A Particle Swarm Optimization Guided Approximate Key Search Attack on Logic Locking in The Absence of Scan Access

Rajit Karmakar and Santanu Chattopadhyay
Department of E&ECE, Indian Institute of Technology Kharagpur, India
Email: {rajit, santanu}@ece.iitkgp.ernet.in

***Abstract*—Logic locking is a well known Design-for-Security(DfS) technique for Intellectual Property (IP) protection of digital Integrated Circuits(IC). However, various attacks on logic locking can extract the secret obfuscation key successfully. Although Boolean Satisfiability (SAT) attacks can break most of the logic locked circuits, inability to deobfuscate sequential circuits is the main limitation of this type of attacks. Several existing defense strategies exploit this fact to thwart SAT attack by obfuscating the scan-based Design-for-Testability (DfT) infrastructure. In the absence of scan access, Model Checking based circuit unrolling attacks also suffer from scalability issues. In this paper, we propose a particle swarm optimization (PSO) guided attack framework, which is capable of finding an approximate key that produces correct output in most of the cases. Unlike the SAT attacks, the proposed attack framework can work even in the absence of scan access. Unlike Model Checking attacks, it does not suffer from scalability issues, thus can be applied on significantly large sequential circuits. Experimental results show that the derived key can produce correct outputs in more than 99% cases, for the majority of the benchmark circuits, while for the rest of the circuits, a minimal error is observed. The proposed attack framework enables partial activation of large sequential circuits in the absence of scan access, which is not feasible using the existing attack frameworks.**

***Index Terms*—IP Protection, Logic Locking, Oracle Guided Attack, Particle Swarm Optimization, Approximate Key;**

I. Introduction

The ever-increasing cost of building and maintaining a modern fabrication lab pushes the semiconductor industry towards going fabless and outsourcing the manufacturing process to dedicated specialist fab houses. However, this outsourcing reliant cost-effective global business model comes with a certain degree of security concerns. Since the entire layout of the design gets revealed to the third-party fab labs, it opens the backdoor of several security vulnerabilities such as Intellectual Property (IP) piracy, overbuilding, counterfeiting, and insertion of hardware Trojans (HT) [1]. Not only the third-party fab labs are threats to the secrecy of the hardware designs, but also an end-user can extract the internal details of an IC using advanced reverse engineering set-ups.

Logic locking [2], [3] is a popular countermeasure against these threats, which offers protection against the potential adversaries in the fab labs as well as the end-users. It obfuscates a design by introducing some redundant logic elements (often called as key-gates). The obfuscated design exhibits correct functionality only under the application of a correct key, which is the designer's secret. Without the knowledge of the right key, an unauthorized agent cannot use or redistribute the illegally acquired IP of a design.

Traditional logic locking techniques encrypts a netlist by inserting XOR/XNOR key-gates [2], [3]. Few other techniques also use AND/OR gates, MUX, or look-up-tables (LUT) as key-gates. Figure 1 shows an exmaple XOR/XNOR based logic locking. The original netlist has been converted to a logic locked netlist by inserting four additional XOR/XNOR key-gates. For a key value of "1001", the logic locked netlist behaves the same as the original netlist.

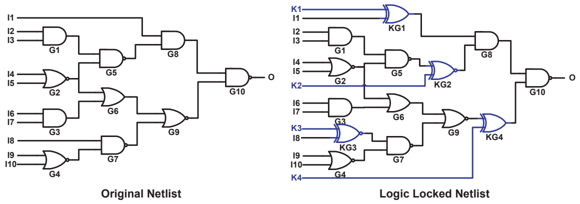


Figure 1: An example of XOR/XNOR based logic locking

However, time and again, the security of different logic locking strategies have been questioned by various attacks. Among numerous attacks, SAT attack has tremendous power of breaking virtually all possible combinational logic lockings [4]. However, the application of SAT attack is limited to only combinational circuits [5]. This attack can be applied on sequential circuits only if the internal state elements are externally accessible via the scan-based DfT infrastructure. This limitation has opened up a rather straightforward avenue of thwarting SAT attack by blocking the scan access to unauthorized users. Several recent approaches have relied on either obfuscating the scan data or blocking the extraction of scan response to mitigate SAT attack [6]–[8].

Alternatively, a sequential circuit can be unlocked using Model Checking based circuit unrolling attacks [5], [9]. These attacks use a Model Checker either Bounded or Unbounded to unroll the circuit and launching subsequent SAT attack [5], [9]. However, all of these attacks suffer from scalability issues. It has been observed that these attacks can successfully unlock only small sequential circuits. For larger circuits, either the attack framework crash or run for prohibitively long time.

The limitations of both SAT and circuit unrolling attacks on large sequential circuits raises the interesting question: ***Does logic locking, along with restricted scan access, guarantee IP protection of significantly large sequential circuits?***

This work is partially supported by Dept. of Higher Education, Science & Technology and Biotechnology, Govt. of West Bengal, India and "Synopsys CAD Laboratory Projects (CADL)", sponsored by Synopsys Inc., USA

978-3-9819263-4-7/DATE20/©2020 EDAA

In this paper, we look into the security of logic locking from a different angle. We investigate the following queries:

Is it mandatory to activate a locked IC using the correct key to be able to use it? What if we activate the IC using a key which produces nearly correct output? Is there any real-life application where an IC can be used, whose output slightly deviates from the correct outputs?

There exist a wide range of applications such as image, audio, video processing, and machine learning algorithms (speech and voice recognition), which can tolerate a certain degree of error in the outputs, provided the generated outputs have acceptable quality [10]. In all these applications, a piece of hardware can be used which may not always produce the exactly correct outputs. Approximate computing is a domain where partially incorrect circuits are frequently used without compromising on the performance of the design [10]. These wide ranges of applications of inaccurate circuits clearly indicate that it is not mandatory to activate a locked IC using the correct key. An IC is still very much usable if it produces small inaccuracy in the outputs. Therefore, we argue that an attacker might be more than happy to search for an approximate key, which can produce nearly correct outputs in the majority of the cases. In case, when finding the correct key using SAT and circuit unrolling attacks is not possible, the attacker can activate a locked IC using an approximately correct key and sell it at a lower price as a degraded version of the original circuit. This relatively poor quality IC can still be used with a fair degree of confidence.

This investigation has motivated us to search for an approximate key of a logic locked IC, rather than looking for the correct key to activate the IC. In this paper, we propose a Particle Swarm Optimization (PSO) guided attack on logic locking, which can find an approximate key that produces significantly accurate outputs. Unlike the SAT attack, the proposed attack does not require scan access. Also, unlike Model Checking attacks, its application is not limited to only small sequential circuits. For any given logic locked netlist, be it combinational or sequential, the attack can find an approximate key in a reasonable time, provided the golden oracle from an activated IC is available. Experimental results on several ISCAS'89 and ITC'99 benchmarks, and a case study on image sharpening application show that an approximate key can also be used to unlock a logic locked sequential circuit with a high degree of accuracy. Therefore, restricting SAT and circuit unrolling attacks does not always guarantee the protection of circuit functionality, even for larger sequential designs.

II. Particle Swarm Optimization Guided Attack on Logic Locking

A. Problem Formulation:

Given a logic locked IC, let *M* be the number of primary inputs, *O* be the number of primary outputs, and *K* be the number of key-inputs (key gates) in the circuit. A logic locked circuit C_{lock} can be represented by the mapping $Y = C_{lock}(I, K_X)$, where $I \in \{0,1\}^M$ represents the input, $Y \in \{0,1\}^O$ represents the output, and $K_X \in \{0,1\}^K$ represents an assignment of inputs to the key-gates. Let the input-output relation of an unencrypted circuit be modeled by the black box $\bar{C}$, given by $Y = \bar{C}(I)$. The attacker's objective is to find an assignment of key-inputs K_X^* to the key gates, satisfying the following relation,

$$C_{lock}(I, K_X^*) \approx \bar{C}(I), \qquad \forall I \in \mathcal{I}, \tag{1}$$

where $\mathcal{I}$ is the universal set of input sequences. Here, K_X^* may not be a singleton, that is, the approximately correct key for a logic locked circuit may not be unique.

B. Threat Model:

To solve this problem, we consider the following threat model, which is consistent with the standard threat model of the oracle-guided attacks on logic locking.

The attacker can be any rogue entity in the supply chain, post design house, either from the third-party foundry or an end user with access to the advanced reverse engineering tools. For extracting the values of the key-inputs, the attacker requires access to the following:

- A **locked gate-level netlist** C_{lock}, wherein the attacker has the knowledge of the logic locking scheme and the location of the key gates; however, the key input is unknown. Such a netlist can be obtained from the GDS II file in case of untrusted foundry or by reverse engineering the target IC obtained from the market.
- A **functional IC** $\bar{C}$ which has been activated by the designer using the correct key. It can be obtained from the market or through spurious means. The attacker uses this IC as a black box to get the oracle (input-output pairs).
- **However, we assume the attacker gets no access to the scan chain of the functional IC. Therefore, only the primary inputs/outputs of the activated IC are externally accessible.**

In this problem, the key search-space is 2^K which is prohibitively large for a sufficiently large value of *K*. Therefore, the brute-force attack for searching the correct key is out of the question. Particle Swarm Optimization (PSO) is a meta-search heuristic which can explore a significantly large search-space and quickly converges towards a near-optimal solution [11]. In this paper, we rely on a PSO guided attack framework to search an approximately correct key.

C. Particle Swarm Optimization Formulation

PSO is a population-based evolutionary technique designed by Eberhart and Kennedy [11]. It starts with an initial population of particles. Each particle corresponds to a probable solution to the problem being solved. Each particle has its fitness value, which is calculated using a cost function related to the problem. Particles evolve over generations guided by self and group-intelligence, and also via their inertia. Any PSO formulation involves choosing a proper representation of the particles, their fitness calculation, and defining an evolution policy. Next, we elaborate our PSO guided attack framework on logic locking. Table I introduce some of the terms and their corresponding definitions which have been used in our attack algorithm.

Table I: Few terms and their definitions used in Algorithm 1

Name		Definition
Particle	:	A particle corresponds to a K-bit key value
CostFunction	:	A function which calculates the fitness of a particle
Particle_Cost	:	The fitness value of the *Particle*
Pbest_Particle	:	Local best structure of a particle
Pbest_Cost	:	The fitness value of the *Pbest_Particle*
Genbest_Particle	:	The best particle of a particular generation
Genbest_Cost	:	The fitness value of the *Genbest_Particle*
Globalbest_Particle	:	The best particle over all the generation
Globalbest_Cost	:	The fitness value of the *Globalbest_Particle*

1) ***Particle Structure**:* For a key-size of K, each particle is represented by a K-bit binary number. Each bit of a particle represents an input to a key-gate. For example, the 4-bit key of the logic locked netlist of Figure 1 can be represented by a particle "1001".

2) ***Cost Function**:* Cost Function is a subroutine which calculates the fitness of each particle (Procedure CostFunction). It generates 500 random test patterns and simulates the activated IC using these test patterns to obtain the corresponding golden responses. After that, it applies the key represented by a particle structure to the locked netlist and obtains the corresponding outputs of the generated test patterns by simulating the circuit. Finally, it calculates the Hamming distances (HD) between the obtained and golden responses as the fitness value of the particle. A particle, which produces smaller Hamming distance, is considered to be a good particle. The fitness values of all the particles of a particular generation are evaluated based on the same 500 random test patterns. However, for different generations, we use a different set of 500 random test patterns. Although 500 random test patterns is a small subset of the entire input search-space, it is often sufficient to distinguish between good and bad particles. Since our objective is to find a key which produces minimal HD (ideally 'zero' HD), the PSO guided attack formulation tries to minimize the fitness value over the generations. The small subset of the verification test patterns helps us to expedite the process of eliminating the bad particles and converging towards the better particles.

```
Procedure CostFunction
Input: Activated IC; Locked Netlist; Particle;
Output: Hamming Distance;
Test_Pat = 500 random test patterns;
Current_Key = Particle;
Golden_Response = CircuitSimulation (Activated IC, Test_Pat);
Obtained_Response = CircuitSimulation (Locked Netlist, Test_Pat,
    Current_Key);
Hamm_Dist = Hamming distance between Golden_Response and
    Obtained_Response;
Return Hamm_Dist;
```

3) ***Evolution of Particles**:* In a PSO formulation, particles evolve over the generations, guided by three factors – its own intelligence, global (swarm) intelligence, and the inertia factor. A particle always remembers its history about its best structure over generations. This is called the local best (*Pbest_Particle*) of the particle. Among all the particles of a generation, the one which produces the best fitness value is marked as the best (*Genbest_Particle*) of the generation. Therefore, each particle has an associated *Pbest_Particle*, whereas each generation has a single *Genbest_Particle*. Over the generations, new particles are created using the *Replace* operator noted next.

Replace Operation: For the i^{th} generation, the *Replace* operator generates new particles by updating each particle of $(i-1)^{th}$ generation with its *Pbest_Particle* and the *Genbest_Particle* of $(i-1)^{th}$ generation, with some probabilities. This is done by applying the *Replace* operation at each individual bit position of a particle. For bit position j of a particle, the bit is replaced by the corresponding bit of the *Pbest_Particle* with a probability α. After the operator has been applied for local best, the same is done with respect to the *Genbest_Particle*, with a probability of replacement, β. In our experimentation, we have kept both α and β values at 0.1. Procedure Replace presents the pseudo-code for the *Replace* operation.

```
Procedure Replace
Input: Particle_a; Particle_b; prob;
for Each bit $i$ of a particle do
    Rand = A random number between 0 to 1;
    if Rand $<$ prob then
        Replace the $i^{th}$ bit of Particle_a with the $i^{th}$ bit of
          Particle_b;
```

Figure 2 shows an example of the *Replace* operation. In this example, the original particle is "0100111010", and the corresponding *Pbest_Particle* and *Genbest_Particle* are "0010101100" and "1001101001", respectively. The first application of the *Replace* operation replaces the 2^{nd} and 5^{th} bit of the particle with the bits of the same positions of *Pbest_Particle*. A further *Replace* operation updates the 4^{th} and 6^{th} bits of the particle with the same bits of the *Genbest_Particle*. Finally, we get a modified particle as "0001101010".

Original Particle	0	1	0	0	1	1	1	0	1	0	
Pbest_Particle	0	0	1	0	1	0	1	1	0	0	$\alpha = 0.1$
Genbest_Particle	1	0	0	1	1	0	1	0	0	1	$\beta = 0.1$
Rand	0.2	**0.05**	0.32	0.79	**0.02**	0.6	0.94	0.12	0.56	0.87	***Replace*(*Particle*,**
Particle Update	0	**0**	0	0	**1**	1	1	0	1	0	***Pbest_Particle*, α)**
Rand	0.73	0.47	0.19	**0.04**	0.62	**0.09**	0.93	0.43	0.27	0.31	***Replace*(*Particle*,**
Particle Update	0	0	0	**1**	1	**0**	1	0	1	0	***Genbest_Particle*, β)**
Evolved Particle	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	

Figure 2: An example of the *Replace* operation

Algorithm 1 presents the pseudo-code of the entire PSO-guided attack framework. For the initial generation, particles are generated randomly. In our experiment, we consider 1000 particle for each generation. The corresponding local best of each particle is initialized to itself. The *Cost_Function* determines the fitness of each particle and the *Genbest_Particle* is identified for the generation. In the successive generations new particles are generated using the *Replace* operator. The fitness of each of the newly generated particles of generation i are evaluated using the *CostFunction* to decide the new set of *Pbest_Particle* and *Genbest_Particle*, which guide the further evolution steps. At any point, the *Globalbest_Particle* stores the best particle that PSO has encountered until that point. There is a possibility that a PSO-guided search process may encounter a local minima. In such scenarios, we may not observe improvement in the result over the generations, even if there is a scope of improvements. To bypass such local minima in the key search process, we use an intermediate particle regeneration process if the *Globalbest_Particle* does not change over the last 50 generations.

Algorithm 1: *PSO Guided Attack on Logic Locking*

```
   Input  : Activated IC; Locked Netlist; Key size ($K$);
   Output : Approximate Key
1  begin
2      $N \leftarrow$ No of particles;
3      Gen_Count = 1;
4      Regenerate_Flag = 0;
5      Generate $N$ random particle of size $K$;
6      for Each particle do
7          Particle_Cost = CostFunction(Particle);
8          Update Pbest_Particle;
9          Update Genbest_Particle;
10         Update Globalbest_Particle;
11     while Gen_Count < MAXGEN do
12         for Each particle do
13             Replace(Particle, Pbest_Particle, $\alpha$);
14             Replace(Particle, Genbest_Particle, $\beta$);
15             Particle_Cost = CostFunction(Particle);
16             Update Pbest_Particle;
17             Update Genbest_Particle;
18             Update Globalbest_Particle;
19         Gen_Count ++;
20         if Globalbest_Cost does not improve over 50 generations then
21             if Regenerate_Flag = 0 then
22                 Regenerate_Flag = 1;
23                 RegenerateParticleSet(Globalbest_Particle, $N$);
24                 Goto line 6;
25             else
26                 Return the Globalbest_Particle as the key;
27                 Break;
28         if Globalbest_Cost = 0 then
29             Return the Globalbest_Particle as the key;
30             Break;
31     Return the Globalbest_Particle as the key;
```

Particle Regeneration: Using this process, we generate an entirely new set of N particles from the *Globalbest_Particle*, where N is the number of particle in each generation. Each of these newly generated N particles differ the *Globalbest_Particle* by p-bits, however, at different bit positions. In our experiment, we set $p = 10$. Procedure RegenerateParticleSet presents the pseudo-code of particle regeneration process.

Procedure RegenerateParticleSet

```
   Input: Globalbest_Particle, $N$
1  for $i \leftarrow 0$ to $N$ do
2      Generate a new particle by flipping $p$ random bits of the
         Globalbest_Particle;
```

In the successive generations, the newly regenerated particles evolve towards further improvements. The attack terminates if we get either a 'zero' fitness value of a particle or observe no further improvement in the last 50 generations or the generation count reaches the maximum generation count. The corresponding *Globalbest_Particle* is returned as the approximate key of the locked netlist.

III. Experimental Results

In this section, we present the results of our experimentation. We have launched our PSO guided attack on several ISCAS'89 and ITC'99 benchmark circuits, assuming no scan access. Each of these circuits has been logic locked using 64-bit and 128-bit keys. Two different types of existing XOR/XNOR-based logic locking schemes have been implemented. The first one inserts the key-gates at random locations inside the circuit [2], while the second scheme inserts the key-gates in optimal positions to ensure protection against key-sensitization and circuit partitioning attacks, as well as high output corruption for wrong keys [3].

The attack framework has been developed using a C code, which also uses a sequential circuit simulator that obtains circuit responses. A 2.20 GHz Intel (R) Xeon(R) CPU E5-2430 with 64 GB RAM has been used to launch the attack. The attack returns an approximate key once any of the terminating condition is reached. Unlike SAT and model checking attacks, the proposed attack does not derive the correct key by solving a set of Boolean equations. Instead, it explores the key-space intelligently to find an approximately correct key. Therefore, even if the fitness of a particle is zero, one cannot guarantee the correctness of the corresponding key, unless it is tested for all possible input patterns. Since applying all possible input patterns is practically infeasible, we have statistically verified the accuracy of the obtained approximate key. For this purpose, we have applied the key to the locked netlist and simulated the circuit using one million random test patterns. We have compared the outputs with the corresponding golden responses of the same set of test patterns and measured the error at the output in terms of Hamming distance.

Table II reports the results of the PSO guided attack on ISCAS'89 and ITC'99 benchmark circuits, which have been locked by placing the key-gates at optimal positions [3]. The first five columns of the table present the circuit descriptions in terms of the circuit name and the number of inputs, outputs, flip-flops, and logic gates present in it. The 6^{th} column reports the time required to find an approximate key when the circuits are locked with a 64-bit key. The next six columns present the error statistics of the approximate key for 1 million random test cases. The first of these six columns reports the percentage of cases where the approximate key has produced a 0% error at the output. The successive columns report the statistics of 1%, 2%, 3-5%, 6-7%, and greater than 7% erroneous outputs, respectively. The 13^{th} column of the table presents the attack time for $K = 128$, while the last six columns report the corresponding error statistics. Table III presents the same set of findings of the attack on random logic locking [2].

It can be observed from Table II that for six circuits, the attack could find a 64-bit key, which produces correct output in 100% test cases. For the rest of the circuits, the extracted key could produce correct outputs in more than 99% test cases. These results show the effectiveness of the attack in finding a key which can produce outputs with high accuracy. As can be observed from the table, the efficiency reduces slightly for $K = 128$. Still, in four circuits, we have found 0% error in more than 99.99% test cases. For the rest of the circuits, in the majority of the cases, the error lies within 2%, which is very much acceptable in most of the error-tolerant applications. Only in rare cases, we could observe an error of 3-5%. Error exceeded 7% in none of the observed cases.

Table II: Results and observations from the PSO guided attack on optimal logic locking [3]

Circuit Name	Inputs	Outputs	Flip-Flops	Logic gates	K = 64							K = 128						
					Time	Statistics of the % of erroneous outputs for the Approximate Key (for 1000000 random patterns)						Time	Statistics of the % of erroneous outputs for the Approximate Key (for 1000000 random patterns)					
						0%	1%	2%	3%-5%	6%-7%	>7%		0%	1%	2%	3%-5%	6%-7%	>7%
s5378	35	49	179	2958	7m2s	99.52%	-	0.48%	0.0024%	0%	0%	10m21s	71.44%	-	26.26%	2.28%	0.0108%	0%
s9234	36	39	211	5808	9m41s	100%	-	0%	0%	0%	0%	10m28s	87.44%	-	12.52%	0.0018%	0.0337%	0%
s13207	62	152	638	8589	16m12s	100%	0%	0%	0%	0%	0%	47m55s	43.07%	45.69%	4.570%	3.47%	3.19%	0%
s15850	77	150	534	10306	4m2s	100%	0%	0%	0%	0%	0%	50m23s	96.98%	1.46%	1%	0.0383%	0.0173%	0%
s38584	38	304	1426	20679	1h16m	100%	0%	0%	0%	0%	0%	4h53m	24.8%	35.27%	16.12%	10.19%	0%	0%
s38417	28	106	1636	23815	37m36s	100%	0%	0%	0%	0%	0%	1h23m	99.99%	0.0004%	0%	0%	0%	0%
b14	32	54	245	10012	5m46s	99.37%	-	0.0079%	0.1%	0.517%	0%	6m31s	100%	-	0%	0%	0%	0%
b17	37	97	1415	32192	15m28s	99.88%	0.073%	0.032%	0.0098%	0%	0%	16m7s	99.99%	0.0017%	0.0001%	0%	0.0003%	0%
b18	37	23	3320	114561	8h13m	99.3%	-	-	0.7%	0%	0%	10h5m	100%	-	-	0%	0%	0%
b19	24	30	6642	231266	9h21m	100%	-	-	0%	0%	0%	11h34m	97.03%	-	-	2.97%	0%	0%

Table III: Results and observations from the PSO guided attack on random logic locking [2]

Circuit Name	Inputs	Outputs	Flip-Flops	Logic gates	K = 64							K = 128						
					Time	Statistics of the % of erroneous outputs for the Approximate Key (for 1000000 random patterns)						Time	Statistics of the % of erroneous outputs for the Approximate Key (for 1000000 random patterns)					
						0%	1%	2%	3%-5%	6%-7%	>7%		0%	1%	2%	3%-5%	6%-7%	>7%
s5378	35	49	179	2958	2m39s	99.6%	-	0.4%	0%	0%	0%	8m42s	51.69%	-	47.88%	0.41%	0.238%	0%
s9234	36	39	211	5808	57s	99.3%	-	0.3%	0.4%	0%	0%	2m19s	99.7%	-	0.018%	0.284%	0%	0%
s13207	62	152	638	8589	2m34s	48.7%	50.93%	0.37%	0%	0%	0%	4m49s	49.46%	48.1%	2.5%	0%	0%	0%
s15850	77	150	534	10306	4m40s	99.3%	0.64%	0%	0%	0%	0%	8m23s	98.01%	1.37%	0.54%	0.069%	0%	0%
s38584	38	304	1426	20679	48h25m	99.89%	0.11%	0%	0%	0%	0%	2h45m	98.86%	1.03%	0.115%	0%	0%	0%
s38417	28	106	1636	23815	5m40s	100%	0%	0%	0%	0%	0%	18m57s	100%	0%	0%	0%	0%	0%
b14	32	54	245	10012	17m47s	100%	-	0%	0%	0%	0%	22m31s	97.61%	-	0.8%	1.3%	0.29%	0%
b17	37	97	1415	32192	9m47s	100%	0%	0%	0%	0%	0%	15m28s	98.31%	1.23%	0.46%	0%	0%	0%
b18	37	23	3320	114561	6h25m	100%	-	-	0%	0%	0%	9h27m	99.63%	-	-	0.37%	0%	0%
b19	24	30	6642	231266	8h7m	100%	-	-	0%	0%	0%	10h47m	99.08%	-	-	0.92%	0%	0%

A similar kind of results have been observed in the case of random logic locking as well (Table III). For $K = 64$, the extracted key could produce correct output in more than 99% cases for all the circuits, except $s13207$. Similarly, for $K = 128$, we have observed correct outputs in more than 97% test cases, except $s5378$ and $s13207$. However, in both circuits, the error crosses more than 2% vary rarely. The only difference between the observations of the Tables II and III is the time required to launch the attack. It has been observed that the PSO guided attack could find an approximately correct key much faster when the key-gates are inserted randomly. Nonetheless, the run-times of the attack on optimal logic locking are also very reasonable, even for relatively larger sequential circuits.

Figure 3 demonstrates how the PSO guided attack framework starts with a set of random keys and converges towards approximately correct key over the generations. It plots the fitness value of the best particle of each generation until the attack terminates. As we can observe, the best key of the first generation produces more than 20-bits error on average for the circuit $s15850$. Over the generations, this value improves, and eventually, we obtain 'zero' bit error on average. This explains the power of PSO in exploring large search space. It can be observed that the attack terminates early for $K = 64$ compared to $K = 128$. This is expected since the search space for the 64-bit key is much less compared to the 128-bit key. A similar kind of trend can be observed in the rest of the circuits as well. Due to space limitations, we have not shown them all.

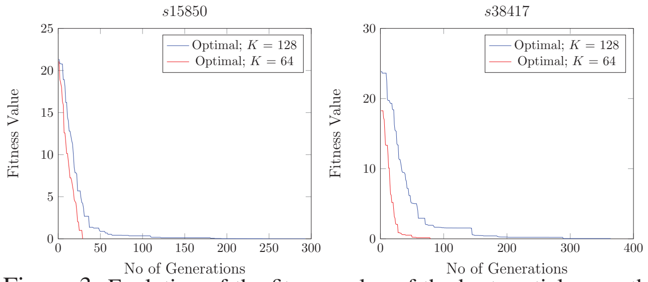


Figure 3: Evolution of the fitness value of the best particle over the generations

We have compared our attack with the BMC based circuit unrolling attack, which uses NuSMV as the back-end model checker [12]. We have developed the BMC attack set-up and launched the attack on all the experimental benchmark circuits. Table IV presents the comparison between PSO guided approximate key search attack and BMC guided circuit-unrolling attack. It can be observed that out of 10 experimented circuits, BMC based attack is successful only on three circuits, which are relatively smaller in size. These observations clearly show the limitations of circuit-unrolling attacks. On the other hand, the PSO based attack could successfully extract an approximate key for all the circuits irrespective of their sizes. As shown in Tables II and III, the accuracy of the obtained keys are also very high. This demonstrates the importance of this attack as an effective alternative framework of attacking large sequential circuits, which are otherwise secure against SAT and circuit-unrolling attacks.

Table IV: Comparison of run time with BMC Attack ($K = 128$)

Circuit Name	Attack run time		Circuit Name	Attack run time	
	PSO	BMC [5]		PSO	BMC [5]
s5378	10m21s	-	**s38417**	1h23m	-
s9234	10m28s	-	**b14**	6m31s	5h50m
s13207	47m55s	24m4s	**b17**	16m7s	-
s15850	50m23s	1h48m	**b18**	10h5m	-
s38584	7h53m	-	**b19**	11h34m	-

A. *Case Study on Image Sharpening Application*

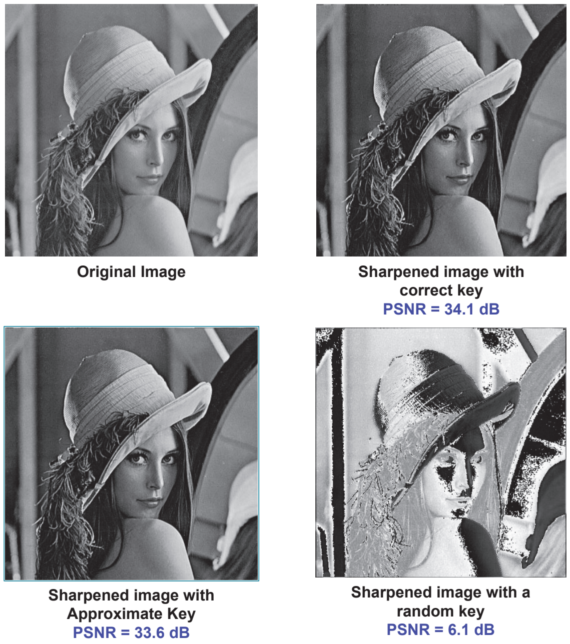
In order to demonstrate the practicality of the attack in real-life circuits, we have chosen the image sharpening application.

Image sharpening is an image processing technique which is used to enhance the quality of digital images. It uses a filter to sharpen blurred images. The processed image quality is measured by the peak signal-noise ratio (PSNR).

We have implemented the image sharpening algorithm and logic locked the corresponding gate-level netlist using a 128-bit key. For the experimentation purpose, we have taken a standard 512×512 8-bit Lena image. The PSO guided attack has been applied onto the locked netlist, which has returned an approximate key. For the investigation purpose, we have considered the following three cases:

1) First, we have applied the correct key to the locked netlist and extracted the corresponding sharpened image.
2) Next, we have used the approximate key and extracted the corresponding sharpened image.
3) Finally, we have extracted the sharpened image by applying a random key having 25% wrong key bits.

Figure 4 presents the output images of our experiment. We can observe a minimal difference in the PSNR values between the images, which have been extracted using the correct key and the approximate key, respectively. However, the random key results in a corrupted image with inferior PSNR value. Although the output of the approximately correct key is very much usable, one cannot use the output of the random key for further processing. This experiment shows the effectiveness of the attack on error-tolerant applications.



Original Image

Sharpened image with correct key
PSNR = 34.1 dB

Sharpened image with Approximate Key
PSNR = 33.6 dB

Sharpened image with a random key
PSNR = 6.1 dB

Figure 4: A case study of the PSO guided attack on image sharpening application

B. Limitation of PSO Guided Attack

One limitation of the proposed PSO guided attack is it is not applicable on point-function based SAT resilient techniques [13], [14]. For any wrong key, the point-function based schemes corrupt only a limited number of output bits (sometimes only a single output bit) for very few input patterns (sometimes only one input pattern). For the majority of the inputs, these schemes produce correct output for any random wrong key. This inhibits the PSO algorithm from differentiating between the good and the bad particles. Nonetheless, for these kinds of defense mechanisms, it is not necessary to find an approximate key to activate the locked IC. Any random key can be used as an approximate key with the assurance that the circuit would produce correct output in the majority of the cases. This also highlights the limitations of point-function based techniques in protecting the designer's IP.

IV. Conclusion

In this paper, we have highlighted the limitation of the existing logic locking strategies in protecting designer's IP even if the techniques are resistant to SAT and circuit unrolling attacks. An approximate key search attack, like the one proposed in this paper, can find a key, which is capable of producing highly accurate outputs. This kind of attacks neither require scan access nor suffer from scalability issues. Error-tolerant applications are more vulnerable to this attack. Our future direction of research would like to focus on finding out a countermeasure against this type of approximate attacks.

References

[1] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[2] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *DATE*, 2008, pp. 1069–1074.

[3] R. Karmakar, H. Kumar, and S. Chattopadhyay, "On finding suitable key-gate locations in logic encryption," in *IEEE ISCAS*, 2018, pp. 1–5.

[4] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *HOST*, 2015, pp. 137–143.

[5] M. El Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *ICCAD*. IEEE, 2017, pp. 33–40.

[6] U. Guin, Z. Zhou, and A. Singh, "Robust design-for-security architecture for enabling trust in ic manufacturing and test," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018.

[7] R. Karmakar, S. Chattopadhyay, and R. Kapur, "Encrypt flip-flop: A novel logic encryption technique for sequential circuits," *arXiv preprint arXiv:1801.04961*, 2018.

[8] ——, "A scan obfuscation guided design-for-security approach for sequential circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

[9] 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)*. IEEE, 2019, pp. 534–539.

[10] K. Roy and A. Raghunathan, "Approximate computing: an energy-efficient computing technique for error resilient applications," in *2015 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2015, pp. 473–475.

[11] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4. Citeseer, 1995, pp. 1942–1948.

[12] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, Mar 2000. [Online]. Available: https://doi.org/10.1007/s100090050046

[13] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," *IACR Cryptology ePrint Archive*, vol. 2016(590), 2016.

[14] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Conference on CCS*. ACM, 2017, pp. 1601–1618.