2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST) | 978-1-6654-6114-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASIANHOST56390.2022.10022024

A Structural and SAT Analysis of SANSCrypt

James Geist[*], Shaojie Zhang[*], Yier Jin[†], Travis Meade[*]
[*]*Department of Computer Science, University of Central Florida*
[†]*Department of Electrical and Computer Engineering, University of Florida*
jimg@knights.ucf.edu, shzhang@cs.ucf.edu, yier.jin@ece.ufl.edu, travis.meade@ucf.edu

***Abstract*—**
As IP theft remains an important problem, and attacks against logic locking mechanisms are becoming more sophisticated, the complexity of proposed encryption schemes are also becoming more complex. One goal of adding complexity is to increase the required time to perform an attack by forcing the required SAT problems to become so unwieldy, or require so many individual solves, that it is infeasible to break the encryption in reasonable time. In this paper, by discussing an attack against one such scheme, we demonstrate a combination of structural and SAT instances can break a design.

Keywords: Netlist Reverse Engineering, Hardware Security, Netlist Security Enhancement

I. Introduction

Many of the challenges involved in developing Integrated Circuits (ICs) at the VLSI scale boil down to NP-hard problems [14]. Developing optimal IC layouts to reduce overhead and increase performance has improved incrementally using heuristic solutions. These slightly better improvements lead to more powerful computers that can search for more efficient IC layouts. The cycle of improvements of to hardware creates a bootstrapping model that drives up IC production. Intellectual Property (IP) providers will continue to develop new ICs either to improve current designs by using better transistor and/or layout technology, meet the current demands for IC, or continue production of novel designs for consumers.

Not every IP developer has access to their own foundry at the needed fabrication level. For this reason it is not uncommon for IP providers to outsource part of the fabrication process to untrusted, third party foundries. A potential drawback to using third parties is the potential for foundries or others involved in the production process to determine the high level function involved in the IP by way of Reverse Engineering [9]. With knowledge of such high level function, meaningful and damaging Hardware Trojans can be implanted thereby compromising any combination of confidentiality, integrity, and/or accessibility (CIA) [6].

Researchers have aggressively investigated methods to reestablish IC CIA. Many methods are reduced to either modifying the fabrication process to secure designs, (e.g. split manufacturing) or obfuscating the designs such that high-level recovery becomes intractable [12]. Both methods can be broken down into several approaches. This paper will focus on obfuscation. Obfuscation can involve renaming wires, modifying gates such that their functionality is unknown to those outside of the production process, or inserting additional logic which can corrupt the design when haphazardly using the IC. The additional logic is typically bypassed either by some sequential input which involves as state machine built into the design or by applying some static key that fixes some part of the ICs combinational behavior . Since exploring all possible methods is unfeasible for a single paper, this paper will focus on developing and demonstrating attacks of sequential locking schemes.

The contributions of this paper are:

- An automated workflow which unlocks a SANSCrypt design in much less time than found by the original authors, which demonstrates how a combination of structural and state analysis together can reduce an unlocking problem to a more tractable form.
- A SAT problem formulation which allows discovery of an FSM's state graph, without the need to unroll the design multiple times to account for the passage of time.
- Recommendations for SANSCrypt, which generalize to other sequential locking schemes, which make it less vulnerable to the attack methodology we propose.

The rest of the paper is orgazized as follows. In Section 2, a brief outline of attacks and defenses of sequential and combinational protections schemes is presented. In Section 3, the defense model is discussed, and motivation of the attack is presented. Section 4 presents the result of the proposed attack on the discussed defense. Section 5 offers recommendations based on the method used in this paper for exploitation. A brief conclusion of the research is presented in Section 6.

II. Related works

A. Attacks

In [7], El Massad et al. introduce a SAT attack for camouflaged combinational circuits. A SAT problem is set up to discover circuit inputs (called discriminating inputs) which, for different hypotheses of values for camouflaged gates, the circuit outputs differ. By using an oracle circuit, invalid hypotheses can be eliminated, until only one working hypothesis remains. In [10], the authors extend this solution to sequential circuits. Using sequences of input values, the method performs bounded queries on the circuit, increasing the length of the input sequence, until the algorithm guarantees that only one hypotheses produces correct results. To accomplish this, the sequential circuit is duplicated the requisite number of times (referred to as unrolling) to account for the number of clock cycles being queried. While the work discusses obfuscation in terms of camouflaged gates, the arguments hold for sequential encryption using keys as well.

Shamsi et al. [13] examine the SAT problems generated in the sequential algorithm in [10] and find multiple points of redundancy in the formulation. By reformulating the problem with multiple optimizations, they are able to speed up the solution significantly. However, the need to unroll the netlist remains.

Another sequential attack discovers the FSM registers and enumerates the transitions. If the locking is dependent upon transitioning to a known state in the FSM before the circuit will function, then the transition graph can help discover a path to the start state. RELIC [8] and RELIC-FUN [15] both attempt to distinguish FSM control signals from other circuit signals using structural analysis, based on the idea that control signals implement boolean functions which are unlikely

Authorized licensed use limited to: University of Central Florida. Downloaded on March 24,2023 at 13:17:27 UTC from IEEE Xplore. Restrictions apply.

to be duplicated elsewhere in the circuit. RELIC examines purely structural components of signals such as gate type and fan-in size; RELIC-FUN takes cuts of each register's fan-in cone and counts how many times the computed boolean function occurs in that context. REFSM [9] can, given a state word constructed from RELIC or RELIC-FUN, recover the transition graph of the FSM by recursive constraint-based solving of the netlist.

B. Defenses

SLED [17] embeds a PRNG in an external secure memory module and uses the PRNG bits as a key; incorrect key values cause the design to malfunction. In order to avoid attackers from discovering the key algorithm by inspection of the netlist, SLED's key detection unit examines properties of the key transitions rather than the key transitions themselves.

SANSCrypt [16] attempts to increase the complexity of deobfuscation by requiring aperiodic reauthentication with pseudo-random keys. Both the key value and interval between authentications depend on a PRNG. By allowing the circuit to run normally for many clock cycles between authentications, a SAT solution would have to unroll through those many cycles to get to the next required key. In this paper, we will analyze SANSCrypt in detail.

III. Motivation and Proposed Solution

A. Motivation

In the results section of [16], the authors apply a sequential SAT-based attack as in [11] to SANSCrypt to measure the time required to recover the initial key sequence. They then relax their threat model to allow the attacker to know when future authentication phases will occur. Given this knowledge, they unroll the netlist the appropriate number of times and reapply the attack, noting that the effort is exponential in the number of key sequences.

While this type of analysis is common when evaluating protection methods, a sequential SAT analysis is only one weapon in the attacker's arsenal when the threat model includes, as it does in the SANSCrypt paper, access to the encrypted netlist. While the proposed method attacks the SANSCrypt method with more sophisticated techniques which combine structural and state-based algorithms, this paper's goal is not to expose weaknesses in SANSCrypt itself. Rather, this paper demonstrates that when evaluating designs for weakness, there can be multiple points of failure; a more thorough analysis may find faster ways to unlock a design, sometimes using the very complexity that was introduced to defeat such attacks.

B. Overview of SANSCrypt

As mentioned in Section II, SANSCrypt adds complexity by requiring an external circuit to provide a pseudo-random key at pseudo-random intervals. Figure 1 from the original work shows a block diagram of the scheme. In the top half, sequential and combinational logic implement an FSM that contains both encrypted mode states which handle authentication and functional mode states which drive normal operation of the device. As in HARPOON [5], a key presented across multiple post-reset clock cycles on the primary inputs causes transitions through the encrypted mode states which eventually cause a transition into the starting state of the functional mode part of the FSM.

The lower half of the figure shows components unique to SANSCrypt. A pseudo-random number generator (PRNG) determines both the number of cycles between authentications and the required key. Unlike HARPOON, SANSCrypt authentication can start in any state in encrypted mode; when authentication starts, the state register is loaded from the PRNG with a value guaranteed to be an E state. After authentication a target clock cycle count is loaded from the PRNG; after a counter reaches the target, authentication starts again. The backjumping FSM (BJ-FSM) is responsible for coordinating transitions between encrypted and functional mode in both directions. While in encrypted mode, the encryption FSM (ENC-FSM) is combined with the FSM's outputs, causing them to be corrupt and changing through the authentication phase.

Figure 2, also from the original work, shows the transition diagram of the FSM. The blue arrows from functional mode indicate transitions that can happen from any N state to any E state the PRNG can generate. The red arrows indicate reentry into functional mode from encrypted mode. From the outgoing state of encrypted mode, $E4$, any N state could be the next state. In fact, we must return to the N state that was interrupted when authentication began, so there must be a save register that holds this state while authentication proceeds.

Searching for keys in SANSCrypt using existing SAT algorithms is shown to be intractable as the discriminating input sequences are very long; they must traverse the interval between authentication phases. When discussing the performance of the algorithm, the authors only consider using existing tools (RELIC, REFSM) without considering a more sophisticated attack which combines those algorithms with structural analysis.

C. Threat Model

We will assume the same threat model as in the original work. We as the attacker are allowed access to the encrypted netlist, an oracle which produces correct outputs for any inputs, and knowledge of how SANSCrypt is implemented. There is no scan chain access, nor can we probe the internal state of the oracle.

Although it is not explicitly stated, we understand that the authors' intent is that the seed or key for the PRNG is in protected memory. They state "... when and where the the back-jumping operation happens is unpredictable unless the attacker is able to break the PRNG or find its seed." [16]. We therefore assume that while we may recover the PRNG algorithm from the encrypted netlist, the starting state is a secret hidden from us.

D. Structural Analysis

In the first part of the analysis, we examine the register dependency graph $G_{rd}(V_{rd}, E_{rd})$, where the vertices V_{rd} are the registers of the design. If the value of register S in the next clock cycle depends on the current value of register R in the current clock cycle then S depends on R, and the edge $R \rightarrow S$ exists in E_{rd}. A *strongly connected component*, or *SCC*, is a set of vertices in a graph such that for any vertex in the SCC, there is a path in graph to any other vertex in the SCC. An SCC in G_{rd}, then, is a set of registers that all depend on each other, possibly over multiple clock cycles.

In figure 1, it is easy to see that the sequential logic, the backjumping FSM, and the encryption FSM are all mutually dependent. Although not shown, the counter is loaded from the PRNG by the backjumping FSM, and its value is also tested by the backjumping component to determine when to transition into authentication. Hence we would expect all of these registers to comprise one SCC in the dependency graph. The PRNG is free running, and thus does not depend on any of the aforementioned registers; the PRNG registers therefore form another SCC.

As the first step in analyzing a design, we use a breadth first search (BFS) of the netlist to build G_{rd}, and then Tarjan's Algorithm [1] to find the SCC sets of the dependency graph. We define dependency between SCC's as follows: if for SCC sets S_0 and S_1, there exists

Authorized licensed use limited to: University of Central Florida. Downloaded on March 24,2023 at 13:17:27 UTC from IEEE Xplore. Restrictions apply.

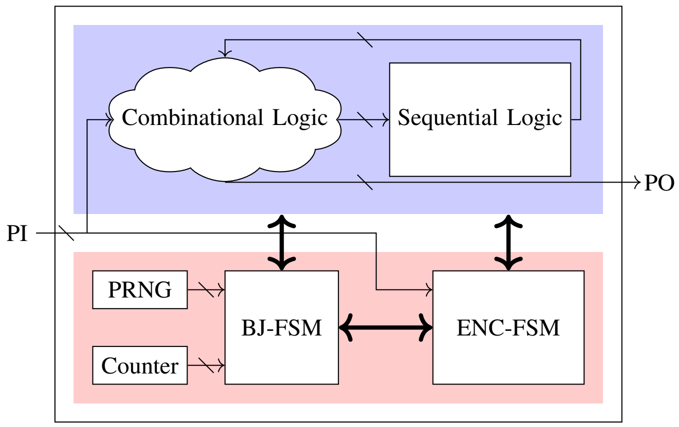

Fig. 1. Schematic view of SANSCrypt. [16]

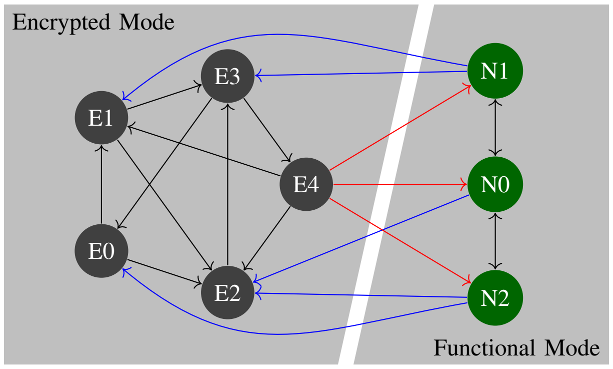

Fig. 2. State transition diagram of SANSCrypt. [16]

a register $R_0 \in S_0$ and a register $R_1 \in S_1$, and the edge $R_0 \rightarrow R_1$ exists in the dependency graph, then S_1 depends on S_0. This defines another graph, $G_{rscc}(V_{rscc}, E_{rscc})$, whose vertices are the SCC's of G_{rd} and whose edges represent SCC dependency. We search this graph for two SCC's, SCC_{SANS} (the large SCC containing the original state registers and most of the SANSCrypt machinery) and SCC_{PRNG} (the SCC containing the PRNG). From the functional description in [16], we can estimate a lower bound on $\|SCC_{SANS}\|$ and an upper and lower bound on $\|SCC_{PRNG}\|$. Using this method recovers SCC_{SANS} and the PRNG.

The next step in analysis is to find the state word registers R_{state} (labeled Sequential Logic in figure 1) from SCC_{SANS}. In figure 2, we note that upon reaching state $E4$, the circuit must transition back to the functional state it was in before authentication was requested. This implies the existence of another word register, of the same size as the state word, which we will call R_{save}. We also note that any path through the encrypted mode side of the state transition graph must visit $E4$ before reaching a functional state. Further, $E4$ may lead to *any* functional state as it will result in a load of R_{state} from R_{save}. Each flip-flop of R_{save} will be mutually dependent with a flip-flop of R_{state} in G_{rd}, and that no other flip-flop will depend on the flip-flops in R_{save}. For each flip-flop in SCC_{SANS}, we count the number of outgoing edges and collect the set of flip-flops with only one edge. These are the flip-flops with only one dependent. We call this set R'_{save} as it contains R_{save} but potentially some other flip-flops as well.

In order to determine R_{save} from R'_{save}, we make use of another observation: all of the registers of R_{state} will depend on each other. This is a property of FSM implementations: each state transition depends on the value of the current state and other circuit conditions. Let R'_{state} be all registers dependent on the registers of R'_{save}. We partition R'_{state} using a disjoint set algorithm. Initially, all registers in R'_{state} are in singleton sets. For each pair of registers $R_0, R_1 \in R'_{state}$, if there is an edge $R_0 \rightarrow R_1 \in E_{rd}$, we merge the sets containing R_0 and R_1. The largest of the result sets is reliably R_{state}. Algorithm 1 implements this method.

Algorithm 1 Recovering the State Register

```
function RECOVER-STATE-REG
    $dependents \leftarrow MAP()$                ▷ Compute dependent counts
    for $reg \in SCC_{SANS}.vertices$ do
        $dependents[reg] \leftarrow 0$
    end for
    for $edge \in SCC_{SANS}.edges$ do
        $dependents[edge.from] += 1$
    end for                          ▷ Build potential state register set
    $P'_{state} \leftarrow DISJOINT-SET()$
    $pregs \leftarrow SET()$
    for $reg \in SCC_{SANS}.vertices$ do
        if $dependents[reg] == 1$ then
            for $edge \in SCC_{SANS}.edges - from(reg)$ do
                $pregs.add(edge.to)$
                $P'_{state}.add-set(edge.to)$
            end for
        end if
    end for                          ▷ Build mutually dependent sets
    for $reg \in pregs$ do
        for $edge \in G_{rd}.edges - from(reg)$ do
            $P'_{state}.join(reg, edge.to)$
        end for
    end for
     return Largest set in $P'_{state}$
end function
```

E. State Analysis

With the state registers in hand, we proceed to state analysis. For this analysis, we will use a SAT solver as a way to query the netlist. The SAT problem we create represents the netlist with each flip-flop R split into two variables: the current register value R_{curr}, which is the signal the flip-flop's Q output drives, and the next register value (i.e. the value of the register after one clock cycle has passed) R_{next}, which is the signal that drives the flip-flop's D input. With state factored out, translating the remaining combinational logic into SAT clauses is trivial. This is similar to the construction in [10], except that we do not create a copy of the netlist for the next clock cycle. For each query, we add to the problem a small number of clauses constraining the primary inputs and current state register values. We then solve the SAT problem; if it is satisfiable, we read back the next register values from the model. Algorithm 2 shows pseudocode for building the state transition graph using this method.

Using Algorithm 2, we build the transition graph G_{trans} defined as (V_{trans}, E_{trans}), where V_{trans} are the possible states and E_{trans} are the possible transitions. Our first goal is to find the state in encrypted mode which transitions to functional mode (in figure 2, this is node $E4$). We will call this the pivot state S_{pivot}. We note as above that any path from encrypted mode (E-space, the set of all E states) to functional mode (N-space, the set of all N states) must pass through

Authorized licensed use limited to: University of Central Florida. Downloaded on March 24,2023 at 13:17:27 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 FSM State Exploration

```
function BUILD-STATE-GRAPH($n$ : $netlist$, $st$ : $state\ word$, $rs$ : $reset\ state$)
    for $r \in n.flipflops$ do
        $next[r] \leftarrow r.D$
        $curr[r] \leftarrow r.Q$
        $n.remove(r)$
    end for                                    ▷ $n$ is now stateless
    $sat \leftarrow$SAT-INSTANCE($n$)
    $q \leftarrow$QUEUE()
    $seen \leftarrow$SET()
    $q.push(rs)$
    while $q$ is not empty do                  ▷ BFS to find transitions
        $cr \leftarrow q.pop()$
        if $cr \notin seen$ then
            $seen.add(cr)$
            $con \leftarrow$SAT-CONSTRAINTS()
            $con.add(curr[st] = cr)$
            while $sat.solve(con)$ succeeds do
                $nx \leftarrow sat.model(next[st])$
                $q.push(nx)$
                $con.add(next[st] \neq nx)$
                Save $cr \rightarrow nx$ as a transition
            end while
        end if
    end while
end function
```

S_{pivot}. As constructed, the combined transition graph of the entire FSM is one SCC. Any E-space node may reach any N-space node by passing through S_{pivot}, and any N-space node may reach any E-space node by virtue of a state load from the PRNG when the authentication phase starts.

Figure 3 shows this situation. E is the set of all E states, N is the set of all N states, and U is the set universe; i.e. all possible $2^{\|R_{state}\|}$ bit patterns. The solid green arrows represent intended transitions; from the pivot state to N-space, and from N-space back to E-space. The dashed red arrows show unintended transitions: from the pivot state back to E-space, and to any other state in the set universe. These unintended transitions will not happen in the actual design; however, they are found by our SAT transition finder as transitions are only constrained by the state of the circuit in the previous clock cycle. The constraints that present the red transitions from being taken happen several cycles earlier, when R_{save} is loaded from R_{state}. We note this as a potential performance bottleneck: our transition graph solver must deal with the worst case of an FSM that contains all possible states, as least until we have identified and removed the pivot state. Fortunately many real-world FSM's have a small number of state word bits.

For most states, removing the state will shrink the SCC by that one state, as the E-space states will be densely connected for key variation and all N-space states have transitions from the pivot node and back to arbitrary E-space nodes. However, if the pivot node is removed, then all of the red transitions in Figure 2 are also removed. N-space is then no longer accessible from E-space, and the transition graph will have two SCC's. We can exploit this fact by removing each state and examining the resulting SCC's in the transition graph.

With the pivot node identified, extracting the key required from any E-space node is trivial. As we have the transition graph, we can use either breadth search first (BFS) or depth first search (DFS) to

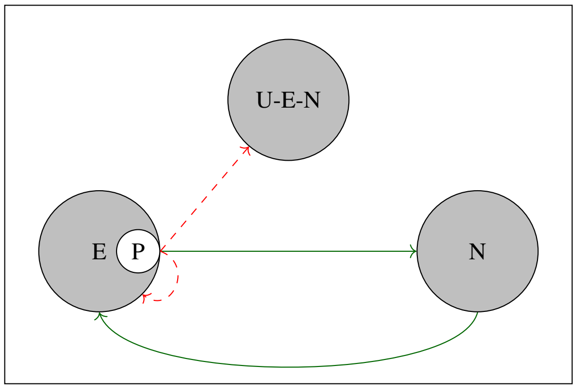

Fig. 3. State space of SANSCrypt encrypted FSM. [16]

find the path from any node in E-space to S_{pivot}, and we can use the SAT problem we have already created to find input values that will cause each needed transition.

F. Finding the Auth Load Signal

If, when authentication starts, the state register is loaded from the PRNG, there must be some signal that enables that load. We can use the SAT solver to find this signal. We assume that this signal can be found in the combinational logic connecting the registers in SCC_{SANS}. For each such potential signal P, we use the solver as in Algorithm 2 to find all the reachable states from the functional mode start state. We do this twice, once asserting P and once asserting the complement, $\neg P$. If in one case only N-space states are reachable, and in the other case E-space states are reachable, then we take P as the load signal. Keeping P in the state where only N-space states are reachable disables authentication from transitioning into E-space.

G. Defeating the Encryption

We present two ways of using the progress so far to unlock the design. First, we assume we cannot recover the PRNG sequence. Then, we discuss the case where we can recover the PRNG and provide a better solution in that case.

If we cannot recover the PRNG sequence, then we have no way of predicting the authentication intervals. However, because we understand the structure of the FSM, we can unlock the design by modification. We know that at reset, the proper key sequence on the inputs will cause the FSM to transition to S_{pivot}, and then to the start state of the functional mode. We can determine that start state by looking for the reset value of R_{save}, which will be loaded into the state word during the transition out of S_{pivot}. We have previously found *authload*, the signal which loads the state word with an E state to force authentication. If we cut *authload* from the design and resynthesize it, then the FSM will never go into authentication, and we are free to use it as we please.

If we can recover the PRNG sequence, then we do not have to resynthsize; we can use the off-the-shelf hardware by replicating the external key circuit. Given the netlist we described above, we can simulate the design's intended outputs for any sequence of inputs, as if the encryption did not exist. We are allowed access to an oracle: the hardware, properly working with the intended key generation attached, with the caveat that we cannot read any internal state. If we run the simulated netlist in lock step with the oracle, the two will eventually disagree; the disagreement occurs when the simulated circuit is still executing in functional mode, while the oracle has entered authentication. We now have two important data points:

Authorized licensed use limited to: University of Central Florida. Downloaded on March 24,2023 at 13:17:27 UTC from IEEE Xplore. Restrictions apply.

$\Delta auth$, the number of clock cycles until the first authentication past the initial startup authentication, and t_0, the clock cycle at which functional operation began. Note that since the authentication counter is loaded at t_0, we now know that the value of the PRNG at t_0 is $\Delta auth$.

If the PRNG is implemented as an LFSR, as proposed in the original work, then these data points are sufficient to break the encryption. We have shown how to determine the PRNG registers. From that and the encrypted design, the specification for the LFSR can be obtained. We assume per the original work that the seed is protected. However, we can derive the original polynomial that the LFSR is based on by examining the encrypted netlist. It is a property of LFSR's that, by implementing an LFSR with mirrored taps from the original LFSR, the mirrored LFSR will produce the same output as the original LFSR but in reverse order. [3] If $r(x)$ is the random number generator, then we have $r^{-1}(x)$ from this construction. We know at cycle t_0, the PRNG value is $\Delta auth$. The PRNG value at reset, the seed value, is therefore $r^{-t_0}(\Delta auth)$.

H. Runtime

Building the register dependency graph using BFS and finding the associated SCC's using Tarjan's algorithm both take $O(\|V_{rd}\| + \|E_{rd}\|)$ time. Searching for the SCC_{SANS} and the PRNG is a check over each SCC and each of its dependencies, so the bound is $O(\|V_{rscc}\| + \|E_{rscc}\|)$. Merging two sets with the disjoint set algorithm takes $O(n)$ where n is the number if items in the set universe. When finding R_{state} from R'_{state}, we check each state and potentially merge all states into one set, for a bound of $O(\|R'_{state}\|^2)$.

Building the state graph is the most expensive part of the algorithm. As we are traversing the graph via a breadth first search (BFS), the bound is $O(\|V_{trans}\| + \|E_{trans}\|)$ which is linear in the number of states and transitions. However, the primitive operation executed on each step is a SAT solve, which is known to have a worst-case exponential bound in the length of the formula to be satisfied, which in turn is proportional to the netlist size. Two things help us here: much research into SAT solver heuristics which only take the upper bound in pathological cases, and the fact that we do not need to unroll the netlist multiple times in the same SAT instance to capture the concept of state changes over time. At worst, we will add N clauses of W variables on each problem, where N is the number of transitions out of the state we are examining, and W is the number of variables (bits) in the state word.

Finding the pivot state requires running Tarjan's SCC algorithm once per state in the transition graph. This is $O(\|V_{trans}\|^2 + \|E_{trans}\| \cdot \|V_{trans}\|)$. If the transition graph is dense and has approximately a directed edge from each state, to each state, then $\|E_{trans}\|$ approaches $\|V_{trans}\|^2$ and the complexity is $O(\|V_{trans}\|^3)$.

Finding the auth load signal takes $O(\|sigs\|)$ SAT solver runs, where $sigs$ is the set of all signals in the netlist.

IV. Experimental Results

We implemented the algorithms described in section III in C++ using MiniSat [4]. All experiments were run on a 3.3GHz Intel i7-5820K system with 64G of memory.

In the original work, the authors use s1238 from the ISCAS '89 suite as a performance test in the case where the threat model is relaxed to allow the attacker to know when to apply the correct key sequences. This circuit is characterized as combinational logic with randomly inserted registers [2]. The authors used existing tools and an attack based on unrolling the circuit multiple times [11] in order to recover the initial key. While we wanted to show results on a circuit of similar scale, we felt that the randomness of s1238 combined with SANSCrypt implementation might demonstrate unfair advantage to our algorithm, as the SANSCrypt elements have more structure. We therefore implemented a sequential design, SANS-ALU, of similar scale: a circuit which includes 3 4-bit registers and a simple FSM-driven ALU. The FSM was then augmented with the SANSCrypt protection and compiled with Synopsys Design Compiler. Table I compares the metrics of the two circuits. We also included a larger design augmented with SANSCrypt: a PDP-8 minicomputer implemented in Verilog.

TABLE I
TEST CIRCUIT CHARACTERISTICS

Netlist	Primary Inputs	Primary Outputs	Registers	Gates	Recovery Time (s)
s1238	14	14	18	508	< 0.1
SANS-ALU	10	12	59	355	< 0.1
PDP-8	21	42	482	6329	14.5

In [16], the authors claim recovery of the initial key sequence using brute force REFSM in 4 seconds. With our implementation, we recover the important parts of the design as well as the initial key sequence in 0.1 seconds on small designs and 14.5 seconds on a larger design. As argued above, since both we and the original work used an LFSR as the PRNG for the implementations, and we are given knowledge of when the key must be applied, we do not need to unroll the circuit in order to determine future key sequences. We can simply work backwards from the time interval to determine the LFSR seed value, which will give us the starting state in E-space for every unlock sequence.

V. Recommendations

A. PRNG

Being able to synchronize to the PRNG in our attack is only possible due to the ability to efficiently construct an inverse for the PRNG function without knowledge of the initial seed. The original work mentions AES as a higher cost alternative (and does mention that the LFSR is less secure, though that claim is not discussed further). While using AES does nothing to impede the structural analysis, it will make reconstructing the PRNG stream much harder; the attacker will no longer be able to predict the reauthentication interval or entry E-state. Our recommendation is to always use a PRNG that is cryptographically secure.

B. FSM

The foundation of our attack on the FSM is that there is a single state which leads (via a reload of the state word from the previously saved state) from E-space to N-space. This led to an algorithm to partition the the states, from which it is trivial to remove the sequential locking. Inserting multiple transition states would make this part of the recovery much more difficult. Removing any one state from the transition graph will no longer cause a partition. If there are k pivot states, then we must examine $\frac{\|V_{trans}\|!}{k!(\|V_{trans}\|-k)!}$ combinations of states to remove, computing SCC's for each. With large enough k, this increases the bounds for the pivot state recovery step to $O(\|V_{trans}\|^4)$.

C. Temporal Instability

In parts of our analysis, we made use of the fact that the primary outputs become corrupt during the authentication phase, and that this may be a detectable condition. This assumption only holds if

Authorized licensed use limited to: University of Central Florida. Downloaded on March 24,2023 at 13:17:27 UTC from IEEE Xplore. Restrictions apply.

changes in the outputs are relevant on every cycle. If the outputs use asynchronous handshaking – for example, a memory controller, where the peer circuit must wait for the controller to indicate that a read or write operation has completed – then this assumption does not hold. SANSCrypt may be much more secure in such applications.

VI. Conclusion

In conclusion, we have shown an algorithm that can deconstruct a circuit protected with SANSCrypt and either remove the locking mechanism or, in some cases, provide the correct key sequence for any reauthentication request. While the algorithm is of limited use in and of itself, the methodology of finding the important parts of the locking mechanism via structural analysis and using that information to do more constrained state analysis is more generally applicable. Designers of protection should consider attack models more nuanced than an unrolled SAT attack on the entire design.

Acknowledgements

This work is partially supported by the National Science Foundation (NSF-1812071).

References

[1] R. Tarjan, “Depth first search and linear graph algorithms,” *SIAM JOURNAL ON COMPUTING*, vol. 1, no. 2, 1972.

[2] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in *IEEE International Symposium on Circuits and Systems,*, 1989, 1929–1934 vol.3. DOI: 10.1109/ISCAS.1989.100747.

[3] P. Udaya, “Euclid’s algorithm and lfsr synthesis,” in *2000 IEEE International Symposium on Information Theory (Cat. No.00CH37060)*, 2000, pp. 420–. DOI: 10.1109/ISIT.2000.866718.

[4] N. Sörensson and N. Een, “Minisat v1.13-a sat solver with conflict-clause minimization,” *International Conference on Theory and Applications of Satisfiability Testing*, Jan. 2005.

[5] R. S. Chakraborty and S. Bhunia, “Harpoon: An obfuscation-based soc design methodology for hardware protection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009. DOI: 10.1109/TCAD.2009.2028166.

[6] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, “Hardware trojan: Threats and emerging solutions,” in *2009 IEEE International High Level Design Validation and Test Workshop*, 2009, pp. 166–171. DOI: 10.1109/HLDVT.2009.5340158.

[7] M. El Massad, S. Garg, and M. Tripunitara, “Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes,” Jan. 2015. DOI: 10.14722/ndss.2015.23218.

[8] T. Meade, Y. Jin, M. Tehranipoor, and S. Zhang, “Gate-level netlist reverse engineering for hardware security: Control logic register identification,” in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2016.

[9] T. Meade, S. Zhang, and Y. Jin, “Netlist reverse engineering for high-level functionality reconstruction,” in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2016.

[10] M. E. Massad, S. Garg, and M. Tripunitara, “Reverse engineering camouflaged sequential circuits without scan access,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 33–40. DOI: 10.1109/ICCAD.2017.8203757.

[11] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, “Revisit sequential logic obfuscation: Attacks and defenses,” in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050606.

[12] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, “Provably-secure logic locking: From theory to practice,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1601–1618.

[13] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, “Kc2: Key-condition crunching for fast sequential circuit deobfuscation,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 534–539. DOI: 10.23919/DATE.2019.8715053.

[14] X. Chen, G. Liu, N. Xiong, Y. Su, and G. Chen, “A survey of swarm intelligence techniques in vlsi routing problems,” *IEEE Access*, vol. 8, pp. 26 266–26 292, 2020.

[15] J. Geist, T. Meade, S. Zhang, and Y. Jin, “Relic-fun: Logic identification through functional signal comparisons,” in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[16] Y. Hu, K. Yang, S. Nazarian, and P. Nuzzo, “Sanscrypt: A sporadic-authentication-based sequential logic encryption scheme,” *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, pp. 129–134, 2020.

[17] Y. Kasarabada, V. Muralidharan, and R. Vemuri, “Sled: Sequential logic encryption using dynamic keys,” in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020, pp. 844–847. DOI: 10.1109/MWSCAS48704.2020.9184664.

Authorized licensed use limited to: University of Central Florida. Downloaded on March 24,2023 at 13:17:27 UTC from IEEE Xplore. Restrictions apply.