SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks

Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun and Avesta Sasan

ECE Department, George Mason University, Fairfax, USA, {kzamiria,hmardani,hhomayou,asasan}@gmu.edu

Abstract.
In this paper, we introduce the Satisfiability Modulo Theory (SMT) attack on obfuscated circuits. The proposed attack is the superset of Satisfiability (SAT) attack, with many additional features. It uses one or more theory solvers in addition to its internal SAT solver. For this reason, it is capable of modeling far more complex behaviors and could formulate much stronger attacks. In this paper, we illustrate that the use of theory solvers enables the SMT to carry attacks that are not possible by SAT formulated attacks. As an example of its capabilities, we use the SMT attack to break a recent obfuscation scheme that uses key values to alter delay properties (setup and hold time) of a circuit to remain SAT hard. Considering that the logic delay is not a Boolean logical property, the targeted obfuscation mechanism is not breakable by a SAT attack. However, in this paper, we illustrate that the proposed SMT attack, by deploying a simple graph theory solver, can model and break this obfuscation scheme in few minutes. We describe how the SMT attack could be used in one of four different attack modes: (1) We explain how SMT attack could be reduced to a SAT attack, (2) how the SMT attack could be carried out in Eager, and (3) Lazy approach, and finally (4) we introduce the Accelerated SMT (AccSMT) attack that offers significant speed-up to SAT attack. Additionally, we explain how AccSMT attack could be used as an approximate attack when facing SMT-Hard obfuscation schemes.

Keywords: Reverse Engineering, Logic Locking, Boolean Satisfiability, Satisfiability Modulo Theory, SMT, Theory Solver.

1 Introduction

The cost of building a new semiconductor fab was estimated to be $5.0 billion in 2015, with large recurring maintenance costs [Yeh12][GFT13], and sharply increases as technology migrates to smaller nodes. To reduce the fabrication cost, most of the manufacturing and fabrication is pushed offshore [Yeh12]. However, many of the offshore fabrication facilities are considered to be untrusted. Manufacturing in untrusted foundries has raised concern over potential attacks in the manufacturing supply chain, with an intimate knowledge of the fabrication process, the ability to modify and expand the design prior to production, and an unavoidable access to the fabricated chips during testing. Accordingly, fabrication

To counter these threats, various hardware *design-for-trust* techniques have been proposed, including *watermarking*, *IC metering*, *split manufacturing*, *IC camouflaging*, and *logic locking* [AK07, KLMS+98, KH16, IEGT13, CCBW13, RKM10]. The watermarking and IC metering techniques are passive protection models that could be used to detect overproduction or illegal copies, however, they cannot prevent IP theft or overproduction. The *Camouflaging* techniques use logic gates (or other physical structures such as dummy vias) with high structural similarity, that are indistinguishable from one another to protect against reverse engineering. However, camouflaging is only effective against post-manufacturing attempt(s) of reverse engineering, while it provides no limitations against a foundry's attempt at reverse engineering, as a foundry has access to all masking layers and is not trapped by structural ambiguity for being able to logically extract a netlist. The obfuscation (*logic locking*) [RKM10] on the other hand, introduce limited programmability by inserting key programmable gates to hide or lock the functionality. By using obfuscation, the target chip produces the correct output only when the key inputs are correct. The purpose of obfuscation is to protect against RE at an untrusted foundry. By using obfuscation, even by having all mask information, the attacker cannot generate the correct functionality of a circuit without the correct key values, and such key values are not shared with the manufacturer.

Shortly after the introduction of first published obfuscation schemes, a new and powerful attack based on Boolean Satisfiability (SAT) was formulated and revealed [EMGT15, SRM15]. In this attack model, the attacker has access to a reverse engineered but obfuscated netlist, and a functional and unlocked chip. Using this attack model, the formulated Boolean Satisfiability Attack (SAT Attack) can effectively break all previously proposed logic encryption techniques, including random insertion (RLL), fault-analysis (FLL), interference-based logic locking (SLL), and logic barriers [RKM10, BTZ10, RPSK12, RZZ+15, WSM+16]. The SAT solver iteratively eliminates sets of incorrect keys and finds the correct key within a small time, and unlike Brute force attack that requires attack time exponential with respect to the number of inputs, its execution time grows almost polynomially. Existing SAT attack, which can be modeled with query-by-disagreement (QBD) or uncertainty-sampling, minimizes the number of queries (inputs) required to learn (deobfuscate) the target function (obfuscated logic). Also, SAT attack terminates when no more disagreeing inputs can be found, at which time the attack guarantees to find the correct key. However, to defend against powerful SAT attacks, different obfuscation schemes have been proposed, such as SARLock and Anti-SAT [KAG+18, YMRS16, XS16]. However, further research illustrated that some of these locking schemes are vulnerable to other types of attacks such as Signal Probability Skew (SPS) and removal attacks [YMSR17b].

In addition, introducing approximate-based attacks, such as AppSAT [SLM+17a] or Double-DIP [SZ17] worsens the problem. Unlike the existing SAT attack, which needs exact learning model, approximate-based attacks can be modeled using approximate learning problems, such as the probably-approximately-correct (PAC) setting [EHKV89, SPP+18]. Based on the PAC model, an attack A, with a probability of λ, will provide an ϵ-approximation (approximately correct) of the target function (obfuscated logic). Note that, an ϵ-approximation of the target function is a function with only $\%\epsilon$ ($\epsilon \in O(\frac{1}{2^n})$) disagreement with correctly unlocked circuit. Accordingly, the approximate SAT attacks can find an approximate key which produces a very small error ($\%\epsilon$) in the behavior of the unlocked circuit in comparison with a correctly unlocked circuit. The approximate attacks are shown to effectively find an approximate key for SAT-resilient defenses including SARLock [YMRS16], and Anti-SAT [XS16]. Furthermore, Bypass attack [XSTF17] is

Table 1: List of Abbreviations Frequently used in this Paper.

Term	Description	Term	Description
SMT	Satisfiability Modulo Theory	**SAT**	Boolean Satisfiability
CNF	Conjunctive Normal Form	**DI**	Discriminating Input
DIP	Discriminating Input Pattern	**DLL**	Delay + Logic Locking
TDK	Tunable Delay Key-gate	**RLL**	Random Logic Locking
TDB	Tunable Delay Buffer	**KPG**	Key Programmable Gates
KPC	Key Programmable Circuit	**KDC**	Key Differentiating Circuit
DIVC	DI Validation Circuit	**HD**	Hamming Distance
SH	SAT-Hard Obfuscation	**HC**	High Corruption Obfuscation
TO	Timeout	**RLL**	Random Logic Locking

even small error in the behavior of the unlocked circuit by approximate key, and behave completely the same compared to correctly unlocked chip.

The SAT attack benefits from the Directed Acyclic Graph (DAG) based nature of input netlist and the ability of SAT-attack to logically model the obfuscation into a satisfiability problem. To counter the SAT attack, recently some design obfuscation schemes have been proposed to violate these assumptions. For instance, in the approach adopted in [RMKS18], the DAG nature of netlist is altered by introducing cycles into the netlist for the purpose of trapping a SAT attack. Another example is the approach adopted in [XS17], where the obfuscation, in addition to logical properties of the netlist, targets the setup and hold properties (timing properties) of the circuit as a locking mechanism. Considering that setup and hold time are not logical properties, they cannot be translated into CNF statements for formulating a SAT attack. However, in this paper, we illustrate that even using these non-logical properties for obfuscation, does not increase the security of an obfuscated netlist, indicating the need for further study and exploration in this domain to generate obfuscation schemes with provable security.

The contributions of this paper is as follows:

I) Introducing SMT Attack: We present Satisfiability Modulo Theory (SMT)-based attack on obfuscated circuits, that expands the capabilities of previously proposed SAT attack by assigning theory solvers to monitor the behavioral and non-functional properties of the obfuscated circuit. To illustrate the capabilities of SMT attack, we use an SMT solver and invoke a graph-theory solver to break the logic and timing obfuscation scheme introduced in [XS17].

II) Introducing Accelerated SMT Attack: We illustrate that by adopting theory solvers, not only we can observe, monitor and learn from the non-functional behavior of an IC during an attack, but also we can significantly reduce the attack time in specific obfuscation schemes. In this paper we use a *BitVector* theory solver to reduce the execution time of SMT attack (compared to a SAT attack) by means of finding discriminating inputs that have much higher pruning power over the SMT solver's decision tree.

III) Approximate-based SMT Attack: We further equip our proposed Accelerated SMT attack with logistic to discriminate between SAT-hard and non-SAT-hard obfuscations if a hybrid-obfuscation scheme contains both. Using this approach, we quickly find the non-SAT-hard obfuscation keys, detect the SAT-hard solution and without spending exponential time, we exit and generate the approximate key. In addition, unlike previous approximate SAT attacks, we can guarantee that the approximately unlocked circuit at most have d bits of difference in the worst case (for any given input) with the correctly locked circuit, with d being 1 for all previously proposed SAT-hard solutions.

IV) Open Source SMT-Attack Tool: For enabling the community to have the capability of reproducing the results, we release the SMT-Attack open-source tool with this paper in the hope of, and to help with, finding SMT and SAT hard obfuscation solutions that could be tested and verified using this open-source tool [AKHS18].

The rest of the paper is organized as follows: Section 2 presents the background and

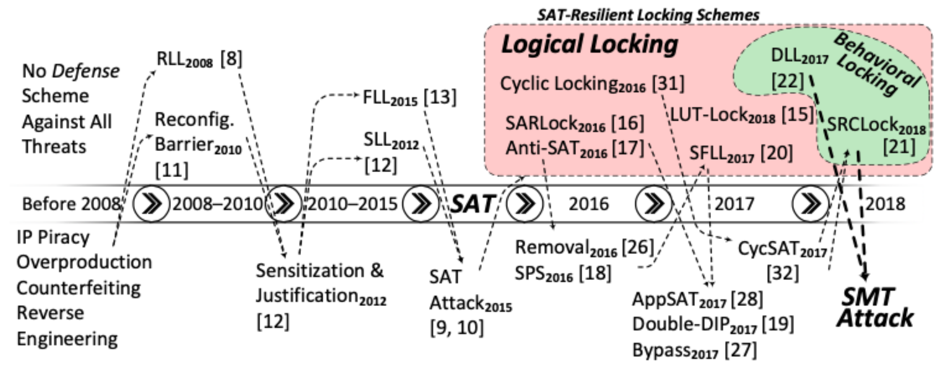


Figure 1: 10-Year History of Logic Obfuscation, the Defense Solutions, and Attack Models.

The SMT attack on obfuscated circuits and study four different modes of the SMT attack. Section 7 captures our experimental and simulation results. Finally, Section 8 concludes the paper.

2 Background

2.1 Boolean Logic Obfuscation

Logic locking and netlist obfuscation schemes introduce limited programmability into a netlist by means of inserting additional key programmable gates at design time. After fabrication, the functionality of the IC is programmed by loading the correct key-values. The key-inputs could be stored in and driven by an on-chip tamper-proof memory [TSŠ+06]. The purpose of inserting key-gates is protecting the IC design from untrusted foundries. Since the functionality of a design is locked with a secret key, the attacker cannot learn the functionality of the obfuscated netlist after reverse engineering. Logic locking and obfuscation schemes vary in terms of the usage of different key-gates types and key-gates insertion policies [RKK14, MKP08]. For combinational circuits, logic locking can be classified based on key-gates types to different categories. XOR/XNOR based logic locking [RPSK12, RKM10], MUX based logic locking, and LUT based logic locking [WSM+16, KAG+18] are the most common mechanisms. Also, there are different algorithms for inserting the key-gates in the circuit. Some of these policies include random insertion (RLL), fault-analysis (FLL) insertion, and interference-based logic locking (SLL) algorithms, SARLock, Anti-SAT , etc. [RKM10, RPSK12, RZZ+15, YMRS16, XS16].

Fig. 1 captures the evolving history of obfuscation defense schemes and attack formulations since the year 2008 to the current date. After introduction of SAT attack, in 2015 in [SRM15, EMGT15], as illustrated in this figure, researchers proposed various mechanisms for building SAT hard obfuscation solutions. However, many of such obfuscation schemes were later broken using newer attacks like as SPS, removal, bypass, and AppSAT [YMSR17b, SZ17, YMSR17a, XSTF17, SLM+17a], making the current defense schemes unreliable. After 2017, a new breed of obfuscation schemes instead of building logical obfuscation schemes has been introduced, relied on breaking the SAT assumptions for building SAT hard solutions without having the vulnerabilities of the previous SAT-Hard solution. For example, Cyclic obfuscation [SLM+17b] and its improved defense, the SRCLock[RMKS18], by introducing cycles into netlist break the SAT model as the netlist can no longer be represented by a Directed Acyclic Graph (DAG). Alternatively, the Delay Logic Locking (DLL) [XS17] extends the reach of obfuscation beyond logic and model the non-logical and behavioral aspects of a circuit operation.

2.2 Behavioral logical obfuscation

As previously discussed, the logic-based obfuscation schemes that rely on extending the Boolean behavior of a circuit can be broken by at least one of the state-of-the-art attacks, including SAT, SPS, removal, bypass, and AppSAT [EMGT15, SRM15, YMSR17b, SZ17, YMSR17a, XSTF17, SLM$^+$17a]. Hence, recent researches have been focused on obfuscation schemes that fundamentally violate the assumptions of these attacks with respect to the nature of obfuscated circuit, or use non-logical properties of a netlist to obfuscate its behavior [RMKS18, XS17, SLM$^+$17b].

For lack of EDA tool support and limited knowledge in designing cyclic Boolean logic, most of all netlists designed and fabricated today are acyclic. One of the first attempts to break the state of the art attacks, including SAT attack, was proposed in [SLM$^+$17b] which suggested using *cycles* in combinational circuits, and illustrated that use of cycles results in either a SAT solver being trapped, or it generates incorrect key even after timely termination. This obfuscation scheme, however was shortly after broken by CycSAT attack in [ZJK17]. In the CycSAT attack, the netlist is first pre-processed based upon which a set of constraining clauses are generated. The CycSAT attack then uses these constraining clauses, in the original SAT attack, allowing the SAT solver to effectively open the cycles without being trapped, or incorrectly terminated. However, the limitation of [SLM$^+$17b] was addressed in SRCLock [RMKS18] to prevent a pre-processor from extracting all needed constraints from a cyclically locked circuit. SRCLock focuses on building an exponential relation between the number of inserted feedbacks and number of generated cycles by means of creating *super cycles*.

The second obfuscation of interest to this paper is the logic and timing obfuscation scheme in [XS17]. In this obfuscation scheme, the delay properties of a circuit are obfuscated with the ultimate goal of introducing setup and hold violation if the correct key is not applied. In this case, the obfuscation, in addition to the logical behavior of the netlist, attempts to change its behavioral (timing) properties. Considering that timing is not translatable to CNF, the SAT solver remains oblivious to the keys used for timing obfuscation. Hence using a SAT attack to deobfuscated this circuit, result in a discovery of all keys used for logic obfuscation, but random assignment to all keys used for timing obfuscation and the circuit remains locked.

In this paper, we construct an attack based on Satisfiability Module Theory solvers, and illustrate that the capability of this attack goes far beyond that of SAT attacks. More precisely, with specific formulation, we illustrate that SMT attack on obfuscated circuits could be significantly faster and more efficient compared to SAT attacks on Boolean logic obfuscation. Additionally, it could be used to attack behavioral logic obfuscation schemes, which is not possible by a pure SAT-based attack. To illustrate the second point, we attack and break the timing-logic obfuscation scheme in [XS17], based on which we generalize and illustrate how other similar SMT attacks could be formulated.

3 Attack Model

The SMT attack is an ***oracle-guided attack***. We assume that the attacker has the reverse engineered but obfuscated netlist and a functional IC (*oracle*) that is unlocked. The attacker can query the oracle with any stimuli i, and observe its output o. The purpose of the attack is to find the key inputs, that make the obfuscated netlist logically equivalent

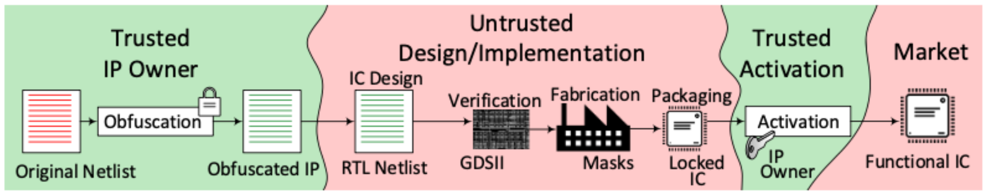


Figure 2: ASIC Design Flow Integrated with Obfuscation/Activation.

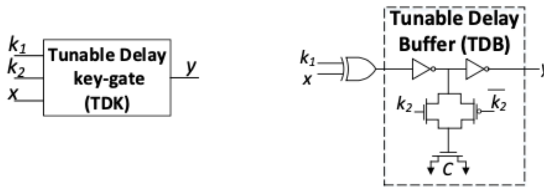
untrusted regime, attacker is able to obtain the obfuscated (locked) netlist from (1) the IC design, or by reverse engineering the (2) synthesis/implementation (layout), (3) mask, or (4) a manufactured IC. In addition, the attacker is able to buy the correctly unlocked (activated) IC in the open market. Consequently, the attacker can apply arbitrary input to activated IC and observe its corresponding output.

4 Limitation of SAT Attack

A SAT attack works perfectly fine if the logic obfuscation is of Boolean nature. This is because any Boolean logic could be easily transformed into its Conjunctive Normal Form (CNF) and be converted into a satisfiability assignment problem. But in case of Behavioral logic obfuscation, the locking mechanism is designed to control aspects of circuit operations that could not be translated to CNF as required by a SAT solver. The delay-locking (DLL) scheme proposed in [XS17], cyclic-based obfuscation presented in [SLM+17b], and SRCLock [RMKS18] are good instances of such locking mechanism. For the purpose of locking, DLL uses a tunable delay key-gate (TDK) which is illustrated in Fig. 3. TDK consists of a conventional key-gate (XOR/XNOR) with a tunable delay buffer (TDB). The capacitive load of the buffer is controlled by a transmission gate, where activating the transmission gate increases the wire load capacitance of the internal wire, resulting in larger TDK propagation delay. Hence, the functionality and propagation delay of a TDK, both, depends on the value of its key-inputs.

In DLL, the TDK cells are used to control the *setup* and *hold* time violations such that only one sequence of activation keys guarantees that circuit operates with no violation. To apply the DLL, a design is first altered such that most timing paths are balanced to be sensitive with respect to small changes in the path delay, such that a small variation in delay causes setup or hold violations. This is achieved by means of carefully engineering the clock skew, cell sizing, and V_{th} swapping. Then the TDK cells are inserted in the common portions of setup and hold critical paths, such that attempting to only fix setup causes hold violation, and attempting to fix hold causes setup violation with the exception of one sequence of correctly configured TDK keys that assures all timing paths meet both setup and hold check timing constraints. Considering that the delay is not a logical behavior, the TDK cell behavior could not be completely captured by CNF, hence the delay locking is not directly attackable by a SAT attack. In [XS17] it was illustrated that even a mixed integer linear programming (MILP) based attack has up to 39% timing violation ratio (TVR). However, as we will illustrate in this paper, by employing an SMT attack and by instantiating an integrated graph theory solver along with its SAT engine, we could find the keys to this obfuscation problem in few minutes.

5 SMT Solver



(a) (b)

k_1k_2	f(x)	Delay
00	$y = x$	d_0
01	$y = x$	d_1
10	$y = \overline{x}$	d_0
11	$y = \overline{x}$	d_1

(c)

Figure 3: Overall Structure of Tunable Delay Key-Gate (TDK) used in Delay Logic Locking (DLL)[XS17]. (a) TDK Gate with Two Keys, one for Logic and one for Delay Locking. (b) Layout of TDK with a Tunable Delay Buffer (TDB), (c) Functionality and Delay of the TDK

5.1 SMT Usage and Capabilities

A Satisfiability Modulo Theory (SMT) is used to solve a decision problem while honoring constraints that could be expressed using first-order theories such as equality, reasoning, arithmetic, graph-based deduction, etc. Hence, it could be considered as a solver for a broad set of problems that could be categorized as Constraint Satisfaction Problems (CSP), which is a superset of Boolean Satisfiability Problems (BSP) that are solvable by SAT solvers. Additionally, the ability to express theories such as inequality (e.g. $3x + y < z$) provides a much richer Application Programming Interface (API) to the end user to define a problem compared to that of a SAT solver.

In general, there are two different approaches for solving an SMT problem. The first approach is based on translating the problem into a Boolean SAT instances denoted by *Eager* approach; In this approach the existing Boolean SAT solvers are used as is. However, the SMT solver has to work a lot harder for solving some problems that are otherwise very obvious (e.g. for checking the equivalence of two 32-bit values). However, by deploying a theory solver, this could be achieved in no time. For this reason, many SMT solvers follow another approach which referred to as the *Lazy* Approach. The Lazy approach integrates the Boolean satisfiability solvers, which are based on the Davis-Putnam-Logemann-Loveland (DPLL) in modern SAT, and theory solvers that decide the satisfiability of formulas over specific theories. Each theory solver provides two capabilities: (1) theory propagation among various theory solvers for checking possible conflicts on partial assignments, and (2) clause learning result of which is shared by the SAT solver to speed-up pruning the decision tree. Additionally, since several applications of SMT deal with formulas involving two or more theories at ones, modern SMT solvers provide the capability of combining theory solvers using Nelson-Oppen [NO80] or Shostak [Sho82] method to support a more expressive language. In combining theory solvers, if two theories Γ_1 and Γ_2 are both defined axiomatically, their combination can simply be defined as the theory axiomatized by union of the axioms of the two theories, Γ_1 and Γ_2. For example, Consider Γ_1 and Γ_2 are two different theories, it is possible to define $\Gamma_1 \oplus \Gamma_2$ as a combined theory of Γ_1 and Γ_2, where $\Gamma_1 \oplus \Gamma_2$ is the set of all models that satisfy $\Gamma_1 \cup \Gamma_2$. This is adequate if the signatures of the two theories are disjoint. Otherwise, if Γ_1 and Γ_2 have symbols in common, one has to consider whether a shared function symbol is meant to stand for the same function in each theory or not. In the latter case a proper signature renaming must be applied to the theories before taking the union of their axioms. in [TR03] they have described general conditions for the combination of theories that may have symbols in common. The ability to combine theory solvers proves extremely useful when dealing with applications such as model checking and predicate abstraction-based model check in which we need to check the satisfiability of formulas over several data types.

Theories are defined as classes of models with the same signature. More precisely, a used within an SMT solver should have the following properties [BT18]: *(1) Model Generation*: theory solver should be able to produce a Γ-model of the problem description μ. *(2) Conflict Set Generation*: when the theory solver reaches inconsistency, it should be able to produce a subset η of μ which has caused the inconsistency. The subset η is referred to as *theory conflict*. *(3) Incrementality*: The Γ-solver should be able to save and keep its status across invocation calls to avoid recomputation. *(4) Backtrackability*: it is important for theory solver to has the ability to undo the step if it is needed. Equality with Uninterpreted Functions (EUF), linear real arithmetic (LRA), linear integer arithmetic (LIA), Mixed Integer and Real Arithmetic, Difference Logic, Bit Vectors, Arrays, etc. are the examples of theories commonly used in SMT.

In this paper, we use an SMT solver and formulate some attacks against specific obfuscated circuits, illustrating the power of adapting various theory solvers for extending the capabilities of attack by constraining and monitoring non-logical properties of a netlist. For this purpose, and to illustrate that SMT attack is a super-set to the SAT attack, we first illustrate that the original SAT attack against obfuscated circuits could be effectively formulated using an SMT solver, resulting in similar performance. Then we illustrate how the SMT solver could be used to attack logic obfuscation problems out of the reach of pure SAT attacks, and for that purpose we break the logic and timing obfuscation in [XS17] which is not possible by a pure SAT attack. We illustrate that this attack could be achieved using both Eager and Lazy approach of SMT attack. Then we illustrate how the SMT attack could become significantly more efficient than a SAT attack by adopting the capabilities of theory solvers like *BitVector*, and formulate an accelerated SMT attack, that requires substantially smaller iterations and runtime compared to a SAT attack against specific obfuscation schemes. In addition, we formulate the accelerated SMT attack to be capable of approximate attacks.

6 SMT Attack

When building an SMT attack on obfuscated circuits, as illustrated in Fig. 4, the SMT attack could be invoked with any number and combination of theory solvers, and a SAT solver. In order to use the SMT solver to formulate an attack, few preliminary steps should be taken. The first step is to make a minor modification to an extracted netlist after reverse engineering, providing the capability of testing various behaviors of the obfuscated circuit to the SAT or SMT solver. The transformation is simply replacing the obfuscated cells with their equivalent Key Programmable Gates (KPG). A KPG performs the same function as the obfuscated cell, however, it allows building a key controlled representation of the logical behavior of the obfuscated cell for the purpose of logical-model building. Fig. 5 captures the KPG translation gates for each type of the gates that have previously used in recent literature for the purpose of obfuscation. For example, when attacking a camouflaged cell that could be either an AND gate or an XOR gate, it is replaced with its KPG which is simply a MUX with each of its input tied to one of the camouflaged cell possibilities. The function performing the KPG replacement in the algorithms described in this paper is ***ReplaceKPG(N$_{obf}$)*** that replaces all obfuscated cells in an obfuscated module with their KPGs equivalent based on translation table in Fig. 5.

When using an SMT solver, before invoking a theory solver, the input model or input behavior should be translated to a model μ which is understood by that theory solver. As illustrated in Fig. 4, the translation step may be different for each theory solver used. As an example, to break the Delay Logic Locking in [XS17], we use a graph theory solver and translate the obfuscated netlist to a graph model that is understood by the graph-theory

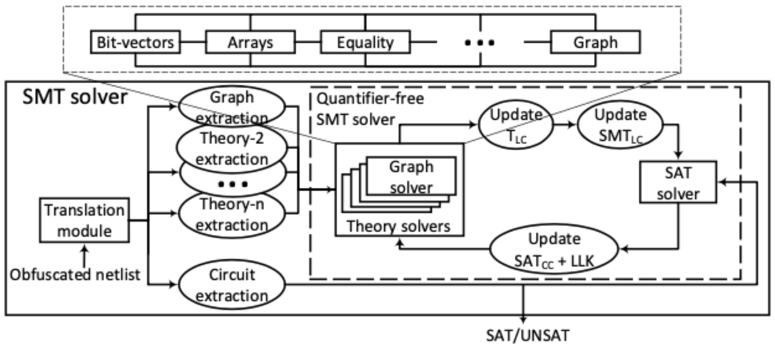


Figure 4: Overall Architecture of SMT Attack for Circuit Deobfuscation. In SMT attack multiple theory solvers could be simultaneously invoked along with a SAT solver to model complex attack scenarios.

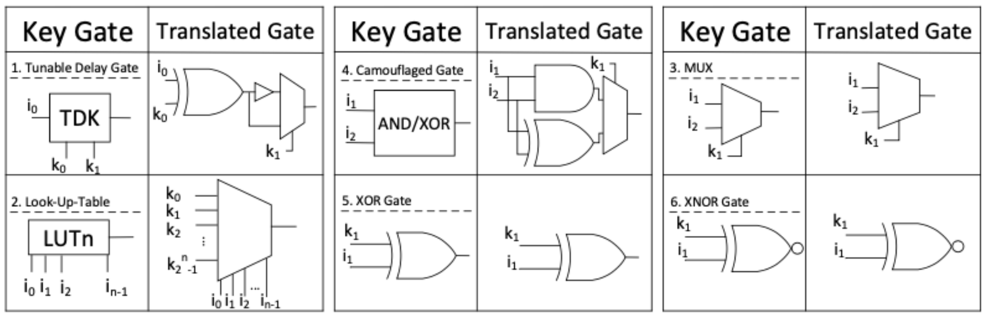


Figure 5: Translation Table for Converting the Camouflaging and Obfuscation Gates to Key Programmable Gates (KPG).

in our graph translation routine, that annotate each edge with the logical effort needed to drive that edge as a measure of its delay. We could alternatively use a second theory solver to capture the static timing of the netlist and exchange information with the graph theory solver for more accurate results. The final step before invoking the SMT/SAT attack is the translation of the netlist under attack into its CNF form as described in [SRM15].

After building model μ for each Theory and SAT solver, the SMT attack is formulated based on the flow of information exchange between theory and SAT solver. In General, the formulation of the SAT portion of SMT solver is similar to that of pure SAT attack as described in [SRM15]. However, in addition to the SAT solver, each theory solvers is then tuned by declaration of theory constraint. At this stage, invoking the SMT solver returns a satisfiable assignment and a list of learned theory and conflict clauses for theory solver and SAT solver respectively. The SMT attack is then achieved by composing the correct control flow for invocation of theory and SAT solver(s), and by managing the intermediate sequence of CNF-based information exchange. The general flow of information in an SMT formulated problem, including that of SMT attack, is illustrated in Fig. 4.

6.1 Attack Mode 1: SMT reduced to SAT Attack

As was mentioned previously, the SAT attack finds a functionally correct key for an obfuscated circuit by checking a small subset of all input patterns, hence removing the need for brute-force testing of all input patterns. Considering that SMT solver is a superset of SAT solver and contains a SAT solver, any attack formulated for SAT could be similarly formulated for an SMT solver.

Algorithm 1 SMT Reduced to SAT Attack in [EMGT15, SRM15]

```
 1: function SAT_ATTACK(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
 2:     $KPC \leftarrow$ ReplaceKPG($N_{obf}$);
 3:     $C(X, K, Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
 4:     $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
 5:     $SCKVC = TRUE$;
 6:     $SATC = KDC \wedge SCKVC$;
 7:     $LC = TRUE$;                                              ▷ Learned Clauses
 8:     $SMT_{LC} \leftarrow SATC$;
 9:     while $(((X_{DI}, K_1, K_2, CC) \leftarrow SMT.Solve(SMT_{LC})) = TRUE)$ do
10:         $Y_f \leftarrow C_{org}(X_{DI})$;
11:         $DIVC = C(X_{DI}, K_1, Y_f) \wedge C(X_{DI}, K_2, Y_f)$;
12:         $SCKVC = SCKVC \wedge DIVC$;
13:         $LC = LC \wedge CC$
14:         $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
15:     $Key \leftarrow SMT.Solve(SMT_{LC})$;
```

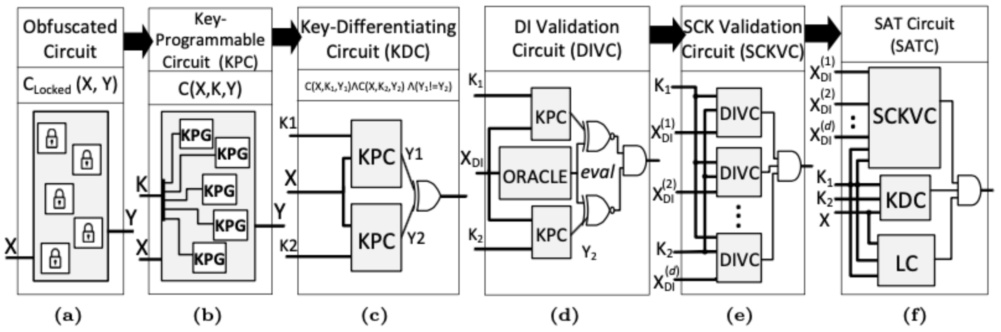


Figure 6: (a) Obfuscated Circuit (b) Translated to KPC (c) Generating KDC (d) Validation Circuit for Discriminating Inputs (DIVC) (e) Generating SKCVC (f) constructing The final Satisfiability Circuit (SATC) for finding a new DI.

The SAT attack in Alg. 1 follows the steps illustrated in Fig. 6. In this algorithm, the obfuscated gates are first replaced with key programmable gates (KPG) to create the Key Programmable Circuit (KPC). Then the CNF representation of the circuit is generated. Two KPCs are then used to generate a Key Differentiating Circuit (KDC). The KDC receives an input and two different keys and determines whether they generate the same output or not. The KDC is then used as the first SMT satisfiability problem represented by SMT_{LC} for the first invocation of SMT solver. Calling the SMT *solve* function on the posed formula then return an assignment for keys K_1, K_2, and the discriminating input X_{DI} such that the formulated SMT_{LC} is satisfied. In addition, the SMT solver returns a list of learned Conflict Clauses (CC). In line 10, the correct output (Y_f) for the discriminating input X_{DI} is found. In the next step, the SMT formula needs to be updated to use the discriminating input and learned clauses to further constrain the satisfiability problem. This is done in multiple steps. In line 11, the discriminating input found in the current iteration is used to create a Discriminating Input Validation Circuit (DIVC) which is illustrated in Fig. 6(d). The DIVC circuits formed at each iteration are ANDed together to create a circuit that checks the correctness of a key for all previously found discriminating inputs. This circuit is referred to as Set of Correct Key Validation Circuit (SCKVC). In line 13, the currently found Conflict Clauses are added to the set of previously found Learned Clauses (LC). Note that this step is done implicitly for SMT is a stateful solver. Finally, in line 14 the SMT satisfiability problem is constrained by ANDing together the KDC, SCKVC and LC clauses. The SAT attack formulated using SMT solver continues until the SMT solver returns UNSAT. A final call to the SMT solver returns the correct key. Note that this SMT attack is a one-to-one translation of the original SAT attack in

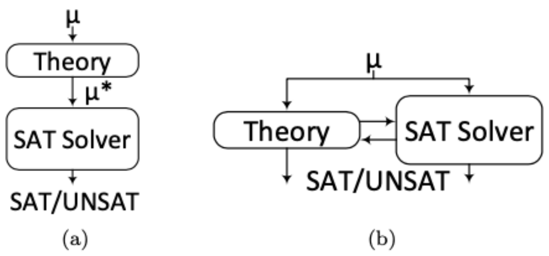


Figure 7: Execution Flow of (a) SMT Eager vs. (b) SMT Lazy approach

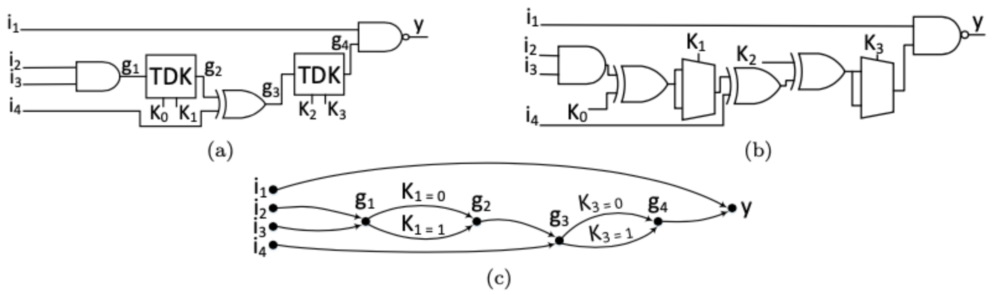


Figure 8: Converting a (a) netlist obfuscated with TDK cells to a (b) Key Programmable Circuit (KPC), and (c) the representative graph of the netlist annotated with TDK cell delays.

extend its capabilities to attack obfuscation schemes that could not be logically modeled.

6.2 Attack Mode 2: Eager SMT Attack

Theory solvers could be used to extend the capabilities and performance of SMT solver compared to that of a SAT solver. This, as illustrated in Fig. 7 could be done by (1) using the theory solver to extract all required clauses that complete the CNF description with respect to the obfuscation scheme and then to perform a SAT attack, referred to as the SMT *Eager* approach. This could be thought (2) by invoking the theory and SAT solver in parallel to simultaneously model and solve the problem, referred to as *Lazy* approach.

In this section, we illustrate how the Eager approach of SMT attack could be used to attack the obfuscation schemes that could not be broken or understood by a pure SAT attack. For this purpose, we formulated an SMT attack on the delay-locking (DLL) scheme proposed in [XS17]. Notice that the proposed approach could be used in formulating attacks on other obfuscation techniques that rely on non-logical properties of circuit obfuscation such as timing, power, delay, etc. by using the appropriate theory solvers.

Fig. 8 illustrates the translation steps for converting a DLL[XS17] obfuscated circuit (using translation table in Fig. 5) to its key programmable circuit and captures its graph representation. As illustrated in Fig. 8(b), K_1 effectively has no impact on the logical behavior of the circuit and only changes its delay properties. Hence, subjecting this obfuscated circuit to a SAT attack results in a random assignment to K_1. Therefore, by having k TDK cells, which have $2k$ keys in total, a SAT solver returns one logically correct key sequence among 2^k different set of such logically correct keys that control the TDK cells, however, only one of such keys doesn't result in setup and hold violations. Hence, a correct attack should consider the delay and timing properties of the netlist in addition to its logical correctness.

The shortcoming of SAT attack to capture the delay and timing properties of the netlist, when attacking DLL obfuscation, is remedied in an SMT attack by means of using a graph

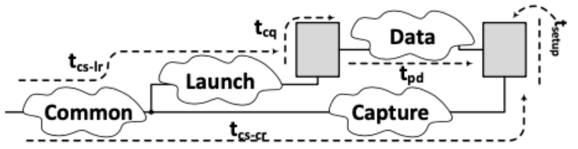


Figure 9: The Naming Convention for Different Sections, and Various Delay Components of a Timing Path

between all primary inputs and outputs of the obfuscated netlist. This VPCC is a CNF presentation of all valid assignment of the keys, such that no setup or hold violation is created. Note that among many such possibilities, only one possibility has both the correct timing and the correct logical behavior.

To build the VPCC clauses, we should compute the setup and hold constraints on every timing path. The setup and hold timing checks for a timing path is expressed using the following inequalities:

$$t_{cs-lr} + t_{clk-q} + t_p + t_{setup} + U \leq t_{cs-cr} + T_{clk} \tag{1}$$

$$t_{cs-lr} + t_{clk-q} + t_{cd} \geq t_{hold} + t_{cs-cr} + U \tag{2}$$

In this equation which uses the notation in Fig. 9, the t_{cs-lr} is the clock source to launch register delay, t_{cs-cr} is the clock source to capture register delay, U is the clock jitter/uncertainty, t_{clk-q} is the clock to q delay of the launch register, t_{setup} is the capture-register setup time, t_{hold} is the hold time requirement for the capture register, t_p is the propagation delay through the longest path in the timing path, and finally the t_{cd} is the propagation delay through the shortest pah in the logic. Considering that the DLL logic is only inserted on *Data* sections a of timing path (according to the tormentingly in Fig. 9), it can only affect the t_p and t_{cd}. Note that it is also possible to enhance the DLL obfuscation beyond that described in [XS17] and use the TDK cells for building clock skew in the clock network, however, a similar attack still could be formulated. For now, let's consider that DLL, as described in [XS17], only affects the Data section of timing path. The equations 1 and 2 could be re-written as:

$$\mathbf{t}_p \leq T_{clk} + (t_{cs-cr} - t_{cs-lr}) - t_{clk-q} - t_{setup} - U = Upper \tag{3}$$

$$\mathbf{t}_{cd} \geq t_{hold} + (t_{cs-cr} - t_{cs-lr}) - t_{clk-q} + U = Lower \tag{4}$$

Before performing any reverse engineering, we know the T_{CLK} from the functional chip purchased on market. Note that a functional chip (the oracle) is needed to perform the SAT or SMT attack as explained in section 3. Now let's consider a netlist obtained after reverse engineering. The end-point and start-point registers for each timing path are known. Hence, by means of spice simulation, the register could be characterized and the t_{clk-q}, t_{setup} and t_{hold} are extracted. Note that there are limited type of registers used in a physical design, and at this step only a handful of registers need to be characterized. Extracting a measure for uncertainty could be also achieved by means of spice simulation.

At this point, considering that a TDK cell can change the delay of a timing path, the delay of each timing path (D_j) could be divided into a constant delay (C_j) and a variable delay ($V_j(K)$), where the variable delay is a function of the number of TDK cells in that timing path, and the key assumed for each TDK. Hence, Delay of Timing path j from start point s to endpoint p ($D_j^{s\to p}$) that passes through N TDK cells each with delay $D_{TDK}^{s\to p}(i)$, depending on the value of key K_i is obtained from:

$$D_j^{s\to p} = C_j^{s\to p} + V_j^{s\to p}(k) \tag{5}$$

$$D_j^{s\to p} = C_j^{s\to p} + \sum_{i=1}^{N} K_i \times D_{TDK}^{s\to p}(i) \tag{6}$$

For a given timing path, and by using the equation 6, we could rewrite the delay constraints in equations 3 and 4 as:

$$\forall j| \quad D^{s\rightarrow p}_{j_{min}} = C^{s\rightarrow p}_{j_{min}} + \sum_{i=1}^{N} K_i \times D^{s\rightarrow p}_{TDK}(i) \geq Lower \tag{8}$$

These inequalities capture the lower and upper bound delay constrain for every pair of input-output pins in a design, and collectively capture the model μ of the graph theory solver. Based on this formulation, the number of added inequalities is M $\times$ N, in which M is the number of primary inputs, and N is the number of primary outputs. However, one inequality bounds all timing paths between the selected input-output pin pair, removing the need to express the inequality for every timing path in the design as needed in MILP-based attack that was suggested in [XS17].

After writing these inequalities for each input-output pair, a call to the SMT *solve* function returns all key combinations for which all paths constraints/inequalities are satisfied. In the other word, by assuming any of the returned key combinations, the circuit will not violate its setup and hold timing checks. However, note that only one (or few) of these key values is logically correct. The correct key value then could be extracted by invoking a SAT solver, and by providing the set of key combinations (in CNF format) as a constraint to the logical circuit satisfiability problem. This process is illustrated in Alg. 2. As it can be seen in Alg. 2, function $GenTLC$ is responsible for generating all inequalities. Line 7-8 of $GenTLC$ function generates inequality (7) and (8) for each input (Sp) to each output (Ep).

Algorithm 2 Eager SMT Attack on DLL [XS17]

```
 1: function SMT_EAGER_ATT(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
 2:     $KPC \leftarrow$ ReplaceKPG($N_{obf}$);
 3:     $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
 4:     $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
 5:     $SCKVC = TRUE$;
 6:     $SATC = KDC \wedge SCKVC$;
 7:     $LC = TRUE$;                                          ▷ Learned Clauses
 8:     $G(X,K) \leftarrow$ Graph_Translation($N_{obf}$);
 9:     $T_{LC} \leftarrow GenTLC(G(X,K))$;                           ▷ Theory Learned Clauses
10:     $SMT_{LC} \leftarrow SATC \wedge T_{LC}$;                              ▷ SMT Clauses
11:     while ((($X_{DI}$,$K_1$,$K_2$,CC) $\leftarrow SMT.Solve(SMT_{LC})$)= $TRUE$) do
12:         $Y_f \leftarrow C_{org}(X_{DI})$;
13:         $DIVC = C(X_{DI},K_1,Y_f) \wedge C(X_{DI},K_2,Y_f)$;
14:         $SCKVC = SCKVC \wedge DIVC$;
15:         $LC = LC \wedge CC$
16:         $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
17:     $Key \leftarrow SMT.Solve(SMT_{LC})$;
```

Pre-Processing step by using a graph theory solver for SMT attack (*Eager*)

```
 1: function GENTLC(Graph $G$)
 2:     $Inputs \leftarrow G$.find_start_points();
 3:     $Outputs \leftarrow G$.find_end_points();
 4:     $T_{LC} \leftarrow []$
 5:     for each ($Sp$ in $Inputs$) do
 6:         for each ($Ep$ in $Outputs$) do
 7:             Upper(Sp,Ep)(K) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));
 8:             Lower(Sp,Ep)(K) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);
 9:             $T_{LC} \leftarrow$ SMT.solve(Upper(Sp,Ep)(K) $\wedge$ Upper(Sp,Ep)(K) $\wedge$ $T_{LC}$);
10:     return $T_{LC}$
```

This algorithm is similar to Alg. 1, with the additional step of using a theory solver for pre-processing the netlist in line 8, extraction of all key combination resulting in correct timing behavior in line 9, and providing these constraints to the SAT solver in the next step in line 10. Note that the *solve* function in the Eager approach is called in two places; first for generating the timing valid key combination clauses (inside GenTLC function), problem is the SRCLock [RMKS18]. The authors have shown that the obfuscation is SAT hard, since without pre-processing the cycles, the SAT solver will be trapped or produce an incorrect key. Additionally, they have suggested two mechanisms by which the number of cycles in a netlist could exponentially grow with respect to the number of inserted feedbacks. For attacking cyclic logic, as suggested by CycSAT attack [ZJK17] we need to pre-process the netlist and extract the No Cycle Conditions to prevent the SAT solver from being trapped. However, in SRCLock[RMKS18] the number of cycles grow exponentially, and therefore the runtime of pre-processing step also grows exponentially, preventing us to ever reach the SAT attack. For such problems, the Eager approach that relies on reduction of the problem to a SAT problem does not work. However, the Lazy approach of the SMT attack provides a solution.

6.3 Attack Mode 3: Lazy SMT Attack

Using the Lazy approach of SMT attack relaxes the requirement of Eager approach to complete the pre-processing step before invoking the SAT attack. In the Lazy approach the SAT solver and theory solver(s) simultaneously check different models of a unified satisfiability problem, exchange clauses, and check each other's literal assignment. This could significantly prune the decision tree of a SAT solver search space for finding a satisfying assignment and remove the need for a complete and unbounded execution of theory solver as it only has to check the validity of constraints for SAT assigned literals.

Algorithm 3 Overall SMT Attack (*Lazy* Approach)

```
 1: function SMT_LAZY_ATT(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
 2:     $KPC \leftarrow$ ReplaceKPG($N_{obf}$);
 3:     $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
 4:     $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
 5:     $SCKVC = TRUE$;
 6:     $SATC = KDC \wedge SCKVC$;
 7:     $LC = TRUE$;                                         ▷ Learned Clauses
 8:     $G(X,K) \leftarrow$ Graph_Translation($N_{obf}$);
 9:     $T_{CE}(K) \leftarrow GenTCE(G(X, K))$;       ▷ Theory Constraint Expressions (Not Solved)
10:     $T_{CE}(K1, K2) \leftarrow T_{CE}(K1) \cup T_{CE}(K2)$ ;
11:     while $(((X_{DI},K_1,K_2,CC) \leftarrow SMT.Solve(SATC, T_{CE}(K1, K2))) = TRUE)$ do
12:         $Y_f \leftarrow C_{org}(X_{DI})$;
13:         $DIVC = C(X_{DI},K_1,Y_f) \wedge C(X_{DI},K_2,Y_f)$;
14:         $SCKVC = SCKVC \wedge DIVC$;
15:         $LC = LC \wedge CC$
16:         $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
17:     $Key \leftarrow SMT.Solve(SMT_{LC}, T_{CE}(K))$;
```

Initialization of constraints for SMT attack (*Lazy* Approach)

```
 1:     function GENTCE(Graph $G$)
 2:         $Inputs \leftarrow G$.find_start_points();
 3:         $Outputs \leftarrow G$.find_end_points();
 4:         $T_{CE}(K) \leftarrow []$
 5:         for each ($Sp$ in $Inputs$) do
 6:             for each ($Ep$ in $Outputs$) do
 7:                 Upper(Sp,Ep)(K) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));
 8:                 Lower(Sp,Ep)(K) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);
 9:                 $Range(Sp,Ep)(K) \leftarrow$ Lower(Sp,Ep)(K) $\wedge$ Upper(Sp,Ep)(K);
10:                 $T_{CE}(K) \leftarrow T_{CE}(K) \cup Range(Sp,Ep)(K)$;
11:         return $T_{CE}(K)$
```

In order to illustrate the Lazy approach of SMT attack, in this section, we formulate an SMT attack to again break the DLL [XS17] obfuscation. The Lazy approach of SMT attack on DLL [XS17] is illustrated in Alg. 3. The big difference in the Lazy and Eager approach is that after model generation for theory solver, the SMT *solve* function is not is then called to find an assignment for a discriminating input X_{DI}, and two different keys $K1$ and $K2$ such that generated outputs are different at least in one bit, however both keys generate a valid timing scenario. Since the SAT model (SATC) and Theory models ($T_{CE}(K1, K2)$) share literals and are subjected to a unified set of constraints, the decision tree and search space for the SMT solvers is significantly reduced.

6.4 Attack Mode 4: Accelerated Lazy SMT Attack (AccSMT)

In this section, we argue that re-formulating the Lazy SMT which benefits from capabilities of *BitVector* theory solver allows us to build a more efficient attack.

6.4.1 Motivation:

Our modification to the SAT attack is inspired by the observation that higher output corruption, reduces the SAT hardness of an obfuscation scheme. A discriminating input X_{DI}, is an input capable of sensitizing the logic paths of the netlist under study, such that (1) some of the differences in the values of internal nodes in the result of application of two different keys K_1 and K_2 are propagated to at least one output. (2) none of the previously found DIPs (that were used in building a DIVC) were able to propagate the generated inconsistency to a primary output. This mechanism is continued until the number of sensitized paths, reaches a point where any inconsistency that from application of two different keys is propagated to the primary outputs using the constructed set of DIVC circuits. At this point, the set of previously found X_{DI}s form a complete set of discriminating inputs, such that if a key generates the correct output for all inputs in this set, it will generate the correct output for all other inputs.

Different DIPs have different pruning power. A DIPs strength could be assessed based on the number of inconsistencies that it could sensitize to the primary outputs conditioned that previous DIPs were incapable of doing so. Hence, depending on the pruning power of DIPs, the size of the complete set of DIPs could be different. A minimal complete set of DIPs is the smallest set of DIPs that could de-obfuscate the circuit. In our Lazy approach for SMT attack, we propose a mechanism to reduce the size of the complete set of DIPs pushing it towards the minimal set. Since in each SAT or SMT iteration one DIP is found, having a smaller number of DIPs result in smaller number of iterations.

In SAT attack, it requires only a single bit difference in the output for generation of a DIP. In SMT attack, we could make a stronger requirement for the generation of DIPs. This could be achieved by forcing the SMT solver to find DIPs with the largest possible Hamming distance of primary outputs of the KPC circuits when for the same input, two different keys are applied. Such a DIP has a much higher pruning capability, and is able to sensitize a larger number of key-related inconsistencies to the output. The discovery of such powerful DIPs reduces the number of required DIPs that is needed to form a complete set of DIPs that could de-obfuscate the circuit. This is because when the hamming-distance is larger either the KPC circuits differ in (1) key-bit(s) that are located close to the inputs, or (2) large number of assumed key-bits (in the middle of timing paths or close to primary outputs) are different in two KPC circuits, or (3) the combination of two scenarios. In both cases, the added DIP and the resulting learned clauses eliminate the cause of obtaining such large hamming distance, resulting in the elimination of a large number of inputs as possible future DIPs while eliminating a larger

6.4.2 Using BitVector Theory Solver:

Assessing DIPs based on hamming distance of the primary output is easily implementable in SMT solver by using a *BitVector* theory solver. The bitVector theory solver allows us to perform integer-oriented arithmetic operations such as addition, subtraction, and multiplication. The Hamming Distance (HD) of output Y_1 and Y_2 is obtained using:

$$HD(C(X_{DI}, K_1), C(X_{DI}, K2)) = HD(Y_1, Y_2) = \sum_{i=1}^{N} Y_1(i) \oplus Y_2(i) \tag{9}$$

The HD is then used to write the constraining expressions that are posed on the BitVector theory solver using the formulation:

$$Th_{Lower} \leq HD(Y_1, Y_2) \leq Th_{Upper} = Size(Output) \tag{10}$$

The upper threshold Th_{Upper} is kept constant equal to the size of output pins, but the lower threshold Th_{Lower} is defined as a variable, allowing us to sweep the hamming distance constraint posed on BitVector theory solver from a maximum value of the number of output bits to a minimum value of 1. The lower bound could be reduced every time the SMT solver returns UNSAT, indicating there is no other DIP that satisfies the HD requirement poset on theory solver. The process terminates when the SMT cannot even find a DIP that causes HD of 1. Adaption of this constraint forces the SMT solver to find DIPs with higher pruning power, reducing the size of a complete set of DIPs.

6.4.3 Using TimeOut:

For an SMT or a SAT attack, the execution time is determined based on the formula, $\sum_{i=1}^{N} t(i)$, where $t(i)$ is the execution time of the i^{th} iteration of an SMT attack. Hence, by just reducing the number of SAT iterations N, we cannot guarantee a shorter execution time, because finding a DIP with tighter constraint may pose a more difficult problem to the SMT solver and increase $t(i)$. For this purpose, we can limit the time allowance for finding a DIP in each iteration. The timeout limit TO prevents the SMT solver from spending a long time for finding a DIP with large HD, when finding such DIP has become excessively difficult. By adapting the timeout feature, during an SMT attack, the HD requirement is reduced when either (1) the SMT solver returns UNSAT, indicating there exist no such input, or when (2) we encounter time-out interrupt. In this case, the HD constraints posed on BitVector theory solver is reduced by one and the SMT solver is called. Note that the time interrupt is supported by MonoSAT [BBHH15] used in this paper, and many other freely available SMT solvers. Also, note that use of time interrupt pushes the final solution away from a minimal complete set of DIPs. However, our experiments illustrate that this usually results in considerably smaller execution time.

6.4.4 Enabling Approximate Attacks:

Our objective is to enable the SMT attack to be carried against a netlist similar to that of Fig. 10, which is obfuscated by both SAT Hard (SH) and high Corruption (HC) obfuscation schemes, to find all keys for the HC obfuscation, and to detect the trap of SH obfuscation and exit while generating an approximate key.

The SAT hard obfuscation mechanisms suggested in recent literature, such as *SARLock, Anti-SAT, and SFLL* [YMRS16, XS16, YSN+17], have a very small output corruption, and the SAT hardness is maximized when there is only a single input for a given key that results in an incorrect output. The pruning power of DIPs found in each iteration of the SAT solver for SH obfuscation solutions is very small, and each DIP eliminates a single

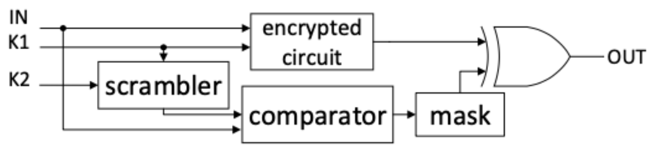


Figure 10: A hybrid obfuscation scheme consists of a SAT Hard (SH) obfuscation (SARLock) and a High Corruption (HC) Logic Locking scheme

of approximate attacks is to find the correct key for the HC obfuscation without being trapped by SH obfuscation.

The accelerated SMT attack could significantly improve the performance of approximate attacks. Since HC obfuscation schemes result in high output corruption, finding DIPs that lead to larger HD at the output biases the SMT attack to find the HC related obfuscation keys in the earlier iterations. The remaining problem is the design of a termination strategy for the accelerated and approximate SMT attack to detect the trap of SH obfuscation, exit and report the approximate key. For this purpose, we use a constraint on the number of allowed repetitions R when HD is very small (e.g. 1). If the remaining and un-found keys are only the SH keys, the SMT keeps finding weak DIPS (HD of 1) and iterations are completed very quickly. By setting the repetition limit R to an appropriately large value, we can detect the trap and terminate the attack.

The unique feature of accelerated approximate attack is that if we remove the timeout (TO) requirement, then the approximate attack guarantees that the HD of the approximately unlocked circuit and that of the functional circuit is at most HD_{Low} bits different, with HD_{Low} being the hamming distance requirement in which the R repetition is taken place. This could be proven as follows: Suppose that there exists an undiscovered discriminating input and two keys that cause larger than HD_{Low} bit difference (HD_{Low}+D) in the primary outputs. Hence, the SMT solver when constrained by bitVector theory solver expression for finding HD = HD_{Low}+D should return SAT. This contradicts the SMT previous execution control state where the SMT attack for that HD has returned UNSAT, otherwise the HD constraint was not reduced.

Algorithm 4 Accelerated SMT Attack

```
function AccSMT_Attack(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
    $HD_{High}$ = Number of output bits;                 ▷ Upper hamming distance limit;
    $HD_{Low}$ = $HD_{High}$ - 1;                        ▷ Lower hamming distance limit;
    TO = 50s;                                            ▷ Timeout constraint;
    R = 20;                                              ▷ Repetition limit;
    $R_{HD}$ = 1;                                        ▷ Repetition condition;
    $R_{count}$ = 0;                                     ▷ Repetition count variable;
    $KPC \leftarrow$ Replace_KPG($N_{obf}$);
    $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
    $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
    $SCKVC = TRUE$;
    $SATC = KDC \wedge SCKVC$;
    $LC = TRUE$;                                         ▷ Learned Clauses
    $BV(X,K) \leftarrow$ Circuit_Output_to_BitVector($N_{obf}$);
    $BVS(X, K_1, K_2)$ = SUM_of_1s($BV(X,K_1) \oplus BV(X,K_2)$)
    $T_{CE} \leftarrow BVS(X,K_1,K_2) \geq HD_{Low}$;    ▷ Theory constraint expression;
    $T_{CE} \leftarrow T_{CE} \cup (BVS(X, K_1, K_2) \leq HD_{High})$;
    while $HD_{Low} \geq 1$ do
        while $(((X_{DI},K_1,K_2,CC) \leftarrow SMT.Solve(SMT_{LC},\ T_{CE},\ TO)) = T)$ do
            $Y_f \leftarrow C_{org}(X_{DI})$;
            $DIVC = C(X_{DI},K_1,Y_f) \wedge C(X_{DI},K_2,Y_f)$;
            $SCKVC = SCKVC \wedge DIVC$;
            $LC = LC \wedge CC$
            $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
            if ($HD_{Low} \leq HD_R$) then
                if ($R_{count}$ == R) then
                    Break;
                $R_{count}$ ++;
        HD
```

Table 2: ISCAS-85 Benchmarks and Their Characteristics.

Circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c7552
# of Inputs	36	41	60	41	33	233	50	178	207
# of Outputs	7	32	26	32	25	140	22	123	108
# of Gates	120	162	320	506	603	872	1179	1726	2636

6.4.5 Accelerated SMT attack formulation:

Alg. 4 demonstrates the reformulated Lazy approach of SMT attack on obfuscated circuits. In this algorithm, the HD_{High}, and HD_{Low} are the high and low threshold requirement for hamming distance on primary outputs, TO is the timeout limit per iteration, R is the repetition allowance before exiting and generating an approximate key, and R_{HD} is the hamming distance after which the repetition condition is checked. The BitVector theory solver input model is defined in lines 14 and 15, and converted to theory constraint expressions in lines 16 and 17. The T_{CE} poses an upper and lower bound on the hamming weight difference of the outputs of two instances of the same circuits with the same input, but two different keys. The SMT attack sweeps the hamming distance in the first while loop, while the second while loop formulate the modulo satisfiability theory attack. The SMT solver receives the SMT_{LC} model, the BitVector theory solver constraint T_{CE} and the timeout allowance TO and check whether there is a valid assignment for SMT_{LC} conditioned that T_{CE} is valid withing TO time allowance. If it exists, the while loop is satisfied. Additionally, it returns the discriminating input X_{DI}, the two keys found (K_1, K_2) and a list of learned conflict clauses CC. Then the X_{DI}, similar to the original SAT attack is used to construct additional DIVC and update the satisfiability model SMT_{LC}. At the end of each iteration, the algorithm checks whether the hamming distance is reduced to the limit, where the repetition condition for SH problems is checked. In this case, if the repetition count reaches the specified threshold value R, the SMT attack is terminated. additionally if for HD of 1, the SMT solver can no longer find a satisfying assignment, the SMT attack is terminated. A final call to SMT solver with the constructed satisfiability module theory model generates the key.

7 Experimental Results

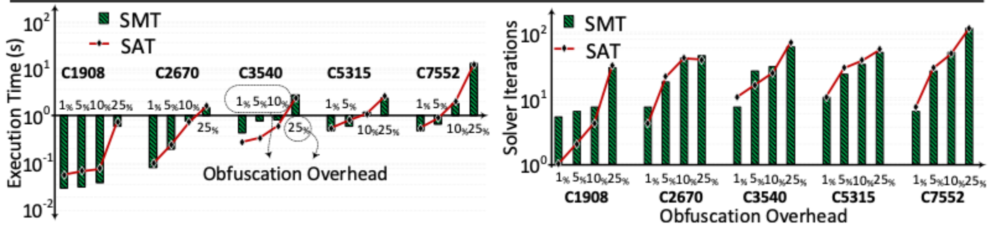
For evaluating different modes of SMT Attack, we used a farm of desktops with 4-core Intel Core-i5 CPU, running at 1.8GHz, with 8 GB RAM. The operating system on desktops was Ubuntu Server 16.04.3 LTS. For a fair comparison, and to reduce the impact of the operating system background processes, we dedicated one desktop to each SMT solver at a time. For benchmarking, we used most of ISCAS-85 benchmarks, characteristics of which is listed in Table 2. Since MiniSAT has been used in the SMT Solver as its built-in SAT solver, we use the default values of resource limits in MiniSAT as resource limits of the SMT attack (68 years for the CPU time limit and $\approx$ 2147 TB for the memory usage limit). As the baseline for comparing SMT attack performance against a pure SAT attack, we employed the Lingeling-based SAT attack by [SRM15]. In addition, for each attack we ran the solvers *Five* times on SMT and SAT solvers [RTG+18] and reported the average runtime.

7.1 Evaluation of SMT reduced to SAT Attack

As explained in section 6.1, and explained by Alg. 1 the SMT solver could be used

Table 3: Comparing the Execution Time and the Number of Iterations of SMT Attack in SAT Attack mode (Attack Mode 1) with a Pure SAT Attack based on Lingling Solver [SRM15] on ISCAS-85 Benchmarks Obfuscated using Random XOR/XNOR Insertion (RLL) [RKM10] (50%-50%).

Circuit	c2670				c3540				c5315				c7552			
	SAT		SMT		SAT		SMT		SAT		SMT		SAT		SMT	
	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
1%	3	0.102	5	0.474	10	0.513	8	1.31	9	0.405	10	0.441	11	0.577	19	0.806
5%	45	1.514	57	3.589	19	1.502	25	1.249	32	1.354	24	2.433	67	5.271	42	4.261
10%	312	14.08	342	15.752	36	1.782	36	2.973	59	3.798	57	4.881	97	15.82	94	15.67
25%	781	114.5	692	108.6	77	9.796	65	8.462	95	19.63	107	22.48	215	225.6	228	270.8



(a) (b)

Figure 11: Comparing the Performance of SMT-attack when Reduced to a SAT Attack (Attack Mode 1) with that of original SAT-attack that uses Lingling solver [SRM15]. The ISCAS-85 Benchmarks are obfuscated using IOLTS'14 Insertion [DBDN+14]: (a) Comparison of the Execution Times, (b) Comparison of the Number of Iterations

SMT solver is a superset of SAT attacks, and with the same formulation provides similar performance. For this comparison, we employed two obfuscation methods: (1) random XOR/XNOR insertion (RLL) [RKM10], and (2) obfuscation using nets with unbalanced probabilities (*IOLTS'14*) [DBDN+14]. ISCAS-85 benchmarks are obfuscated using these schemes with obfuscation overhead ranging from 1% to 25%.

Table 3 compares the execution time of SMT attack and the SAT attack proposed in [EMGT15, SRM15] when RLL obfuscation is deployed. As captured in this table, the execution time of the SMT attack when reduced to SAT Attack is approximately equivalent, in terms of number of iteration and execution time, with that of an original SAT attack across all benchmarks and all ranges of obfuscation overhead. Fig. 11 illustrates the same comparison when the *IOLTS'14* obfuscation method is deployed. As illustrated, the SMT reduced to SAT, in terms of performance, behaves similar to the SAT attack.

7.2 Evaluation of Eager SMT Attack

We used the Delay Logic Locking scheme [XS17] in our case study to show the extended capabilities of the SMT attack in solving obfuscation problems that cannot be modeled in a SAT attack. The Eager approach of SMT attack is evaluated in this section, and the Lazy approach is evaluated in the following section. Additionally, to increase the obfuscation difficulty and demonstrate the strength of the SMT attack, in addition to obfuscation using DLL, we obfuscated the circuit with additional MUX and XOR gates using gate insertion policy in *IOLTS'14* [DBDN+14], such that 50% of the keys are used for *DLL*, and 50% for *IOLTS'14* obfuscation. Finally, we used some of the keys for both logic and delay obfuscation to create dependencies such that the solvers could not divide and conquer the attack.

Table 4: Execution Time of the SMT Attack in the Eager Mode (Attack Mode 2) on ISCAS-85 Benchmarks with Different Overhead when Obfuscated using DLL[XS17]+ RLL[RKM10] (50%-50%) Obfuscation Schemes.

Circuit	c1908	c2670	c3540	c5315	c7552
1%	0.077 + 1.663	0.068 + 170.0	0.053 + 4.054	1.291 + 114.6	0.580 + 138.6
2%	0.016 + 1.919	0.221 + 175.6	0.200 + 5.001	1.535 + 144.6	1.808 + 185.5
3%	0.054 + 2.161	0.337 + 212.7	1.359 + 6.328	3.057 + 160.4	2.247 + 245.9
5%	0.075 + 2.810	0.495 + 248.4	1.553 + 8.325	3.891 + 256.9	7.812 + 353.3
10%	0.499 + 3.812	38.78 + 407.1	1.524 + 14.35	16.19 + 550.3	33.92 + 782.7
25%	8.951 + 21.71	112.4 + 972.5	9.459 + 92.42	60.30 + 1567	2920 + 5244

SMT execution time = $x + y$, x : The execution time of the SAT engine of the SMT Solver, y : The execution time of the theory engine of the SMT Solver

all valid key combinations. The key combinations are converted into CNF statement, which is passed to the SAT solver. In the second phase, the SAT solver attacks the circuit satisfiability problems augmented with these additional CNF clauses on valid key combinations, and make a new round of calls to the SMT solvers. As illustrated in Fig. 7(a), the invocation of theory and SAT solver, and the overall SMT attack is serialized. Accordingly, in order to reflect our experimental results for evaluating of Eager approach, we separate the execution time of theory solver and that of the SAT solver.

Table 4 captures the results of Eager SMT attack for different ISCAS-85 benchmarks with different obfuscation overhead. The theory execution time indicates the time required by graph theory to find the all possible and valid key combinations (where only one of them is valid). Similarly, SAT execution time demonstrates the time taken by SAT solver to find a valid key, given the additional theory solver generated constraining clauses. As illustrated in this table, the SMT attack, in all cases is concluded and reported the correct key. The result of the pure SAT attack is not reported, as it always produces the wrong key for being oblivious to the DLL key values. Hence, the SMT solver in this respect extends the attack capability by means of including various theory solvers.

Note that the execution time of the SAT solver (the x value in each column of reported data in Table 4) depends on the (1) size of the circuit, and (2) the percentage of obfuscated cells. Hence the circuit c7552, for being larger than c1908 has a longer SAT attack time across all percentage obfuscation points. In addition, the increase in the SAT attack time is only slightly super-linear (close to polynomial) with respect to increase in the degree of obfuscation. On the other hand, the execution time of the theory solver (the y value in each column of reported data in Table 4) depends on (1) the number of input, (2) the number of outputs, and (3) the degree of obfuscation. Hence, a circuit with larger number of IOs has a longer execution time for its theory solver, but the execution time is bounded by O(NM), with M and N being the number of inputs and outputs respectively. This indicate that the run-time of theory solver (unlike the MILP-based attack that was suggested in [XS17]) does not exponentially increase with respect to number of timing paths in a netlist, as it only depends on the number of IOs and not the total number of timing paths. In addition, as illustrated, by increasing the degree of obfuscation, similar to SAT attack, the execution time of theory solver grows slowly with a close to polynomial paste.

7.3 Evaluation of Lazy SMT Attack

The Lazy approach of SMT attack, as illustrated in Fig. 7(b), uses the SMT solve function to simultaneously solve the theory and circuit SAT problem. In this approach, the theory model is defined but is not solved. In many applications, the Lazy approach outperforms

Table 5: Execution Time of SMT Attack in the Lazy Mode (Attack Mode 3) on ISCAS-85 Benchmarks with Different Obfuscation Overhead, Obfuscated using DLL[XS17]+RLL[RKM10] (50%-50%).

Circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c7552
1%	0.033	0.177	0.263	0.567	0.466	20.44	0.983	11.53	13.07
2%	0.049	0.262	0.325	0.676	0.596	21.86	3.443	11.76	17.83
3%	0.065	0.329	0.350	0.877	0.723	23.39	2.436	15.27	19.04
5%	0.049	0.340	0.517	1.085	1.456	28.87	2.587	38.87	45.96
10%	0.204	0.503	1.195	5.622	3.334	83.06	6.712	94.80	319.6
25%	0.599	1.481	2.036	297.2	95.67	2706	126.3	552.8	8045

Table 6: Comparing the Execution Time and the Number of Iterations of the Accelerated Lazy SMT Attack (AccSMT) (Attack Mode 4) with that of Original SAT Attack (based on Lingling solver) in [SRM15] on ISCAS-85 Obfuscated Circuits using RLL[RKM10] Obfuscation Policy.

Circuit	c2670				c3540				c5315				c7552			
	SAT		AccSMT		SAT		AccSMT		SAT		AccSMT		SAT		AccSMT	
	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
1%	3	0.102	2	0.316	10	0.513	3	0.185	9	0.405	2	0.163	11	0.577	3	0.374
5%	45	1.514	11	3.589	19	1.502	6	0.761	32	1.354	6	0.408	67	5.271	17	2.607
10%	312	14.08	26	5.817	36	1.782	11	1.236	59	3.798	12	1.753	97	15.82	19	4.721
25%	781	114.5	107	24.05	77	9.796	16	1.606	95	19.63	27	7.916	215	225.6	24	23.52

Eager approach is not even applicable. However, the parallel invocation of the theory and SAT solver, and the resulting literal exchange, and the additional constraints posed on the solver could result in significant reduction in the time needed to explore the problem's decision tree, and removes the need to complete the pre-processing before starting the SAT attack. Hence, if the execution time of theory solver poses a runtime beyond acceptable, the problem could only be attacked by the Lazy SMT approach.

Table 5 shows the Lazy SMT attack execution time on ISCAS-85 benchmarks that were obfuscated using the process that was explained in the previous section (mixing 50% DLL+ 50% IOLTS). Considering the SAT and theory solver are invoked simultaneously, we have a single execution for the entire SMT problem, and unlike Eager approach we cannot separate the execution time of theory solver and the SAT solver. As illustrated, in comparison with the Eager approach, in most cases the Lazy approach finds the key obfuscation key in shorter time.

In the Lazy approach, the number of iterations decreases drastically compared to the Eager approach. However, the execution time of each iteration increases. This is because each DIP needs to satisfy both the theory constraints and the circuit SAT formulation. However, when a DIP is found, it is a stronger DIP with higher pruning power.

By comparing the results of Eager and Lazy approach of SMT attack in Table 4 and Table 5 we observed that in majority of cases, the Lazy approach outperforms the Eager approach. However, in some cases (e.g. for Benchmark C1908 with 50% overhead), the Lazy approach may become slower than Eager approach, indicating that Lazy approach doesn't always result in the stronger attack. However, note that there exist a set of

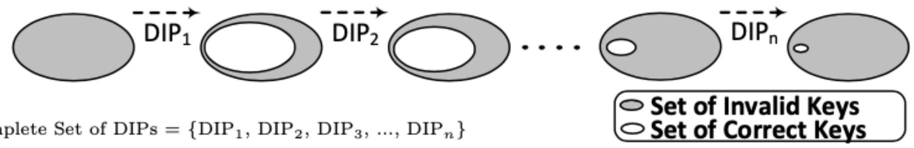

DIP$_1$
DIP$_2$
DIP$_n$
Set of Invalid Keys
Set of Correct Keys
Complete Set of DIPs = {DIP$_1$, DIP$_2$, DIP$_3$, ..., DIP$_n$}

Figure 12: Set of potentially valid keys (the Remaining Keys to be Evaluated) reduces in each iteration of SMT or SAT attack when a new Discriminating Input (DI) is discovered and is added to the Discriminating Input Validation Circuit (DIVC).

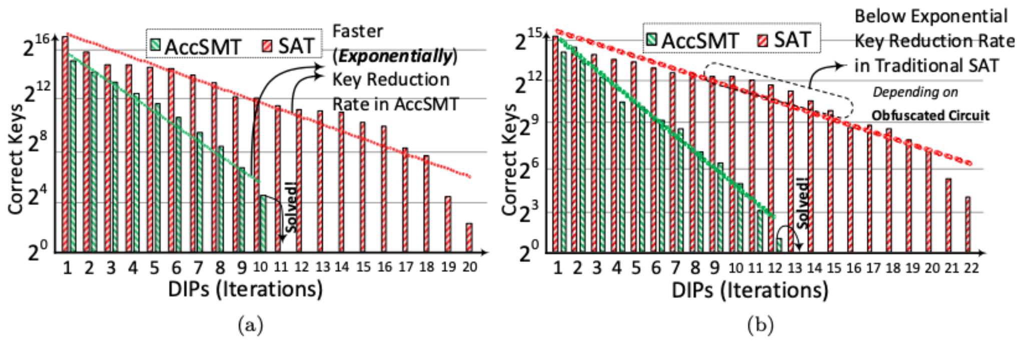


Figure 13: Rate of Reduction in the Number of Potentially Valid Keys (Remaining Keys) in each Iteration of the Original SAT Attack and the AccSMT Attack, when Attacking ISCAS-85 Benchmarks Obfuscated using RLL[RKM10] Obfuscation Scheme: (a) C2670 (b) C7552.

7.4 Evaluation of Lazy AccSMT Attack

7.4.1 Ability to find stronger DIPs:

Before invoking the SMT or SAT attack, any key could be considered as a *potentially valid key*. The strength of a DIP comes from its ability in reducing the size of this set in each iteration. After finding each DIP, as illustrated in Fig. 12, the size of *potentially valid key* set reduces. When reaching a complete set of DIPs, any key left in this set is a correct key. As discussed in section 6.4, a stronger DIP could sensitize a larger number of inconsistencies (due to application of a discriminating input and two different keys) to the primary outputs. Hence, its natural for such a DIP to have a higher pruning power in reducing the number of *potentially valid keys*. To evaluate this claim, we profiled the number of *potentially valid key* after each iteration of SMT and SAT attack, when working on the same obfuscation problem. Fig. 13 illustrates the key reduction rate in three ISCAS-85 benchmarks obfuscated by RLL[RKM10]. In all scenarios the DIPs found by AccSMT solver are stronger, as the number of remaining keys is reduced at a significantly higher rate. As illustrated, the number of iterations is also significantly reduced because the complete set of DIPs, when the pruning power of DIPs is higher, is of smaller size.

7.4.2 Stronger and shorter attack:

The stronger DIPs found by the AccSMT attack, result in significant reduction of the number of DIs needed for a complete discriminating input set. Each DI is found in one iteration, Hence, smaller number of DIs indicates a smaller number of iterations. Table 6 compares the execution time and the number of iterations between the SAT solver and the AccSMT solver. The ISCAS-85 benchmarks for this simulation are obfuscated using

Table 7: Execution Time and the Number of Iterations of Accelerated Lazy SMT Attack (AccSMT) (Attack Mode 4) on SARLock + *IOLTS'14* for finding K_1 (Traditional Keys).

Circuit	c1908		c2670		c3540		c5315		c7552	
	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
1%	7	0.512	16	3.075	8	1.304	3	0.384	7	2.905
5%	18	0.701	25	11.91	15	1.681	11	1.707	33	17.56
10%	31	4.085	51	26.47	21	3.779	35	7.402	61	44.07
25%	71	8.605	105	76.8	66	22.91	56	16.64	88	58.32

7.4.3 Ability to carry approximate attack:

As described in section 6.4.4, the AccSMT attack is able to distinguish between SAT-hard (SH) and high-corruption (HC) obfuscation. It quickly finds the correct keys for HC obfuscation, detects the SH trap, exits, and reports the approximate key.

To evaluate the approximate mode of the AccSMT attack, we have obfuscated the ISCAS-85 benchmarks using *SARLock* + *IOLTS14* as suggested in [YMRS16]. The overall structure of the obfuscated circuit is illustrated in Fig. 10. In this hybrid obfuscation scheme, the SARLock is the SH obfuscation, and the RLL is the HC obfuscation protocol. The invocation of the original SAT attack in [SRM15][EMGT15] results in a timeout, due to SARLock trap. However, the AccSMT can very quickly find all the keys for HC obfuscation, detect the SH trap, and report the approximate key. Table 7 depicts the number of iterations and execution time of AccSMT attack for finding the approximate keys for each instance of the obfuscated circuit under attack. Note that repetition count (R=20 in our case study) is excluded from this table.

8 Conclusion

In this paper, we introduce a class of Satisfiability Modulo Theory (SMT) attacks on obfuscated circuits. The SMT attack benefits from the expressive nature of theory solvers, that allow the attacker to express constraints that are difficult or even impossible to express using CNF, including timing, delay, power, arithmetic, graph and many other first-order theories. We first illustrated that a SAT attack could be easily implemented using SMT solver to prove that SMT attack is a superset of the SAT attack. Then we proposed two variants of SMT attack on obfuscated circuits using Eager and Lazy approach of SMT solver. We illustrated that using the Eager and Lazy approach, we could break the Delay Logic Locking [XS17] obfuscation that cannot be broken by a SAT attack, proving that SMT attack's capabilities go beyond a SAT attack. It shows that by only using non-logical properties of a netlist for obfuscation, we not provably increase the security of an obfuscated netlist, indicating the need for further study and exploration in this domain to generate obfuscation schemes with provable security. Then we proposed the Accelerated SMT attack (AccSMT), and we illustrated that by using theory solvers (BitVector theory solver in this paper), we could significantly speed-up the attack against specific obfuscated circuits, and reported significant reduction in the execution time of the AccSMT compared to SAT attack. Finally, we illustrated that with a small modification, the AccSMT could be used as an approximate attack, allowing us to find an approximate key for obfuscation schemes that combine a SAT hard obfuscation with high corruption obfuscation.

References

[illegible]

[AKHS18] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan. Smt decryption tool binaries, benchmark circuits, and codes. https://github.com/gate-lab/SMTAttack, 2018.

[BBHH15] Sam Bayless, Noah Bayless, Holger H Hoos, and Alan J Hu. Sat modulo monotonic theories. In *AAAI*, pages 3702–3709, 2015.

[BT18] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. 2018.

[BTZ10] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing ic piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 27(1), 2010.

[CCBW13] Ronald P Cocchi, Lap Wai Chow, James P Baukus, and Bryan J Wang. Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing, 2013. US Patent 8,510,700.

[DBDN+14] Sophie Dupuis, Papa-Sidi Ba, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *IEEE Int'l Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 49–54. IEEE, 2014.

[EHKV89] Andrzej Ehrenfeucht, David Haussler, Michael Kearns, and Leslie Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–261, 1989.

[EMGT15] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes. In *NDSS*, 2015.

[GFT13] Ujjwal Guin, Domenic Forte, and Mohammad Tehranipoor. Anti-counterfeit techniques: from design to resign. In *Microprocessor Test and Verification (MTV), Int'l Workshop on*, pages 89–94, 2013.

[IEGT13] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh V Tripunitara. Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation. In *USENIX Security Symposium*, pages 495–510, 2013.

[KAG+18] H. Mardani Kamali, K. Zamiri Azar, K. Gaj, H. Homayoun, and A. Sasan. Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 405–410, 2018.

[KH16] Hadi Mardani Kamali and Shaahin Hessabi. A fault tolerant parallelism approach for implementing high-throughput pipelined advanced encryption standard. *Journal of Circuits, Systems and Computers (JCSC)*, 25(09):1650113, 2016.

[KLMS+98] Andrew B Kahng, John Lach, William H Mangione-Smith, Stefanus Mantik, Igor L Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang, and Gregory

[MKP08] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Testing techniques for hardware security. In *IEEE Test Conference*, pages 1–10, 2008.

[NO80] Greg Nelson and Derek C Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, 27(2):356–364, 1980.

[RKK14] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[RKM10] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Ending piracy of integrated circuits. *Computer*, 43(10):30–38, 2010.

[RMKS18] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. Srclock: Sat-resistant cyclic logic locking for protecting the hardware. In *Proceedings of the Great Lakes Symposium on VLS (GLSVLSI)*, pages 153–158, 2018.

[RPSK12] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Proceedings of the Design Automation Conference (DAC)*, pages 83–89, 2012.

[RTG$^+$18] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan. Benchmarking the capabilities and limitations of sat solvers in defeating obfuscation schemes. In *IEEE Int'l Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 275–280, 2018.

[RZZ$^+$15] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Transactions on computers*, 64(2):410–424, 2015.

[Sho82] Robert E Shostak. Deciding combinations of theories. In *Int'l Conference on Automated Deduction*, pages 209–222, 1982.

[SLM$^+$17a] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. Appsat: Approximately deobfuscating integrated circuits. In *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, pages 95–100, 2017.

[SLM$^+$17b] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. Cyclic obfuscation for creating sat-unresolvable circuits. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 173–178, 2017.

[SPP$^+$18] Hossein Sayadi, Nisarg Patel, Sai Manoj PD, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, pages 137–143, 2015.

[TR03] Cesare Tinelli and Christophe Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Computer Science*, 290(1):291–353, 2003.

[TSŠ+06] Pim Tuyls, Geert-Jan Schrijen, Boris Škorić, Jan Van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 369–383, 2006.

[WSM+16] Theodore Winograd, Hassan Salmani, Hamid Mahmoodi, Kris Gaj, and Houman Homayoun. Hybrid stt-cmos designs for reverse-engineering prevention. In *Proceedings of the Design Automation Conference (DAC)*, page 88, 2016.

[XS16] Yang Xie and Ankur Srivastava. Mitigating sat attack on logic locking. In *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pages 127–146, 2016.

[XS17] Yang Xie and Ankur Srivastava. Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction. In *Proceedings of the Design Automation Conference (DAC)*, page 9, 2017.

[XSTF17] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks. In *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pages 189–210, 2017.

[Yeh12] Age Yeh. Trends in the global ic design service market. *DIGITIMES research*, 2012.

[YMRS16] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Sarlock: Sat attack resistant logic locking. In *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, pages 236–241, 2016.

[YMSR17a] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal attacks on logic locking and camouflaging techniques. *IEEE Transactions on Emerging Topics in Computing*, (1):1–1, 2017.

[YMSR17b] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Security analysis of anti-sat. In *Design Automation Conference (ASP-DAC), Asia and South Pacific*, pages 342–347, 2017.

[YSN+17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1601–1618, 2017.

[ZJK17] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *Proceedings of the Int'l Conference on Computer-Aided Design (ICCAD)*, pages 49–56, 2017.