Exploiting the Scan Side Channel for Reverse Engineering of a VLSI Device

Leonid Azriel, Ran Ginosar, and Avi Mendelson

CCIT Report #897
May 2016

CCIT Report #897 May 2016

Exploiting the Scan Side Channel for Reverse Engineering of a VLSI Device

Leonid Azriel, Ran Ginosar, and Avi Mendelson

Technion, Israel Institute of Technology,
{leonida, ran, avi.mendelson}@technion.ac.il

Abstract. This paper presents a novel non-invasive method of reverse engineering of digital VLSI devices that exploits the scan chains originally inserted into the device for production test automation. The scan chains unfold the sequential logic of the device to form a combinational function. The device's logical functionality can then be discovered by examining this function. This potentially allows for the adversary to carry out a reverse engineering attack using simple off-the-shelf equipment for accessing the scan chains combined with Boolean function learning methods. To demonstrate the effectiveness of the method, we apply a set of heuristic learning algorithms that take advantage of common properties of digital circuits, in particular limited transitive fan-in of combinational logic and sub-circuit sharing properties. With these algorithms we achieve successful and fast reconstruction of popular cryptographic function implementations such as the AES cryptographic accelerator. The algorithm used for reconstruction of the AES is scalable and therefore can be used with significantly larger circuits. Finally, we discuss the existing countermeasures against scan-based side channel attacks and find that the presented method is immune to some of them.

Keywords: Side Channel Analysis, Scan Side Channel, Reverse Engineering

1 Introduction

Reverse engineering of a VLSI device is a complex task that traditionally requires tedious work and expensive equipment [25]. The ultimate goal of the reverse engineering process is, given the physical device, to discover its underlying algorithm; i.e., the device's behavioral definition. Roughly, we can represent the discovery task as a two-stage process: (1) Extraction of the circuit from the physical device and (2) Extraction of the behavioral model from the circuit.

The boundary between the two stages sometimes may be blurred; nevertheless, these are usually two distinct tasks. The first stage, as a rule, involves a sequence of invasive techniques, such as removing the package, performing cross-section, delayering, and imaging of nanoscale [18, 25]. The second stage is usually algorithmic [1, 14, 16, 22]. This paper addresses the first stage - circuit extraction. The complexity and cost of invasive circuit extraction methods commonly used today rise with the advancement in semiconductor manufacturing technology. We propose a new, non-invasive approach for circuit extraction that exploits the internal scan chains. This approach provides notable accuracy, while using simple and inexpensive equipment. In contrast to the dynamic data oriented attacks. this method reveals the logical function of the circuit.

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, supported by all the major synthesis tools. Furthermore, scan insertion is usually enforced by ASIC vendors. The high controllability and observability that the scan provides contribute to high fault coverage, but make a significant drawback when security is a concern. Scan insertion provides a side channel, which, unless properly protected, may become an easy target for the attacker as was already shown by recent research [5, 8–10, 13, 20]. Previous work primarily focuses on the vulnerability of the registers that participate in the scan chain. Easy access to these registers may reveal confidential information, such as cryptographic keys, machine state, etc. This paper presents a different attack that reveals functionality of the design itself. This security breach broadens the scope of possible exploits of the scan mode from security devices to any device that may contain trade secrets or proprietary algorithms. The proposed reverse engineering method can also serve benign purposes, such as competitive analysis, IP theft detection, or discovery of malicious hardware implanted by a third party during the physical implementation process.

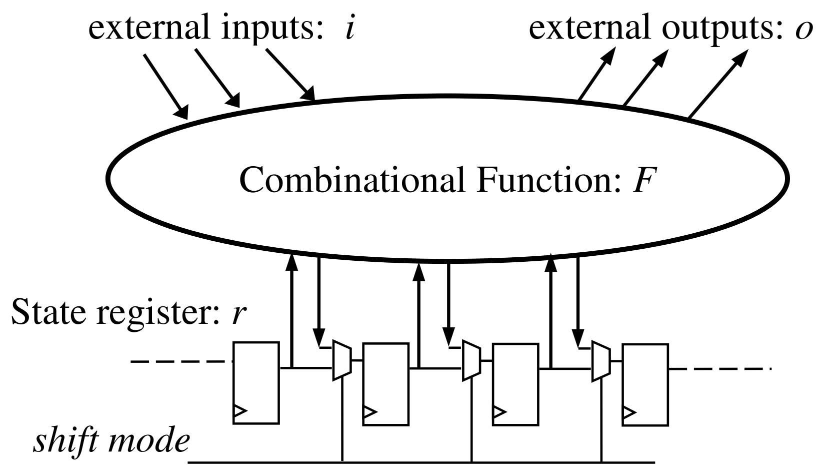


Fig. 1. Scan Design

The scan insertion algorithm adds to the circuit a special *shiftmode*, which arranges all the internal registers in one or several shift registers, called scan chains, see Fig.1. Hence, the external tester may place the circuit in the desired state (*ShiftIn* operation) and sample the current state (*ShiftOut* operation) of the circuit. One can combine the *ShiftIn* and *ShiftOut* operations with a single functional (*Capture*) cycle to learn (*Probe*) the output of the combinational function F for a given input. An exhaustive search over all possible states of the device's registers and input pins then reveals a truth table that fully describes the function F.

The exhaustive search, due to its exponential complexity, is practical only for very small devices, containing no more than a few dozens of registers. For a general case of learning a Boolean function, the number of possible functions is 2^{2^n}, where n is the number of inputs of the function . However, the search space in our case is much smaller as shown in Theorem 1:

Theorem 1. *The number of n-input Boolean functions realizable by a digital circuit is significantly smaller than the number of all possible n-input Boolean functions, if n is large enough.*

Proof. The thorem is a direct corollary from the Shannon Effect [15], which states that almost all Boolean functions have a complexity close to the maximum possible for the uniform probability distribution. Namely, an arbitrary function with n variables will have asymptotic complexity of $O(2^n)$ with probability close to 1. The complexity of a Boolean function is equivalent to circuit complexity (the number of gates) [26]. Clearly, for large n, a circuit of size $O(2^n)$ is not realizable. Hence, the conclusion is that for large n, almost all Boolean functions are not realizable by a digital circuit. $\square$

Devising an upper bound for the number of n-input Boolean functions with some limited complexity is an open question. If such a bound exists, it may define the size of the search space for our problem. An additional unanswered question is about the properties of all the functions that satisfy the complexity bound. Until these question are answered, we take a heuristic approach. In this paper, we assume properties of the functions to be learned and propose learning algorithms that exploit these properties for reduction of complexity. In a sense, our work correlates to the field of Computational Learning theory, which studies learning algorithms for functions with certain properties, such as linear functions, decision trees, DNF, junta functions and more [19]. We use some of the algorithms developed in this field, in particular algorithms for learning juntas. To summarize, the contributions of this paper are:

1. Finding of a vulnerability: exploitation of the test scan logic for reverse engineering of a digital VLSI device
2. Proposal of a heuristic-based algorithm for learning of a Boolean function realizable by a digital circuit
3. Demonstration of full reconstruction of an AES engine via scan side channel with a junta learning algorithm

The remainder of this paper is organized as follows. Section 2 introduces formal definitions of the circuit and supplies definitions of the learning algorithms with proofs and examples. Section 3 shows the experimental results obtained from simulation. Section 4 discusses related work including known countermeasures. Finally, Section 5 concludes and suggests directions for future work.

2 Algorithms for Learning a Boolean Function with Probes

The scan based attack turns the problem of reverse engineering of a circuit to the problem of learning a stateless Boolean function. This section describes the algorithms for learning the circuit's combinational function F exposed by the scan. We start with a formal definition of the circuit (see Fig.1) and a few notations.

Definition 1. *Let S be a digital circuit comprising a vector of inputs $i = (i_1, \dots, i_a) \in \{0,1\}^a$, a vector of outputs $o = (o_1, \dots, o_b) \in \{0,1\}^b$, a state register $r = (r_1, \dots, r_n) \in \{0,1\}^n$, a clock input $ck \in \{0,1\}$ and a collection of combinational gates that implement the next state and output function F such that $(r \parallel o)_{next\_ck\_cycle} = F(r \parallel i)$.*

Notations:

literal a variable with or without inversion. A literal is negative if it is inverted and is positive otherwise

term product of literals

implicant of function F: a term, which implies F=1

relevant or irredundant literal with respect to a function F is a literal that corresponds to a variable that affects the value of F. Otherwise, the literal is **redundant**

cover: term $\acute{m}$ is a cover of term m if m can be obtained by removing literals from $\acute{m}$

mapping: A binary vector v maps to a term m by mapping bits in v equal to 1 to positive literals and bits equal to 0 to negative literals in m

The scan insertion algorithm arranges the bits of the state register r in c scan chains accessible from the circuit interface. An additional *shiftmode* input signal switches the circuit operation from regular to scan shift mode. For brevity, we omit here the description of the additional logic controlling the scan chains, and assume immediate access to the circuit state register r. However, for the sake of computing the time complexity, we keep in mind that it takes $2n/c$+1 clock cycles to perform the *Probe* operation, consisting of one cycle for *Capture* and n cycles for each of *ShiftIn* and *ShiftOut* operations. We also assume that c is constant and for large n: $c \ll n$. A probe operation $Probe(S, v)$ over circuit S is defined as the following sequence:

```
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
```

Further in this section, we present five algorithms for learning the circuit S using *Probe* operations. The two most important of them are *ISoTT* and *JSoTT*, while the remaining three are provided to ease the explanation. The *ESoTT* algorithm is a simple exhaustive search that we use as a baseline for the algorithms evaluation. The *KSoTT* algorithm leverages the limited transitive fan-in property of the combinational logic cones for reducing the learning complexity. *CSoTT* is an improved version of *KSoTT* that adds on-the-fly minimization for reduction of memory space requirements. The *KSoTT* and *CSoTT* make gradual introduction to the heuristic *ISoTT* algorithm. Finally, the algorithm *JSoTT* implements the adaptive junta learning known from the computational learning field [7, 17].

2.1 Algorithm ESoTT (Exhaustive Search over Truth Table)

The *ESoTT* algorithm realizes the brute-force approach by doing an ehaustive search over all possible input values.

```
1: procedure ESoTT($S$ from Definition 1)
2:     for $v$ from 0 to $2^{n+a}$-1 do
3:         $TT[v]$ := Probe($S$,$v$)
4:     end for
5: end procedure
```

The array TT contains the truth table of the function F at the end of the algorithm execution. This directly follows from the definitions of the circuit and the algorithm. The time complexity of *ESoTT* is proportional to the number of steps[1] multiplied by the length of the scan chains; and the space it takes is proportional to the size of the array:

$$T_{ESoTT} = O[(2n/c + 1) \cdot 2^{n+a}] \tag{1}$$

$$S_{ESoTT} = O[(n + b) \cdot 2^{n+a}] \tag{2}$$

Note that we obtain the time complexity by counting the probe operations. The probes are 'expensive', since their runtime depends on n. Moreover, they must run serially. The processing, in contrast, can run in parallel to probes, and also can be parallelized. Hence, the number of probes determines the algorithm time complexity.

2.2 Algorithm KSoTT (K-bounded Search over Truth Table)

The *KSoTT* algorithm leverages the limited dependency property. Consider a subset B_K of the functions in B, where every bit in the output vector depends

[1] Cumulative number of register and input bits determines the search space for function F. Below, we will use the notations: $N = n + a$ and $N_o = n + b$. Also, typically $a, b \ll n$, resulting $N, N_o \approx n$

on a limited number of input bits. Namely, for any j, $0 \leq j \leq n+b$ output bit $y_j = f(G)$ where G is a subset of bits in the input vector such that $|G| \leq K_{max}$. In this case, it is sufficient to examine the function with a set of input vectors that covers all value combinations of all subgroups G s.t. $|G| \leq K_{max}$. As a trivial example, when $K_{max} = 1$, the function can be learned using input vectors, in which only one bit is set with the addition of the 0 vector. More generally, to reconstruct a function in B_K, it is sufficient to test it with a group of vectors with hamming weight (number of bits equal to 1) of up to K_{max}. Below we describe the algorithm $KSoTT$ that implements this learning, followed by a formal proof.

```
1: procedure KSoTT Learn($S$ from Definition 1)
2:     $F_0$ := Probe($S$,0)
3:     for all $v \in \{0,1\}^N$ s.t. $HW(v) \in (1,\ldots,K_{max})$ do
4:         $P$ := Probe($S$,$v$) $\oplus F_0$
5:         for all $i$ s.t. $P_i = 1$ do
6:             Record tuple $< v, i > \in TT$
7:         end for
8:     end for
9: end procedure
10: procedure Post Process                          ▷ Eliminate redundant literals
11:     for $i$ from 1 to $n+b$ and $j$ from 1 to $N$ do
12:         if $\forall v$ s.t. $< v, i > \in TT$: $< (v \setminus v_j) \cup \neg v_j, i > \in TT$ then
13:             remove literal $v_j$ from all tuples $< v, i >$
14:         end if
15:     end for
16:     for all $< v, i > \in TT$ s.t. $HW(v) = K_{max}$ do
17:         remove all negative literals from v
18:     end for
19: end procedure
```

Theorem 2. *If the circuit S implements $F \in B_K$ (see Definition 1), at the end of $KSoTT$ execution, table TT will contain all and only implicants of all the output bits in the function $F \oplus F_0$, which is sufficient for full reconstruction of the function F.*

Proof. First, we prove the correctness of the algorithm for a function F, for which $F(0) = 0$. Let f be a hypothesis function for F resulting from the table TT. The algorithm is correct if $f_i(v) = F_i(v)$ for every input vector $v \in \{0,1\}^N$ and every output bit $1 \leq i \leq n+b$.

Case 1: Hamming weight of v is K_{max} or smaller. By definition of the algorithm, after stage 1 TT will contain tuple $< v, i >$ iff $F_i(v) = 1$. Now, we prove that stage 2 of the algorithm removes only redundant literals. Assume by contradiction that stage 2 removes some irredundant literal u_j. Literal u_j being irredundant implies that there exists a vector u with hamming weight of K_{max} or smaller, for which $f_i(u)$=1 and $f((u \setminus u_j) \cup \neg u_j)$=0. This contradicts the condition of the reduction; therefore u_j will not be removed in stage 2a. If

u_j is removed in stage 2b, then vector u contains at most K_{max}-1 irredundant positive literals. Consequently, vector u contains at least one redundant positive literal. We reached contradiction, because, as it will be proved in the next clause, all redundant positive literals are removed in stage 2a. We proved that stage 2, which comprises stages 2a and 2b does not remove irredundant literals.

Case 2: Hamming weight of v is greater than K_{max}. For function in B_K, there are at most K_{max} irredundant bits in v. Define $\acute{v}$ as:

$$\acute{v} = (\acute{v}_1, \ldots, \acute{v}_N) \in \{0,1\}^N \mid \acute{v}_j = \begin{cases} v_j, & v_j\ irredundant \\ 0, & otherwise \end{cases}$$

Hamming weight of $\acute{v}$ is at most K_{max}, hence, as demonstrated in the previous clause, after stage 1: $f(\acute{v})$=$F(\acute{v})$=$F(v)$. Now, we prove that stage 2 removes all the redundant literals. Assume by contradiction that stage 2 leaves some redundant literal u_j. Literal u_j being redundant implies that for any vector u: $f_i(u)$=1 iff $f_i(\acute{u}$=$(u \setminus u_j) \cup \neg u_j)$=1. If hamming weight of u is less than K_{max} or if u_j=1, hamming weight of $\acute{u}$ is less or equal than K_{max}. Hence, both $< u, i >$ and $< \acute{u}, i >$ will either appear or not in the table TT, which means that literal u_j will be removed in stage 2a. If hamming weight of u is equal to K_{max} and u_j=0, u_j will be removed in stage 2b. We reached contradiction.

Finally, we prove the correctness of the algorithm for any function F. Let F^0=$F \oplus F(0) \rightarrow F^0(0) = 0$. Let f^0 be a hypothesis function for F^0. We have $F = F^0 \oplus F(0) = f^0 \oplus F(0)$. □

Time Complexity Number of probes in KSoTT is equal to the number of elements in $\{0,1\}^N$ with hamming weight of K_{max} or smaller, that is $\sum_{i=0}^{K_{max}} \binom{N}{i} \leq 1 + N^{K_{max}}$. Hence, the bound for the time complexity of the algorithm can be written as:

$$T_{KSoTT} = O[(2n/c + 1) \cdot (1 + N^{K_{max}})] \quad (3)$$

Space Complexity The size of the table TT at the end of stage 1 defines the space requirement of the algorithm. TT can be stored in the form of a sparse matrix, such that for every entry only bits equal to 1 in v are stored. In the worst case from the space perspective, every output depends on one input bit only. Hence, the bound for space is:

$$S_{KSoTT} = O[K_{max} \cdot N \cdot (1 + N^{K_{max}-1})] \quad (4)$$

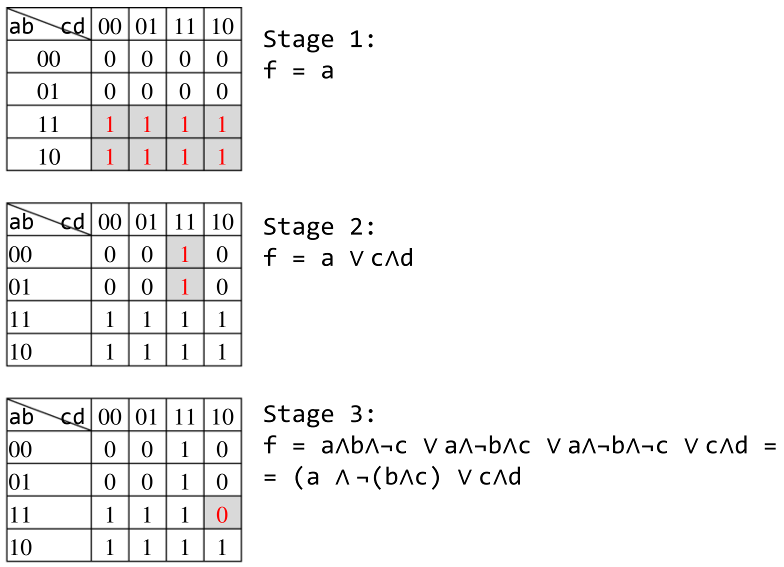
At the end of the execution, the *KSoTT* algorithm yields a correct and compact Disjunctive Normal Form (DNF) circuit representation, if followed by an additional minimization step at the end. However, larger memory space is required for the intermediate results, that is the table TT after the first stage (4).

2.3 Algorithm CSoTT (Compact K-Search over Truth Table)

The *CSoTT* solves the *KSoTT* space complexity problem by reduction of redundant literals and partial minimization during the runtime. This algorithm iteratively probes the function F with vectors v, starting with hamming weight HW=0 and going up to K_{max}. If bit i of the probe result is equal to 1, the *Onset* table is updated with a new implicant candidate, unless a cover of the corresponding term already exists in the table. If the bit is 0, but the *Onset* table includes an implicant candidate t, which is a cover of v, then t is removed from the table. Instead, *Onset* is updated with all the terms covered by t, all literals of which match bits in v equal to 1, excluding v itself.

```
1:  procedure CSoTT Learn($S$ from Definition 1)
2:      $F_0$ := Probe($S$,0)
3:      Onset[$i$] = $\emptyset$ for all $i$ from 1 to $n$+$b$
4:      for $K$ from 1 to $K_{max}$ do
5:          for all $v \in \{0,1\}^N$ s.t. $HW(v)$=$K$ do
6:              $P$ := Probe($S$,$v$)$\oplus F_0$
7:              $T$ := $\wedge T_j$ s.t. $v_j$=1      ▷ compose term from positive literals that
    correspond to bits in $v$ equal to 1
8:              for $i$ from 1 to $n$+$b$ do
9:                  if $P_i$=1 AND cover($T$)$\notin$Onset[$i$] then
10:                     add(Onset[$i$],$T$)
11:                 else if $P_i$ = 0 then
12:                     call updateOnset($T$,$i$)
13:                 end if
14:             end for
15:         end for
16:     end for
17: end procedure
18: procedure updateOnset($T$,$i$)
19:     for all $t$ ∈Onset[$i$]=cover($T$) do
20:         for all $\hat{t}$=cover($v$) s.t. $t$=cover($\hat{t}$) do
21:             $t_{inv}$ := $\wedge\neg t_n$ for $n$ s.t. $t_n$=1 AND $t_n \notin \hat{t}$
22:             $\tilde{t}$ := $\hat{t}$ ∥ $t_{inv}$
23:             add(Onset[$i$],$\tilde{t}$)
24:         end for
25:     end for
26:     remove(Onset[$i$],$t$)
27: end procedure
```

The algorithm's iterative operation can be demonstrated using Karnaugh maps. Figure 2 shows the algorithm stages for an example single-bit function $F(a,b,c,d) = (a \wedge \neg(b \wedge c)) \vee c \wedge d$. At stage 1 (Hamming Weight = 1), the only recorded implicant candidate is a. At stage 2 (Hamming Weight = 2) the implicant $c \wedge d$ is added At stage 3 (Hamming Weight = 3), the implicant candidate a is found bogus, since the probe operation on a vector $\{a,b,c,d\}$ = 1110 yields a result of 0. Thus, it is replaced with higher order terms as shown in the cor-



ab \ cd	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

```
Stage 1:
f = a
```

ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	1
10	1	1	1	1

```
Stage 2:
f = a ∨ c∧d
```

ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	0
10	1	1	1	1

```
Stage 3:
f = a∧b∧¬c ∨ a∧¬b∧c ∨ a∧¬b∧¬c ∨ c∧d =
= (a ∧¬(b∧c) ∨ c∧d
```

Fig. 2. *CSoTT* algorithm stages with their respective Karnaugh maps

responding Karnaugh map in the figure. At the end of the algorithm execution, the Onset table contains a DNF (Disjunctive Normal Form) representation of the function F. Note that minimal DNF is not guaranteed in every case, however compactness is provided heuristically.

CSoTT Time and Space Complexity The time complexity of the $CSoTT$ is equal to the one of the $KSoTT$ algorithm (3). The space required for $CSoTT$ is bounded by the worst case size of a DNF representation of a function with K_{max} variables.

$$S_{CSoTT} = O[N_o \cdot K_{max} \cdot 2^{K_{max}-1}] \tag{5}$$

In practice, the memory space for $CSoTT$ is approximately the size of the minimal DNF representation of F. Both space and time of the algorithm grow exponentially with K_{max}.

In the domain of digital circuits, K_{max} represents the transitive fan-in of the combinational logic cones that end in sequential elements or outputs of the circuit. Fig.3 shows a histogram of transitive fan-ins of flip-flops and outputs of the circuits from the ITC99 benchmark [3] set. Clearly, due to the exponential growth, K_{max} of a typical circuit is too high for the algorithm to be practical. Lower K can be selected for partial reconstruction of the function F. For example, in approximately half of the cases from the ITC'99 statistics, the transitive fan-in is smaller than 50, and for 25% of them it is below 32. This partial information may provide an adversary with sufficient information to carry out the

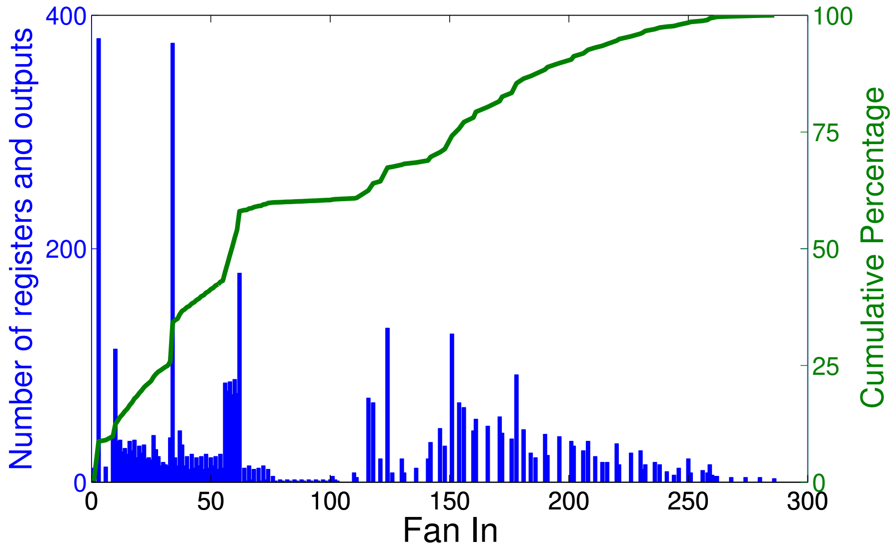


Fig. 3. Transitive fan-in histogram for ITC'99 benchmark circuits

attack. We define a metric δ_k, circuit reconstruction accuracy, as a ratio between the number of correctly reconstructed bits and the total number of bits.

2.4 Algorithm ISoTT (Incremental CSoTT)

The $ISoTT$ is a speculative algorithm that invokes $CSoTT$ a number of times with different Hamming Weight, setting some bits of the input vector to constant values based on previously learned results. In this way it leverages the presence of subcircuits, which are shared between a number of logic cones. Large digital circuits comprise hierarchical levels, which combine parts of the circuit into 'dense' structures, in which the same sub-circuit may belong to a number of logical cones. One example of this phenomenon is the carry propagation logic in arithmetic circuits. In this case, implicants with K variables can be obtained by extension of the $CSoTT$ algorithm from implicant candidates with hamming weight $K - K_{step}$, where K_{step} is some constant. Below, we define the $ISoTT$, incremental $CSoTT$ algorithm that takes advantage of this property:

```
1: procedure ISoTT($S$ from Definition 1)
2:     Pick $K_{init}$, $K_{step}$
3:     $K := K_{init}$
4:     Run CSoTT($K_{max}$=$K$) on $S$
5:     do
6:         for all $m \in$Onset s.t. HW($m$)=$K_{init}$ do
7:             for all $j$ s.t. $m_j \in m$: set $v_j$ to constant 1;
8:             Run CSoTT($K_{max}$=$K$) on the remaining bits of $v$
```

```
 9:         end for
10:         $K := K_{max} + K_{step}$
11:     while there is a change in Onset
12: end procedure
```

ISoTT implements a greedy best-first search method. At every step of the algorithm, *CSoTT* is called as many times as the number of implicants with the maximum Hamming Weight in the table. The worst case runtime of *ISoTT* is equal to the runtime of $CSoTT(K_{max}{=}K_{init})$. For a more general case, we can write the time complexity of the single algorithm step as:

$$T_{ISoTT} \leq O[S_{ISoTT(step-1)} \cdot T_{CSoTT}(K_{max} = K_{step})] \qquad (6)$$

From (6), the *ISoTT*'s performance depends on the structure of the circuit and on the size of the minimal DNF representation of the circuit. For some examples, such as the arithmetic circuits, *ISoTT* achieves full reconstruction of the circuit.

2.5 Algorithm JSoTT (Junta based Search over Truth Table)

The JSoTT is a randomized algorithm that also exploits the limited transitive fan-in property. In computational learning theory, a function f: $\{0,1\}^n \to \{0,1\}$ 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: $\{0,1\}^n \to \{0,1\}$ and $i_1, \cdots, i_k \in [n]$ [19]. Hence, algorithms for learning junta functions from queries can be leveraged for reconstructing combinational circuits (or logic cones) with a limited transitive fan-in. We take the adaptive algorithm from [6]. The algorithm runs separately for every output bit and comprises two stages: finding dependencies and function discovery. At the first stage, a set of probes with random inputs is prepared. The results of the probes are used to find input bits that influence the output (Relevant Variables or RV) with a binary search-alike method. At the second stage, a logical function is discovered by checking all the value combinations of the variables relevant to this output bit.

```
 1: procedure JSoTT Find RVs($S$ from Definition 1)
 2:     $F_0$ := Probe($S$,0)
 3:     RVs[$i$] = $\emptyset$ for all $i$ from 1 to $n$+$b$
 4:     repeat
 5:         $v$ := random(1,...,$2^N$-1)
 6:         $P$ := Probe($S$,$v$)$\oplus F_0$
 7:         add(Probes, $\langle v,P \rangle$)
 8:     until done $K_{max} \cdot 2^K_{max}$ times
 9:     for $i$ from 1 to $n$+$b$ do
10:         for all $\langle v, P \rangle$ in Probes do
11:             $\hat{v} := \{\hat{v}_1, \ldots, \hat{v}_N\} | \hat{v}_j = (v_j \in$RVs[$i$]) ? $v_j$ : 0
12:             $\hat{P}$ := Probe($S$,$\hat{v}$)$\oplus F_0$
13:             if $P_i \neq \hat{P}_i$ then
```

```
14:             find next RV by binary search on $v$ keeping all $v_j \in RVs[i]$ fixed[2]
15:             add(RVs9$i$), RV)
16:         end if
17:     end for
18: end for
19: end procedure
20: procedure JSoTT Find function
21:     Call CSoTT[3] for output bit $i$ with input composed from bits in RVs[$i$] and $K_{max} = |RVs[i]|$
22: end procedure
```

JSoTT Runtime and Space Complexity. From [6] time complexity of *JSoTT* can be written as:

$$T_{JSoTT} \leq O[K_{max} \cdot 2^{K_{max}} \cdot logN_o + N_o \cdot logN_o + N_o \cdot 2^{K_{max}}] \tag{7}$$

The algorithm's time complexity dependency on n is superlinear at most, which means *JSoTT* scales well with the circuit size. Hence, *JSoTT* is advantageous versus *ISoTT* for large circuits. However, *JSoTT* lacks the heuristic component, hence *ISoTT* is still preferable for certain circuit types. The algorithm's space complexity is similar to the *CSoTT*'s (5).

3 Experimental Results

We used a software simulator to evaluate the effectiveness of the algorithms from Section 2. The simulator works at the algorithmic level. It models the circuit under test with a *Probe* function, abstracting away from the underlying scan based sequence that implements the *Probe*. But first we checked the correctness of the scan based implementation of *Probe*, with the Validation of Concept experiment.

3.1 Validation of Concept

For this experiment, we used a simple incrementor circuit S. The circuit description was written in Verilog HDL and synthesized into gate level, followed by scan insertion. On the resulting gate level with scan we ran the *ESoTT* algorithm using Verilog behavioral simulator. Note that the algorithm execution was preceded by a learning of the length of the scan chain by shifting in a pattern to the scan input and counting number of cycles until it was observed at the scan output. The reconstructed circuit S' (Fig.4) comprises (1) a register R

[2] During the binary search, the vector v 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

[3] In number of probes, it is equal to running exhaustive search. However, we invoke CSoTT to get a compact function representation.

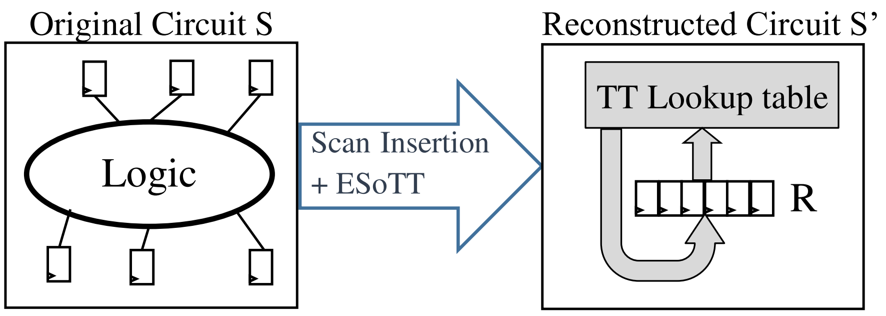


Fig. 4. Validation of Concept Circuit Diagram

comprising all n flip-flops found along the scan chain, and (2) a combinational function (represented by a truth table) that covers all value combinations of the n flip-flops. Finally, we confirmed the equivalence of the reconstructed circuit S' to the original circuit S using a formal logic equivalence tool. The remainder of this section presents results obtained from the software simulator, which uses the Probe abstraction and assumes correctness of its underlying scan based implementation.

3.2 Methodology

For additional experiments, we built a software simulator that models the functionality of digital circuits under test with a *Probe* operation: an operation that represents the circuit's next state function F according to Definition 1. The simulator also implements the algorithms defined in Section 2. The output of the algorithm, the circuit hypothesis function f, is further matched against the original circuit function F. The simulator performs the matching by comparing the outputs of the functions F and f for all possible inputs. If the circuit is too large for checking all the values, the simulator performs the matching using statistical method with a sufficiently large sample set of randomly selected inputs.

Both the $ESoTT$ and the $KSoTT$ algorithms require deterministic time and space, thus, they can be calculated. Therefore, in the graphs we present analytical data for these algorithms. For the $CSoTT$ and $ISoTT$, we present the simulation results. Also, in the case of $ISoTT$, we perform a series of runs with different K_{init} and K_{step} parameters, and then select the best results for constructing the graphs. As a criterion for evaluation, we define use the metric δ_K, circuit reconstruction accuracy, as a ratio between the number of correctly reconstructed bits and the total number of bits. Note that we use the number of probes as a measure of time. See **??** for details.

3.3 Arithmetic and Datapath Elements

At first, we evaluate the algorithms when applied to the common building blocks of the digital circuits. We start with the arithmetic circuits and measure the runtime and space required to achieve a full reconstruction of an adder circuit (δ_K=1). On one hand, the arithmetic circuits are characterized by tight dependency, which makes the limited fan-in optimization non-efficient. On the other hand, their regular and recursive structure is a useful property for the incremental $ISoTT$ algorithm. Indeed, as shown in Fig.5, $ISoTT$ is the most efficient for the adder. Note that the $ISoTT$ and $JSoTT$ have the same space requirements as the $CSoTT$.

The adder implements a single function, where the limited fan-in approach has little advantage; thus, all the algorithms still run in exponential time. We expect to see the speedup with hierarchical structures with loose dependencies between their sub-structures. As an example, we took a pipelined accumulator circuit. It is built of pipeline stages, each of them merely adds the result of the previous stage to the input vector. When unfolded to a combinational structure, it turns to a set of adders in a parallel construction. Here, K_{max} is derived from the size of the single adder and does not depend on the number of pipeline stages. Therefore, we observe polynomial growth of space and runtime in Fig.5

Next, we evaluate our algorithms with a multiplexer, which is a typical element of datapath structures. For example, unfolding a register file with scan

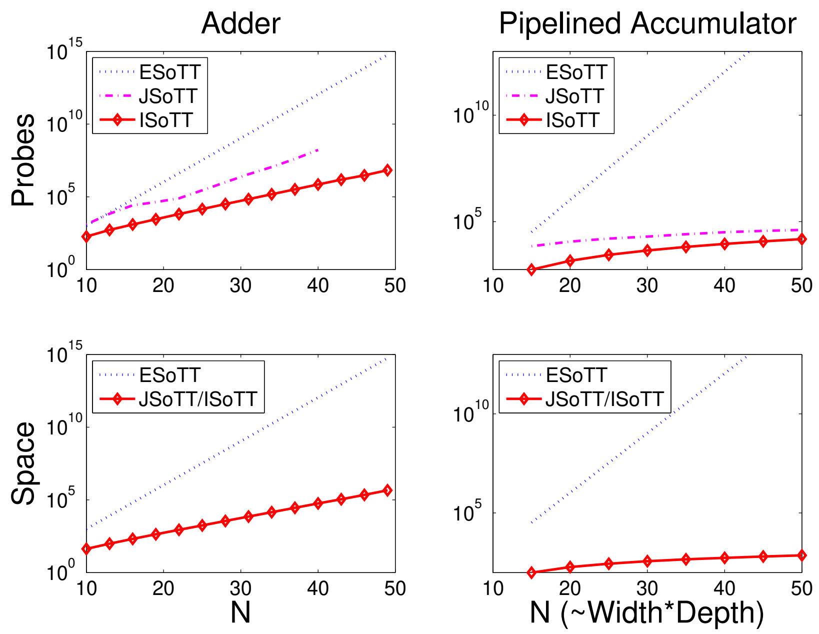


Fig. 5. Algorithm Results with Arithmetic Circuits

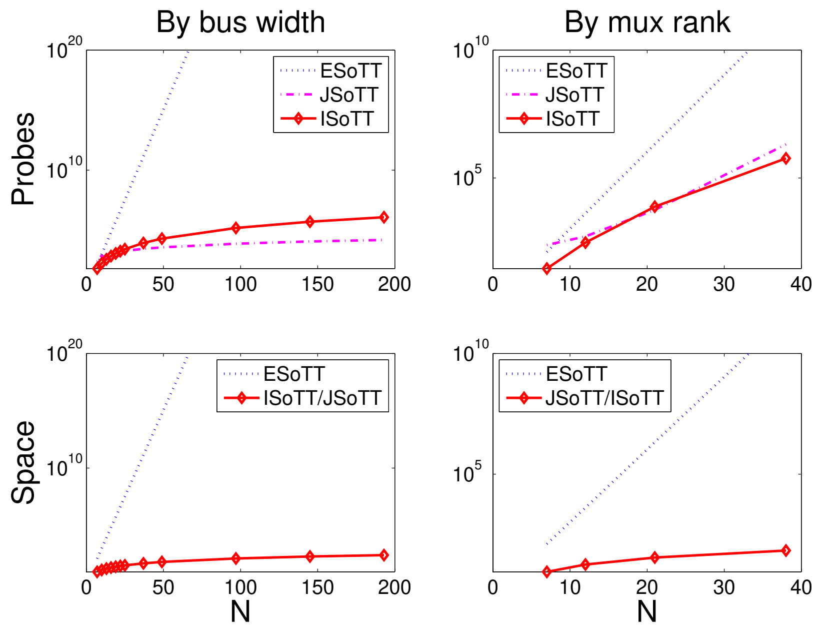


Fig. 6. Algorithm Results with a Multiplexer

results in a structure resembling a multiplexer. The size of the input vector to the multiplexer with bus width W and rank (number of input busses) R is $N{=}R{\cdot}W$. However, every output bit depends on only $R{+}log_2R$ bits, namely K_{max} does not depend on the bus width. This makes the algorithms, leveraging the limited fan-in, particularly efficient for wide datapath structures. Fig.6 illustrates this phenomenon. Interestingly, $ISoTT$ performs better for all the arithmetic circuits and even for multiplexer circuits with high rank, in spite of the fact that $JSoTT$ has lower theoretical complexity, Two explanation to these phenomena. First is the heuristics of the $ISoTT$. Second, if for $JSoTT$ K_{max} represents the upper bound on the transitive fan-in of combinational circuits, for $ISoTT$ it is enough to use the maximum implicant size as K_{max}. In this sense, $ISoTT$ can be regarded as a DNF formulae learning algorithm [2].

3.4 AES Accelerator

In previous section we observed the $ISoTT$'s superior performance with regular and structured circuits. Naturally, the more structure (or less entropy) is present in the circuit, the better $ISoTT$ can exploit it. Alternatively, $JSoTT$ fits better to circuits having high entropy. Such circuits are characterized by the avalanche effect that eases exploration of influences. We took the tiny AES core from the Open Cores repository [11] as an ultimate example of a high-entropy circuit. This is a highly pipelined implementation of the AES encryption/decryption, which contains 6848 internal registers. The maximum transitive fan-in of the circuit is 8. Nevertheless, thanks to the avalanche effect, K_{max} as low as 4 appears suffices for detecting all the influences. Hence, $JSoTT$ learns the AES circuit precisely and with little effort, despite its formidable size, see Table 1. The parameter δ_K denotes a circuit reconstruction accuracy, defined as a percentage of the correctly reconstructed bits from the output vector. Note that due to the algorithm's random nature, results differ slightly between algorithm invocations.

Table 1. AES Circuit Reconstruction

Selected K_{max}	**Accuracy (δ_K)**	**Number of probes**
2	44%	389109
3	93%	1251169
4	100%	1594819

The interesting observation from the results of this experiment is that cryptographic functions that are considered complex may actually present an easy target for the reverse engineering attack. How easy it is, depends on the implementation. Additionally, thanks to the scalability of $JSoTT$, the size of the circuit is less important, and what matters is its structure. For example, deeper pipeline is expected to increase the chance of successful reverse engineering attack thanks to the finer partitioning of the logic.

4 Related Work

Several works discuss non-invasive methods for reverse engineering. Kash et al. [12] proposes an optical method for monitoring switching activity of different locations in the circuit under test. Further, this information can be compared against reference models for partial extraction of the circuit functionality. This method requires prior knowledge of the circuit and access to test patterns. A number of works [1, 14, 16, 22, 24] present various techniques addressing phase two of the reverse engineering process, behavioral model extraction. Our work focuses on the first phase, netlist extraction.

Saab et al [21] were the first to propose using the scan for extraction of functionality of a digital integrated circuit. They present a randomized algorithm for finding dependencies between registers and evaluate it with a few small benchmarks and an AES function. In this paper, we add the discovery phase and provide a systematic learning method adding discussion of algorithms' complexity and scalability metrics. We also propose a heuristic algorithm tuned for certain types of circuits, such as arithmetic circuits.

A number of papers propose countermeasures against the scan base side channel attack. Da Rolt et al. [4] present their comprehensive classification, which divides the countermeasures to the following groups: Advanced DFT Structures, BIST, Secure Wrappers, Unbounding, Scrambling, Access Restrictions, Secret-Free Test, Modified Scan Chain and Countermeasures against Microprobing. Below 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 (Built-in Self-Test) 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 diagnostic. 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 reverse engineering attack. 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 adversary can separate between the two. Finally, the Secret-free Test solution prevents volatile secret data from being exposed. However, it doesn't 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 DPA [23]. Often, such mechanisms use global secrets; hence, it is sufficient to hack a single unit for gaining access to all the other units of the same product.

Previous work usually relies on the circuit ability to switch dynamically from mission mode to scan mode and back for retrieving runtime information. As a result, one of the popular countermeasures against scan-based attacks is enforcing reset when switching between the modes [9]. The reverse engineering attack retrieves static design information from the scan chains, and therefore is immune to this countermeasure.

5 Conclusions and Future Work

This paper presents new methods of the scan based side channel attack, an attack that allows for the non-invasive reverse engineering of entire device at a logical netlist level. The methods take into advantage the limited transitive fan-in property for efficient learning. This leads to a novel heuristic that enables full reconstruction of some circuits in a polynomial time. By using an algorithm that employs junta learning, we successfully reconstructed a hardware implementation of the AES cryptographic function. Hence, our experimental results show that exploitation of the scan side channel for reverse engineering is a real threat, which can give the adversary full or partial information on the circuit functionality. Even partial information may reveal sufficient data to serve the malicious purposes. The presented attack is also immune to some of the popular countermeasure techniques.

An important contribution of this paper is in demonstrating the feasibility of a non-invasive reverse engineering attack using heuristic algorithms that exploit common properties of digital circuits. Our future work will focus on examining algorithms that provide higher efficiency or that are tuned for specific circuit types. The scalability of $JSoTT$ makes the presented method potentially applicable also to large scale SoC devices. Combining $JSoTT$ with algorithms that exploit additional heuristics (such as $ISoTT$) can target complex logic. Algorithms from the fields of learning automata, probabilistic learning, machine learning and SAT based algorithms are good candidates for this purpose. Different algorithms may match different types of circuits. A case of special interest is when some part of the circuit is known a priori. Here, the reverse engineering technique may help in detecting hardware Trojans or accidentally inserted back-doors.

In the domain of the scan usage, there is a number of questions that call for continued research for addressing modern complex SoC devices. When the device does not fully comply with the definition of fully synchronous single clock digital synchronous design, additional steps are required. For example, when part of the design is asynchronous or there are several clock domains or non-standard cell based sections present, such as memories, analog circuits or non-scannable logic. In a typical scan insertion flow, such parts are masked with a special scan logic. Scan chains, where some of the registers are inverted represent an additional problem to be solved. In addition, an effect of scan compression should be examined. According to [5] and [8], the existing scan compression techniques have limited contribution to security. However, their effect on the success of reverse engineering is a separate research question.

Finally, protection techniques from the reverse engineering with scan should be explored. Previous work suggests a number of techniques for general protection from the scan side channel attacks, such as encryption, authentication and obfuscation. Some of them apply to this type of attack as well. Protection techniques optimized for the reverse engineering attack, such as obfuscation of the combinational functions, which preserves automatic test capability, present an additional interesting extension to the research.

References

1. Bouchaour, H., Ouali, M., Lebbah, Y.: Towards a Method for VLSI Circuit Reverse Engineering. In: Amine, A., Mohamed, O.A., Benatallah, B., Elberrichi, Z. (eds.) CIIA. CEUR Workshop Proceedings, vol. 825. CEUR-WS.org (2011)
2. Bshouty, N.H., Jackson, J.C., Tamon, C.: More efficient PAC-learning of DNF with membership queries under the uniform distribution. Journal of Computer and System Sciences 68(1), 205–234 (feb 2004)
3. Corno, F., Reorda, M., Squillero, G.: RT-level ITC'99 benchmarks and first ATPG results. IEEE Design & Test of Computers 17(3), 44–53 (2000)
4. Da Rolt, J., Das, A., Di Natale, G., Flottes, M.L., Rouzeyre, B., Verbauwhede, I.: Test Versus Security: Past and Present. IEEE Transactions on Emerging Topics in Computing 2(1), 50–62 (mar 2014)

5. Da Rolt, J., Di Natale, G., Flottes, M.L., Rouzeyre, B.: New security threats against chips containing scan chain structures. 2011 IEEE International Symposium on Hardware-Oriented Security and Trust pp. 110–110 (jun 2011)
6. Damaschke, P.: Adaptive Versus Nonadaptive Attribute-Efficient Learning. Machine Learning 41(2), 197–215 (2000)
7. Damaschke, P.: On parallel attribute-efficient learning. Journal of Computer and System Sciences 67(1), 46–62 (aug 2003)
8. Das, A., Ege, B., Ghosh, S., Batina, L., Verbauwhede, I.: Security Analysis of Industrial Test Compression Schemes. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(12), 1966–1977 (dec 2013)
9. Hely, D., Bancel, F., Flottes, M., Rouzeyre, B.: Test Control for Secure Scan Designs. European Test Symposium (ETS'05) pp. 190–195 (2005)
10. Hely, D., Rosenfeld, K., Karri, R.: Security challenges during VLSI test. In: IEEE 9th International New Circuits and systems conference. pp. 486–489. Ieee (jun 2011)
11. Hsing, H.: tiny_aes IP project (2012), http://opencores.org/project,tiny_aes
12. Kash, J., Tsang, J., Knebel, D.: Method and apparatus for reverse engineering integrated circuits by monitoring optical emission (dec 2002)
13. Lee, J., Tehranipoor, M., Patel, C., Plusquellic, J.: Securing Designs against Scan-Based Side-Channel Attacks. IEEE Transactions on Dependable and Secure Computing 4(4), 325–336 (oct 2007)
14. Li, W., Wasson, Z., Seshia, S.A.: Reverse Engineering Circuits Using Behavioral Pattern Mining. In: IEEE International Symposium on Hardware-Oriented Security and Trust. pp. 83–88 (2012)
15. Lupanov, O.: On circuits of functional elements with delay. Probl. Kibern (23), 43–81 (1970)
16. McElvain, K.S.: Methods and apparatuses for automatic extraction of finite state machines (2001)
17. Mossel, E., O'Donnell, R., Servedio, R.P.: Learning juntas. In: Proceedings of the thirty-fifth ACM symposium on Theory of computing - STOC '03. p. 206. ACM Press, New York, New York, USA (jun 2003)
18. Nohl, K., Evans, D., Starbug, S., Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: USENIX Security,. pp. 185–193. USENIX Association (jul 2008)
19. O'Donnell, R.: Analysis of boolean functions. Cambridge University Press (2014)
20. Rolt, J.D., Di Natale, G., Flottes, M.L., Rouzeyre, B.: Thwarting Scan-Based Attacks on Secure-ICs With On-Chip Comparison. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22(4), 947–951 (apr 2014)
21. Saab, D.G., Nagubadi, V., Kocan, F., Abraham, J.: Extraction based verification method for off the shelf integrated circuits. In: 2009 1st Asia Symposium on Quality Electronic Design. pp. 396–400. IEEE (jul 2009)
22. Shi, Y., Ting, C.W., Gwee, B.H., Ren, Y.: A highly efficient method for extracting FSMs from flattened gate-level netlist. Proceedings of 2010 IEEE International Symposium on Circuits and Systems pp. 2610–2613 (may 2010)
23. Skorobogatov, S., Woods, C.: Breakthrough silicon scanning discovers backdoor in military chip. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems CHES 2012. Lecture Notes in Computer Science, vol. 7428, pp. 23–40. Springer Berlin Heidelberg, Berlin, Heidelberg (sep 2012)
24. Subramanyan, P., Tsiskaridze, N., Pasricha, K., Reisman, D., Susnea, A., Malik, S.: Reverse Engineering Digital Circuits Using Functional Analysis. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013. pp. 1277–1280. IEEE Conference Publications, New Jersey (2013)

25. Torrance, R., James, D.: The State-of-the-Art in IC Reverse Engineering. In: Clavier, C., Gaj, K. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2009. Lecture Notes in Computer Science, vol. 5747, pp. 363–381. Springer Berlin Heidelberg, Berlin, Heidelberg (aug 2009)
26. Wegener, I.: The Complexity of Boolean Functions. John Wiley & Sons, Inc. (1987)