

Chapter 3

Logic Encryption

Jeyavijayan (JV) Rajendran and Siddharth Garg

3.1 Introduction

Logic encryption¹ hides the functionality and the implementation of a design by inserting additional gates into the original design [4–6]. In order for the design to exhibit its correct functionality (i.e., produce correct outputs), a valid key has to be applied to the encrypted design. The gates inserted for encryption are the *key-gates*. Upon applying a wrong key, the encrypted design will exhibit a wrong functionality (i.e., produce wrong outputs).

Example. Consider the circuit shown in Fig. 3.1 which is encrypted using key-gates $K1$ and $K2$. The inputs $I1$ – $I6$ are the functional inputs, and $K1$ and $K2$ are the *key-inputs* connected to the key-gates. On applying the correct values of the keys ($K1 = 0$ and $K2 = 1$), the design will produce a correct output; otherwise, it will produce a wrong output.

EPIC [1] incorporates logic encryption into the IC design flow, as shown in Fig. 3.2. In the untrusted design regime, the IC is encrypted, and its functionality is not revealed. Post-fabrication, the IP vendor activates the encrypted design by applying the valid key. The key is stored in a tamper-evident memory inside the design to prevent access to an attacker.

Logic encryption prevents attacks such as piracy and hardware Trojans. Since the design is encrypted by the designer, the foundry cannot use any copies or overproduce

¹Researchers have previously used the terms “logic obfuscation” [1, 2] and “logic locking” [3] for this purpose.

J. Rajendran (✉)
Department of Electrical Engineering, The University of Texas at Dallas,
800 W Campbell Road, Richardson, TX 75080, USA
e-mail: jv.ee@utdallas.edu

S. Garg
New York University, New York, NY, USA
e-mail: siddharth.garg@nyu.edu

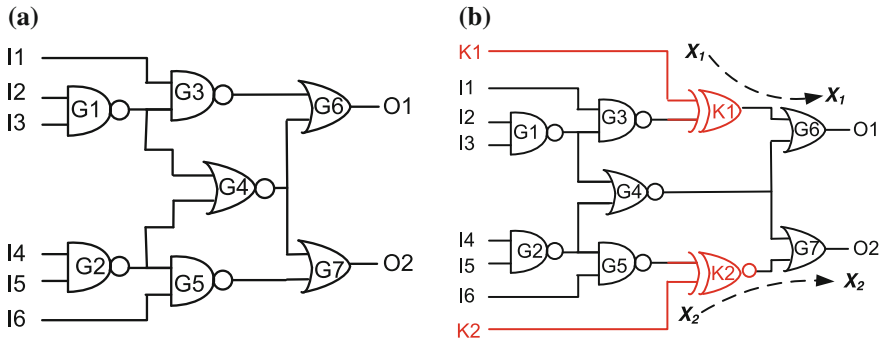


Fig. 3.1 a Original circuit. b A circuit encrypted using two key-gates K1 and K2 based on the technique proposed in [1]. By applying the input pattern 100000, an attacker can sensitize key-bits K1 and K2 to the outputs O1 and O2

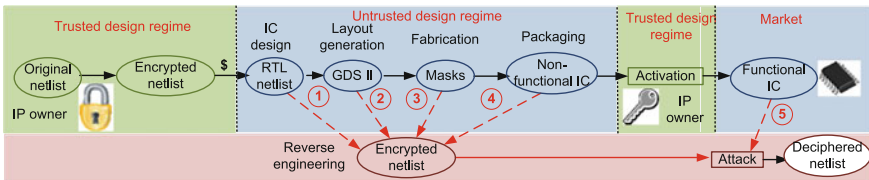


Fig. 3.2 The top blue box represents the EPIC design flow [1]. The design is in the encrypted form in the untrusted design regime. In the untrusted regime, an attacker can obtain the encrypted netlist from (1) the IC design or by reverse engineering the (2) layout, (3) mask, or (4) a fabricated IC, and (5) the functional IC from the market. Using this attack, the attacker can get a deciphered netlist and make pirated copies

ICs without the secret keys. Furthermore, it prevents an attacker from analyzing the structural behavior of the design, thereby hindering Trojan insertion.

Outline of the chapter. Section 3.2 explains the protocol on how logic encryption can be used in an IC supply chain, and Sect. 3.3 defines the threat model for logic encryption, listing out the capabilities of the attackers and their limitations. Section 3.4 lists the different security properties and metrics for logic encryption. Since its introduction in 2008, several attacks have been introduced against logic encryption. Section 3.5 details a set of attacks that enables an attacker to learn the correct outputs of the design, even when the design is subjected to logic encryption. In another set of attacks explained in Sect. 3.6, an attacker can learn the correct key used for logic encryption, by observing input–output pairs. Additionally, this section also explains the set of countermeasures against this class of attacks. Recently, new vulnerabilities due to an untrusted test facility have undermined the security of logic encryption. Unlike the previous attacks, this class of attacks relies only on the test patterns and responses. Section 3.7 details this class of attacks. Finally, there are several

techniques that rely on the security offered by logic encryption, which are explained in Sect. 3.8. Section 3.9 concludes this chapter by comparing different attacks and their countermeasures.

3.2 Protocol

The protocol for logic encryption is as follows [1].

Step 1: The designer encrypts the design with a common key, *CK*. This is the key shown in Fig. 3.2. On applying *CK*, the encrypted design produces correct outputs. The target IC is then designed along with the encrypted design, a public key cryptographic algorithm (e.g., RSA) and an on-chip random number generator. The IC also has a master public key (Master-Pub). Master-Pub’s private pair is master private key (Master-Pri), which is not stored on-chip.

Step 2: The designer sends this design to the untrusted foundry, where the chip is manufactured. The manufactured chip is then sent to the designer.

Step 3: The designer activates the on-chip random number generator to generate a public–private key pair, *RCK-Pub* and *RCK-Pri*, respectively. *RCK-Pub* is known to everyone. The designer encrypts *CK* with *MK-Pri* and *RCK-Pub*. The resultant ciphertext is called input key (*IK*).

Step 4: *IK* is given to the user of this chip. The user applies *IK* to the chip to activate it. The public key cryptographic module within the chip decrypts *IK* with *MK-Pub* and *RCK-Pri* to obtain *CK*. The on-chip infrastructure applies *CK* to the encrypted design and makes the IC functional. This process is called activation. Since *IK* is encrypted with public–private key pairs, it does not reveal *CK*. Thus, the user may not be able to unlock (i.e., reverse engineer) the design, even if he is able to unlock his chip (i.e., make the IC functional).

Identifying *MK-Pri* and *RCK-Pub* will help an attacker to overproduce the ICs. An attacker does not need design to a new mask. He can reuse the existing mask and unlock them by determining their *IKs* using *MK-Pri* and *RCK-Pub*. Identifying *CK* will help an attacker to pirate the design and identify “safe” places in a design to insert Trojans². An attacker can extract the protected design (without the public key cryptographic algorithm and on-chip random number generator), unlock it using *CK*, create a new mask, and manufacture the pirated ICs. For the rest of this chapter, we will consider how an attacker can extract *CK*. We refer to *CK* as the “key.”

3.3 Threat Model

The attacker can be either in the foundry or be the end user. The objective of the attacker is to determine the secret keys used for logic encryption. By determining the keys, he/she can decipher the functional netlist, make pirated copies, and sell

²Safe places in a design in the context of hardware Trojans refer to circuit nodes with low observability, low controllability, minimal impact on power and delay [7, 8]. places to insert Trojans.

them illegally, thereby defeating the purpose of logic encryption. Furthermore, with the knowledge of the keys, he/she can analyze the structural behavior of the design, thereby inserting Trojans at “safe places.”

The attacker needs the encrypted netlist and a functional IC. He/she can obtain the encrypted netlist from (1) the IC design, or by reverse engineering the (2) layout, (3) mask, or (4) a manufactured IC as shown in Fig. 3.2. The functional IC, (5) in Fig. 3.2, is bought in the open market.

3.4 Security Properties and Metrics

In the rest of this chapter, we focus on security of logic encryption assuming that the common key (CK) is the most critical security asset. Indeed, if CK is compromised, the attacker learns the IC’s intended functionality, thus compromising the designer’s IP and opening the door to hardware Trojan insertion. Furthermore, we note that even though the EPIC protocol does protect against overbuilding even if CK is compromised (each chip is activated only after the designer supplies the chip-specific IK), a determined attacker can still overbuild ICs using a new mask in which CK is hardwired to its compromised value. Such an attack is easily within the range of capabilities of a foundry attacker.

To thwart these attacks, a logic encryption technique has to satisfy the following properties:

1. **Correctness.** A logic encryption technique should produce a correct output upon applying the correct key. If it produces an incorrect output upon applying the correct key, the encrypted IC/design will violate the design specification, and the design will be considered “defective.”
2. **Resilient against output-guessing attacks.** A logic encryption technique should prevent an attacker from guessing the correct outputs from previously observed input–output pairs. To thwart such attacks, the output entropy upon applying the wrong key should be maximized. In other words, the Hamming distance between the outputs of the design upon applying the incorrect key should be 50%, as this value maximizes the entropy [4].
3. **Entanglement.** One should not be able to remove the key-gates from the protected circuit. Otherwise, an attacker can remove the key-gates, analyze the unprotected components, and obtain the original design.
4. **Resilient against key-guessing attacks.** A logic encryption technique should prevent an attacker from guessing the correct key value from previously observed input–output pairs. To thwart such attacks, key-gates should be inserted such that the number of input–output pairs required to obtain the key value is exponential to the size of the key.
5. **Overhead.** The logic encryption technique should aim to minimize area, delay, and power overheads, but not at the expense of the security objectives listed above.

3.5 Thwarting Output-Guessing Attacks

In logic encryption, different combinational logic elements are inserted in a circuit to conceal the functionality of a design. These elements can be XOR/XNOR gates [1, 4, 5, 9], AND/OR gates [6], MUXes [3, 4], or a combination of these elements [10]. One of the inputs to these gates serves as a key-input, which is a newly added signal driven by a tamper-evident on-chip memory. Unless the correct key is loaded onto the on-chip memory, a design will not work correctly. The activation of an encrypted IC can be conducted either prior to or after the manufacturing test. Secure communication infrastructure is needed if the keys are to be loaded remotely onto the chip [9, 11].

EPIC [9] is a logic encryption framework which inserts XOR/XNOR gates, referred as key-gates, such that these gates have minimal impact on circuit delay. One can configure these gates as buffers or inverters using these key-inputs. The insertion of the gates is done after logic synthesis and before physical synthesis. The design can then be resynthesized. If the key-gates were left as such without any other modifications to the circuit, the key-bits could be extracted by inspecting if a key-gate is XOR or XNOR. To eradicate such a simple deduction analysis of the key-gate types and the key values, the netlist can be synthesized such that the XOR/XNOR key-gates are replaced with other gates like AND/OR/NAND, or the inverters in the design can be moved around to change the polarity of the key-gates.

Since EPIC inserts the key-gates based on only delay overhead as a constraint, there is no guarantee that an incorrect key produces incorrect output for all the input patterns. Rajendran et al. [4, 5] developed a method, based on the principles of VLSI testing, that inserts XOR/XNOR gates or MUXes to achieve controllable corruption of the output bits. This technique maximizes the Hamming distance between the correct output and the incorrect outputs on applying a random incorrect key.

Testing principle-based insertion of key-gates Key-gates should be inserted 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. These observations are used 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 a stuck-at-1 (s-a-1) fault will get excited when key-gates are used for encryption.

Consider the C17 circuit (from the ISCAS'85 benchmark set) encrypted with one XOR gate (E1) as shown in Fig. 3.3(b). Here, E1 is the key-gate. 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 the same as exciting an s-a-0 (when $A = 1$) or an s-a-1 (when $A = 0$) fault at the output of G7 as shown in Fig. 3.3(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

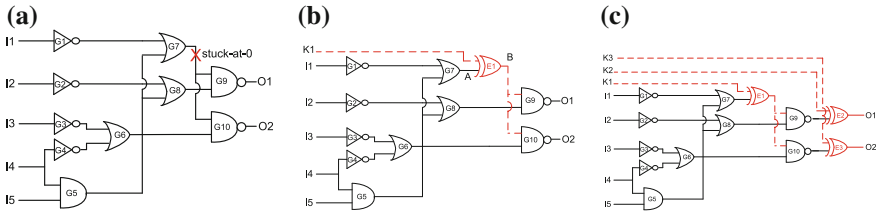


Fig. 3.3 Relation between logic encryption and IC testing: **a** fault excitation, **b** propagation, and **c** masking

the scenario where not all input patterns can propagate the effect of a fault to the output [12].

Consider the circuit shown in Fig. 3.3(b). Let a wrong key ($K1 = 1$) 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 are equivalent to exciting a single stuck-at fault. Likewise, inserting multiple key-gates and applying a wrong key are 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. 3.3(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.

Fault impact. To insert an XOR/XNOR as a key-gate, one needs 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, one uses fault impact defined by Eq. 3.1. From a set of test patterns, one can compute the number of patterns that detect the s-a-0 fault (NoP_0) at the output of a gate G_x and the total number of output bits that get affected by that s-a-0 fault (NoO_0). Similarly, NoP_1 and NoO_1 for s-a-1 faults are computed.

$$\text{Fault impact} = (NoP_0 \times NoO_0) + (NoP_1 \times NoO_1) \quad (3.1)$$

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.

Improving fault analysis-based insertion. XOR/XNOR key-gates are combined with MUX key-gates to achieve a Hamming distance closer to 50% [10]. Dupuis et al. [6] propose a technique that inserts AND/OR key-gates to minimize the number of low controllability locations in a circuit, making it difficult to insert hardware Trojans in the circuit.

3.6 Key-Guessing Attacks

Multiple attacks have been presented against existing logic encryption techniques. The objective of an attacker is to figure out the key used for encryption of the circuit [3, 10, 13]. These attacks assume that the attacker has access to an encrypted netlist and a functional IC, on which one can apply inputs and observe outputs. There are two main types of attacks — key propagation and SAT attacks — that are described below:

3.6.1 Key Propagation Attacks [13]

The value of an *key-bit* can be determined if it can be sensitized³ without being masked/corrupted by the other key-bits and/or inputs. By observing the output, the value of sensitized key-bit can be determined, given that other key-bits (similar to unknown X-sources⁴) do not interfere with the sensitized path.

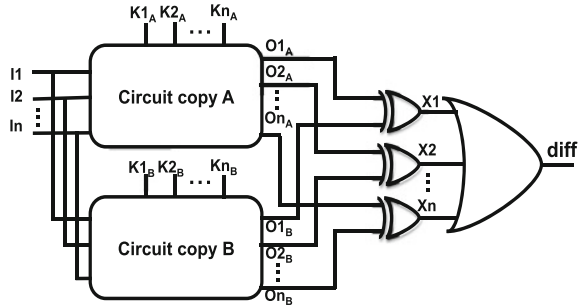
Once an attacker determines an input pattern that sensitizes the key-bit to an output without any interference, it is applied to the functional IC, i.e., the IC with the correct keys. Now, this pattern will sensitize the correct value of the key-bit to an output. An attacker can observe this output and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

Example: Consider the key-input K1 in Fig. 3.1. It will be sensitized to output O1 if the value at the other input of gate G6 is 0 (non-controlling value for an OR gate). This can be achieved by setting I1 = 1, I2 = 0, and I3 = 0. As the attacker has access to the functional IC, he/she can apply this pattern and determine the value of

³Sensitization of an internal line l to an output O refers to the condition (values applied from the primary inputs to justify the side input of gates on the path from l to O to the non-controllable values of the gates) which surjectively maps l to O and thus renders any change on l observable on O .

⁴X-sources: Uninitialized memory units, bus contentions, or multicycle paths are the source of unknown response bits, i.e., unknown-Xs in testing. They are non-controllable.

Fig. 3.4 Miter-like circuit to determine DIPs [13]



$K1$ on $O1$. For example, if the value of $O1$ is 0 for that input pattern, then $K1 = 0$; otherwise, $K1 = 1$.

3.6.2 Boolean Satisfiability (SAT) Attacks [13, 14]

The SAT attack iteratively rules out incorrect key values using distinguishing input patterns (DIPs). A distinguishing input pattern X_d is an input value for which at least two different key values, $k1$ and $k2$, produce differing outputs, $o1$ and $o2$, respectively. Since $o1$ and $o2$ are different, at least one of the key values or both of them are incorrect. It is possible for a single DIP to rule out multiple incorrect key values.

The DIPs are found by constructing a miter-like circuit as illustrated in Fig. 3.4. The primary inputs are common to the two copies of the encrypted circuit, while the key-inputs are left independent. The corresponding outputs of the two circuits are XORed and then Ored to generate *diff* signal. The conjunctive normal form (CNF) of the resultant circuit is generated and passed to a SAT solver. The SAT solver finds a DIP X_d for which $diff = 1$, i.e., the outputs of the two circuits are different. X_d is applied to the functional IC, and correct output I_d is obtained. The input–output pair (X_d, I_d) is used to identify incorrect key values.

A single pattern may not rule out all incorrect keys. Hence, an iterative process is used in the SAT attack, as shown in Fig. 3.5. A new pair (X_d, I_d) is added to the SAT formula in each iteration, and the SAT formula is updated. The generated DIP is applied to the functional IC, and the set of keys that results in an incorrect output is eliminated. The attack is successful when no further DIP is found, which implies that all incorrect key values have been pruned. **Example.** Let us consider the application of the SAT attack on the encrypted example circuit in Fig. 3.6. Figure 3.7 presents the output of the original circuit in column Y and the output of the encrypted circuit for different key values in the following columns. For three key-inputs, there are eight possible key values, which are represented as $k0, k1, \dots, k7$. When the SAT attack is launched on the encrypted circuit, it takes four DIPs to identify the correct key [13]. In iteration 1, the DIP 011 is used. For this DIP, the key value $k4$ alone

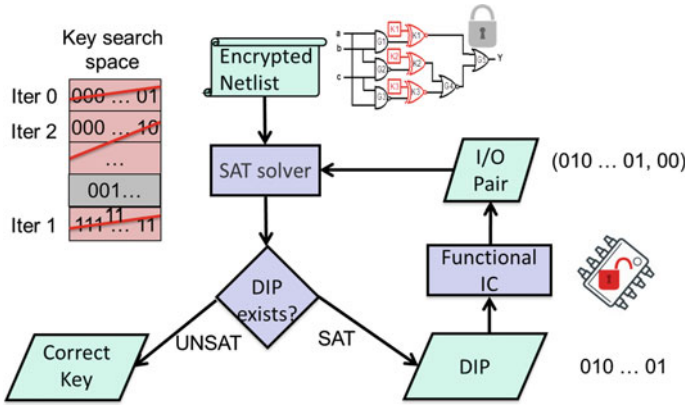


Fig. 3.5 SAT attack on logic encryption [13]

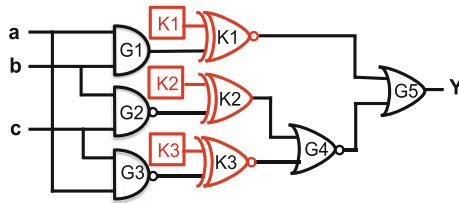


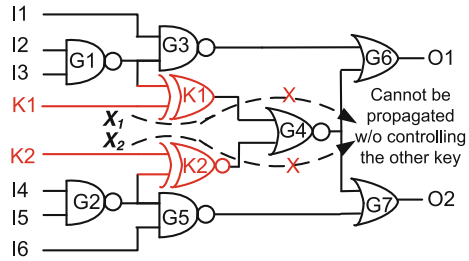
Fig. 3.6 Logic encryption using XOR/XNOR gates [1]. The correct key value is 110. This technique is vulnerable to SAT attack [13]

| No. | a | b | c | Y | Output Y for different key values | | | | | | | | Pruned key values |
|-----|---|---|---|---|-----------------------------------|----|----|----|----|----|----|----|-----------------------|
| | | | | | k0 | k1 | k2 | k3 | k4 | k5 | k6 | k7 | |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | |
| 2 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | iter 1: k4 |
| 4 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | iter 4: all incorrect |
| 5 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | iter 3: k7 |
| 6 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | iter 2: k1 |

Fig. 3.7 Analysis of the SAT attack against logic encryption [13]. Columns k0–k7 show the encrypted circuit’s output for different key values. Red entries in each row denote an incorrect output. The correct key is k6

produces a wrong output as highlighted in red. Thus, only one incorrect key is ruled out in the first iteration. In the second and third iterations, key values k1 and k7 are ruled out, using the patterns 111 and 101, respectively. The pattern 100, used in the

Fig. 3.8 A circuit encrypted using two key-gates K1 and K2 based on the technique proposed in [15]. This prevents key propagation attacks



fourth iteration, eliminates all incorrect keys and the attack successfully identifies the correct key as k_6 .

The attack could have succeeded in the first iteration with a single DIP 100, if this input pattern was tried first. Thus, the execution time of the attack depends on the order in which the input patterns are applied for the SAT attack. The SAT attack, however, chooses the DIPs arbitrarily [13]. The larger the number of incorrect key values ruled out per DIP, the fewer the patterns needed for the attack, which implies a smaller execution time of the attack algorithm.

3.6.3 Countermeasures to Attacks

Countermeasures to Key Propagation Attacks

In order to thwart key propagation attacks, key-gates are inserted such that an attacker cannot propagate the output of a single key-gate [15]. This way, the observed output value is a function of multiple key-gates. Key-gates are inserted such that their sensitization path is blocked by each other. Such key-gates form a “clique.” As the size of the clique increases, the attacker’s effort increases.

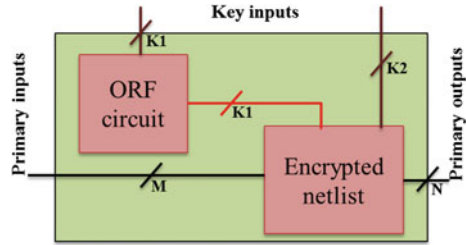
Example. Consider the circuit shown in Fig. 3.8 which is the same functional circuit shown in Fig. 3.1, but the two key-gates $K1$ and $K2$ are at different locations. Here, if the attacker has to propagate the effect of either of the keys, then one has to force a “0” (non-controlling value of NOR gates) on the other input of $G4$. In order to force this value, one has to control the key-inputs, which are inaccessible. Thus, one cannot propagate the effect of a key to an output, failing to determine the values of the key.

To break this scheme, the attack proposed in [10] targets the logic cones with the smallest number of key-inputs and recovers the secret key by employing brute force. The process is then repeated for the remaining logic cones in the circuit, sorted in an increasing order by the logic cone size. To increase the complexity, the number of MUX key-gates is increased to increase the size of the logic cone.

Countermeasure to SAT-Based Attacks [16]

To thwart this attack, clique-based insertion is used along with a cryptographic primitive called *one-way random functions (ORFs)* [17]. ORFs, such as AES with a fixed

Fig. 3.9 ORF-based countermeasure against SAT-based attacks. K_1 out of K key-inputs in the encrypted netlist is connected to the ORF circuit



secret key, prevent an attacker from determining the inputs from the output [18]. First, the designer synthesizes an AES design with a fixed secret key (unknown to the attacker). The resultant design implements a random function. Then, he applies a randomly selected input to the AES with the fixed secret key, which serves as the key for logic encryption. The output of the AES (with fixed secret key) is connected to a subset of XOR/XNOR key-gates added for logic encryption. The designer knows the fixed secret key to the AES and the input applied to AES, and he can configure the key-gates as XOR/XNORs accordingly. This technique is illustrated in Fig. 3.9. The original netlist is encrypted with $K = K_1 + K_2$ number of key-bits. K_1 key-inputs of the encrypted netlist are connected to the output of the AES (with fixed secret key) circuit, and the remaining K_2 key-inputs are connected to the on-chip memory.

This modified scheme will now withstand the SAT attack. A property of the AES is that it is computationally infeasible to determine the inputs of AES from its outputs when the key is unknown [18]. Thus, one cannot backtrace from the outputs of the design and determine the inputs to the AES. In other words, the input to the AES is the secret key for logic encryption.

The limitation of this scheme is it assumes the function-to-be-protected is an unknown logic. Thus, it cannot protect against known functions, because an attacker can “carve out” the logic implemented by the known function. It also assumes that the AES with a fixed key implements a random function. When this random function is cosynthesized with the target unknown function, an attacker cannot classify whether a given gate in the resultant design is part of the random function or the function-to-be-protected.

3.7 Impact of Testing on Logic Encryption

3.7.1 Motivation

Each fabricated chip goes through a manufacturing test that screens out the defective chips. Design for testability (DfT) engineers target generating test patterns that maximize the fault coverage, minimize test pattern count, and reduce test power consumption [19]. The state-of-the-art logic encryption frameworks pursue two different

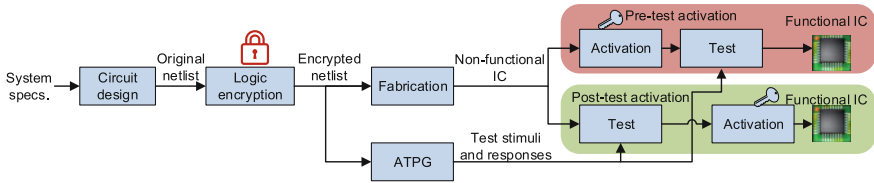


Fig. 3.10 Logic encryption and IC activation in the IC design flow. Pretest activation and post-test activation models

activation models, pretest and post-test activations, that differ in the time of activation of an IC with respect to the manufacturing test. The two models are illustrated in Fig. 3.10 and highlighted in red and green colors, respectively.

Pretest activation. The ICs are activated prior to the manufacturing test, typically conducted in the foundry or outsourced to an OSAT. Since the IP owner does not want to reveal the secret key to the untrusted foundry, on-chip public key cryptographic infrastructure [1] is used to load the secret key securely on the chip. On passing the manufacturing test, the ICs are shipped for assembly/sales directly from the foundry, which is useful in meeting time-to-market constraints.

Post-test activation. The ICs are activated after performing the manufacturing test. Either *remote activation* [20] or *in-house activation* [21] can be employed. In-house activation requires shipping of the encrypted IC from the foundry to the trusted facility, eliminating the need for on-chip cryptography.

Evolving business and threat models. Fabless and fab-lite are the evolving business models for semiconductor companies [22]. Fabless companies, such as Apple Inc., outsource IC fabrication (to Samsung and TSMC [23]), testing, and assembly services. Fab-lite companies such as TI [22] may outsource IC fabrication (to SMIC [24]) and testing, but conduct packaging and assembly in-house. Given the above business models, Apple may activate the ICs remotely using pretest activation, and TI may activate the ICs in-house using post-test activation. However, the test and security implications of these scenarios have never been studied.

3.7.2 Pretest Activation

In pretest activation, the secret key is loaded onto the IC prior to the manufacturing test. The manufacturing test can be conducted in the foundry or a separate test facility (OSAT [25]). Since an IP owner wants to protect the secret key from being exposed to either the foundry or the OSAT, he can load the secret key securely on the chip using public key cryptography infrastructure. Such infrastructure can incur significant area overhead [1].

As the test is to be conducted with the key in place, the secret key values are applied as constraints on the key-inputs during the test generation phase, which can impact the test quality and costs, as well as the security of logic encryption.

3.7.2.1 Threat Model

The attacker is a person in the foundry or test facility with access to the following:

1. An encrypted netlist E_K , which can be obtained by reverse engineering [26] or IP piracy [27].
2. Test stimuli T and responses Γ .

Impact on Security

To highlight the security vulnerabilities of pretest activation, we develop a *test data mining attack* that can reveal the secret key used in pretest activation of logic encryption.

Attack methodology. During the test pattern generation phase, a DfT engineer will apply the correct key K_{corr} as a constraint and obtain a set of test patterns that maximize fault coverage.

An attacker can therefore apply the test stimuli as input constraints and the test responses as output constraints and search for the potentially correct key K_P that maximizes the fault coverage under the specified constraints. The attack is an optimization problem: The objective is to maximize the fault coverage FC under the test stimulus T and test response Γ constraints, as follows:

$$\begin{aligned}
 & \text{maximize } FC \\
 & \text{subject to } \quad \forall_{1 \leq i \leq N} E_K(K_P, T_i) = \Gamma_i \\
 & \text{solve for } K_P
 \end{aligned} \tag{3.2}$$

The rationale for the attack to return the correct key is that the test patterns have been generated with the objective of maximizing the fault coverage in the presence of the correct key as a constraint. When the same set of test patterns are used as constraints, the key that maximizes the fault coverage will be the one that is used to generate the patterns.

Equation 3.2 formulates a system of Boolean equations which can be solved using techniques such as Boolean satisfiability (SAT) or integer linear programming (ILP) [28]. Test generation (ATPG) algorithms are capable of solving a system of Boolean equations while maximizing the fault coverage at the same time; ATPG is, therefore, a natural candidate for solving the optimization problem in Eq. 3.2. The complexity of the attack is NP-hard [29].

Let us consider the netlist shown in Fig. 3.8. When the correct key value $K_{corr} = 00$ is used as a constraint, eight test patterns are generated by the ATPG tool as listed in Table 3.1. An attacker will launch the attack described in Eq. 3.2 by applying the test stimuli and responses as constraints and search for the potential key K_P that maximizes the fault coverage. The only key K_P that maximizes the fault coverage and satisfies these test pattern constraints is 00, and the corresponding fault coverage is 82.43%. None of the other key values satisfies the test pattern constraints.

Table 3.1 Test patterns (pretest activation) for the netlist in Fig. 3.8. The correct key K_{corr} is used as a constraint during ATPG

| Key(K_{corr}) | Stimulus (T) | Response (I) |
|-------------------|------------------|------------------|
| 00 | 011001 | 10 |
| 00 | 101010 | 01 |
| 00 | 101111 | 01 |
| 00 | 011101 | 10 |
| 00 | 111010 | 11 |
| 00 | 000111 | 11 |
| 00 | 110001 | 00 |
| 00 | 001011 | 10 |

3.7.3 Post-test Activation

In post-test activation, the manufacturing test is conducted on an encrypted IC with the rationale that manufacturing test is a “structural” test and that the chip need not be functional during the test. The IC can be activated post-test in one of the following ways:

1. After manufacturing test, defect-free ICs are shipped to a trusted facility, activated by the IP owner, and shipped out for sale.
2. Tested ICs can also be activated remotely, similar to the case of pretest activation, via public key cryptography infrastructure [30].

Impact on security. In post-test activation, both test pattern generation and manufacturing test are conducted in the absence of the secret logic encryption key. Any analysis performed by the attacker will only reveal these arbitrary key values and not the secret key value. Therefore, post-test activation has no detrimental impact on the security of logic encryption.

3.7.4 Hill Climbing Attack

The hill climbing search-based attack [21] uses test data information to guess the secret key for pretest activated ICs. The attack tries to achieve zero Hamming distance HD_O between the test response and the encrypted circuit, for multiple random key guesses. The individual bits in the initial key guess are flipped if the flip minimizes the Hamming distance HD_O .

Example. Consider the circuit shown in Fig. 3.8 and the test patterns shown in Table 3.1. In iteration 1, the hill climbing attack starts with a random key value, say 00. With this key value, the test patterns are applied and the responses are collected. The cumulative Hamming distance between the collected responses and the responses from the test set is calculated. In this case, it is 12. Now, one of the key-bits will be flipped, say the new key is 01. The cumulative Hamming distance between the

collected responses and the responses from the test set is now reduced to 6. Since there is a reduction in the cumulative Hamming distance, the second bit is retained as 1. Now, the remaining bit is flipped (key = 00) and the cumulative Hamming distance is 0. Thus, the attacker identifies that 00 is the correct key.

3.8 Other Techniques Based on Logic Encryption

Logic encryption is used not only in the context of IP protection, but also in other applications as well. Two of them are described below:

3.8.1 *Secure Split Test (SST) [20, 30]*

An untrusted test facility can mark a defective IC, which failed the test, as a good quality one. Consequently, a designer unknowingly sells low-quality ICs, spoiling his reputation. Further, a test facility can also label “good” ICs as “faulty” ICs and sell them in the black market. Such an attack is possible only when the attacker (malicious tester) can identify whether an IC is defective or not. In order to prevent this attack, a designer should hide the test responses. For this purpose, the test infrastructure should be protected. Logic encryption aids in protecting the test infrastructure, thereby hiding the correct responses.

3.8.2 *Securing Processor Architectures [31]*

Modern processors are equipped with several security modules to aid detection and prevention of attacks. These modules usually inform the processor pipeline about the attack through a signal, which is carried by a wire. Unfortunately, such wires are susceptible to Trojan attacks, enabling an attacker to modify the target signal value at will. Detecting a Trojan that modifies a single wire is difficult. Consequently, he can launch traditional software attacks and still go undetected.

To prevent such attacks, logic encryption encrypts a processor module and stores the key within a security module. The security module, instead of sending the signal, sends the correct key to unlock a processor pipeline, when it does not detect an attack. When it detects an attack, it sends a wrong key, resulting in an incorrect computation. Since the size of Trojan required to mask/modify a multibit key is bigger than the one required to modify a single wire, such Trojans can be easily detected.

Table 3.2 A comparison of the attacks against logic encryption

| Study | Attacker's location | Attacker's capabilities | Attack method | Defense |
|------------------------------|---------------------------|---------------------------------------|--|---|
| Rajendran et al. [15] | Foundry and end user | Encrypted netlist and an activated IC | Sensitization of key-bits to outputs | Clique-based insertion [15] |
| Subramanyan et al. [13] | Foundry and end user | Encrypted netlist and an activated IC | SAT-based algorithm to rule incorrect keys | Strong logic encryption OWF [16] |
| Plaza and Markov [21] | Foundry and test facility | Encrypted netlist and test set | Find the key that minimizes Hamming distance | Test-aware combinational logic encryption |
| Test data mining attack [32] | Foundry and test facility | Encrypted netlist and test set | Find the key that maximizes fault coverage | Post-test activation |
| SST [30] | Foundry | Activated ICs to be tested | Reduce yield | Encrypting test circuits |

3.9 Conclusion

Logic encryption is an emerging technique to thwart IP piracy, reverse engineering, and hardware Trojans. Initially, most of the techniques proposed in the literature are based on VLSI testing principles. However, recent attacks have broken these techniques, even though the complexity of those techniques is NP-hard. Researchers are now trying to adopt concepts from cryptography and apply them to logic encryption.

Table 3.2 summarizes the different attacks and their countermeasures on logic encryption. Clique-based insertion of key-gates can prevent sensitization attacks but is susceptible to SAT attacks. Strong logic encryption can prevent both SAT and sensitization attacks, but it is applicable only for random unknown logic. Test data mining attack is applicable only when the attacker has access to test patterns and responses, which are generated for the correct key; they can be prevented by post-test activation.

References

1. Roy J, Koushanfar F, Markov IL (2008) EPIC: ending piracy of integrated circuits. In: Proceedings of the IEEE/ACM design, automation and test in Europe, pp 1069–1074
2. Chakraborty RS, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. *IEEE Trans Comput-Aided Des Integr Circuits Syst* 28(10):1493–1502
3. Plaza SM, Markov IL (2015) Solving the third-shift problem in IC piracy with test-aware logic locking. *IEEE Trans Comput-Aided Des Integr Circuits Syst* 34(6):961–971

4. Rajendran J, Zhang H, Zhang C, Rose G, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. *IEEE Trans Comput* 64(2):410–424
5. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Logic encryption: a fault analysis perspective. In: *Proceedings of the IEEE/ACM design, automation and test in Europe*, pp 953–958
6. Dupuis S, Ba P, Natale GD, Flottes M, Rouzeyre B (2014) A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In: *Proceedings of the IEEE international on-line testing symposium*, pp 49–54
7. Karri R, Rajendran J, Rosenfeld K, Tehranipoor M (2010) Trustworthy hardware: identifying and classifying hardware trojans. *IEEE Comput* 43(10):39–46
8. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. *IEEE Des Test Comput* 27(1):10–25
9. Roy JA, Koushanfar F, Markov IL (2010) Ending piracy of integrated circuits. *Comput* 43(10):30–38
10. Lee Y-W, Toubia N (2015) Improving logic obfuscation via logic cone analysis. In: *Proceedings of the Latin-American test symposium*, pp 1–6
11. Contreras GK, Rahman MT, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: *Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems*, pp 196–203
12. Bushnell ML, Agrawal VD (2000) *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Kluwer Academic Publishers, Boston
13. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: *Proceedings of the IEEE international symposium on hardware oriented security and trust*, pp 137–143
14. Massad ME, Garg S, Tripunitara MV (2015) Integrated circuit (IC) decamouflaging: reverse engineering camouflaged ICs within minutes. In: *NDSS*
15. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: *Proceedings of the IEEE/ACM design automation conference*, pp 83–89
16. Yasin M, Rajendran J, Sinanoglu O, Karri R (2015) On improving the security of logic locking. *IEEE Trans Comput-Aided Des Integr Circuits Syst* 99:1–1
17. Matsuzaki N, Tatebayashi M (1994) Apparatus and method for data encryption with block selection keys and data encryption keys. *US Patent 5,351,299*
18. Goldreich O (2001) *Foundations of cryptography: basic tools*, vol 1. Cambridge University Press, Cambridge
19. Bushnell M, Agrawal VD (2000) *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, vol 17. Springer, New York
20. Contreras G, Rahman M, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: *Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems*, pp 196–203
21. Plaza SM, Markov IL (2014) Protecting integrated circuits from piracy with test-aware logic locking. In: *Proceedings of the IEEE/ACM international conference on computer-aided design*, pp 262–269
22. McLellan P (2013) A brief history of the foundry industry, part 2, [Sep 1, 2015]. <https://www.semiwiki.com/forum/content/2109-brief-history-foundry-industry-part-2-a.html>
23. AppleInsider (2015) Samsung reportedly nabs 75% of Apple's next-gen 'A9' SoC orders, [Aug 10, 2015]. <http://appleinsider.com/articles/15/01/26/samsung-to-reportedly-take-75-of-apples-nextgen-a9-soc-orders>
24. Releases SP (2014) SMICs Beijing fab wins TI quality excellence award, [Aug 10, 2015]. http://www.smics.com/eng/press/press_releases_details.php?id=107870
25. Wire B (2014) Research and markets: outsourced semiconductor assembly and test market (OSAT) trends, [Aug 22, 2015]. <http://www.businesswire.com/news/home/20140324005628/en/Research-Markets-Outsourced-Semiconductor-Assembly-Test-Market>
26. Torrance R, James D (2011) The state-of-the-art in semiconductor reverse engineering. In: *Proceedings of the IEEE/ACM design automation conference*, pp 333–338

27. Rostami M, Koushanfar F, Karri R (2014) A primer on hardware security: models, methods, and metrics. *Proc IEEE* 102(8):1283–1295
28. Clarke E, Gupta A, Kukula J, Strichman O (2002) SAT based abstraction-refinement using ILP and machine learning techniques. In: *Proceedings of the computer aided verification*, Springer, pp 265–279
29. Krishnamurthy B, Akers SB (1984) On the complexity of estimating the size of a test set. *IEEE Trans Comput* 33(8):750–753
30. Rahman MT, Forte D, Shi Q, Contreras GK, Tehranipoor MM (2014) CSST: preventing distribution of unlicensed and rejected ICs by untrusted foundry and assembly. In: *Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems*, pp 46–51
31. Rajendran J, Kanuparthi AK, Zahran M, Addepalli SK, Ormazabal G, Karri R (2013) Securing processors against insider attacks: a circuit-microarchitecture co-design approach. *IEEE Des Test* 30(2):35–44
32. Yasin M, Saeed SM, Rajendran J, Sinanoglu O (2016) Activation of logic encrypted chips: pre-test or post-test?. In: *Proceedings of the IEEE/ACM design, automation and test in Europe*