Fault Analysis-Based Logic Encryption

Jeyavijayan Rajendran, *Student Member*, *IEEE*, Huan Zhang, *Student Member*, *IEEE*, Chi Zhang, Garrett S. Rose, *Member*, *IEEE*, Youngok Pino, *Senior Member*, *IEEE*, Ozgur Sinanoglu, *Member*, *IEEE*, and Ramesh Karri, *Senior Member*, *IEEE*

Abstract—Globalization of the integrated circuit (IC) design industry is making it easy for rogue elements in the supply chain to pirate ICs, overbuild ICs, and insert hardware Trojans. Due to supply chain attacks, the IC industry is losing approximately $4 billion annually. One way to protect 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, but 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 enables a designer to controllably corrupt the outputs. Specifically, to maximize the ambiguity for an attacker, this technique targets 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 using a smaller number of additional gates when compared to random logic encryption.

Index Terms—Automatic test pattern generation, combinational logic circuit, hardware security, IC piracy, integrated circuit testing, IP piracy, logic obfuscation

1 Introduction

1.1 Motivation

DUE to the ever increasing complexity of constructing and/or maintaining a foundry with advanced fabrication capabilities, many semiconductor companies are becoming fabless. Such fabless companies design integrated circuits (IC) and send them to an advanced foundry, which is usually off-shore, for manufacturing. Also, the criticality of time-to-market has forced companies to buy several IC intellectual property (IP) blocks to use them in their systems-on-chip. The buyers and sellers of these IP blocks are distributed worldwide.

Globalization of the IC design industry has led to several new kinds of attacks on hardware. An attacker, anywhere in the design flow, can reverse engineer the functionality of an IC/IP [1], [2]. He/she can then steal and claim ownership of the IP [3]. An untrusted IC fabrication company may also overbuild ICs and sell them illegally. Finally, rogue elements in foundries may insert malicious circuits into the design without the designer's knowledge [4]. Due to such attacks, the semiconductor industry loses $4 billion annually [1], [2]. Such attacks have led IP and IC designers to re-evaluate trust in hardware [4].

- *J. Rajendran, H. Zhang, C. Zhang, and R. Karri are with the Electrical and Computer Engineering Department, Polytechnic Institute of New York University, Brooklyn, NY 11209. E-mail: {jrajen01, hzhang10, czhang10}@students.poly.edu, rkarri@poly.edu.*
- *G.S. Rose is with the Trusted Systems Branch, Air Force Research Laboratories, Rome, NY 13441. E-mail: garrett.rose@rl.af.mil.*
- *Y. Pino is with the Information Sciences Institute, Arlington, VA 22209. E-mail: ypino@isi.edu.*
- *O. Sinanoglu is with the New York University, Abu Dhabi, UAE. E-mail: os22@nyu.edu.*

Manuscript received 19 Nov. 2012; revised 04 Sep. 2013; accepted 17 Sep. 2013. Date of publication 30 Sep. 2013; date of current version 16 Jan. 2015. Recommended for acceptance by P. Schaumont.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.193

While the IC design flow spans many countries, not all countries have strict laws against intellectual property theft. As reported in [5], only a few countries such as USA and Japan have strict laws to protect IC designs against intellectual property theft. Thus, every IC/IP designer bears an additional responsibility to protect his/her design. 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 [6], [7]. For this purpose, researchers have proposed a technique called logic encryption.

1.2 Logic Encryption

Logic encryption[1] hides the functionality and the implementation of a design by inserting some additional gates called *key-gates* into the original design. In order for the design to exhibit its correct functionality (produce correct outputs), the 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).

Logic encryption techniques can thwart an untrusted foundry from illegally copying, reverse engineering, overproducing the IC design [3], [5]–[8], [11], and Trojan insertion [12]. As shown in Fig. 1, the IP provider and the designer are trusted. The foundry is not trustworthy or there is a rogue element in the foundry. The designer encrypts the modules using the proposed technique, synthesizes them using trustworthy computer-aided design tools, and sends the generated layout masks to the untrustworthy foundry. The key-inputs of the key-gates are connected to the data lines of a tamper-proof

1. Logic encryption of a hardware design does not mean encrypting the design file by a cryptographic algorithm, and rather it means encrypting a design's functionality. Obfuscation of a module hides only its functionality but it does not prevent black-box usage [25]. Logic encryption prevents this black box usage in addition. Hence, we use the term "encryption" and not "obfuscation."

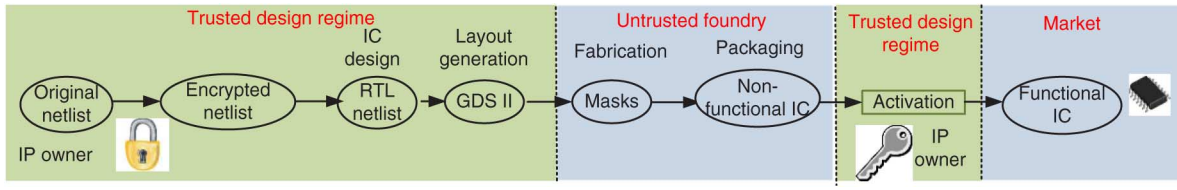

Fig. 1. An IC design flow enhanced with logic encryption capabilities to thwart IC piracy [6], [7]. Before sending the design to an untrusted foundry, the designer encrypts the design using logic encryption techniques. The foundry then manufactures this encrypted design. On receiving the encrypted hardware, the IC designer activates it by applying the secret key and the IC is then sold in the market.

memory. When the designer sends the encrypted design to the foundry, he does not load the secret key into this memory as it can be recovered by an attacker in the foundry.

The foundry manufactures the IC and returns them to the designer. The designer then loads the secret key into the tamper-proof memory and makes the ICs functional. To prevent a user from reading out the secret key from the memory, the designer removes read/write access to this memory by blowing out the fuses in the read/write circuit. Furthermore, to prevent an attacker from reading-out the contents of the memory, it is designed to be tamper-proof. The designer or a trusted third party performs functional validation and manufacturing testing on this functional IC. Once they pass these tests, the functional ICs are packaged and sold.

An attacker in the foundry has access to the layout. He can copy the layout and overproduce the ICs (black-box usage). Alternatively, he can reverse engineer the netlist from the layout and reuse this netlist. Logic encryption seeks to prevent these attacks by encrypting all or critical modules in a design. Since the design is encrypted by the designer, the lack of the secret keys by the foundry renders any copies or overproduced ICs unusable. The attacker does not have access to good functional input-output pairs (as the design has already been encrypted previous to his access to the layout). Further, the attacker does not have access to the RTL and the test vectors. Unlike obfuscation techniques [3], [5]–[8], [11], [12], logic encryption protects against black-box usage and reverse engineering.

Though their application may differ depending up on the target attack, any logic encryption technique should satisfy two criteria [6], [7], [11], [12]: (1) wrong outputs should be produced on applying a wrong key, and (2) an attacker should not be able to retrieve the secret key. On inserting a sufficient number of key-gates, it becomes computationally infeasible for an attacker to determine the secret key. In this work, we propose fault analysis-based logic encryption to satisfy the first criterion. The proposed technique enables the designer to controllably corrupt the outputs.

1.3 Contributions

Previously proposed logic encryption techniques insert key-gates at random locations in a design [6], [7] (Section 7 describes these techniques in detail). We show that when gates are inserted randomly into the design, a wrong key may not necessarily affect the output as its effects may not be propagated to the outputs.

We then overcome this problem by relating it to an IC testing scenario where the effect of a fault may not propagate to the output. Furthermore, we also analyze how fault-analysis techniques such as fault activation, fault propagation, and fault masking can help perform stronger logic encryption.

We then leverage traditional IC testing algorithms to perform logic encryption. Our technique uses conventional fault simulation techniques and tools such as HOPE [10] to guide key-gate insertion and corrupt the output bits on applying a wrong key.

We also use 2:1 multiplexers (MUXes) as key-gates. We use fault-analysis techniques to guide MUX insertion.

The proposed techniques are analyzed by comparing the Hamming distance between the outputs on applying the valid key and a wrong key. The area, power, and delay overhead of the proposed techniques are reported.

We acknowledge the scenario where a designer has a limited power, delay, or area overhead budget for logic encryption. Hence, we also analyze the ability to produce wrong outputs for a given power/delay/area overhead for different logic encryption techniques.

Our work has the following unique features that differentiate it from the previous work:

1. Analyzes logic encryption from IC testing perspective.
2. Uses test principles to relate invalid key-bits to corrupted outputs.
3. The proposed fault-analysis approach is generic as it can be applied to any logic encryption mechanism.

1.4 Organization

Section 2 describes a metric for logic encryption followed by Section 3 with a motivating example explaining the necessity of the proposed work. Section 4 details the relationship between IC testing and logic encryption and proposes an algorithm to insert XOR gates and MUXes for logic encryption. Section 5 explains how to perform logic encryption when the available resources (power, area, delay overheads) are constrained. Discussions on security and limitations of the proposed technique are listed Section 6. Section 7 describes the previous work on logic encryption. Section 8 concludes the paper.

2 Metric for Logic Encryption

The defender (designer) has to prevent his IP from being copied by an attacker in the foundry and to prevent black-box usage. The attacker does not know the secret key used for encryption. Hence, he will apply a random key and in turn expect the module to become functional (i.e. to produce correct outputs). If he is lucky and if the module indeed produces correct outputs even when a random wrong key is applied, then it benefits him. If he is not lucky, the attacker has

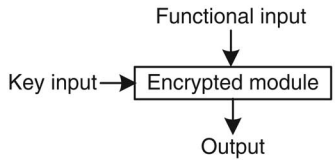

Fig. 2. A system with an encrypted module.

to try another key combination. Increasing the key-size increases the effort of an attacker. Thus, the objective of the defender is to make it harder for an attacker to retrieve the secret key. To make this happen, the defender needs the encrypted design to produce wrong outputs on applying a wrong key.

To formalize this, consider the system shown in Fig. 2. The system has an *M-bit* input, *N-bit* output, and is encrypted with K key bits. These bits are either logic '1' or '0'. Let $\mathrm{B} = \{0, 1\}$. Let $x \in \mathrm{B^M}$ be a functional input. Let $y \in \mathrm{B^N}$ be the correct output. Let $c \in \mathrm{B^K}$ be the correct key.

A module f encrypted with a key c should behave as follows:

1. On applying the correct key c, the module produces correct outputs for all input patterns.

$$\mathrm{f(x,z)}|_{\mathrm{z=c}} = \mathrm{y} \forall \times \in \mathrm{B^M}, \mathrm{y} \in \mathrm{B^N}.$$

2. On applying a wrong key, the module produces wrong outputs for all input patterns.

$$\mathrm{f(x,y)}|_{\mathrm{z} \neq \mathrm{c}} = y' \forall \times \in \mathrm{B^M}, \mathrm{z} \in \mathrm{B^K}, y' \in \mathrm{B^N}, \text{ where } y' \neq \mathrm{y}.$$

The Hamming distance between y' and y (HD(y,y')) can measure the difference between a correct output y and the corresponding wrong output y'. If $\mathrm{HD(y,y')} = 0$, then the outputs of the encrypted module are correct independent of the applied key. Thus, the corresponding encryption is weak. If $\mathrm{HD(y,y')} = \mathrm{N}$, then the wrong outputs of the encrypted design are still correlated to the original outputs, but this time inversely. The corresponding encryption is still weak and the attacker can obtain the correct output by complementing the output.

A defender has to encrypt the module such that an attacker, with the knowledge of the publically available logic encryption objectives and algorithms, is not able to obtain the correct outputs by applying a wrong key. This can be done by minimizing the correlation between the corrupted and the original outputs, and thus by maximizing the ambiguity for the attacker. If there are P output-bit combinations that an attacker is forced to consider corresponding to every input combination, then larger values of P imply greater ambiguity for the attacker. Obviously, a defender has to maximize P. This is analogous to traditional cryptography, where increasing the key-size increases the ambiguity for an attacker.

If *Q-out-of-N* output bits are wrong (i.e. $\mathrm{HD(y,y')} = \mathrm{Q}$), then P can be computed as $\binom{\mathrm{N}}{\mathrm{Q}}$.

If $\mathrm{Q} = 0$ (i.e. $\mathrm{HD(y,y')} = 0$), then $\mathrm{P} = 1$; it benefits the attacker. If $\mathrm{Q} = \mathrm{N}$ (i.e. $\mathrm{HD(y,y')} = 0$), then $\mathrm{P} = 1$; it benefits the attacker. P is maximum when $\mathrm{Q} = \mathrm{N}/2$ (i.e. when $\mathrm{HD(y,y')} = \mathrm{N}/2$).

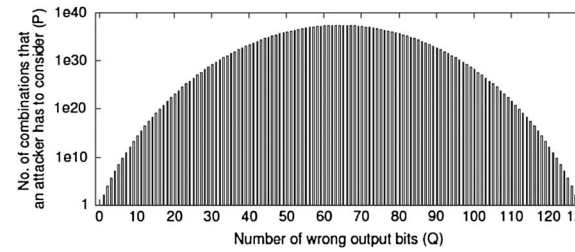

Fig. 3. Number of combinations (P) that an attacker has to explore when key size $\mathrm{N} = 128$ for different values of Q. A system with an encrypted module.

Fig. 3 plots the number of combinations (P) that an attacker has to consider for key size $N(= 128)$ for different values of Q. P is maximum when $\mathrm{Q} = 64$. Thus, the ambiguity for an attacker will be maximum when $\mathrm{HD(y,y')} = \mathrm{N}/2$. Hence, the logic encryption technique should insert key-gates such that the HD between the outputs on applying the correct key and the wrong key (HD(y,y')) is N/2, i.e., 50% of the output bits should be corrupted on applying a wrong key.

The proposed fault analysis approach enables a designer to have *control* over the corruption effects of a logic encryption technique. Certain designs may benefit from lower levels of HD (by corrupting certain targeted parts of the design, or targeted outputs). The proposed fault-analysis approach provides the control needed even in these situations, thus making the necessary key gate insertions to achieve the targeted corruption.

3 Motivational Example

Let us consider the combinational logic encryption technique proposed in [6], [7]. In this technique, XOR/XNOR gates are inserted at random locations. For instance, consider the C17 circuit shown in Fig. 4 encrypted with one XOR-gate, E1. This gate is inserted at the output of gate G2 which is part of the original design.

The design will produce the correct output on applying the correct key value, $\mathrm{K1} = 0$. On applying a wrong key ($\mathrm{K1} = 1$), wrong outputs are produced. For example, on applying the input pattern "01000", a wrong output "00" is produced instead of the correct output "10".

Unfortunately, the design produces correct outputs for certain input patterns even on applying a wrong key. For example, the input pattern "11100" produces the correct output "11" even with a wrong key applied. In fact, this design produces a wrong output only for twelve input patterns out of the possible 32 input patterns. In other words, the design produces correct outputs for 75% (24) of the input patterns despite applying the wrong key. Thus, this encryption procedure is weak as it does not ensure wrong outputs are produced for wrong keys, let alone 50% HD criterion. In this work, we propose a technique that will not only guarantee wrong outputs for wrong keys but also meets 50% HD criterion.

4 Fault Analysis-Based Logic Encryption

4.1 Logic Encryption: A Fault Analysis Perspective

We will now describe our technique to encrypt a design using key-gates (e.g., XOR/XNOR) in such a way that any wrong

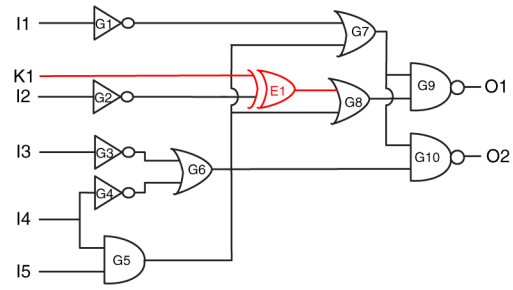

Fig. 4. A circuit encrypted with one XOR-gate (E1). The valid key is $\mathrm{K1} = 0$. However, the design produces correct outputs ($\mathrm{HD} = 0$) for 24 inputs patterns out of the 32 possible input patterns.

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 key-gates are used for encryption.

Consider the C17 circuit encrypted with one XOR gate (E1) as shown in Fig. 5(b). Here, E1 is the key-gate. If a wrong key ($\mathrm{K1} = 1$) is applied to the circuit, the value of net B is the negated value of net A. This is the same as exciting an s-a-0 (when $\mathrm{A} = 1$) or s-a-1 (when $\mathrm{A} = 0$) fault at the output of G7 as shown in Fig. 5(a). Please note that s-a-0 (s-a-1) fault activation can be attributed to the case where the net in question is supposed to yield a value of 1 (0) during the functional mode of operation.

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 [20].

Consider the circuit shown in Fig. 5(b). Let a wrong key ($\mathrm{K1} = 1$) be applied to the circuit. For the input pattern 00000, an s–a–0 fault gets excited at the output of E1 and propagates to both outputs. The value at the output of E1 is 0 instead of 1, and the output is 11 instead of 00.

For the input pattern 01110, even though the s–a–0 fault gets excited at the output of E1, the output is 11, which is the correct 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 key-gate and applying a wrong key is equivalent to exciting a single stuck-at fault. Likewise, inserting multiple key-gates and applying a wrong key is equivalent to simultaneously exciting multiple stuck-at faults.

However, when multiple faults are excited, they might mask one another. Therefore, in logic encryption, when multiple key-gates are inserted, the effect of one key-gate might mask the effect of other key-gates.

Consider the encrypted circuit shown in Fig. 5(c). When the key bits 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. Consequently, the design produces the correct output, 00. Similar to fault masking in IC testing, the effect of one XOR gate is masked by the effect of the other two XOR gates.

Even though the above scenario corresponds to masking the effects of faults (key-gates), the typical scenario in IC testing occurs when the effects of the same fault cancel due to re-convergent fan-out structures. Fault masking occurs despite the single fault assumption in IC testing.

Goal: Insert the key-gates such that a wrong key will affect 50% of the outputs for any input pattern. 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 on applying an input pattern.

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 be used to insert only one key-gate at a time. We overcome this challenge by using a greedy iterative approach where key-gates are inserted iteratively. In every iteration, the fault that has the potential of propagating to a maximum number of outputs dictates the location of the key-gate to be inserted. For every iteration (except the first), the key-gates inserted at previous iterations are provided with random wrong keys thereby emulating a multiple stuck-at fault scenario and accounting for all previous key-gate insertions. An algorithm is presented in the subsection 4.2.3 to perform this logic encryption.

4.2 Logic Encryption Using XOR/XNOR Gates

4.2.1 *Fault Impact*

To insert an XOR/XNOR as a key-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 fault impact defined by (1). From a set of test patterns, we compute the number of patterns that detect the s-a-0 fault (NoP_0) at the output of a gate Gx and the total number of output bits that get affected by that s-a-0 fault (NoO_0). Similarly, we compute NoP_1 and NoO_1 for s-a-1 faults.

$$\text{Fault Impact} = (\mathrm{NoP}_0 \cdot \mathrm{NoO}_0 + \mathrm{NoP}_1 \cdot \mathrm{NoO}_1). \quad (1)$$

By inserting an XOR/XNOR key-gate 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), indirectly enabling the logic encryption technique to reach the 50% Hamming distance metric.

Algorithm 1 Fault analysis-based insertion of XOR/XNOR gates

Input: Netlist, KeySize, EncryptionKey

Output: Encrypted netlist

// Location Selection Phase

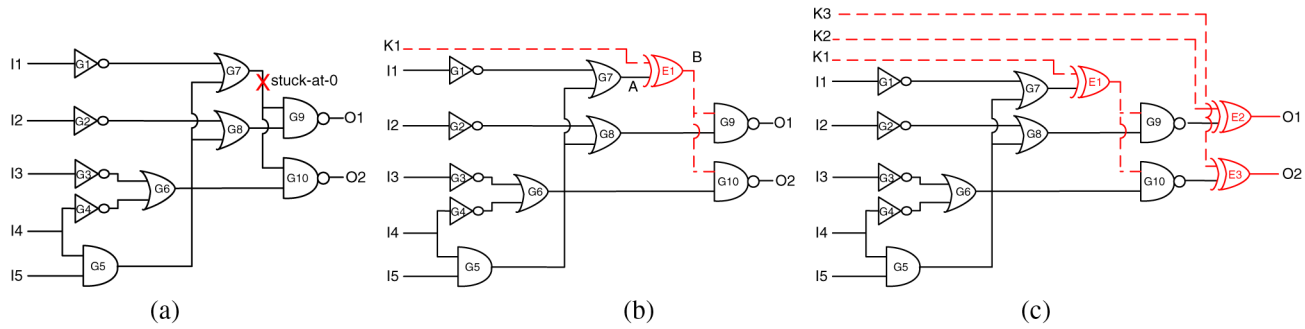

Fig. 5. Relation between logic encryption and IC testing: (a) fault excitation, (b) propagation, and (c) masking.

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
// Modification Phase
Generate R; //a random number
foreach bit i $\in$ R **do**
  if i == 1 **then**
    Insert an inverter at the o/p of corresponding key-gate;
  end
end
GateType = EncryptionKey $\oplus$ R;
foreach bit i $\in$ R **do**
  if i == 1 **then**
    Replace the XOR key-gate with an XNOR key-gate;
  end
end

4.2.2 User-Defined Key

A designer can use either an XOR or XNOR gate as a key-gate. However, an attacker can easily determine the value of the key-bit. The value of the correct key-bit is '0' in case of XOR gates and '1' in the case of XNOR gates. Hence, to deceive an attacker, a designer can add an inverter at the input or output of every key-gate. If the key-bit is '0', then the key-gate structure can be either 'XOR- gate' or 'XNOR $-$ gate $+$ inverter'. Similarly, if the key-bit is '1', then the key-gate structure can be either 'XNOR-gate' or 'XOR $-$ gate $+$ inverter'. The synthesis tools can bubble push[2] the inverters added for logic encryption. An attacker cannot identify which inverters are part of the original design and which are inserted for logic encryption.

4.2.3 Algorithm to Insert XOR/XNORs

Algorithm 1 greedily selects the best 'N' locations in a circuit to insert the XOR/XNOR key-gates. The algorithm has two parts—location selection phase and modification phase. In the location selection phase, the location with the highest fault impact is calculated and an XOR gate is inserted at that location. The algorithm considers the previously inserted XOR gates in this calculation. This phase terminates on inserting as many key-gates as that of the length of the user-defined encryption key.

In the modification phase, the inserted XOR gates are either retained or modified to XNOR gates. In addition, inverters are added to the output of randomly selected XOR/XNOR gates such that the user-defined encryption key correctly unlocks the design.

Consider encrypting the C17 circuit, shown in Fig. 5(a), using the above algorithm. The NoP_0, NoO_0, NoP_1, and NoO_1 values for different nodes in the circuit, on applying 1000 random input patterns, are listed in Table 1. In addition, the corresponding fault impact values calculated using (1) are also listed. Based on the fault impact value, gate G7 is selected as the best location to insert the key-gate, E1, for the first iteration. Similarly, for the subsequent iterations the corresponding XOR/XNOR gates are inserted using Algorithm 1.

4.3 Logic Encryption Using Multiplexers

4.3.1 Key Idea

In MUX-based encryption, MUXes are inserted such that one input of the MUX will be the true (original) wire in the design. The second input to the MUXr, referred as the false input, is another wire in the design. The select line of the MUX is the associated key bit. On applying the correct key bit, the true wire is selected, retaining the correct functionality of the design; otherwise, the functionality is modified by selecting the false wire.

The true wire can be connected to either the first or the second input of the MUX. This enables the possibility of the correct key bit (select line) to be either 0 or 1. This leads to the following dilemma for an attacker: is the true wire connected to the first or the second input of the MUX? While logic encryption using XOR/XNOR gates requires additional inverters to create the dilemma, MUXes create that dilemma inherently.

2. Techniques for bubble pushing are described in [24].

TABLE 1
Fault Impact Value of Different Nodes in C17 Circuit Shown in Fig. 5(a)

Node	NoP_0	NoO_0	NoP_1	NoO_1	Fault Impact
G1	193	274	150	216	85282
G2	88	88	79	79	13985
G3	89	89	85	85	15146
G4	89	89	55	55	10946
G5	304	473	85	114	153482
G6	89	89	195	195	45946
G7	193	274	267	391	157279
G8	88	88	196	196	46160
I1	150	216	193	274	85282
I2	79	79	88	88	13985
I3	85	85	89	89	15146
I4	142	199	131	173	50921
I5	112	147	85	114	26154
O1	196	196	304	304	130832
O2	195	195	305	305	131050

Similar to XOR-based encryption, MUX-based encryption can also be related to IC testing principles such as fault activation, fault propagation, and fault masking.

Fault activation: In MUXes, on applying a wrong key, the false wire will be selected instead of the true wire. However, the corruption effect will not happen when the two wires have identical values, preventing excitation. This is different from XOR-based encryption where fault activation is always guaranteed on applying a wrong key.

Consider the C17 circuit encrypted with one MUX (E1) as shown in Fig. 6. If a wrong key ($K1 = 1$) is applied, the value of net Y is the false value of the wire F instead of the true value of the wire T. For the input pattern 1X110, the values on T and F are both 0's. On applying a wrong key, a s-a-0 fault is excited at the output of G7. For the input pattern 1X100, the values on T and F are 0 and 1, respectively. On applying a wrong key, a s-a-1 fault is excited at the output of G7. However, for the input 1X110, the values on T and F are 0 and 0, respectively. On applying a wrong key, no fault is excited at the output of G7.

The effects of fault-propagation and fault-masking principles on MUX-based encryption are similar to that of XOR-based encryption. Hence, they are not repeated. However, the effect of fault activation in these two techniques is different; XOR-based encryption always guarantees fault activation whereas MUXes do not.

4.3.2 Fault Analysis-Based Insertion of MUXes

Similar to XOR-based encryption, MUXes can be inserted at the output of gates whose fault metric is the highest. The output of the selected gate will act as the *true* wire. However, a designer needs to carefully select the *false* wire. This is because fault excitation in MUX-based encryption will happen only if the value on the true wire is different from the value on the false wire.

While one can select the true and false wires based on the number of input patterns for which the value at those wires differ, such a method is computationally expensive. It requires a designer to apply all possible input patterns or at the least it requires $O(N^2)$ comparisons between the outputs of all the gates, where N is the number of gates in the design. Hence, we propose the following metric to select the false wire for MUX key-gate insertion.

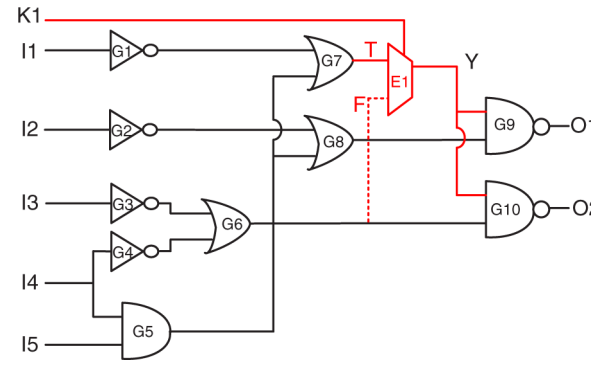

Fig. 6. A circuit encrypted with one multiplexer (E1). The false wire is shown as a dotted line. The logic value in the true and the false wires differ for 16 input patterns out of the total 32 input patterns.

While the true wire is selected based on the fault-impact metric, the false wire is selected based on another metric called *Contradiction Metric*.[3] This metric aims at maximizing the probabilities of having different values on the true and the false wires and is given as:

$$\text{Contradiction Metric} = (\mathrm{P}_{0,\text{true}} \times \mathrm{P}_{1,\text{false}}) + (\mathrm{P}_{1,\text{true}} \times \mathrm{P}_{0,\text{false}}), \quad (2)$$

where $\mathrm{P}_{0,\text{true}}$ and $\mathrm{P}_{1,\text{true}}$ are the probabilities of getting a 0 and 1 on the true wire, respectively. $\mathrm{P}_{0,\text{false}}$ and $\mathrm{P}_{1,\text{false}}$ are the probabilities of getting a 0 and 1 on the false wire, respectively. Computing these probabilities requires $O(N)$ computations for a design with N gates [20].

In addition to this metric, the designer should also select the false wire that will not result in a combinational loop with the true wire. This is because combinational loops are rare in a design except in the cases of flip-flops, latches, and ring oscillators. If a designer selects the false wire such that it forms a combinational loop with the true wire, an attacker can easily determine that the "feedback" wire is the false wire. Consequently, he can identify the corresponding key bit.

Consider the circuit shown in Fig. 6. Here, the output of gate G7 is chosen as the true wire and the output of gate G6 is chosen as the false wire. On an applying an input pattern 0X11X, the value of the true wire is '1', while the value of the false wire is '0'. In fact, the values of true and false wires differ for 16 input patterns.

4.3.3 Algorithm to Insert MUXes

Algorithm 2 greedily selects the best 'N' locations in a circuit to insert the MUXes. Similar to Algorithm 1, Algorithm 2 has two parts: the location selection phase and the modification phase. In the location selection phase, the location with the highest fault impact is calculated and it is selected as the true wire. Then, a list of wires that do not form a combinational loop with the true wire is formed. The contradiction metric of

3. To reduce delay overhead, one can constrain the algorithm to select the true and false wires from non-critical paths.

the wires within this list is calculated using (2). The gate with the highest contradiction metric is selected as the false wire. A MUX is then inserted at the true wire location. The true and false wires are connected to the first and second input of the MUX. In the modification phase, the input order of a MUX is rearranged depending upon its key bit.

Algorithm 2 Fault analysis-based insertion of MUXes

Input: Netlist, KeySize, EncryptionKey

Output: Encrypted netlist

// Location Selection Phase

for i $\leftarrow$ 1 to KeySize **do**

foreach gate j $\in$ Netlist **do**

Compute FaultImpact;

end

Select the gate with the highest FaultImpact as the *true* wire;

ListOfFalseWires $= \Phi$;

foreach wire j $\in$ Netlist and j $\neq$ true wire **do**

if CombinationLoop(j, true wire) == False **then**

ListOfFalseWires = ListOfFalseWires U *j;*

Compute Contradiction metric;

end

end

Select the wire with the highest contradiction metric as false wire

Insert MUX and update the Netlist;

Apply Test Patterns;

end

// Modification Phase

foreach bit i $\in$ EncryptionKey **do**

if i == 1 **then**

Connect *true* and *false* wires to the second and first inputs of the MUX;

end

end

Consider encrypting C17, shown in Fig. 5(a), using the above algorithm. The output of G7 is selected as the true wire based on the fault impact values shown in Table 1. Table 2 lists the contradiction metric of the wires that do not form a combinational loop with G7. The output of G5, which has the highest contradiction metric, is selected as the false wire. A MUX is then inserted at the output of G7. For the subsequent iterations the corresponding MUXes are inserted using Algorithm 2. Note that in Table 2, the best contradiction metric achieved for C17 is 0.57. This value is much less than the ideal value of.

TABLE 2
Contradiction Metric Values of Different Nodes in C17 Circuit on Selecting G7 as the True Wire

Node	$P_{0,\ false}$	$P_{1,\ false}$	Contradiction metric
I1	0.5	0.5	0.5
I2	0.5	0.5	0.5
I3	0.5	0.5	0.5
I4	0.5	0.5	0.5
I5	0.5	0.5	0.5
G1	0.5	0.5	0.5
G2	0.5	0.5	0.5
G3	0.5	0.5	0.5
G4	0.5	0.5	0.5
G5	0.27	0.75	0.57
G6	0.75	0.25	0.4375
G8	0.625	0.375	0.46875

4.4 Results

4.4.1 Experimental Setup

The effectiveness of the proposed technique is analyzed using ISCAS-85 combinational and ISCAS-89 sequential benchmarks. We analyzed the performance of the logic encryption techniques on OpenSPARC T1 Processors [25]. In case of processors, not all the modules need to be encrypted. Since the designer's valuable IP is typically in the controllers, one can encrypt only the controllers. A side benefit of encrypting controllers is that they are usually small ($< 1\%$) [3]. Consequently, the overhead due to encrypting the controllers will be negligible at the system level.

We used the HOPE fault simulation tool [10] 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 maximum 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 HD between the corresponding outputs.[4] The area, power, and delay overhead were obtained using Cadence RTL Compiler.

4.4.2 Hamming Distance Analysis

The fault analysis based approach is compared with the random insertion approach [6], [7] and the corresponding results are shown in Figs. 7 and 8. Let us analyze the performance of the encryption gates when they use XOR/XNOR gates (Fig. 7). Fig. 7(a) and (c) show the results of random insertion of XOR/XNOR gates in ISCAS and OpenSPARC designs, respectively. The results of fault analysis-based insertion of XOR/XNOR gates in ISCAS and OpenSPARC designs are shown in Fig. 7(b) and (d), respectively.

When the XOR/XNOR gates are randomly inserted [Fig. 7(a) and (c)], 50% HD is not achieved. Fault masking is the main reason for this poor performance. The effects of wrong keys are blocked for most of the input patterns as

4. In case of sequential designs, we not only corrupt the outputs but also the FSM state bits. Since the state values are stored in the flip-flops, we considered each flip-flop as a pseudo output to evaluate the HD.

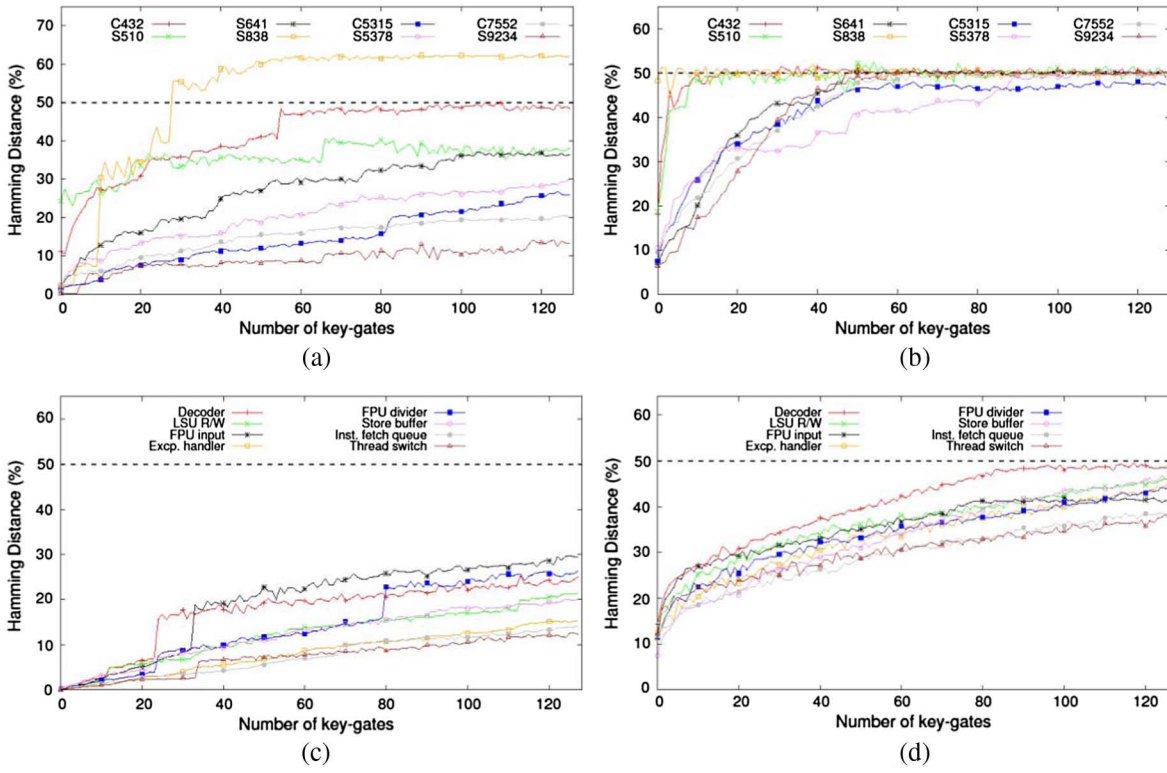

Fig. 7. Hamming distance between the outputs of designs on applying the correct key and a random wrong key: (a) Random insertion of XORs in ISCAS designs [6], [7], [11], (b) fault analysis-based insertion of XORs in ISCAS designs, (c) random insertion of XORs in OpenSPARC [6], [7], [11], and (d) fault analysis-based insertion of XORs in OpenSPARC units.

discussed in Section 4. However, fault analysis based insertion [Fig. 7(b) and (d)], achieves 50% HD for all benchmarks except for C5315, C7552, and OpenSPARC controllers. The number of outputs in benchmarks is very high (> 100) and hence it is hard to achieve 50% HD. However, the fault analysis based approach performs well in all the other benchmarks including sequential designs as it takes fault masking effects into account.

The slope of the lines in Figs. 7 and 8 indicates the performance of the random and the fault analysis based insertions. If the line is steeper, 50% HD can be achieved with a smaller number of additional key-gates; hence power, area, and 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 HD.

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

Let us analyze the performance of MUX-based encryption. Fig. 8(a) and (c) show the results of random insertion of MUXes in ISCAS and OpenSPARC designs, respectively. The results of fault analysis-based insertion of MUXes in ISCAS and OpenSPARC designs are shown in Fig. 8(b) and (d), respectively. In MUX-based encryption (Fig. 8), the fault is not always excited as it requires the logic value on the false wire to be different from that on the true wire. This is the main reason why MUX-based encryption is not able to achieve 50% HD. However, fault-analysis-based insertion [Fig. 8(b) and (d)] of MUXes still yields a better HD than random insertion of MUXes [Fig. 8(a) and (c)]. Unfortunately, a higher number of MUXes are required than are XOR gates to achieve the 50% HD mark.

Table 3 compares the best HD (the one that is close to 50%) and the number of key-gates required to achieve that HD between the random and fault analysis based logic encryptions. It can be seen that, in the case of XOR gates, on average, fault analysis based logic encryption achieves the 50% HD value with a key size that is four times less than that of random insertion. In the MUX based approach, on average, fault analysis based logic encryption achieves a HD value which is close to the 50% mark, while the random insertion technique achieves only 25% HD. This is because fault analysis based logic encryption identifies more effective locations to insert the gates than the random insertion based logic encryption. The performance of the XOR- and MUX-based insertions differ because XOR/XNOR gates always guarantee fault activations whereas MUX-based insertion cannot guarantee fault activation.

4.4.3 *Power, Area, and Delay Overhead*

Fig. 9 shows the power, delay, and area overhead of the benchmarks that are encrypted with the number of key-gates listed in Table 3 using random insertion [6], [7], [11] and the proposed fault-analysis based insertion.

In all cases shown, the following three trends are observed. First, the random insertion of key-gates (XORs and MUXes) takes more overhead to achieve their highest HD. Specifically, the fault analysis-based insertion of key-gates takes much less overhead when compared to their random insertion counterparts.

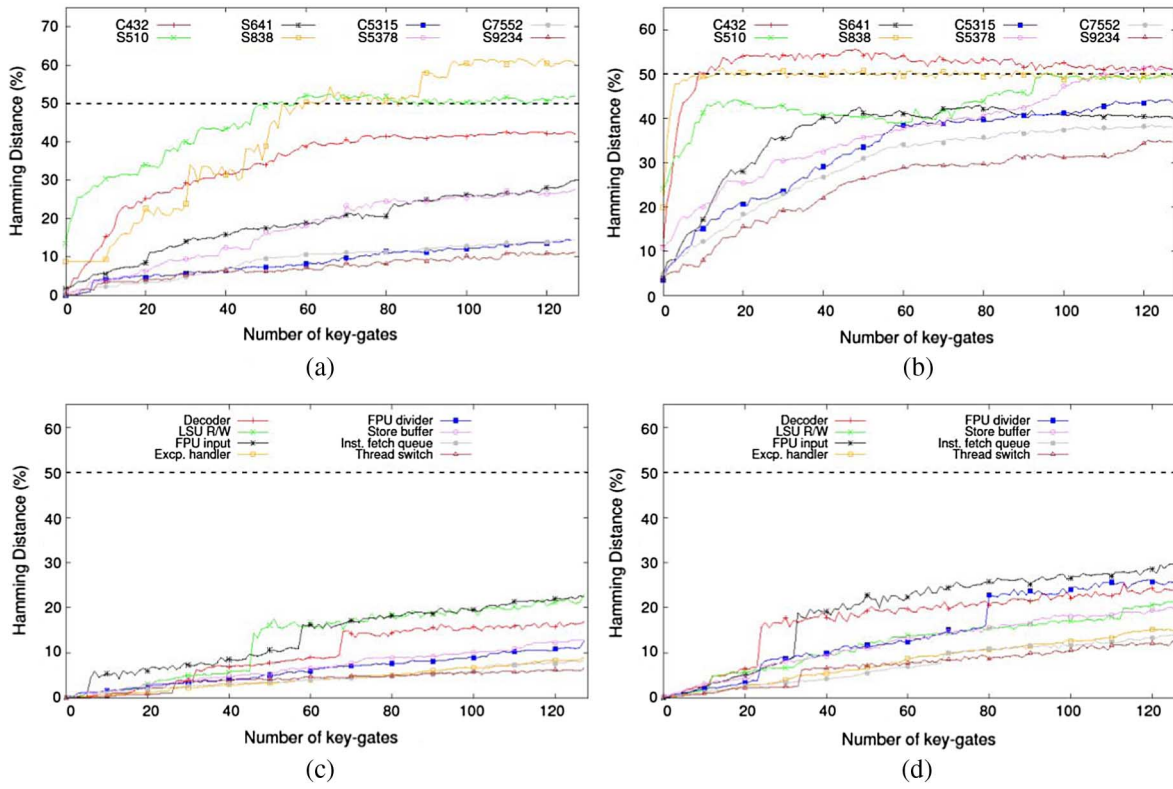

Fig. 8. Hamming distance between the outputs of designs on applying the correct key and a random invalid key: (a) Random insertion of MUXes in ISCAS designs, (b) fault analysis-based insertion of MUXes in ISCAS designs, (c) random insertion of MUXes in OpenSPARC units, and (d) fault analysis-based insertion of MUXes in OpenSPARC units.

Second, the overhead of the MUX-based method is more than that of the XOR-based method irrespective of the encryption technique. There are two reasons for this. The first is, as mentioned before, the MUX-based encryption technique takes more key-gates to achieve the target HD than the XOR-based encryption technique because of non-guaranteed fault activation. The second reason is the standard cell implementation of MUX in the 45 nm library uses NAND gate-based implementation of MUXes. This type of implementation consumes more power, increases delay and occupies more area. The power, area and delay overhead of MUXes would have been better if the standard cell library had a transmission gate-based MUX.

Finally, the overhead is very high for smaller designs (< 2000 gates) such as C432, S510, S641, and for some OpenSPARC units. This is because even a mere 30 additional XOR gates required for logic encryption is on the order of the total number of gates used to construct these small circuits. Conversely, in case of large designs (> 3000 gates), the overhead for fault analysis-based techniques (both XOR and MUX) is less than 5%. This highlights that the fault analysis-based encryption is highly feasible and does not cause much overhead especially for larger designs. Furthermore, since we are encrypting only the controllers, which occupy only a tiny part (1% [3]), the overhead will be negligible at system level.

5 Resource-Constrained Logic Encryption

In certain scenarios, a designer can only offer a limited power, area, or delay overhead for logic encryption. Hence, it is necessary to identify the security offered in relation to subsequent power, area, and delay overheads.

In constrained insertion, instead of inserting a pre-defined number of key-gates, the designer inserts key-gates until the encrypted design exceeds the allowed power, area, or delay limit. A designer can insert a key-gate at a location and calculate the power/area/delay overhead due to that key-gate. If the overhead is acceptable, he can insert the key-gate at that location. If not, he can skip that location. Such a method will be computationally expensive. Hence, we perform the resource-constrained logic encryption in the following way.

In every iteration, after inserting a key-gate, the designer calculates the overhead for encryption. If this overhead is within the allowable limit, then he inserts an additional key-gate and repeats the same process until the limit is exceeded. To perform this analysis, we follow the same insertion algorithms listed in Sections 4 and 5, and we stop the insertion once the overhead exceeds a pre-defined limit. As an instance, we chose a limit of 5%.

Fig. 10 shows the HD achieved and the number of key gates inserted for different insertion techniques when a designer is constrained to spend only 5% power, delay, and area overhead, respectively for logic encryption. One can make the following three observations.

First, for a given overhead, the HD achieved by the fault analysis-based method is typically higher than that of the random method. This is because the fault analysis-based insertion technique accounts for fault activation, propagation, and masking effects. In case of delay-constrained insertion [Fig. 10(b)], inserting a MUX increases the delay by more than 5% and thus, no MUX was inserted for some of the designs.

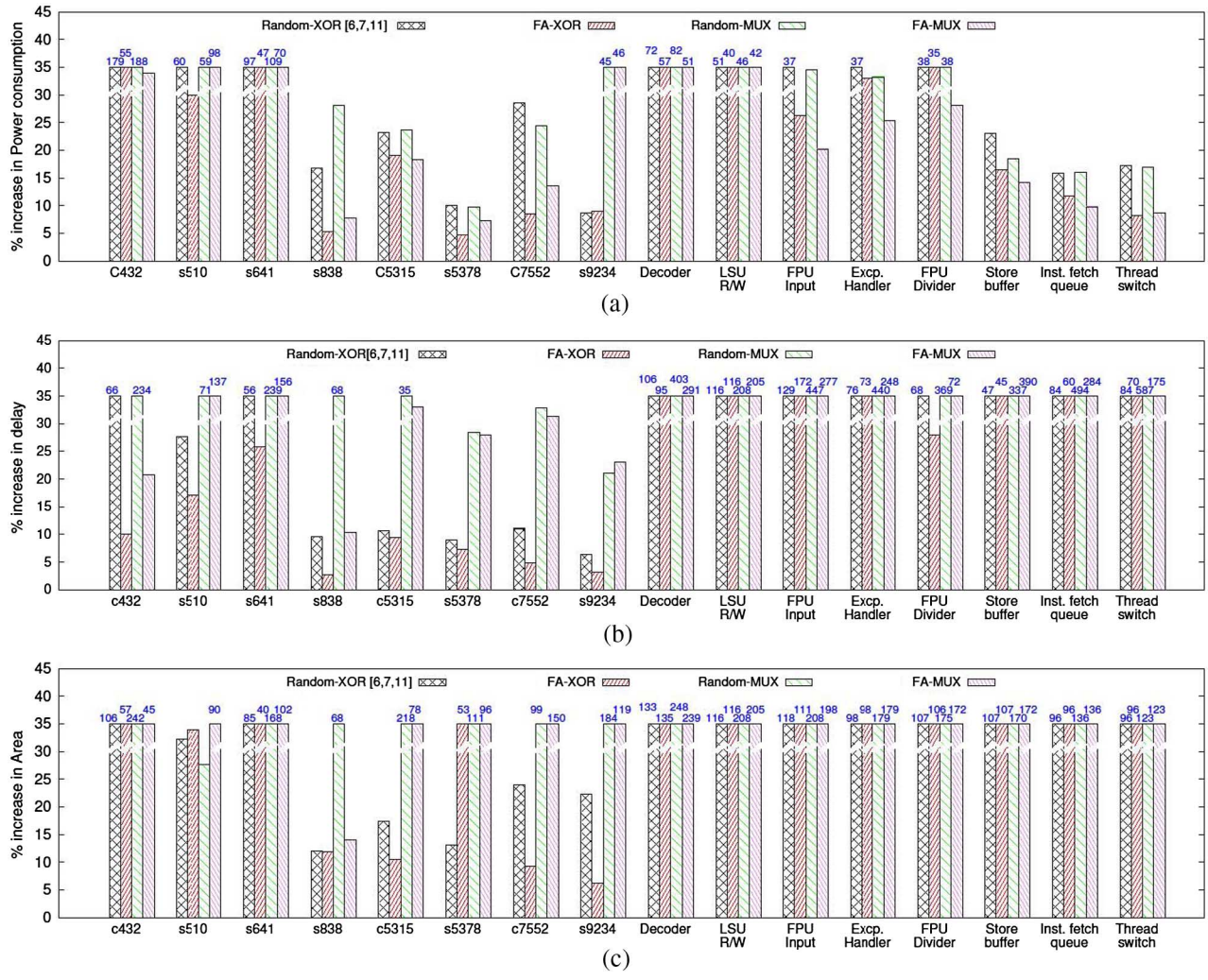

Fig. 9. Overhead for different logic encryption techniques for a key size that results in a HD close to 50%. (a) Power overhead, (b) delay overhead, and (c) area overhead.

TABLE 3
The Best Hamming Distance Achieved (Close to 50% Mark) and the Number of Key-Gates Required to Achieve that Distance for Different Logic Encryption Techniques

Logic Encryption	C432	S510	S641	S838	S5378	C5315	C7552	S9234
Random–XOR[6,7,11]	50/110	39/67	37/104	55/28	29/128	27/124	20/126	14/118
FA-XOR	50/16	50/42	50/29	50/2	50/106	48/109	50/55	50/39
Random-MUX	43/112	50/49	30/126	50/58	27/128	14/116	13/107	11/111
FA-MUX	50/9	50/46	50/46	50/26	50/110	43/109	38/102	43/92

Logic Encryption	Decoder	LSU R/W	FPU In	Excp. Handler	FPU Div.	Store buffer	Inst. Fetch queue	Thread switch
Random–XOR[6,7,11]	25/113	21/128	29/125	15/124	26/127	20/124	14/127	12/126
FA-XOR	49/117	46/128	41/100	44/125	44/124	46/127	39/126	38/125
Random-MUX	16/128	22/128	22/126	8/127	12/127	12/125	8/127	6/125
FA-MUX	49/119	35/124	48/117	39/127	44/124	43/127	33/117	36/126

Second, the HD achieved with XOR/XNOR gates as key-gates is higher than with MUXes as key-gates because XOR/XNOR gates always guarantee fault activation. In addition, since the power consumption, delay, and area of XOR/XNOR gates are smaller than that of MUX, a designer is able to insert a higher number of XOR/XNOR gates than MUXes for a given budget.

Finally, in the case of large designs a designer is able to insert more key-gates since the percentage overhead is proportional to the size of the design. Thus, for such designs the inserted key-gates were enough to achieve the 50% HD mark with the fault-analysis technique. On the other hand, in the cases of random insertions, 50% HD was not guaranteed. This shows that the power, delay or area overhead spent by a

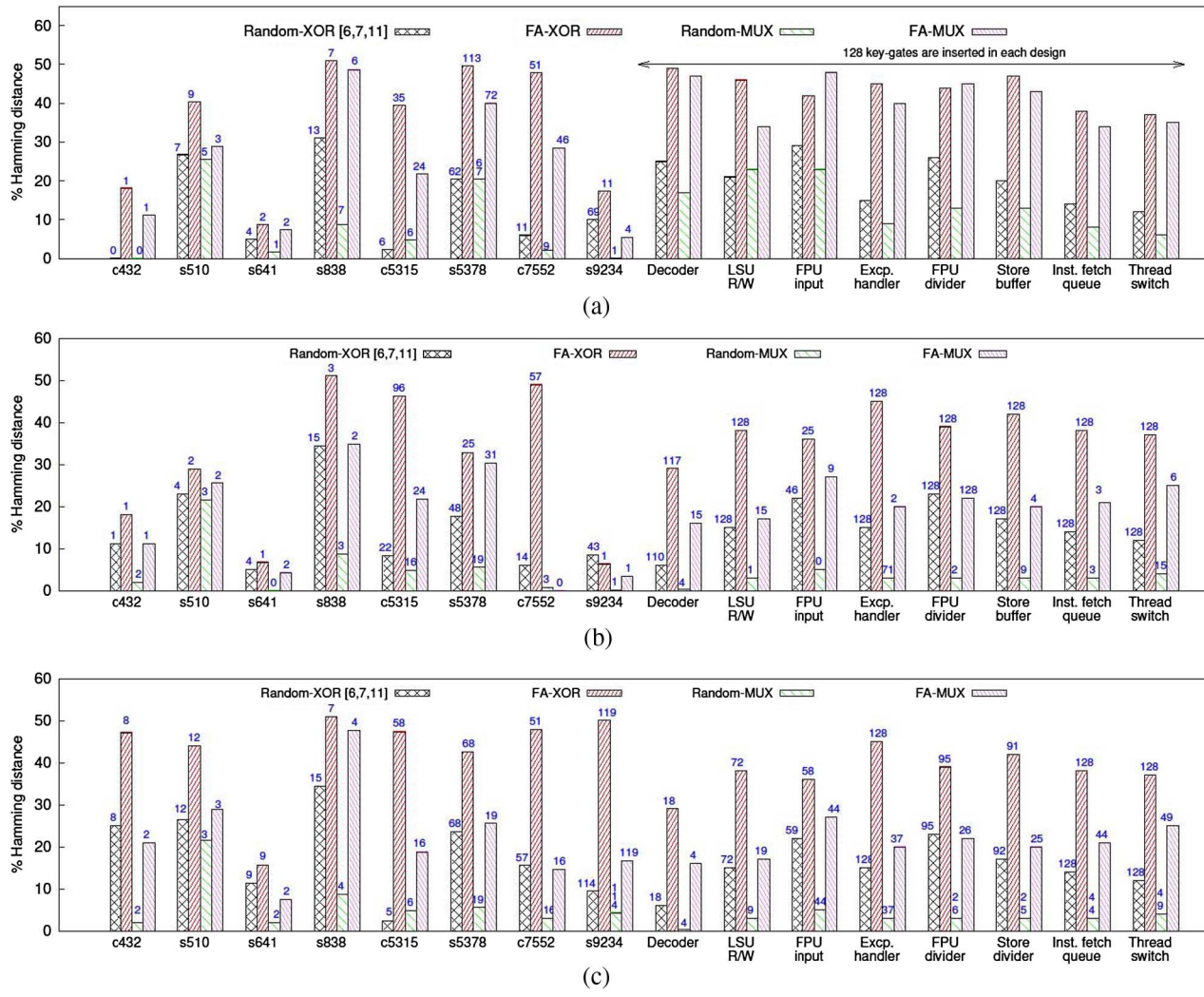

Fig. 10. Hamming distance achieved for different insertion mechanisms when the allowed overhead for logic encryption is 5%. (a) Power constrained, (b) delay constrained, and (c) area constrained. The numbers on top of the bar shows the number of key-gates inserted.

designer on logic encryption would be ineffective if he follows the random insertion technique. However, a designer can effectively reap the benefits of logic encryption on using a fault-analysis based insertion technique.

6 Discussion

6.1 Can We Insert Key-Gates Only at the Outputs?

A designer can insert key-gates only at the outputs to account for fault activation, propagation, and masking. However, in such insertions a key-gate will affect only one output-bit. The fault analysis-based insertion technique makes use of the fan-out structures to identify the best location within the circuit such that multiple outputs are affected by a single key-gate. Thus, each output-bit will not be directly correlated with a key-bit. Consequently, an attacker cannot determine the key-bit.

Fig. 11 shows the number of key-gates, which are required to achieve 50% HD that are inserted at the outputs and inside the design for different fault analysis-based insertion techniques. One can observe that the number of key-gates required to achieve 50% HD on using fault analysis-based insertion method is less than the number of outputs in the design. For example, consider the design C5315. This design has 123 outputs. However, fault analysis-based insertion of XOR/XNOR gates requires only 109 XOR/XNOR gates. Similarly, fault analysis-based insertion of MUXes requires only 109 gates. Let us consider another design C7552. This design has 108 outputs. However, fault analysis-based insertion requires 55 and 102 key-gates for the XOR and MUX approaches, respectively. This shows that a designer can find effective

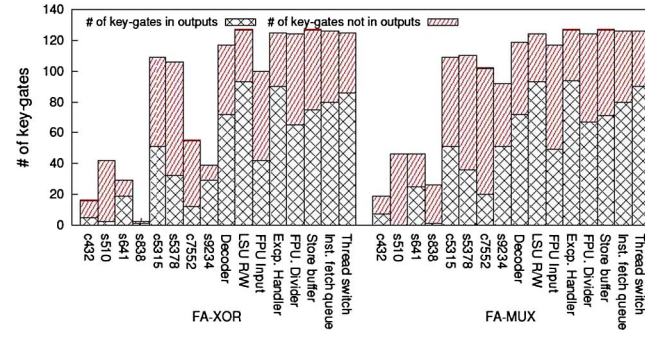

Fig. 11. Number of key-gates inserted at the outputs and inside the designs for different fault analysis-based insertion techniques.

TABLE 4
Number of Output Combinations that an Attacker Is Forced to Consider on an Encrypted Netlist for Various Types of Key-Gates and Logic Encryption Techniques (a) ISCAS circuits and (b) OpenSPARC controllers

(a)

Logic encryption	C432	S510	S641	S838	S5378	C5315	C7552	S9234
Random–XOR [6,7,11]	35	1.7E+03	2.7E+11	8.2E+08	5.3E+58	2.5E+30	4.9E+22	6.7E+42
FA-XOR	35	1.7E+03	1.1E+12	1.2E+09	2.3E+67	7.4E+35	2.5E+31	9.1E+73
Random-MUX	35	1.7E+03	3.7E+10	1.2E+09	5.2E+56	1.8E+21	8.8E+17	9.5E+36
FA-MUX	35	1.7E+03	1.1E+12	1.2E+09	2.3E+67	2.4E+35	1.7E+30	9.1E+72

(b)

Logic encryption	Decoder	LSU R/W	FPU In	Excp. Handler	FPU Div.	Store buffer	Inst. Fetch queue	Thread switch
Random–XOR[6,7,11]	1.1E+22	2.9E+44	3.0E+35	5.9E+32	2.5E+50	5.6E+34	6.4E+43	1.7E+41
FA-XOR	8.1E+26	7.7E+59	5.0E+39	5.4E+52	3.2E+60	1.7E+48	1.8E+72	9.6E+74
Random-MUX	7.8E+16	3.8E+45	8.8E+30	3.1E+21	1.1E+32	2.8E+25	2.4E+30	1.7E+25
FA-MUX	8.1E+26	1.9E+56	4.3E+40	2.7E+51	3.2E+60	5.9E+47	3.7E+68	7.3E+73

places inside the circuit, not just at the outputs, to insert key-gates such that the 50% HD metric is achieved.

6.2 Security Analysis

To undermine the security offered by logic encryption, an attacker can perform the following attacks:

1. **Brute-force attack**: In this attack, an attacker tries all possible key combinations until he finds the correct key. However, increasing the key size will make it harder for an attacker to retrieve the secret key.
2. **Correcting the wrong output bits**: In this attack, an attacker can correct the wrong output bits by inverting them. To perform this attack, he has to know which output bits are wrong. The number of output-bit combinations that the attacker has to consider for every input combination dictates the ambiguity created for the attacker. This is analogous to traditional cryptography, where the key size dictates the ambiguity for an attacker. Table 4 shows the number of output combinations that an attacker is forced to consider on an encrypted netlist for different logic encryption techniques. It can be seen that fault-analysis based logic encryption results in more ambiguity for an attacker than random insertion.
3. **Key-gate removal attack:** The attacker can attempt to remove the key-gates from the encrypted netlist and replace them randomly with a buffer or an inverter. Consider the following case from an attacker's perspective. An "$\mathrm{XOR}-\mathrm{gate}+\mathrm{inverter}$" is inserted into the design for a key-bit of value '1'. On seeing this XOR-gate connected to a key input, an attacker will recognize that the XOR-gate is added for logic encryption. However, on seeing the inverter, he might not know whether it is part of the original design (resulting in correct key-bit of value '0') or added for logic encryption (resulting in correct key-bit of value '1'). Thus, this creates a dilemma to an attacker. This dilemma is further exacerbated by the synthesis tool as it also uses inverters (not for the purpose of logic encryption) while synthesizing a design. In case of MUX-based encryption, the attacker can try to connect the true wire with the output. But, he does not know which wire is the true wire.
4. **Retrieving good input-output pairs:** An attacker can buy a working IC from the market. Thereby, he will have access to good input-output pairs of the IC. However, this does not undermine the strength of the proposed logic encryption technique. This is because of the following reason. In the logic encryption, the fault impact is calculated by considering not only the primary inputs and outputs but also considering each flip-flop is considered as a pseudo input and as pseudo output. Thus, in the context of logic encryption "good input-output pairs" include: (1) primary input-primary output pairs, (2) primary input-pseudo output pairs, (3) pseudo input-primary output pairs, and (4) pseudo input-pseudo output pairs. After manufacture-testing and validation, the designer disables the scan test access port of the IC [28]. Disabling the scan access port prevents the access to the flip-flops. Consequently, an attacker cannot access the pseudo-inputs and pseudo-outputs. Thus, an attacker on buying a functional IC does not have access to primary input-pseudo output pairs, pseudo input-primary output pairs, and pseudo input-pseudo output pairs.

6.3 Logic Encryption and PUFs

The security of any logic encryption technique can be improved by using Physical Unclonable Functions (PUFs) by assigning unique unlocking keys to each instance of an IC [6], [7], [11]. PUFs are circuits that exploit inherent physical disorders due to process variations to produce a chip-dependent output on applying an input. The technique works as follows: The designer embeds a symmetric key cryptographic module and a PUF circuit along with the encrypted module. Post-fabrication, the designer applies an input to the PUF and obtains an output. Simultaneously, the output of the PUF is fed into the cryptographic module. This output is used as the key for the cryptographic module. This key is burnt into the fuses and thus remains non-volatile. Once the designer obtains the output from the PUF, the PUF is no longer accessible (one can use fuses to shut down the access).

The designer uses the output of the PUF as the key to cryptographic module. He uses the secret key to unlock the

encrypted module as the plaintext to the cryptographic algorithm and computes the ciphertext. This ciphertext is then fed into the target chip. Inside the target chip, the ciphertext is fed into the cryptographic module which uses the PUF's output as the secret key. The output of the cryptographic module will then be the secret key to unlock the encrypted design. This is fed to unlock the encrypted module. Since the PUF's output will be different on different chips, the ciphertext of one chip cannot be used to unlock the design in another chip.

6.4 Can a Logic Encryption Technique Produce Wrong Outputs for Only Certain Inputs?

If a module produces wrong outputs for a few input patterns, an attacker can still use the module by excluding those input patterns. For example, if a processor produces wrong outputs for just 2-3 instructions, an attacker can recompile his program by excluding those instructions from the instruction set. Thus, as highlighted in [11], it becomes necessary to produce wrong outputs for many input patterns for a random, wrong key.

In general, a designer cannot tune a logic encryption to produce wrong outputs only for certain inputs by assuming that the attacker will use only those inputs. This is because he does not have access to the attacker's input-set. Thus, a defender has to perform logic encryption based on generic input patterns.

6.5 Limitations

We generated random input patterns to calculate the fault impact of a node in a design. Although this does not cover the entire input space, it gives a designer a rough estimate of the impact of the fault at that node. However, one can also develop a systematic algorithm to calculate the fault impact rather than applying random input patterns by using the proposed fault metric as a basis.

Fault impact metric is only a heuristic and does not guarantee one to achieve 50% HD. However, it enables to reach 50% HD as can be seen in Figs. 7 and 8. Unlike cryptographic modules, the designs in the benchmark suite and in Open SPARC processor do not have a regular structure. Thus, one cannot guarantee 50% HD because of the fault masking effects described in Section 4.1.

In this work, only one key-gate is inserted per iteration. Such insertion may be computationally expensive for large designs. This method took two hours to encrypt the C7552 circuit. However, one can partition the circuit into multiple segments and encrypt each of the segments individually to achieve the global objective of 50% HD.

One can ensure that brute force effort is required to retrieve the secret key by formally proving the capability of the logic encryption technique. However, such a proof will be design dependent. Generating such formal proofs for every design may not be practical. The proposed techniques are design independent and increase the effort for an attacker.

7 Related 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 [3], [11], [12]. 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 [6], [7]. Usually, one of the inputs in these inserted gates serves as a 'key input' which is a newly added primary input. One can configure these gates as buffers or inverters using these key inputs.

CLIP introduces process variation sensors into a circuit [5]. Post-fabrication, special test vectors are applied to these sensors to determine the impact of process variation. Based on this impact, the designer configures these sensors such that correct outputs are produced. A wrong configuration results in a wrong output. The advantage of this technique is that every chip inherently has a unique decryption key. However, the maximum HD between the outputs of the correct and wrong configurations achieved by this technique is only 18%.

At the micro-architectural level, processor encryption uses the logic encryption capabilities to selectively encrypt units of a microprocessor [27], thereby enhancing the capabilities of a Trojan detection technique to detect Trojans. It randomly inserts the key-gates into the design, i.e. it uses the algorithm proposed in [6], [7]. It dynamically loads and unloads the key to make a unit to function at will. The proposed techniques can aid processor encryption to insert key-gates within a microprocessor unit such that an incorrect key results in an incorrect output.

Apart from sequential and combinational elements, memory elements are also inserted into the design [8]. The circuit will function correctly only when these elements are configured/programmed correctly. However, the introduction of memory elements into the circuit will incur significant performance overhead.

Techniques such as watermarking and passive metering are also proposed to detect IC piracy. In watermarking techniques, a designer embodies his/her signature into the design [13]. During litigation, the designer can reveal the watermark and claim ownership of the IC/IP. Watermarks are constructed by adding additional states to the finite state machine of the design, adding secret constraints during high-level [14], logical, or physical synthesis [15] steps.

In passive metering techniques, a unique device ID for every IC is formed leveraging process variations [3]. Physical Unclonable functions are leveraged to produce such IDs [17]–[19]. If a user pirates an IC, he/she will be caught by tracking the device ID. Unfortunately, both watermarking and passive metering techniques can only detect piracy but not prevent it; only logic encryption techniques can prevent IC piracy.

8 Conclusion

Fault analysis based logic encryption achieves 50% HD between the correct and the corresponding wrong outputs when an invalid key is applied to the design. While we used only one of the cryptographic criteria namely, HD, there are other criteria such as Avalanche criterion, Strict Avalanche criterion, and Bit independent criterion [20]. Cryptographically

strong designs (for instance, S-boxes, the primitives of Advance Encryption Standard [21]) have to satisfy all these criteria. Evaluation of a logic design against these criteria is computationally complex as it requires applying all possible input combinations. Thus, these criterion cannot be directly applied to logic design where applying all possible input patterns will be computationally inhibitive. To overcome this problem, cryptographic researchers have to develop new techniques to evaluate the security of logic encryption.

In this work, we took the average HD as the assessment criterion. To overcome the problems of averaging, one can perform insertion by assigning weights based on 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.

Acknowledgment

Received and cleared for public release by AFRL on November 19, 2012, case number 88ABW-2012-6072. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL or its contractors.

References

[1] KPMG. (2006). *Managing the risks of counterfeiting in the information technology* [Online]. Available: www.agmaglobal.org/press_events/press_docs/Counterfeit_WhitePaper_Final.pdf

[2] SEMI. (2008). *Innovation is at risk as semiconductor equipment and materials industry loses up to $4 billion annually due to IP infringement* [Online]. Available: www.semi.org/en/Press/P043775

[3] Y. M. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Proc. USENIX Secur.*, 2007, pp. 291–306.

[4] Department of Defense. (2005). *Defense Science Board (DSB) study on High Performance Microchip Supply* [Online]. Available: http://www.aoq.osd.mil/dsb/reports/ADA435563.pdf

[5] W. P. Griffin, A. Raghunathan, and K. Roy, "CLIP: Circuit level IC protection through direct injection of process variations," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 5, pp. 791–803, 2012.

[6] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Des. Autom. Test Eur.*, 2008, pp. 1069–1074.

[7] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[8] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 66–75, 2010.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. Des. Autom. Conf.*, 2012, pp. 83–89.

[10] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 15, no. 9, pp. 1048–1058, 1996.

[11] R. S. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, 2009.

[12] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," in *Proc. IEEE Int. Conf. Comput. Aided Des.*, 2009, pp. 113–116.

[13] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Proc. IEEE/ACM Des. Autom. Conf.*, 1998, pp. 776–781.

[14] F. Koushanfar, I. Hong, and M. Potkonjak, "Behavioral synthesis techniques for intellectual property protection," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 3, pp. 523–545, 2005.

[15] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Robust IP watermarking methodologies for physical design," in *Proc. Des. Autom. Conf.*, 1998, pp. 782–787.

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

[17] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. IEEE/ACM Des. Autom. Conf.*, 2007, pp. 9–14.

[18] J. W. Lee *et al.*, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *Proc Symp. VLSI Circuits*, 2004, pp. 176–179.

[19] K. Lofstrom, W. R. Daasch, and D. Taylor, "IC identification circuit using device mismatch," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2000, pp. 372–373.

[20] M. Bushnell and V. Agarwal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Boston, MA, USA: Kluwer, 2000.

[21] H. M. Heys and S. E. Tavares, "Avalanche characteristics of substitution-permutation encryption networks," *IEEE Trans. Comput.*, vol. 44, no. 9, pp. 1131–1139, 1995.

[22] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Crypto*, 1999, pp. 388–397.

[23] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proc. Des. Autom. Test Eur.*, 2004, pp. 246–251.

[24] D.M. Harris and S.L. Harris, *Digital Design and Computer Architecture*. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[25] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," *J. ACM*, vol. 2, no. 6, pp. 1–48, 2012.

[26] Sun Microsystems. (2006). *OpenSPARC T1 processor* [Online]. Available: http://www.opensparc.net/opensparc-t1/index.html

[27] J. Rajendran, A. K. Kanuparthi, M. Zahran, S. K. Adepalli, G. Ormazabal, and R. Karri, "Securing processors against insider attacks: A circuit-microarchitecture co-design approach," *IEEE Des. Test Mag.*, vol. 30, no. 2, pp. 35–44, 2013.

[28] ARM. (2010). *i.MX35 applications processors for industrial and consumer products* [Online]. Available: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX357&fpsp=1&tab=Documentation_Tab

Jeyavijayan Rajendran is a fourth year PhD candidate in the Electrical and Computer Engineering Department, Polytechnic Institute of New York University, Brooklyn. His research interests include hardware security and emerging technologies. He is a student member of ACM.

Huan Zhang received the BS and MS degrees in computer engineering from the Polytechnic Institute of New York University, Brooklyn, in 2011 and 2013, respectively. He is currently a researcher at a federally funded research and development center.

Chi Zhang is a senior year undergraduate in the Electrical and Computer Engineering Department, Polytechnic Institute of New York University, Brooklyn. His research interests include hardware security and trust.

Garrett S. Rose received the BS degree in computer engineering from Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, in 2001, and the MS and PhD degrees in electrical engineering from the University of Virginia, Charlottesville, in 2003 and 2006, respectively. Currently, he is with the Air Force Research Laboratory, Information Directorate, Rome, NY, where his work is focused on nanoelectronic computing research.

Youngok Pino received the BS degree in computer science from the City University of New York, City College, and the MS and PhD degrees in electrical engineering from Rensselaer Polytechnic Institute, Troy. She is a computer scientist at Information Sciences Institute (ISI), where her research is focused on secure hardware design for trust in different platforms such as FPGAs and ASICs. She has served on program committees for several conferences.

Ozgur Sinanoglu received the PhD degree in computer science and engineering from the University of California, San Diego, CA, in 2004. Currently, he is an assistant professor at New York University, Abu Dhabi, UAE. His main research interests include the reliability and security of integrated circuits, mostly focusing on design for testability and design for trust. He is the recipient of the Best Paper Award of the VLSI Test Symposium, 2011.

Ramesh Karri received the PhD degree in computer science and engineering from the University of California, San Diego, CA. He is a professor of electrical and computer engineering at Polytechnic Institute of New York University, Brooklyn. His research interests include trustworthy ICs and processors, high assurance nanoscale IC architectures and systems, VLSI design and test, and interaction between security and reliability. He is the recipient of NSF CAREER Award and Sir Alexander Von Humboldt Award.