures and for learning the exact function of these blocks.

3. LEARNING FLOW

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. Here, 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. 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 design components. The Boolean function analysis field [20] studies algorithms for learning Boolean functions that belong to certain classes. In particular, the junta learning method [21, 22] works for functions with the number of inputs limited by some constant K. Our work 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 dependency graph, which is further processed to identify the required structures. Following are the steps of the learning flow:

Algorithm 1 Probe(Circuit S, vector v)

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

1: Find the partial dependency graph using probes and the junta algorithm
2: Partition the graph using the shared nearest neighbors (*SNN*) clustering algorithm
3: Find missing dependency 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

3.1 Creating a Dependency Graph with K-Junta Learning

The probe operation abstracts away the stateful behavior of the circuit and represents it as a combinational Boolean function, where

primary inputs and register outputs of the circuit serve as inputs to the function, and primary outputs with register inputs of the circuit serve as outputs of the function. We can depict this Boolean function as a directed bipartite graph (Figure 3), where the edges between the nodes designate dependency relations. For a Boolean function $y_j = f(x_1, x_2, \cdots, x_m)$, input node x_i and output node y_j are connected if and only if there exists an input vector x^0 such that $f(x^0|x_i = 0) \neq f(x^0|x_i = 1)$. At the beginning of the learning flow, the dependency graph contains no edges. The input nodes are located at the left side, and the output nodes at the right side of the graph. The k-junta algorithm described below finds a subset of dependency relations within the function. More details about the use of the k-junta algorithm for reverse engineering of a digital circuit can be found in [16].

Junta algorithm: In computational learning theory, a function f: $\{0,1\}^n \rightarrow \{0,1\}$ is called a *k-junta* for $k \in x$ if it depends on at most k of its input coordinates; i.e., $f(x) = g(x_{i_1}, \cdots, x_{i_k})$ for some g: $\{0,1\}^n \rightarrow \{0,1\}$ and $i_1, \ldots, i_k \in [n]$ [23]. 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 [24]. We use the first stage of the algorithm, the stage that finds dependencies. For this, a set of probes with random inputs is prepared. The results of the probes are used to find input bits that affect the output (relevant variables or RV) with a binary search-alike method. This process runs for every output bit separately.

Algorithm 2 Junta Learning(Circuit S, k)

```
 1: init RV[i] = ∅ for all i from 1 to n+b
 2: repeat
 3:     v := random(1,...,2^N-1)
 4:     P := Probe(S,v)
 5:     add(Probes, ⟨v,P⟩)
 6: until done k·2^k times
 7: for i from 1 to n+b do                    ▷ Repeat for every output bit
 8:     for all ⟨v,P⟩ in Probes do
 9:         v̂ := {v̂_1,...,v̂_N}|v̂_j = (v_j ∈RVs[i]) ? v_j : 0
10:         P̂ := Probe(S,v̂)
11:         if P_i ≠ P̂_i then
12:             find next RV by binary search on v keeping all v_j ∈
    RV[i] fixed[1]
13:             add(RV[i], RV)
14:         end if
15:     end for
16: end for
```

The k-junta algorithm time complexity is $O(log(n) \cdot k \cdot 2^k)$, when measuring it in the number of probes. Taking into account that the time complexity of the probe operation itself is $O(n)$, the cumulative time complexity of k-junta with scan is $O(n \cdot log(n) \cdot k \cdot 2^k)$. If junta learning runs with k lower than the bound of the circuit's transitive fanout, it will discover only some of the dependencies. We use the notation of $Influence$, which measures the extent to which certain input affects the function [20]. Namely, the influence of variable x_i on function f: $\{0,1\}^n \rightarrow \{0,1\}$ is defined to be a probability that for a random input x, inverting the variable x_i changes the output of the function:

$$Inf_i[f] = \underset{x \sim [0,1]^n}{Pr}[f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, \neg x_i, \ldots, x_n)] \quad (1)$$

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 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 dependency graph, which consists mainly of links with influence of $1/2^k$ and higher (Figure 3a).

3.2 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 dependency 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 dependency 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 dependency 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 3.1), the majority of the edges entering the result bit i in the dependency graph will originate from input bits i to $i - l$, where l is a function of k in k-junta. Hence, result bits i and $i + 1$ will share on average $min(l, i) \cdot d$ neighbors in the graph[2], where d is the number of operands of the adder (Figure 3a). Thus, the adder structure can be isolated using the shared nearest neighbors (SNN) algorithm [25].

Our clustering groups only right-side vertices of the bipartite graph (Figure 3b) 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 under-fitting, 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.

3.3 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. For this purpose, we

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

take advantage again of the distinctive structure of the adder. At the first stage, we sort the cluster members according to their estimated bit position in the result vector of the adder. The transitive fan-in parameter (equivalent to the number of incoming edges) can be used as a classifier. The lowest l bits can be sorted based on the the partial dependency graph. The remaining bits will have an approximately equal number of edges due to the limitation of k-junta. The following algorithm 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 (Figure 3c).

3.4 Function Reconstruction

After the dependency graph has been completed, the next stage is to find 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 7-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.

In addition to the nodes that belong to clusters, there are standalone nodes. We assume that 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.

3.5 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 Figure 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, for example as shown in Figure 2b. 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.

4. TEST CASE: BITCOIN SHA-256 ACCELERATOR

The Bitcoin bookkeeping system requires a heavy mining process [26], 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 [27] 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.

4.1 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. 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 4- core processors running at 2.90GHz. 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.

4.2 Results

The Bitcoin SHA-256 accelerator design was synthesized and translated to C++. The partial dependency 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 4.1, the k-junta run, the longest step of the flow, takes approximately two hours.

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

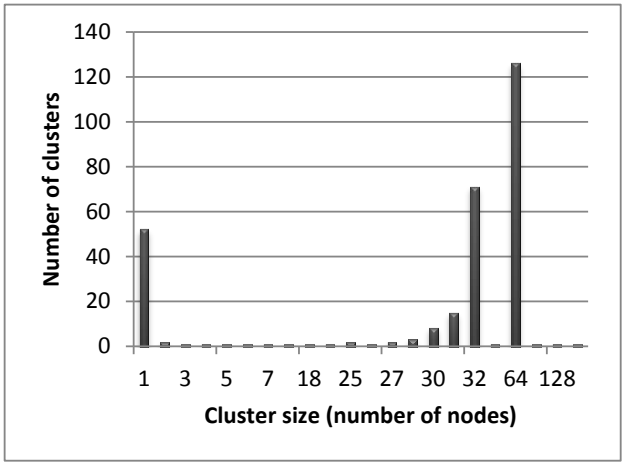

Figure 4: Histogram of cluster sizes in the Bitcoin SHA-256 accelerator. The biggest group includes clusters with 64 nodes, hence it matches the compression stages. The second biggest group includes clusters with 32 nodes, hence it matches the message schedule stages.

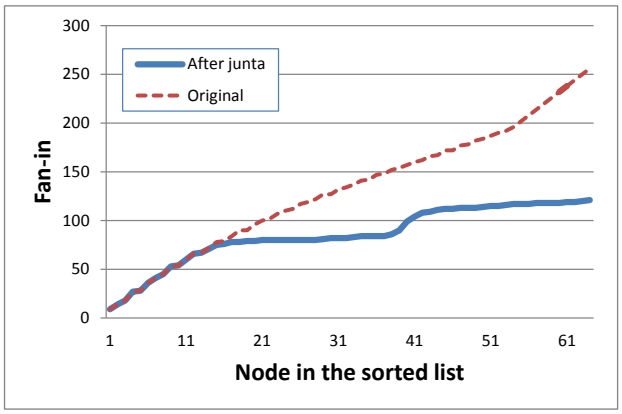

Figure 5: Sample cluster fan-in map. The nodes in the cluster were sorted by fan-in. The lower curve is the result of dependency finding by junta. The upper curve is the calculated fan-in map from the SHA-256 compression stage.

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 achieved with the threshold of five. The resulting histogram is shown in Figure 4. 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 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) that 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 under- 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. Figure 5 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 3.3. Note that the knee in the curve appears because two adders (one 5-element and the other 7-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 estimate that this cluster contains the logic of the compression stage, we match this bit with bit 0 of the output word e (Figure 2b). For the compression stage $i+1$, bit $e_{0,i+1}$ is a result of a 9-way XOR function:

$$e_{0,i+1} = \oplus[d_0, h_0, e_6, e_{11}, e_{25}, k_0, w_0, (e_0 \wedge f_0), (\neg e_0 \wedge g_0)]_i \quad (2)$$

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, g_0 were identified by measuring influence (1). 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 (2) by collapsing all the lower bits of the operands into the carry-in indicator. 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 to the expected fan-outs of the inputs to the compression stage. Figure 6 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 fanout presumably contains the bits from the word e. The group with the second highest fanout 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

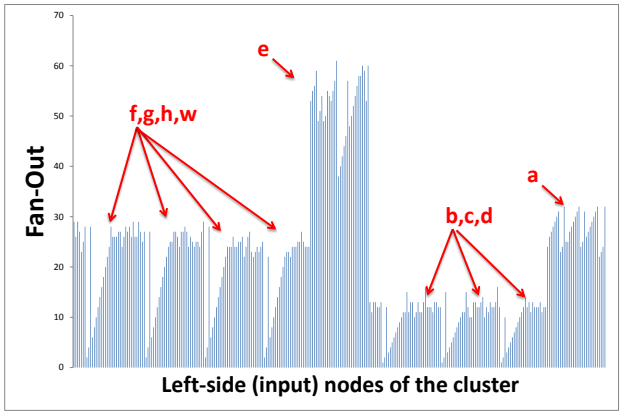

Figure 6: Histogram of fan-outs for the left-side nodes connecting to the sample cluster of the right-side nodes. Eight groups, corresponding to the eight 32-bit stage input words, can be clearly seen. The fan-out can be used to associate the groups with specific words.

module has a deep pipeline: one pipeline stage per compression stage, which means it is capable of calculating one hash function per cycle; (3) the pipeline has no flow control, which means the calculation never stops. Additional details may be extracted in accordance with the objective of the learner.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a novel method for detecting IP theft. We exploit the embedded test scan chains and combine 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 highly 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 soft and hard IP. The detectability of the IP theft depends on the ability to observe IP-specific elements at the logic cone boundaries.

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.

The scan side channel is known mostly as a security threat. Unlike the case of security-oriented VLSI devices, where the designers may exclude sensitive parts from the scan, in the case of IP theft, any attempt to hide a circuit may indicate that the IP is intentionally stolen, a more severe violation of law. Moreover, leaving large modules disconnected from the scan may lead to unacceptable reduction in production test coverage and therefore product quality.

We are working to extend this work in a number of directions. We are examining the flow with more benchmarks of designs with different structure. We are also studying harnessing the proposed detection method for detection of IP protection watermarks in physical devices. An additional application of the algorithm that we are exploring is detection of deviation of the design from the original function, which may indicate the presence of Trojan hardware.

6. ACKNOWLEDGMENTS

We thank the anonymous referees for their valuable comments and suggestions. We also thank Prof. Nader Bshouty from the Technion Computer Science Department for his invaluable advice. This work was partially supported by the Cyber Security Center at the Technion. L. Azriel was supported by the Hasso Plattner Institute (HPI). Prof. A. Mendelson was partially supported by a research grant from Kinneret College, Israel. S. Gueron was partially supported by the PQCRYPTO project, which was partially funded by the European Commission Horizon 2020 research Programme, grant #645622, and by the Blavatnik Interdisciplinary Cyber Research Center (ICRC) at Tel Aviv University.

7. REFERENCES

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

[2] T. Guneysu, B. Moller, and C. Paar, "New Protection Mechanisms for Intellectual Property in Reconfigurable Logic," in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*. IEEE, Apr 2007, pp. 287–288.

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

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

[5] F. Koushanfar, "Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, Feb 2012.

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

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

[8] T. M. Mitchell, "Machine Learning," Mar 1997.

[9] X. Wang, S. Narasimhan, A. Krishna, and S. Bhunia, "SCARE: Side-Channel Analysis Based Reverse Engineering for Post-Silicon Validation," in *2012 25th International Conference on VLSI Design*. IEEE, Jan 2012, pp. 304–309.

[10] D. Hely, K. Rosenfeld, and R. Karri, "Security challenges during VLSI test," in *IEEE 9th International New Circuits and Systems Conference*. IEEE, Jun 2011, pp. 486–489.

[11] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "New security threats against chips containing scan chain structures," *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 110–110, Jun 2011.

[12] J. D. Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Thwarting Scan-Based Attacks on Secure-ICs With On-Chip Comparison," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 947–951, Apr 2014.

[13] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing Designs against Scan-Based Side-Channel

Attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 325–336, Oct 2007.

[14] A. Das, B. Ege, S. Ghosh, L. Batina, and I. Verbauwhede, "Security Analysis of Industrial Test Compression Schemes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1966–1977, Dec 2013.

[15] D. Hely, F. Bancel, M. Flottes, and B. Rouzeyre, "Test Control for Secure Scan Designs," *European Test Symposium (ETS'05)*, pp. 190–195, 2005.

[16] L. Azriel, R. Ginosar, and A. Mendelson, "Exploiting the Scan Side Channel for Reverse Engineering of a VLSI Device," Technion, Israel Institute of Technology, Tech. Rep. CCIT Report # 897, May 2016.

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

[18] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, "Reverse Engineering Digital Circuits Using Functional Analysis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. New Jersey: IEEE Conference Publications, 2013, pp. 1277–1280.

[19] 'FIPS', "180-4," *Federal Information Processing Standards Publication, Secure Hash*, 2012.

[20] I. Wegener, *The Complexity of Boolean Functions*. John Wiley & Sons, Inc., 1987.

[21] P. Damaschke, "On parallel attribute-efficient learning," *Journal of Computer and System Sciences*, vol. 67, no. 1, pp. 46–62, Aug 2003.

[22] E. Mossel, R. O'Donnell, and R. P. Servedio, "Learning juntas," in *Proceedings of the Thirty-fifth ACM Symposium on Theory of Computing - STOC '03*. New York, New York, USA: ACM Press, Jun 2003, p. 206.

[23] R. O'Donnell, *Analysis of Boolean functions*. Cambridge University Press, 2014.

[24] P. Damaschke, "Adaptive Versus Nonadaptive Attribute-Efficient Learning," *Machine Learning*, vol. 41, no. 2, pp. 197–215, 2000.

[25] R. Jarvis and E. Patrick, "Clustering Using a Similarity Measure Based on Shared Near Neighbors," *IEEE Transactions on Computers*, vol. C-22, no. 11, pp. 1025–1034, Nov 1973.

[26] "Developer Guide - Bitcoin." [Online]. Available: https://bitcoin.org/en/developer-guide#mining

[27] Y. Peng, "Bitcoin Double SHA256 project." [Online]. Available: http://opencores.com/project,btc_dsha256