KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation

Kaveh Shamsi*, Meng Li†, David Z. Pan†, and Yier Jin*
*Department of Electrical and Computer Engineering, University of Florida
†Department of Electrical and Computer Engineering, University of Texas at Austin
kshamsi@ufl.edu, meng_li@utexas.edu, dpan@ece.utexas.edu, yier.jin@ece.ufl.edu

***Abstract*—Logic locking and IC camouflaging are two promising techniques for thwarting an array of supply chain threats. Logic locking can hide the design from the foundry as well as end-users and IC camouflaging can thwart IC reverse engineering by end-users. Oracle-guided SAT-based deobfuscation attacks against these schemes have made it more and more difficult to securely implement them with low overhead. Almost all of the literature on SAT attacks is focused on combinational circuits. A recent first implementation of oracle-guided attacks on sequential circuits showed a drastic increase in deobfuscation time versus combinational circuits. In this paper we show that integrating the sequential SAT-attack with incremental bounded-model-checking, and dynamic simplification of key-conditions (Key-Condition Crunching or KC2), we are able to reduce the runtime of sequential SAT-attacks by two orders of magnitude across benchmark circuits, significantly reducing the gap between sequential and combinational deobfuscation. These techniques are applicable to combinational deobfuscation as well and thus represent a generic improvement to deobfuscation procedures and help better understand the complexity of deobfuscation for designing secure locking/camouflaging schemes.**

I. Introduction

Several security and privacy challenges exist with respect to the Integrated Circuit (IC) supply chain. For fabless design houses, the entire layout of the design is revealed to the foundry. This allows the foundry to pirate the design, or overproduce the IC. Perhaps the most concerning threat is that the foundry can insert malicious logic into the design known as hardware Trojans which can be very extremely difficult to detect [1]. This can violate the assumption of hardware as the root of trust in systems. Another challenge is that end-users can reverse engineer the physical IC to recover the design for the goal of IP theft or system exploitation [2].

Logic locking, IC camouflaging, and split-manufacturing are three prominent techniques for partially hiding the design of an IC from an untrusted foundry or end-user [1]. Split-manufacturing is based on fabricating the upper metal layers in a trusted facility hiding the design from the foundry. While this method appears to be the most secure, it requires a trusted foundry [3]. IC camouflaging is based on special nano-device structures that are difficult to disambiguate using conventional IC reverse engineering techniques, but it does not hide the design from the foundry. Logic locking is based on adding programmability to the design which is configured post-fabrication, hiding the design partially from the foundry. If the programmable elements are also resilient to physical reverse engineering, logic locking can thwart end-user attackers as well [4].

Although different in silicon implementation, logic locking and IC camouflaging can be modelled using similar semantics, and various threat models can be discussed for their security. The oracle-guided threat model allows an attacker to query chosen inputs on an unlocked/functional IC and use correct input-output pairs to disambiguate the netlist [5]. The strongest oracle-guided attack is the SAT attack and its variants that use SAT solver calls to derive new queries iteratively and deobfuscate the netlist [5]–[7]. The SAT attack has mainly been limited to combinational circuits and has shown great success in deobfuscating low-overhead locking and camouflaging schemes on relatively large logic cones (thousands of gates) in a matter of seconds.

If the oracle-guided attacker does not have access to all internal flip-flops, which is relatively easy to ensure, the deobfuscation problem becomes sequential. The SAT attack can be extended to such a sequential circuit by unrolling the circuit [8] and using model-checking queries. However, the only two implementations of sequential attacks demonstrate prohibitively larger runtime compared to the combinational attack [8], [9]. This leads some to believe that sequential deobfuscation is simply impossible on large-scale circuits.

In this paper we present several techniques that can significantly reduce the runtime of sequential attacks. At the heart of these techniques is the integration of the sequential deobfuscation flow with an iterative bounded model checker and the use of several facts specific to the deobfuscation problem to dynamically simplify the SAT problems throughout the attack. Specifically the paper makes the following contributions:

- A sequential deobfuscation algorithm is presented based on an integrated iterative unrolling and incremental SAT solving flow for which a proof-of-concept implementation shows a speedup of 150x with respect to the state-of-art attack that uses an advanced recent model checker as a black-box. We have released the attack toolset along with the benchmark circuits to the community [10].
- We propose various dynamic simplification techniques that can crunch the iterative conditions that can pile up unnecessarily in the SAT solver. These techniques can be applied to the combinational SAT attack as well. We demonstrate how these simplification techniques can help in deobfuscating hard instances that require long sequences of input-output pairs to deobfuscate.

II. Preliminaries

A. Combinational Deobfuscation

Logic locking and IC camouflaging on combinational circuits can be both modeled as transforming an original Boolean circuit $c_o : I \to O$, where $I = \{0,1\}^n$ and $O = \{0,1\}^m$ are the input and output space respectively, to an "augmented/obfuscated/extended" Boolean function $c_e : I \times K \to O$ where $K = \{0,1\}^l$, and there exists $k_* \in K_*$ such that $\forall i \in I,\ c_e(i, k_*) = c_o(i)$. Generally, an attacker reverse engineering a combinational circuit which includes ambiguous elements can encode this ambiguity in the form of "key-variables" and define the function space $\mathcal{C} = \{c_e(i,k) | k \in K\}$ for which disambiguation/deobfuscation is defined as finding the correct function $c_o \in \mathcal{C}$ or at least approximating it with high accuracy [7]. The goal of the defender is to ensure that this is at least as difficult as some known brute-force level with the minimum amount of overhead. A good metric for analyzing the quality of locking/camouflaging schemes is their security per layout area and power overhead [4].

Various threat models can be defined with respect to the above definition which can give rise to different notions of security. The weakest threat model is the oracle-less attack model in which the attacker only has access to the circuit representation of c_e. Attacks in this model are not quite successful even on low overhead schemes [11]. A stronger threat model is the oracle-guided model in which it is assumed that the attacker has input-output (oracle) access to c_o, i.e. the attacker can pass arbitrary x to an oracle and receive $c_o(x)$. Under the oracle-guided threat model the strongest attack is the SAT-based attack. The attack begins by building a mitter SAT problem $M \equiv (c_e(i, k_1) \neq c_e(i, k_2))$ solving which will return an input pattern $\hat{i}$ for which k_1 and k_2 affect the output value. This pattern which is known as a discriminating input pattern (DIP) is queried on the oracle $\hat{o} \leftarrow c_o(\hat{i})$ and the I/O-constraint $c_e(\hat{i}, k_1) = c_e(\hat{i}, k_2) = \hat{o}$ is stored back in the solver and the mitter problem is solved again. When the mitter+constraints problem is no longer satisfiable, the constraints alone will identify a $k_* \in K_*$.

B. Sequential Deobfuscation

The minimal circuit unit for an oracle-guided deobfuscation attack is a single-ended logic cone with controllable inputs and observable outputs. If within any such cones there exists flip-flops that are part of a strongly-connected-component (i.e. reside on a feedback path), then the cone is stateful and the combinational SAT attack cannot be formulated. To capture this we can extend our formal model: sequential obfuscation is transforming $c_{os}(i, s_o)$ for which s_o is an ls_o-bit state register, to $c_{es}(i, s_e, k)$ for which s_e is an $ls_e \geq ls_o$-bit state register such that for a correct key we have $k_* \in K_*\ c_{os}(i, s_o) = c_{es}(i, s_e, k_*)$, for all sequential traces $\langle \hat{i}^0, \hat{i}^1, ..., \hat{i}^u \rangle \in I^\infty$ starting from the reset state. Note that the key in this case, k, is a static/frozen input in that it does not change along the trace as opposed to i[1].

A SAT-based oracle-guided attack can be formulated in this model using unrolling. The attacker can unroll the obfuscated sequential circuit by up to u times getting c_e^u which takes in u input patterns producing u outputs. This unrolling is a combinational circuit and a mitter can be formed. Queries using SAT will reveal a sequence of DIPs or a discriminating input sequence (DIS) $\hat{\mathrm{I}} = \langle \hat{i}^0, \hat{i}^1, ..., \hat{i}^u \rangle$ which will cause a disagreement between $c_e^u(\hat{\mathrm{I}}, \hat{k}_1)$ and $c_e^u(\hat{\mathrm{I}}, \hat{k}_2)$. This DIS can be queried on the oracle, cycle by cycle, revealing an unrolled I/O-constraint $c_e^u(\hat{\mathrm{I}}, \hat{k}_1) = c_e^u(\hat{\mathrm{I}}, \hat{k}_2) = \hat{\mathrm{O}}$. Every time the unrolled mitter becomes unsatisfiable at some depth u, the set of keys that satisfy the I/O-constraints (the version-space [7]) will produce a hypothesis circuit that does not deviate from the oracle for up to u rounds. The attacker extends the unrolling until a termination condition is detected.

Note that the unrolling step where a DIS is searched for, is essentially a model checking query using a safety property. The DIS query is to check whether the transition system defined by $c_es(i, s_1, k_1) \times c_es(i, s_2, k_2)$ satisfies the safety property $G(\neg M)$. The transition system is updated with the addition of each DIS constraint and the query is repeated. A query with a bounded depth corresponds to bounded model checking (BMC) and an unbounded one to unbounded model checking (UMC). Algorithm 1 shows this overall flow.

Algorithm 1 Given oracle access to c_{os} and the netlist of c_{es} return a correct key $k_* \in K_*$.

```
1: function SeqDecrypt($c_{es}$, $c_{os}$ as black-box)
2:     $j \leftarrow 0$, $b \leftarrow 1$
3:     $M \leftarrow c_{es}(i, s_1, k_1) \neq c_{es}(i, s_2, k_2)$
4:     $F_j \leftarrow true$
5:     while !Termination($F_j$) do
6:         if Bmc($F_j \wedge M$, G($\neg M$), $b$) $\rightarrow Fail$ then
7:             $\hat{\mathrm{O}}_j \leftarrow c_{os}(\hat{\mathrm{I}}_j)$    $\hat{\mathrm{I}}_j \leftarrow$ Cex(G($\neg M$))
8:             $F_{j+1} \leftarrow F_j \wedge (c_e^b(\hat{\mathrm{I}}_j, k_1) = \hat{\mathrm{O}}_j) \wedge (c_e^b(\hat{\mathrm{I}}_j, k_2) = \hat{\mathrm{O}}_j)$
9:             $j \leftarrow j + 1$
10:        else
11:            $b \leftarrow b * 2$
12:        end if
13:    end while
14:    satisfy $F_j$ with $k_1$ and $k_2$
15: return $k_1$ as key
16: end function
```

El Massed et al. [8] specified three different conditions for detecting termination during the process (the Termination procedure in Algorithm 1). 1) *Unique Key (UK)*: when there is only one key that satisfies the I/O-constraints F_j, 2) *Combinational Equivalence (CE)*: where the transition function c_{es} for k_1 and k_2 is combinationally equivalent (i.e. equivalent next-state and output, given any input and start state). 3) *Unbounded Model Checking (UMC)*: if a call to an unbounded model checker concludes that $\neg M$ is invariant in the reachable state-space then we can terminate the attack.

[1]A dynamic key that is generated internally using the state can be modeled as an internal variable and not the key. An unknown initial state can also be modeled using additional key-variables.

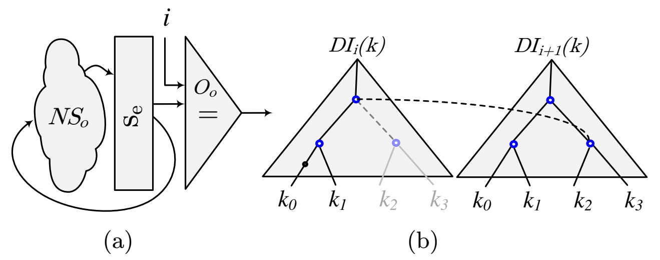

Fig. 1. (a) A common sequential circuit for which deobfuscation can need exponential queries that are exponentially long; (b) DIP conditions in an exponential query scheme have potentially many equivalent nodes.

III. Fast Sequential Deobfuscation

A. Unnecessary Clause Inflation

El Massed et al.'s implementation of the procedure in Algorithm 1 demonstrated a significant runtime increase as compared to combinational deobfuscation. Their attack was tested on camouflaged ISCAS circuits using gate-level camouflaging schemes with 32 inserted camouflaged gates. ISCAS benchmarks with 32 camouflaged gates under combinational attacks rarely exceeds a minute of attack time, whereas for the sequential attack the runtime would reach well into hours. Furthermore, some benchmarks such as the s444 or the s400 resulted in very long DISes making it difficult to deobfuscate them completely in the given deadline.

Since the SAT attack is an iterative attack, its complexity can be modeled as the number of queries times the average time of the SAT queries. This has naturally resulted in an array of attempts to thwart the SAT attack by increasing the minimum number of queries [12] which is possible by ensuring that each DIP/DIS disqualifies only a limited number of incorrect keys forcing the attacker to query large portions of the input space. This is typically done by XORing a function $h(i,k)$ with the output, which uses comparators to corrupt the output on a small number of input patterns. While these schemes increase query count exponentially, they skew the error rate of the obfuscation ($\Pr_{i\in I,\ k\in K}[c_e(i,k) \neq c_o(i)]$) exponentially as well, allowing efficient approximation of the function [7]. As pointed out in [12], there is a fundamental contention between the two metrics of query count and error rate (non-approximability).

Another way to defeat the SAT attack is by making the SAT queries more complex. While predicting the complexity of a SAT problem is a fundamental open question, it is observed that certain problems when reduced to SAT cannot be solved efficiently with modern solvers. Examples are factoring (i.e. solving large multipliers), and cryptanalysis of block or stream ciphers. While integrating such instances with circuit obfuscation can provide strong security they come with prohibitive overhead (thousands of gates for a single cipher round).

It is important to pay attention to the distinction between the two different forms of complexities in SAT attacks. While the deobfuscation complexity in the second case stems from a genuine hardness of the underlying problem, this is not the case in the first category. In the case of exponential query schemes an exponential number of circuit instances end up being added to the solver with current SAT attacks. Typically the I/O-constraints when attacking such schemes resemble comparator circuits such as the one shown in Figure 1b. With each DIP, a copy of this circuit is added to the solver. This increases the runtime of the SAT calls in a super-linear trend with respect to the number of queries. This is while we know that fundamentally the task of deobfuscating this scheme is to search a range of input patterns which should be linear to the number of patterns in this range. With every query a single incorrect key is disqualified.

In fact, not only should the runtime not increase at a super-linear rate, the condition on the key can lose its complexity. If the first query reveals that $k \neq 10010$ and the second query reveals that $k \neq 10011$, the conjunction of the two conditions is that $k \neq$ 1001x which can be represented with a smaller circuit/Binary-Decision-Diagram (BDD)/clause/cube.

While in the case of combinational circuits exponential query circuits are contrived, in the case of sequential circuits low-activity comparator logic is quite common. Consider the up-counter circuit in Figure 1a that is waiting for a particular count value at i to fire the output. If a key input is randomly inserted into the output logic there is a high chance that it will require an exponentially large number of DIS queries that grow in length in each iteration. One core idea in this paper is to investigate generic methods for avoiding this unnecessary piling up of clauses in the SAT solver during the attack through dynamic simplification.

B. Incremental SAT-Solving

The majority of single-threaded modern SAT solvers rely heavily upon the Conflict-Driven Clause Learning (CDCL) scheme. The solver selects variables, assigns a value to them, and propagates constants through the clauses extracting *implications* along the way. If the formula is satisfied, the solver terminates, otherwise a conflict occurs. Conflict-analysis extracts a *conflict-clause* from this event which is added to the solver and helps avoid taking this branch in the future. Modern solvers also allow support for *assumptions* in which the solver can be called under the assumption that certain variables are true or false without having to construct a new solver instance for each call.

If subsequent to a call to a SAT solver, additional variables and clauses are added they can be added to the solver's data structures without having to completely erase previously learned conflict-clauses and implications. A great number of modern verification tasks rely on such an incremental approach to SAT solving, where a single solver is called thousands of times each time with added clauses/variables/assumptions [13]. In our attack we closely integrate the deobfuscation procedure with the BMC engine so that the same SAT solver instance can be used for various tasks, including BMC calls, termination checking, simplification, and even unbounded checking. We allow the algorithm to decide when it is time to reconstruct the SAT solver.

C. *Key-Condition Sweeping*

We can simplify various conditions during the attack using sweeping techniques which detect equivalent nodes in a circuit and merge them. *BDD-Sweeping* is done by converting circuit nodes to their canonical BDD in a topological order discovering equivalent nodes along the way and merging them. If the BDD size for a node exceeds a size limit s, a new variable is added instead and the procedure continues. *Cut-Sweeping* presented in [14] is based on enumerating all *cuts* of a node in an And-Inverter-Graph (AIG) and then merging equivalent cuts. A cut c of a node n is any set that includes n and a contiguous range of its transitive-fanin. The functionality of a cut can be represented using a BDD or a Boolean vector representing its truth-table [15]. *SAT-Sweeping* is the only sweeping technique that can allows detecting equivalent nodes under some external constraints on the inputs. Incremental SAT solver calls are made to the solver to test whether two nodes are equivalent. Previous conditions can be included in the solver before the sweeping procedure begins.

A critical step in deobfuscation attacks is adding knowledge from a DIP/DIS to the current model. In both the sequential and the combinational attack, I/O-constraints are in the form of a Boolean condition $c_e(\hat{\mathrm{I}}_\mathrm{j}, k_1) = \hat{\mathrm{O}}_\mathrm{j} \wedge c_e(\hat{\mathrm{I}}_\mathrm{j}, k_2) = \hat{\mathrm{O}}_\mathrm{j}$ for which $\hat{\mathrm{I}}_\mathrm{j}$ and $\hat{\mathrm{O}}_\mathrm{j}$ are constants that if propagated will result in a condition only on the key-variables in the mitter, $DI_j(k_1||k_2)$. In each iteration a new conditions is discovered and conjoined to the previous conditions : $F_j = \bigwedge_{i=0}^{j} DI_i(k_1||k_2)$. With each iteration of the attack the space of possible keys is reduced, so $F_i \supseteq F_i + 1$, $(F_i + 1 \rightarrow F_i)$ and the correct key condition is in the fixed-point of this monotone trace. On each conjunction we perform $F_i + 1 \leftarrow F_i \wedge DI_i(k_1||k_2)$.

Given the above iterative procedure we derive the following two sweeping-based simplification strategies:

I. AND the two circuits $F_i \wedge DI_i(k_1||k_2)$ and simplify the result with no additional constraints. BDD/Cut/SAT-sweeping can all be used for this. It is not necessary but intuitive to reconstruct the SAT solver instance after every such simplification step. This is because the existing condition F_i will be mixed with the incoming condition $DI_i(k_1||k_2)$ so the existing clauses relating to F_i may no longer be valid.

II. An important result from [16] is that if two circuits with common variables are being conjoined we have:

$$A(s) \wedge B(s) = A(s) \wedge B(s)|_{A(s)}$$

where: $B(s)|_{A(s)} \begin{cases} B(s), & \text{if } A(s) = 1 \\ \textit{don't care} & \text{otherwise} \end{cases}$. Using this with SAT-sweeping we can simplify the new I/O-constraint $DI_i(k_1||k_2)$ in $F_i \wedge DI_i(k_1||k_2)$ with the assumption that $F_i = 1$. This is better suited than strategy-I to incremental solving since there is no reductive operation on the SAT solver and only the new conditions in $DI_i(k_1||k_2)$ are simplified rather than existing clauses/conditions F_i.

The above two strategies can be combined so that strategy-I is used in iterative steps, whereas every some number of iterations strategy-II can be performed along with a solver reconstruction.

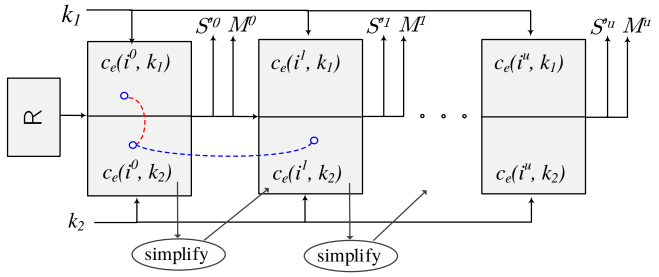

Fig. 2. Iterative unrolling/simplification/termination/settlement-checking. By removing the initialization condition R we can study inductive properties, check for u-round termination, and simplify the transition relation for the next round. Merges between corresponding nodes in the mitter (red and blue dotted-lines) represent settled nodes for which their fanin functions is fully deobfuscated.

D. *Key-Condition to BDD Conversion*

As was discussed in Section III-A, we need to avoid unnecessary build-up of clauses in the solver. One approach that we used in our attack is to convert the key-condition F_i to the BDD representing it. While F_i can be a large conjunction of circuit copies, the canonical information that it provides regarding the key can be very small. For a relatively small number of key-bits we observed that when adding incoming I/O-constraints to a BDD that represents the space of correct keys, the accumulated BDD's size remains small and can in fact drop throughout the attack on benchmarks with deep DISes. The solver can be reconstructed with the accumulated BDD condition by translating BDD nodes to MUXes and subsequently to a CNF formula or an AIG.

E. *Negative Key-Condition Compression*

Another possible technique for key-condition simplification is keeping track of disqualified keys. Each time a discriminating query is made, one or both of $\hat{k}_1$ and $\hat{k}_2$ will be disqualified by comparing the output of $c_e^u(\hat{\mathrm{I}}_j, \hat{k}_1)$ and $c_e^u(\hat{\mathrm{I}}_j, \hat{k}_2)$ with the output from the oracle. We can represent these key negations using BDDs or Zero-suppressed BDDs (ZDDs) and compress them using well known cube-covering (Irreduntant Sum-of-Products) algorithms. Note that each key-disqualifying constraint can be represented as a clause. We can attempt to remove literals from this clause to generalize the negative key-condition to a larger space of possible keys. This can be done by simulating patterns with bit-flips, or iterative SAT calls across the literals in a key-disqualifying clause (This resembles IC3's inductive cube generalization [17]). The solver can be reconstructed to this condition when the algorithm detects a build up of clauses in the solver.

F. *Transition Relations*

The sequential deobfuscation attack detects DISes by unrolling a mitter circuit $M(i, s_{e1}||s_{e2}, k_1||k_2) \equiv (c_{es}(i, s_{e1}, k_1) \neq c_{es}(i, s_{e2}, k_2))$. Each frame of this unrolling is a copy of M applied to the state at cycle u, s_e^u, producing the output and the next-state s_e^{u+1} at cycle $u + 1$ (see Figure

2). We can add the initialization condition R to this unrolling of M (e.g. $R \equiv s_{e1}^0||s_{e2}^0 \leftrightarrow 0$) to make it represent a trace that starts from the reset state. Without this condition the unrolling will represent a u-long trace starting from any state.

As for simplification of M^u, we can simplify $R \wedge M^u$ after every unrolling but we will need to reconstruct the solver for checking the condition. However, if we remove the initialization condition R, and then simplify M^u, since the simplification is invariant with respect to the starting state, we can use the simplified transition in subsequent unrolling steps. This was first presented in [16]. It was also noted in [16] that during the simplification of the uninitialized M^u if we identify a merge between two nodes corresponding nodes v_i and v_{i-t} from different frames, this constitutes an inductive invariant. There are additional benefits to transition relation simplification with respect to the deobfsucation problem.

First, M^u can directly be used for building I/O-constraints. Since the I/O-constraint unrolling is a copy of the transition relation, simplifying it directly translates to simpler key-conditions. Second, during simplification, if a node in $c_{es}(i, s_{e1}, k_1)$ is merged with its corresponding node in $c_{es}(i, s_{e2}, k_2)$, this constitutes a *settlement* of that node over the state space meaning that the functionality of the fanin cone of this node is resolved and a correct sub-key for this cone can be extracted from the solver. Note that this technique is stronger than the backbone analysis presented in [18] which checks the settlement of one key bit at a time. This can be applied to combinational attacks as well.

G. Termination Conditions

As for the three different termination conditions presented by El Massed et al., we first note that the CE condition is stronger than the UK condition so we can simply discard UK. As for CE given our integration of the deobfuscation process with the BMC solver, we extend this termination condition checking as follows: 1) we include a next-state comparator in M^u denoting the output of this comparator as S'^u; 2) at deobfuscation depth u we remove the initialization assumptions R; 3) add assumptions that will activate clauses which will connect s_{e1} and s_{e2} causing the two circuit copies in the mitter to start from the same unconstrained initial state; and 4) checking $S'^u \vee M^u$ at frame u which will be unsatisfiable upon termination. This termination condition procedure can span multiple rounds, hence is more powerful despite the fact that it does not create an additional copy of the unrolling or a separate solver.

Unbounded termination queries rarely become necessary in practice. However, we can integrate McMillan's unbounded interpolation-based model checking (IMC) [19] with this procedure. Whenever an unrolling at depth u becomes unsatisfiable, we extract a *Craig Interpolant* from the SAT solver between M^0 and $M^{1,\dots,u}$. This inerpolant IM^u represents an over-approximation of reachable states that are at distance $u-1$ from a mitter disagreement. If $\text{IM}^u \rightarrow \text{IM}^{u-1}$ then all reachable states have been checked for disagreements and we can terminate with a correct key. Algorithm 2 shows a high-level flow of the final sequential deobfuscation routine.

Algorithm 2 Given oracle access to c_{os} and the sequential model expression for c_{es} return a correct key $k_* \in K_*$.

```
function FastSeqDecrypt($c_{es}$, $c_{os}$ as black-box)
    $j \leftarrow 0$, $R \leftarrow (s_{e1} \leftrightarrow s_{e2} \leftrightarrow 0)$, $b \leftarrow 1$
    $M \leftarrow c_{es}(i, s_{e1}, k_1) \neq c_{es}(i, s_{e2}, k_2)$
    $F_j \leftarrow true$
    while !MultiStepTermination($F_j$, $j$) do
        $Merges \leftarrow$ RelativeSimplifyTr($F_j$, $M$, $j$)
        AnalyzeInductiveMerges($Merges$, $F_j$, $M$)
        if Bmc($F_j \wedge M$, $G(\neg M)$, $b$) then
            $O_j \leftarrow c_{os}(\hat{I}_j)$    $\hat{I}_j \leftarrow$ Cex($G(\neg M)$)
            $DI_j(k_1||k_2) \leftarrow$ Propagate($c_{es}, \hat{I}_j, \hat{O}_j$)
            $F_{j+1} \leftarrow$ Sweep($F_j \wedge DI_j(k_1||k_2)$)
            $j \leftarrow j + 1$
        else
            $b \leftarrow b * 2$
        end if
    end while
    satisfy $F_j$ with $k_1$ and $k_2$
return $k_1$ as key
end function
```

IV. Experiments

We created a proof-of-concept implementation of the above ideas in an in-house developed C++ framework. We used the CUDD package for BDD/ZDD operations. We used Glucouse for SAT solving without its CNF-simplification routines. While El Massed et al. did not specify the details of their implementation they do mention that they used the NuSMV model checker as a back-end. As the baseline implementation we used nuXmv (a more recent version of NuSMV with IC3 and other advanced model checking algorithms) as a black-box solver and communicated with it using its file interface. All tests were run on an AMD EPYC server with 96 cores and 256GB of memory running at 1.9GHz, where it was made sure that parallel tests do not exceed 90 to allow for equal distribution of resources. We used the sequential ISCAS benchmarks listed in Table I. We used random XOR/XNOR locking [18] to obfuscate the designs with various overhead values. The overhead being the increase in gate-count of the design when mapped to primitive gates using ABC.

Table II show the results comparing nuXmv to our iterative BMC-based deobfuscation with a 3 hour deadline. Overall a speed-up of 150X is observed across the benchmarks. As for the key-condition crunching approaches, we took a number of hard instances that result in deep and exponentially many queries such as the s400 benchmark and carried out various key-crunching techniques. We observed that BDD and SAT-sweeping of the key-condition can consistently reduce up to 80% of the nodes in incoming I/O-constraints. Converting the key-condition to BDDs was surprisingly effective in deobfuscating hard benchmarks with less than 40 key bits. While the circuit conditions required several thousands of gates at an unrolling depth of 10, the BDD size of the key-condition remained below a few hundred nodes regardless of the unrolling depth. Reconstructing the SAT solver to this condition allowed deobfuscating s444 with 40 key bits in

TABLE I
ISCAS SEQUENTIAL BENCHMARK SET

bench	ins	outs	dffs	gates	bench	ins	outs	dffs	gates
s27	4	1	3	13	s967	16	23	29	423
s298	3	6	14	133	s953	16	23	29	424
s386	7	7	6	165	s938	34	1	32	478
s499	1	22	22	174	s1238	14	14	18	526
s344	9	11	15	175	s991	65	17	19	538
s349	9	11	15	176	s1196	14	14	18	547
s382	3	6	21	179	s1269	18	10	37	606
s400	4	6	21	185	s1494	8	19	6	653
s444	3	6	21	202	s1488	8	19	6	659
s526	3	6	21	214	s1423	17	5	74	731
s526n	3	6	21	215	s3271	26	14	116	1688
s510	19	7	6	217	s3384	43	26	183	1868
s832	18	19	5	292	s4863	49	16	104	2446
s820	18	19	5	294	s5378	35	49	179	2958
s635	2	1	32	318	s6669	83	55	239	3319
s641	35	24	19	398	s9234	19	22	228	5825
s713	35	23	19	412	s15850	14	87	597	10369
s967	16	23	29	423	s35932	35	320	1728	17793

TABLE II
NAIVE DEOBFUSCATION USING NUXMV VERSUS OUR INTEGRATED INCREMENTAL BMC-BASED DEOBFUSCATION RUNTIME IN SECONDS

overhead	1%		5%		10%	
circuit	nuXmv	int	nuXmv	int	nuXmv	int
s27	0.17	0.07	0.21	0.17	0.25	0.10
s298	0.35	0.08	7.95	0.18	58.01	0.21
s386	0.45	0.08	3.53	0.14	22.13	0.32
s499	0.30	0.09	12.62	0.28	26.96	0.38
s344	0.55	0.07	7.41	0.19	20.17	0.19
s349	0.25	0.07	3.20	0.13	12.87	0.18
s382	13.71	0.23	2498.70	179.51	-	1135.26
s400	17.44	0.30	4077.86	117.68	-	-
s444	2.32	0.12	2216.46	194.12	-	-
s526	27.37	0.50	1122.35	125.99	-	-
s526n	456.52	88.22	2833.76	97.97	-	-
s510	27.60	4.80	422.47	11.55	255.69	9.63
s832	1.83	0.12	63.06	0.59	1024.45	1.68
s820	0.86	0.10	136.18	0.87	318.64	0.75
s635	-	368.54	-	339.89	-	-
s641	1.67	0.09	64.24	0.18	-	7.00
s713	4.94	0.10	129.97	0.26	-	16.13
s967	7.86	0.16	140.20	0.58	1447.80	2.21
s953	14.03	0.33	200.27	0.69	422.75	1.11
s938	-	0.19	-	-	-	-
s1238	6.30	0.17	101.43	0.39	2209.64	1.06
s991	6.50	0.12	271.04	0.41	-	9.09
s1196	2.37	0.13	122.78	0.58	635.76	0.67
s1269	11.37	0.30	86.35	0.72	582.39	3.27
s1494	6.45	0.14	140.47	1.19	998.18	18.49
s1488	13.26	0.20	427.02	12.36	1015.46	16.85
s1423	2140.56	-	3784.56	2589.40	-	-
s3271	601.39	3.41	5607.40	33.73	-	-
s3384	2486.12	220.16	-	380.26	-	1445.66
s4863	7076.11	234.62	-	257.70	-	362.43
s5378	-	840.60	-	1170.56	-	2002.50
s6669	-	32.02	-	353.54	-	-
s9234	-	35.60	-	-	-	-
s15850	-	54.03	-	-	-	-
s35932	8268.39	3855.50	-	-	-	-

less than 10 minutes, the best result compared to all other simplification techniques.

As for the negative key-condition-tracking, for large key vectors recovering good cubes was challenging. However, with this method every deobfuscation run can add new disqualifying key-conditions to the cube space allowing parallelization and ensuring that the information from a failed deobfuscation task is not discarded. Generalizing negative conditions through simulations and SAT calls is a topic of our future research.

V. Conclusion

In this paper we presented an array of dynamic simplification techniques for faster sequential deobfuscation. With a proof-of-concept implementation we demonstrated two orders of magnitude improvement compared to the baseline algorithm using a sate-of-art model checker. We have released our attack tool and plan to extend it by improved cube generalization, CNF simplification, advanced AIG simplification, and unbounded model checking techniques [10].

Acknowledgement

This work was partially supported by DARPA (FA8650-18-1-7822).

References

[1] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, p. 6, 2016.

[2] R. Torrance and D. James, "The state-of-the-art in ic reverse engineering," in *Proc. Int. Conf. on Cryptographic Hardw. and Embed. Systems*. Springer, 2009, pp. 363–381.

[3] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ics using split fabrication," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2014.

[4] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. ACM, 2018, pp. 147–152.

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

[6] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*. IEEE, 2017, pp. 49–56.

[7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2017, pp. 46–51.

[8] M. E. Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential integrated circuits without scan access," *arXiv preprint arXiv:1710.10474*, 2017.

[9] T. Meade, Z. Zhao, S. Zhang, D. Z. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *The IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[10] Attack tool and benchmarks: https://bitbucket.org/kavehshm/neos/.

[11] M. E. Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism," *arXiv preprint arXiv:1703.10187*, 2017.

[12] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, 2019.

[13] A. R. Bradley, "Sat-based model checking without unrolling." in *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, vol. 6538. Springer, 2011, pp. 70–87.

[14] N. Een, "Cut sweeping," *Cadence Design Systems, Tech. Rep*, 2007.

[15] Z. Hassan, Y. Zhang, and F. Somenzi, "A study of sweeping algorithms in the context of model checking," *Ganesh Gopalakrishnan University of Utah USA*, p. 30, 2011.

[16] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*. IEEE Computer Society, 2004, pp. 50–57.

[17] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*. FMCAD Inc, 2011, pp. 125–134.

[18] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*. IEEE, 2015, pp. 137–143.

[19] K. L. McMillan, "Interpolation and sat-based model checking," in *International Conference on Computer Aided Verification*. Springer, 2003, pp. 1–13.