Logic Encryption: A Fault Analysis Perspective

Jeyavijayan Rajendran
NYU–Poly
Brooklyn, NY
jrajen01@students.poly.edu

Youngok Pino
Air Force Research Lab
Rome, NY
youngok.pino@rl.af.mil

Ozgur Sinanoglu
NYU–AD
Abu Dhabi, UAE
ozgursin@nyu.edu

Ramesh Karri
NYU–Poly
Brooklyn, NY
rkarri@poly.edu

***Abstract*—The globalization of Integrated Circuit (IC) design flow is making it easy for rogue elements in the supply chain to pirate ICs, overbuild ICs, and insert hardware trojans; the IC industry is losing approximately $4 billion annually [1], [2]. One way to protect the ICs from these attacks is to encrypt the design by inserting additional gates such that correct outputs are produced only when specific inputs are applied to these gates.**

The state-of-the-art logic encryption technique inserts gates randomly into the design [3] and does not necessarily ensure that wrong keys corrupt the outputs. Our technique ensures that wrong keys corrupt the outputs. We relate logic encryption to fault propagation analysis in IC testing and develop a fault analysis based logic encryption technique. This technique achieves 50% Hamming distance between the correct and wrong outputs (ideal case) when a wrong key is applied. Furthermore, this 50% Hamming distance target is achieved by using a smaller number of additional gates when compared to random logic encryption.

I. Introduction

A. 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 [4]. As the IC design flow is distributed worldwide, hardware is prone to new kinds of attacks such as reverse engineering and IP piracy [5]. An attacker, anywhere in this design flow, can reverse engineer the functionality of an IC/IP. He/she can then steal and claim ownership of the IP. An untrusted IC fabrication company may overbuild ICs and sell them illegally. Finally, rogue elements in the fabs may insert malicious circuits into the design without the designer's knowledge [4]. Because of these attacks, the semiconductor industry loses $4 billion annually [1], [2].

If a designer is able to conceal the functionality of an IC while it passes through the different, potentially untrustworthy phases of the design flow, these attacks can be thwarted [3].

B. Logic encryption

Logic encryption[1] hides the functionality and the implementation of a design by inserting some additional circuit elements into the original design. In order for the design to exhibit its correct functionality (produce correct outputs), a valid key has to be supplied to the encrypted design. Upon applying a wrong key, the encrypted design will exhibit a wrong functionality (produce wrong outputs).

While it passes through the untrusted design phases, an IC will be in an encrypted form so that its functionality is not revealed; this prevents reverse engineering, cloning, trojan insertion and overbuilding. The designer gives the valid key to the end-user of the IC so that the end-user can enable the IC to exhibit its correct functionality.

C. Criteria for logic encryption

In an encrypted design, a wrong key should result in a wrong output for all input patterns. If a correct output is produced for a wrong key, then the encryption procedure is weak and the attacker will benefit. If a wrong key affects only one or a few of the output bits, then the attacker might be able to tolerate the wrong outputs. If all the output bits are affected, then the wrong output is the complement of the correct output. Hence, ideally, a wrong key should affect half of the output bits i.e., *the Hamming distance between the correct and wrong outputs should be 50%* [7]. This 50% Hamming distance renders a very high obscureness to an attacker.

Furthermore, in another form of attack, end-users can collude by sharing their valid keys. To prevent this collusion attack, *each IC should have its own unique key* [8].

D. Previous work

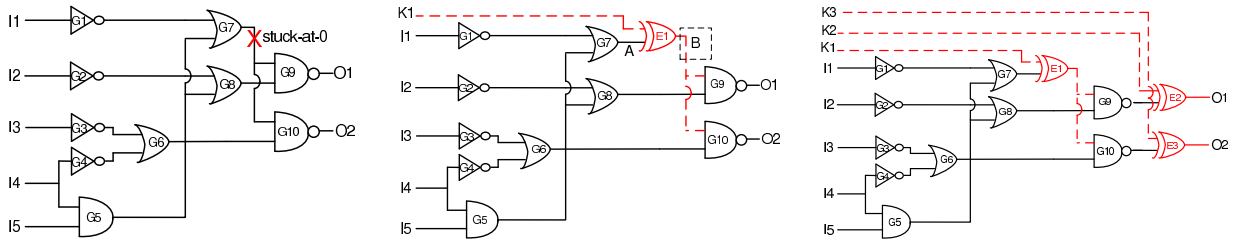
Logic encryption techniques can be broadly classified into two types—sequential and combinational. In sequential logic encryption, additional logic (black) states are introduced in the state transition graph [5], [6]. The state transition graph is modified in such a way that the design reaches a valid 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 encryption, XOR/XNOR gates are introduced to conceal the functionality of a design [3]. Usually, one of the inputs in these inserted gates serves as a 'control input' which is a newly added primary input. One can configure these gates as buffers or inverters using these control inputs. The values applied to these control inputs are the keys.

Apart from sequential and combinational elements, memory elements are also inserted into the design [7]. The circuit will function correctly only when these elements are configured/programmed correctly. However, the introduction of

[1]Logic encryption of hardware does not mean encrypting the design file by a cryptographic algorithm, instead it means hiding the hardware's functionality. Researchers have previously used 'logic obfuscation' [3], [6] for this purpose. Obfuscation, however, has a different meaning in software. An obfuscated program is difficult to reconstruct even if its functionality is known. Obfuscation hides the implementation and not the function. To highlight this difference we use logic encryption to denote that the functionality is encrypted when the valid key is not applied to the design.

978-3-9810801-8-6/DATE12/



(a) A faulty circuit

(b) An encrypted circuit with a wrong key (K1 = 1) equivalent to the faulty circuit

(c) A circuit encrypted with three XOR gates (E1, E2, and E3)

Fig. 1: Relation between logic encryption and fault analysis in IC testing – fault excitation, propagation, and masking.

memory elements into the circuit will incur significant performance overhead.

E. Our approach

In combinational logic encryption, XOR/XNOR gates are introduced at random locations in a design [3]. We show that, when gates are randomly inserted into the design, a wrong key does not necessarily affect the output as its effects are not propagated to the outputs. This is similar to an IC testing scenario where the effect of a fault may not propagate to the output. Our approach relates these two scenarios and ensures that the effect of wrong keys always propagates to the outputs.

As traditional IC testing algorithms analyze the effect of faults in a circuit and provide ways to propagate their effect to the circuit outputs, we leverage them to perform logic encryption. Our technique uses conventional fault simulation techniques and tools [9] to guide the XOR/XNOR insertions using and corrupts 50% of output bits for a wrong key.

In order to prevent collusion attack, Physical Unclonable Functions (PUFs) are incorporated to produce unique user keys for each IC even though all the ICs are encrypted with the same valid key [3], [8]. To prevent a user getting access to the valid key used for encryption, an RSA unit separates the user key from the valid key. As an RSA unit has a huge overhead, we have designed and used a simple Logic Encryption Cell (LEC) which includes XOR/XNOR gates for encryption.

The contributions of this paper are:

- relating logic encryption to fault analysis in IC testing
- a fault analysis-based logic encryption algorithm
- design of a low-overhead logic encryption cell

II. Fault Analysis Based Logic Encryption

A. Logic encryption from a fault analysis perspective

We will now describe how to encrypt a design using XOR/XNOR gates in such a way that any wrong key causes a wrong output. This is similar to the situation where a circuit produces a wrong output when it has a fault that has been excited and propagated to the outputs. The following observations relate logic encryption and fault analysis in IC testing. We will use these observations to insert XOR/XNOR gates.

Fault excitation: Application of a wrong key can be associated with the activation of a fault. For a wrong key, either a stuck–at–0 (s–a–0) or stuck–at–1 (s–a–1) fault will get excited, when XOR/XNOR gates are used for encryption.

Example 1: Consider the C17 circuit encrypted with one XOR gate (E1) as shown in Figure 1(b). If a wrong key (K1=1) is applied to the circuit, the value of net B is the negated value of net A. This is same as exciting a s-a-0 (when A=1) or s-a-1 (when A=0) fault at the output of G7 as shown in Figure 1(a).

Fault propagation: Not all wrong keys can corrupt the output as the effects of a wrong key may be blocked for some of the input patterns. This is similar to the scenario where not all input patterns can propagate the effect of a fault to the output.

Example 2: Consider the circuit shown in Figure 1(b). Let a wrong key (K1 = 1) be applied to the circuit. For the input pattern 00000, a s–a–0 fault gets excited at the output of E1 and propagated to both outputs. The value at the output of the gate E1 is 0 instead of 1, and the output is 11 instead of the correct output 00.

For the input pattern 01110, even though the s–a–0 fault gets excited at the output of E1, the output is 00, which is the same as the functional output, as the fault effects have been blocked.

To propagate the effect of an excited fault, in our case the wrong key, non-controlling values should be applied to the other inputs of the gates that are on the propagation path of the fault. Since not all input patterns guarantee the non-controlling values on the fault propagation path, a wrong key will not always corrupt the output.

Fault masking: Inserting a single XOR/XNOR gate and applying a wrong key is equivalent to exciting a single stuck-at fault. Inserting multiple XOR/XNOR gates and applying a wrong key is equivalent to exciting multiple stuck-at faults. However, when multiple faults are excited, they might mask each others' effect. Therefore, in logic encryption, when multiple XOR/XNOR gates are inserted, the effect of one XOR/XNOR gate might mask the others' effect.

Example 3: Consider the encrypted circuit shown in Figure 1(c). When the key bits (K1, K2, and K3) are 000, the correct functional output is 00 for the input pattern '00000'. However, if the key bits are 111 (wrong key), the effect introduced by the XOR gate E1 is masked by the XOR gates E2 and E3 and produces a correct output 00. Thus similar to fault masking, the effect of one XOR gate is masked by the effect of the other two XOR gates.

Goal: Insert XOR/XNOR gates such that a wrong key will affect 50% of the outputs. In terms of fault simulation, this goal can be stated as finding a set of faults, which together

will affect 50% of the outputs for a wrong key.

Challenge: Fault simulation tools rely on the assumption of a single stuck-at fault model (only one fault can be present at any time). Thus, existing commercial fault simulation tools can insert only one XOR/XNOR gate at a time.

We overcome this challenge by using a greedy iterative approach where XOR/XNOR gates are inserted iteratively. In each iteration, the fault that has the potential of propagating to a maximum number of outputs dictates the location of the XOR/XNOR gate to be inserted. For every iteration (except the first iteration), the XOR/XNOR gates inserted at previous iterations are provided with random wrong keys thereby emulating a multiple stuck-at fault scenario, and accounting for all the previous XOR/XNOR insertions. An algorithm is presented in the following subsection to perform this logic encryption.

B. Fault impact

To insert an XOR/XNOR gate, we need to determine the location in the circuit where, if a fault occurs, it can affect most of the outputs for most of the input patterns. To determine this location, we use the concept of *fault impact* defined by Equation 1. From a set of test patterns, we compute the number of patterns that detect the s-a-0 fault (No. of Test Patterns$_{\text{s-a-0}}$) at the output of a gate G_x and the cumulative number of output bits that get affected by that s-a-0 fault (No. of Outputs$_{\text{s-a-0}}$). Similarly, we compute No. of Test Patterns$_{\text{s-a-1}}$ and No. of Outputs$_{\text{s-a-1}}$.

$$\begin{aligned}\text{Fault impact} = &(\text{No. of Test Patterns}_{\text{s-a-0}} \times \text{No. of Outputs}_{\text{s-a-0}}) \\ &+(\text{No. of Test Patterns}_{\text{s-a-1}} \times \text{No. of Outputs}_{\text{s-a-1}})\end{aligned} \tag{1}$$

Upon inserting the XOR/XNOR gate for encryption at the location with the highest *fault impact*, an invalid key will likely have the most impact on the outputs (i.e., the wrong outputs appear).

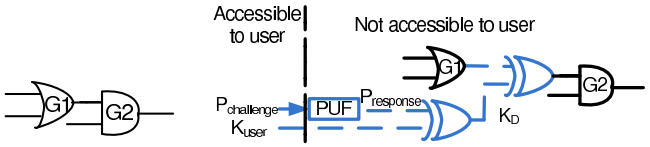
C. Algorithm to insert XOR/XNORs

Algorithm 1 greedily selects the best 'N' locations in a circuit to insert the XOR/XNOR gates. The location with the highest fault impact is calculated and an XOR/XNOR gate is inserted at that location. The algorithm considers the previously inserted XOR and XNOR gates in this calculation.

Input : Netlist, KeySize
Output: Encrypted netlist
for $i \leftarrow 1$ **to** $KeySize$ **do**
  foreach *gate* $j \in Netlist$ **do**
    Compute $FaultImpact$;
  end
  Select the gate with the highest $FaultImpact$;
  Insert XOR gate and update the $NetList$;
  Apply $Test\ Patterns$;
end

Algorithm 1: Fault analysis based logic encryption.

The algorithm terminates on inserting the maximum number of XOR/XNOR gates.



(a) Before encryption (b) After encryption

Fig. 2: Logic encryption cell (dotted components): A primitive for encrypting logic.

D. Logic encryption cell

We incorporate a PUF circuit into each IC to produce a unique key for that IC. To make an encrypted design functional, two keys, K_{user} and $P_{challenge}$, are given to the user [3]. $P_{challenge}$ is applied as a challenge to the PUF. The response from the PUF, $P_{response}$, is used as a decryption key in an RSA decryption unit. K_{user} is used as the cipher text to the RSA decryption unit and the resultant plain text will be K_D, the key to make the encrypted design functional [3]. The RSA unit prevents the user to determine K_D from the user keys, K_{user} and $P_{challenge}$.

An RSA decryption unit uses 10,000 gates resulting in a huge area overhead [3], [10]. This problem can be overcome by using a simple Logic Encryption Cell (LEC) shown in Figure 2. We replace the RSA decryption unit with a set of XOR gates to obtain K_D from K_{user} and $P_{response}$, the response from the PUF. In this LEC, to determine K_D, the attacker should have access to K_{user} and $P_{response}$. However, the attacker cannot determine $P_{response}$ from $P_{challenge}$ due to the PUF circuit's characteristics. Thus, the LEC comprising of a PUF and XOR/XNOR gates provides a security level equivalent to the logic encryption mechanism that uses a PUF and an RSA unit with XOR/XNOR gates.

III. Results

A. Experimental Setup

The effectiveness of the proposed technique is analyzed using ISCAS-85 combinational benchmarks. We used the HOPE fault simulation tool [11] to calculate the fault impact of each gate. We applied 1000 random input patterns to a netlist and observed the true outputs. We set the key size as 128 bits. We then calculated the fault impact for all possible faults in the circuit. We applied valid and random wrong keys to an encrypted netlist and determined the Hamming distance between the corresponding outputs. The area, power, and delay overhead were obtained using the Cadence RTL compiler.

B. Analysis of Hamming distance

Fault analysis based approach is compared with the random insertion approach [3] and the corresponding results are shown in Figure 3. When the XOR/XNOR gates are randomly inserted, 50% Hamming distance is not achieved except for the smallest benchmark C17. Masking of faults is the main reason for this poor performance. The effect of wrong keys are blocked for most of the input patterns as discussed in

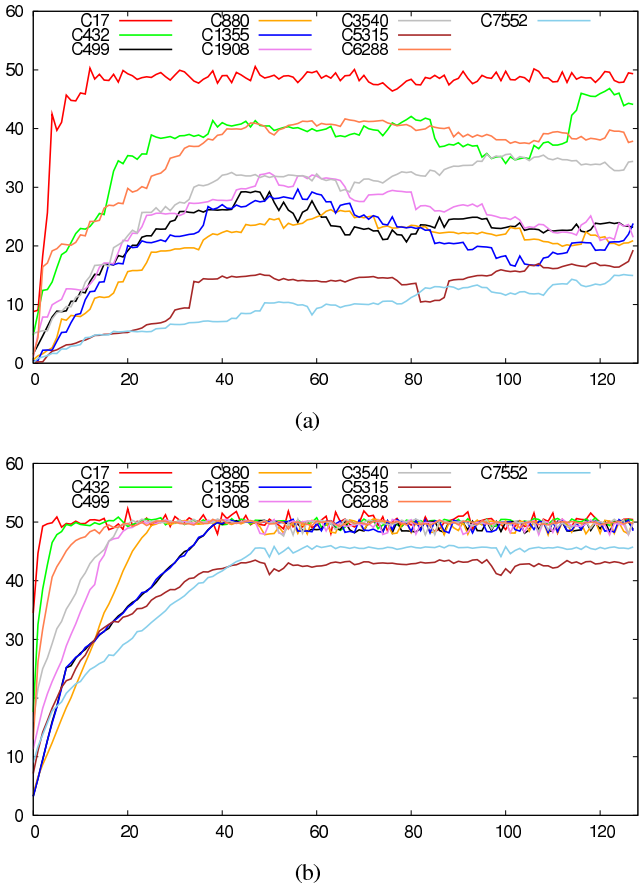


Fig. 3: Comparison of (a) random insertion based logic encryption and (b) fault analysis based logic encryption for different ISCAS-85 benchmark circuits.

Example 4. However, fault analysis based insertion achieves 50% Hamming distance for all benchmarks except for two — C5315 and C7552. The number of outputs in benchmarks C5315 and C7552 is 123 and 108, respectively which is very high and hence it is hard to achieve the 50% mark. However, the fault analysis based approach performs well in all the other benchmarks as it takes the fault masking effects into account.

The slope of the lines in Figure 3(a) and 3(b) indicate the performance of the random and the fault analysis based insertions. If the line is steeper, 50% Hamming distance is achieved with a smaller number of additional encryption, hence performance overhead will be smaller. Fault analysis based logic encryption has a smaller overhead than the random insertion as it uses a smaller number of additional gates to achieve the target Hamming distance.

In fault analysis based logic encryption, once a design achieves the 50% mark, its Hamming distance value does not deviate more on inserting more gates. Hence, one can increase the key size without deviating from the 50% Hamming distance mark.

Table I compares the Hamming distance between the random and fault analysis based logic encryptions. The second column shows the number of XOR/XNOR gates to be inserted to achieve 50% Hamming distance using the fault analysis based insertion approach. The third and fourth columns show the corresponding Hamming distance of the random and fault analysis based logic encryptions, respectively, for the number of XOR/XNOR gates listed in the second column. It can be seen that, on average, the fault analysis based logic encryption achieves a Hamming distance value which is twice that of the random insertion (except for the smallest benchmark C17). This is because fault analysis based logic encryption identifies more effective locations to insert the gates than the random insertion based logic encryption.

TABLE I: The number of XOR/XNOR gates required to achieve 50% Hamming distance using fault analysis based logic encryption and the corresponding Hamming distance for random insertion based logic encryption.

Benchmark	No. of XOR/XNOR gates	Hamming distance (%)	
		Random	Fault analysis
C17	6	42	51
C432	17	29	50
C499	40	26	50
C880	28	19	50
C1355	42	26	50
C1908	28	26	50
C3540	22	23	50
C5315	97	15	44
C6288	27	32	50
C7552	89	13	46

C. Resiliency against attacks

Even though a logic encryption technique may meet a security criterion, it can be susceptible to several attacks which are described below.

1) Brute-force attack: In this attack the attacker applies all combinations of key bits and tries to figure out the correct key. Increasing the key size to a large value such as 128 makes this attack computationally infeasible for an attacker. However, increasing the key size should not degrade other security properties. While increasing the key size decreases the Hamming distance for random insertion based logic encryption, the Hamming distance value is not degraded for fault analysis based logic encryption as depicted in Figure 3(a) and Figure 3(b).
2) Collusion attack: In this attack, the attacker uses the key meant for IC_X on IC_Y. However, when the IC_X's key is used on IC_Y, the resulting response from PUF_Y will be different from the PUF_X. A wrong value is applied to the key inputs of the encrypted design in IC_Y and the circuit will not produce correct outputs.
3) LEC removal attack: The attacker can attempt to remove the LECs from the encrypted netlist and replace them randomly with a buffer or an inverter. However, the attacker cannot easily distinguish between the XOR/XNOR gates used for encryption and the gates in the original netlist as a logic synthesizer will merge the XOR/XNOR gates with the original gates in the netlist.

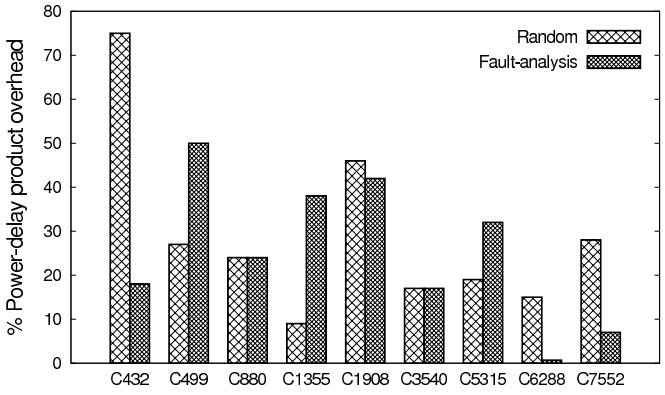


Fig. 4: Power-delay product overhead of random and fault analysis based logic encryptions.

D. Power and delay overhead

Figure 4 shows the power-delay product overhead of the benchmarks that are encrypted with the number of XOR/XNOR gates listed in the second column of Table I. The overhead for the C17 circuit is 262% and 480% using the random insertion and the fault analysis based logic encryptions, respectively. For the benchmarks C499, C1355, C3540, and C5315, the fault analysis based logic encryption inserts the XOR/XNOR gates at the critical path of the design thereby increasing the delay of the circuit. For the benchmarks C6228 and C7552, the XOR/XNOR gates are inserted in paths with high slack. Hence the impact on the power-delay product is low. Even though for some of the benchmarks, the random insertion based logic encryption has a smaller overhead, it achieves only half the Hamming distance value achieved by the fault analysis based logic encryption.

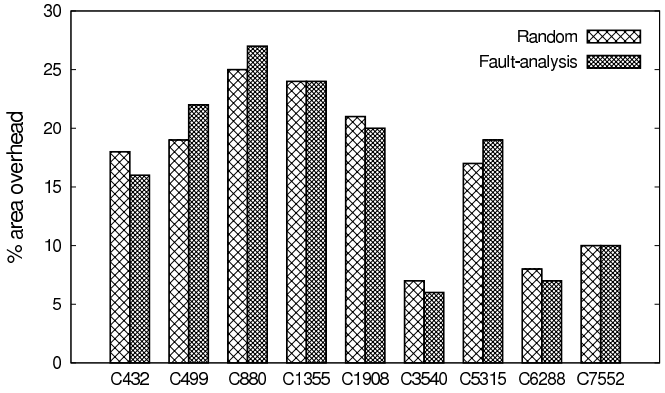


Fig. 5: Area overhead of random and fault analysis based logic encryptions.

E. Area overhead

Figure 5 shows the area overhead of the benchmarks that are encrypted with the number of gates shown in the second column of Table I. Here, we do not include the overhead due to the PUF and XOR gates in the LEC as both techniques will use the similar LEC structure. Even though same number of XOR/XNOR gates are inserted in both the methods, depending upon the inserted XOR/XNOR gates' location, the logic synthesizer merges the inserted XOR/XNOR gates and the gates in the original netlist. Hence, the area overhead is different for fault analysis based logic encryption and random insertion based logic encryption. However, for a given overhead (number of gates) the fault analysis based logic encryption has better security properties (Hamming distance) than random insertion. The RSA decryption unit used in [3] has an overhead of 10,000 gates [3], [10]. We replace this RSA with 128 XOR gates without reduction in security.

IV. Discussion

A. Beyond Hamming distance criterion

Satisfying the Hamming distance criterion does not necessarily mean that the encryption technique is strong. Hence, Heys et al. described the *avalanche criterion* for a stronger evaluation [12]. An encrypted function/design satisfies the avalanche criterion, if changing one key bit changes half of the output bits. The avalanche factor of a design with N key bits and M outputs is defined as

$$\text{Avalanche factor} = \frac{1}{M \times 2^N} \times \sum_{j=1}^{M} \sum_{i=1}^{N} a_{i,j} \qquad (2)$$

where $a_{i,j}$ is '1' if flipping the key input 'i' flips the output bit 'j'. When 'N' is large, then a random set of patterns from 0 to 2^N is selected and applied. According to the avalanche criterion, the avalanche factor has to be 50%. Strong encryption techniques such as AES and DES [12], [13] have an avalanche factor of 50%.

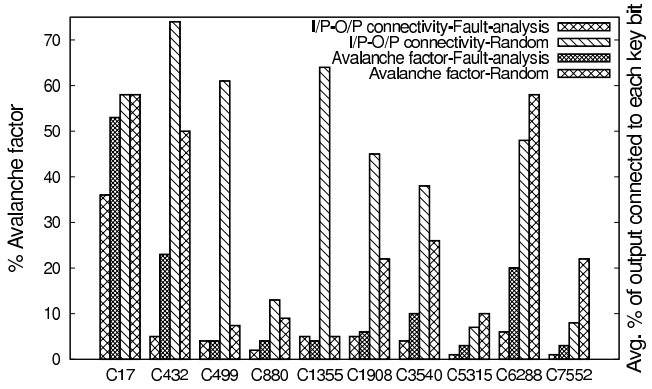


Fig. 6: Avalanche factor for the random and the fault analysis based logic encryptions.

Let us now assess the strength of the random and fault analysis based logic encryption techniques in terms of the avalanche criterion. Figure 6 shows the avalanche factor for the benchmarks. Only the smallest benchmark, C17, achieved the 50% target for the avalanche criterion when encrypted using the fault analysis based approach. For all the other benchmarks, neither of the techniques achieved the 50% target for the avalanche criterion. The avalanche factor is low in both types of encryption because the key bits are not connected to all output bits.

The average percentage of output bits connected to each key bit is plotted in Figure 6. It shows that the connectivity,

on average, is 45% and 18% for random insertion and fault analysis based logic encryption techniques, respectively. Even though random insertion based logic encryption has a better connectivity between outputs and key bits, it has a lower avalanche factor than fault analysis based logic encryption. This is because in random insertion based logic encryption, the propagation of faults to the outputs are blocked whereas fault analysis based logic encryption enables the fault propagation.

B. Beyond XOR/XNOR gates

One can also use other gates such as the AND, the OR, the inverter, and the multiplexer to perform logic encryption. The security properties of the design may vary depending upon the type of gates used for encryption.

C. Beyond PUFs

While PUFs are a low-cost security primitive to provide unique per-chip keys, their stability is affected by device aging and operating conditions. Thus, they need special error correcting mechanisms [14]. Instead of PUFs, one can use simple fuse-based RAMs and program the keys into them in different ICs to create unique keys for each chip.

V. Conclusion

Fault analysis based logic encryption achieved 50% Hamming distance between the correct and the corresponding wrong outputs when an invalid key is applied to the design. However, our technique does not achieve 50% avalanche factor. This is because not all key bits are connected to all outputs. One can develop a similar insertion algorithm to achieve the avalanche criterion. In this work, we took the average Hamming distance as the criterion. To overcome the problems of averaging, one can perform insertion by giving more weighs to the number of inputs that affect the key.

Since we have used a single fault simulator, we developed an iterative algorithm to determine the *fault impact* in the presence of fault masking. Logic encryption can also be performed non-iteratively by using a fault simulator that supports multiple stuck-at fault models to account for fault masking effects. To encrypt a large design such as C7552, our algorithm takes two hours. The algorithm can be scaled to large designs by applying it on smaller partitions in these designs

VI. 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.

References

[1] KPMG, "Managing the risks of counterfeiting in the information technology," www.agmaglobal.org/press_events/press_docs/Counterfeit_WhitePaper_Final.pdf, 2006.

[2] 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.

[3] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *Proceedings of the IEEE/ACM Design, Automation and Test in Europe*, pp. 1069–1074, 2008.

[4] "Defense Science Board (DSB) study on High Performance Microchip Supply," http://www.acq.osd.mil/dsb/reports/ADA435563.pdf, Feb. 2005.

[5] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," *Proceedings of USENIX security*, pp. 291–306, 2007.

[6] R. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[7] 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.

[8] G. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," *Proceedings of the IEEE/ACM Design Automation Conference*, pp. 9–14, 2007.

[9] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits," *Kluwer Academic Publishers, Boston*, 2000.

[10] "Sciworx RSA Co-Processor," http://www.sci-worx.com/products/cryptography/rsa-co-processor.html.

[11] H. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.

[12] H. Heys and S. Tavares, "Avalanche characteristics of substitution-permutation encryption networks," *IEEE Transactions on Computers*, vol. 44, no. 9, pp. 1131 –1139, 1995.

[13] "Specification for the Advanced Encryption Standard (AES)," *Federal Information Processing Standards Publication 197*, 2001.

[14] M.-D. Yu and S. Devadas, "Secure and Robust Error Correction for Physical Unclonable Functions," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.