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

Anonymous Submission

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. Unlocking is handled by the designer only after production, hence an adversarial foundry should not be able to unlock overproduced chips.**

In this work, we highlight major shortcomings of proposed logic locking schemes. They exist primarily due to the absence of a well-defined and realistic attacker model in the current literature. We characterize the physical capabilities of adversaries, especially with respect to invasive attacks and a malicious foundry. 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, which is accompanied by two case studies, shows that none of the previously proposed logic locking schemes 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* have been proposed over the last years, including publications in the major conferences, e.g., USENIX [3] and CCS [4]. 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. Only the holder of the Intellectual Property (IP) rights who is in possession of that key should be able to unlock the IC. Hence, although possessing all information required to fabricate the integrated circuit, a malicious foundry lacks the secret key to unlock any overproduced ICs and subsequently sell them on the black market. Likewise, plain reverse engineering yields a locked design, i.e., the original IP is obfuscated to some extend. 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 in the literature so far.

Contributions: The work at hand provides three main contributions:

1) Based on systematic analysis of the attacker models and protection goals in previous work, we identify the lack of real-world physical capabilities of adversaries in existing attacker models. Therefore, we introduce a realistic attacker model for logic locking schemes which is not restricted to non-invasive attacks (as in previous work) but also considers invasive approaches. For a sound assessment of security, we include a precise definition of goals for a successful attack.
2) We develop two generic attack methodologies that target the foundations of all proposed locking schemes. Our attacks are based on *probing* and *minimal mask modification* as attack vectors. We support the attacks with experimental case studies that demonstrate semi-automated identification of points of interest.
3) As the main contribution, we generalize our findings for virtually any logic locking scheme. The generic applicability of our attacks renders all existing countermeasures insufficient. Finally, we argue that logic locking will most likely never succeed against a determined adversary.

Outline: The remainder of this work is structured as follows: we start with background information on logic locking and introduce important terminology in Section II. Next, we summarize prior work on attacks on logic locking in Section III. In Section IV, we briefly categorize existing schemes and identify two underlying principles that are common to all existing schemes. With the current state of logic locking laid out, we analyze and revise the existing attacker model(s) in Section V. Based on the new model, we introduce two invasive attack vectors in Section VI before using them to demonstrate attacks in Section VII. We conclude our work and discuss the general impact of our this work in Section VIII.

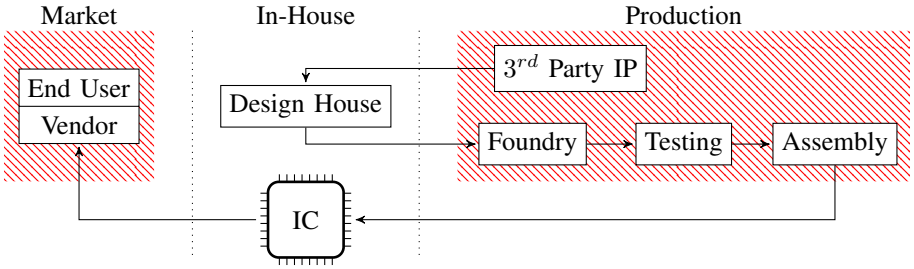


Figure 1: Simplified view on an exemplary, outsourced design fabrication process. Every entity hatched red is untrusted and can potentially be malicious.

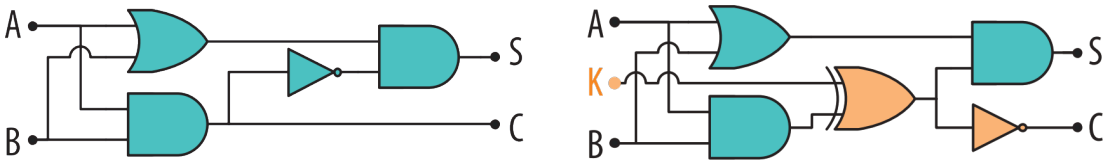


Figure 2: An example of combinational logic locking with EPIC as depicted in [5]. Left is the original design and right the locked result.

II. Background on 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 not within the direct control of the design house, they must be considered potentially malicious environments. Every external entity in the fabrication chain is hence *untrusted* (shaded red in Figure 1).

a) Protection Goals: 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. The primary focus of logic locking is to prevent overproduction, i.e., only the design house should be able to control the (amount of) ICs available on the market. While some schemes, e.g., [5], [6], [7], [4], also mention a protection against IP-theft through reverse-engineering, i.e., by providing obfuscation, this is not a primary goal of logic locking that **all** schemes try to accomplish.

b) General Locking Procedure: Simplified, logic locking extends the existing design with a dedicated locking circuitry. This additional logic is closely intertwined with existing cells and affects the overall IC functionality through a key. If the correct key is given, the IC works as intended. However, for an incorrect key, the IC malfunctions and thus cannot be used. Said key is only known to the design house/IP-rights holder and is inserted after fabrication in non-volatile on-chip memory. Therefore, in theory, no malicious entity in the supply chain is able to sell overproduced ICs, since they are simply not functional before reaching the design house again.

More specifically, logic locking can be divided into two major categories: combinational and sequential logic locking.

B. Combinational Logic Locking

The concept of combinational logic locking and the first scheme “EPIC” were introduced in 2008 by Roy et al. [8]. The main idea is to lock the Boolean logic of a design’s data path. A dominant strategy is to insert several X(N)OR gates and optionally inverters further down the wires such that a correct key does not change the original output. With an incorrect key, the IC will thus “make mistakes” during computations. A simple example of an EPIC-locked circuit is shown in Figure 2.

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 targeted. Here, the locking circuitry extends the original state transitions with additional dummy states. Details depend on the actual scheme, but common examples are states that lead to infinite loops or wildly jump between each other. Simplified, the key nullifies the modification, e.g., by correctly traversing these dummy states or by providing the required control signals for intended state transitions.

Four basic classes of sequential logic locking schemes exist as identified in [9]. They were initially presented in the following contributions: HARPOON by Chakraborty et al. [10], Dynamic State Deflection by Dofe et al. [11], Active Hardware Metering by Alkabani et al. [3], and Interlocking Obfuscation by Desai et al. [12]. There has been considerable follow-up work, mostly improving specific aspects of the original schemes. An example of a HARPOON-locked circuit is shown in Figure 3.

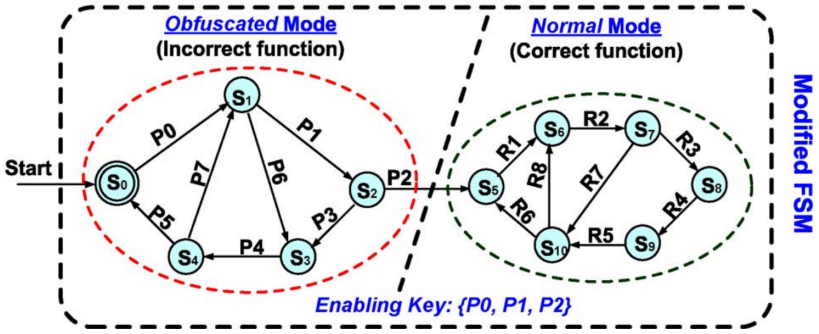


Figure 3: An example of sequential logic locking with HARPOON as depicted in [10]. The locking circuitry facilitates the dummy states on the left while only the correct state sequence leads to the original FSM on the right.

D. Notation and Terminology

In the current literature, the terms "logic locking", "logic encryption", and "logic obfuscation" are used synonymously. We want to emphasize a remark from Plaza and Markov that this mixed terminology is ill-advised and in fact misleading [13]. Indeed "encryption" is tied to making data indecipherable through transformation of the data itself, "obfuscation" transforms a structure into an alternative but functionally equivalent representation, while only "locking" describes restricting access to functionality until unlocked. Hence,"logic locking" is notably the most appropriate term and we will use it in the remainder of this work.

The general terminology in the logic locking literature has not been consistent as well. Examples for the key that is connected to the locking circuitry include "secret key", "unlock key", "master key", "internal key", and "chip key". Therefore, we select generic terms below which appear to be most suitable to address all existing schemes while avoiding confusion. The terms and their relation are also visualized in Figure 4.

- The **Internal Key** is used by the locking circuitry and only known to the IP-rights holder. While most schemes are unlocked using the same internal key for all ICs, in one of the schemes the internal key is not fixed in the design, i.e., it is individual for each IC.
- A **Key Preprocessor** is an optional module preceding a locking scheme. While its output, the internal key, may be the same for all ICs, its input is unique for each IC. This is commonly achieved through the use of a Physically Unclonable Function (PUF).
- The **Chip Key** is derived from the internal key for the unlocking process and eventually transmitted to the IC. Without a key preprocessor, the chip key is simply the internal key itself. Otherwise, the chip key is the input of the key preprocessor, which in turn computes the internal key.
- **Individual** chip/internal key indicates that the respective key is different for each IC.
- **Global** chip/internal key indicates that the respective key is identical for each IC.

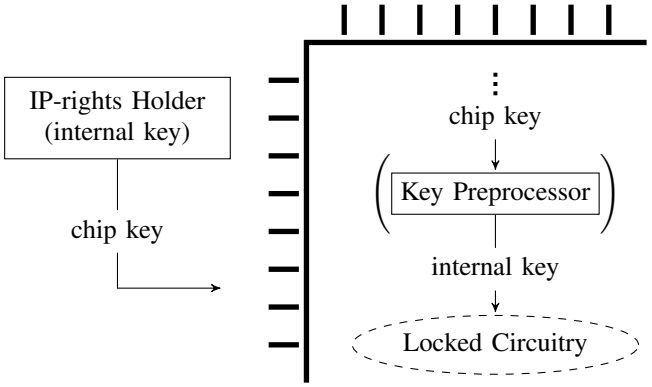


Figure 4: Visualization of the relation between the different keys and elements in logic locking. Note that the key preprocessor is optional.

E. Attacker Model in Previous Work

It is worth noting that the logic locking literature lacks a well-defined attacker model. In most publications, the adversary gets oracle access to several locked and unlocked ICs, as well as a gate-level netlist of the locked design. 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.

In previous work, the attacker's capabilities are only discussed in-depth with respect to non-invasive attacks, most dominantly SAT-based attacks. The adversary is allowed to observe the input/output behavior of the ICs **only** via black-box access. Furthermore, he is allowed to analyze and simulate the locked netlist. Invasive attacks have been mentioned only in passing, cf. [8], [5], but either written off as unrealistic or protection is claimed.

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 thwart overproduction, some schemes additionally argue to protect against reverse engineering or hardware Trojan insertion. Several SAT attacks claim success after recovering the internal key [14]. However, recovering the internal key alone might not suffice to attack the locking scheme, e.g., if it is used with EPIC's key preprocessor [8], [5].

III. Related Work on Attacks

Logic locking has been subject to several kinds of attacks. In this section, we will shortly introduce the on-going work in this area. For more details we refer the interested reader to [15] and [16].

A. Attacks on Combinational Logic Locking

Attacks on combinational logic locking schemes are mostly based on SAT-solving. The general premise is that an attacker is in possession of the locked netlist, e.g., through theft or reverse engineering, and has obtained an already unlocked device, e.g., from the open market or an insider. Then, the main idea is to find a key which produces the same I/O behavior as observed from an unlocked device. The recovered key is not necessarily equal to the actual internal key. The only guaranty is that both keys produce the same outputs for the tested inputs.

The first SAT-based attack on logic locking was presented by Subramanyan et al. [14]. The authors focus on identifying so-called Distinguishing Input Patterns (DIPs), i.e., patterns that aim to exclude multiple wrong key candidates at once. Eventually, a key which is consistent with the input/output behavior of the circuit is found through said DIPs, practically unlocking the correct functionality. With respect to locking schemes that insert key gates at random, Subramanyan et al. effectively broke all schemes available at the time.

This result sparked further research in the same direction: with SARLock [17] and AntiSAT [18], defense mechanisms to thwart SAT-based attacks have been presented shortly after. Prominent examples for attacks against said SAT defenses are the Signal Probability Skew (SPS) attack [19] or the Novel Bypass Attack [20] which in turn inspired more powerful SAT-based attacks, including Double DIP [21] or the most recent SMT attack [22]. The aforementioned attacks were again followed by even stronger defense algorithms as presented in [23]. From a high-level perspective, this line of research developed into an arms race between novel attacks against existing SAT defenses and corresponding countermeasures.

While the majority of attacks focus on SAT-solving, other attack vectors have received attention as well.

Yasin et al. [7] and Sengupta et al. [24] investigated the effectiveness of side-channel attacks against logic locking. In the first work, Yasin et al. provide experiments which discover more than half of a 32-bit secret key using Differential Power Analysis (DPA) for schemes that insert their key gates randomly. They find that more sophisticated placement of key gates rendered their DPA attacks less efficient. Sengupta et al. extend the aforementioned work by performing further experiments and provide ideas how to thwart side-channel attacks on logic locking in the future.

Another class of attacks takes analysis of (parts of) the netlist into account. Li et al. introduced an attack that can extract more than half of the bits of the internal key by identifying redundant logic [25]. Notably, no unlocked IC is needed for this attack. Similarly, experiments show that the so-called Functional Analysis attack on Logic Locking (FALL) [26] can defeat logic locking without oracle access in approx. 90% of the cases. The *desynthesis attack* presented by El Massad et al. enables an attacker to discover several bits of the secret key by re-synthesizing the design according to the current key candidate [27]. Again, the attacker does not need access to an unlocked IC but precise information on the originally employed synthesis tools and options. In [28], Yang et al. present how to unlock the supposedly provable secure locking scheme Stripped-Functionality Logic Locking (SFLL) [4] within minutes since traces of the protected input patterns can still be found in the reverse-engineered design. Another example is the SURF attack by Chakraborty et al. [29], which combines the machine-learning-based SAIL attack [30] with functional analysis in order to successfully recover the internal key.

B. Attacks on Sequential Logic Locking

Sequential logic locking received far less attention by researchers than its combinational counterpart, both in terms of schemes as well as attacks.

However, Fyrbiak et al. [9] successfully attacked the four classes of sequential logic locking they identified (cf. Section II-C). 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 common weaknesses of the existing classes. However, there is a tremendous overhead in terms of area and latency due to the reconfigurability. Even though their new approach does not constitute a practical solution, it underlines that a fundamental rethinking to sequential logic locking is required.

IV. The Current State of Logic Locking Schemes

In order to find generic attacks against all logic locking schemes, it is mandatory to analyze the existing schemes for common properties. To this end, we first perform a high-level analysis of existing solutions before elaborating on two basic but generic observations.

A. High-Level View on Logic Locking

From a high-level perspective, a logic locking solution is comprised of up to two basic components, namely (1) the locking scheme itself and (2) a key preprocessor. The former is a mandatory part of all solutions while the latter is an optional building block that can be prepended to any scheme (cf. Section II-D). Given individual chip keys, i.e., key material that is unique for each IC, a key preprocessor derives the internal key that is used for the actual unlocking.

Furthermore, two types of locking schemes can be distinguished: almost all schemes incorporate a *global internal key*, i.e., all ICs that are produced with the same set of masks, can be unlocked by the same key (cf. Section II-D). However, one scheme features *individual internal keys*.

For a more in-depth recap of existing schemes we refer to the numerous surveys on logic locking, including [31], [32], [33], [16].

1) Locking Schemes with a Global Internal Key: Most locking schemes facilitate a global internal key. The first combinational logic locking scheme was "EPIC" [8], [5]. Here, the locking circuitry, namely X(N)OR and inverter gates, is randomly inserted into the existing combinational logic. Incorrect key bits result in bit flips in signals, thus leading to faulty computations.

EPIC triggered considerable follow-up work which introduced new schemes with improved placement of the locking circuitry, for example based on a small set of randomly placed gates [34], fault simulation techniques [35], [36], [37], deducibility of key bits from output [38], or combinations of such methods [39]. Numerous schemes not only focus on placement but also on the type of locking circuitry. [6], [40], [4] are based on look-up tables which [41] extends with one-way functions, [42] makes use of AND resp. OR gates to also protect against insertion of hardware Trojans, and [13], [27] employ MUXes.

This description of prior work is not exhaustive but includes prominent examples of the different variants of logic locking schemes that employ global internal keys.

2) Locking Schemes with Individual Internal Keys: In contrast to the vast amount of schemes featuring global internal keys (cf. above), there is only a single scheme which generates an individual internal key for each IC: CLIP by Griffin et al. [43]. To this end, CLIP employs process variation sensors which measure slight differences in transistor threshold voltages, yielding a mechanism comparable to a weak PUF [44], [45]. Since CLIP is part of our experimental evaluation, we shortly recap the scheme in the following. The design principle of CLIP's locking circuitry is illustrated in Figure 5. In order to protect a combinational function f, CLIP locally duplicates it. The original inputs of f are fed into a so-called "switchbox" which then provides inputs to both f-instances. Internally, the switchbox forwards the original inputs to only one f instance while the input values of the other instance may be modified. Which instance receives the original values is selected by the output value of one of the aforementioned process variation sensors. Hence, one instance of f computes wrong results while the other instance outputs expected values. An "output selector", i.e., a multiplexer, decides, based on one bit of the internal key, which output of the two f-instances becomes the final output. As a result, only if the key bit correctly selects the output of the f-instance with unmodified inputs, the circuit will behave correctly. With CLIP, each IC produces its own individual internal key, however, even the designer has no way of knowing a specific internal key beforehand. Therefore, the designer has to query each produced chip with a test set in order to recover the internal key from the circuit's outputs and subsequently unlock the device. The authors acknowledge that this can also be done by a malicious entity.

3) Key Preprocessors: Since in most locking schemes all ICs share a global internal key, disclosure of said key would immediately enable overproduction. The need of a mechanism to individualize key material without adjusting the mask for each IC has been noted by various authors. A key preprocessor can be used to achieve this exact goal, i.e., every IC will have an individual chip key from which the internal key is computed

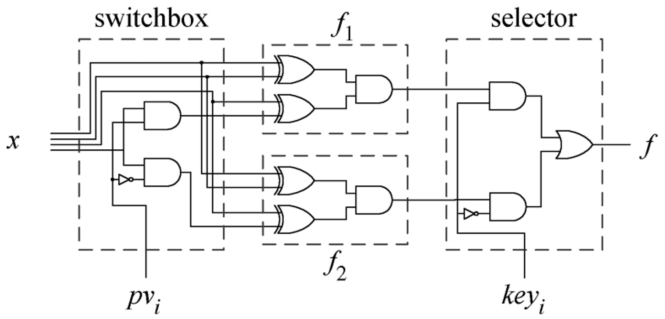


Figure 5: Illustration of CLIP as depicted in [43].

internally. Simplified, this derivation is performed with help of an IC-unique value, for example obtained via a PUF. This way, despite a scheme having a global internal key, every IC requires its own individual chip key that is according to its PUF response. If the correct chip key of one IC was used to unlock a different IC, the key preprocessor would thus compute an incorrect internal key and the design remains locked. As a direct consequence, even if an adversary obtained the internal key itself, other ICs would still not be unlockable if he cannot forge a chip key. In the literature, only two key preprocessors have been proposed so far:

Roy et al. presented the first key preprocessor together with EPIC [8], [5]. The initial motivation was enabling remote unlocking of ICs via asymmetric cryptography, i.e., the IC does not have to be physically send back to the design house for unlocking. 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 internal key using the individual RSA public key of that IC and then signs the resulting ciphertext with its own private key. The chip key sent to the IC contains both, the encrypted internal key and the signature. The key preprocessor derives the internal key from the chip key by verifying the signature and subsequently decrypting the ciphertext with its own private key. 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 [35]. For LECs, the chip key 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 internal key. With respect to EPIC's key preprocessor, the authors argue that a PUF circuit can be implemented much smaller than an RSA engine, however, they do not provide any information regarding concrete instantiation or PUF setup. Furthermore, it is claimed that, even if an adversary obtained the chip key, i.e., PUF challenge and user key, security was not compromised since the attacker cannot guess 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 chip keys.

B. Common Fundamental Aspects

After this brief summary of existing logic locking schemes and key preprocessors, we identified two aspects that are common in all of logic locking and are inherent to the underlying structure.

1) Internal Key Handling: After unlocking an IC by the IP-rights holder, the internal key has to be stored on-chip in non-volatile memory. While most publications do not address storage details, [4] and [32] propose to use read- or tamper-proof memory. However, using dedicated secure memory is not beneficial in the context of logic locking: In typical use cases for protected 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 exposure of sensitive data is limited to the bare minimum. However, for logic locking, the internal key has to be constantly available during IC operation (cf. [37]). It is loaded into a register during boot and remains static until shutdown. This leads to the following generic observation:

Observation 1. *Every logic locking scheme will have its internal key constantly available in a register during operation.*

2) Individualization: While most schemes facilitate a global internal key, CLIP manages to have individual internal keys. Likewise, by using a key preprocessor, schemes with a global internal key can be enhanced to require individual chip keys. These approaches make use of an IC-unique random value. In the examples of CLIP and LECs, a PUF is used while EPIC's key preprocessor also gives the option to use a classical TRNG. Regardless of the way this (crucial) random value is processed, it is the only source of individualization. This leads to the following generic observation:

Observation 2. *All individualization of key material is based on some kind of internal entropy, e.g., derived from a TRNG or a PUF.*

V. A Revised Attacker Model

We recall that the primary goal of logic locking is to defend against IC overproduction and subsequent trade, e.g., on the black market (cf. Section II-A). Another goal which is oftentimes also mentioned is IP protection with respect to reverse engineering. Regarding plain overproduction, the malicious foundry can easily be identified as the main adversary, while reverse engineering can potentially be performed by other parties as well, including competitors or even end users. In this section, we investigate the capabilities of strong adversaries along the design chain and how these capabilities can be used to defeat logic locking.

A. The Attacker in Previous Work

As mentioned in Section III, previous work assumed that an adversary is in possession of the design's netlist and, in most cases, also obtained (multiple) locked and unlocked ICs. Note that with respect to the first requirement, a malicious foundry poses a particularly strong adversary: it receives the design in form of GDSII/OASIS files and generates the lithographic masks for the subsequent fabrication of ICs, hence, it has full access to error-free representations of all layers of the design. For other parties, e.g., competitors, obtaining an error-free netlist is already a challenging task [46].

With access to the netlist (and an unlocked circuit), the adversary can then employ SAT-solving or other non-invasive attacks to find a suitable internal key. While the success of SAT attacks greatly relies on the targeted locking approach, so far, none of the available non-invasive attacks are able to defeat logic locking entirely, e.g., the FALL attack achieves a success rate of 81% against SFLL [26]. Crucially, even when assuming a higher success rate, all SAT-based attacks will not allow to use overproduced chips as soon as a key preprocessor is employed, even if the internal key is compromised. Crucially, even if a SAT-based attack succeeds in recovering the internal key, overproduced ICs cannot necessarily be unlocked if a key preprocessor is present.

B. A Foundry-Level Attacker Model

We argue that restricting the attacker to black-box access severely underestimates the capabilities of a foundry-level adversary who, again, is the main threat regarding overproduction. The key advantages of a malicious foundry include complete knowledge about the design on netlist level as well as physical access within the manufacturing process. Its extensive insights into the layout of all components allow to identify points of interest with ease. Also, modern foundries need to be equipped with (production) failure analysis and process debugging tools. Such tools can be easily misused to perform invasive attacks such as probing and editing. We note that invasive attacks are not limited to foundry-level adversaries. Other adversaries with the corresponding equipment and expertise can in theory perform similar attacks, albeit with potentially more effort. We want to emphasize that we do not merely introduce a more powerful adversary, but rather argue that the existing models do not capture the full capabilities of the primary adversaries in logic locking.

We will now introduce a more comprehensive attacker model covering foundry-level adversaries 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:

- **The gate-level netlist** of the locked design, which can be extracted from the GDSII/OASIS files (cf. Section II-A).
- **Multiple locked ICs**, which can be obtained during the regular production process. For behavioral analysis, simulation of the locked netlist may already suffice.
- **Multiple unlocked ICs** obtained on the market or directly in the foundry if remote unlocking is used (cf. Section IV-A3). We note that in some use cases, e.g., military hardware, obtaining unlocked ICs can be hard or close to impossible.

- **The lithographic masks** and other fabrication artifacts used to manufacture the ICs.
- **State-of-the-art IC analysis equipment**, i.e., testing equipment and tools to perform invasive analysis.

2) Defining a Successful Attack: We define the primary target of an attacker to successfully overproduce functional ICs. However, depending on the scenario, an attacker may already be successful if he can extract the protected IP, i.e., commit IP-theft on a logical level. There are three principal goals to achieve this:

1) The adversary is able to unlock arbitrary locked ICs without design modification.
2) The adversary is able to interfere with the fabrication process to disable or weaken the locking scheme, thus enabling to unlock ICs that are produced with the adversary-induced modification.
3) The adversary is able to remove the locking scheme on netlist level (*removal attack*), thus obtaining an unlocked netlist.

If the adversary follows the primary target, i.e., physical overproduction, Goal 1 is not necessarily achieved by recovering the internal key (e.g., through a SAT-based attack) since a key preprocessor may prevent the attacker from generating correct chip keys. If the locking scheme can be removed or circumvented (Goal 2 and Goal 3), 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 (Goal 2). In the worst case, he has to generate entirely new masks (especially for Goal 3), which is considerably more expensive than production with modified existing masks. Consequently, the threat of these attacks depends not only on technical aspects but also on the financial overhead. For example, a removal attack with succeeding reproduction of IP may only be worthwhile if the black market revenue is expected to amortize the production costs of new masks.

On the other hand, if the adversary follows the target of IP-theft on a logical level, achieving Goal 1 or Goal 2 is not necessarily sufficient, since the IP is not guaranteed to be stripped of the obfuscating elements. Goal 3 however, immediately returns the desired results.

In total, a suitable security assessment of a locking scheme needs to take these factors into account. Because the goals' implications highly depend on the target, i.e., protection against overproduction and/or IP-theft through reverse engineering, we recommend that schemes clearly state their strength against all three goals and name the targets they want to protect against. Likewise, we recommend that attacks specify which goals are reached in order to assess their effectiveness.

VI. New Attack Vectors on Logic Locking

The new attacker model allows for several new attack vectors which have not been considered in previous work. Especially, invasive attacks can (and should) now be considered as well. In the following, we will exemplarily present two new attack vectors with broad impact. Even though they do not cover all attack vectors available to an invasive adversary, we will show in Section VII that they are already sufficient to target the underlying mechanics of logic locking in a general manner.

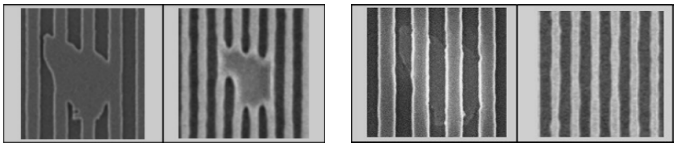
1) Attack Vector: Probing Registers: The internal key is a core asset that should only be known to the design house. If an attacker gets hold of said key, the strength of the scheme is notably reduced or even entirely nullified. Hence, a strong attack vector involves probing the internal key of an unlocked IC during operation.

While probing of signal values can be difficult for a generic attacker, modern foundry-level adversaries have access to sophisticated testing labs [47], which are needed, e.g., for failure analysis. Hence, the attacker is well acquainted with probing techniques. If a signal of interest is routed close to the top layer, probing needles can extract signal values quite easily after Focused Ion Beam (FIB) preparation. Note that this routing information is especially readily available to the foundry in form of the GDSII/OASIS files. If a frontside approach is not an option, backside probing techniques can be leveraged, such as e-beam or laser voltage probing as used in standard testing routines [48], [49], [50], or electro-optical probing and electro-optical frequency modulation [51]. The latter can be further improved by preprocessing the backside with a FIB [52]. We emphasize again that in addition to the technical capabilities, a malicious foundry has unobstructed insights into the design without the need of expensive and error-prone reverse engineering (cf. Section II-A).

In addition, the adversary only needs to probe static data. A locked IC only operates correctly as long as the correct internal key is input to the locking circuitry, i.e., it is a static signal (cf. Section IV-B). Thus, extracting an internal key or chip key 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.

2) Attack Vector: Minimal Mask Modification: Producing lithographic masks is a costly step in modern IC manufacturing. However, making **minimal** adjustments to fabricated masks is feasible using mask repair techniques. With minimal mask modification, we refer to, for example, cutting selected individual wires at cell inputs and reconnecting them to VCC/GND. Hence, these modifications can, for example, tie a normally switching signal to a known constant value. We explicitly do not require insertion of additional logic cells, which is a significantly more complex task. In contrast, connecting a logic-cell's input wire to VCC or GND is quite feasible since the VCC and GND rails run directly beneath the cell rows in CMOS.

Mask repair techniques commonly available to foundries include e-beams [53], [54], [55] or nano-machining via atomic force microscopy [56], which can even be used beyond 20 nm technology. In 2012, Zeiss, a major provider of such equipment, said that their "current system performance is significantly smaller than the claimed limit of 20 nm." [56]. Hence, nowadays, it is reasonable to expect that an ever growing number of foundries is capable of performing the aforementioned minimal adjustments. As an example for the effectiveness of minimal mask modification even on small feature sizes, we show in Figure 6 a mask (32nm reticle) before and after repair, taken from [55].



(a) Erroneous mask and resulting wafer print.

(b) Repaired mask and resulting wafer print.

Figure 6: Examples of mask repair [55].

VII. Attack Vector Evaluation

In this section, we will show how the new attack vectors following from our revised attacker model can lead to successful attacks against virtually all logic locking schemes proposed to date. Crucially, an adversary can exploit the common aspects identified in Section IV-B. Therefore, the premises of the attacks are applicable regardless of the used scheme or key preprocessor. We first discuss how to attack locking schemes without a key preprocessor and subsequently present attacks on the key preprocessors. Our attacks are accompanied by case-studies to demonstrate semi-automated identification of points of interest. All resources for the case studies, i.e., netlists and code, are available online at GitHub[1].

1) Probing Attacks: We recall that probing of values is readily available to the adversaries considered in the attacker model. This allows an adversary to directly target the internal key. If the locking scheme does not produce an individual internal key for each IC, such a *probing attack* can disclose the global internal key and any overproduced IC can subsequently be unlocked. Note that the vast majority of proposed schemes is solely based on a global internal key (cf. Section IV-A).

a) Locating the Internal Key: In order to launch such a probing attack, the adversary must be able to locate the internal key signals. Based on Observation 1, i.e., that the internal key is constantly available in a register during operation, said register has quite unique characteristics that allow for semi-automated identification. In the following, we will give an algorithmic approach on netlist level that identifies signals for probing. Note that in practice, other approaches are viable as well, e.g., manual inspection by an experienced reverse engineer or monitoring of startup behavior.

Following Observation 1, the key is kept in non-volatile memory and subsequently stored in a register through a memory interface during boot. The inputs of said register are only connected to the memory interface, since the register is never loaded with anything else. In order to access the key from its permanent storage inside the IC, a shift register has to be used since an adequate key size exceeds common memory bus widths. Hence, our strategy is to identify shift registers that are loaded from memory and filter the results afterwards through additional criteria such as "loaded only once".

We outlined the general data path structure of a register that is loaded from memory in Figure 7. First, an analyst has to manually identify the signals which form the memory cell outputs. This step is straightforward since memory cells can be

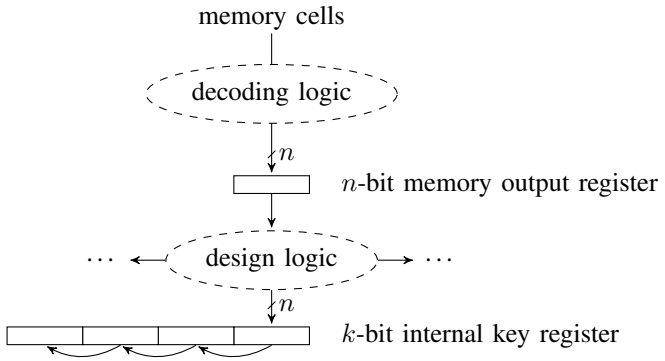


Figure 7: Simplified illustration of the data path from an n-bit memory interface to a k-bit key register, where $k > n$.

easily distinguished from digital logic. These memory output signals are traced to the next set of Flip Flops (FFs) which mark the "memory output register" (cf. Figure 7) and thus the starting point for the remaining analysis.

For each FF in the memory output register, all succeeding FFs are treated as starts of shift registers. By following the chain of directly connected FFs, the full shift registers are identified. Note that a shift register that shifts, for example, by 8 bits at a time, is actually implemented as eight single-bit shift registers with common control signals. Therefore, after identifying said single-bit shift registers, they are grouped by their length and common control signals.

As mentioned earlier, registers that receive data from multiple locations can be ignored, since the key register is only loaded from memory once and then never updated.

By probing the remaining registers and then trying the resulting key candidates on a locked IC, the correct key is identified.

b) Experimental Evaluation: We automated the aforementioned analysis using the open-source hardware analysis framework HAL [57]. All resources, i.e., our algorithm for HAL and the netlists, are available online[2]. In order to evaluate our approach, we ran the algorithm on locked benchmark circuits from Trust-Hub[3] [58]. We randomly picked three designs from the category "multiple modification techniques" ranging from 1000-5000 gates, namely c1908-NS3550, c3540-NR1820, and c6288-NC2240. Since all Trusthub benchmark circuits have the internal key as a primary input, we prepended the design with a UART core as shown in Figure 8. All designs were tested on a Xilinx Artix7 35T FPGA and then synthesized using the academic Nangate 45 standard cell library. While the memory interface was abstracted with a direct I/O interface, our algorithm remains applicable as it depends solely on the selection of the starting point.

Analogously to the memory output register, we selected the UART receiver register as our starting point and ran the analysis in HAL. After manually filtering out register groups of small size, e.g., less than 20 bit, we verified correctness of the remaining results by inspecting the gate names of the

[1]Resources are available at https://gofile.io/?c=PaO0kE to preserve review anonymity.

[2]Resources are available at https://gofile.io/?c=PaO0kE to preserve review anonymity.

[3]http://www.trust-hub.org/benchmarks/obfuscation

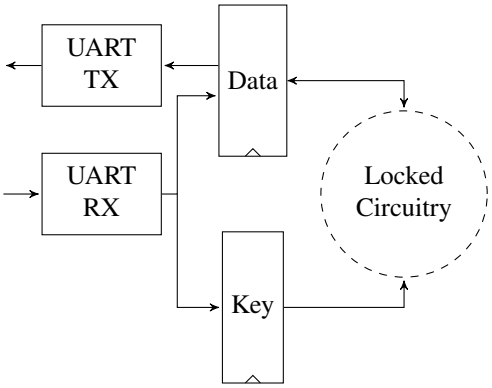
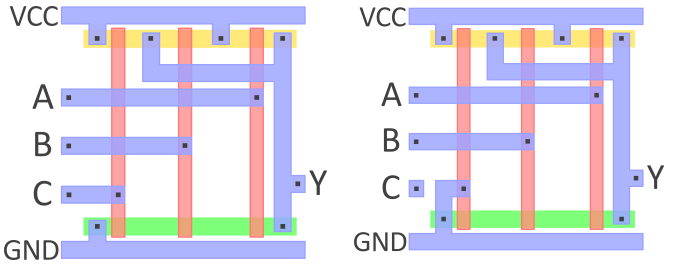


Figure 8: Simplified illustration of our experimental setup.



(a) Exemplary CMOS layout of a NAND3 cell.

(b) Modified cell with input C fixed to GND.

Figure 9: Application of a mask modification attack. Wire spacings and thickness have been adjusted to improve readability. The wire on input C was cut and directly connected to GND.

annotated netlist. Naturally, our algorithm does not make use of naming information as this would be stripped/anonymized in a real-world scenario and it was only used for verification.

Additional details on the case study and the output of the algorithm can be found in Section A.

c) Results: For all three designs our algorithm correctly recovered the eight single-bit shift registers that together form the key register. The data register (cf. Figure 8) was always filtered out during the FF-chain analysis, since its FFs were separated by multiple combinational gates. These gates acted as multiplexers since the data register can be loaded from both, memory and the locked circuitry output.

Note that we neither designed the protected designs, nor the locking circuitry ourselves and did not take any scheme-specific details into account. In order to mimic a realistic design, we had to generate an interface circuitry of the locked designs ourselves, which we instantiated with the UART core. Whether the starting point is the output register of a UART interface or a memory interface is irrelevant to the analysis algorithm.

In the end, the algorithm outputs a small set of candidate registers for probing. Even if more than one register were returned by the algorithm, the overall search space has been heavily reduced to the point where exhaustive testing becomes feasible. If the bit order of the key, i.e., the order of the single-bit shift-registers, cannot be deduced from memory layout, all permutations have to be considered.

Note that the given algorithm is just one way of identifying key registers. Other approaches may observe the startup behavior of FFs through simulation of the netlist or optical inspection of an IC in order to narrow the set of candidates FFs to probe. Identifying FFs that are loaded only at the beginning of operation and remain static afterwards are immediate candidates for the key register.

Crucially, the general premise of a probing attack is always applicable to logic locking. While the specific approach to find the key register may require small modifications depending on the cell library or design architecture, identification of said register suffices to enable the attack. Especially the aforementioned startup behavior is fixed and therefore not mitigated easily.

The probing attack aims for Goal 1 and Goal 3 of our attacker model: unlocking arbitrary ICs without authorization and removing the locking scheme on netlist level. While a key preprocessor might thwart reaching Goal 1, Goal 3 is always reached through a probing attack.

Recall that a prerequisite of the attack is that the adversary has access to an unlocked IC. This requirement can be viewed analogously to a known-plaintext attack in cryptanalysis and is consistent with both, our revised attacker model as well as the attacker models used in previous work. As mentioned earlier, unlocked ICs can be obtained, for example, on the open market or from an insider.

On a side note, locking schemes with individual internal keys are not susceptible to mere probing attacks, since obtaining the internal key of one IC does not provide any information on the internal 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: We recall from Observation 2 that in order to individualize locking schemes, an entropy source, e.g., a TRNG or a PUF, is required. 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, i.e., using a global key. Note that this does not necessarily disclose sensitive key material but makes all modified ICs unlock with the same chip key.

As discussed in Section VI-2, such alterations are particularly easy with mask modification, since they only require to connect the input/outputs of selected standard cells to the nearby VCC and GND rails. Figure 9 illustrates this situation by the example of a `NAND3` cell. It is evident that only minimal edit is required to connect any of the inputs to GND, as done for input C in the example.

a) Locating the Signals of Interest: Similar to finding the key register, the adversary needs to identify the location at which the mask modification will be applied. In general, the entropy source (that is, in practice either a TRNG or a PUF) itself can be identified due to its specific structure, which is different from the structures employed in the remaining IC. Examples include analog components within digital logic, combinational loops used for ring-oscillator PUFs or TRNGs, or transistor groups that do not match any standard cells as in the PUF from [43].

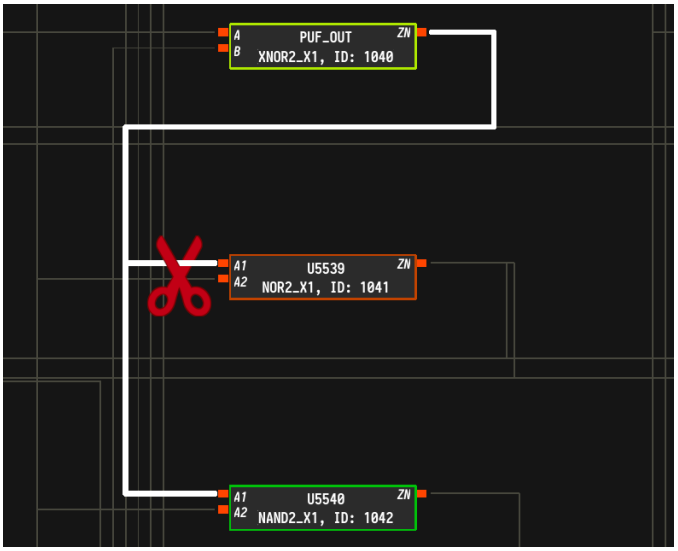


Figure 10: Visualization of a wire split in HAL. The red scissor marks a safe position for wire cutting to affect only the orange cell.

We note that simply tying the output of the entropy source to a constant value is not sufficient for a realistic attack. In this case, other modules that are not related to logic locking but which also make use of said entropy source would malfunction. An example for such a module is a self-test of a TRNG which are widely used in practice. Therefore, it is necessary to affect only the "entropy signals" that reach the logic-locking-related circuitry. For demonstration purposes, we use HAL to visualize a suitable position for modification, shown in Figure 10: We assume that the yellow cell on top was identified as the final cell of a PUF circuit, and hence renamed to `PUF_OUT`. The orange cell is in the path connected to the key register, while the green cell belongs to a different module. If the outgoing wire (highlighted in white) is modified at the red scissor, only the red cell is affected and the green cell still gets the original PUF output.

We now give a strategy how such a position can be determined automatically: First, the internal key signals are identified with the same strategy as for the aforementioned probing attack. Then, succeeding gates are traced up to a certain depth. All gates along the traversed paths are marked as *key-affected gates*. As a second step, the outputs of the entropy sources are then traced until a key-affected gate is hit. Finally, the paths from entropy source to key-affected gates are traversed backwards to find the last wire split of the original entropy signal. The identified positions are equivalent to the example shown in Figure 10. The given strategy is applicable to circuits where the key is computed/derived from the entropy source or where the entropy source and the key are both inputs to converging combinational logic cones.

b) Experimental Evaluation: We modeled a mask modification attack on CLIP [43], since it is the only available scheme with individual internal keys. The authors of CLIP introduced a special process variation sensor for their scheme, i.e., a weak PUF, shown in Figure 11 which is based on process variation in transistor threshold voltages. As indicated in the preceding paragraph, the employed PUF can be identified by processing the IC layout and identifying non-standard-cell groups of transistors (cf. Figure 11).

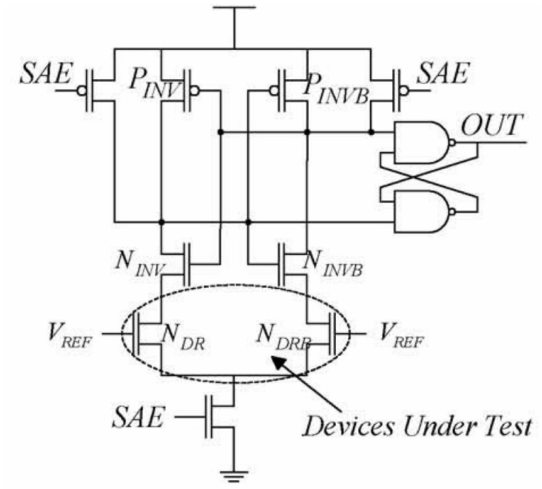


Figure 11: Process variation sensor based on differences in transistor threshold voltages as depicted in [43]

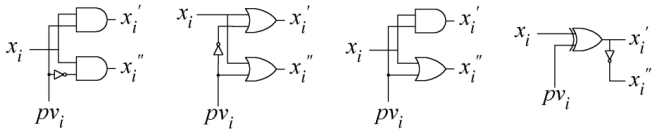


Figure 12: Four types of single-bit switchboxes that guarantee either $x_i' = x_i$ or $x_i'' = x_i$ depending on process variation sensor output pv_i as depicted in [43]. Note that, depending on the value of pv_i, some switchboxes may actually have $x_i = x_i' = x_i''$, i.e., are not suitable in all situations.

As a case study, we again use the analysis framework HAL to evaluate the aforementioned strategy to identify the locations for mask modification. Since TrustHub does not offer any designs locked with CLIP, we applied the locking scheme to the unprotected c3540 design of the ISCAS'85 benchmark suite and embedded it in our I/O architecture shown in Figure 8 which we also used for the previous case study. In order to protect a combinational logic cone with CLIP, we need to add the locking circuitry: a switchbox with a corresponding PUF on one or more inputs and a multiplexer on the output (cf. Section IV-A2). We randomly selected eight combinational logic cones of varying size as protection targets, using each of the four switchbox types presented by Griffin et al. (cf. Figure 12) twice. Furthermore, two PUFs were additionally connected to a dummy register. This dummy register will in practice be accessed by circuit modules which must not be affected by the mask modification. We tested the design in simulation and again synthesized the design using the academic Nangate 45 standard cell library. Since the PUFs do not correspond to standard cells, they are represented as custom standard cells of unknown functionality in the netlist. Note that only eight protected cones, i.e., an 8-bit key, combined with random placement would result in a very

weak locking scheme, but this case study solely aims at finding suitable points for an attack where only locking circuitry is affected, hence the example suffices.

We ran our algorithm on two variants of the locked netlist: in the first variant, every PUF is directly connected to its switchbox. In the second variant, all PUF outputs go through an additional buffer and two PUFs bits were used for two switchboxes each. The originally used PUFs for said switchboxes were left unconnected.

Additional details on the design of the case study and the output of the algorithm can be found in Section B.

c) Results: Our algorithm automatically detected the correct cell inputs/outputs for modification in both netlist variants for all eight protected logic cones and successfully avoided PUF output wires which were connected to locking-unrelated registers. Again, all resources, i.e., the source code of the algorithm for HAL and the netlists, are available online[4].

A mask modification as shown in Figure 9 can now be applied to manipulate these signals, e.g., connecting them to GND. All ICs produced with the modified masks are identical clones of each other with respect to the locking scheme, since any chip-unique random input has been changed to a common constant. This allows an adversary to unlock all subsequently fabricated ICs with a global key. Note that, in general, 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. However, in the special case of CLIP, the choice of PUF outputs directly sets the internal key.

We stress again that the manipulation only affects the locking scheme, while potential internal tests of the entropy source or other modules that use it stay unaffected. Hence, in our small-scale case study we were able to show that detection of relevant signals can be automated with open-source analysis software after manual detection of PUF circuits.

The mask modification attack aims for Goal 2 of our attacker model: disabling the locking scheme during fabrication. Note that, in the case of CLIP, the attack also sets the internal key to an attacker-chosen value, hence also reaching Goal 3. A mask modification attack is particularly effective if the ICs are ordered in a single batch, as is often the case in military products.In that case, mask modification can be performed after all genuine chips have been shipped and the legitimate designer will never receive modified instances.

As mentioned before, mask modification is just one attack vector that targets Observation 2. The same effect could also be achieved with other techniques, e.g., dopant changes in transistors as shown by Becker et al. [59], but for the sake of clarity, we stay with the mask modification 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.

3) Discussion: In this section, we argued that if used without a key preprocessor, all existing locking schemes can be successfully attacked with at least one of the presented techniques. Furthermore, we demonstrated in case studies that the points of interest for an attack can be found in an semi-automated manner, i.e., after manual identification of starting points. We argued that said starting points are easily identifiable by a human analyst.

Crucially, the attacks target common fundamental design properties of all locking schemes: existence of a key register (Observation 1) and individualization through randomness (Observation 2). For these attacks, the main difficulty lies in finding suitable points for an attack, while the remaining part of the attack requires only well-known techniques (cf. Section VI).

Even if new schemes that are perfectly secure against any kind of non-invasive attacks (such as SAT-based attacks) were discovered, schemes with a global internal key will never be secure against a probing attack. Likewise, while the target of mask modification attacks may be slightly different for various schemes, the attack vector will always be applicable. This result is summarized in Table IIa.

From the attacks presented so far, we can already conclude that **without a key preprocessor, none of the available schemes are secure**.

A. Attacking Key Preprocessors

As noted in Section IV-A3, employing a key preprocessor is advantageous because it can retain security even if the internal 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.

In previous work, two key preprocessors have been presented, namely the EPIC key preprocessor [8], [5] and LECs [35] (cf. Section IV-A3).

1) LECs: LECs make use of a PUF to facilitate individual chip key. However, when working out the specifics, several weaknesses arise. The only details given in [35] 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, which is typically applied in the reconstruction step of a strong PUF, is mentioned explicitly. The PUF output is XORed to the so called "user key" (cf. Section IV-A3) to generate the internal 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, an adversary that is already in possession of the internal key could use this very interface in a similar way for an attack. Alternatively, without knowledge of the internal key, the adversary can model the PUF of each IC via machine learning [60], [61]. 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 internal 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, [35] does not provide definite detail, hence we have to consider LECs unusable in their current state and neglect them in the following.

[4]Resources are available at https://gofile.io/?c=PaO0kE to preserve review anonymity.

However, the mere application of a PUF is directly vulnerable to a mask modification attack, comparable to our case study with CLIP (cf. Section VII-2).

2) EPIC's Key Preprocessor: We recall that the EPIC key preprocessor uses asymmetric cryptography to allow for an individual chip key for each IC, even if the underlying locking scheme incorporates a global internal key. Furthermore, the chip key contains not only the encrypted internal key but also a digital signature for authentication.

We will now present multiple attacks, based on our generic attack vectors, probing and mask modification. We first analyze the potential of each attack vector individually, before evaluating them in a combined fashion. We note that these are the first published attacks against EPIC's key preprocessor.

a) Probing Attack: EPIC does not provide in-depth details on how the unlocking procedure works. For instance, it is not clear whether the chip key is stored directly on-chip and the key preprocessor is invoked with each power-up, or whether only the derived internal key is stored after initial unlocking. However, in both cases, a probing attack against EPIC's key preprocessor can eventually reveal the internal key which reaches Goal 3 of our attacker model. If only the internal key is stored, it can be probed directly. If the chip key is stored, the internal key can be probed from the key preprocessor's output wires. Similar to the probing attacks in the previous section, the main obstacle for an adversary is identifying the signals of interest. Another viable approach is to extract the chip key itself as well as the IC's internal RSA key pair, by probing the fuses (as shown for eFuses on FPGAs in [62]). This key pair can then be used to decrypt the internal key from the chip key.

However, knowledge of the internal key alone is not sufficient to directly unlock other ICs due to the digital signature as already outlined by the authors of EPIC [8], [5]. Crucially, we will show that a combination of probing attack and mask modification attack is indeed successful.

b) Mask Modification Attacks: A mask modification attack on the EPIC key preprocessor can target the randomness that is used to generate the internal RSA key. This will lead to a situation where all ICs use the same (attacker-chosen) key pair and ultimately accept the same chip key. Alternatively, the same behavior 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 again share the same key pair. Without valid chip keys, this attack alone does not fully break the scheme, though. However, recall that the initial motivation for EPIC's key preprocessor was remote unlocking, where upon request the design house transmits a valid chip key for an IC. By requesting to unlock a single modified ICs the attacker can then unlock all "clones" of that IC with the same data, no probing required, reaching Goal 2 of the attacker model.

Another attack target can be the signature verification mechanism for authenticity of the chip key. No matter how the verification is implemented, it eventually comes down to a binary decision. By forcing this signal to always-true, every chip key 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.

Attack	Unlocking Scenario	Reached Goals
Probing	Remote	Goal 3
	In-house	Goal 3
Mask Modification	Remote	Goal 2
	In-house	-
Combined	Remote	Goals 2 and 3
	In-house	Goals 2 and 3

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

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

Internal Key	Probing	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.

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

Table II: Summary of the security of logic locking with respect to various adversary capabilities. A 🛡 indicates *not vulnerable*, ϟ indicates always *vulnerable*. For non-invasive attacks (ϟ) indicates that even the best attacks do not achieve a success rate of 100% against sophisticated schemes.

c) 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 internal key invalidates the EPIC key preprocessor even in that scenario. Once the key has been obtained, e.g., through probing attacks, the adversary has two options: he either fixes the input signals to the internal key register to the extracted internal key, or, with only a single modification, disables signature verification which enables the forgery of chip keys knowing the internal key. Both options reach Goal 2 and Goal 3 of the new attacker model. The attack vectors for both scenarios are summarized in Table I. While in [8], [5] an attack that modifies masks was regarded as not realistic, we argued in Section VI-2 that minimal mask modification is not only a viable technique but also a widely used method in modern semiconductor manufacturing and especially applicable to manipulate standard cell inputs/outputs.

d) Summary: In contrast to plain locking schemes without a key preprocessor, non-invasive attacks alone have not been successful yet in attacking EPIC's key preprocessor. However, we showed that EPIC's key preprocessor can be attacked when invasive attacks are taken into account by again targeting our basic observations from Section IV-B. As a result, Table IIb highlights that regardless of the underlying locking scheme, invasive attacks can fully circumvent EPIC's key preprocessor.

VIII. Conclusions — The End of Logic Locking

The starting point of this work was the fact that the actual physical capabilities of adversaries have been overlooked in the literature. A malicious foundry, as the main threat regarding overproduction, is capable of a variety of invasive attacks in addition to the non-invasive attacks considered in previous work. We gave two exemplary attack vectors that are enabled by the capabilities of a foundry-level attacker and demonstrated their effectiveness. It is noteworthy that, somewhat counterintuitively, it does not matter whether a scheme incorporates a global or individual internal keys. As a key contribution, we showed that in contrast to existing non-invasive attacks, our invasive attack vectors stem from two fundamental observations that are given in all logic locking schemes, regardless of scheme specifics.

In previous work, no attacks against key preprocessors were presented. We showed that via invasive attacks, a malicious foundry is able to invalidate the benefits of the only viable key preprocessor which was introduced together with EPIC [8], [5].

Table III aggregates our results. Crucially, no combination of scheme and key preprocessor provides sufficient security against a malicious foundry.

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

Table III: Condensed overview on the security of logic locking against invasive and non-invasive attacks. A 🛡 indicates *not vulnerable*, ϟ indicates always *vulnerable*. For non-invasive attacks (ϟ) indicates that even the best attacks do not achieve a success rate of 100% against sophisticated schemes.

Generalizing Our Findings:
The results at hand demonstrate that IP-theft on logical level, i.e., reaching Goal 3 of the attacker model, is always possible with a probing attack, regardless of a potential key preprocessor. They also 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 internal keys are always vulnerable to probing attacks. Likewise, randomness that is used to individualize key material can always be meaningfully overwritten via mask modification in a way that only the locking circuitry is affected. All security is now reduced to the difficulty of finding the signals of interest. However, following our case studies, there are strong indications that identifying these signals is feasible and even automatable. First, a foundry has full access to an error-free netlist, and knows many other details about the target IC. Second, the regions of interest are either key registers or entropy sources, both of which exhibit unique characteristics which further ease identification. Hence, foundry-level invasive attacks appear to be a major threat against the underlying aspects of both, locking schemes and key preprocessors, in all available configurations.

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

[4] 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.

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

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

[7] M. Yasin, B. Mazumdar, S. S. Ali, and O. Sinanoglu, "Security analysis of logic encryption against the most effective side-channel attack: Dpa," in 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), 2015, pp. 97–102.

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

[9] 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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," CoRR, vol. abs/1905.05896, 2019. [Online]. Available: http://arxiv.org/abs/1905.05896

[16] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, Hardware Security and Trust: Logic Locking as a Design-for-Trust Solution. Cham: Springer International Publishing, 2019, pp. 353–373.

[17] M. Yasin et al., "SARLock: SAT Attack Resistant Logic Locking," in 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3-5, 2016, 2016, pp. 236–241.

[18] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in CHES. Springer, 2016, pp. 127–146.

[19] M. Yasin et al., "Security Analysis of Anti-Sat," in 22nd Asia and South Pacific Design Automation Conference, ASP-DAC 2017, Chiba, Japan, January 16-19, 2017, 2017, pp. 342–347.

[20] X. Xu et al., "Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks," in Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, 2017, pp. 189–210.

[21] Y. Shen and H. Zhou, "Double Dip: Re-Evaluating Security of Logic Encryption Algorithms," in Proceedings of the on Great Lakes Symposium on VLSI 2017. ACM, 2017, pp. 179–184.

[22] K. Azar, H. Kamali, H. Homayoun, and A. Sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2019, no. 1, pp. 97–122, Nov. 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7335

[23] K. Juretus and I. Savidis, "Increasing the sat attack resiliency of in-cone logic locking," in 2019 IEEE International Symposium on Circuits and Systems (ISCAS), May 2019, pp. 1–5.

[24] A. Sengupta, B. Mazumdar, M. Yasin, and O. Sinanoglu, "Logic locking with provable security against power analysis attacks," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1–1, 2019.

[25] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in 2019 Design, Automation Test in Europe Conference Exhibition (DATE), March 2019, pp. 540–545.

[26] 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

[27] 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.

[28] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (sfll-hd) – unlocked," IEEE Transactions on Information Forensics and Security, vol. 14, no. 10, pp. 2778–2786, 2019.

[29] P. Chakraborty, J. Cruz, and S. Bhunia, "Surf: Joint structural functional attack on logic locking," in 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2019, pp. 181–190.

[30] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: machine learning guided structural analysis attack on hardware obfuscation," CoRR, vol. abs/1809.10743, 2018.

[31] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," Proceedings of the IEEE, vol. 102, no. 8, pp. 1283–1295, 2014.

[32] 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.

[33] S. Dupuis and M. Flottes, "Logic Locking: A Survey of Proposed Methods and Evaluation Metrics," J. Electronic Testing, vol. 35, no. 3, pp. 273–291, 2019.

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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

[54] 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

[55] 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.

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

[57] 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.

[58] S. Amir, B. Shakya, X. Xu, Y. Jin, S. Bhunia, M. Tehranipoor, and D. Forte, "Development and Evaluation of Hardware Obfuscation Benchmarks," Journal of Hardware and Systems Security, vol. 2, no. 2, pp. 142–161, 2018.

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

[60] 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.

[61] 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.

[62] 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.

APPENDIX

A. Case Study: Probing Attack

In this section, we provide additional explanation and figures on the internals and output of our case study for finding the key register in order to conduct a probing attack. Recall that we applied our algorithm to three benchmark circuits. Listing 1 shows the common output at the start of the analysis for all three circuits: starting from the FFs at the interface outputs, succeeding FFs are analyzed as potential starting points of shift registers.

Listing 1: Common output of our algorithm for all three netlists at the beginning of the analysis.

```
searching from rx_REG_reg_4_
  analyzing potential chain, starting with
      data_reg_out_reg_4_
  analyzing potential chain, starting with
      key_reg_out_reg_4_
searching from rx_REG_reg_3_
  analyzing potential chain, starting with
      data_reg_out_reg_3_
  analyzing potential chain, starting with
      key_reg_out_reg_3_
searching from rx_REG_reg_5_
  analyzing potential chain, starting with
      data_reg_out_reg_5_
  analyzing potential chain, starting with
      key_reg_out_reg_5_
searching from rx_REG_reg_6_
  analyzing potential chain, starting with
      data_reg_out_reg_6_
  analyzing potential chain, starting with
      key_reg_out_reg_6_
searching from rx_REG_reg_2_
  analyzing potential chain, starting with
      data_reg_out_reg_2_
  analyzing potential chain, starting with
      key_reg_out_reg_2_
searching from rx_REG_reg_1_
  analyzing potential chain, starting with
      data_reg_out_reg_1_
  analyzing potential chain, starting with
      key_reg_out_reg_1_
searching from rx_REG_reg_0_
  analyzing potential chain, starting with
      data_reg_out_reg_0_
  analyzing potential chain, starting with
      key_reg_out_reg_0_
searching from rx_REG_reg_7_
  analyzing potential chain, starting with
      data_reg_out_reg_7_
  analyzing potential chain, starting with
      key_reg_out_reg_7_
```

Listings 2, 3, and 4 show the remaining output of the analysis for the circuits c1908-NS3550, c3540-NR1820, and c6288-NC2240 respectively. The netlists where synthesized with default options, resulting in only D-FFs being used in the design. As a consequence, filtering the found shift registers by common control signals was not possible without control path analysis. However, all FFs were preceded by a gate which basically acted as an input multiplexer between the current output, i.e., no change in the internal state, and a different input signal. Our analysis had to be adapted to allow for a single combinational gate between FFs in a shift register. Still, the data register was not detected, since it has multiple combinational gates between FFs in order to be loaded from memory and from the output of the protected logic.

Listing 2: Output of our algorithm for the c1908-NS3550 netlist.

```
found 16 shift registers
grouped the shift registers into 3 groups
analyzing group of 4 0-to-4-bit shift registers
  contains 6 bits in total -> too small for a key
analyzing group of 8 43-to-47-bit shift registers
  contains 355 bits in total -> key register
      candidate
  success: register 0 contains only key FFs
  success: register 1 contains only key FFs
  success: register 2 contains only key FFs
  success: register 3 contains only key FFs
  success: register 4 contains only key FFs
  success: register 5 contains only key FFs
  success: register 6 contains only key FFs
  success: register 7 contains only key FFs
analyzing group of 4 4-to-8-bit shift registers
  contains 20 bits in total -> too small for a key
verification result: key fully identified
```

Listing 3: Output of our algorithm for the c3540-NR1820 netlist.

```
found 16 shift registers
grouped the shift registers into 3 groups
analyzing group of 8 22-to-26-bit shift registers
  contains 182 bits in total -> key register
      candidate
  success: register 0 contains only key FFs
  success: register 1 contains only key FFs
  success: register 2 contains only key FFs
  success: register 3 contains only key FFs
  success: register 4 contains only key FFs
  success: register 5 contains only key FFs
  success: register 6 contains only key FFs
  success: register 7 contains only key FFs
analyzing group of 7 5-to-9-bit shift registers
  contains 49 bits in total -> too small for a key
analyzing group of 1 0-to-4-bit shift registers
  contains 2 bits in total -> too small for a key
verification result: key fully identified
```

Listing 4: Output of our algorithm for the c6288-NC2240 netlist.

```
found 16 shift registers
grouped the shift registers into 3 groups
analyzing group of 2 3-to-7-bit shift registers
  contains 8 bits in total -> too small for a key
analyzing group of 8 27-to-31-bit shift registers
  contains 224 bits in total -> key register
      candidate
  success: register 0 contains only key FFs
  success: register 1 contains only key FFs
  success: register 2 contains only key FFs
  success: register 3 contains only key FFs
  success: register 4 contains only key FFs
  success: register 5 contains only key FFs
  success: register 6 contains only key FFs
  success: register 7 contains only key FFs
analyzing group of 6 0-to-4-bit shift registers
  contains 13 bits in total -> too small for a key
verification result: key fully identified
```

B. Case Study: Mask Modification Attack

In this section, we provide additional explanation and figures on the internals and output of our case study. We synthesized the netlists with the PUF signals as external inputs. Then, post synthesis we manually removed said external input signals and instead instantiated the "custom standard cells" that drive the respective wires. This way we were able to facilitate the case study netlist without a semi-custom design flow. While all locked combinational cones consist of multiple gates on behavioral level, the synthesizer managed to merge them into less standard cells (cf. respective netlist files).

Recall that we applied our algorithm to two variants of the CLIP-locked netlist. In the first variant, all PUF outputs are directly connected to switchboxes. PUF 0 and PUFs 3 were also connected to dummy registers through combinational logic, which must not be affected by a mask modification. The surrounding circuitry of each PUF in this netlist is shown in Figure 13. Arrows indicate wires between cells and the names correspond to the cell names of the netlist. The red cells are unrelated to CLIP and thus must not be affected by a mask modification attack.

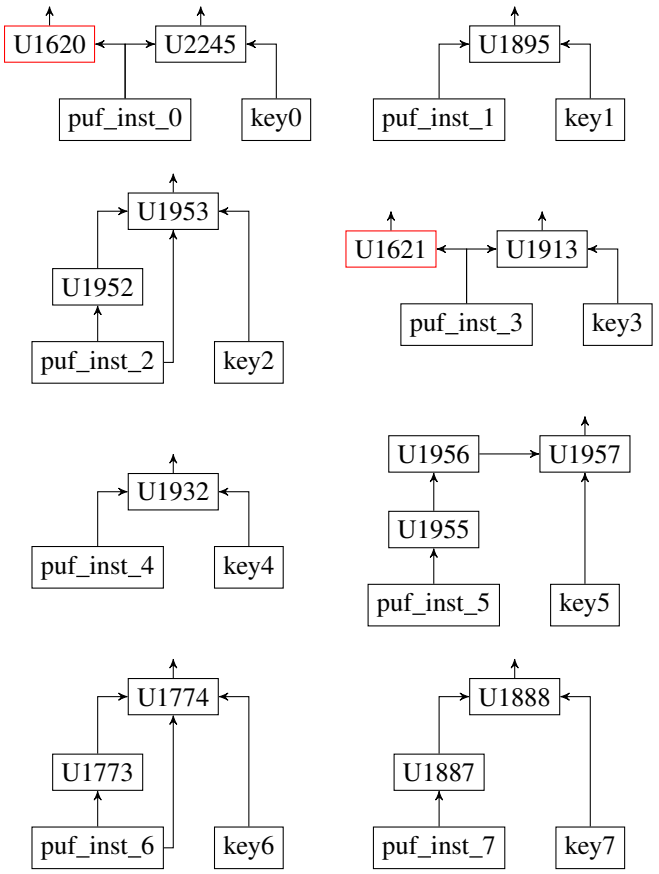


Figure 13: Connectivity of the PUF-related gates in the first variant of the CLIP netlist.

Listing 5 shows the output of our algorithm on the first netlist variant. During path traversal, the number in braces indicates the distance from the respective PUF in cells. As required for PUF 0 and PUF 3, specific inputs of cells that succeed the PUF are returned as mask modification targets since both PUFs are also connected to unrelated circuitry which must not be affected. For the remaining PUFs the algorithm correctly identified that a modification directly at the PUF output is sufficient.

Listing 5: Output of our algorithm on the first variant of the CLIP netlist.

```
found 92 key-dependent gates
traversing path from puf_inst_0
  traced path(s) puf_inst_0 (0), U2245 (1)
traversing path from puf_inst_1
  traced path(s) puf_inst_1 (0), U1895 (1)
traversing path from puf_inst_7
  traced path(s) puf_inst_7 (0), U1887 (1), U1888
      (2)
traversing path from puf_inst_6
  traced path(s) puf_inst_6 (0), U1774 (1), U1773
      (1), U1774 (2)
traversing path from puf_inst_5
  traced path(s) puf_inst_5 (0), U1955 (1), U1956
      (2), U1957 (3)
traversing path from puf_inst_4
  traced path(s) puf_inst_4 (0), U1932 (1)
traversing path from puf_inst_3
  traced path(s) puf_inst_3 (0), U1913 (1)
traversing path from puf_inst_2
  traced path(s) puf_inst_2 (0), U1953 (1), U1952
      (1), U1953 (2)

analyzing path starting with puf_inst_7:
  fix output of puf_inst_7
analyzing path starting with puf_inst_6:
  fix output of puf_inst_6
analyzing path starting with puf_inst_5:
  fix output of puf_inst_5
analyzing path starting with puf_inst_4:
  fix output of puf_inst_4
analyzing path starting with puf_inst_3:
  fix input B of U1913
analyzing path starting with puf_inst_2:
  fix output of puf_inst_2
analyzing path starting with puf_inst_1:
  fix output of puf_inst_1
analyzing path starting with puf_inst_0:
  fix input B of U2245
```

The second netlist variant includes a few changes which are shown in Figure 14. First, all PUFs are connected to a buffer before reaching the switchboxes or other circuitry. Second, PUF 1 and PUF 5 were disconnected and the respective switchboxes were instead connected to PUF 0 and PUF 6 respectively. This results in two instances where two switchboxes are controlled by a single PUF. Furthermore, PUF 0 is now connected to two switchboxes and unrelated circuitry. Again, the red cells are unrelated to CLIP and thus must not be affected by a mask modification attack.

Listing 6 shows the output of our algorithm on the second netlist variant. Again the locking-unrelated cells were correctly identified and correct modification locations were returned.

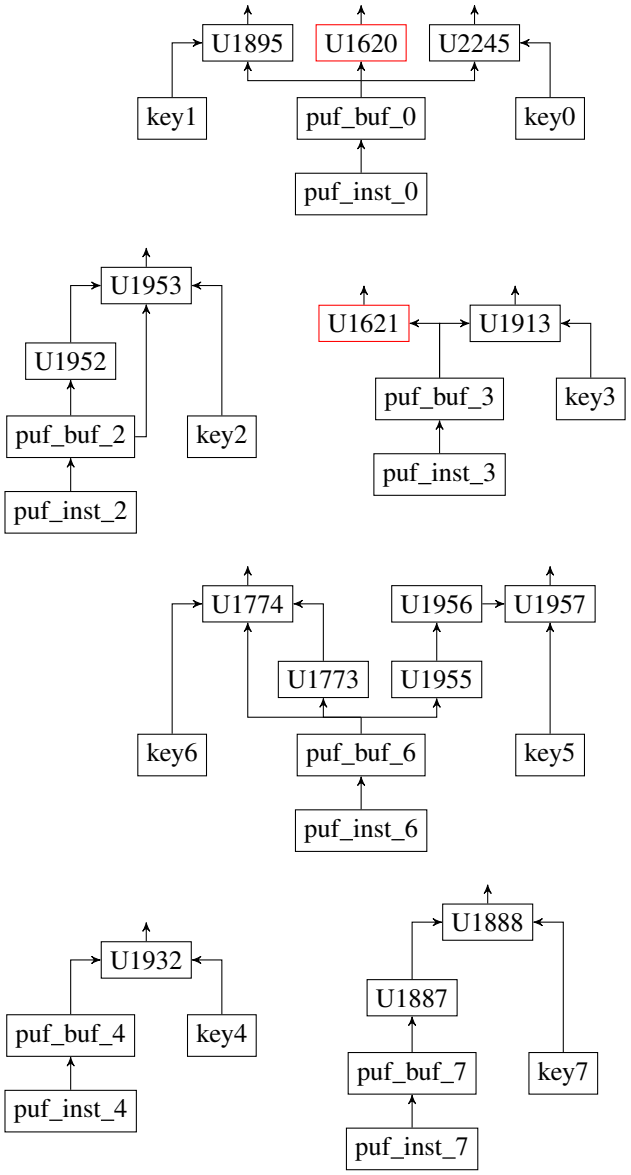


Figure 14: Connectivity of the PUF-related gates in the second variant of the CLIP netlist.

Listing 6: Output of our algorithm on the second variant of the CLIP netlist.

```
found 92 key-dependent gates
traversing path from puf_inst_7
traced path(s) puf_inst_7 (0), puf_buf_7 (1), U1887
    (2), U1888 (3)
traversing path from puf_inst_6
  traced path(s) puf_inst_6 (0), puf_buf_6 (1),
      U1774 (2), U1773 (2), U1955 (2), U1774 (3),
      U1956 (3), U1957 (4)
traversing path from puf_inst_5
  no key-dependent gates were hit
traversing path from puf_inst_4
  traced path(s) puf_inst_4 (0), puf_buf_4 (1),
      U1932 (2)
traversing path from puf_inst_3
  traced path(s) puf_inst_3 (0), puf_buf_3 (1),
      U1913 (2)
traversing path from puf_inst_2
  traced path(s) puf_inst_2 (0), puf_buf_2 (1),
      U1953 (2), U1952 (2), U1953 (3)
traversing path from puf_inst_0
  traced path(s) puf_inst_0 (0), puf_buf_0 (1),
      U1895 (2), U2245 (2)
traversing path from puf_inst_1
  no key-dependent gates were hit

analyzing path starting with puf_inst_7:
  fix output of puf_inst_7
analyzing path starting with puf_inst_6:
  fix output of puf_inst_6
analyzing path starting with puf_inst_4:
  fix output of puf_inst_4
analyzing path starting with puf_inst_3:
  fix input B of U1913
analyzing path starting with puf_inst_2:
  fix output of puf_inst_2
analyzing path starting with puf_inst_0:
  fix input A3 of U1895
  fix input B of U2245
```