Security Analysis of Logic Obfuscation

Jeyavijayan Rajendran†, Youngok Pino‡, Ozgur Sinanoglu§, and Ramesh Karri†
†Polytechnic Institute of New York University ‡Air Force Research Labs §New York University-Abu Dhabi
E-mail: jrajen01@students.poly.edu, Youngok.Pino@rl.af.mil, os22@nyu.edu, rkarri@poly.edu

ABSTRACT

Due to globalization of Integrated Circuit (IC) design flow, rogue elements in the supply chain can pirate ICs, overbuild ICs, and insert hardware trojans. EPIC [1] obfuscates the design by randomly inserting additional gates; only a correct key makes the design to produce correct outputs. We demonstrate that an attacker can decipher the obfuscated netlist, in a time linear to the number of keys, by sensitizing the key values to the output. We then develop techniques to fix this vulnerability and make obfuscation truly exponential in the number of inserted keys.

Categories and Subject Descriptors
K.6.5 **Management of Computing and Information Systems** Security and Protection-Physical Security
General Terms
Security
Keywords
IP protection, Logic obfuscation

1. INTRODUCTION

1.1 Motivation – Preventing IP Piracy

Globalization of Integrated Circuit (IC) design is making IC/Intellectual Property (IP) designers and users re-evaluate their trust in hardware [2]. As the IC design flow is distributed worldwide, hardware is prone to new kinds of attacks such as reverse engineering and IP piracy [1]. An attacker, anywhere in this design flow, can reverse engineer the functionality of an IC/IP. One can then steal and claim ownership of the IP. An untrusted IC foundry may overbuild ICs and sell them illegally. Finally, rogue elements in the foundry may insert malicious circuits (hardware trojans) into the design without the designer's knowledge [3]. Because of these attacks, the semiconductor industry loses $4 billion annually [4].

If a designer can hide the functionality of an IC while it passes through the different, potentially untrustworthy phases of the design flow, these attacks can be thwarted [1].

1.2 Logic obfuscation

Logic obfuscation hides the functionality and the implementation of a design by inserting additional gates into the original design. In order for the design to exhibit its correct functionality (i.e., produces correct outputs), a valid key has to be supplied to the obfuscated design. The gates inserted for obfuscation are the *key-gates*. Upon applying a wrong key, the obfuscated design will exhibit a wrong functionality (i.e., produce wrong outputs).

Consider the circuit shown in Figure 1 which is obfuscated using key-gates *K1* and *K2*. The inputs *I1* – *I6* are the functional inputs and *K1* and *K2* are the key inputs connected to the key-gates. On applying the correct key values (K1=0 and K2=1) the design will produce a correct output; otherwise, it will produce a wrong output.

EPIC [1] incorporates logic obfuscation into the IC design flow, as shown in Figure 2. In the untrusted design phases, the IC is obfuscated and its functionality is not revealed. Post-fabrication, the IP vendor activates the obfuscated design by applying the valid key. The keys are stored in a tamper-evident memory inside the design to prevent access to an attacker, rendering thes key inputs unaccessible by an attacker.

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

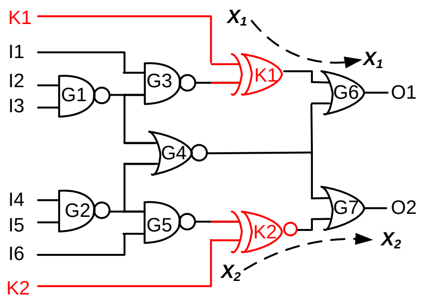

Figure 1: A circuit obfuscated using two key-gates K1 and K2 based on the technique proposed in [1]. By applying the input pattern 100000, an attacker can sensitize key bits K1 and K2 to the outputs O1 and O2, and observe their values.

1.3 Attacks against logic obfuscation

The purpose of logic obfuscation is defeated if an attacker can determine the secret keys used for obfuscation. By determining the keys, one can decipher the functional netlist, and make pirated copies and sell them illegally.

We propose an attack where the attacker applies specific input patterns, observes the outputs for these pattern, and deciphers the secret key. To perform this attack, one needs the obfuscated netlist and a functional IC. An attacker can obtain the obfuscated netlist from (1) the IC design, or by reverse engineering the (2) layout, (3) mask, or (4) a manufactured IC as shown in Figure 2. The functional IC, (5) in Figure 2, is bought in the open market.

The value of an unknown key can be determined if it can be sensitized[1] to an output without being masked/corrupted by the other key-bits and inputs. By observing the output, the sensitized key bit can be determined, given that other key-bits (similar to unknown X-sources[2]) do not interfere with the sensitized path.

Once an attacker determines an input pattern that propagates the key-bit value to an output without any interference, it is applied to the functional IC i.e., the IC with the correct keys. Now, this pattern will propagate the correct key value to an output. An attacker can observe this output and resolve the value of that key-bit.

Motivational example 1 (attack): Consider the key input K1 in Figure 1. It will be sensitized to output O1 if the value at the other input of gate G6 is 0 (non-controlling value for an OR gate). This can be achieved by setting I1=1, I2=0 and I3=0. As the attacker has access to the functional IC, one can apply this pattern and determine the value of K1 on O1. For example, if the value of O1 is 0 for that input pattern, then K1 = 0, otherwise K1=1.

This problem is analogous to the fault sensitization problem in the presence of unknown-X values that can possible block/mask the fault propagation [5]. The key-bits K1 and K2 are equivalent to X-sources X_1 and X_2 in Figure 1.

Similarities and differences between fault detection and key-propagation: Both objectives require an input pattern that sensitizes the fault effect/key bit by

- blocking the effect of some or all of the X-sources/other key bits, and preventing their interference.

[1]Sensitization of an internal line l to an output O refers to the condition (values applied from the primary inputs to justify the side input of gates on the path from l to O to the non-controllable values of the gates) which bijectively maps l to O and thus renders any change on l observable on O.
[2]X-sources: Uninitialized memory units, bus contentions or multi-cycle paths are the source of unknown response bits, i.e., unknown-Xs in testing. They are non-controllable.

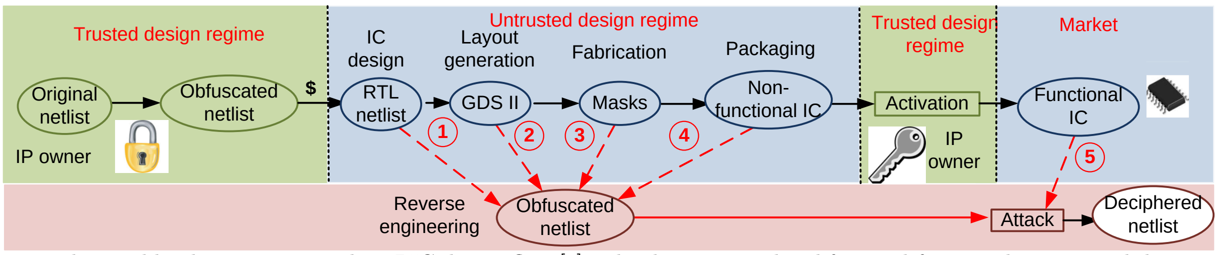

Figure 2: The top blue box represents the EPIC design flow [1]. The design is in the obfuscated form in the untrusted design regime. In the untrusted regime, an attacker can obtain the obfuscated netlist from (1) the IC design, or by reverse engineering the (2) layout, (3) mask, or (4) a fabricated IC, and (5) the functional IC from the market. Using this attack, the attacker can get a deciphered netlist and make pirated copies.

- justifying the side input of all the gates on the sensitization path to non-controlling values of the gates

The two problems differ slightly:

- fault detection also involves fault activation by justifying the fault site to the fault value; key propagation requires only sensitization
- fault detection aims at blocking/avoiding unknown X's; key propagation aims identifying unknowns one at a time resulting in an iterative and dynamic process

While an Automatic Test Pattern Generation tool [5] capable of handling X's during test generation can be readily used by the attacker to identify the patterns that decipher key bits, we take a closer look at various interference scenarios for the key bits (unknown sources) with the ultimate goal of building a strong defense, i.e., a smart logic obfuscation technique.

1.4 Smart logic obfuscation

To prevent such attacks, key-sensitization has to be hampered by inserting key-gates in such a way that propagation of a key value will be possible only if certain conditions are forced on other key inputs. As these key inputs are not accessible by the attacker, one cannot force the values necessary to propagate the effect of a key. Thus, brute force has to be employed.

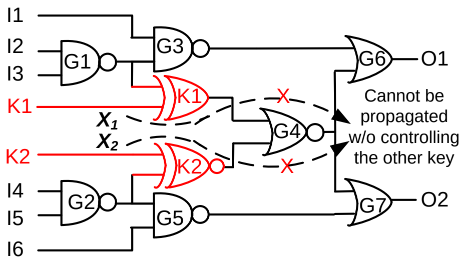

Figure 3: The attacker cannot propagate the effect of key bits K1 and K2 individually to the outputs. Hence, the attacker has to brute force to determine the values of K1 and K2.

Motivational example 2 (defense): Consider the circuit shown in Figure 3 which is the same functional circuit shown in Figure 1 but the two key-gates *K1* and *K2* are at different locations. Here, if the attacker has to propagate the effect of either of the keys, then one has to force a '0' (non-controlling value of NOR gates) on the other input of G4. In order to force this value, one has to control the key inputs, which are inaccessible. Thus one cannot propagate the effect of a key to an output, failing to determine the values of the key.

Depending upon the location of key-gates, different techniques have to employed to propagate the effect of a key. In Section 2, we describe the different types of key-gates based on their locations and also the strategies that an attacker may follow to decipher the key bits. In section 3, we introduce a graph notation to capture the interference between key bits, enabling algorithmic development for smarter logic obfuscation. Section 4 compares the obfuscation strength and performance results between the random insertion and the proposed logic obfuscation technique.

1.5 Contributions

The contributions of this paper are

- an attack on logic obfuscation based on IC testing concepts
- strategies used by an attacker to decipher keys based on their interference
- a logic obfuscation algorithm based on key-gates interference graph

2. ATTACK STRATEGIES

A logic obfuscation technique can insert the key-gates anywhere in the circuit. Depending upon their location, the attacker develops different strategies to determine the key bits. In this section, we will classify key-gates based on their type of interference with other key-gates and the corresponding strategy used by the attacker.

2.1 Runs of key-gates

A set of key-gates connected in a back-to-back fashion forms a run of key-gates.

Example 3: In Figure 4(a), the key-gates K1 and K2 form a run as they are connected back-to-back.

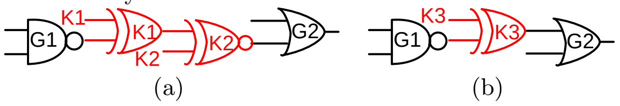

Figure 4: (a) A run of two key-gates K1 and K2. (b) K3 replaces K1 and K2.

Runs of key-gates reduce the effort of an attacker as they increase the valid key space. If N key-gates form a run, then the valid key space increases from the ideal, 1 valid key, to 2^{N-1} valid keys. In the above example, 01 and 10 are valid keys, one of which suffices for the attacker.

Attack strategy: An attacker can replace a run of key-gates by a single key-gate, thereby reducing the number of key bits. Once the value of that key-gate is determined, one can find the entire valid key space. For example, in Figure 4(b), the attacker replaces K1 and K2 with a key-gate K3. After the value of K3 is resolved as 1, one determines that the valid key space is 01 and 10.

2.2 Isolated key-gates

If there is no path from a key-gate to all the other key-gates and vice-versa, then such a gate is called an isolated key-gate.

Example 4: Consider the gate K1 in Figure 1. As there is no path between K1 and K2, K1 and K2 are isolated gates.

Attack strategy: An attacker prefers isolated key-gates as there is no interference with other key-gates. An attacker identifies a pattern that uniquely propagates the effect of an isolated key-gate's key to an output. One then applies the pattern to the functional IC and determines its value.

As mentioned before, in Figure 1, the pattern 100XXX propagates the value of K1 to output O1. An attacker, upon observing this output, can identify that the value of K1 is 0.

2.3 Dominating key-gates

If there are two key-gates K1 and K2 such that K2 lies on every path between K1 and the outputs, then K2 is called a dominating key-gate.

Example 5: The gate K2 in Figure 5 is a dominating key-gate.

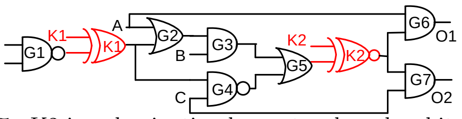

Figure 5: K2 is a dominating key-gate whose key bit value can be determined only after muting the effect of K1. Patterns that make either C = 0 and A=1 or C=0 and B =1 will mute the effect of K1. However, only if A = 1, the effect of K2 can reach O1.

Attack strategy: An attacker can determine the value of K2's key bit only if the effect of K1's key bit is prevented (muted) from reaching key-gate K2 while simultaneously sensitizing K2's key bit to an output. An input pattern that can perform muting as well as sensitization is called the golden pattern. On applying this golden pattern, the attacker can determine the value of K2. If muting of K1 and propagation of K2 cannot be performed simultaneously, then the attacker cannot determine value of K2. In such cases, the golden pattern does not exist, forcing an attacker to employ brute force.

The effect of a key can be muted before it reaches the other key, by using patterns that force controlling values in any of the gates on the path between K1 and K2. If there are multiple paths from key-gates K1 and K2, then the effect of key-input K1 has to be muted on every path.

Example 6: Consider the circuit shown in Figure 5. K2 can be determined only if the effect of K1 is muted. If there is a pattern that justifies the output of G5 to 1, then the effect of K1 will be muted. Patterns that make either C = 0, or A=1 and B =1 will assure this condition, thereby muting the effect of K1. However, the attacker should select the pattern that propagates the effect of K2 to an output. If C =0, G7 blocks the propagation of K2 as its output will always be 0. The condition A = 1, allows K2 to propagate through G6. Hence, an attacker will select the pattern that makes A=1 and C =0, so that one can mute the effect of K1 as well as propagate the effect of K2 to an output.

2.4 Convergent key-gates

Even if there are no paths between two key-gates, the sensitization paths might interfere. Such scenarios happen if these two or more key-gates converge. Depending upon the type of convergence, key-gates can be classified into 1) concurrently mutable, 2) sequentially mutable, and 3) non-mutable key-gates.

2.4.1 *Concurrently mutable convergent key-gates*

If two key-gates K1 and K2 converge at some other gate, such that K1's key bit can be determined by muting K2, and K2's key bit can be determined by muting K1, then K1 and K2 are called concurrently mutable key-gates.

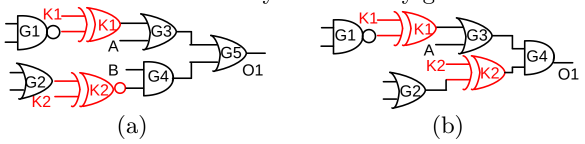

Figure 6: (a) Concurrently mutable key-gates: K1 and K2 converge at G5 and can be muted. (b) Sequentially mutable key-gates: K1 and K2 converge at G4, but only K1 can be muted.

Example 7: Consider the circuit shown in Figure 6(a). The key-gates K1 and K2 converge at the gate G5. The value of K1 can be determined by applying a pattern that mutes K2 (B=0). Similarly, the value of K2 can be determined by applying a pattern that mutes K1 (A=1).

Attack strategy: The attacker determines the golden pattern that mutes one key and simultaneously sensitizes the other key to an output, or vice-versa. If a golden pattern does not exist, then the attacker has to perform brute force only on that set of concurrently mutable key-gates.

2.4.2 *Sequentially mutable convergent key-gates*

If two gates K1 and K2 converge at some other gate, such that K2's key bit can be determined by muting K1's key while K2's key cannot be muted to determine K1's key, then K1 and K2 are called sequentially mutable convergent key-gates, as they can be deciphered only in a particular order.

Example 8: Consider the circuit shown in Figure 6(b). The value of K2 can be determined by applying a pattern that mutes K1 (A=1), while K2 cannot be muted as it directly feeds the gate where K1 and K2 converge.

Attack strategy: An attacker will first determine K2's value by muting K1 using the golden pattern. One then updates the netlist by replacing K1 with a buffer or an inverter based on the value of K2. Then K1 is targeted. If the golden pattern does not exist, then the attacker has to perform brute force only on that set of sequentially mutable key-gates.

2.4.3 *Non-mutable convergent key-gates*

If two key-gates K1 and K2 converge at some other gate, such that neither of the key bits can be muted, then K1 and K2 are called non-mutable convergent key-gates.

Example 9: Consider the circuit shown in Figure 3. The key-gates K1 and K2 are connected to the same gate G4.

Attack strategy: To propagate either of the key bits the other one has to be muted. However, as an attacker cannot access key inputs, one cannot force those values. Hence, one is forced to perform brute force attacks.

```
Input  : Obfuscated netlist, Functional IC, Key Inputs
Output: Original netlist
Determine Runs of Keys;
Replace them with XOR gates;
Update Netlist;
for the remaining keys do
    For each Isolated Key do
        Compute and apply propagation pattern;
        Determine KeyBits and update Netlist;
    end
    For each Consecutive||Concurrent||Sequential key do
        if there exists a golden pattern then
            Apply the golden pattern;
            Determine KeyBits, Update Netlist, Break;
        else
            ApplyBruteForce(), Break;
        end
    end
    For each Non-mutable Key do
        ApplyBruteForce(), Break;
    end
end
```

```
ApplyBruteForce();
For each possible key combination do
    Generate random input patterns;
    Simulate the patterns and obtain the outputs $OP_{sim}$;
    Apply the patterns on IC and obtain the outputs $OP_{exe}$;
    if $OP_{sim}$==$OP_{exe}$ then
        Valid Key = current key combination;
        Update netlist;
    end
end
```

Algorithm 1: Attack on logic obfuscation.

2.5 An attacker's action plan

By considering all the different types of interference between key-gates, an attacker uses Algorithm 1 to determine the secret key. The attacker first removes the runs of key-gates and targets the isolated key-gates. Each isolated gate can be removed by one test patterns. After that, one targets consecutively mutable, concurrently mutable, and sequentially mutable key-gates. Only if one is able to generate a golden pattern that simultaneously mutes effects of the other keys and sensitizes the effect of the target key, the value of the target key can be determined. Finally, the non-mutable keys are identified via brute force. As the key bits are identified gradually in every iteration, the corresponding key-gates can be replaced by a buffer or an inverter, possibly changing the type of other key-gates. Thus, in every iteration, the key-gate types need to be re-computed.

3. STRONG LOGIC OBFUSCATION

Strong logic obfuscation hinges on inserting key-gates with complex interferences among them. Next, we relate types of key-gates to the kind of interference they introduce using a graph-based notation.

3.1 Interference graph

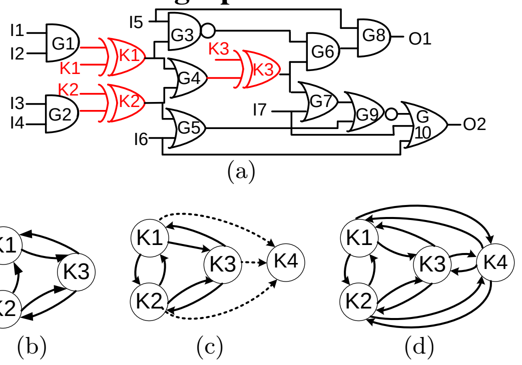

Figure 7: (a) An example circuit with three key-gates. (b) Interference graph of the key-gates. Non-mutable keys are connected by solid edges. (c) If the new key-gate is inserted at the output G10, it creates mutable edges (dotted lines) with the other key-gates (d) If the new key-gate is inserted at the output G5, it creates non-mutable edges (solid lines) with the other key-gates.

To insert key-gates, we form an interference graph of key-gates. In this graph, each node represents a key-gate and an edge connects two nodes, if two gates interfere. Isolated key-gates are represented with isolated nodes. A run of key-gates is denoted by a single node. Non-mutable key-gates are represented are connected with non-mutable edges, concurrently mutable key-gates are connected with mutable edges. Sequentially mutable key gates are connected by two edges; a non-mutable edge arises from the key-gate that is non-mutable and a mutable edges arises from the key-gate that is mutable.

Example 10: Consider the circuit with three key-gates shown in 7(a). They interfere with each other as follows

- K1 and K2 are non-mutable and so they are connected by non-mutable edges as shown in Figure 7(b).
- The key-gates K1 and K3 converge at the gate G6, hence they are converging key-gates. Specifically, they are sequentially convergent; K3's effect cannot be muted while K1's effect can be muted by applying I5=0. However if I5 is 0, then both key bits are blocked at G8. Hence, K1 and K3 are non-mutable and so they are connected by non-mutable edges as shown in Figure 7(b).
- K2 and K3 converge at the gate G9, through G5 and G7, respectively. However, neither of the key bits can be muted and sensitized individually. For instance, making I6=1, mutes K2 but also blocks the sensitization of K3 at G10. Making I7=1, mutes K3 but also blocks the sensitization of K2 at G10. Hence, K2 and K3 are non-mutable as shown in Figure 7(b).

For a stronger logic obfuscation, the number of non-mutable edges in the interference graph should be maximized, as they force an attacker to perform brute force. On the other hand, if there are more mutable edges, then the attacker can mute the effect of keys and can easily determine their values. Hence, a defender prefers non-mutable edges to mutable edges.

Example 11: Consider the circuit shown in Fig 7(a). If a new key-gate, K4, is inserted at the output of G10, then it creates mutable edges with all the other key-gates. By setting I6=1 or I7=1, the attacker can mute the effects of K1, K2, and K3, and can decipher easily. Hence, G10 is connected with mutable edges with the other key-gates as shown in Figure 7(c).

If the new key-gate, K4, is inserted at the output of G5, then it creates non-mutable edges with the other key-gates as shown in Figure 7(d). Thus, it is better to insert the new key-gates at the output of G5.

```
Input   : Original netlist, KeySize
Output: Obfuscated netlist
KeyGateLocations = {};
Randomly insert 10% key-gates;
Add that location to KeyGateLocations;
Construct KeyGraph;
for $i \leftarrow 2$ to KeySize do
    For each $Gate_j$ in Netlist do
        if $Gate_j \notin$ KeyGateLocations then
            Cum. Weight = $\sum$ weight of edges in KeyGraph;
            For each $Key\text{-}gate_k$ in KeyGateLocations do
                Cum. $\text{Weight}_j$ += FindMetric($\text{Gate}_j$,
                $\text{Key-gate}_k$);
            end
        end
    end
    Select the Gate with the highest Hardness Metric;
    Add the selected gate to KeyGateLocations;
    Insert a key-gate at the output of the selected gate;
    Update KeyGraph;
end
```

```
FindMetric(K1, K2);
if K1 and K2 are isolated then Return 0;
if K1 and K2 are consecutive||concurrent||sequential then
    if a golden pattern exists then
    Return weight of mutable edge;
    else Return weight of non-mutable edge;
end
if K1 and K2 are non-mutable then
Return weight of non-mutable edge;
```

Algorithm 2: Insertion of key-gates

3.2 Insertion of key-gates

A defender can use the interference graph to insert key-gates. Algorithm 2 is used to insert key-gates. At every iteration, a key-gate is inserted at a location such that the number of non-mutable edges in the graph is maximized.

Initially, 10% of the total key-gates are inserted at random locations in the circuit. Such random distribution will insert key-gates in different parts of the circuit thereby affecting multiple outputs. Here, we considered 10% for initial distribution (one also can chose a different amount of initial distribution and the impact of this amount on obfuscation is beyond the scope of this paper). Then the graph of key-gates is constructed. Then, the remaining key-gates are introduced iteratively. In every iteration, for each gate in the netlist, we determine the type of edge with the previously inserted key-gate. Depending upon the type of edge, we assign weights; non-mutable edges are given a higher weight than the mutable edges. We then calculate the sum of weights of edges in the graph for that gate. The gate that maximizes the sum of weight of edges in the graph is selected, and a key-gate is inserted at its output. The graph is then updated by including the new key-gate. This procedure is repeated for inserting all the key-gates.

In every iteration, the defender has to check for the presence of golden patterns which might increase the computational complexity of the algorithm. Hence, a defender can assume that there always exists a golden pattern and skip the search for the golden pattern. This is a pessimistic scenario for a defender because some golden patterns might not exist.

4. RESULTS

4.1 Experimental Setup

The proposed technique is analyzed using ISCAS-85 combinational benchmarks. We used the Atalanta testing tool [6] to determine the input patterns for muting and propagation the effects of keys. To obfuscate a circuit with a reasonable performance overhead, we selected the key size as 5% of number of gates in that circuit. While obfuscating a circuit, we assumed that there always exists a golden pattern. While attacking the circuit, we used the techniques proposed in Section 2 where we search for the presence of a golden pattern. For every brute force attempt, we applied 1000 random patterns to determine the value of a key. The area, power, and delay overheads were obtained using the Cadence RTL compiler.

We compared the effectiveness of four types of insertions: random-insertion [1], random insertion with no runs of gates, unweighted insertion where both mutable and non-mutable edges are given the same weight of 1, and weighted insertion where non-mutable edges are given a higher weight (weight = 2) than the mutable edges (weight = 1).

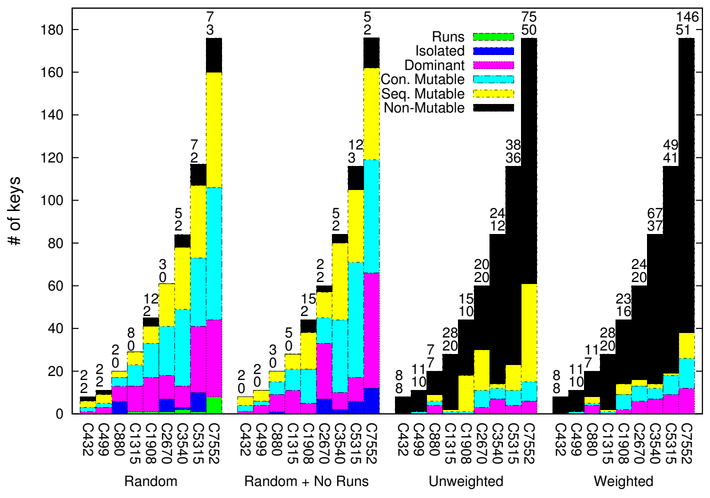

Figure 8: Types of key-gates inserted by different logic obfuscation techniques. Effective key size from an attacker's perspective (top) and from a defender's perspective (bottom) are shown as numbers on top the bars.

4.2 Types of keys and effective key-size

Figure 8 shows the number of types of keys in different benchmarks for different types of insertions. In the random insertion method, most of the keys are concurrently mutable. Some number of keys are inserted in runs benefiting the attacker. Only 30% of keys are non-mutable and sequentially mutable which require brute force approach. In the 'Random + No Runs' method, keys are not inserted in runs thereby increasing the effort of the attacker.

In the unweighted and weighted insertions, around 90% of keys are of non-mutable and sequentially mutable types. Most of the keys in weighted insertion is either non-mutable or sequentially mutable because they are given a higher weight. There are no isolated keys in either of the insertion techniques, as they are not given any weights.

Effective key size: Due to random insertion of the first 10% of key-gates, multiple disconnected graphs might exist within a key-interference graph. The keys in a graph can be either isolated, dominant, or convergent. Since a defender pessimistically assumes that the golden patterns always exist, the effective key size from his perspective is the maximum number of non-mutable keys in a connected key-interference graph. If there are N non-mutable key gates (effective key-size), the number of brute force attempts is $2^{N\text{-}1}$. However, when an attacker tries to attack, not all the golden patterns will exist. For those keys, he has to try for all possible combinations. Hence, from an attacker's perspective, the effective key size is the largest key size on which brute force is attempted. If the number of brute force attempts is 2^{M}, then the effective key size for an attacker is M.

In Figure 8, the effective key-sizes for a defender and an attacker are shown as numbers on top of the bars. For both the attacker and defender, the effective key sizes of random insertions are less than that of the unweighted and weighted insertions. Therefore, the number of brute force attempts required to decipher the keys inserted using random insertions is exponentially smaller than that of the unweighted and weighted insertions. The attacker's effective key size is always greater than that of the defender's because of the absence of golden patterns which forces the attacker to perform brute force. For example, consider the benchmark C7552, the attacker needs 2^{146} brute force attempts and hence the effective key size is 146. On the other hand, for a defender, the largest number of non-mutable key-gates in a connected graph is 51 and hence the effective key size is 51.

4.3 Number of test patterns

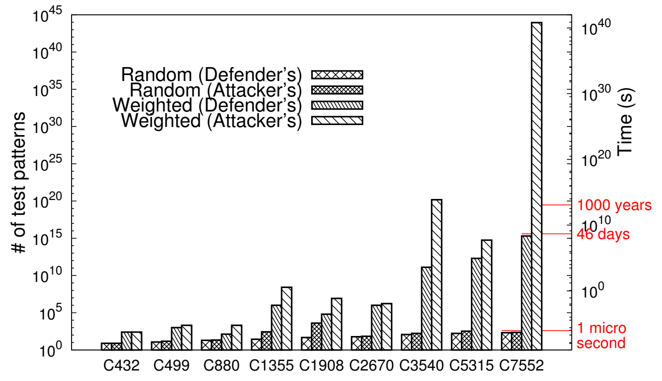

Figure 9: Number of test patterns required by the attacker to determine the keys inserted using random and weighted insertion from an attacker and a defender's perspectives. The time scales are drawn assuming that one billion input patterns can be applied per second.

Figure 9 shows the number of test patterns required to decipher the key from a defender and an attacker's perspectives for the random and weighted insertion method. The time scales are calculated assuming that an attacker can apply a billion patterns per second. On one hand, from a defender's perspective, the number of test patterns are calculated assuming that golden patterns exist. On other hand, from an attacker's perspective, the number of test patterns are more realistic as they are determined using the attack methodology proposed in Section 2. It can be seen that the defender's perspective on timescale is several orders of magnitude smaller than the realistic scenario. For example, in C7552 circuit, a defender thinks that it will take 46 days to decipher the netlist while the attacker will take more than a thousand years. However, from both attacker's and defender's perspectives, a few thousand test patterns are sufficient to figure out the keys when they are inserted randomly. On the other hand, when the weighted key insertion method is used, the number of test patterns required to recover the keys increases by several orders of magnitude. For example, in case of C7552 to about 10^{18} which will take several years to figure out the key bits.

4.4 Effect of the weight of a non-mutable edge

Table 1: No. of non-mutable keys out of the total 176 keys in the benchmark C7552 for different weights of non-mutable edges.

Weight of non-mutable edge	1	2	10	100	1000
# of non-mutable key-gates	115	138	149	156	163

By increasing the weight of the non-mutable edges, the algorithm will create a design that has a large number of non-mutable key-gates. Table 1 shows the number of non-mutable key-gates for different weights of non-mutable edges in one of the ISCAS-85 benchmark circuit, C7552. This circuit was obfuscated with 176 key-gates. While increasing the weight of the non-mutable edges increases the number of non-mutable key-gates in the design, the rate of increase is not the same rate. Increasing the weight from 1 to 2 increases the number of non-mutable key-gates from 115 to 138. But increasing the weight from 2 to 10, increases the number of non-mutable key-gates from 138 to 149.

4.5 Area overhead

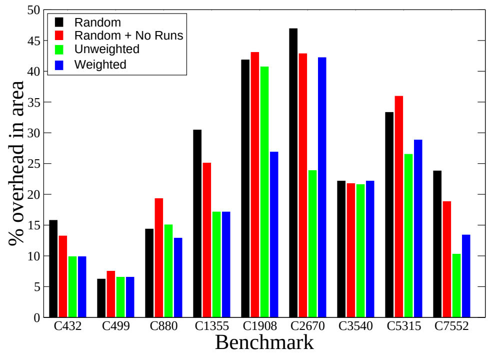

Figure 10: Area overhead for different insertion algorithms.

Figure 10 shows the area overhead for different key-gate obfuscation algorithms. Even though the number of key-gates inserted is 5% of the number of gates in the original design, the area overhead is high as the key-gates are XOR/XNOR gates that consists of a large number of transistors. Unweighted and weighted insertion techniques entail less overhead than random insertion techniques.

4.6 Power-delay product

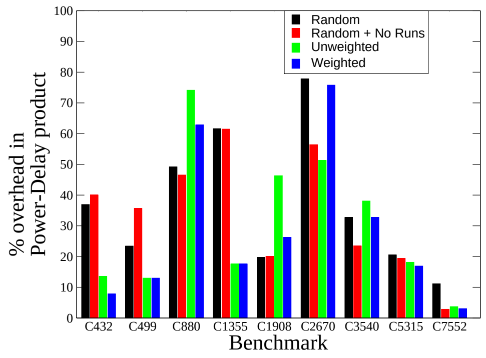

Figure 11: Power delay product overhead for different insertion algorithms.

Figure 11 shows the power-delay product overhead for different insertions. Random insertion yields an average overhead of 25% while weighted and unweighted insertion yields an average overhead of 21%. To minimize this overhead, one can pursue a power and delay constrained obfuscation.

4.7 Logic obfuscation with PUFs

Physical Unclonable Functions (PUFs) are circuits that leverage process variations in IC manufacturing, to produce secret keys. In [1], PUFs are used to give unique keys for each IC even though they are all obfuscated with the same key. The design is first obfuscated with a key and a PUF circuit is attached to it. Upon applying the user key (challenge) to the PUF, the PUF's response will be the key used for obfuscation. In the proposed attack, the attacker is trying to figure out this response i.e., the key used for obfuscation. On getting this response, the attacker can remove the PUF circuit from the netlist and apply the correct keys directly to the original design. To break the influence of PUFs or any cryptographic algorithms, an attacker can determine the wires that carry these signals and disconnect them.

5. RELATED WORK

Logic obfuscation techniques can be broadly classified into two types—sequential and combinational. In sequential logic obfuscation, additional logic (black) states are introduced in the state transition graph [7,8]. The state transition graph is modified in such a way that the design reaches a valid state only on applying a correct sequence of key bits. If the key is withdrawn, the design, once again, ends up in a black state. However, the effectiveness of these methods in producing a wrong output has not been demonstrated. In combinational logic obfuscation, as mentioned before, XOR/XNOR gates are introduced to conceal the functionality of a design [1].

Obfuscation is also performed by inserting memory elements [9]. The circuit will function correctly only when these elements are programmed correctly. However, using memory elements will incur significant performance overhead.

6. CONCLUSION

Logic obfuscation is weak when the inserted key-gates are isolated or their effect can be muted. If mutable gates are employed, then the attacker is able to determine the key bits within a second. However, it can be strengthened by inserting key-gates such that their effects are not mutable. In such insertions when the key size is greater than 100, it will take several years for an attacker to determine the key bits. Our analysis reveal that even though a defender pessimistically assumes a smaller effective key size, the actual key size encountered by the attacker is much higher.

IC testing techniques allow designers and testers to peek into the design, by controlling only the inputs and observing the outputs. On one hand, an attacker can use such capability to subvert logic obfuscation. On the other hand, a defender can perform better logic obfuscation by making such process infeasible using the lessons learnt from testing.

7. ACKNOWLEDGEMENT

This material is based upon work funded by AFRL under contract No. FA8750-11-2-0274. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL.

8. REFERENCES

[1] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *Proc. of Design, Automation and Test in Europe*, pp. 1069–1074, 2008.
[2] "Defense Science Board (DSB) study on High Performance Microchip Supply," http://www.acq.osd.mil/dsb/reports/ADA435563.pdf, 2005.
[3] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *IEEE Computer*, vol. 43, no. 10, pp. 39–46, 2010.
[4] SEMI, "Innovation is at risk as semiconductor equipment and materials industry loses up to $4 billion annually due to IP infringement," www.semi.org/en/Press/P043775, 2008.
[5] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits," *Kluwer Academic Publishers, Boston*, 2000.
[6] H. Lee and D. Ha, "An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation," *Proc. of IEEE International Test Conference*, pp. 946–955, 1991.
[7] R. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design*, vol. 28, no. 10, pp. 1493–1502, 2009.
[8] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," *Proc. of USENIX security*, pp. 291–306, 2007.
[9] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

APPENDIX
(The intuition behind key interference based logic obfuscation)

We will analyze the key-interference graphs of a circuit obfuscated using random and weighted insertion methods both from a defender and an attacker's perspectives. As mentioned in Section 3, the defender always assumes that the golden pattern, that mutes and sensitizes the effect of key-gates simultaneously, does not exist, and constructs his key interference graph accordingly. However, an attacker tries to find the golden patterns to mute key-gates to decipher the value of keys, and constructs his key interference graph accordingly.

A. RANDOM INSERTION

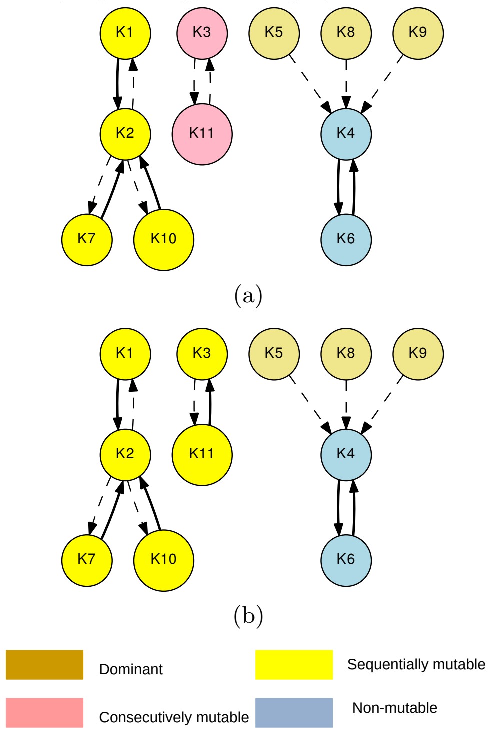

Figure 12: Key-interference graphs of C499, an ISCAS-85 benchmark circuit, obfuscated with 11 key-gates. Dotted lines represent mutable edges and solid lines represent non-mutable edges. (a) Key-interference graph from a defender's perspective with an assumption that the edge K11→K3 is mutable. (b) Key-interference graph from an attacker's perspective. The golden pattern to mute the edge K11→K3 does not exist.

Defender's perspective: Figure 12 shows the key interference graph of one of the ISCAS-85 benchmark circuit, C499, which is obfuscated by inserting 11 key-gates. Key-gates K5, K8, and K9 are classified as dominant key-gates[3]. Key-gates K3 and K11 are classified as consecutively mutable key-gates. Key-gates K1, K2, K7, and K10 are classified as sequentially mutable key-gates. Key-gates K4 and K6 are classified as non-mutable key-gates. From a defender's perspective, since there two non-mutable key-gates, the effective key size is two.

Attacker's perspective: Consider the scenario where an attacker tries to search for the golden pattern for the edge K11→K3 that simultaneously mutes K11 and sensitizes K3. He concludes that such a pattern does not exist. Thus, from an attacker's perspective, the edge from K11→K3 is non-mutable as shown in Figure 12(b). Hence, the key-gates K3 and K11 are classified as sequentially mutable key-gates. As the largest key size on which brute force is attempted is two, the effective key size is two. Notice that even though eleven key-gates are inserted, the effective key size is only two.

[3]Please refer Section 2 for the definitions of different types of key-gates.

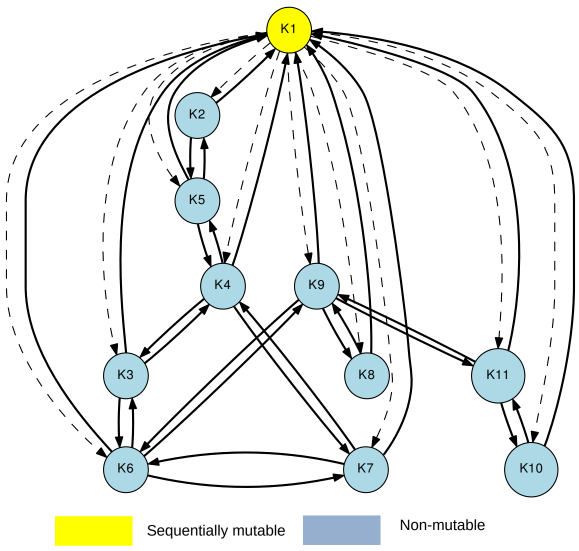

Figure 13: Key-interference graph of C499 from a defender's perspective with an assumption that the edges K1→K2, K1→K5, K1→K10, and K1→K11 are mutable. Dotted lines represent mutable edges and solid lines represent non-mutable edges. Effective key size is 10.

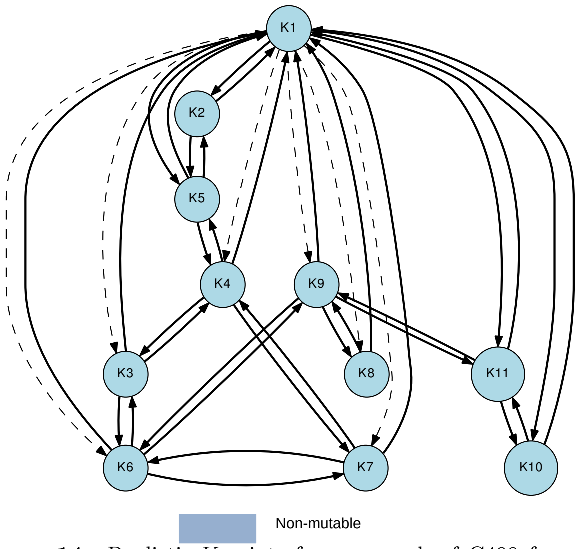

Figure 14: Realistic Key-interference graph of C499 from an attacker's perspective. The golden patterns to mute the edges K1→K2, K1→K5, K1→K10, and K1→K11 do not exist. Hence, K1 is non-mutable increasing the effective key size to 11.

B. WEIGHTED INSERTION

Defender's perspective: As shown in Figure 12(a), the edges K1→K2, K1→K5, K1→K10, and K1→K11 are mutable. Hence, the key-gate K1 is classified as a sequentially mutable key-gate and all the other gates are classified as non-mutable key-gates. From a defender's perspective, since there are ten non-mutable key-gates, the effective key size is ten.

Attacker's perspective: The attacker searches for the golden pattern that mutes the key-gate K1. As such a pattern does not exist, he classifies the edges K1→K2, K1→K5, K1→K10, and K1→K11 as non-mutable. Therefore, the key-gate K1 also becomes non-mutable. As the attacker has to try all combinations of the keys, K1 to K11, the effective key size is eleven. While the effective key size in random insertion is two, the proposed method has an effective key size of eleven.