The End of Logic Locking? A Critical View on the Security of Logic Locking

Susanne Engels, Max Hoffmann, Christof Paar, *Fellow, IEEE*
Ruhr University Bochum, Germany, Horst Görtz Institute for IT-Security
firstname.lastname@rub.de

Index Terms—**Logic Locking, Logic Encryption**

***Abstract*—With continuously shrinking feature sizes of integrated circuits, the vast majority of semiconductor companies have become *fabless*, i.e., chip manufacturing has been outsourced to foundries across the globe. However, by outsourcing critical stages of IC fabrication, the design house puts trust in entities which may have malicious intents. This exposes the design industry to a number of threats, including piracy via unauthorized overproduction and subsequent reselling on the black market. One alleged solution for this problem is *logic locking*, also known as *logic encryption*, where the genuine functionality of a chip is "locked" using a key only known to the designer. If a correct key is provided, the design works as intended but with an incorrect key, the circuit produces faulty outputs. As the keys are inserted by the designer after production, an adversarial foundry should not be able to unlock overproduced chips.**

In this work, we highlight major shortcomings of proposed logic locking schemes. They are primarily due to the absence of a well-defined and realistic attacker model in the current literature. To this end, we characterize physical capabilities of a malicious foundry, especially with respect to invasive attacks. This allows us to derive an attacker model that matches reality, yielding attacks against the foundations of locking schemes beyond the usually employed SAT-based attacks. Our analysis shows that no previously proposed logic locking scheme is able to achieve the intended protection goals against piracy in real-world scenarios. As an important conclusion, we argue that there are strong indications that logic locking will most likely never be secure against a determined malicious foundry.

I. Introduction

In today's semiconductor industry, many steps of the fabrication chain are outsourced for complexity and cost reasons. Most semiconductor companies have become *fabless*, with chip manufacturing, testing, and assembly performed at specialized providers across the globe. While avoiding the substantial costs of maintaining and upgrading own foundries, new threats arise when designs are sent to offshore fabs: Integrated Circuits (ICs) become susceptible to overproduction, counterfeit, and reverse engineering. Apart from the financial loss for semiconductor companies [1], counterfeited products can lead to major safety and security concerns [2].

In order to secure a design against rogue players in the fabrication chain, countermeasures such as *logic locking*, also known as *logic encryption*, have been proposed over the last years. The idea of logic locking is to integrate a locking mechanism into the circuit such that it produces faulty outputs whenever an incorrect key is provided. Although possessing all information required to fabricate the integrated circuit, a malicious foundry lacks the secret unlocking key to activate any overproduced ICs and sell them on the black market. Only the holder of the Intellectual Property (IP) rights who is in possession of that key should be able to unlock the IC. Even though the prospect of logic locking sounds promising to protect against piracy, the lack of a well-defined attacker model yielded many loosely argued security sketches in the past. More critically, while research on logic locking has become an arms-race between increasingly specialized SAT-based attacks and corresponding countermeasures, only restricted attacker models have been considered so far in the literature.

Contributions The three main contributions of the work at hand are (i) a comprehensive survey of logic locking schemes, (ii) a realistic attacker model and, crucially, (iii) we demonstrate that all proposed **logic locking schemes are insecure** in real-world settings, i.e., when facing an adversarial foundry. We survey existing combinational logic locking schemes and characterize them by their locking approach, extending the work of Yasin et al. [3]. Based on this survey, we identify common pitfalls when modeling attacker's capabilities and discuss invasive techniques available to a malicious foundry. Taking capabilities of realistic adversaries into account, we derive a novel attacker model that does not only include non-invasive, but also invasive approaches. Finally, we present profound attacks that successfully break **all** existing schemes. In contrast to previous attacks, which were mainly SAT-based, our approaches target the building blocks that are the foundation of logic locking instead of individual scheme specifics. Concluding this work, we generalize our findings, arguing that logic locking will most likely never succeed against a malicious foundry.

Outline The remainder of this work is structured as follows: we start by introducing the motivation, basic concept, and various existing attacker models for combinational logic locking, before shortly addressing sequential logic locking and introducing important terminology in Section II. Next, we summarize existing combinational logic locking schemes and introduce a classification of prior work in Section III. In Section IV, we present the pitfalls of existing attacker models and discuss invasive and non-invasive attack capabilities of an adversarial foundry. Section V presents a new (realistic) attacker model and shows that, when taking invasive attacks into account, all proposed schemes can be successfully attacked. We conclude our work in Section VI.

II. Logic Locking

In this section, we provide background information on logic locking. We first introduce the motivation and main goals, before outlining how logic locking works in general. We explain the difference between combinational and sequential logic locking and summarize the assumptions made in previous research regarding the adversary.

A. Motivation and Objectives

Most design houses have become *fabless*. They outsource the physical production, assembly, and testing of their integrated circuits to service providers across the globe. Figure 1 shows a simplified view on a common fabrication chain. Since outsourced processes are outside the direct control of the design house, they must be considered potentially malicious environments. Every external entity in the fabrication chain is hence *untrusted* (hatched red in Figure 1).

Logic locking aims to protect an IC against piracy by untrusted parties in the fabrication chain, starting from the point when it leaves the design house, throughout the manufacturing process, and its remaining life cycle. Piracy can be of physical nature, as in overproduced products or IP-piracy through reverse engineering. A malicious foundry poses a particularly strong adversary in this scenario. Since it receives the design in form of GDSII/OASIS files and generates the lithographic masks for subsequent fabrication of ICs, it has full access to error-free representations of all layers in the design. Note that obtaining error-free images when reverse-engineering a finished IC is a challenging task [4] which the malicious foundry does not have to face.

a) Protection Goals: The primary goal of logic locking is to defend against IC overproduction and subsequent trade, e.g., on the black market. Only the design house should control the ICs available on the market. Some schemes also claim to protect against reverse-engineering, however, this is not a primary goal of logic locking — protection against reverse-engineering is traditionally addressed by obfuscation.

Logic locking can be divided into two major categories: combinational and sequential logic locking.

B. Combinational Logic Locking

Combinational logic locking is a concept which was first introduced by Roy et al. in their 2008 publication EPIC [5]. In this section we present the general idea, while more details on EPIC are provided in Section III.

Combinational logic locking extends the Boolean logic of a design with additional locking gates and provides an input mechanism for an unlocking key. If the correct key is applied to the protected design, i.e., it is unlocked, its output behavior is identical to the original design. However, if an incorrect key is supplied, the locking circuitry manipulates computations such that faulty outputs are generated. An example of an EPIC-locked circuit is shown in Figure 2.

The unlocking key is only known to the design house. Post fabrication, it is once securely transmitted onto the IC, where it is stored in dedicated on-chip memory. Thus, the design house can control which products are unlocked and a malicious foundry cannot sell overproduced chips since it lacks the key. Since the locking circuitry is closely intertwined with the existing logic, plain removal of all locking-related gates would result in a broken circuit.

Combinational logic locking schemes have been subject to several non-invasive attacks, mostly based on SAT-solving. The general idea is to find an unlocking key which produces the same I/O behavior as observed from an unlocked device. More details on non-invasive attacks against combinational logic locking are given in Section IV-B.

C. Sequential Logic Locking

In sequential logic locking, the data path of a design remains untouched while its control logic, i.e., Finite State Machines (FSMs), is protected. Simplified, only a specific state sequence, analogously to the unlocking key in combinational logic locking, will lead to the start state of the original FSM while all other sequences lead to dead ends or infinite loops.

Four basic classes of sequential logic locking schemes exist. They were initially presented in the following contributions: HARPOON by Chakraborty et al. [7], Dynamic State Deflection by Dofe et al. [8], Active Hardware Metering by Alkabani et al. [9], and Interlocking Obfuscation by Desai et al. [10]. There has been considerable follow-up work, mostly improving specific aspects of the original schemes.

However, sequential logic locking schemes based on FSMs are considered broken, since all four classes have been successfully attacked in recent work by Fyrbiak et al. [11]. Notably, the authors focused on underlying characteristics of FSMs before taking scheme specific properties into account. They also provide a novel sequential locking approach based on reconfigurable logic that eliminates the common weakness of the existing sequential schemes. However, there is a large overhead in terms of area and latency due to the reconfigurability. Even though the new approach does not constitutes a perfect solution, it underlines that a fundamental rethinking to sequential logic locking is required.

Following the results of Fyrbiak et al. [11], we consider sequential logic locking in its current state as broken. Still, while this work does not focus on sequential logic locking, keep in mind that the underlying principles of all attacks presented in this work are also applicable to sequential logic locking.

D. Notation and Terminology

In the current literature, the terms “logic locking” and “logic encryption” are used synonymously. We want to emphasize a remark from Plaza and Markov that this mixed terminology is ill-advised [12]. Indeed “encryption” is tied to making data indecipherable through transformation of the data itself, while “locking” describes blocking functionality until unlocked. Hence,“logic locking” is a notably more appropriate term and we will use it in the remainder of this work. Moreover, in this work, the term “logic locking” will implicitly refer to combinational logic locking.

The terminology in the logic locking literature has not always been consistent. Below, we introduce generic terms which

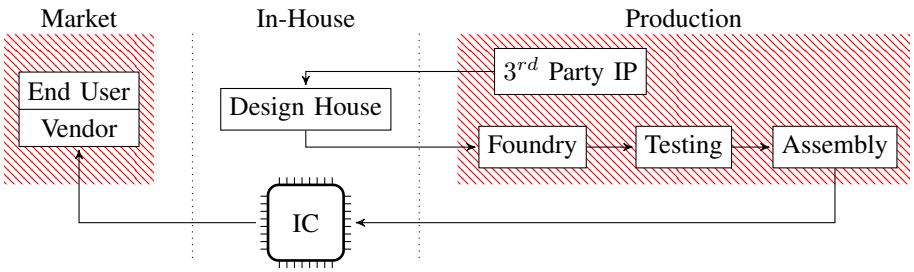


Figure 1: Simplified view on an exemplary, outsourced design fabrication process. Every entity hatched red is untrusted and can potentially be malicious.

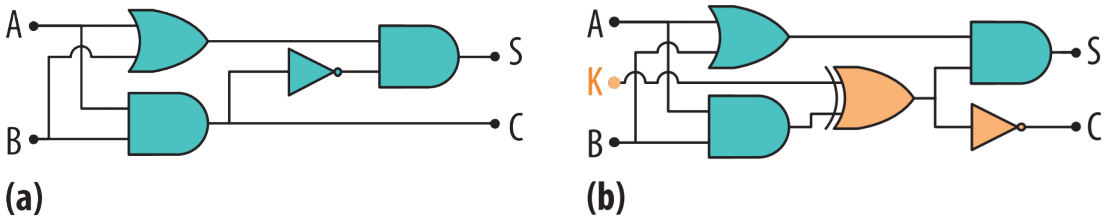


Figure 2: An example of locking circuitry as depicted in [6]. (a) is the original design and (b) the locked result.

appear to be most suitable to address all existing schemes while avoiding confusion:

- **Chip Key** is the key which unlocks one specific IC. In some schemes, all ICs are unlocked using the same chip key, while others require individual keys for activation.
- **Unlocking Information** is the data which is given as input to a locked chip in order to activate it. In some schemes, the unlocking information is the chip key itself, in other schemes the chip key is derived internally via a key preprocessor (cf. below) from the unlocking information.
- **Individual** chip key or unlocking information means that each IC has its own unique data.
- **Global** chip key or unlocking information means that the respective data is identical for each IC.
- **Key Preprocessor** is an optional module preceding a locking scheme. It receives the unlocking information provided by the design house and derives the internal chip key, passing it to the underlying locking scheme. If no preprocessor is used, the unlocking information (in that case the plain chip key) is the direct input to the locking scheme.

E. Attacker Model in Previous Work

It is worth noting that the logic locking literature lacks a well-defined attacker model. Most schemes define the information available to the adversary as follows: he has access to several locked and unlocked ICs, as well as a gate-level netlist of the design. Note that access to unlocked ICs implies that they are already available on the open market or that the attacker has an insider at some stage where ICs are unlocked.

Critically, in previous work, the attacker's capabilities are only discussed in-depth with respect to non-invasive attacks (such as SAT-based attacks). The adversary is allowed to observe the input/output behavior of the ICs **only** via black-box access and to analyze and simulate the locked netlist. In previous work, invasive attacks have been mentioned only in passing.

Additionally, the protection goals of logic locking as well as the definition of what constitutes a successful attack vary in the literature. While all schemes have the goal to protect against overproduction, some schemes additionally claim to protect against reverse engineering or hardware Trojan insertion. Concrete goals for an attacker are rarely named and several SAT attacks claim success after recovering the chip key [13], although, depending on the scheme, the recovered chip key alone cannot be used to attack the locking scheme, e.g., in the case of EPIC.

III. Classification of Existing Logic Locking Schemes

In this section, we give a summarizing overview on previously proposed schemes for combinational logic locking and classify them based on their individual locking circuitry. Note that we focus on novel logic locking schemes, not on building blocks for logic locking to defend against specific SAT attacks. The timeline in Table I sorts the previous work by publication year. An extended overview with details on each individual logic locking scheme can be found in Section A.

In the proposed schemes, two basic logic locking components can be identified, namely (1) the locking scheme itself and (2) a key preprocessor. The former is part of all solutions while the latter is an optional building block that can be prepended to any scheme (cf. Section II-D). Therefore, we analyze locking schemes and key preprocessors separately: Starting with locking schemes that incorporate a global chip key, we continue with schemes that feature individual chip keys, before finally focusing on the available key preprocessors. Keep in mind that in this section we will only present the available schemes and key preprocessors, their security is discussed later in Section V.

2008	[5], [6]:	EPIC: Ending Piracy of Integrated Circuits
2010	[14]:	Preventing IC Piracy Using Reconfigurable Logic Barriers
2012	[15]:	Security Analysis of Logic Obfuscation
	[16]:	Logic Encryption: A Fault Analysis Perspective
	[17]:	CLIP: Circuit Level IC Protection Through Direct Injection of Process Variations
2014	[18]:	A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans
2015	[12]:	Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking
	[19]:	Fault Analysis-Based Logic Encryption
2016	[20]:	On Improving the Security of Logic Locking
2017	[21]:	Logic locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism
	[22]	A New Logic Encryption Strategy Ensuring Key Interdependency
	[23]:	What to Lock? Functional and Parametric Locking
	[24]:	Provably-Secure Logic Locking: From Theory to Practice.
2018	[25]:	ATPG-Based Cost-Effective, Secure Logic Locking
2019	[26]:	Vulnerability and Remedy of Stripped Function Logic Locking

Table I: Timeline of publications on combinational logic locking schemes

A. Locking Schemes with a Global Chip Key

In the following we will provide an overview on previous work incorporating a global chip key. We further categorize them by their main focus, that is whether they focus on (1) a new type of locking circuitry, (2) placement of the locking circuitry, or (3) a combination of both.

1) Focus: Type of Locking Circuitry: EPIC is the initial work that introduced the concept of logic locking in 2008, with the goal to protect ICs against overproduction and reverse engineering [5], [6]. It consists of two main contributions, (1) simple locking circuitry and (2) an RSA-based key preprocessor, which we will discuss in detail in Section III-C. The locking circuitry features XOR and XNOR gates that are randomly placed in the design while avoiding critical paths. If a correct chip key bit results in inversion of the original signal value, an additional inverter is added further down the signal.

2) Focus: Placement of Locking Circuitry: In 2012, Rajendran et al. noted that if key gates are positioned at random, they might potentially "mute" each other's effect [15], thus weakening the security of the locking scheme. Depending on how one key gate affects another, the authors detail on various attack strategies to determine bits of the chip key. To thwart such attacks, they present a placement algorithm that only inputs 10% of the key gates at random and then finds optimal positions for the remaining key gates in an iterative manner.

In a second contribution from 2012, Rajendran et al. used fault simulation techniques to enhance logic locking [16]. They regard a key gate which receives an incorrect key input similar to a (stuck-at) fault during computation. Analogously to the work in [15], the authors argue that randomly inserted key gates might cause faults that are not propagated to the output. To mitigate that problem, they present an algorithm which identifies key gate positions with a good "fault impact". A Hamming distance of 50% between outputs under wrong resp. correct chip keys is suggested in order to maximize ambiguity for the adversary.

In 2015 Rajendran et al. focused on improving the aforementioned algorithm from [16] to find suitable key gate positions more optimally [19]. On a side note, they provide experimental results showing that fault-based locking works slightly better with XOR/XNOR gates than with multiplexers.

The initial work of Rajendran et al. [15] also forms the foundation of the 2016 contribution by Yasin et al. [20], which further elaborates on attack strategies that leverage key gates affecting each other. They try to sensitize all key bits possible to the output, i.e., making key bits deducible from outputs signals via I/O testing, and thus reduce the complexity of a brute-force attack. The authors provide an algorithm which finds the correct attack strategy depending on the positioning of key gates. Building on their attack insights, they present Strong Logic Locking (SLL) which inserts key gates with the help of an interference graph to minimize the amount of key bits that can be sensitized to the output.

In 2017 Karmakar et al. combined both previous approaches to algorithmically determine optimal key positions [22]. The premise of both [19] and [20] is to find a suitable key position, but they optimize for different goals, i.e., high fault impact and sensitization mitigation. To provide the best of both worlds, Karmakar et al. propose a logic locking approach based on

fault analysis, while at the same time inserting a part of the key gates according to the rules of SLL.

In their 2018 work [25], Sengupta et al. improved the placement of the locking circuitry with Stripped-Functionality Logic Locking (SFLL) [24], a locking scheme that strips functionality and uses a keyed restore-unit for unlocking (cf. next section). Instead of randomly selecting signals to protect, the authors use automated fault injection on every node of the circuit to identify where functionality stripping is most effective.

3) Focus: Combination of Type and Placement: In 2010, Baumgarten et al. published the first follow-up work on EPIC [14]. Instead of using X(N)OR gates, the authors propose to insert "reconfigurable logic barriers", i.e., programmable look-up tables, into the design which will be configured after fabrication. These "micro FPGAs" are supposed to protect the original design against reverse engineering and overproduction. Furthermore, they presented an algorithm for placement of logic barriers that minimizes attack potential.

Dupuis et al. proposed a logic locking scheme based on AND resp. OR gates in 2014 [18]. An additional goal of this scheme is to protect against insertion of hardware Trojans. According to the authors, Trojans are often triggered by a rarely changing signal. Hence, by placing key gates in the paths of rarely switching signals, switching probabilities of signals are adjusted and the insertion of hardware Trojans is thwarted.

The work of Plaza and Markov focuses on a malicious testing lab and describes logic locking based on multiplexer gates [12]. The authors highlight that for EPIC-based locking schemes, the IC needs to be unlocked before it can be tested because locked ICs will fail testing. This enables an attacker in the testing lab to deduce the chip key from the test responses of the unlocked circuit. To thwart this problem, the authors propose a novel locking scheme based on multiplexer gates. Using their scheme, an IC can be functionally tested before it is unlocked if the multiplexers are inserted in a clever fashion, thus preventing the deduction of the chip key at the testing facility.

With "Meerkat", El Massad et al. presented a new logic locking technique designed to conquer the novel *desynthesis attack* discovered by the same authors [21]. In a desynthesis attack, wrong key candidates can be excluded bit-by-bit via resynthesis of the netlist for a key guess and comparison of the output with the locked netlist. This attack requires knowledge about the employed locking scheme as well as the originally used synthesis tools, but in contrast to other attacks, the adversary does not need access to an already unlocked circuit. This property is the main strength of the desynthesis attack. To counteract, Meerkat is applied before/during the synthesis step instead of locking an already synthesized netlist. Internally it is based on reduced ordered binary decision diagrams that are implemented via MUXes. The authors also formalize mathematical requirements for a locking scheme to be secure against desynthesis attacks.

In 2017 Yasin et al. presented TTLock [23] which was later refined to Stripped-Functionality Logic Locking (SFLL) [24]. The main idea is to strip parts of the logic and add a restore unit which will correct the now faulty signals if the correct chip key is present. This restore unit consists mainly of look-up tables in tamper-proof memory. The authors argue that their method also protects against reverse engineering, since functionality is actively removed during locking, i.e., a reverse engineer can only acquire an incomplete design.

In 2019, Zhou et al. reviewed SFLL and presented the bit-coloring attack, which allows for deducing the secret key in polynomial time with the help of a single known protected input pattern [26]. As a remedy, the authors propose to disguise the primary inputs of the restore unit with one-way functions, thus thwarting the possibility to use the circuit as an oracle.

B. Locking Schemes with Individual Chip Keys

In contrast to the schemes presented in Section III-A, there is only a single scheme which generates an individual chip key for each IC: CLIP by Griffin et al. [17]. To this end, CLIP employs process variation sensors which measure slight differences in transistor threshold voltages, yielding a mechanism comparable to a weak Physically Unclonable Function (PUF) [27], [28]. In order to protect a combinational function f, CLIP locally duplicates f. Selected by the value of one of the aforementioned process variation sensors, the inputs to one of the f-instances are manipulated. Hence, one instance of f will compute wrong results while the other instance outputs correct values. An output multiplexer decides, based on one bit of the chip key, which of the two computations becomes the final output. As a result, only if the key bit selects the output of the f-instance which gets unmodified inputs, the circuit behaves correctly. With CLIP, each IC produces its own individual chip key, even the designer has no way of knowing a specific chip key beforehand. Therefore, the designer has to query each produced chip with a test set in order to recover the chip key from outputs and subsequently unlock the device. However, the authors acknowledge that this can also be done by a malicious entity.

C. Key Preprocessors

Since in most locking schemes all ICs share a global chip key, the need of a mechanism to individualize ICs has been solicited by various authors. Such a mechanism, i.e., a key preprocessor, allows for each IC to accept individual unlocking information which is internally processed, yielding the actual chip key. Again, in this section we will only present the available key preprocessors, their security is discussed later in Section V.

Roy et al. presented the first key preprocessor together with EPIC [5], [6]. Their initial motivation was enabling remote unlocking of ICs via asymmetric cryptography. The key preprocessor features a PUF or True Random Number Generator (TRNG) and an RSA engine, capable of key generation, decryption, and signature verification. While the public RSA key of the design house is hardcoded, an individual RSA key pair for the IC is generated during the initial power-up using the PUF or TRNG as a source of randomness. This key pair is then burnt into fuses. In order to unlock an IC, the design house encrypts the global chip key using the individual RSA public key of that IC and then signs the

resulting ciphertext with its own private key. The unlocking information sent to the IC contains both, the encrypted chip key and the signature. The IC then internally derives the chip key from the unlocking information by verifying the signature and subsequently decrypting the ciphertext with its own private key. Effectively, the EPIC key preprocessor yields individual unlocking information for each IC even if the chip key is global, thus overproduced chips cannot be unlocked even if said chip key is disclosed. Thanks to the signature, remote unlocking is enabled since an attacker cannot forge a valid signature.

The idea for a different key preprocessor, a Logic Encryption Cell (LEC), was briefly mentioned by Rajendran et al. in 2012 [16]. For LECs, the unlocking information consists of a PUF challenge and additional data the authors refer to as *user key*. On chip, the PUF response is XORed with the user key to generate the chip key. The authors argue that a PUF circuit can be implemented much smaller than an RSA engine, however, they do not provide any information regarding instantiation and PUF setup. Furthermore, it is claimed that, even if an adversary obtains the unlocking information, i.e., PUF challenge and user key, security is not compromised since the attacker cannot compute the PUF response. According to the authors, LECs therefore provide the same security as an RSA engine. However, it is not described how the design house can learn/challenge the PUF prior to unlocking in order to generate valid unlocking information.

D. Summary

In total, existing locking schemes can be categorized as visualized in Table II. Regarding key preprocessors, only two instantiations have been proposed, namely the EPIC key preprocessor [5], [6] and LECs [16].

Type of Chip Key	Locking Circuitry Focus	Scheme
Global	Type	[5], [6]
	Position	[15] [19] [20] [21] [22] [25]
	Combined	[14] [18] [12] [24]
Individual	Combined	[17]

Table II: Categorization of existing locking schemes.

IV. Classification of a Foundry's Capabilities

With respect to the logic locking literature, we argue that there are two shortcomings regarding security modeling: protection goals are inconsistent and, more importantly, assumptions regarding the adversary's capabilities are incomplete. Both aspects are crucial for a sound assessment of the security of logic locking schemes. Invasive attacks in general have been mostly neglected in previous work, even though they pose a major and realistic threat. In this section, we introduce two possible invasive attack vectors that will be used in Section V-B to actually attack logic locking. We also summarize proposed non-invasive attacks.

A. Invasive Attack Vectors for Foundry-level Adversaries

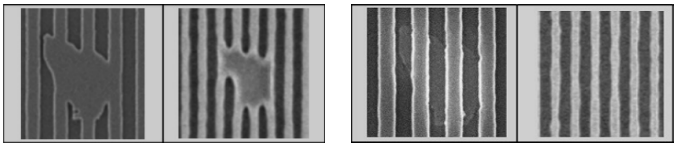
All logic locking schemes attempt to defend against malicious entities in the fabrication chain (cf. Section II-E). In previous work, it is assumed that the adversary has only black-box access to locked and unlocked ICs and is in possession of a gate-level netlist of the locked design. In other words, invasive approaches are excluded. However, this severely underestimates the capabilities of a foundry-level adversary, who is the very actor involved in overproduction. The foundry produces the ICs itself, i.e., it has physical access in the most unobstructed way. Its extensive knowledge about the layout of all components allow to identify points of interest with ease. Therefore, the black-box-only assumption is insufficient. Focusing on these invasive capabilities against logic locking, we will now introduce the attack vectors *extracting static information* and *minimal mask modification*.

1) Attack Vector: Extracting Static Information: In logic locking, the chip key is a core asset that should only be known to the design house. If an attacker gets hold of said key, the strength of the scheme is notably reduced or even entirely nullified. Hence, a strong attack vector involves probing the chip key or, more generally, the unlocking information on an unlocked IC during operation.

While probing of signal values can be difficult for a generic attacker, e.g., malicious IC vendors, every foundry can draw on the sophisticated testing labs available to virtually all modern fabs [29]. Hence, it is well acquainted with backside probing techniques, such as e-beam or laser voltage probing as used in standard testing routines [30], [31], [32], or electro-optical probing and electro-optical frequency modulation [33], which can be further improved by preprocessing the IC backside with a FIB (Focused Ion Beam) [34]. Recall that in addition to the technical capabilities, the foundry has unobstructed insights into the design without the need of expensive and error-prone reverse engineering (cf. Section II-A).

Note that an attacker only needs to probe static data. A locked IC only operates correctly as long as the correct chip key is input to the locking circuitry. In other words, the unlocking information has to be constantly present at the internal memory's output wires or a dedicated register (cf. [25]), i.e., it is a static signal. Thus, extracting a chip key or unlocking information is not temporally restricted, i.e., the adversary does not need control over the clock or other information about the timing of the device under attack.

a) A Note on Tamper-/Read-Proof Memory: While most publications do not address storage of the unlocking information, references [24] and [3] propose to use a tamper-proof memory. However, with respect to logic locking, read- or even tamper-proof memories are not suitable in general: In typical use cases for read-proof memory, e.g., to store cryptographic keys, data is read from the protected memory only when needed and cleared from internal registers as soon as possible. The



(a) Erroneous mask and resulting wafer print.

(b) Repaired mask and resulting wafer print.

Figure 3: Examples of mask repair, from [37].

exposure of sensitive data is limited to the bare minimum. For logic locking the chip key has to be constantly available throughout IC operation (cf. [25]). Hence, the static nature of the chip key invalidates the benefits of read-proof memory with respect to probing.

2) Attack Vector: Minimal Mask Modification: Producing lithographic masks is one of the most costly steps in modern IC manufacturing. However, making **minimal** adjustments to fabricated masks is feasible using mask repair techniques. With minimal mask modification, we refer to connecting selected individual signals to VCC/GND or creating small gaps in connections, effectively cutting a wire. Hence, these modifications can, for example, tie a signal to a known constant value. We explicitly do not require insertion of additional logic cells, which is a significantly more complex task.

Mask repair techniques commonly available to foundries include e-beams [35], [36], [37] or nano-machining via atomic force microscopy [38], which can even be used beyond 20 nm technology. Citing a senior product manager of Zeiss, a major provider of such equipment: "Our current system performance is significantly smaller than the claimed limit of 20 nm." [38]. Note that this statement stems from 2012, and it is reasonable to expect that an ever growing number of foundries is capable of performing the aforementioned minimal adjustments. The effectiveness of mask repair can be seen in Figure 3.

B. Non-Invasive Attacks

In previous work, non-invasive attacks have been the main methods used for analyzing the security of logic locking schemes. While these attacks are not the main focus of this work, we briefly review proposed techniques for the sake of completeness.

Most non-invasive attacks employ SAT-solving to find valid chip keys. To this end, access to the locked netlist in conjunction with black-box access to locked and unlocked ICs is required. The reason to employ a SAT solver is that exhaustively trying all key candidates and comparing I/O behavior with an unlocked IC is infeasible. A SAT solver can find a key which fits to all input/output pairs it has seen so far, but it does not guarantee correctness to other, unknown inputs. To verify that the found key is really consistent with all inputs is impractical.

The first SAT attack was presented in 2015 by Subramanyan et al. [13], effectively breaking all prior schemes that are not protected with a key preprocessor. What followed has been an ongoing arms race between new SAT-based attacks of minimal increment and corresponding countermeasures, where new defenses are immediately met with novel attacks and vice versa.

Recently, more sophisticated attacks such as the desynthesis attack [21] (cf. Section III-A3), or Functional Analysis against Logic Locking (FALL) [39] have been proposed. We recall from Section III-A3 that the desynthesis attack does not require an unlocked IC in order to disclose the chip key, hence it can be executed immediately. However, it requires knowledge on the synthesis tools employed by the design house. The FALL attack allows for more efficient extraction of chip keys via a two-stage process. First, the netlist is analyzed to narrow down the search space of key candidates. In the second stage SAT solvers are used to select a correct key from the smaller amount of remaining candidates. The authors reported to have successfully attacked even the most recent schemes, e.g., achieving a success rate of 81% against SFLL.

However, none of the available non-invasive attacks are able to defeat logic locking entirely. Critically, all SAT-based attacks neglect the potentially available key preprocessor that may retain security even if the chip key is compromised. The desynthesis attack requires knowledge on used synthesis tools and settings which would require yet another insider. Finally, the FALL attack, although promising, is still too far from the desired 100% success rate.

V. Evaluation

Thus far, we have classified existing schemes by their locking approach (cf. Section III) and discussed the capabilities of a malicious foundry as well as invasive and non-invasive attack vectors (cf. Section IV). We are now in a position to introduce a more comprehensive attacker model that also takes an adversarial foundry into account, including its physical capabilities. This enables us to demonstrate in Section V-B that current logic locking fails to defend against overproduction, regardless of the used scheme or key preprocessor.

A. Attacker Model for Foundry-level Adversaries

The key advantages of a foundry-level adversary include complete knowledge about the design's details down to the semiconductor level as well as physical access within the manufacturing process (cf. Section IV-A). These two points enable exploitation of the aforementioned invasive attack vectors. Hence, we argue that it is crucial for reliable security assessments not to restrict the attacker to black-box access. Therefore, the abilities of a foundry-level adversary have to be extended from **non-invasive attacks only** (as considered in previous work, cf. Section IV-B) to also include **invasive attacks** as discussed in Section IV-A. This gives rise to a more comprehensive attacker model introduced below that considers both, invasive and non-invasive approaches, and a definition of a successful attack.

1) Adversarial Capabilities: The adversary has access to several assets:

Multiple locked ICs, which can be obtained during the regular production process or, if needed, through overproduction.

Multiple unlocked ICs obtained on the market or directly in the foundry if remote unlocking is used (cf. Section III-C).

We note that in some use cases, e.g., military hardware, obtaining unlocked ICs can be a non-trivial task.

The lithographic masks used to manufacture the genuine ICs.

The gate-level netlist, which can be extracted from the GDSII/OASIS files (cf. Section II-A).

State-of-the-art manufacturing equipment and expertise, i.e., testing equipment and tools to perform invasive attacks available to modern foundries.

2) Defining a Successful Attack: The goal of an attacker is to successfully overproduce functional ICs. There are two principal ways to achieve this:

1) The adversary is able to unlock arbitrary locked ICs without authorization.
2) The adversary is able to remove or disable the locking scheme prior to fabrication (removal attack), thus creating a functionally equivalent but unprotected design.

We emphasize that Goal 1 is not necessarily achieved by recovering the internal chip key. If the attacker is unable to generate correct unlocking information that needs to be provided to the IC inputs, e.g., because of a key preprocessor, overproduction is still not successful.

If the locking scheme can be removed or circumvented prior to fabrication (Goal 2), the adversary can directly manufacture unprotected ICs. In the best case for the adversary, he can modify the lithographic masks that were used for the genuine order and start to overproduce. In the worst case, he has to generate entirely new masks, which is considerably more expensive than production with modified existing masks. Consequently, the threat of these removal attacks depends not only on technical aspects but also on the internal cost structure of the malicious foundry. For example, removal attacks may only be executed if the rewards are sufficiently high, e.g., the adversary is able to produce ICs that are of relevance for national security or the black market revenue is expected to amortize the high production costs of new masks.

B. Attacking Locking Schemes

We can now assess the security of existing schemes with respect to malicious foundry adversaries. Taking both, the attacker model defined above and the technical attack vectors described in Section IV, into account, we first discuss how to attack locking schemes without a key preprocessor and subsequently present attacks on the available key preprocessors.

1) Extraction Attacks: We recall that extraction of static values is possible for modern fabs, cf. Section IV-A1. This allows an adversarial foundry to target the chip key. If the locking scheme does not produce an individual chip key for each IC, an extraction attack can disclose the global chip key as illustrated in Figure 4 and any overproduced IC can be unlocked. Note that the vast majority of proposed schemes is based on a global chip key (cf. Table II).

In order to launch such an extraction attack, the adversary must be able to locate the chip key signals. It is crucial to observe that the structure of locking circuitry is very specific, and not commonly occuring in general VLSI design. The locking circuitry incorporates specific interconnections and unusual use of a large number of similar Boolean gates, which enables efficient automated identification. For example, in the widely used XOR-based locking schemes, there is a large number of X(N)OR gates where one input, i.e., the key input, is connected through minimal or no logic to a storage element, e.g., a Flip Flop (FF). Since all FFs that hold chip key bits are instrumented by common control signals, finding those chip key signals is straightforward: First, all X(N)OR gates that are preceded by a FF through minimal combinational logic are identified. Then, these preceding FFs are grouped by their interconnections (e.g., for shift registers), control signals, and possibly location information. After filtering for appropriate sizes, this is expected to leave only a single to a few groups which can then be manually inspected. The aforementioned steps can be easily automated with suitable tools, for example the open-source hardware analysis framework HAL [40]. Adaption to other schemes that are, for example, based on MUXes (Meerkat, CLIP) or LUTs (SFLL) is straightforward by identifying (groups of) respective elements.

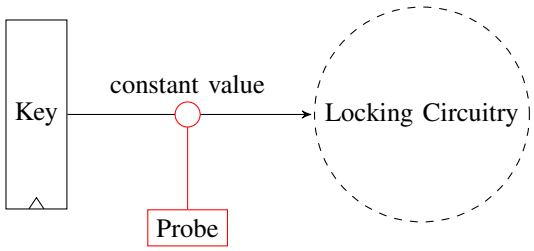


Figure 4: Simplified illustration of an extraction attack applicable to all locking scheme with a global chip key.

In summary, if used without a key preprocessor, all available locking schemes with a global chip key are insecure against an adversary that is capable of extraction attacks. Note that this approach does solely rely on the fact that probing the chip key is possible and that the key signals can always be quickly identified, but not on any properties of the individual locking schemes. Another prerequisite in the attacker model is that the adversary has access to an unlocked IC. This requirement is somewhat analog to a known-plaintext attack in cryptanalysis and has always been part of the attacker model in previous work. Unlocked ICs can be obtained, for example, from the open market or from an insider. Locking schemes with individual chip keys are not susceptible to mere extraction attacks, since obtaining the chip key of one IC does not provide any information on the chip key of other ICs. However, these schemes are inherently vulnerable to mask modification attacks as we will show in the next section.

2) Mask Modification Attacks: In order to facilitate individual unlocking information, a source of randomness, e.g., a TRNG or a PUF, is employed, whose output is used, for example, as a comparator value for the chip key or input to a key preprocessor. If it were possible to modify a design such that instead of the random output known fixed values were used, the locking scheme would essentially fall back to being deterministic. Note that this attack not necessarily discloses a chip key but makes all modified ICs unlock with the same unlocking information.

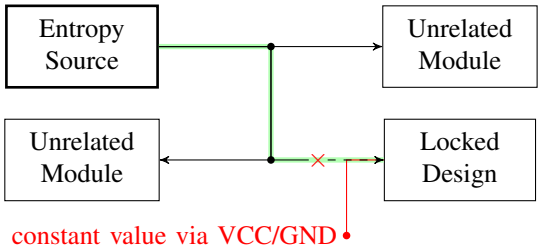


Figure 5: Illustration of a mask modification attack. Note that the genuine circuitry stays unaffected and that the modification is independent of locking scheme internals.

A malicious foundry can use minimal mask modification to achieve this goal as illustrated in Figure 5. First, the chip key signals are identified with the same strategy as for the aforementioned extraction attacks. Similarly, the entropy source (TRNG or PUF) can quite easily be identified due to its specific structure, e.g., the common use of analog components. The outputs of the entropy sources are traced until the chip key signals are hit (highlighted green in Figure 5). Finally, the paths are analyzed to find the wire split after which only the chip key is eventually affected. This marks the point ($\times$ in Figure 5) where a modification would not influence unrelated modules of the design, e.g., self tests. Hence, the found wires are selected for modification. As evident from Figure 5, fixing these wires to a constant signal is equivalent to overwriting the random input to the locked design with a constant. All ICs produced with the modified masks will be identical clones of each other with respect to the locking scheme, since any chip-unique random input has been changed to a common constant. Note that the adversary does not have to select a specific value for this constant since his goal is just to remove the "uniqueness" of each IC. It is important to recite that the manipulation only affects the locking scheme, while potential internal tests of the entropy source or other modules that use it stay unaffected. The same effect could also be achieved with other techniques, e.g., dopant changes in transistors as shown by Becker et al. [41], but for the sake of clarity, we stay with the mask manipulation technique for the remainder of this work. Again, note that the general approach does not rely on locking scheme details, but solely on the fact that random signal values can be turned into constants with mask modification.

Adapting this attack to CLIP, the only available scheme with individual chip keys, the output of the process variation sensors would be effectively overwritten by fixing their output signals all to GND, allowing to unlock all subsequently fabricated ICs with the all-zero key. However, recall that in the special case of CLIP no sophisticated attack is required, since the chip key can also be deduced directly via I/O testing due to the design of CLIP itself (cf. Section III-B).

3) Non-Invasive Attacks: In the literature, only non-invasive attacks against logic locking have been studied so far. This led to a wide range of SAT-solver based attacks and a few advanced approaches (cf. Section IV-B). However, most non-invasive attacks focus on the details of individual schemes, i.e., internal details, in order to be successful.

4) Summary: Attacks on Locking Schemes: All existing locking schemes can be broken with at least one of the presented techniques if used without a key preprocessor. However, in contrast to non-invasive attacks, which focus on scheme-specific properties, the invasive attacks presented in this work are derived from the capabilities of a malicious foundry and are **scheme-agnostic**. In other words, our attacks target a common fundamental design property of all locking schemes. For example, while new schemes may be discovered that are perfectly secure against SAT attacks, schemes with a global chip key will never be secure against a malicious foundry capable of extraction attacks. This result is summarized in Table IVa. While the existing non-invasive attacks are currently also able to break the plain locking schemes at high probability, no generic statement about their feasibility can be made since they are focused on scheme internals.

Hence, we can already conclude that **without a key preprocessor, none of the available schemes are secure**. Several authors noted in previous work that a key preprocessor should be used, especially since it retains security even if the chip key is disclosed. While this is true when considering only non-invasive attack, we show in the following that no existing key preprocessor protects against a malicious foundry capable of invasive attacks.

C. Attacking Key Preprocessors

In previous work only two key preprocessors have been presented, namely the EPIC key preprocessor [5], [6] and LECs [16], cf. Section III-C,

1) LECs: An interesting aspect of LECs is the use of a PUF. The idea of using PUFs for logic locking seems promising, as they are unique due to process variations and can, hence, provide every chip with individual unlocking information. However, when working out the specifics, several weaknesses arise. The only details given in [16] are that there is a PUF which receives a challenge and outputs a response. This mirrors the structure of a strong PUF, although no helper data is explicitly mentioned which is typically applied in the reconstruction step of a strong PUF. The PUF output is XORed to the user key (cf. Section III-C) to generate the chip key. Hence, in order to construct a valid user key, the design house has to know the PUF response for the chosen challenge, i.e., an interface to query the PUF is required. However, a malicious foundry could use this very interface in a similar way for an attack: first, the chip key of the underlying scheme is recovered, i.e., via a non-invasive or invasive attack. The attacker can then unlock arbitrary ICs by using the PUF interface to query one challenge and computing the LEC user key from the known chip key and the PUF response, thus generating valid unlocking information. Alternatively, the adversary can model the PUF of each IC via machine learning [42], [43]. He then obtains an unlocked IC and uses the corresponding model to compute the PUF response for the respective IC's challenge, which in turn reveals the chip key through an XOR with the user key and enables the aforementioned attack.

We emphasize that these attacks are based on our assumptions regarding the PUF instantiation. Unfortunately, [16] does not provide definite detail, hence we have to consider LECs unusable in their current state and neglect them in the following.

2) EPIC's Key Preprocessor: Recall that the EPIC key preprocessor uses asymmetric cryptography to allow for individual unlocking information for each IC, even if the underlying locking scheme has a global chip key. Furthermore, the unlocking information contains not only the encrypted chip key but also a digital signature for authentication. We will first analyze the potential of each individual attack method before combining them to successfully attack EPIC's key preprocessor.

a) Non-Invasive Attack: The EPIC key preprocessor relies on the security of RSA and exposes only its own public key and the public key of the design house, hence it is not susceptible to non-invasive attacks. This is, of course, assuming that the implementation does not leak exploitable side-channel information. Otherwise, side-channel attacks may lead to disclosure of the IC's private key [44], however these attacks are out of the scope of this work.

Note that this immunity to non-invasive attacks only applies to the key preprocessor. If the underlying locking scheme is susceptible to non-invasive attacks, the chip key may still be disclosed. However, knowledge of the chip key alone does not enable an attacker to generate valid unlocking information in presence of the EPIC key preprocessor because of the signature check as already mentioned in [5], [6]. Therefore, disclosure of the chip key was not regarded as a security issue.

b) Extraction Attack: EPIC does not provide details on whether the unlocking information is stored directly on-chip and the key preprocessor is envoked with each power-up, or whether after unlocking only the chip key is stored. In both cases, an extraction attack against EPIC's key preprocessor can eventually reveal the chip key. If the chip key is stored, it can be extracted directly. If only the unlocking information is stored, the chip key can be extracted from the key preprocessor's output wires. Another way would be to extract the unlocking information itself as well as the IC's internal RSA key pair by probing the fuses (as shown for eFuses on FPGAs in [45]). This key pair can then be used to decrypt the chip key from the unlocking information. However, as explained in the previous paragraph regarding non-invasive attacks, knowledge of the chip key alone is not sufficient to directly unlock other ICs because of the digital signature. We will show later that a combination of extraction attack and mask modification attack is indeed successful.

c) Mask Modification Attacks: Generally speaking, mask modification can overwrite a random value with a constant value (cf. Figure 5), comparable to the approach in [41]. Adapting this to the EPIC key preprocessor circumvents the randomness used to generate the internal RSA key pair. Then, all ICs use the same attacker-chosen key pair and ultimately accept the same unlocking information. This can also be achieved by targeting the fuses where the RSA key pair is stored: by fixing the output signals of said fuses to constant values on mask level, all ICs share the same key pair. Note that without valid unlocking information this attack alone does not fully break the scheme. However, EPIC suggests remote unlocking, where upon request the design house transmits valid unlocking information for this IC. By requesting to unlock a single modified ICs the attacker can then unlock all "clones" of that IC with the same data.

Another attack target can be the signature verification mechanism for authenticity of the unlocking information. No matter how the verification is implemented, it eventually comes down to a binary decision. By forcing this signal to always-true, every unlocking information passes signature verification. Note that this only disables authentication, not the locking mechanism itself and that this attack alone again does not fully break the scheme.

d) Combining Attacks: As shown above, mask modification attacks can successfully disable the benefits of EPIC's key preprocessor if remote unlocking is used. If unlocking is performed solely back at the design house, any of the aforementioned attacks alone are not enough for a successful attack. However, combining mask modification with an attack that discloses the chip key invalidates the EPIC key preprocessor even in that scenario. Extraction attacks or non-invasive attacks can be used to obtain the chip key, either via attacking the key preprocessor as described in the respective paragraphs or the underlying locking scheme (cf. Section V-B). The adversary either fixes the input signals to the chip key register to the the extracted chip key or, with only a single modification, disables signature verification which enables the forgery of unlocking information knowing the chip key. The attack vectors for both scenarios are summarized in Table III. While in [5], [6] an attack that modifies masks was regarded as not realistic, we argue in Section IV-A2 that minimal mask modification is not only a viable technique but also widely used method in modern semiconductor manufacturing. It is also worth noting that the mask modification takes place after the legitimate order has been processed, hence the design house will receive genuine ICs without modification.

Unlocking Scenario	Attack
Remote Activation	Mask Modification
In-house Activation	Mask Modification + (Extraction or Non-Invasive)

Table III: Summary of required techniques to facilitate attacks against EPIC's key preprocessor.

e) Summary: Attacks on Key Preprocessors: In contrast to the plain locking schemes without a key preprocessor, non-invasive attacks alone cannot be used to successfully attack EPIC's key preprocessor. However, depending on whether ICs are unlocked remotely or at the design house, EPIC's key preprocessor can be attacked when invasive attacks are taken into account. As a result, Table IVb highlights that **regardless of the underlying locking scheme**, invasive attacks can fully circumvent EPIC's key preprocessor.

VI. Conclusions — The End of Logic Locking

The starting point of this work was the fact that the actual physical capabilities of a foundry-level adversary have been overlooked in the literature. A malicious foundry, as the main threat regarding overproduction, is capable of several invasive attacks in addition to the non-invasive attacks considered in

(a) Security of logic locking schemes without a key preprocessor against various adversary capabilities.

Chip Key	Extraction	Mask Mod.	Non-Invasive
Global	ϟ	🛡	(ϟ)
Individual	🛡	ϟ	(ϟ)

(b) Security of logic locking schemes with EPIC's key preprocessor against various adversary capabilities.

Chip Key of Underlying Scheme	Invasive	Non-Invasive
Global	ϟ	🛡
Individual	ϟ	🛡

Table IV: Summary of the security of logic locking against with respect to various adversary capabilities. A 🛡 indicates *not vulnerable* while a ϟ indicates *vulnerable*.

previous work. We showed that, without a key preprocessor, all available schemes are susceptible to non-invasive attacks developed in earlier work, but also to invasive attacks considered here. It is noteworthy that, somewhat counterintuitively, it does not matter whether the scheme incorporates a global or individual chip keys. As a key contribution, we showed that, in contrast to existing non-invasive attacks, our invasive attacks apply to the foundations of logic locking schemes and not to design specifics. Thus far, it was believed that EPIC's key preprocessor provides protection even when a chip key is disclosed. We showed that via invasive attacks, a malicious foundry is able to invalidate the benefits of EPIC's key preprocessor. Table V aggregates these results. Crucially, no combination of scheme and key preprocessor provides sufficient security against a malicious foundry in a realistic setting.

Scenario	Non-Invasive [Previous Work]	Invasive and Non-invasive [This Work]
Plain Logic Locking Schemes	ϟ	ϟ
Scheme + Key Preprocessor	🛡	ϟ

Table V: Condensed overview on the security of logic locking against invasive and non-invasive attacks. A 🛡 indicates *not vulnerable* while a ϟ indicates *vulnerable*.

a) Generalizing Our Findings: The results at hand raise the question whether it is possible at all to thwart overproduction with the current logic locking approaches. We showed that schemes which use only global chip keys are always vulnerable to extraction attacks. Furthermore, it is safe to say that any kind of dynamic logic locking key management requires a source of randomness. We argued that it can be reasonably assumed that the output of such building blocks can always be meaningfully overwritten via mask modification. Hence, invasive attacks seem to be fatal against the underlying building blocks of both locking schemes and key preprocessors in all available configurations.

We therefore conclude that logic locking, while certainly forcing a malicious foundry to perform additional steps, will never be a successful countermeasure against overproduction by a determined modern foundry.

References

[1] KPMG, "Managing the Risks of Counterfeiting in the Information Technology Industry," Online, 2006, https://www.agmaglobal.org/uploads/whitePapers/KPMG-AGMA_ManagingRiskWhitePaper_V5.pdf.

[2] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," Proceedings of the IEEE, vol. 102, no. 8, pp. 1207–1228, Aug 2014.

[3] M. Yasin and O. Sinanoglu, "Evolution of Logic Locking," in Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on. IEEE, 2017, pp. 1–6.

[4] B. Lippmann, M. Werner, N. Unverricht, A. Singla, P. Egger, A. Dübotzky, H. Gieser, M. Rasche, O. Kellermann, and H. Graeb, "Integrated Flow for Reverse Engineering of Nanoscale Technologies," in Proceedings of the 24th Asia and South Pacific Design Automation Conference. ACM, 2019, pp. 82–89.

[5] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," in DATE, 2008, pp. 1069–1074.

[6] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," Computer, vol. 43, no. 10, pp. 30–38, 2010.

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

[8] J. Dofe and Q. Yu, "Novel dynamic state-deflection method for gate-level design obfuscation," IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 37, no. 2, pp. 273–285, 2018.

[9] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in USENIX Security Symposium, 2007.

[10] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking obfuscation for anti-tamper hardware," in CSIIRW. ACM, 2013, p. 8.

[11] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the Difficulty of FSM-based Hardware Obfuscation," IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, no. 3, pp. 293–330, Aug. 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7277

[12] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," IEEE Trans. on CAD of Integrated Circuits and Systems, pp. 961–971, 2015.

[13] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015, 2015, pp. 137–143.

[14] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," IEEE Design & Test of Computers, pp. 66–75, 2010.

[15] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karrl, "Security Analysis of Logic Obfuscation," in The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012, 2012, pp. 83–89.

[16] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic Encryption: A Fault Analysis Perspective," in 2012 Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 953–958.

[17] W. P. Griffin, A. Raghunathan, and K. Roy, "CLIP: Circuit Level IC Protection Through Direct Injection of Process Variations," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, no. 5, pp. 791–803, 2012.

[18] S. Dupuis, P. S. Ba, G. Di Natale, M. L. Flottes, and B. Rouzeyre, "A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans," in 2014 IEEE 20th International On-Line Testing Symposium (IOLTS), 2014, pp. 49–54.

[19] J. Rajendran et al., "Fault Analysis-Based Logic Encryption," IEEE Trans. Computers, pp. 410–424, 2015.

[20] M. Yasin, J. J. V. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," IEEE Trans. on CAD of Integrated Circuits and Systems, pp. 1411–1424, 2016.

[21] M. E. Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic Locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism," arXiv preprint arXiv:1703.10187, 2017.

[22] R. Karmakar, N. Prasad, S. Chattopadhyay, R. Kapur, and I. Sengupta, "A New Logic Encryption Strategy Ensuring Key Interdependency," in 30th International Conference on VLSI Design and 16th International Conference on Embedded Systems, VLSID 2017, Hyderabad, India, January 7-11, 2017, 2017, pp. 429–434.

[23] M. Yasin et al., "What to lock?: Functional and parametric locking," in Proceedings of the on Great Lakes Symposium on VLSI 2017, Banff, AB, Canada, May 10-12, 2017, 2017, pp. 351–356.

[24] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory to Practice," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, pp. 1601–1618.

[25] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "ATPG-based Cost-Effective, Secure Logic Locking," in VLSI Test Symposium (VTS), 2018 IEEE 36th. IEEE, 2018, pp. 1–6.

[26] H. Zhou, Y. Shen, and A. Rezaei, "Vulnerability and remedy of stripped function logic locking," Cryptology ePrint Archive, Report 2019/139, 2019, https://eprint.iacr.org/2019/139.

[27] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Fpga intrinsic pufs and their use for ip protection," in International workshop on cryptographic hardware and embedded systems. Springer, 2007, pp. 63–80.

[28] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE. IEEE, 2007, pp. 9–14.

[29] R. Torrance, "The State-Of-The-Art in IC Reverse Engineering," in CHES. Springer, 2009, pp. 363–381.

[30] R. Schalangen, R. Leihkauf, U. Kerst, and C. Boit, "Backside E-Beam Probing on Nano Scale Devices," in Test Conference, 2007. ITC 2007. IEEE International. IEEE, 2007, pp. 1–9.

[31] C. Boit, C. Helfmeier, D. Nedospasov, and A. Fox, "Ultra high precision circuit diagnosis through seebeck generation and charge monitoring," in Physical and Failure Analysis of Integrated Circuits (IPFA), 2013 20th IEEE International Symposium on the. IEEE, 2013, pp. 17–21.

[32] U. Kindereit, "Fundamentals and future applications of laser voltage probing," in Reliability Physics Symposium, 2014 IEEE International. IEEE, 2014, pp. 3F–1.

[33] S. Tajik, H. Lohrke, J.-P. Seifert, and C. Boit, "On the power of optical contactless probing: Attacking bitstream encryption of fpgas," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, pp. 1661–1674.

[34] C. Boit, C. Helfmeier, and U. Kerst, "Security risks posed by modern ic debug and diagnosis tools," in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on. IEEE, 2013, pp. 3–11.

[35] K. Edinger, H. Becht, R. Becker, V. Bert, V. A. Boegli, M. Budach, S. Göhde, J. Guyot, T. Hofmann, O. Hoinkis, A. Kaya, H. W. Koops, P. Spies, B. Weyrauch, and J. Bihr, "A novel electron-beam-based photomask repair tool," Proc.SPIE, vol. 5256, pp. 5256 – 5256 – 10, 2003. [Online]. Available: https://doi.org/10.1117/12.532866

[36] K. Edinger, H. Becht, J. Bihr, V. Boegli, M. Budach, T. Hofmann, H. W. P. Koops, P. Kuschnerus, J. Oster, P. Spies, and B. Weyrauch, "Electron-beam-based photomask repair," Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena, vol. 22, no. 6, pp. 2902–2906, 2004. [Online]. Available: https://avs.scitation.org/doi/abs/10.1116/1.1808711

[37] T. Bret, R. Jonckheere, D. Van den Heuvel, C. Baur, M. Waiblinger, and G. Baralia, "Closing the Gap for EUV Mask Repair," in Extreme Ultraviolet (EUV) Lithography III, vol. 8322. International Society for Optics and Photonics, 2012, p. 83220C.

[38] E. Sperling, "Mask Repair Enters The Spotlight," Online, October 2012, https://semiengineering.com/mask-repair-enters-the-spotlight/.

[39] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," CoRR, vol. abs/1811.12088, 2018. [Online]. Available: http://arxiv.org/abs/1811.12088

[40] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm, T. Weidlich, R. Tessier, and C. Paar, "HAL-The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion," IEEE Transactions on Dependable and Secure Computing, 2018.

[41] G. T. Becker et al., "Stealthy Dopant-Level Hardware Trojans," in CHES. Springer, 2013, pp. 197–214.

[42] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010, pp. 237–249.

[43] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "Puf modeling attacks on simulated and silicon data," IEEE Transactions on Information Forensics and Security, vol. 8, no. 11, pp. 1876–1891, 2013.

[44] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in Annual International Cryptology Conference. Springer, 1999, pp. 388–397.

[45] S. M. Trimberger and J. J. Moore, "Fpga security: Motivations, features, and applications," Proceedings of the IEEE, vol. 102, no. 8, pp. 1248–1265, 2014.

[46] S. Dupuis, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Identification of Hardware Trojans triggering signals," in First Workshop on Trustworthy Manufacturing and Utilization of Secure Devices, 2013.

[47] C. Paar and J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners. Springer Publishing Company, Incorporated, 2009, p. 153.

APPENDIX

We gave a short summary of previous work on logic locking (cf. Table I) in Section III. In this section, we present the succeeding schemes to EPIC in more detail and highlight the main focus and contribution of each paper.

Preventing IC Piracy Using Reconfigurable Logic Barriers [14]

Paper focus: type and placement of locking circuitry

In 2010, Baumgarten et al. published the first follow-up paper to EPIC. Instead of using X(N)OR gates, the authors propose "reconfigurable logic barriers", which means that they add look-up tables into the design which will be programmed after fabrication. These "micro FPGAs" protect the original design against reverse engineering as they are only programmed after fabrication, hence, part of the original circuit would be missing in a locked copy. The authors argue that in contrast to EPIC, where the foundry fabricates a design which contains the whole functionality obstructed by the locking gates, their approach offers better protection since parts of the circuit are simply missing. Using multi-input LUTs allows for a larger state space than a single XOR gate, hence, it becomes exponentially more complicated to find the correct configuration.

The authors also advocate the use of public-key cryptography to allow for a secure communication between designer and foundry to activate the circuits after fabrication and refer to EPIC's key preprocessor. Additionally, the authors propose an algorithm to decide at which positions logic barriers should be implemented to minimize the risk of "bypassing the locks or guessing the correct configuration".

Security Analysis of Logic Obfuscation [15]

Paper focus: placement of locking circuitry

Rajendran et al. focused solely on the positioning of key gates. They identified four different ways to place key gates in the circuit, namely (1) runs of key gates, (2) isolated key gates, (3) dominating key gates, and (4) convergent key gates.

A *run of key gates* is characterized by two or more key gates which are chained together, i.e., the output of one key gate is the input to the next one. For an attacker, this reduces the search space as he can logically collapse this run to a single key gate.

If there exists only one key gate between an input and an output, this gate is called *isolated key gates*. The attacker can then simply apply a suitable input pattern such that the output reveals the key bit.

Similarly, a *dominating key gate* is a key gate which lies on every path between another key gate and the outputs. Only

if an attacker can mute the effect of the other key gate, it is possible to determine the value for the dominating key gate. As this is may not always be possible, dominating key gates can sometimes only be attacked by brute-force attacks.

The last class is *convergent key gates* where no "forward" paths between the key gates exist until the output of both key gates eventually meet again further along the paths. Hence, these key gates still influence one another. The attack strategy of such a construct is again to determine the correct value for one key gate by muting the other one. If the convergent key gates are non-mutable, the remaining attack strategy is brute-force.

Hence, the defender's goal is to maximize the number of non-mutable key gates for convergent and dominating key gates. Rajendran et al. introduce an algorithm which initially inputs 10% of the key gates at random and then finds the optimal position for the remaining key gates step-by-step.

Logic Encryption: A Fault Analysis Perspective [16]

Paper focus: placement of locking circuitry

In 2012, Rajendran et al. used fault simulation techniques to enhance logic locking: to make a key gate effective, an incorrect key bit must definitely result in a wrong output, otherwise, an attacker might be able to use a wrong key and still have a fully functional circuit. In other words, wrong outputs have to be provided for all input vectors if the chip key is incorrect. The authors argue that, when key gates are inserted at random, the effect of a wrong key bit might not be propagated to the output because its effect is masked by another (wrong) key bit (cf. [15]). The authors compare this problem to "fault propagation analysis", where the effects of a fault on the whole system is modeled. In these models usually only a single stuck-at fault is considered simultaneously, hence, the authors overcome this challenge by iteratively allowing another fault. The result is an algorithm which inserts key gates depending on their "fault impact", i.e., the impact a wrong key bit at this position has. The aim is to guarantee not only wrong outputs if a wrong key is applied, but also a Hamming distance of 50% between wrong outputs and correct ones, maximizing ambiguity for the attacker. Although only experimentally tested for XOR and XNOR gates, the authors state that their algorithm also works for other types of key gates such as AND, OR, inverter and MUX gates.

A second contribution of the paper — though mentioned only briefly — is the logic encryption cell (LEC) which the authors suggest can be used instead of EPIC's RSA engine in order to generate individual unlocking information. A LEC consists of a PUF and several XOR gates, and works as follows: the circuit receives a PUF challenge which is directly fed into the PUF and a user key. The PUF response and the user key are then XORed to generate the chip key which unlocks the circuit. Naturally, such a construct occupies way less area than a RSA decryption unit, and the authors argue that the security level is equivalent to the one presented in EPIC.

CLIP: Circuit Level IC Protection Through Direct Injection of Process Variations [17]

Paper focus: type and placement of locking circuitry, focus on individual chip keys

CLIP is a combinational logic locking scheme presented in 2012 which takes a different approach. Interestingly, this scheme is not included in the bibliography of most subsequent publications. In contrast to the other proposed schemes, it is the only scheme with an individual chip key per device and does not insert key gates comparable to the other schemes. Each IC is equipped with multiple process variation sensors which reliably generate a 0 or 1 bit, unique to each device. These sensors are basically small PUFs, spread over the entire chip. In order to protect a combinational function f, CLIP locally duplicates the function and feeds inputs in a special de-multiplexer. Simplified, this module forwards the original input signals to one of the instances of f, and outputs different values to the other instance. Which of the instances receives the original inputs is selected via a sensor output. Hence, one instance of f will compute wrong results while the other instance outputs correct values. An output multiplexer decides based on one bit of the chip key, which of the two computations becomes the final output. As a result, only if the key bit selects the output of the f-instance which gets unmodified inputs, the circuit behaves correctly.

Since each chip produces its own individual chip key, even the designer has no way of knowing a device's chip key beforehand. However, the authors demonstrate a technique to determine a minimal test set of input signals, that enable deduction of the sensor values from the locked chip's output signals. The designer has to query each produced chip with this test set in order to recover the chip key and subsequently unlock the device.

A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans [18]

Paper focus: type and placement of locking circuitry

Dupuis et al. proposed a logic locking scheme based on AND resp. OR gates in 2014. Additionally to the goals of logic locking, the authors also explicitly try to thwart the insertion of hardware Trojans. Hence, instead of randomly placing key gates into the design, their approach consists of the following three steps.

The first step is to identify rarely triggered signals, i.e., signals with a low switching probability. Such signals are often used as triggers for hardware Trojans, since they are unlikely to be found while testing. This step is done by propagating the switching probability of each signal throughout the circuit, i.e., from input to output, as described by the same authors in [46]. Key gates are placed as predecessors of signals with low switching probability. In order to keep the timing specifications of the original circuit, the second step is to choose the signal whose slack time allows for the addition of another gate. Then, in the last step, the probability of the chosen signal decides whether an AND gate – if it has a high probability to be '1' –

or an OR gate – for a high probability that it is '0' – is added to the circuit. The designers know that for an OR gate, the corresponding key bit is '0', and for an AND gate it is '1', respectively. They claim that for an attacker, the key input has a probability of $\frac{1}{2}$ to be either '0' or '1'. Then, by design, the key gates change the probability of the rarely triggered signals significantly (cf. Figure 4 of [18]). Again, as all circuits produced by this design have the same key, the authors refer to other solutions for individualizing the circuits as in [5] and [16].

Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking [12]

Paper focus: type and placement of locking circuitry

The work of Plaza and Markov describes logic locking based on multiplexer gates. According to the authors, one of the main drawbacks of EPIC-based logic locking techniques is that the circuit needs to be activated before it can be functionally tested, i.e., its unlocked functionality is known during the testing phase. They show an iterative attack that can derive the secret chip key by "gradually improving" a random key candidate according to the test responses. Hence, in order to prevent this kind of attack, a logic locking algorithm that preserves test responses and allows for testing of a locked circuit is introduced. A core feature of this approach is that for all possible key combinations, the test responses are identical. This comes at the cost of pre-computation and simulation to find suitable positions for the key gates. Empirical validation shows that the overhead of finding locking candidates is in the range of seconds and since it needs to be performed once for each design, is negligible.

Fault Analysis-Based Logic Encryption [19]

Paper focus: placement of locking circuitry

Rajendran et al. further elaborated on fault-based logic locking in 2015, based on [16]. They enhanced their algorithm to find a suitable position with a strong fault impact. By adding a modification phase that exchanges XOR to XNOR gates according to the chip key and randomly inserts inverters, the attacker is prevented from easily identifying which gates belong to the original circuit and which are part of the locking. Rajendran et al. extend their fault-based locking technique to work on MUX gates, showing that XOR/XNOR gates perform slightly better under their Hamming distance model using less gates with a more reliable fault excitation. Experimental results show that for larger designs, achieving a Hamming distance of 50% between correct and faulty outputs is easily possible for fault-based logic locking, while this is not guaranteed for random insertions.

On Improving the Security of Logic Locking [20]

Paper focus: placement of locking circuitry

In their 2016 contribution, Yasin et al. propose an attack against existing logic locking schemes which is heavily based on the research of Rajendran et al. in 2012 [15]. The attacker is in possession of an unlocked circuit and queries it with selected input patterns in order to deduce single bits of the chip key from the outputs. This is possible since randomly inserted key gates are often either isolated or blocking each other. Both can be exploited by the attacker until the remaining key space is small enough to enable brute-force attacks. For each class of key gates listed in [15], the authors elaborate on how an attack is performed. Afterwards, they improve the algorithm of the previous paper which inserts the key gates such that aforementioned attacks are thwarted, e.g., by more carefully selecting the position of the first key gate as it is the base for the remainder of the algorithm. In an experimental phase, they then analyze the new "enhanced strong logic locking" and show that the attacks presented in the beginning of the paper are no longer applicable. To secure strong logic locking (SLL) against SAT-based attacks, *One-way Random Functions* [47] can be used. Experimental results show that with this combination, execution time of SAT-based attacks grows exponentially.

Logic Locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism [21]

Paper focus: type and placement of locking circuitry, applying locking during synthesis

El Massad et al. present a new logic locking technique called "Meerkat", which includes logic locking into the synthesis step as opposed to locking a synthesized netlist. Meerkat is designed to be secure against a new attack the authors have developed, the *desynthesis attack*. The authors assume that the attacker with access to the locked netlist can distinguish key inputs from regular inputs and knows which synthesis tool and locking technique have been used. Note that the attacker does not need access to a functional circuit. The goal is then to discover several bits of the secret key. The attacker identifies the Boolean function which is executed when applying a key candidate. He then tries to re-lock that function by adding key gates according to the rules of original locking mechanism, re-synthesizes this design, and checks whether the result is equal to the original locked netlist. If not, he can rule out the corresponding key. As "brute-forcing" all key candidates is infeasible, the authors present an algorithm which adopts a greedy search heuristic that improves a random key guess iteratively. They evaluate the desynthesis attack on 100 netlists which were locked using [5] or [15]. Their results show that the desynthesis attack is able to recover a large part of the key bits correctly, exceedingly better than random guessing.

To thwart the desynthesis attack, Meerkat uses reduced ordered binary decision diagrams (ROBDDs) to introduce key bits prior to synthesis. The interested reader is referred to the original paper for a detailed description. The resulting MUX-based netlist can then be synthesized to output the final netlist. Meerkat has a considerable overhead in area (and delay), but the authors argue that provable security naturally comes with a cost.

A New Logic Encryption Strategy Ensuring Key Interdependency [22]

Paper focus: placement of locking circuitry

In 2017, Karmakar et al. combine two previous works of Rajendran et al. [19] and Yasin et al. [20] to create a logic locking technique with the best key gate positions. Their logic locking approach is based on fault analysis, while at the same time inserting some of the key gates according to the rules of SLL. In a further step, a key-dependency block is added to increase the dependency of the key gates. The unlocking information is fed into this module, consisting of several XOR/XNOR gates, while its output, i.e., the chip key, is connected to the key gates of the circuit. Finally, the authors provide experimental results, stating that "although area, power and delay overheads are a bit higher than the conventional logic encryption", their method is the first which provides a high output corruption for wrong keys, while at the same time enforcing a high dependency among the key gates.

What to Lock? Functional and Parametric Locking [23]

Paper focus: type and placement of locking circuitry

With TTLock, Yasin et al. present a novel logic locking scheme which aims to protects against current attacks such as SAT-based attacks. The idea is to minimally modify the original logic cone such that for one input pattern, the output of the circuit is inverted. A restore unit reverts this inversion by XORing a flip signal to the modified circuit's output iff the correct key is present. Otherwise, the flip signal causes an additional inversion for an originally correct output. While TTLock only allows for a single protected input combination, it was later refined and renamed to SFLL (cf. next paragraph).

Provably-Secure Logic Locking: From Theory to Practice [24]

Paper focus: type and placement of locking circuitry

Yasin et al. describe the motivation of their 2017 work as to "develop a logic locking technique that can withstand all known and anticipated attacks", i.e., SAT attacks as well as removal attacks and reverse-engineering. The idea is based on TTLock [23] by the same authors: the goal is to only protect specific input patterns for which a wrong chip key will result in incorrect outputs, i.e., the circuit behaves correctly for unprotected input patterns, regardless of the supplied chip key. Therefore, the original logic is modified such that the output of one protected input pattern is inverted. This can be compared to a built-in fault in the design. A reverse engineer can only acquire the design which is stripped of parts of its functionality, hence, the authors refer to their technique as Stripped-Functionality Logic Locking (SFLL). To restore complete functionality, an additional restore unit corrects the inserted errors if the correct chip key is present. For unprotected input patterns the restore unit has no effect.

Depending on the number and specification of the input cubes to be protected, different variants of SFLL exist: SFLL-HD^h protects all input patterns which have Hamming distance h to the chip key. The other variant is SFLL-flex$^{c \times k}$, which allows the designer to specify the individual input patterns to be protected. In this variant, c protected k-bit input patterns are stored in the restore unit together with their restore signal (called flip signal), which is implemented as a LUT in tamper-proof memory. Whenever the input to the circuit matches an entry of the LUT, the restore unit outputs the corresponding flip signal, which is XORed to the output of the stripped circuit to repair the built-in error. The authors evaluated security against SAT attacks and showed that the number of DIPs required for a successful attack grows exponentially with the key lentgh for both SFLL-HD and SFLL-flex. Finally, they show that SFLL is more secure against known attacks than existing schemes.

ATPG-based Cost-Effective, Secure Logic Locking [25]

Paper focus: type and placement of locking circuitry

In 2018, Sengupta et al. introduce SFLL-Fault, a framework which reduces implementation cost of the original work on SFLL by Yasin et al. [24] by 35%. The main idea is to use fault injections to find locations at which circuit functionality can be stripped most cost-effectively. A stuck-at fault is injected at a node of the original circuit, which in turn allows for further optimization since part of the remaining logic now carry constant signals. Now, for some inputs, the circuit is no longer functional. These inputs become the protected input patterns, hence the SFLL restore unit will resolve the original stuck-at fault. ATPG tools can be used to identify these patterns. In order to find the best circuit in terms of area, the authors present an algorithm which produces a stuck-at fault at every possible node and then computes the corresponding restore LUT. Upon termination, the algorithm presents the overall optimal circuit with respect to the desired level of security. Concerning security the authors argue that SFLL-fault behaves similarly to SFLL-flex. They also present detailed experimental results highlighting the effectiveness of SFLL-fault in comparison with SFLL-flex.

Vulnerability and Remedy of Stripped Function Logic Locking [26]

Paper focus: type and placement of locking circuitry

In 2019, Zhou et al. reviewed SFLL [25] and presented the bit-coloring attack: knowing only a single protected input pattern, the authors show that it is possible to deduce the secret key in polynomial time. Bits of the known input pattern are then flipped, and by querying an activated circuit to verify whether this new input is also a protected input pattern, these flipped bits are sorted into two groups, accordingly. As a result, one group consists of all bits needed to be flipped for the original protected input in order to derive the secret key. Note that although not explicitly said, their attack apparently focuses on the SFLL-HD. As a remedy, Zhou et al. advocate the use of

one-way functions in the input path of the primary inputs to the restore unit (cp. [25]).