# Improving Logic Obfuscation via Logic Cone Analysis

Yu-Wei Lee and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712
*ywlee@utexas.edu, touba@utexas.edu*

**Abstract - Logic obfuscation can protect designs from reverse engineering and IP piracy. In this paper, a new attack strategy based on applying brute force iteratively to each logic cone is described and shown to significantly reduce the number of brute force key combinations that need to be tried by an attacker. It is shown that inserting key gates based on MUXes is an effective approach to increase security against this type of attack. Experimental results are presented quantifying the threat posed by this type of attack along with the relative effectiveness of MUX key gates in countering it.**

## 1. Introduction

Logic obfuscation has been proposed as a means to hide the functionality of a design to protect it from reverse engineering, hardware Trojan, and IP piracy [Roy 08, 10], [Chakraborty 09a], [Baumgarten 10], [Rajendran 12, 14]. One approach is to use combinational obfuscation where "key gates" are inserted in a design which have primary inputs that are referred to as "key inputs". The design will only function correctly if the correct key values are used. This approach was proposed in [Roy 08, 10]. The IP vendor activates the obfuscated design by storing the correct key values in a tamper-evident memory or using a physically unclonable function (PUF) [Suh 07] to generate them in a way that an attacker cannot access the key values. Without the key values, the design is unusable even if the netlist is obtained by either stealing the design or reverse engineering it.

Logic obfuscation with XOR and MUXes is investigated in [Rajendran 14]. In order to evaluate the effectiveness of the key bits, hamming distance (HD) was used to evaluate the result after application of a wrong key. 50% of HD is desired to ensure the malicious users will not be able to obtain correct output easily without the correct key.

In addition to raising the effectiveness of key insertion, [Rajendran 12] also pointed out the importance of protecting the key bits against “attacks”. If the attacker has the netlist and purchases a functional IC in the open market, then the attacker can try to determine the correct key input values to unlock the design. This is done by simulating input patterns on the netlist and comparing them with the correct output values obtained by running the same input patterns on the functional IC. It is shown in [Rajendran 12] that if an input pattern can be found which sensitizes a key input to a primary output without any interference from other key inputs, then the input pattern will propagate the correct key value to the primary output when running the pattern on the functional IC. The attacker can use ATPG to try to find an input pattern that sensitizes a key input to a primary output treating all other key inputs as *X*'s. Each time such a pattern is found, the attacker runs it on the functional IC and looks at the output to resolve the key value. Whenever a key value is resolved, it is no longer an *X* when sensitizing the remaining key inputs thereby making the problem iteratively easier. After as many keys are resolved as possible using this approach, the remaining keys can be determined through brute force. The brute force procedure is to try each possible combination of values for the keys until the correct one is found. Checking if the candidate combination of key values is correct is done by simulating a set of random patterns on the netlist using the candidate key values and then comparing that with the output response obtained by simulating the same patterns on the functional IC to see if they match.

In [Rajendran 12], a heuristic procedure is proposed for inserting key gates so as to increase the amount of interference between the key gates and reduce the chance of finding patterns that sensitize a key input to an output without interference from other key inputs. The goal is to maximize the number of key values for which brute force must be employed thereby increasing the difficulty for the attacker. The heuristic procedure is based on constructing a graph in which each node is a key, and weighted edges are placed between the nodes to indicate the amount of interference that is created. Each key gate is iteratively inserted in the design so as to maximize the resulting sum of the weights on all edges in the graph at each step.

In this paper, a new form of attack is considered where the attacker can reduce the complexity of brute-force attacks through analysis of logic cones. The ideas proposed here build on the concepts and overall framework introduced in [Rajendran 12]. An attack strategy is proposed in which the attacker considers the design one logic cone at a time. It is shown that in many cases, the number of brute force key combinations that need to be tried by the attacker can be significantly reduced. A technique for improving the strength of the logic obfuscation with respect to the considered attack strategy is proposed. It is based on inserting key gates based of MUXes to increase the size and overlap of logic cones.

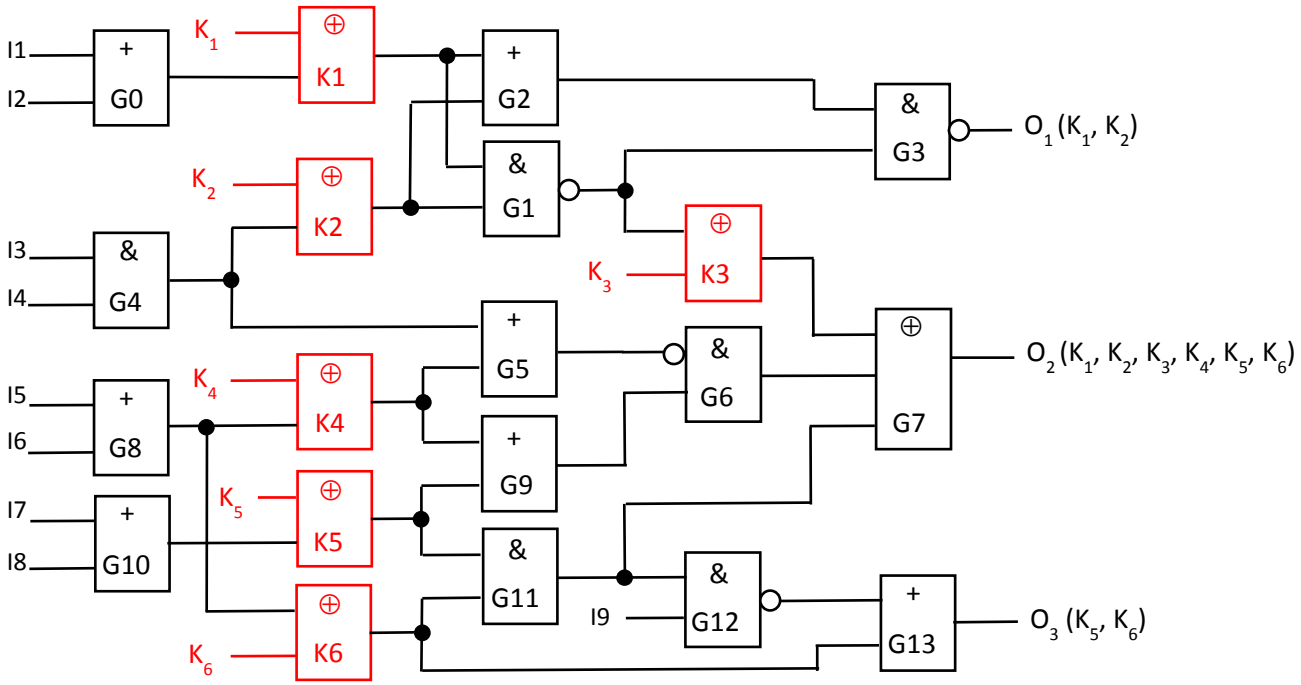


**Figure 1.** Example of Circuit with 6 Key Gates Inserted.

MUXes were used in [Charkraborty 09b] to introduce additional inputs in order to protect hardware at system level. In the proposed idea, MUXes are used to avoid the situation where the attacker can take advantage of less secure logic cones to easily resolve key values and thereby iteratively attack the overall logic obfuscation. The MUX key gate insertion is done in a non-deterministic manner so that even if an attacker knows the key insertion algorithm, the attacker cannot exploit this knowledge to decipher the logic obfuscation.

The paper is organized as follows: Section 2 describes the new attack strategy based on logic cones. Section 3 discusses how the effective key size is computed for cone-based attacks. Section 4 presents the proposed approach for countering a key-based attack by inserting key gates based on MUXes. Section 5 shows experiments results, and Section 6 is a conclusion.

## 2. Brute Force Attack Strategy Based on Logic Cones

The attack strategy proposed here can be applied after the key sensitization techniques proposed in [Rajendran 12] are applied to resolve as many keys as it can. For the remaining keys, the idea here is that brute force can be applied first to the logic cone containing the fewest number of key inputs, i.e., the least secure logic cone. All combinations of key inputs can be tried to identify for which combinations of values the simulated netlist output for that logic cone matches the functional IC output. In many cases, there will be only one combination that matches in which case all the key values for that cone are resolved. In some cases, there may be more than one combination of key values that matches, but it will likely be very few combinations which greatly reduces the search space for subsequent keys. Brute force is then applied for the next logic cone that has the fewest remaining unresolved key inputs. The process iterates through all the logic cones from easiest to hardest in terms of number of remaining unresolved key inputs in each cone. For each cone, additional keys are resolved or the number of possible solutions is greatly reduced which helps to simply brute force for the next logic cone.

This process is illustrated in Fig. 1. The circuit has 6 key inputs and if brute force is used on the circuit as a whole, $2^6$ different key combinations would have to be tried in the worst-case. However, output $O_1$ only depends on two key inputs ($K_1$ and $K_2$). So brute force could be applied on that first requiring only $2^2$ different key combinations in the worst-case to resolve $K_1$ and $K_2$. Next, output $O_3$ only depends on two key inputs ($K_5$ and $K_6$). Once those are resolved trying $2^4$ combinations worst-case, then there are only two keys remaining ($K_3$ and $K_4$) for which brute force needs to be performed for $O_2$. So rather than $2^6$ operations, only $2^4+2^4+2^4$ operations would be required in the worst-case. While the difference in the number of operations is not large in this small example, it quickly scales up exponentially as the number of keys are increased.

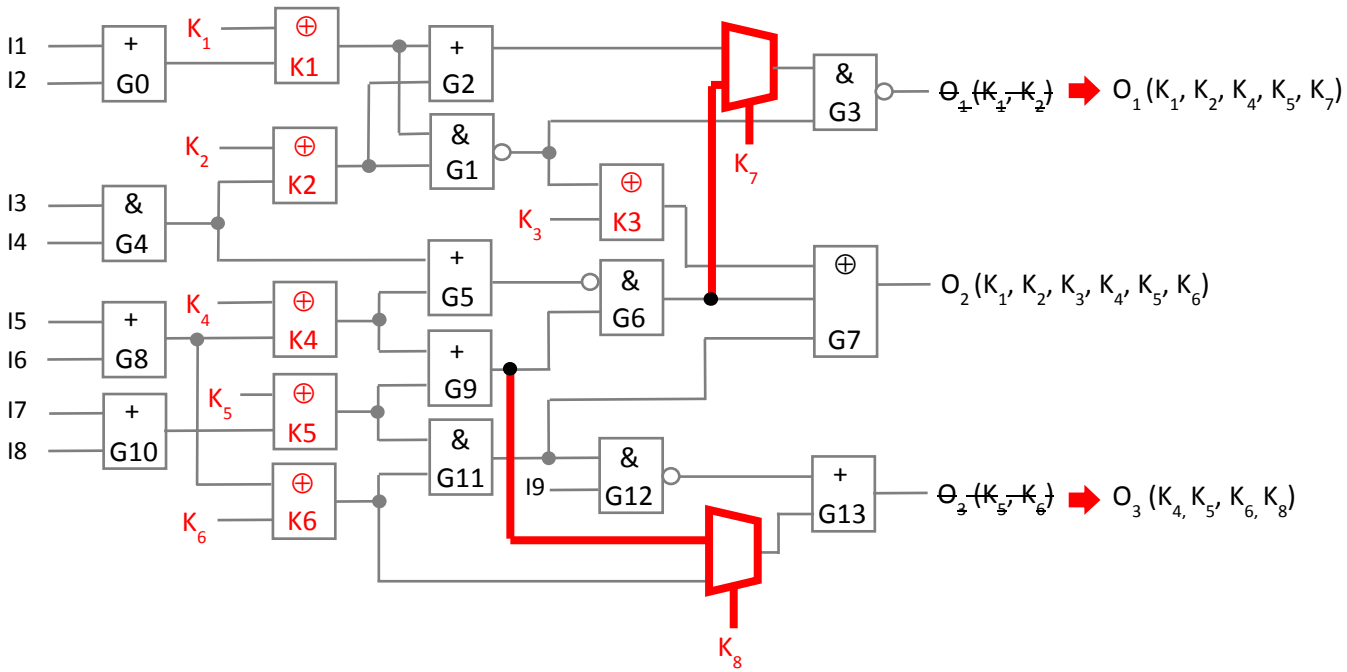


Figure 2. Example of Circuit with 2 MUX Key Gates Inserted

## 3. Effective Key Size for Cone-Based Attack

As explained in detail in [Rajendran 12], there are a number of ways that the effects of a key can be "muted". For example, for some input patterns the output of a key gate can be logically masked thereby reducing the number of keys that the circuit output depends on which helps to reduce the difficulty for the attacker. Different relationships between key gates such as dominating and convergent are studied in [Rajendran 12], and a procedure for constructing an interference graph is described. From the interference graph, the *effective key size,* which is the number of keys that have to be solved through brute-force, can be computed. The number of brute force attempts required to decipher the keys is exponential in the effective key size. In this paper, the procedure described in [Rajendran 12] for computing the effective key size from the defender's perspective (which is a conservative measure) is used as the basic metric for measuring security. The procedure for building the interference graph and computing this value is beyond the scope of this paper, however, a detailed explanation of it can be found in [Rajendran 12].

The new aspect of this paper is that the effective key size is computed w.r.t. each logic cone. Here we compute an effective key size against a cone-based attack as follows. It is assumed that the logic cone with the smallest effective key size is attacked first and its keys are resolved and removed from consideration when computing the effective key size for the remaining logic cones. From the remaining logic cones, it is assumed that the one with the smallest effective key size is attacked next and its keys are resolved. This process is then repeated iteratively until all keys are resolved. The logic cone with the largest effective key size when attacked defines the overall effective key size against a cone-based attack.

## 4. Insertion of Key Gates to Counter Attack

In order to counter the logic cone based attack strategy and ensure sufficient difficulty for the attacker to obtain the key values, it is necessary to consider the structure of the logic cones in the circuit when inserting key gates. One way to improve the logic obfuscation would be to insert more key gates in a way that the number of key gates in each logic cone is sufficiently large. The number of logic cones is equal to the number of outputs which potentially could be quite high for large designs. Thus it may require inserting a lot of key gates to get sufficiently high security for each and every logic cone.

The proposed approach involves inserting key gates based on MUXes. In this case, a MUX is inserted on some line in the circuit and the select line for the MUX is controlled by a key input. One data input for the MUX comes from the original line in the circuit, while the other data input comes from another line in the circuit which can be picked arbitrarily provided it does not create a feedback loop. The advantage of using MUX key gates is that they can increase the logic cone size and create more overlap between logic cones. This tends to increase the number of keys present in each logic cone thereby increasing the difficulty for an attacker using a cone-based attack. In fact, as will be seen in the experimental results, a relatively small number of MUX key gates can create a high degree of overlap among the output cones greatly reducing the effectiveness of a cone-based attack.

**Table 1.** Comparison of Different MUX Input Selection Strategies in Terms of Effective Key Size

| Circuit Name | Keys Insert | Random | | Cone Size Based | | Key Size Based | |
|---|---|---|---|---|---|---|---|
| | | 50% MUX | 100% MUX | 50% MUX | 100% MUX | 50% MUX | 100% MUX |
| b07 | 50 | 10.4 | 16.1 | 16.3 | 24.5 | 11.4 | 14.6 |
| | 80 | 21.9 | 29.5 | 31.7 | 42.2 | 19.8 | 30 |
| | 100 | 24.2 | 39.6 | 37.6 | 50.5 | 26.2 | 47.8 |
| b10 | 50 | 8.3 | 13.8 | 11.5 | 13.8 | 11.1 | 13 |
| | 80 | 15.7 | 27.2 | 38.2 | 43.5 | 18.2 | 25.8 |
| | 100 | 19.2 | 34.8 | 44.2 | 58.6 | 30.5 | 58.4 |
| b11 | 50 | 7.2 | 9.6 | 11.1 | 18.5 | 9.7 | 18.9 |
| | 80 | 11.9 | 18.7 | 27.8 | 38.2 | 25.3 | 36.4 |
| | 100 | 15.2 | 25.2 | 42.1 | 53.2 | 41.2 | 49.9 |
| b12 | 50 | 3.9 | 3.8 | 4.7 | 7 | 4.8 | 5.5 |
| | 80 | 5.3 | 7.6 | 9.9 | 12.9 | 8.7 | 13.1 |
| | 100 | 6.4 | 8.1 | 13.8 | 20.4 | 11.4 | 19.4 |
| b13 | 50 | 5.5 | 6.4 | 9.9 | 11.3 | 8.7 | 10.5 |
| | 80 | 7.7 | 13.3 | 17.2 | 18.8 | 14.3 | 22 |
| | 100 | 10.5 | 17.1 | 23.4 | 32.5 | 21.1 | 28.9 |

Figure 2 presents an example where two additional MUX key gates, $K_7$ and $K_8$, are inserted in the example from Figure 1. Both $K_7$ and $K_8$ expand the logic cones for $O_1$ and $O_3$ so that they include more keys. As a result, the overall effective key size is increased.

The effectiveness of each MUX key gate could be optimized by using an algorithm to carefully select the location of each MUX and the secondary data input for each MUX to maximize its impact. However, if the attacker knows the deterministic algorithm that is used for this, then the attacker may be able to exploit this information to decipher which input of the MUX is the real one and which is the artificial one. To avoid this, some randomness needs to be introduced. The proposed approach for inserting MUX key gates is the following. Identify the logic cone with the smallest effective key size. Randomly choose a node in that cone in which to insert the MUX key gate. Then randomly select three candidate locations from all nodes circuit-wide from which to connect the secondary data input for the MUX (which would not create feedback). Finally, use a heuristic to select one of the three candidates. Two possible heuristics were considered, one is to select the candidate that has the largest number of gates in its cone of the logic, and the other was to select the candidate that has the largest number of keys in its cone of logic. Experiments were performed for each of these heuristics on ITC99 circuits in which 50, 80, and 100 keys were inserted. The results are shown in Table 1. The first set of results is for the case where pure random selection of the secondary data input of the MUX is used (provided it doesn't create a feedback loop). The second set of results are for choosing the candidate with the largest number of gates in its cone. The last set of results are for choosing the candidate with the largest number of key gates in its logic cone. Results are shown for the case where a mix of 50% MUX key gates and 50% XOR key gates are used, and for the case where all key gates are MUX key gates. The results show that using the cone size based or key size based strategy is a significant improvement over random. In most cases, selecting the candidate that has the largest number of gates in its cone of logic proved to be more effective overall. That may be because the number of keys is continually increasing as the key gate insertion process proceeds, so the heuristic based on the number of keys has only partial information available at the time when it is used.

Note that the use of MUX key gates has a little more routing complexity than XOR key gates because it is a three input gate versus a two input gate, so it may be advantageous to use a mix of XOR and MUX key gates. This is explored in the experiments presented in the next section.

## 5. Experimental Results

Experiments were performed on the ITC99 benchmark circuits. In Table 2, results are shown for inserting 50, 80, and 100 keys in each benchmark circuit. The first set of results are for computing effective key size across the whole circuit without considering a cone-based attack. Area overhead was estimated with the 45nm FreePDK library [NCSU 11].

Results are shown where different percentage mixes of keys based on XOR gates and keys based on MUXes are randomly inserted in the design (100% XOR, 50% XOR and 50% MUX, and 100% XOR). Because the results vary depending on where they key gates are inserted, the experiment was performed many times and the average effective key size across all experiments are reported in the table. In the second set of results, random insertion of key gates is used again, but the effective key size is computed for a cone-based attack (as described in Sec. 3). As can be seen, the effective key size is much lower when the attacker employs the logic cone based attack described in this paper. It is also clear to see that using key gates based on MUXes helps considerably to improve the effective key size as it causes the logic cones to become highly overlapped. The last set of results is for using the proposed heuristic non-deterministic procedure for inserting key gates. As can be seen, this approach does not help when only inserting XOR key gates, but if MUX key gates are inserted, then it is very effective in increasing the effective key size in comparison to random insertion. This is because it guides the insertion procedure to more efficiently overlap the logic cones to counter a cone-based attack.

**Table 2.** Comparison of Effective Key Size

| Circuit Name | Keys Inserted | Whole Circuit | | | Cone-Based | | | | | | Area Overhead | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Random Insertion | | | Random Insertion | | | Heuristic Insertion | | | | | |
| | | 0% MUX | 50% MUX | 100% MUX | 0% MUX | 50% MUX | 100% MUX | 0% MUX | 50% MUX | 100% MUX | 0% MUX | 50% MUX | 100% MUX |
| b07 | 50 | 23.2 | 26.7 | 28.3 | 5.7 | 10.4 | 16.1 | 6 | 34.5 | 45.7 | 14.9% | 16.8% | 18.5% |
| | 80 | 33.7 | 40.5 | 43.9 | 7.6 | 21.9 | 29.5 | 6.9 | 56.1 | 71.1 | 21.9% | 24.4% | 26.7% |
| | 100 | 37 | 48.6 | 53 | 7.6 | 24.2 | 39.6 | 8.2 | 70.9 | 88.4 | 25.9% | 28.7% | 31.3% |
| b10 | 50 | 15.2 | 22.7 | 28.3 | 6.8 | 8.3 | 13.8 | 7 | 30.3 | 36.9 | 25.9% | 28.6% | 31.2% |
| | 80 | 21.2 | 34.8 | 46.3 | 9.5 | 15.7 | 27.2 | 10.5 | 50.5 | 62.4 | 35.8% | 39.1% | 42.1% |
| | 100 | 25 | 43.2 | 52.8 | 11.6 | 19.2 | 34.8 | 10.8 | 58.4 | 80.5 | 41.1% | 44.5% | 47.6% |
| b11 | 50 | 19.4 | 22.7 | 25.5 | 6.5 | 7.2 | 9.6 | 6.4 | 32.9 | 45.7 | 9.9% | 11.2% | 12.5% |
| | 80 | 28.6 | 36.8 | 39.7 | 8.3 | 11.9 | 18.7 | 8.7 | 52.3 | 67.5 | 14.9% | 16.8% | 18.6% |
| | 100 | 34.5 | 45.7 | 49.8 | 10.6 | 15.2 | 25.2 | 10.4 | 70.4 | 85.2 | 18.0% | 20.1% | 22.2% |
| b12 | 50 | 9.4 | 12.7 | 12 | 3.5 | 3.9 | 3.8 | 3.8 | 7.4 | 30.7 | 6.7% | 7.6% | 8.5% |
| | 80 | 14.9 | 17.3 | 24.1 | 4.7 | 5.3 | 7.6 | 5 | 29.5 | 59.8 | 10.3% | 11.7% | 13.0% |
| | 100 | 17 | 22.6 | 26.7 | 4.5 | 6.4 | 8.1 | 5.3 | 37.5 | 80 | 12.6% | 14.2% | 15.7% |
| b13 | 50 | 8.6 | 13.9 | 16.5 | 4.1 | 5.5 | 6.4 | 4.3 | 13.5 | 28.8 | 20.9% | 23.3% | 25.5% |
| | 80 | 12 | 21.3 | 29.9 | 5.1 | 7.7 | 13.3 | 4.6 | 29 | 55 | 29.7% | 32.7% | 35.4% |
| | 100 | 12.2 | 27.5 | 34.8 | 5.4 | 10.5 | 17.1 | 5.4 | 39.1 | 75 | 34.5% | 37.7% | 40.7% |
| b14 | 50 | 16.7 | 17.9 | 18.5 | 7.3 | 7.4 | 8.1 | 6.8 | 26.3 | 45.6 | 1.2% | 1.3% | 1.5% |
| | 80 | 25.5 | 26.9 | 27.8 | 10.9 | 11.3 | 11.6 | 11 | 52.6 | 74.8 | 1.9% | 2.1% | 2.4% |
| | 100 | 30.3 | 31.8 | 32.9 | 12.1 | 12.8 | 13.7 | 13.8 | 67.8 | 93.3 | 2.3% | 2.7% | 3.0% |
| b15 | 50 | 19.9 | 19.9 | 21.2 | 4.3 | 6 | 6.3 | 7.1 | 16.8 | 28.3 | 0.8% | 0.9% | 1.0% |
| | 80 | 32.4 | 33.3 | 33.9 | 5.3 | 8.4 | 12.1 | 7.7 | 52.1 | 74 | 1.2% | 1.4% | 1.6% |
| | 100 | 41.2 | 43.5 | 43.8 | 6.1 | 14 | 17.7 | 7.7 | 62 | 90.2 | 1.5% | 1.7% | 2.0% |

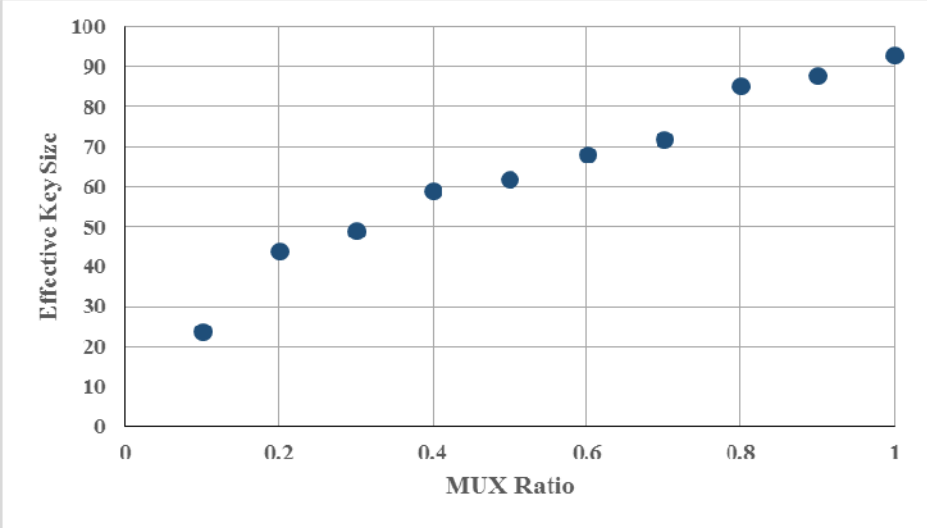


**Figure 3.** Effective Key Size for Cone-Based Attack for Insertion of 100 Keys in Circuit *b13*

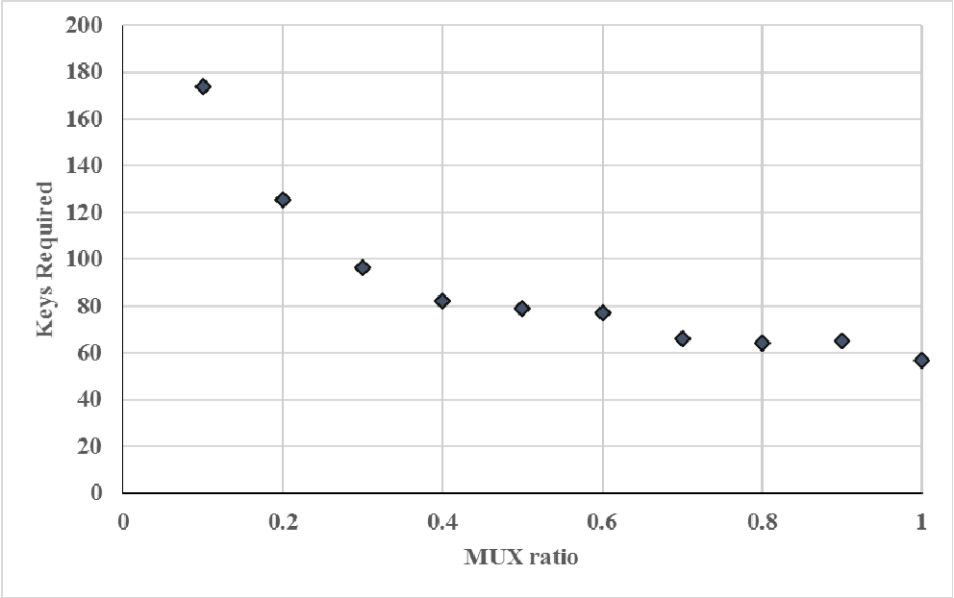


**Figure 4.** Number of Keys Inserted to Reach 50 Effective Keys for Cone-Based Attack for Circuit *b11*

Regardless of the percentage of MUX and XOR used, for smaller circuits like *b10*, adding 100 key gates results in a large area overhead. However, for larger circuits like *b15*, the overhead for adding 100 keys is only around 2%.

Figure 3 shows the number of effective keys when 100 key gates are inserted for circuit *b13* under different ratios of MUX and XOR key gates. The *X*-axis shows the percentage of MUXes inserted. The *Y*-axis shows the number of effective keys can be extracted after insertion. It is clear that the number of effective keys monotonically increased as the ratio of MUXes increased.

Figure 4 shows a graph measure how many total gates need to be inserted to achieve an effective key size of 50 for a cone-based attack for circuit *b11* for different ratios of MUX key gates and XOR key gates. The *x*-axis shows the percentage of MUX key gates that are inserted (with the rest being XOR key gates), and the y-axis shows the total number of key gates that need to be inserted to reach an effective key size of 50. As can be seen, as the percentage of MUX key gates is increased, fewer total key gates need to be inserted to achieve the same effective key size. However, the marginal improvement tapers off at around 40% of MUX gates. Since there is less overhead for inserting XOR key gates because they are 2-input gates versus MUX key gates that are 3-input gates, it may be cost effective to only insert 40% MUX key gates and the rest XOR key gates in this example to achieve a particular level of security as opposed to using all MUX key gates.

## 6. Conclusions

The worst-case number of operations for a brute force attack to determine key values can be significantly lower if each logic cone is iteratively considered going from easiest to hardest. Techniques to counter this are needed to ensure strong logic obfuscation. It was shown that inserting MUX key gates are an effective approach for increasing the security against this type of attack. The most cost effective approach is to use a mix of XOR key gates and MUX key gates.

## References

[Baumgarten 10] 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, Jan. 2010.

[Chakraborty 09a] R. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," *Proc. of International Conference on Computer-Aided Design, (ICCAD)*, pp.113-116, 2009

[Chakraborty 09b] R. Chakraborty, and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Trans. on Computer-Aided Design*, Vol. 28, No. 10, pp. 1493-1502, Oct. 2009.

[NCSU 11] NCSU EDA FreePDK45 Wiki, Last modified 2011, http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[Rajendran 12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *Proc. of Design Automation Conference (DAC)*, pp. 83-89, 2012.

[Rajendran 14] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, R. Karri, "Fault Analysis-Based Logic Encryption," *to appear in IEEE Transactions on Computers*.

[Roy 08] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *Proc. Design Automation and Test in Europe (DATE)*, pp. 1069-1074, 2008.

[Roy 10] J. Roy, F. Koushanfar, and I. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, pp. 30-38, 2010.

[Suh 07] G. Suh and S. Devadas. "Physical unclonable functions for device authentication and secret key generation," *Proc. of Design Automation Conference (DAC)*, pp. 9-14, 2007.