Journal of Hardware and Systems Security (2025) 9:205–215
https://doi.org/10.1007/s41635-025-00171-z

Knowledge Is Power: A Knowledge-Guided Oracle-Less Attack on Logic Locking

Abir Akib[1] · **Yuntao Liu**[2] · **Ankur Srivastava**[1]

Received: 15 May 2024 / Accepted: 29 September 2025 / Published online: 22 October 2025

Abstract

Logic locking has emerged as a technique for safeguarding intellectual property (IP) in chip designs. The continuous cat-and-mouse game between logic locking techniques and attackers has led to the development of new protection mechanisms and subsequent attack methods. However, existing attack scenarios fail to capture the real-world threats that logic locking techniques face. This paper investigates a previously overlooked attack scenario against logic locking in which the knowledgeable attacker possesses the locked netlist, knowledge about chip designs (represented as a library of designs), but no working system (as assumed by SAT and other attacks). We also propose a Knowledge-guided Oracle-Less Attack (KOLA) which leverages an adversary's prior knowledge represented by a library of previously encountered designs and employs commercial synthesis tools to measure similarity between the locked design and each library design. Our approach is applicable to both direct locked versions of library designs and modified/upgraded locked designs derived from the library. Experimental results demonstrate KOLA's ability to identify locked designs, even when significant changes have been made from the original library designs. This analysis sheds light on the vulnerabilities of logic locking techniques against knowledgeable attackers even if a working system is unavailable, highlighting the need for enhanced logic locking techniques that are able to thwart attacks from knowledgeable fab houses which can unlock systems even if a working system is unavailable.

Keywords Logic locking · Oracle-less attack · KOLA · Untrusted foundry · Knowledge library

1 Introduction

The rising cost of maintaining IC fabrication facilities has led many chip designers to outsource their fabrication to offshore foundries. However, these foundries are not under the designer's control, which can pose a security risk to the IC supply chain. Malicious end users or other adversaries may also attempt to steal design secrets from fabricated ICs. Logic locking is a technique for protecting intellectual property (IP) in chip designs from being exposed to adversaries [1]. It involves adding a secret key input to the circuit, which is known only to the designer. The circuit will only function properly when the correct key is applied. There are various types of logic locking mechanisms, which vary in their complexity and effectiveness. Some of the simplest mechanisms involve inserting XOR/XNOR gates into the design netlist [2]. More advanced mechanisms use VLSI testing principles to produce high corruption at the output bits when an incorrect key is applied [3, 4].

Logic locking is an effective way to protect IP in chip designs. It is relatively easy to implement and can be used to protect a wide range of IP. However, it is important to note that no security mechanism is perfect. Multiple attack methods against logic locking have been developed. The most prominent type of attacks on logic locking is the Boolean satisfiability-based (SAT) attacks [5], which was able to defeat all logic locking techniques before it. The SAT attack is an oracle-guided attack, where the oracle is a working system that has been configured with the correct key and acts as a black box. Correct input–output relationship can be queried from the oracle. In each iteration of the SAT attack, the attack

✉ Abir Akib
aakib@umd.edu

Yuntao Liu
yule24@lehigh.edu

Ankur Srivastava
ankurs@umd.edu

[1] Department of Electrical and Computer Engineering, University of Maryland, College Park, Maryland 20742, USA

[2] Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015, USA

creates a SAT problem based on the locked netlist and results from querying the oracle. By solving the SAT problem iteratively, the key search space is reduced, and the correct key will finally be found. In addition to the SAT attack, there are other attacks, such as removal attack [6], GNNUnlock [7], SAIL [8], and desynthesis attack [9]. Unlike the SAT attack, these attacks analyze only the locked netlist and do not need an oracle. Consequently, they do not guarantee to find the correct key. It is noteworthy that neither type of existing attacks has accounted for the knowledge and experience that an untrusted fab adversary may have. In this work, we will investigate how the knowledge of known designs can reshape the landscape of attacks on logic locking.

1.1 A Reality Check on the Attacks Against Logic Locking

The development of logic locking techniques has been a never-ending game of cat and mouse. New logic locking techniques are developed to protect designs, but attackers quickly find ways to defeat them. As a result, the focus of many new logic locking techniques is on counteracting the latest attack techniques. Each attack method has an underlying *attack scenario* which describes the resources and capabilities that the attacker has. It is important to consider the attacker model when evaluating the security of a logic locking technique. A realistic attacker model will take into account the resources and capabilities that are actually available to an attacker. Existing attacks on logic locking mainly consider two attack scenarios:

- **Scenario 1**: The attacker only has the locked netlist. Under this scenario, the attacker can analyze the structure and the functionality of the locked netlist. Examples include removal attack [6], GNNUnlock [7], SAIL [8], and desynthesis attack [9].
- **Scenario 2**: The attacker has an *activated chip* in addition to the locked netlist. An activated chip is a locked chip on which the correct key is applied. The argument for the availability of an activated chip is that it can be procured from the open market. SAT-based attacks usually belong to this scenario [5, 10–12].

These two scenarios do not give us a complete picture of the threats faced by the logic locking technique. Scenario 1 does not account for any past experience of the attacker. The untrusted foundry may have manufactured similar designs in the past. For example, the foundry may have seen an older and unprotected version of the design. By comparing the locked design with the previously seen designs, the attacker may be able to find the intent of the design. Again, it is not always fair to assume the availability of an activated chip, especially when it comes to non-consumer (e.g., military) chips. We address these issues by looking into a third attack scenario.

- **Scenario 3**: The attacker has a locked netlist and knowledge about chip designs in general (represented by a library of previously seen designs).

Scenario 3 captures an experienced attacker without access to an activated chip. To our best knowledge, no attack technique on logic locking has considered this scenario so far. In this work, we examine how a pre-existing knowledge library can be utilized by an attacker to decipher the locked designs.

1.2 Contribution of This Work

We propose KOLA, an attack methodology that captures the experienced and knowledgeable attacker of Scenario 3. KOLA is applicable both when the locked design is directly a locked version of a library design and when the locked design is a modified/upgraded and locked version of a library design. To determine the identity of a locked design, we use commercial synthesis tools to measure the similarity between the locked design and each library design. Our experiment results show that KOLA can identify the locked design even if it is derived from one of the library designs with significant changes.

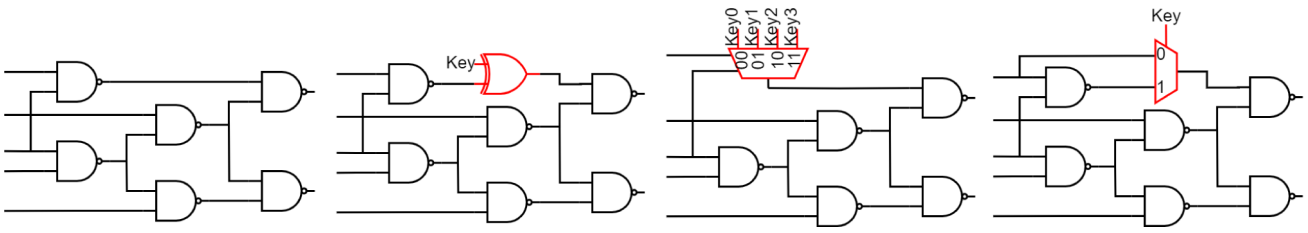
2 Background and Related Works

2.1 Logic Locking

Logic locking is a technique used in the design phase to protect the IP from an adversary based on an untrusted foundry as well as from malicious users [13]. This technique involves the insertion of extra gates and inputs called keys to the original netlist to make it secure [4]. Only the IC designer has the correct key combination which can be loaded into the circuit to make it work. The circuit performs its desired operation when correct key is provided and shows erroneous results in case of a wrong input. An adversary might have the locked netlist and the working system available, but as long as they do not decipher the key, the IP is protected and overproduction is not possible.

A wide range of logic locking techniques has been developed. These techniques can be broadly classified into the three categories that primarily differ in their key insertion strategy [14]. Figure 1 shows the classes of logic locking techniques.

- XOR/XNOR-based logic locking [2, 3, 15]: XOR/XNOR-based logic locking is one of the most popular tech-



(a) Unobfuscated Circuit **(b)** XOR/XNOR-based Locking **(c)** LUT-based Locking **(d)** MUX-based Locking

Fig. 1 Logic locking techniques

niques owing to its simplicity. In this method, a wire is replaced by an XOR/XNOR gate. The other input of the XOR/XNOR gate is a key. The key thus has the capability of flipping the signal of the wire. The logic to the input of the XOR/XNOR gate is so conditioned that some instances would yield correct output when the wire is flipped, and some instances would yield incorrect output for the same. Thus, providing a wrong key pattern would generate a wrong output.

- LUT-based logic locking [16–19]: LUT-based locking techniques introduce a reconfigurable part replacing some of the logic circuits. For correct operation, the correct bit configuration is loaded into the LUTs while wrong bit configurations implement different functionality than the locked design.
- MUX-based logic locking [4]: Multiplexers are used as key gates where one input of multiplexers is an actual wire and the other input, called the false input, is a wire connected from a different logic. The select signal is the key for choosing the correct wire.

2.2 Attacks on Logic Locking

With the advancement of logic locking techniques, there has been substantial work in attacks on logic locking as well, giving rise to a cat-and-mouse chase between the attack and defense counterparts. The attacker aims to break down obfuscation techniques, and the defense team in turn improves their techniques to minimize the security vulnerabilities to be exploited by the attacks. In order to evaluate the security of the logic locking mechanism, researchers have proposed numerous attack methods. Each of those attack methods requires an adversary to have a certain level of access to the design. We have proposed three levels of access as follows:

Black-box access: Under this scenario, the adversary has an input–output oracle access to a locked chip.

Gray-box access: An adversary having gray-box access can observe some internal registers of target chip through scan chain alongside input–output access.

No oracle access: Under this scenario, the adversary has a reverse-engineered netlist of the locked design but does not have access to an activated chip. Thus, the adversary is incapable of observing the correct input–output pattern of the locked chip.

In this section, we will discuss attacks on logic obfuscation with different levels of access.

2.2.1 Black-Box Attack

Under this scenario, the locked chip's primary inputs and outputs are accessible to the attacker, but the access to the internal registers is disabled. For a combinational circuit, such access can help launch a SAT attack [5]. SAT attack is one of the most popular attacks against logic obfuscation which requires input–output access to the circuit and locked circuit netlist. The attack finds some distinguished inputs for which different candidate key values produce different outputs. Since only one of the outputs can be correct and can be known by querying the oracle, it prunes out the incorrect keys. Every iteration of SAT finds a distinguished input, and as a result, the space of possible correct keys is reduced. This goes on till no new distinguished inputs can be found, and at this point, the remaining keys are the correct keys. Inaccessibility of internal flip-flops poses some complications in attacks on sequential designs as full combinational functionality is not known. Sequential circuits can still be attacked by sequentially unrolling the locked circuit [20]. A bounded model checker is used to rule out sets of incorrect keys by finding distinguishing input sequences. Attack runtime can be reduced by incremental unrolling and integrating the results with an incremental SAT solver.

2.2.2 Gray-Box Attack

Gray-box attack assumes that an adversary has all the access that an adversary having black-box access has. In addition, an adversary with gray-box access can access the values of internal registers through the scan chain. As a result, full combinational functionality of a design is exposed enabling SAT attack. Internal access to all flip-flops allows the attacker to use flip-flops as primary input/outputs which enable a sequential circuit to be broken into combinational

sub-circuits for launching a SAT attack. Full-lock is an obfuscation technique that resists SAT-style attacks by including a large SAT-hard instance in the design [21]. The instance is a switching network whose output is the permutation of the inputs, and keys are configuration bits which determine the path of signals through the network. The sensitization attack on such routing-based obfuscation also assumes scan chain access. Another form of gray-box access is physical probing where imaging techniques such as electro-optical probing or electro-optical frequency mapping are used. The voltage of the technology node changes its reflectivity, so oscillating voltages can be detected which has been used to detect the secret key of obfuscated circuits. The combined logical and physical attack (CLAP) [22] observes both oracle output and probe results increasing the information utilized by the attacker. The physical portion of the CLAP attack can probe any die feature. To do this, distinguished inputs are used to create voltage oscillations. Distinguished inputs consist of two primary input vectors. The primary input vectors are so chosen that for 1 key, the probed node has the same value for both primary inputs, but for another key, the node changes the value when the input changes. While probing the oracle, 1 key that will not match observed behavior is eliminated. A SAT-style attack is executed for the logical portion of the CLAP attack, but the keys eliminated during the physical portion are excluded from consideration. If the attack still cannot find the correct key after several iterations of the SAT attack, the attacker can return to the physical attack. The attacker can switch between the two portions to get a solution.

2.2.3 Oracle-Less Attack

Under this scenario, the adversary has access to a reverse-engineered locked netlist only. The adversary does not have any activated chips. Desynthesis attack is one of the oracle-less attacks that re-synthesizes the locked circuit for different key guesses and compares those with the locked design. The key configuration which creates the most similarity of the re-synthesized circuit with the locked one is considered the correct key guess [9]. Partial keys can be extracted from this attack, and scalability is an issue. SWEEP attack, similar to desynthesis attack, re-synthesizes the locked circuit with a static key input and analyzes the synthesis report to identify the structural design features that are correlated to correct keys [23]. Machine learning has been used for the detection and tracing of features. SWEEP does not work on XOR/XNOR-based obfuscation. Another machine learning–based attack SAIL exploits on local and predictable changes in circuit topology incorporated by obfuscation to retrieve the gate-level structure of the obfuscated design [8]. This attack is only applicable for XOR/XNOR-based obfuscation. Desynthesis, SWEEP, and SAIL attacks assume the adversary has knowledge of and access to the exact obfuscation tool and transformation used during logic locking. In practice, the design house can employ any one of several obfuscation strategies, and inferring the exact toolchain from a locked netlist is non-trivial. Topology-guided attack works on the observation that some logical functions like adders and counters are repeated in a logic cone [24]. It attempts to find repeated functions within a circuit and recover the original circuit when key gates are placed in the repeated functions. Its scope is limited to XOR/XNOR-based obfuscation, and the attack is topology dependent. These are some of the well-known oracle-less attacks.

In the real world, adversaries are likely experienced engineers and/or institutions who not only have the tools and resources needed to extract secrets from a chip, but also have years of experience dealing with different designs. Attacks under this scenario focus on mapping the obfuscated netlist with the existing designs in the library. One approach in this direction is finding the structural similarity between the obfuscated netlist and the library netlist. A similarity-based circuit partitioning algorithm has been proposed by Cheng et. al. [25] where the similar sub-circuits existing in different netlists have been extracted. Another way of exploiting the structural similarities is using the graph similarity algorithm presented by Fyrbiak et. al. [26]. A netlist can be expressed in the form of a directed graph, and the spectral distance between the graphs of two netlists is an indicator of how closely related the two netlist graphs are. Such structural similarity-based attack methods rely on topological information and might be unsuitable in cases where the library design and the obfuscated netlist have been synthesized differently.

Under this scenario, a functional similarity mapping-based attack is a more appropriate approach in determining circuit similarity due to the fact that functional similarity between circuits is agnostic to their topology. This work proposes KOLA, which leverages commercial synthesis tools for functional mapping of exposed parts of obfuscated circuits with designs in the library to determine the intent of the locked design. KOLA is both topology and locking mechanism-agnostic, which differentiates KOLA from existing oracle-less attacks.

3 Threat Model

This work assumes an adversary under Scenario 3 from Section 1.1. The adversary is assumed to be based on a foundry who has acquired knowledge about designs from their prior encounters. The adversary has the following:

- Access to the netlist of the locked circuit acquired by reverse engineering of the layout.

- A rich functional library of prior designs the foundry might have fabricated. The library is a functional representation of the past designs the adversary has encountered and is topology-agnostic.

The adversary's prior knowledge is represented by a functional library built from prior encounters with designs. The libraries being functional in nature have no particular topological information stored in them. This attack surface leads to two sub-scenarios:

- **Scenario 3.1**: Adversary has a locked design which is directly a locked version of a library design: The adversary under this scenario has encountered the original (unprotected) version of the locked design before and has it in their library.
- **Scenario 3.2**: Adversary has a locked design which is either a modified version of one of the designs of library, or which can be created by evolving one of the designs existing in library: The same Boolean functionality can be implemented in a variety of different ways to improve power consumption, speed of operation, etc. A given design can be evolved to be used in a different, larger, or smaller context. Under this scenario, the adversary has not seen the exact unprotected version of the locked design, but might have seen designs which are either a different implementation of the functionality or evolved from one of the designs from the library.

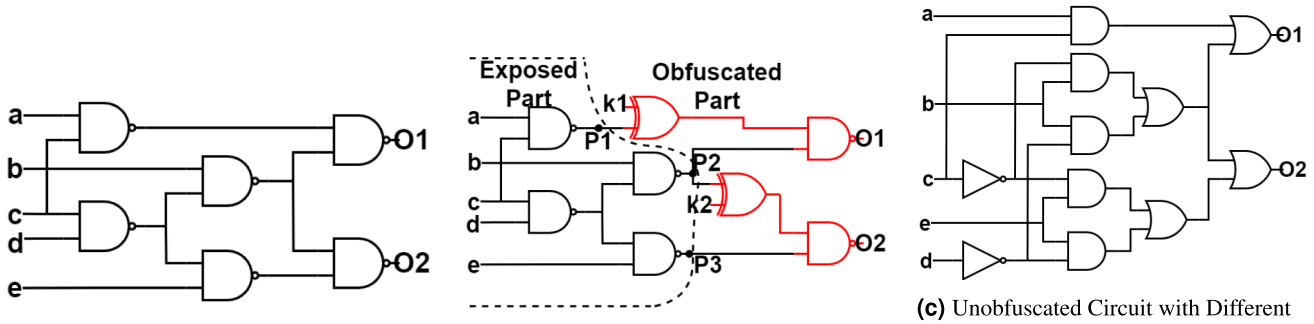
The goal of the adversary is to find the library design from which the locked design has originated with the highest probability.

4 Attack Methodology

In this work, we propose KOLA, an attack methodology for Scenario 3 attackers who utilize their prior knowledge represented as a library of designs to discover the identity of the locked design. The flow of the KOLA is as follows. We use weak division to find similarities between the locked design and each library design. To account for Scenario 3.2 where the locked design is evolved from one of the library designs, we develop a library evolution algorithm to expand the library. Details of the attack techniques are described in the rest of this section.

4.1 Weak Division

For most of the state-of-the-art logic obfuscation techniques, irrespective of whether the technique is XOR/XNOR, LUT, or MUX-based, a locked design has an obfuscated part and a key-independent or exposed part as depicted in Fig. 2b. The exposed part of a design leaks two classes of information—topological and functional. Cheng et. al. have adapted a similarity-based circuit partitioning [25] which could be leveraged to exploit topological information leak. However, topology is dependent on synthesis primitives and also on the fact that the same functionality can be implemented in different ways to achieve different design goals. So topological information extracted from the exposed portion becomes implementation dependent. There are a wide variety of ways a design can be implemented, and so exploiting the topological information becomes complex and thus not feasible. Functional information, on the other hand, is independent of implementation and can be extracted from the exposed part itself. KOLA needs to find the boundary between the exposed and obfuscated part of the design and find out what percentage of nodes in the boundary have a functionality which are sub-function of outputs in their design library. The attacker thus makes this comparison or matching with all designs of their library and finds a matching percentage, which is the percentage of the number of exposed nodes which are sub-functions of a specific library design with respect to the total number of exposed nodes. The greater matching percentage is an indicator of a more similar design, and using the matching percentage, the probability of a locked design being the same as a design from the library is calculated.



(a) Original Circuit **(b)** Locked Circuit **(c)** Unobfuscated Circuit with Different Topology

Fig. 2 Obfuscated and exposed parts of a locked circuit

Mathematically, let P represent an exposed functionality from a locked design and F a Boolean function from a library design. If we can represent F as $F = (P \wedge Q) \vee R$, where P and R are some Boolean functionalities, $Q \neq 0$ and $R \neq 1$, then we can claim that P is a sub-function of F. A particular library design has multiple output pins each with unique functionality, and a locked design also has multiple exposed functionalities. We define the matching index as the percentage of functionalities of exposed nodes of the locked design that can be mapped to the output functionalities of a particular library design. For example, in Fig. 2a, the Boolean expression of the outputs:

$$O_1 = \neg(\neg(a \wedge c) \wedge \neg(b \wedge \neg(c \wedge d))) \quad (1)$$

$$O_2 = \neg(\neg(b \wedge \neg(c \wedge d)) \wedge \neg(e \wedge \neg(c \wedge d))) \quad (2)$$

In Fig. 2b, the expressions at the edge of the exposed portion are as follows:

$$P_1 = \neg(a \wedge c) \quad (3)$$

$$P_2 = \neg(b \wedge \neg(c \wedge d)) \quad (4)$$

$$P_3 = \neg(e \wedge \neg(c \wedge d)) \quad (5)$$

Here, since

$$O_1 = P_1 \wedge \neg(b \wedge \neg(c \wedge d)) \quad (6)$$

$$O_2 = P_2 \wedge \neg(e \wedge \neg(c \wedge d)) \quad (7)$$

$$O_2 = P_3 \wedge \neg(b \wedge \neg(c \wedge d)) \quad (8)$$

P_1, P_2, and P_3 are sub-functions of O_1 and O_2, i.e., the exposed functionalities are parts of the original circuit. Thus, if the obfuscated design is a part of the library, it will have a 100% match with the exposed portion of the locked design. Again, an unobfuscated design might have a different topological representation as shown in Fig. 2c. For example,

$$O_1 = (a \wedge c) \vee (b \wedge \neg c) \vee (b \wedge \neg d) \quad (9)$$

$$O_2 = (b \wedge \neg c) \vee (b \wedge \neg d) \vee (e \wedge \neg c) \vee (e \wedge \neg d) \quad (10)$$

yield the same functionality. If the locked circuit is matched with these sets of functionalities, we find

$$O_1 = \neg(P_1) \vee ((b \wedge \neg c) \vee (b \wedge \neg d)) \quad (11)$$

$$O_2 = \neg(P_2) \vee ((e \wedge \neg c) \vee (e \wedge \neg d)) \quad (12)$$

$$O_2 = \neg(P_3) \vee ((b \wedge \neg c) \vee (b \wedge \neg d)) \quad (13)$$

Even with a different topological implementation, a 100% matching index is observed which advocates that this process of identification of the locked design functionality from the library is topology independent.

KOLA thus targets to find whether the exposed functionalities are a sub-function of a library function. On a closer inspection of the problem, resemblance to the algebraic division is apparent where $F = (P * Q) + R$ indicate that if F is divided by P, Q is the quotient, and R is the remainder, the only difference being that the scope of this problem is Boolean in nature. In order to solve such a problem, algebraic division principles have been adopted to implement Boolean division. One of such widely popular algorithms is the weak division algorithm [27]. The weak division algorithm essentially finds out if a Boolean function F can be divided by another function P, and if the division is possible, it finds the quotient Q and the remainder R. For the problem in our case, if weak division is possible, the exposed functionality of the locked design is a sub-function of a functionality in the library design. Algorithm 1 shows a modified implementation of the weak division algorithm for finding such a match.

Algorithm 1 Weak Division Algorithm

```
1: procedure WEAK DIVISION(LibFunction,ExpFunction)
2:     F = list[minterms(LibFunction)]
3:     P = list[minterms(ExpFunction)]
4:     for P_i in P do
5:         Create empty list V_i
6:         for F_j in F do
7:             if F_j contains all literals of P_i then
8:                 new_minterm = literals in F_j but not in P_i
9:                 V_i = V_i ∪ new_minterm
10:    if any minterm is common across all V then
11:        M=1
12:    return M
```

In Boolean algebra, a minterm is a product term where each variable occurs only once in either its true or complemented form. In the algorithm, F is a set of all minterms in the sum-of-product (SOP) representation of a library function, and P is the set of minterms in the SOP representation of the exposed function. Each minterm of exposed functionality is compared with all the minterms of library functionality. If there exists any minterm in F (F_j) which is a superset of all the literals of a minterm in P (P_i), then a new minterm is developed containing only the literals present in F_j and not in P_i. This new minterm, if it exists, is stored in set V. The process is repeated for all minterms of P. If there exists any common minterm across all elements of V, that minterm is the quotient Q which indicates weak division is possible.

Further analysis of the weak division algorithm implemented in this way shows that the time complexity is exponential. It iterates through all the minterms of the exposed functionality, and for each of the minterms, it iterates through all the minterms of library functionality. Since the number of minterms for both cases is exponential in terms of the number of input variables, the time complexity of the weak division algorithm is very high. Literature has some

quicker algorithms which bring down the complexity, but they require the input minterms to be provided in order of their binary encoding [28], which itself is a complex process for a large design, and it renders weak division infeasible.

All the state-of-the-art synthesis tools have the capabilities of optimizing a design to minimize hardware overheads, which was the key motivation behind the development of algebraic division algorithms. These synthesis tools can be used for quicker and more efficient analysis. If the locked design originates from a library design, each exposed functionality should be a sub-functionality of the original design. We synthesize the exposed and the original functionalities simultaneously. As a synthesis tool tries to minimize the area of the synthesized design, the gates that implement the exposed design should be a subset of those of the original design. In other words, if we find that the output of the exposed functionality is reused by a library design's functionality, then the exposed functionality may belong to the library design. Using a synthesis tool, the computational complexity is dramatically reduced. Implementation of the weak division method requires an exponential time process to be repeated $n * m$ times where n is the number of exposed functionalities and m is the number of functionalities in a library design. Using a synthesis tool reduces the complexity to time required to perform n synthesis.

4.1.1 Time Complexity

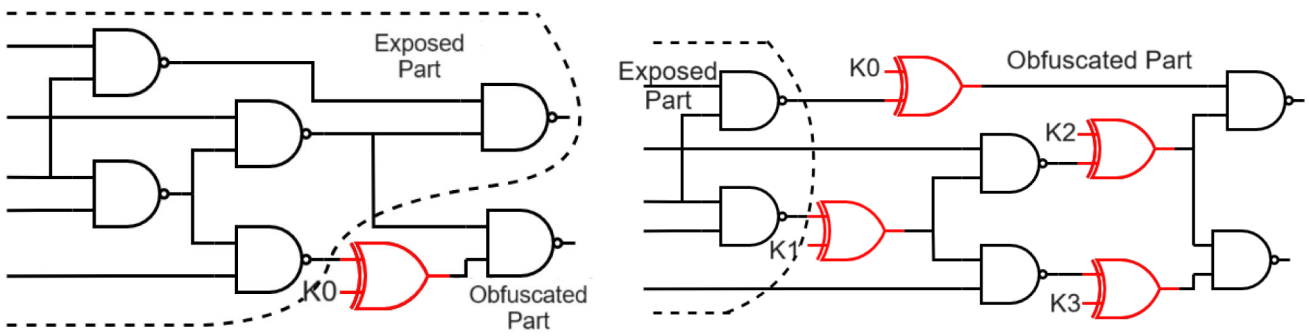
KOLA avoids the exponential cost of brute-force Boolean weak division by using a synthesis-guided functional matching approach, reducing the practical complexity to roughly $O(n \times L \times T)$, where n is the number of exposed functionalities, L is the library size, and T is the maximum time taken for a single synthesis comparison. In practice, runtime depends both on the size of the exposed portion and on the raw key size. Larger keys increase n but decrease T. While industrial-scale designs with millions of nodes pose challenges, KOLA can scale well because each exposed functionality to library candidate comparison is independent and can be run in parallel across multiple machines or compute clusters. This allows a foundry-level adversary with parallel resources to scale the attack to large libraries and large designs. Extensive parallelism thus reduces the time complexity to $O(T)$.

4.1.2 Analysis of Information Leakage in Context of Weak Division Based Attack

Even though a matching profile found using such algorithms is a good indicator of which library design has the most probability of being the oracle of the locked design, the success of the algorithm eventually rests on how much information is being leaked by the exposed portion. If the exposed portion has a large fan-in cone, it essentially leaks more information about a circuit. For example, in Fig. 3b, the design is obfuscated using 4 key-bits and the functionalities of the exposed nodes are generic NAND gates which might be common to most of the library design. Thus, such obfuscation will show a high amount of matching with a wide range of library designs, and the locked circuit will be indistinguishable. On the other hand, for obfuscation with 1 key-bit in the example of Fig. 3a, the functionalities are larger and resemble a particular library design. This is an indication that the level of obfuscation is inversely correlated to the level of information leak, and tuning obfuscation to a point where most of the exposed functionalities are generic will reduce information leakage and the effectiveness of the attack.

4.2 Library Creation and Evolution

The library in this context is a model of the adversary's prior knowledge. A foundry-based adversary has had encounters with a lot of big and small designs. Digital design being an incremental development process, smaller designs often form the building blocks of bigger designs. The adversary has the capability of extracting Boolean functionalities from the designs they encounter. The Boolean functionalities make up the knowledge library, which is ever-expanding based on any new designs the adversary encounters. The library essentially



(a) Circuit Locked with 1 key bit

(b) Circuit Locked with 4 key bits

Fig. 3 Information leaked with different key sizes

characterizes the experience of the adversary, and a more experienced adversary will have a richer library and thus be a stronger attacker. Following the incremental development design process, newer designs are often built on older designs with the following modifications:

- Bit-length or size
- Number of stages of operation
- Nature of one or more operations
- Topology

The adversary has the capabilities of evolving their library to accommodate such modifications in their existing library designs to evolve their library as needed.

4.3 Flow of the Attack

A prerequisite of launching KOLA is that the adversary has a rich functional library which they have built from their experience with different designs. In our threat model, it is assumed that the adversary is an untrusted foundry that has access to the gate-level netlist obtained from the layout. From this netlist, the adversary can identify key gates because they are connected to the tamper-proof memory that stores the key either directly or through intermediate storage elements. This connection allows the adversary to determine which signals constitute the key inputs without knowing their actual values. This assumption is standard in attacks on logic obfuscation [29, 30]. Once these key-related gates are located, the locked netlist is sent through a parser, and the functionalities at every node are extracted. The next step is to identify the boundary of the locked and exposed portions as shown in Fig. 2. From the set of functionalities, the key-independent ones at the boundary of obfuscation are separated for matching. Each of the exposed functionalities is then passed into a weak division. The other input to weak division is all the output functionalities of one library design. In the context of KOLA, weak division is implemented using a synthesis tool, as described in Section 4.1. The weak division algorithm outputs whether the given exposed function is a sub-function of any of the functionalities of the library design. The process is repeated across all designs in the library which yields a matching table showing the level of matching for different library designs for the same locked netlist. The matching table shows the percentage of similarity between the exposed part of the locked design and each library design. The library design with the highest similarity is considered to be the origin of the locked design. The goal of the attack is to identify a small subset of designs from the library which has the highest similarity with the locked design, which will aid attacks mentioned under Scenario 2. The entire attack flow is delineated in Fig. 4.

5 Result and Analysis

To evaluate the effectiveness of KOLA under different scenarios, we have locked some of the netlists from ISCAS-85 benchmarks [31] and some well-known adder and multiplier designs using stripped functionality logic locking (SFLL) [32] and XOR/XNOR-based logic locking using random key insertion with different key sizes. The adversary's library for the evaluation contains c499, c880, c1355, c1908, c2670, 8×8 Dadda multiplier, and 16-bit ripple carry adder. The following sections describe the results in detail:

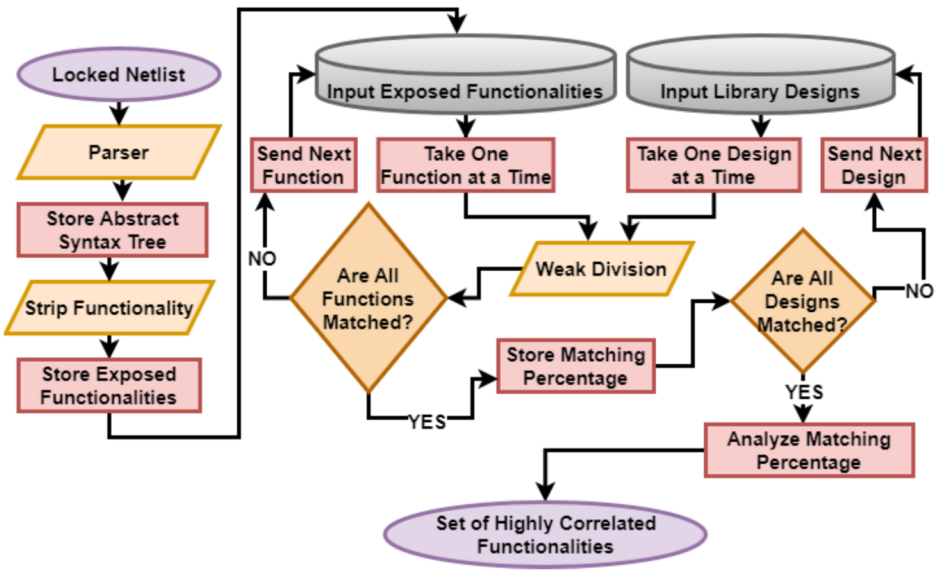


Fig. 4 Attack flow

5.1 Scenario 3.1

Under this scenario, it is assumed that the adversary has encountered the same design in the past and the design exists in the adversary's library. Figure 5 shows the matching percentage of each locked ISCAS benchmark design when compared to different ISCAS library designs. The horizontal axis represents the locked design, the vertical axis represents the matching percentage, and each bar of the histogram represents the matching value for each library design.

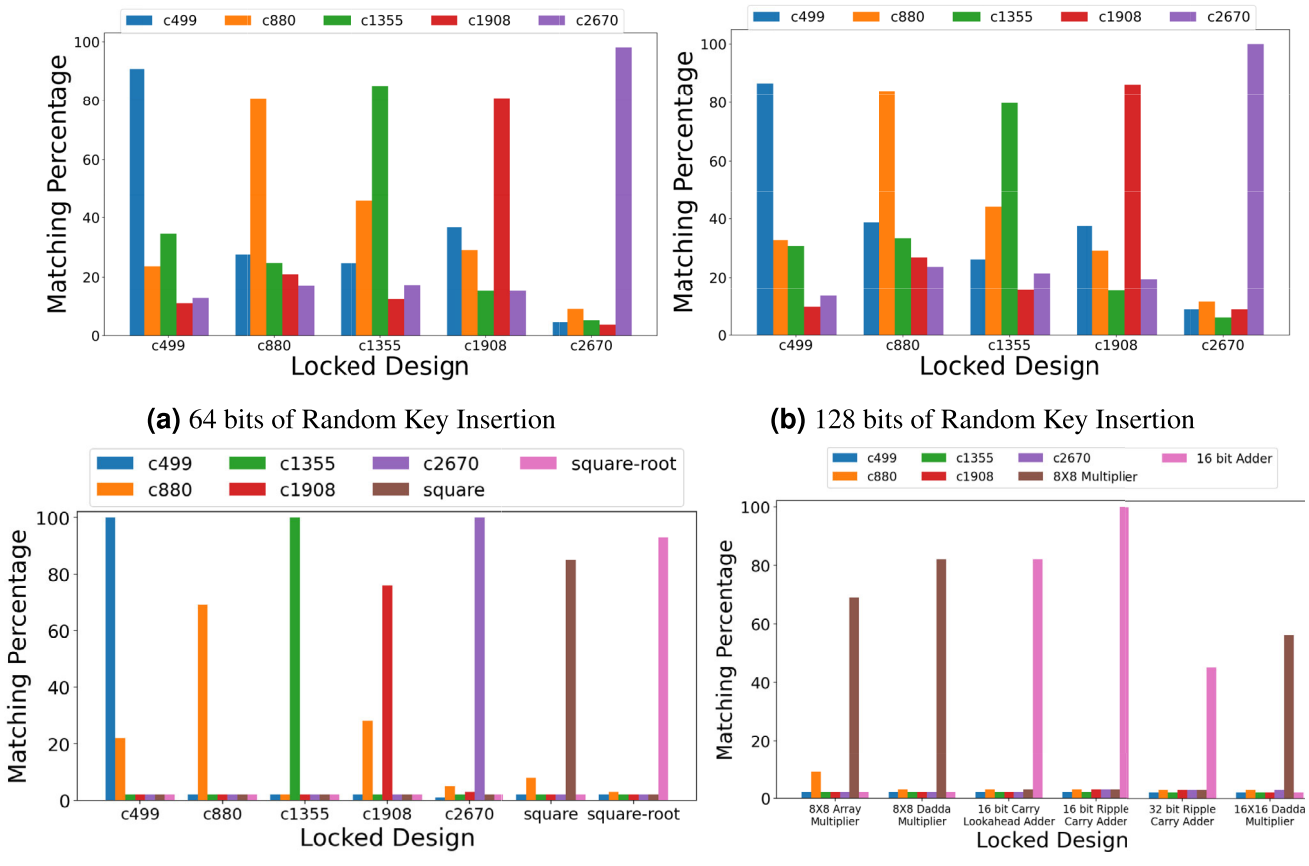
From Fig. 5, it is evident that the highest matching percentage is observed when the locked design is matched with its unprotected design in the library. In all other cases, the matching percentage is low. This indicates that KOLA has been able to identify the unprotected design from the library. Figure 5 a and b illustrate how the matching percentage changes when designs are locked with 64-bit and 128-bit keys, respectively. As the key size increases, the average matching percentage across different library designs also rises, and the gap between the target design and non-target designs becomes smaller. This trend indicates that higher levels of obfuscation reduce the distinguishability of the correct design from other candidates at the cost of increased area overhead. Another noteworthy observation is that for the designs locked using SFLL, there is almost no match with any design other than the correct unprotected library design. The reason behind this is that in SFLL, only a small group of input patterns are protected by stripping their functionalities. Thus, the functionalities of all other input patterns, which may have large fan-in cones, are preserved. Since the exposed parts are not generic, there is a very low percentage of matching with any design that is not the unprotected version of the locked design. This data validates the point raised in Section 4.1 that KOLA relies on the amount of information leaked by the exposed part. Since the amount of information leakage is high, the locked design is more distinguishable when compared with the library designs.

5.2 Scenario 3.2

This scenario can be subdivided into two sub-scenarios.

5.2.1 Adversary Has a Different Implementation of the Design

A difference in topology can be introduced by synthesis primitives or by design requirements in terms of speed and power.



(a) 64 bits of Random Key Insertion

(b) 128 bits of Random Key Insertion

(c) Locked Using SFLL Under Scenario 3.1

(d) Locked Using SFLL Under Scenario 3.2

Fig. 5 Matching percentage between library designs and designs locked with different techniques and key sizes

In order to model such a scenario, two different topologies of 16-bit adders, ripple carry adder and carry lookahead adder, and two different topologies of 8×8 multipliers, Array multiplier and Dadda multiplier, have been locked using SFLL. Functionalities of a 16-bit adder and an 8×8 multiplier have been included in the library. With the new addition to the library, KOLA was launched on all the new locked designs. Figure 5d contains the matching percentage of the new locked designs with respect to all library designs. It is evident that the locked version of both carry lookahead adder and ripple carry adder shows very strong correlation with the adder of the library while all the other designs show almost no correlation. Similarly, only the array and Dadda multipliers show high correlation with the multiplier in the library. This is a clear indication that KOLA is independent of the topology of the implemented design, and an attack is successful if the adversary has a different implementation of the unprotected design in the library.

5.2.2 Adversary Has a Library Design Which Can be Evolved to Generate the Oracle of Locked Design

In order to model such a scenario, a 32-bit Ripple Carry Adder has been locked using the SFLL obfuscation technique. There is no instance of a 32-bit adder in the library. KOLA is launched on the locked 32-bit ripple carry adder with the existing library. From Fig. 5d, it is evident that the locked design shows a good correlation with only a 16-bit adder which is an indication that the functionality of the locked design might be an adder. With that view, the input and output width of the locked design were observed, and two instances of adders from the library were used to implement a 32-bit adder to match the input and output width of the locked design. This evolved design is then included in the library. KOLA is launched on this locked design with only the evolved library design. The locked design shows a 96.77% match with the evolved library design. The same experiment has been replicated with a 16X16 Dadda multiplier, which only matched with the 8×8 multiplier. Extending the 8×8 multiplier to a 16×16 multiplier showed a matching percentage of 81.29%. The experiment indicates that the correct functionality could be developed from evolving the existing designs of the library. This acts as a proof of concept for this scenario.

6 Conclusion and Discussions

This paper presents an oracle-less attack on logic locking techniques exploiting the information leakage from the key-independent exposed portions of locked circuits. KOLA is functional in nature and agnostic to the topology and implementation of the design. The attack methodology takes inspiration from core multi-level synthesis ideas like algebraic division. Experiments show that KOLA is capable of exploiting state-of-the-art logic locking techniques which leak information in key-independent areas of the locked circuits. Existing oracle-less attacks on logic locking did not account for the rich experiences the adversary may have. In this work, we characterize such experiences using a design library, which enabled us to transform the attack surface from a weak key-guessing attack to a strong oracle-guided one. The success of this attack greatly depends on the expertise and experience of the adversary, i.e., the more experienced the adversary, the stronger the attack.

It is possible to counteract KOLA. A possible design technique to evade KOLA would be to introduce dummy logic in the exposed part of the netlist which will diminish the matching percentage when compared with the same design and can also sometimes lead to more matching with a different design. In both cases, the probability of finding the correct design will be reduced. Another countermeasure is to increase the level of obfuscation. Both these techniques come with hardware overheads. A simple, yet effective locking technique can be to insert some key gates very close to the input periphery of the circuit, minimizing the information leakage from exposed nodes. This essentially increases the matching percentage for all the designs, decreasing the probability of finding the correct design without incurring high overhead. KOLA quantifies the information leakage from the exposed areas of a locked circuit and can be applied as a vulnerability assessment tool for the development of future logic locking techniques.

Statements and Declarations

Conflict of Interest The authors declare no competing interests.

Author Contribution Abir Akib contributed to the majority of the theory and implementation of this paper. Yuntao Liu contributed to the formulation of the idea and writing of the paper. Ankur Srivastava supervised the entire project, contributed to the formulation of idea and writing of the paper.

Data Availability No datasets were generated or analysed during the current study.

References

1. Chakraborty A et al (2020) Keynote: a disquisition on logic locking. IEEE Trans Comput Aided Des Integr Circuits Syst 39:1952–1972
2. Roy JA, Koushanfar F, Markov IL (2008) Epic: ending piracy of integrated circuits. In: Design, automation and test in Europe. 1069–1074. https://doi.org/10.1109/DATE.2008.4484823
3. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: DAC Design Automation Conference. 83–89. https://doi.org/10.1145/2228360.2228377
4. Rajendran J et al (2015) Fault analysis-based logic encryption. IEEE Trans Comput 64:410–424. https://doi.org/10.1109/TC.2013.193
5. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: 2015 IEEE international symposium on Hardware Oriented Security and Trust (HOST). 137–143. https://doi.org/10.1109/HST.2015.7140252
6. Yasin M, Mazumdar B, Sinanoglu O, Rajendran, J (2017) Removal attacks on logic locking and camouflaging techniques. IEEE Trans Emerg Topics Comput
7. Alrahis L et al (2021) GNNUnlock: graph neural networks-based oracle-less unlocking scheme for provably secure logic locking. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 780–785
8. Chakraborty P, Cruz J, Bhunia S (2018) Sail: machine learning guided structural analysis attack on hardware obfuscation. In: 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). pp 56–61. https://doi.org/10.1109/AsianHOST.2018.8607163
9. Massad ME, Zhang J, Garg S, Tripunitara MV (2017) Logic locking for secure outsourced chip fabrication: a new attack and provably secure defense mechanism. arXiv:1703.10187
10. Shamsi K et al (2017) AppSAT: approximately deobfuscating integrated circuits. In: 2017 IEEE international symposium on Hardware Oriented Security and Trust (HOST). pp 95–100. https://doi.org/10.1109/HST.2017.7951805
11. Zhou H, Jiang R, Kong S (2017) CycSAT: SAT-based attack on cyclic logic encryptions. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp 49–56. https://doi.org/10.1109/ICCAD.2017.8203759
12. Chakraborty A, Liu Y, Srivastava A 2018 TimingSAT: timing profile embedded SAT attack. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). ACM, pp 1–6
13. Dupuis S, Flottes M-L (2019) Logic locking: a survey of proposed methods and evaluation metrics. J Electron Test 35:1–19. https://doi.org/10.1007/s10836-019-05800-4
14. Kamali HM, Azar KZ, Farahmandi F, Tehranipoor M (2022) Advances in logic locking: past, present, and prospects. Cryptology ePrint Archive, Paper 2022/260. https://eprint.iacr.org/2022/260
15. Yasin M, Rajendran JJ, Sinanoglu O, Karri R (2016) On improving the security of logic locking. IEEE Trans Comput Aided Des Integr Circuits Syst 35:1411–1424. https://doi.org/10.1109/TCAD.2015.2511144
16. Mardani Kamali H, Zamiri Azar K, Gaj K, Homayoun H, Sasan A (2018) LUT-lock: a novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection. In: 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp 405–410. https://doi.org/10.1109/ISVLSI.2018.00080
17. Liu B, Wang BE (2014) Reconfigurable logic for ASIC design obfuscation against supply chain attacks. In: Design, Automation & Test in Europe Conference & Exhibition (DATE). pp 1–6. https://doi.org/10.7873/DATE.2014.256
18. Khaleghi S, Kai Da Z, Wenjing R (2015) IC piracy prevention via design withholding and entanglement. In: The 20th Asia and South pacific design automation conference. pp 821–826. https://doi.org/10.1109/ASPDAC.2015.7059112
19. Khaleghi S, Zhao KD, Rao W (2015) IC piracy prevention via design withholding and entanglement. In: The 20th Asia and South pacific design automation conference. pp 821–826. https://doi.org/10.1109/ASPDAC.2015.7059112
20. Massad ME, Garg S, Tripunitara M (2017) Reverse engineering camouflaged sequential circuits without scan access. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp 33–40. https://doi.org/10.1109/ICCAD.2017.8203757
21. Kamali HM, Azar KZ, Homayoun H, Sasan A (2019) Full-lock: hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks. In: 2019 56th ACM/IEEE Design Automation Conference (DAC). pp 1–6
22. Zuzak M, Liu Y, McDaniel I, Srivastava AA (2022) Combined logical and physical attack on logic obfuscation. In: Proceedings of the 41st IEEE/ACM international conference on computer-aided design. (Association for Computing Machinery, New York, NY, USA, 2022), ICCAD '22. https://doi.org/10.1145/3508352.3549349
23. Alaql A, Forte D, Bhunia S (2019) Sweep to the secret: a constant propagation attack on logic locking. In: 2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). pp 1–6. https://doi.org/10.1109/AsianHOST47458.2019.9006720
24. Zhang Y, Jain A, Cui P, Zhou Z, Guin U (2021) A novel topology-guided attack and its countermeasure towards secure logic locking. J Cryptogr Eng 11:213–226. https://doi.org/10.1007/s13389-020-00243-6
25. Cheng Y, Wang Y, Li H, Li X (2015) A similarity based circuit partitioning and trimming method to defend against hardware trojans. In: 2015 IEEE computer society annual symposium on VLSI. pp 368–373. https://doi.org/10.1109/ISVLSI.2015.93
26. Fyrbiak M, Wallat S, Reinhard S, Bissantz N, Paar C (2020) Graph similarity and its applications to hardware security. IEEE Trans Comput 69:505–519. https://doi.org/10.1109/TC.2019.2953752
27. Hachtel GD (1996) Logic synthesis and verification algorithms / by Gary D. Fabio Somenzi (Kluwer Academic Publishers, Boston, Hachtel
28. Villa T, Brayton RK, Sangiovanni-Vincentelli AL, Crama Y, Hammer PL (2010) Synthesis of multilevel Boolean networks
29. Mondal A, Zuzak M, Srivastava A (2020) StatSAT: a Boolean satisfiability based attack on logic-locked probabilistic circuits. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp 1–6. https://doi.org/10.1109/DAC18072.2020.9218703
30. Azar KZ, Kamali HM, Homayoun H, Sasan A (2019) SMT attack: next generation attack on obfuscated circuits with capabilities and performance beyond the sat attacks. IACR Trans Cryptograph Hardware Embedded Syst 97–122
31. Brglez F (1985) A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran
32. Yasin M et al (2017) Provably-secure logic locking: from theory to practice. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. (Association for Computing Machinery, New York, NY, USA, 2017) CCS '17, pp 1601–1618. https://doi.org/10.1145/3133956.3133985