Circuit Deobfuscation from Power Side-Channels using Pseudo-Boolean SAT

Kaveh Shamsi
Department of Electrical and Computer Engineering
University of Texas at Dallas
Richardson, Texas, USA
kaveh.shamsi@utdallas.edu

Yier Jin
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
yier.jin@ece.ufl.edu

***Abstract*—The problem of inferring the value of internal nets in a circuit from its power side-channels has been the topic of extensive research over the past two decades, with several frameworks developed mostly focusing on cryptographic hardware. In this paper, we focus on the problem of breaking logic locking, a technique in which an original circuit is made ambiguous by inserting unknown "key" bits into it, via power side-channels. We present a pair of attack algorithms we term PowerSAT attacks, which take in arbitrary keyed circuits and resolve key information by interacting adaptively with a side-channel "oracle". They are based on the query-by-disagreement scheme used in functional SAT attacks against locking but utilize Psuedo-Boolean constraints to allow for reasoning about hamming-weight power models. We present a software implementation of the attacks along with techniques for speeding them up. We present simulation and FPGA-based experiments as well. Notably, we demonstrate the extraction of a 32-bit key from a comparator circuit with a 2^{31} functional query complexity, in ∼64 chosen power side-channel queries using the PowerSAT attack, where traditional CPA fails given 1000 random traces. We release a binary of our implementation along with the FPGA+scope HDL/setup used for the experiments.**

***Index Terms*—Hardware Security, Logic Locking, Side-channel Attacks**

I. Introduction

Given a circuit $c(k, x)$, where inputs x are controllable and a set of "key" inputs k are set to a secret vector k_*, the problem of inferring these secret values from chosen/given observations of the form $y_i = c(k_*, x_i)$ appears in various contexts. Logic locking [1], IC camouflaging [2] and split-manufacturing [3] are three categories of techniques that can be used to partially hide the design of an integrated circuit (IC) from an untrusted foundry or end-user. Attacking these schemes given access to a functional IC that can be used as an input-output pattern oracle, can be formulated as such an inference problem. In this context, the problem is sometimes referred to as oracle-guided circuit deobfuscation and has been under study for some years.

A set of generic algorithms have been proposed to attack the problem and used extensively to analyze the security of various design hiding schemes. The most powerful and versatile of these are SAT-based attacks [4]–[8]. These are based on a common flow of formulating a SAT problem that captures input patterns that can teach the attacker useful information about the key, querying these patterns on the oracle, and then using SAT formulae to capture keys that conform to these oracle observations. This in fact is in line with the uncertainty-sampling or query-by-disagreement paradigm in active learning [9]. Importantly, given that SAT solvers exhaustively search the solution space of a SAT problem, SAT attacks can return provably correct keys, which is not possible with statistical attacks.

In this paper, motivated by the power of SAT attacks in circuit deobfuscation from functional queries, we explore a similar framework but using power side-channel observations instead of functional ones.

There are several existing traditional side-channel analysis frameworks such as correlation power analysis (CPA) [10], differential power analysis (DPA) [11], mutual-information analysis (MIA) [12] and template attacks [13]. The majority of the work on power side-channel attacks has been focused on cryptographic hardware, with specific cryptographic functions demanding their own specific adaptations. As for circuit deobfuscation from power side-channels, a DPA attack against locking was presented in [14] which argued that functional secrecy through exponential query schemes implies DPA security. [15] presented a template attack approach against locking.

We show in this paper however, that exponential query schemes can in fact be broken through power side-channels without the need for the extensive profiling needed by template attacks, or DPA which is traditionally geared towards cryptographic hardware and not generic circuits. Instead through an analog of the SAT attack operating on side-channel queries rather than functional queries.

Specifically, we deliver the following:

- We show how similar to the way that sparse functions can create high minimum query counts for functional oracle-guided attacks, a similar pattern can be seen at first with respect to side-channels. E.g. comparator logic is harder to break with baseline CPA/DPA than block-cipher logic. While it has been suggested [14] that this may thwart side-channel attacks against sparse obfuscation schemes, we demonstrate that this is not true and that a smarter querying strategy, which is what baseline CPA/DPA lack, can easily break comparator (point-function) logic.
- We present a query-by-disagreement framework for circuit deobfuscation from side-channels by finding disagreement-producing side-channel queries and then fitting keys to the side-channel observations of the equality form $tr_i = h(k_*, x_i)$, h being a hypothesis power model. We introduce the use of Pseudo-Boolean (PB) SAT-solving for this in an attack we call PowerSATeq.
- Given that precise equality-conditions can be hard to extract from noisy side-channel measurements, we develop a much more practical and powerful attack that finds "query pairs" that produce key-dependent ambiguity in the comparison direction of their corresponding trace pairs $tr_f \overset{?}{>} tr_s$. These differential measurements are then used as conditions on the key to be satisfied using PB SAT in an attack we call PowerSATdiff.
- Given that these PB SAT instances can be much harder to solve than conventional SAT attack calls, we propose several ways to speed up the attack including simplifying conditions, slicing and dicing, and recovering provably correct partial keys.
- We report the performance of the attacks against a simulated hamming-weight model on ISCAS benchmarks. We also demonstrate proof-of-concept attacks on an FPGA implementation. We release a binary of our implementation along with the hardware/software code for an automatic circuit to trace-oracle flow. This we hope can serve as a platform and benchmark set for better studying generic adaptive side-channel attacks, compared to the existing offline block-cipher side-channel challenge traces.

The paper is organized as follows: Section II presents preliminaries, Section III presents the attacks, Section IV presents experiments, and Section V concludes the paper.

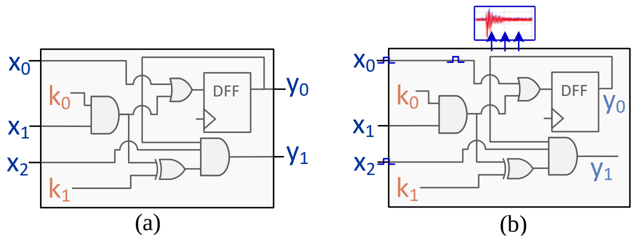

Fig. 1: (a) Functional deobfuscation vs. (b) side-channel deobfuscation threat models. In (b) the attacker can toggle inputs and observe side-channels but does not need to observe outputs.

II. Preliminaries

Logic Locking Model. Formally, logic/circuit locking can be modeled as transforming an original circuit $c_o : I \rightarrow O$ where $I = \{0,1\}^n$ and $O = \{0,1\}^m$ are the n/m-bit input/output space respectively to an obfuscated/locked circuit $c_e : K \times I \rightarrow O$, where $K = \{0,1\}^l$ is the l-bit added key space. There must exist a correct key $k_* \in K_* \subset K$ such that $\forall x \in I, c_e(k_*, x) = c_o(x)$. The space of possible functions that c_e can take is denoted by $\mathcal{C}_e = \{c_e(k,x) | k \in K\}$. The error rate of the obfuscation can be defined as the probability of producing incorrect outputs over the key/input space i.e. $\Pr_{x \in I, k \in K}[c_o(x) \neq c_e(k,x)]$. Note that the attacker has a representation of c_e, i.e. the obfuscated circuit's netlist is public whereas c_o's is private.

SAT Attack. The SAT attack is an oracle-guided attack meaning the attacker has black-box or oracle access to c_o, i.e. can make arbitrary queries of the form $c_o(x_i) \rightarrow y_i$. The attack begins by building a miter condition $M \equiv (c_e(k_1,x) \neq c_e(k_2,x))$. Satisfying M will return some $\hat{x}$, and $\hat{k_1}, \hat{k_2}$, where $\hat{x}$ is called a discriminating-input-pattern (DIP). Since the output of c_e on $\hat{x}$ is different for $\hat{k_1}$ vs. $\hat{k_2}$, at least one should produce an output that disagrees with the oracle's output on $\hat{x}$. This allows us to disqualify at least one wrong key with each DIP query. The queried observation in the form $(c_o(\hat{x}) = c_e(k_1, \hat{x}) = c_e(k_2, \hat{x}))$ is ANDed with M, as an input-output (IO)-constraint and the process repeats until M is no longer satisfiable. At this point, satisfying the IO-constraints F alone should return a correct key $k_* \in K_*$. This is because there exists no remaining two keys that conform to all oracle observations, and produce different outputs functionalities. Hence F must capture only functionally correct keys. Algorithm 1 shows this procedure.

Algorithm 1: Given oracle access to c_o and the circuit c_e returns a guaranteed correct key $k_* \in K_*$ if $c_o \in \mathcal{C}_e$.

```
1 Function SATAttack($c_e$, $c_o$ as black-box):
2     $F \leftarrow true$
3     $M \leftarrow c_e(k_1, x) \neq c_e(k_2, x)$
4     while $F \wedge M$ is solvable do
5         $\hat{x}$, $\hat{k_1}$, $\hat{k_2} \leftarrow$ SAT($F \wedge M$)
6         $\hat{y} \leftarrow c_o(\hat{x})$
7         $F \leftarrow F \wedge (c_e(k_1, \hat{x}) = \hat{y}) \wedge (c_e(k_2, \hat{x}) = \hat{y})$
8     satisfy $F$ with $\hat{k}_1$ and $\hat{k}_2$
9     return $\hat{k}_1$ as a correct key $k_*$
```

Thwarting the SAT Attack. The baseline SAT attack is effective in deobfuscating benchmark circuits with thousands of gates obfuscated with various traditional locking/camouflaging schemes [4]. One way to slow down the SAT attack or any oracle-guided attack is to reduce the disqualifying ability of DIPs/queries. If every possible DIP disqualifies no more than d bad keys, any oracle-guided attack would need $min(2^n, 2^l/d)$ queries to precisely identify the correct key. This typically occurs in the presence of comparator logic (point-functions). A comparator $P(k_*, x) = (x \overset{?}{=} k_*)$ has to be queried on average on half of its entire input space to find the point on which it outputs 1, i.e. where $x_p = k_*$. The low activation rate conversely leads to an exponentially small error rate. This contention between error rate and query count appears to be unavoidable [16], [17]. Several SAT attack variants take an error-aware approach to deobfuscation due to this [6], [18].

Formal Security in Locking. A few formal notions of security for locking were defined in [19]. Exact-functional-secrecy EFS demands that the circuit's precise functionality not be recoverable in less than time t. Approximate-functional-secrecy AFS demands that the circuit not be approximable with accuracy better than $1-\epsilon$ in time t. While exponential AFS ($t > O(2^n)$) can be impossible to achieve in many circuits, EFS and a more relaxed notion of approximation-resiliency (best-possible) BPAFS can be achieved with point-function schemes, and universal circuits respectively [19].

Side-Channel Attacks. The operation of a physical implementation of a Boolean circuit $c_e(k_*, x)$ will emanate side-channel leakage through its power-consumption, electromagnetic radiation (EM), its timing, or a combination of the three. This leakage can be captured by a function describing the precise value of the emanation at time t: $\mathcal{L}(k_*, x, \delta x)[t] = f(k_*, x, \delta x)[t] + \mathcal{N}[t]$ where f and $\mathcal{N}[t]$ capture the state-dependent/independent parts of $\mathcal{L}$ respectively. δx here is used to denote a change in the circuit's input that leads to dynamic power/EM/timing and is necessary for the non-static attacks discussed in this paper. If a circuit $c_e(k_*, x)$ is functionally hiding k_* securely, it is still possible for $\mathcal{L}$ to leak k_*. Given T different $(x_i, \delta x_i)$ we can collect T different side-channel "traces": $\mathcal{T} = \{\mathcal{L}(k_*, x_i, \delta x_i)\}$. An active adaptive attacker can choose on what $(x, \delta x)$ points to collect traces, while a passive non-adaptive/offline attacker is given a trace set for a predetermined (random) $(x, \delta x)$ collection.

Simple power analysis (SPA) [11] aims to infer secrets by manual context-specific observation of $\mathcal{L}$ at chosen points. Correlation power analysis (CPA) assumes a power hypothesis model $h(k, x, \delta x)$, then for all possible hypothesis values of the key k_h computes the (Pearson) correlation between the hypothesis power vector $\{h(k_h, x_i, \delta x_i)\}$ and the actual traces $\{\mathcal{L}(k_*, x_i, \delta x_i)\}$. For $k_h = k_*$, this correlation should be maximized if h is a good approximation of $\mathcal{L}$. Mutual information analysis (MIA) uses the mutual-information function instead of correlation. This allows using a less precise h, creating a somewhat model-free/blind attack.

Differential power analysis (DPA) uses a binary function $\Delta(k_h, x_i, \delta x_i) \rightarrow \{0,1\}$ to decide which of two bins $\mathcal{T}_0$ and $\mathcal{T}_1$ to place each trace in. The difference between the average of the two bins is computed. If Δ is a simple function of key-dependent bits in the circuit, for $k_h \neq k_*$, Δ's trace division becomes decoupled from the circuit's state and arbitrary causing the difference to vanish. For the correct key $k_h = k_*$, the difference of averages tends to spike. Picking the right Δ function is critical to DPA's success.

Deobfuscation from Side-Channels Threat-Model. A depiction of the attacker's view in circuit deobfuscation from side-channels is shown in Fig. 1. An important distinction to note compared to functional attacks is that the attacker need not observe the functional output of the oracle circuit. While functional outputs can be combined with side-channel observations for more powerful attacks, one of the main appeals of side-channel-based deobfuscation is the elimination of this requirement. This is especially important against scan-chain-inaccessible obfuscated circuits. The presence of unobservable state elements in a circuit will force a functional attacker to resort to

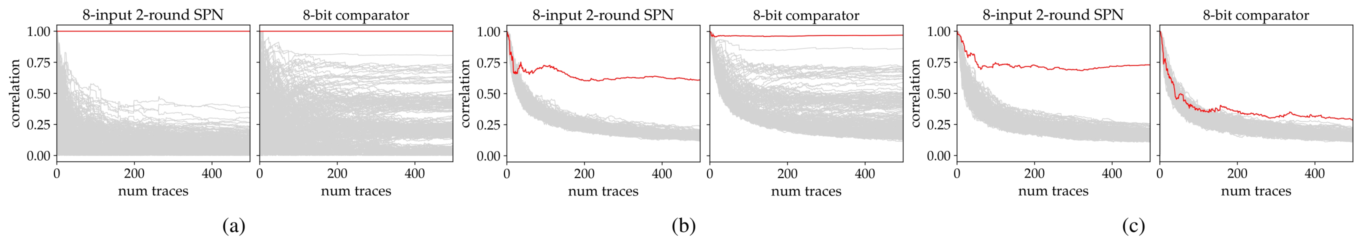

Fig. 2: CPA correlation per key-hypothesis and trace count with the red trace signifying the correct key, using (a) no-noise simulated hamming-weight model, (b) spice simulation, (c) FPGA traces. It can be seen that in all three, the SPN which is a much more entropic circuit has a more “outstanding” correct key correlation line.

sequential deobfuscation such as the model-checking (MC) attack [8], [20]. MC attacks however have runtimes that can range from slightly above the combinational SAT attack to orders of magnitude higher. This is while for deep circuits, the side-channel attacker can truncate the traces and study the logic that is immediately connected to controllable inputs, working through the rest of the circuit [15].

III. SAT-based Deobfuscation from Side-Channels

A. Motivation

CPA, DPA, and MIA all run in time exponential to the number of key-bits, i.e. $O(2^l)$ for an l-bit key. This is because each has to enumerate all key hypotheses and perform computations for each one. In the context of cryptographic circuits, this is typically managed by dividing the key vector into smaller partitions based on some understanding of the circuit's structure. For instance, for AES-128 which consists of 10-rounds, with each round operating on 16 8-bit state blocks, it is best to partition the key along these state blocks. For CPA for instance, $\{h(k_h, x, \delta x)\}$ is computed for all 2^8 possibilities for the first of 16 blocks in the key, while assigning a random fixed value to the rest of the 15 blocks in the key vector, and computing the correlation. This approach can recover the first 8-bits of the key since the real power consumption tends to correlate maximally with a hypothesis power when the first 8-bits are correct, somewhat irrespective of the remainder of the key bits. For DPA, a similar cipher-dependent key partitioning is performed. A generic CPA/DPA/MIA attack that can take in arbitrary circuits requires a generic key partitioning scheme. It is possible to build such generic partitioning using heuristics on the circuit structure which we do in our experiments as well.

The baseline CPA/DPA/MIA are also non-adaptive/offline, which fits well with perhaps the more common real-world scenario where a cryptographic circuit is being observed but not controlled by an attacker. In circuit deobfuscation however, an active attacker is the standard assumption. The attacker in possession of the functional IC can pick input values x, flip the ones that he intends through δx, and record traces/timing on different pins indefinitely.

The offline limitation becomes clear when we run CPA with random input patterns on two very different circuits. Per Fig. 2 we have run baseline CPA on a substitution-permutation network (SPN) and an 8-bit comparator. The SPN implements two rounds, each consisting of a key-mixing stage in which the 8-bit input is XORed with an 8-bit key, a substitution stage, in which the 8-bit state is sent through a $2^8 \rightarrow 2^8$ random look-up-table synthesized to gates, and a permutation stage, in which the 8 bits are shuffled, i.e. ($spn\_round(x, k) = p(s(x \oplus k))$). The 8-bit key is shared among both rounds. The comparator implements $P(k, x) \leftarrow (k \overset{?}{=} x)$.

Since there are only 8-bits in the key, no partitioning is necessary. The hypothesis power model for CPA is taken to be the number of flips occurring on the circuit nodes given a δx of hamming-weight 1. This is done using the power model itself as traces in Fig. 2a, spice simulation using the Nangate15nm library in Fig. 2b, and traces collected from an FPGA implementation in Fig. 2c.

As can be seen from Fig. 2, the correct key for the SPN is more distinct than the case for the comparator under all three trace-collectors. In other words, an incorrect key on the SPN produces much more divergent power signatures than on the comparator. This resembles a similar dichotomous pattern that emerges if the two circuits are placed under a functional oracle-guided attack. On the SPN, incorrect keys produce wildly incorrect output functionalities. This high entropy/error allows resolving the SPN with a smaller number of DIPs, since each DIP is highly likely to disqualify a substantial portion of the key space. Conversely, the comparator takes an exponential number of queries $\sim 2^7$ for its output to activate, at which point the key can be identified. The comparator can also be approximated with exponentially small accuracy by simply picking a random key, while a random key for the SPN is likely to produce a highly inaccurate function relative to the original.

It seems from the above discussion that we should expect functional-secrecy through exponential minimum query counts for a locked circuit to translate into a similar kind of security with respect to side-channel information. In fact, [21] made a similar argument regarding how functional security should imply DPA security. Note that DPA will likely perform worse than CPA if the output node of the comparator is picked as the DPA trace distinguisher. The skewed statistics of the output net will create heavily unbalanced trace bins. [21] argued that this forces the attacker to perform the DPA distinguishing with every internal net/cone and traverse the exponentially large key space at the input of these cones.

However, if we take a step back and consider the problem of deobfuscating from side-channels in the more general view: that an attacker wants to learn the key by querying a side-channel-oracle $\mathcal{L}$ on chosen points $(x, \delta x)$, we can see that breaking the comparator does not require exponentially many power queries. In fact, we can devise an adaptive attack that will extract the key k_* for an arbitrary n-bit comparator $P(k_*, x)$ in $2n$ side-channel queries. While this attack is not generic, i.e. it cannot be applied to non-comparator circuits, it does show that EFS does not guarantee side-channel resilience.

Per Fig. 3 the procedure is as follows. The attacker sets an n-bit arbitrary vector to the input x. Then begins flipping the input x_0 (i.e. $\delta x_0 = 1$). If w_1 which captures the equality of x_1 and k_1 is 1, then this flipping will propagate to w_4. Otherwise, it will get blocked. If no other inputs are flipping, the total number of wires flipping in the circuit and consequently the dynamic power consumption of the circuit is different in the two cases. Hence, we have found a way to relate the equality of x_1 and k_1 to a difference in power consumption. The attacker can set x_1 to 1, and flip x_0, and collect trace tr_0. Then set x_1 to 0, flip x_0, and collect trace tr_1. If the expected difference between these traces is not lost in the noise, then the attacker will

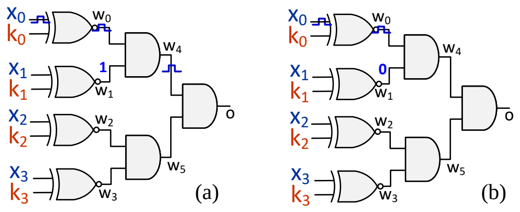

Fig. 3: Comparator circuits under a side-channel attack. If wire $w_1 = 1$ as per (a), then toggling x_0 while keeping all other x_i constant, should produce more bit flips, than the same toggling when $w_1 = 0$ (b). This allows us to determine whether $x_1 = k_1$ or not with two traces.

learn the value of k_1, a single key bit, with only 2 traces. For an n-bit comparator, this procedure can be repeated for every bit and recover the full key successively with $2n$ traces. This is while functionally, the comparator requires an average of 2^{n-1} queries to fully resolve. This instance serves as a counterexample proving the following lemma:

LEMMA 1. *Exponential* EFS *security for $c_e(k_*, x)$ does not imply exponential resiliency against exact functional learning of $c_e(k_*, x)$ from chosen queries to its hamming-weight/flip-rate oracle h_e:*

$$h_e(k_*, x, \delta x) = \sum_{w_i \in wires(c_e)} (w_i(k_*, x) \neq w_i(k_*, x \oplus \delta x))$$

The above h assumes that every wire flip contributes equally to the power consumption. In a real design, however, this is not true since wire-load varies from net to net and impacts power consumption.

The above attack on the comparator circuit required a meticulous step-by-step adaptive approach. Traditional CPA/DPA/MIA schemes that do not optimize for selected input patterns fall short of this. This serves as a motivation to explore generic adaptive attacks since, in circuit deobfuscation, unlike breaking cryptographic primitives, comparator and SPN-like logic can coexist in a general circuit slice c_e. The attack should try to infer a chosen secret with maximum accuracy and perhaps minimum query count regardless.

B. A Side-Channel SAT Attack with Equality Conditions

The SAT attack for circuit deobfuscation from functional queries is a generic adaptive attack. Any arbitrary circuit formatted as a locked netlist $c_e(k, x)$ for which each gate can be translated to a CNF formula can be passed to the SAT attack. Mining for DIPs that guarantee to disqualify a non-zero number of bad keys falls in line with the long-standing query-by-disagreement (QBD) strategy in active learning. Absent an optimal strategy for learning a function f, querying on points that maximize the current hypothesis's uncertainty tend to achieve faster learning rates relative to random sampling [22].

Side-Channel-based Deobfuscation with QBD. We replicate this approach for the case of deobfuscation from side-channels. We search for input patterns that lead to different side-channel patterns under different key hypotheses. We call such an input pattern a power discriminating input pattern (PDIP). Note that a PDIP is an input vector x plus a flip vector δx. Our attack mines for PDIPs, queries them, appends the observed trace to a set of "thus-far observed traces", and repeats the process until no more PDIPs can be found.

To mine for PDIPs, we need to build a power miter condition $PM = (h_e(k_1, x, \delta x) \neq h_e(k_2, x, \delta x))$. The simplest power-model h_e is simply one that counts the number of bit-flips in the circuit as seen in Lemma 1. A more complicated model includes a weight α_i for each wire to capture relative wire-load differences: $h_e(k_*, x, \delta x) = \sum_{w_i \in \text{wires}(c_e)} \alpha_i (w_i(k_*, x) \neq w_i(k_*, x \oplus \delta x))$. More complicated models that capture static leakage can be built as well but are outside the scope of this paper.

Psuedo-Boolean Constraint Solving. Even the simplest h_e condition requires bit summation. PM hence is in fact a "Pseudo-Boolean" (PB) constraint. A PB-constraint captures the comparison of a real-weighted sum of Booleans with a constant. PB-constraint solving has been an active area of research for many years [23], [24]. One can reduce satisfying a PB formula to integer-linear-programming (ILP) and pass it to an ILP solver such as Gurobi. We in fact tried using this approach shortly and observed very poor performance.

In many practical PB problems, including the ones here, the PB part of the problem is dwarfed by the pure Boolean part, in terms of the number of constraints. Take h_e. This formula contains two copies of c_e in their entirety which will translate to $O(n)$ clauses after a Tseitin transform. The PB-constraint in PM is but one statement: i.e. $(w_0 + w_1 + ... \neq w'_0 + w'_1 + ...)$. These problems tend to get solved more efficiently by so-called "SAT-encodings" [24]. With SAT-encodings a PB-constraint is converted to a SAT formula and simply passed on to a SAT solver.

Encoding PB-constraints into CNF has been under study for many years. Given a PB-constraint of the form $\alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... \geq q$, first, the formula is preprocessed [25]. Negative coefficients are made positive, rounded, and minimized by manipulating the constraint and variable polarities. The summation with positive integer coefficients can then be mapped to CNF clauses in several ways.

One is to use a bit-adder network. These can be built by interleaving full-adders and half-adders to create a network with n inputs and $\log_2(n)$ outputs computing a binary representation of the sum of the input bits. Coefficients that are greater than one can be dealt with by replicating an input wire multiple times.

Another way to encode summation is sorter networks. An odd-even merge-sort network can be built using a series of 2-bit-to-2-bit sorter blocks. With n bits fed to its input, at the output all the ones will accumulate on one side. By probing specific indices in the n-bit output we can determine the number of ones in the input (sum) and check for equality/lower/greater conditions. Sorter networks are significantly larger than adder networks but may have better SAT performance due to their increased "implicativeness", i.e. the ease with which the SAT solver can learn implications in the CNF [25].

Lastly, one can use binary-decision-diagrams (BDDs) to encode PB-constraints. The dynamic programming procedure in [25] can be used to build a BDD for a PB-constraint. For instance, a cardinality constraint, stating that less than 4 bits in a 10-bit vector must be 1, will produce a BDD in which all paths that go through more than 4 one-edges will land in the 0-terminal of the BDD.

We show some circuit sizes produced by n-bit PB-constraints translation for the above three techniques in Table I. As can be seen, the BDD method, while producing compact circuits for the ≥ 1 case, explodes with larger values $\geq m$. The adder network produces smaller circuits than the sorter, both of which, unlike the BDD encoding, have circuit sizes that are independent of the right-hand side of the constraint. We use adders in our final experiments but allow for user configuration in our tool.

PowerSATeq. We can use PB-constraint encoding schemes to build QBD attacks. Our first attack called PowerSATeq, uses a SAT-encoding of $PM = (h_e(k_1, x, \delta x) \neq h_e(k_2, x, \delta x))$ to build a CNF SAT miter. This miter when satisfied will return a PDIP $(\hat{x}, \hat{\delta x})$. This is queried on the trace-oracle, and the observed power value $\hat{tr_i} = pwr(\hat{x}, \hat{\delta x})$ is appended back to PM as two equality conditions over k_1 and k_2: $(h_e(k_1, \hat{x}, \hat{\delta x}) = t_i) \wedge (h_e(k_2, \hat{x}, \hat{\delta x}) = t_i)$.

Note that satisfying PM will return PDIPs, and satisfying M will return functional DIPs. The interesting point here is that we can use these together. i.e. even though we are not making functional

	number of input variables									
method	4	8	16	32	64	128	256	512	1024	2048
adder	15	35	75	155	315	635	1275	2555	5115	10235
sorter	10	38	126	382	1086	2942	7678	19454	48126	116734
bdd-1	6	14	30	62	126	254	510	1022	2046	4094
bdd-1/4	6	18	60	216	816	3168	12480	49536	-	-

TABLE I: PB-constraint circuit sizes (gate-count) for different encoding. The encoder circuits are generated for a condition with n terms with all coefficients being 1. "bdd-1" denotes a BDD condition with (≥ 1) and "bdd-1/4" denotes ($\geq n/4$) condition as their right-hand-side.

queries, the functional DIP condition can serve as a termination test to the side-channel attack. If M becomes unsatisfiable during the PowerSATeq loop, this means that the circuit has been provably functionally deobfuscated from side-channel queries alone.

On the other hand, if PM becomes unsatisfiable, while M is still satisfiable, this means that the circuit's functionality cannot be further learned from side-channel queries of the above form. i.e. that the flip-rate of the circuit counted by h_e cannot identify a functionally correct key. This for instance can arise if the circuit $c_e(k, x)$ is flip-rate-invariant for a set of keys, some of which are functionally incorrect. In practice does this mean that such a c_e is side-channel-attack-*proof*? The answer is no in most cases. This is because h_e was a simple flip-rate counter and not the most precise representation of the power consumption of the circuit. Even in a circuit with invariant flip-rates (which some symmetric logic styles can achieve), the mismatch resulting from process variation or layout asymmetry means that there usually exists better power hypothesis functions h_e under which additional key information can be revealed.

The pseudo-code for PowerSATeq can be seen in Algorithm 2. Our baseline implementation of PowerSATeq breaks a 32-bit comparator against a precisely simulated h_e in 13 PDIPs and 5-20 seconds. The number of PDIPs is smaller than the 64 traces required in the routine described in Fig. 3. This is because power equality conditions are more key-restrictive than power difference conditions from Fig. 3. Power difference conditions are the basis for our next attack discussed in the following section.

Algorithm 2: Given power oracle $h_o \equiv h_e(k_*, x, \delta x)$ and the circuit c_e returns the best-possible key k that can be inferred from adaptive queries of h_o.

```
1  Function PowerSATeq($c_e$, $h_e$, $h_o \equiv h_e(k_*, x, \delta x)$):
2      $F \leftarrow true$
3      $M \leftarrow c_e(k_1, x_f) \neq c_e(k_2, x_f)$
4      $PM \leftarrow h_e(k_1, x, \delta x) \neq h_e(k_2, x, \delta x)$
5      while $F \wedge M \wedge PM$ is solvable do
6          $(\hat{x}, \hat{\delta x})$, $\hat{x_f}$, $\hat{k_1}$, $\hat{k_2}$ $\leftarrow$ PBSAT($F \wedge M \wedge PM$)
7          $\hat{tr} \leftarrow h_o(\hat{x}, \hat{\delta x})$
8          $F \leftarrow F \wedge (h_e(k_1, \hat{x}, \hat{\delta x}) = \hat{tr}) \wedge (h_e(k_2, \hat{x}, \hat{\delta x}) = \hat{tr})$
9      if $F$ is unsatisfiable then
10         return error in trace collection, or no correct key exists.
11     else if $M$ is unsatisfiable then
12         return $k_* \leftarrow$ PBSAT($F$) as functionally correct key
13     else
           // $PM$ must be unsatisfiable
14         return functionally correct key not identifiable under
           current power model. $k_\dagger \leftarrow$ PBSAT($F$) is the current
           best key hypothesis.
```

C. A Side-Channel SAT Attack with Difference Conditions

The main problem with PowerSATeq is that it requires recovering the precise number of bit-flips occurring in the circuit from a side-channel trace. While this may be possible in certain cases, this requires at least some profiling of the device beforehand, as to be able to map a collected trace to a precise h_e value in the presence of noise and uncertainty. This situation can be somewhat improved by quantizing the trace value into h_e range bins. This still requires that the upper/lower bound of each bin be somewhat profiled before the attack. In fact, we experimented with machine-learning classifiers for such a mapping of traces to bins with some success but we leave further exploring this to future work.

An approach that avoids the above issues altogether is to use the *difference* in the strength of different traces as the main observable in the attack. Instead of a miter that aims to find disagreement between the absolute side-channel value under two different keys, we can search for *pairs* of query points, for which the comparison direction of the two traces is different under different key hypotheses.

Formally we are looking for a pair of query points $(x, \delta x)$ and $(x', \delta x')$, where there exists two keys conforming to the current key condition k_1 and k_2, for which:

$$h_e(k_1, x, \delta x) > h_e(k_1, x', \delta x')$$

but for k_2 the opposite holds:

$$h_e(k_2, x, \delta x) \leq h_e(k_2, x', \delta x')$$

Or vice versa. The miter condition that captures this is:

$$PDM = \Big([h_e(k_1, x, \delta x) > h_e(k_1, x', \delta x')] \neq [h_e(k_2, x, \delta x) > h_e(k_2, x', \delta x')]\Big)$$

We call a pair of input/flip-patterns that satisfy this PDM condition power-differential discriminating-input-patterns (PDDIPs).

Obtaining Reliable PDDIPs. Querying a PDDIP is a much more straightforward enterprise than querying an equality PDIP. The attacker queries the trace oracle on $(x, \delta x)$ and on $(x', \delta x')$ obtaining tr and tr'. Subtracting the two traces from each other to obtain $\delta tr = tr - tr'$ and looking for the presence and direction of a spike in δtr can be a highly effective method in practice. We deem a query successful/accurate if $h_e(k_*, x, \delta x) > h_e(k_*, x', \delta x')$ leads to tr being stronger than tr' in the collected traces. This can break down if a) the added input-independent noise to tr and tr' is of a magnitude and direction that it flips the comparison result, or b) If the power model h_e is inaccurate enough for trace-comparison results on a particular point to be out of sync with power model comparisons. We explore in Section IV the success rate of different PDDIP query collection techniques on an FPGA hardware implementation of benchmark circuits. Obviously, for a simulated trace oracle which we use in part of our experiments, this accuracy is one.

PDDIP IO-constraints. Once a PDDIP query is made and the result is obtained, it will be added as a PB-constraint to the solver. The PDDIP IO-condition captures that for a pair of $(x, \delta x)$ $(x', \delta x')$ input/flip-patterns the direction of the comparison must be $\hat{\delta tr}$ for the correct key. i.e. $\forall k \in K_* \;\; [h_e(x, \delta x, k) > h_e(x', \delta x', k)] \oplus \hat{\delta tr} = 1$.

The attack continues this process in a loop. Similar to the case of PowerSATeq, an unsatisfiable PDM but still satisfiable M means that the circuit is not exact-learnable from power-difference queries with the current power model. Therefore, the power model must be adjusted to continue learning facts about the key. An unsatisfiable M on the other hand implies precise functional recovery, at which point further PDDIP collection is not needed.

The pseudo-code for the above attack that we call PowerSATdiff can be seen in Algorithm 3.

Algorithm 3: Given flip-count comparison oracle $h_{od}(x, \delta x, x', \delta x') \equiv h_e(k_*, x, \delta x) >^? h_e(k_*, x', \delta x')$, and the circuit c_e, returns the best-possible key k that can be inferred from adaptive queries of the comparison oracle h_{od}.

```
1  Function PowerSATdiff($c_e$, $h_e$, $h_{od}$):
2      $F \leftarrow true$
3      $M \leftarrow c_e(k_1, x_f) \neq c_e(k_2, x_f)$
4      $PDM \leftarrow [h_e(k_1, x, \delta x) > h_e(k_1, x', \delta x')] \neq$
         $[h_e(k_2, x, \delta x) > h_e(k_2, x', \delta x')]$
5      while $F \wedge M \wedge PDM$ is solvable do
6          $\{(\hat{x}, \hat{\delta x}), (\hat{x}', \hat{\delta x}')\}, \hat{x_f}, \hat{k_1}, \hat{k_2} \leftarrow$
             PBSAT($F \wedge M \wedge PDM$)
7          $dt \leftarrow h_{od}(\hat{x}, \hat{\delta x}, \hat{x}', \hat{\delta x}')$
8          $F \leftarrow F \wedge ([h_e(k_1, \hat{x}, \hat{\delta x}) > h_e(k_1, \hat{x}', \hat{\delta x}')] = dt)$
             $\wedge([h_e(k_2, \hat{x}, \hat{\delta x}) > h_e(k_2, \hat{x}', \hat{\delta x}')] = dt)$
9      if $F$ is unsatisfiable then
10         return error in trace collection, or no correct key exists.
11     else if $M$ is unsatisfiable then
12         return $k_* \leftarrow$ PBSAT($F$) as functionally correct key
13     else
           // $PDM$ must be unsatisfiable
14         return functionally correct key not identifiable under
             current power model. $k_\dagger \leftarrow$ PBSAT($F$) is the current
             best key hypothesis.
```

D. Speeding up PowerSAT Attacks

PB-SAT can be much more computationally intensive in practice than SAT. Even though SAT is an NP-complete problem, many SAT instances derived from practical circuits do not end up exploring the full exponentially-sized space of the problem. For instance take the c6288 ISCAS benchmark, which is a multiplier circuit. The baseline SAT attack on c6288 running on RLL [1] locked circuits will produce DIPs quite efficiently, however, oftentimes near the end of the attack as the attack has to prove that there remain no more DIPs, the SAT instance runtime can explode. This is because, while every call to the SAT solver in a SAT-based attack is an NP-complete call, the various problems can reduce to simple or hard problems respectively. A similar behavior but more often is observed with the PowerSAT attacks, where PDIP/PDDIPs are found efficiently in the beginning but as the attack nears the end and has to prove the non-existence of additional queries, runtime explosions occur. In this section, we discuss several novel techniques to improve runtime or extract partial information in the case of a failed attack.

Flip Hamming-weight Constraints. We observed an interesting behavior with respect to the PowerSAT attacks. Adding a constraint on the number of bit-flips in δx dramatically improved performance. δx in the normal case can take on any value, i.e. every input of the circuit can be toggling. However, if we apply a hamming-weight limit of z to δx, where z starts as 1, this limits the attack to revealing key information by toggling one input bit at a time. Our constructed comparator attack discussed with Fig. 3 followed a similar strategy. This limit can then be increased successively if the power miter becomes unsatisfiable at the current flip-weight constraint of z. Adding such a constraint brought down the runtime of PowerSATdiff against a 32-bit comparator from 100> seconds to less than 10 seconds.

Flip-capable Wire-Set. Wires in the circuit that are not in the transitive fanout of toggling input bits ($\delta x[i] = 1$) will not be able to toggle regardless of the key value. This means that we do not need to include these nets in the PB-constraint, i.e. we can use smaller adders/sorters. Since the adder/sorter networks are a main source of size complexity in the PowerSAT attacks, cutting down on their input size can greatly help the algorithm. The flip-weight constraint helps increase the number of such non-toggling nets in each query.

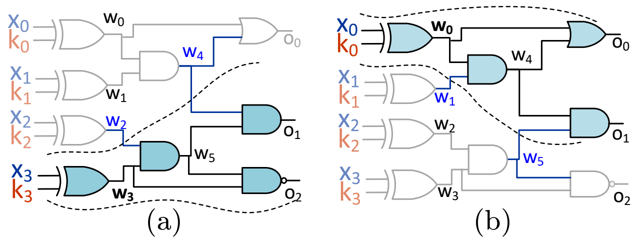

Fig. 4: Slicing procedure. In (a) wire w_3 is under study. In (b) wire w_0.

Circuit Simplification. Key-condition-crunching (KC2) introduced in [8] is the usage of circuit simplification techniques to improve the scalability of SAT attacks. Since PM, PMD, M, and all the other formulae in the SAT attack are represented with circuits, if one can simplify circuits, the resulting CNF can be compressed. This is a major theme used in modern model-checking and SAT-based verification problems. In fact, circuit rewrites and simplifications can sometimes prove properties without the need for SAT solving at all.

In our PowerSAT implementations, we follow the KC2 approach of trying to simplify any condition/circuit that arises in the attack using ABC with a size-minimal script. The power miter condition for instance is simplified before the start of the attack. It is not uncommon to observe compression rates above 50% when doing this.

The key-conditions or IO-constraints are also simplified. However, our implementation is different than baseline KC2 in an important way. Rather than performing simplification every d iterations, which can be time-consuming without immediate performance gains, we perform an "on-demand" simplification. We use a certain propagation budget for the SAT solver that grows by some factor (1.5-4), calling the solver successively and giving it a growing budget each time. If the budget grows beyond a certain point, this lets the algorithm know that a hard SAT instance may have been encountered. At this point, circuit simplification will kick in and simplify the IO-constraints and miter. The attack is then restarted with the simpler circuits.

Settled-Key/Cone Detection and Slicing. [4] first presented the idea of using SAT formula backbones, i.e. variables where forcing them to a particular value renders the formula unsatisfiable, to detect provably resolved partial keys. [8] used the equivalence between the two corresponding nodes in the miter ($c_e^w(k_1, x) \equiv^? c_e^w(k_2, x)$) to determine functionally resolved nodes in the circuit. Note that key inputs contained exclusively behind settled cones are provably settled keys. In our PowerSAT implementations, we use a stronger novel settled-key/cone detection approach. For a node w in $c_e(k, x)$, denoted by the cone $w = c_e^w(k, x)$, in conjunction with asking if c_e^w is equivalent in the miter, i.e. $c_e^w(k_1, x) \neq c_e^w(k_2, x)$ we ask if a difference between $c_e^w(k_1, x)$ and $c_e^w(k_2, x)$ can be propagated to the functional miter's output M. This can be captured by tying together all the side nodes on the cone from w to the output in the miter copies of c_e and checking the satisfiability of M.

The above procedure lends itself to a natural slicing strategy. When analyzing the settlement of wire w, we constrain the attack to operate only on the transitive fanin and fanout of w. Fig. 4 shows this. Per Fig. 4a, when wire w_3 is being studied for settlement only the input wire set $\{x_3\}$ is marked as active and allowed to flip. The transitive fanout cone of w_3 is marked as flip-capable. The side-wires w_2 and w_4 are

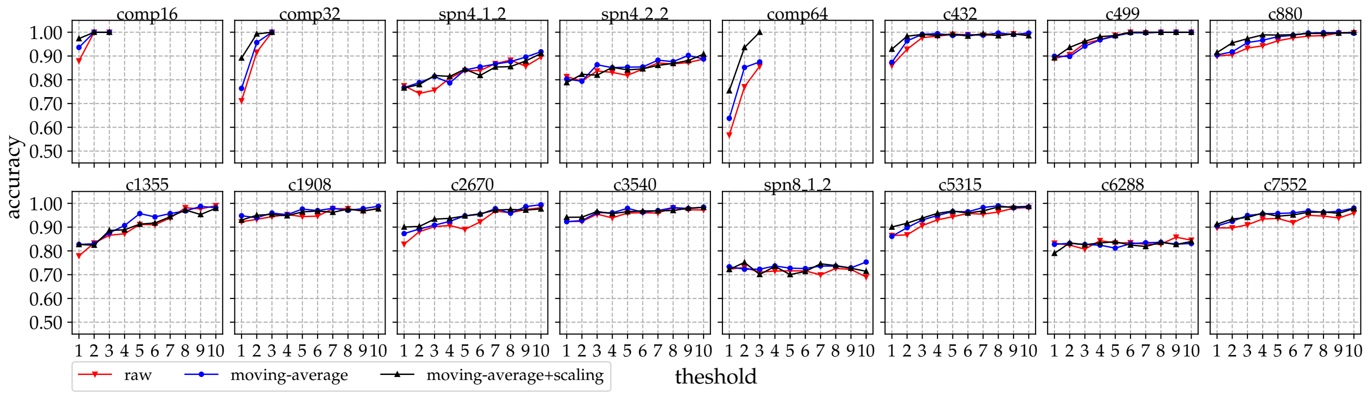

Fig. 5: The percentage of 500 test pairs out of 2000 collected traces, where the difference direction extracted using difference+average+scaling on the actual traces matches the direction suggested by the hamming-weight power model, The hamming model predictions are picked no closer to each other than the threshold value. As the power model values become more distant, the comparison agreement with the real traces increases. i.e. traces that are expected to be more distant, are easier to discern. Cut-offs represent a lack of sufficient threshold-distanced pairs in the trace set.

bench	#ins	#outs	#gates	bench	#ins	#outs	#gates
s298	17	20	119	c880	60	26	383
s386	13	13	159	s1238	32	32	509
c432	36	7	160	c1355	41	32	546
s349	24	26	161	c1908	33	25	880
s400	25	27	164	c2670	157	64	1193
s499	23	44	174	s3271	142	130	1573
c499	41	32	202	c3540	50	22	1669
s510	25	13	211	c5315	178	123	2307
s832	23	24	287	c6288	32	32	2416
s641	54	42	379	c7552	206	107	3512

TABLE II: ISCAS combinational and opened sequential benchmarks used

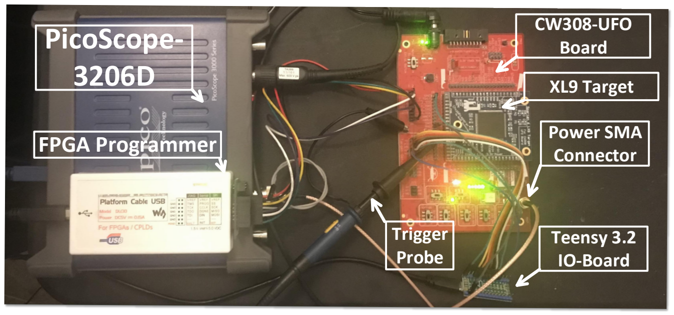

Fig. 6: Hardware setup for trace collection.

kept invariant between the miter copies. The attack loop then proceeds during which w_3 may become settled. This is reinforced in the solver and miter. The attack then moves to the next wire w_0 per Fig. 4b but with knowledge of settled w_3. Counter-examples discovered during this process can be simulated to disqualify non-settling wires from the procedure. This wire-by-wire deobfuscation process besides leading to higher simplification gains, showed improved runtime. Notably, it allowed for full recovery of 64-bit comparators in less than a second with both PowerSATeq and PowerSATdiff, which was the fastest we observed with these circuits. On larger circuits, there seemed to be some diminishing returns. This is a topic of our future work.

IV. Experiments

Benchmarks. We use a set of ISCAS combinational and sequential benchmark circuits shown in Table II.

Simulated Trace Oracle. We implemented PowerSATeq, and PowerSATdiff along with CPA in a C++ framework and ran them on benchmark circuits locked with RLL or AntiSAT+RLL [26] of different overhead levels against a simulated trace oracle (i.e. computing

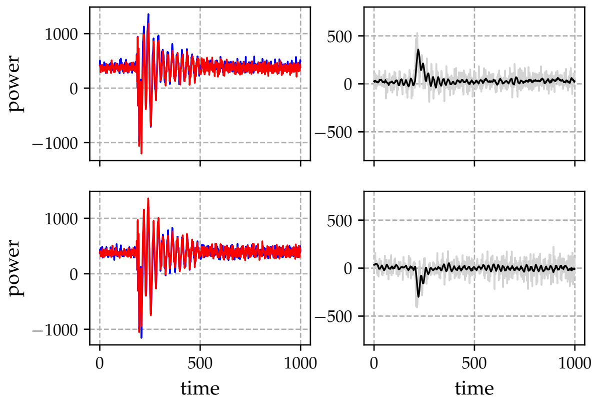

Fig. 7: Trace differences (red vs. blue). The upper row and lower row represent different comparison directions. The black line is the moving average of the subtraction of the traces which can be a better measure for detecting direction

$tr = h_e(k_*, x, \delta x)$). The tests were run in parallel (120 threads) on an AMD ThreadRipper 3990X 128-thread machine with 256GB of memory with each process getting 2GB of memory. No memory explosion was encountered. It was observed that the PowerSAT attacks once they hit hard SAT instances, giving them excessive extra time did not seem to produce obvious improvements. Hence the attack times were limited to 30 minutes, with provably settled keys and the key error rate recovered at the end. A generic CPA attack with a cone-based key-partitioning to 8-bit blocks with 500 traces was run as well. Table III and IV show the results. It can be seen that PowerSAT attacks can find provably correct (partial) keys with a small number of PDIP/PDDIPs. Also, the success of baseline CPA against simulated traces shows that locking is not automatically more secure than cryptographic hardware against side-channel attacks.

Hardware Trace Oracle. We use the chip-whisperer CW308-UFO board along with its XL9 Spartan-6 FPGA target. A Teensy 3.2 72Mhz board was used to shift into the FPGA input and flip pattern. A flip-clock is generated by the Teensy board that is XORed with inputs with $\delta x[i] = 1$. The `keep` synthesis directive was used to keep every net in the circuit alive on the FPGA. The flip-clock also serves as the trigger to a USB3.0 PicoScope 3206D oscilloscope. Using the batch collection function of the 3206D API, a set of 125 traces are read at once from the scope and averaged to produce a single low-noise reading. This framework on Ubuntu 20.04 achieved

params	RLL-5%										
method	#keys	CPA		PowerSATeq				PowerSATdiff			
bench		time	kerr	time	#pdi	kerr	#pkb	time	#pdi	kerr	#pkb
s298	6	0.81	E	4.5	5	E	6	330	5	E	6
s386	8	6	0.15	to	4	E	8	to	5	E	8
c432	9	6	E	0.42	4	E	9	3.9	11	E	9
s349	9	6	0.12	to	3	0.04	6	to	4	0.08	6
s400	9	7	0.07	to	5	0.04	6	to	4	0.04	6
s499	9	8	0.09	to	7	E	9	to	7	E	9
c499	11	9	0.16	to	5	E	11	to	4	0.06	6
s510	11	10	0.08	to	4	0.08	10	to	17	0.08	10
s832	15	21	E	230	9	E	15	860	21	E	15
s641	19	35	0.1	to	10	0.07	15	to	17	0.1	15
c880	20	35	0.15	to	8	0.08	16	to	20	0.12	16
s1238	26	68	0.03	to	15	E	26	to	29	E	26
c1355	28	64	0.16	to	10	0.09	23	to	22	0.12	23
c1908	45	200	0.16	to	20	0.04	42	to	26	0.26	1
c2670	60	453	0.06	to	27	0.05	57	to	35	0.04	9
s3271	79	659	0.11	to	37	0.05	67	to	98	0.06	67
c3540	84	791	0.22	to	3	0.33	0	to	10	0.34	0
c5315	116	1705	0.08	to	24	0.06	0	to	24	0.16	4
c6288	121	1567	0.4	to	2	0.48	0	to	4	0.48	0
c7552	176	to	0.15	to	1	0.14	0	to	3	0.19	0
s9234	280	to	0.19	to	36	0.17	9	to	19	0.23	0

params	RLL-10%										
method	#keys	CPA		PowerSATeq				PowerSATdiff			
bench		time	kerr	time	#pdi	kerr	#pkb	time	#pdi	kerr	#pkb
s298	12	6	0.05	to	7	E	12	to	10	E	12
s386	16	16	0.08	to	6	E	16	to	11	0.08	11
c432	17	15	E	to	6	E	17	7	19	E	17
s349	17	16	0.04	to	13	E	17	to	16	0.04	16
s400	17	18	0.19	to	7	0.07	10	to	9	0.15	10
s499	18	17	0.16	to	13	0.02	16	to	26	0.04	16
c499	21	20	0.22	to	7	0.06	14	to	18	0.16	14
s510	22	26	0.31	to	9	0.08	18	to	25	0.23	18
s832	29	45	0.14	to	14	0.04	27	to	44	0.08	27
s641	38	73	0.36	to	11	0.19	21	to	31	0.26	21
c880	39	84	0.22	to	18	0.12	32	to	37	0.08	32
s1238	51	145	0.11	to	16	E	51	to	77	0.03	50
c1355	55	134	0.41	to	21	0.25	46	to	14	0.41	0
c1908	89	541	0.42	to	4	0.26	0	to	17	0.23	0
c2670	120	896	0.17	to	15	0.1	0	to	21	0.18	2
s3271	158	1607	0.25	to	50	0.12	132	to	164	0.14	0
c3540	167	1704	0.4	to	2	0.38	0	to	9	0.41	1
c5315	231	to	0.3	to	3	0.26	0	to	2	0.2	0
c6288	242	to	0.5	to	1	0.44	0	to	2	0.49	0
c7552	352	to	0.25	to	2	0.25	0	to	1	0.28	0
s9234	560	to	0.35	to	4	0.33	139	to	4	0.34	0

params	RLL-15%										
method	#keys	CPA		PowerSATeq				PowerSATdiff			
bench		time	kerr	time	#pdi	kerr	#pkb	time	#pdi	kerr	#pkb
s298	18	12	0.15	to	8	0.1	15	to	27	0.1	15
s386	24	26	0.61	to	10	0.31	16	to	25	0.15	16
c432	25	24	0.19	61	9	E	25	14	31	E	25
s349	25	24	0.12	to	9	0.04	22	to	20	0.12	22
s400	25	25	0.21	to	12	0.04	20	to	27	0.04	20
s499	27	29	0.27	to	22	0.02	24	to	25	0.07	24
c499	31	37	0.25	to	7	0.12	21	to	26	0.12	21
s510	32	40	0.11	to	10	E	32	to	54	0.08	31
s832	44	82	0.17	to	16	0.04	40	to	55	0.04	40
s641	57	128	0.21	to	23	0.14	43	to	50	0.12	43
c880	58	134	0.37	to	20	0.15	47	to	16	0.29	1
s1238	77	238	0.39	to	23	0.12	68	to	27	0.25	5
c1355	82	236	0.28	to	23	0.14	77	to	12	0.24	0
c1908	133	697	0.47	to	4	0.26	0	to	10	0.33	0
c2670	179	1459	0.16	to	4	0.18	0	to	8	0.21	0
s3271	236	to	0.28	to	55	0.09	210	to	43	0.21	3
c3540	251	to	0.39	to	1	0.49	0	to	3	0.46	0
c5315	347	to	0.25	to	1	0.27	0	to	6	0.21	0
c6288	363	to	0.46	to	1	0.44	0	to	1	0.48	0
c7552	527	to	0.24	to	1	0.28	0	to	1	0.25	0
s9234	840	to	0.42	to	12	0.4	0	to	3	0.43	1

TABLE III: CPA with 500 traces vs PowerSAT attacks run on a precise simulated trace oracle against RLL. The runtime for each attack was set to 30 minutes. #key denotes the number of keys in the particular locked circuit after overhead-based obfuscation. #pdi is the number of queries made. kerr is the error rate (0-1) of the best key hypothesis at the end of the attack. An "E" in this column denotes that the recovered key was proven functionally correct by equivalence checking. #pkb is the number of key bits that were provably resolved through settled-key detection.

params	AntiSAT-20										
method	#keys	CPA		PowerSATeq				PowerSATdiff			
bench		time	kerr	time	#pdi	kerr	#pkb	time	#pdi	kerr	#pkb
s298	40	44	E	0.55	5	E	40	47	30	E	40
s386	40	53	E	1.3	7	E	40	to	21	E	40
c432	40	56	E	2.5	18	E	40	37	127	E	40
s349	40	50	E	1.3	6	E	40	65	29	E	40
s400	40	58	E	0.56	4	E	40	19	30	E	40
s499	40	61	E	2.6	21	E	40	24	108	E	40
c499	40	57	E	34	20	E	40	23	76	E	40
s510	40	67	E	3.2	23	E	40	21	74	E	40
s832	40	86	E	3.2	18	E	40	56	141	E	40
s641	40	87	E	4.0	19	E	40	to	164	E	40
c880	40	98	E	13	14	E	40	310	136	E	40
s1238	40	127	E	6	19	E	40	to	59	E	40
c1355	40	111	E	77	23	E	40	to	111	E	40
c1908	40	211	E	to	23	E	40	to	74	E	40
c2670	40	258	E	47	18	E	40	to	262	E	40
s3271	40	350	E	to	18	E	40	390	116	E	40
c3540	40	364	E	15	6	E	40	to	107	E	40
c5315	40	543	E	220	22	E	40	to	83	0.0	8
c6288	40	to	U	to	15	U	40	to	18	0.0	0
c7552	40	849	E	to	21	E	40	to	7	0.0	6
s9234	40	1325	E	1800	17	E	40	to	44	0.0	8

params	AntiSAT-20+RLL-5%										
method	#keys	CPA		PowerSATeq				PowerSATdiff			
bench		time	kerr	time	#pdi	kerr	#pkb	time	#pdi	kerr	#pkb
s298	46	44	E	1.6	12	E	46	380	57	E	46
s386	48	66	0.23	to	14	0.23	43	to	57	0.15	43
c432	49	64	E	3.1	21	E	49	47	81	E	49
s349	49	68	E	22	12	E	49	to	21	0.01	2
s400	49	72	E	4.7	16	E	49	to	53	0.02	46
s499	49	72	0.09	to	27	0.02	48	to	74	0.02	48
c499	51	74	0.09	to	21	E	51	to	164	0.09	48
s510	51	81	0.08	to	20	E	51	to	92	0.08	50
s832	55	112	0.04	to	23	0.04	54	to	69	0.04	54
s641	59	138	0.07	to	26	E	59	to	135	0.02	56
c880	60	143	0.08	to	28	E	60	to	71	0.08	58
s1238	66	209	0.06	to	25	0.03	64	to	85	0.03	64
c1355	68	192	0.16	to	24	0.06	66	to	35	0.14	1
c1908	85	399	0.27	to	28	E	85	to	19	0.22	0
c2670	100	730	0.06	to	74	0.02	96	to	42	0.03	9
s3271	119	1202	0.1	to	53	0.03	108	to	148	0.08	108
c3540	124	1202	0.1	to	6	0.22	0	to	9	0.3	0
c5315	156	to	0.12	to	67	0.06	144	to	10	0.17	0
c6288	161	to	0.38	to	1	0.34	0	to	4	0.42	0
c7552	216	to	0.15	to	5	0.17	0	to	3	0.21	0
s9234	320	to	0.23	to	12	0.19	0	to	74	0.21	1

params	AntiSAT-30+RLL-10%										
method	#keys	CPA		PowerSATeq				PowerSATdiff			
bench		time	kerr	time	#pdi	kerr	#pkb	time	#pdi	kerr	#pkb
s298	72	102	0.05	to	21	0.05	71	to	32	0.14	0
s386	76	122	0.31	to	20	0.15	70	to	42	0.28	2
c432	77	122	E	7	28	E	77	660	117	E	77
s349	77	127	0.15	to	23	0.15	71	to	42	0.03	11
s400	77	131	0.19	to	22	0.07	72	to	53	0.14	6
s499	78	129	0.16	to	32	0.07	75	to	72	0.07	75
c499	81	136	0.27	to	26	0.09	73	to	110	0.22	73
s510	82	162	0.08	to	18	E	82	to	23	0.12	0
s832	89	223	0.21	to	26	0.17	84	to	32	0.2	0
s641	38	73	0.19	to	18	0.14	26	to	29	0.14	26
c880	99	260	0.28	to	34	0.12	88	to	148	0.27	0
s1238	51	123	0.06	to	22	0.06	49	to	79	0.03	49
c1355	115	412	0.33	to	25	0.17	110	to	20	0.37	0
c1908	149	815	0.34	to	9	0.14	0	to	11	0.24	0
c2670	180	1653	0.21	to	3	0.14	0	to	15	0.22	2
s3271	158	1534	0.27	to	50	0.12	132	to	186	0.13	0
c3540	227	to	0.23	to	2	0.36	0	to	7	0.4	0
c5315	291	to	0.2	to	1	0.17	0	to	6	0.2	0
c6288	302	to	0.47	to	1	0.45	0	to	1	0.47	0
c7552	412	to	0.26	to	2	0.21	0	to	3	0.27	0
s9234	620	to	0.39	to	25	0.36	44	to	5	0.38	2

TABLE IV: This is an extension of Table III, with AntiSAT used in addition to RLL.

a close to 100 traces-per-second performance. An image of the setup can be seen in Fig. r6. The hardware/software for this platform is released at [27].

We tested the viability of differential queries (PDDIPs) on real hardware traces. Fig. 5 shows the success rate of trace comparison matching the power-model comparison for different model distances (threshold) on different benchmark circuits using either a) the raw difference or b) a moving average of it with a 20 point window (an example of which can be seen in Fig. 7), or c) moving average scaled by the value of the trace itself to dampen the effects of non-switching time points in the difference trace. As the distance between two power-model h_e values (threshold) grows, so does the trace difference, and the PDDIP success rate. Many circuits achieve 100% accuracy. This means that for the circuits with high difference accuracy, the PowerSATdiff attack on a simulated oracle can achieve the same results when run on the hardware, especially if a high threshold is set as long as this does not render the circuit unlearnable from PDDIPs. We placed the circuits from Fig. 5 with 100% difference accuracy in a hardware-in-the-loop setting and confirmed this. Note that the attack can be repeated multiple times with a different solver seed to improve accuracy. The 32-bit comparator was deobfuscated using 64 collected traces in less than a second using slicing.

V. Conclusion

We presented in this paper a pair of algorithms for circuit deobfuscation from power side-channels utilizing Pseudo-Boolean SAT solving. We demonstrated their performance against simulated and real hardware traces and provide a binary of the tool and the trace collection platform to the research community. Such algorithms besides showcasing that existing locking schemes do not automatically achieve side-channel resiliency, can be used in benign reverse engineering and hardware integrity checking. Improving the runtime and reliability of the PowerSAT attacks is an important topic of our future research.

References

[1] J. A. Roy, F. Koushanfar, and I. L. Markov. Epic: Ending piracy of integrated circuits. In *Proc. Design, Automation and Test in Europe*, DATE '08, pages 1069–1074, 2008.

[2] Ronald P Cocchi, James P Baukus, Lap Wai Chow, and B Jiangyun Wang. Circuit camouflage integration for hardware IP protection. In *Proc. IEEE/ACM Design Automation Conf.*, 2014.

[3] Kaushik Vaidyanathan, Bishnu P Das, Ekin Sumbul, Renzhi Liu, and Larry Pileggi. Building trusted ics using split fabrication. In *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2014.

[4] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, pages 137–143. IEEE, 2015.

[5] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

[6] Yuanqi Shen and Hai Zhou. Double dip: Re-evaluating security of logic encryption algorithms. In *Proc. IEEE Great Lakes Symp. on VLSI*, pages 179–184. ACM, 2017.

[7] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pages 49–56. IEEE, 2017.

[8] Kaveh Shamsi, Meng Li, David Z Pan, and Yier Jin. Kc2: Key-condition crunching for fast sequential circuit deobfuscation. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 534–539. IEEE, 2019.

[9] Sanjoy Dasgupta and John Langford. A tutorial on active learning. In *Proc. Int. Conf. on Machine Learning*, 2009.

[10] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[11] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[12] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 426–442. Springer, 2008.

[13] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[14] Abhrajit Sengupta, Bodhisatwa Mazumdar, Muhammad Yasin, and Ozgur Sinanoglu. Logic locking with provable security against power analysis attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(4):766–778, 2019.

[15] Abhishek Chakraborty, Yang Xie, and Ankur Srivastava. Template attack based deobfuscation of integrated circuits. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 41–44. IEEE, 2017.

[16] Kaveh Shamsi, Travis Meade, Meng Li, David Z Pan, and Yier Jin. On the approximation resiliency of logic locking and ic camouflaging schemes. *IEEE Transactions on Information Forensics and Security*, 14(2):347–359, 2019.

[17] Hai Zhou. A humble theory and application for logic encryption. Technical report, Cryptology ePrint Archive, Report 2017/696.(2017). https://eprint. iacr. org/2017/696, 2017.

[18] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin. AppSAT: Approximately deobfuscating integrated circuits. In *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, pages 46–51, 2017.

[19] Kaveh Shamsi, David Z Pan, and Yier Jin. On the impossibility of approximation-resilient circuit locking. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 161–170. IEEE, 2019.

[20] Mohamed El Massad, Siddharth Garg, and Mahesh Tripunitara. Reverse engineering camouflaged sequential circuits without scan access. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 33–40. IEEE, 2017.

[21] Abhrajit Sengupta, Mohammed Nabeel, Muhammad Yasin, and Ozgur Sinanoglu. Atpg-based cost-effective, secure logic locking. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2018.

[22] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[23] Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of satisfiability*, pages 695–733. IOS Press, 2009.

[24] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[25] Niklas Een, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pages 125–134. FMCAD Inc, 2011.

[26] Yang Xie and Ankur Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[27] Code release. http://www.bitbucket.com/kavehshm/scadec.