Provably-Secure Logic Locking: From Theory To Practice

Muhammad Yasin
New York University, New York, USA
yasin@nyu.edu

Abhrajit Sengupta
New York University, New York, USA
as9397@nyu.edu

Mohammed Thari Nabeel
New York University Abu Dhabi, UAE
mtn2@nyu.edu

Mohammed Ashraf
New York University Abu Dhabi, UAE
ma199@nyu.edu

Jeyavijayan (JV) Rajendran
The University of Texas at Dallas/
Texas A&M University, Texas, USA
jv.ee@utdallas.edu

Ozgur Sinanoglu
New York University Abu Dhabi, UAE
os22@nyu.edu

ABSTRACT

Logic locking has been conceived as a promising proactive defense strategy against intellectual property (IP) piracy, counterfeiting, hardware Trojans, reverse engineering, and overbuilding attacks. Yet, various attacks that use a working chip as an oracle have been launched on logic locking to successfully retrieve its secret key, undermining the defense of all existing locking techniques. In this paper, we propose stripped-functionality logic locking (SFLL), which strips some of the functionality of the design and hides it in the form of a secret key(s), thereby rendering on-chip implementation functionally different from the original one. When loaded onto an on-chip memory, the secret keys restore the original functionality of the design. Through security-aware synthesis that creates a controllable mismatch between the reverse-engineered netlist and original design, SFLL provides a quantifiable and provable resilience trade-off between all known and anticipated attacks. We demonstrate the application of SFLL to large designs (>100K gates) using a computer-aided design (CAD) framework that ensures attaining the desired security level at minimal implementation cost, 8%, 5%, and 0.5% for area, power, and delay, respectively. In addition to theoretical proofs and simulation confirmation of SFLL's security, we also report results from the silicon implementation of SFLL on an ARM Cortex-M0 microprocessor in 65*nm* technology.

CCS CONCEPTS

•Security and privacy →Security in hardware; Hardware attacks and countermeasures; Hardware reverse engineering; *Hardware security implementation;* **•Hardware →Logic synthesis;** *Electronic design automation;*

KEYWORDS

Design-for-trust, IP piracy, hardware Trojan, reverse engineering, logic locking, Boolean satisfiability (SAT)

CCS'17, Oct. 30–Nov. 3, 2017, Dallas, TX, USA.
 ISBN 978-1-4503-4946-8/17/10…$15.00
DOI: http://dx.doi.org/10:1145/3133956:3133985

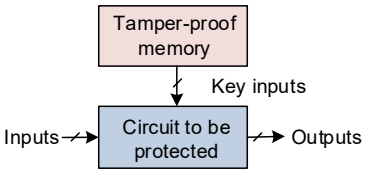


Figure 1: An abstract representation of a logic-locked design. Only on applying the secret key, the design produces correct outputs; otherwise, incorrect outputs are produced.

1 INTRODUCTION

1.1 IP piracy and reverse engineering

The increasing cost of IC manufacturing has forced many companies to go fabless over the years. With the outsourcing of IC fabrication in a globalized/distributed design flow including multiple (potentially untrusted) entities, the semiconductor industry is facing a number of challenging security threats. This fragility in the face of poor state-of-the-art IP protection has resulted in hardware security vulnerabilities such as IP piracy, overbuilding, reverse engineering, and hardware Trojans [9, 13, 19, 20, 37, 39, 45, 47–49].

To address these issues most effectively at the hardware level [32], a number of hardware design-for-trust (DfTr) techniques such as IC metering [1, 22, 23], watermarking [17, 18, 21, 31], IC camouflaging [3, 4, 27, 28, 35, 46, 51, 56, 62], split manufacturing [14, 16], and logic locking [34, 36, 38, 40, 41, 52, 53, 55, 61, 63] have been proposed. Logic locking, in particular, has received significant interest from the research community, as it can protect against a potential attacker located anywhere in the IC supply chain, whereas other DfTr techniques such as camouflaging or split manufacturing can protect only against a limited set of malicious entities as shown in Table 1. Mentor Graphics, a major CAD tool provider, has announced the launch of TrustChain, a framework to support logic locking and camouflaging [26, 42].

1.2 Logic locking: defenses and attacks

Logic locking inserts additional logic into a circuit, locking the original design with a secret key. In addition to the original inputs, a locked circuit has *key inputs* that are driven by an on-chip tamper-proof memory [15, 50], as illustrated in Fig. 1. The additional logic may consist of XOR gates [34, 36, 38] or look-up tables (LUTs) [5]. Fig. 2 presents the IC design flow incorporating logic locking. The locked netlist passes through the untrusted design phases. Without the secret key (i) the design details cannot be recovered (for reverse-engineering), and (ii) the IC is not functional,

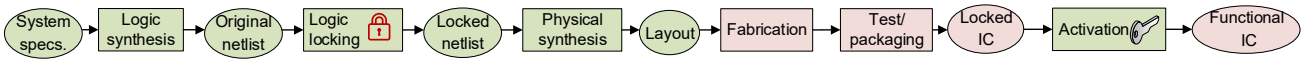


Figure 2: Locking and activation of an IC in IC design flow. The red regions denote untrusted entities; the green regions represent trusted entities.

Table 1: Protection offered by DfTr techniques against untrusted entities in the IC supply chain.

Techniques	Foundry	SoC Integrator	Test	User
IC metering [1, 22, 23]	✗	✓	✓	✓
Watermarking [17, 18, 21, 31]	✗	✗	✗	✓
IC camouflaging [3, 4, 27, 28, 35, 46, 51]	✗	✓	✓	✓
Split manufacturing [14, 16]	✓	✗	✗	✗
Logic locking [34, 36, 38, 52, 55]	✓	✓	✓	✓

i.e., it produces incorrect outputs (for over-production). A locked IC needs to be activated by loading the secret key onto the chip's memory.

Traditional logic locking techniques choose key gate locations based on various gate selection algorithms, such as random (RLL) [38], fault analysis-based (FLL) [5, 36], and strong-interference-based logic locking (SLL) [34, 59]. Over the years, many key-recovery attacks have been mounted that exploit the vulnerabilities of logic locking techniques [33, 34, 44, 54, 60]. A summary of these attacks is presented in Table 2.

A powerful attack that broke all the logic locking techniques existing then is *Boolean satisfiability (SAT)-based key-pruning attack*, referred to as *SAT attack*. The attack is based on the notion of *incorrect key elimination* using *distinguishing input patterns (DIPs)* [44]. DIPs are computed using a miter circuit constructed using two copies of the locked netlist; the two circuits share the primary inputs but have different key inputs. A DIP is found when the two copies of the locked netlist differ in their outputs. A functional IC with the secret key loaded in its memory is used as an *oracle* to identify the incorrect keys in an iterative fashion. The computational complexity of the attack is expressed in terms of the number of DIPs generated by the SAT attack [44]. The latest research works on logic locking have focused on defending against the SAT attack [52, 55, 57].

1.3 SAT attack resilient logic locking

Two SAT attack resilient logic locking techniques—SARLock [55] and Anti-SAT [52] (see Fig. 3)—have been recently proposed. They both use one-point functions to obtain resilience against SAT attacks. SARLock corrupts/inverts the output of the circuit for all the incorrect keys at exactly one input pattern that is different for each incorrect key. The correct key values are hardcoded in logic gates to *mask* the output inversion for the correct key [55]. Anti-SAT employs two complementary logic blocks that converge at an AND gate. The output of the AND gate is always 0 only for the correct

Table 2: Attack resiliency of logic locking techniques against the existing attacks. ✗ denotes susceptibility to the attack and ✓ denotes resilience.

Attack	RLL [38]	FLL [5, 36]	SLL [34]	AntiSAT [52]	SARLock [55]	TTLock [61]	Proposed SFLL
Sensitization [34]	✗	✗	✓	✓	✓	✓	✓
SAT [44]	✗	✗	✗	✓	✓	✓	✓
AppSAT [40]	✗	✗	✗	✗	✗	✗	✓
Removal/SPS [57]	✓	✓	✓	✗	✗	✓	✓

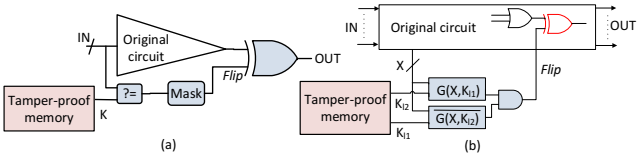


Figure 3: (a) SARLock [55]. (b) Anti-SAT [52]. In both techniques, an incorrect key may assert the *flip* signal, producing incorrect outputs. Source: [61].

key; otherwise, it may be 1. Its output corrupts an internal node in the original design for an incorrect key, to produce incorrect outputs.

SARLock can be intertwined with one of the gate selection-based logic locking techniques, such as RLL, FLL or SLL, providing multiple layers of defense [55]. A variant of SAT attack, referred to as AppSAT [40], was recently proposed to show that a multi-layered defense comprising a point function and a SAT attack vulnerable logic locking technique can be reduced to a single-layer defense comprising the point function alone (e.g., from SARLock+FLL to SARLock). The Double-DIP attack achieves the same objective using more powerful *2-DIPs*, i.e., DIPs that can eliminate at least two incorrect keys in a single iteration [41]. The work in [40, 41] elucidates that simple integration of multiple techniques together fails to combine the resilience offered by individual techniques.

Limitations of existing work: Protection unit that can be isolated and removed. Despite their SAT attack resilience, both SARLock [55] and Anti-SAT [52] exhibit security vulnerabilities, as they leave the original circuit implementation, the IP-to-be-protected, as is. SARLock is vulnerable to *removal attack*: Given a protected/locked netlist, an attacker can identify the comparator/mask blocks and the *flip* signal that directly feeds the output by tracing the transitive-fanout of key-inputs, and remove these blocks, retrieving the original circuit (proprietary IP). Anti-SAT is vulnerable to *signal probability skew (SPS) attack* [57]: Given a protected netlist, an attacker can identify the *flip signal*: it is at the output of the gate whose inputs exhibit the maximum bias towards opposite values. The attacker can then retrieve the original design by re-synthesizing the locked netlist with a constraint **value** 0 (1) on the *flip* signal. Even upon additional obfuscation using additional XOR/XNOR and multiplexer key gates [52], the Anti-SAT block can be isolated using the AppSAT guided removal (AGR) attack [58]. In addition, both SARLock and Anti-SAT are also vulnerable to the *Bypass* attack [53]. The Bypass attack finds a DIP that causes an incorrect output for a wrong key and bypass circuitry is added around the Anti-SAT/SARLock block to fix the output for this DIP. This fix recovers the original design as for both SARLock and Anti-SAT the incorrect key-driven design fails for only one input pattern.

SARLock is re-architected into TTLock in [61] to gain resilience against removal attacks. TTLock makes changes to the original

design to corrupt the output in the absence of the secret key. As SARLock is based on a one-point function, its re-architected version TTLock ends up protecting one input pattern, that is, the modified netlist and the original netlist differ in their outputs for one input pattern only. The work in [61] describes this SAT and removal attack resilient architecture but provides neither a CAD framework to effect the design changes nor a formal analysis proving resilience against various attacks. Furthermore, protection of a single input pattern leads to a rigid scheme where the designer lacks the control to hide an arbitrary amount of IP-critical logic in arbitrary parts of his/her design. Protection of a single input pattern, and thus, low (and uncontrollable) corruptibility also leads to the recovery of an approximate netlist through attacks, such as AppSAT [40] and Double-DIP [41], which SARLock is vulnerable to as well.

1.4 Motivation, proposed approach, and challenges

Motivation. Our ultimate goal is to develop a logic locking technique that can withstand all known and anticipated attacks. Resilience to oracle-guided attacks, such as SAT attack, should be delivered while at the same time hardware implementation of the design should not reveal all the details of the design IP to reverse-engineers (i.e., removal attacks). A logic locking framework should enable the designer to hide the security-critical parts of the design IP, and thereby, to customize logic locking to the security needs of the application.

While hiding any part of the design IP from its hardware implementation may be sufficient to render general applications resilient to reverse engineers (removal attacks), there are applications where a designer may want to specify the parts to hide. Examples include processors with to-be-protected address spaces, for which access is granted only to restricted entities [8]; network-on-chip (NoC) routers where certain IP address ranges may carry particular semantics [12]; intrusion detection systems that rely on pattern matching [24]; and digital signal processing applications, such as comb filters [10], which accentuate/attenuate frequencies at regular intervals.

Approach. Building on the architecture abstracted in [61], the proposed technique *strips* part of the design functionality from its hardware implementation. The design implemented in hardware is therefore no longer the same as the original design, as the former will be missing the stripped functionality. We refer to the proposed technique that can arbitrarily specify this stripped functionality as *Stripped-Functionality Logic Locking (SFLL)*. The hardware implementation can be conceived to have an intentionally and controllable *built-in error*. This error is canceled by a *restore unit* only upon the application of the secret key of logic locking.

The stripped functionality can be captured efficiently in terms of input cubes[1] for which the hardware-implemented design and the original one produce different outputs. We refer to these input cubes as *protected cubes*. They can be stored in bits rather than hardcoded in logic gates. SARLock [55] and Anti-SAT [52] protect zero cubes, as they implement the design IP as is in hardware. Protected cubes can also be conceived as conditions to manifest the built-in error; a reverse-engineer applying the removal attack will obtain a netlist with this error with respect to the original design.

For general applications that require hiding any part of the functionality, it may be sufficient to protect an arbitrary set of cubes. For applications that are specific about the part of the functionality to hide, the proposed SFLL framework should enable the designer to strip functionality based on IP-critical cubes that he/she can specify and provide as input to the framework.

Challenges. The proposed approach necessitates that we discover and explore the connection between resilience to oracle-guided attacks and removal attacks. This translates to another challenge–quantifying the resilience of logic locking to attacks in terms of properties (e.g., the number and size) of cubes protected by them. Only then, we can implement a logic locking framework that can enable the designer to make an informed decision on how to trade resilience to one attack for resilience to another, and identify the sweet spot to protect against all attacks.

1.5 Contributions

- We discover, quantify, and explore the trade-off between resilience of logic locking to different attacks.
- We develop a provably and quantifiably secure and scalable logic locking technique SFLL that thwarts all known and anticipated attacks. We present two versions of SFLL.
  (1) The first version *SFLL-HD*^{h}, a generalized version of TTLock [61], is suitable for general application where stripping an arbitrary part of the functionality is sufficient for protection. This implementation is simple and scalable but is capable of protecting a restricted set of input cubes, all dictated by one secret key. SFLL-HD^h protects $\binom{k}{h}$ input cubes that are of Hamming Distance (HD) h from the k-bit secret key. We provide a security analysis and show that k and h dictate the trade-off between oracle-guided attack resilience and removal attack resilience.
  (2) The second version *SFLL-flex*$^{c \times k}$ allows the user to specify c IP-critical input cubes to be protected, each with k specified bits.
     – We quantify the security level and trade-off in terms of the number and the size of these cubes.
     – We develop a two-stage optimization framework that i) compresses the input cubes, and ii) performs security-aware logic synthesis to strip functionality based on compressed cubes while lowering the cost of implementation and adhering to security requirements.
- This is a complete approach from theory to practice. We develop security definitions and metrics; develop a CAD framework for SFLL; and validate SFLL's security, cost-effectiveness, and scalability through both computer simulations on large-sized benchmark circuits (one with >100K gates) and silicon implementation of ARM Cortex-M0 microprocessor in 65nm LPe technology. We *fabricate both the baseline and the SFLL-locked microprocessors* to perform an accurate comparative analysis.

[1]Input cubes refer to partially-specified input patterns; some input bits are set to logic-0's or logic-1's, while other input bits are don't cares (x's). An n-bit input cube with k specified (care) bits contains 2^{n-k} input patterns.

2 PRELIMINARIES

Before delving into further details, we first define the terminologies that are used in the remainder of the paper.

Notation. We define a set as $\mathcal{S}$ and its elements are denoted as $s \in \mathcal{S}$. We write $s \xleftarrow{\$} \mathcal{S}$ to denote s has been sampled uniformly randomly from the set $\mathcal{S}$. We use ckt_{lock}, ckt_{actv} and ckt_{rec} to denote a logic-locked, an activated, and a reconstructed circuit, respectively. For a circuit ckt the set of all possible inputs and outputs are denoted as I and O respectively. We write $\mathcal{A}^{\mathbb{S}}$ to denote a *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$ following an attack strategy $\mathbb{S}$.

Definition 2.1. A combinational circuit ckt is a netlist that implements a Boolean function $F : I \rightarrow O$, where $I = \{0,1\}^n$ and $O = \{0,1\}^m$ with n inputs and m outputs. A logic locking technique $\mathcal{L}$ can be viewed as a triplet of algorithms, (`Gen, Lock, Activate`), where:

(1) `Gen` is a randomized key generation algorithm, $z \xleftarrow{\$} \texttt{Gen}(1^k)$, where k denotes the key-size,
(2) `Lock` is the algorithm to lock a circuit's functionality, $ckt_{lock} \leftarrow \texttt{Lock}_z(ckt)$, and
(3) `Activate` is a deterministic algorithm that activates the locked circuit, $ckt_{actv} \leftarrow \texttt{Activate}_z(ckt_{lock})$ such that $\forall i \in I, ckt_{actv}(i) = F(i)$.

Threat model. Consistent with the previous works, here we assume the attacker has access to an oracle, denoted, $ckt(\cdot)$, which is a copy of a working chip with the secret key loaded onto its memory. The attacker queries the oracle with a set of input patterns and observes the corresponding outputs. Apart from this, the attacker also has the reverse-engineered netlist ckt_{lock}, which is locked using a logic locking technique $\mathcal{L}$. In this work, we assume the attacker also knows the corresponding elements between the original and the locked netlist; that is, he can identify the location of the protection unit. The attack success for an adversary $\mathcal{A}^{\mathbb{S}}$ implies recovering a circuit such that:

$$\forall i \in I, ckt_{rec}(i) = F(i), \quad \mathcal{A}^{\mathbb{S}} : ckt_{lock} \rightarrow ckt_{rec} \tag{1}$$

SAT attack resilience. SAT attack, a representative and effective oracle-guided attack that iteratively prunes the key space, queries the oracle $ckt_{lock}(\cdot)$ with an input pattern d, called a distinguishing input pattern, to eliminate a set of incorrect keys in each iteration. The attack terminates after querying the oracle with a set of DIPs, and outputting a single key z'. The attacker $\mathcal{A}^{\mathbb{SAT}}$ reconstructs a circuit ckt_{rec} where $ckt_{rec} \leftarrow \texttt{Activate}_{z'}(ckt_{lock})$ such that Eq. (1) is satisfied.

Definition 2.2. A logic locking technique $\mathcal{L}$ is called λ-secure against a PPT adversary $\mathcal{A}^{\mathbb{SAT}}$, making a polynomial number of queries $q(\lambda)$ to the oracle, if he/she cannot reconstruct ckt_{rec} with probability greater than $\frac{q(\lambda)}{2^\lambda}$.

A logic locking technique resilient to the SAT attack is also expected to thwart other variant key-space pruning attacks.

Sensitization attack resilience. Sensitization attack, which is another oracle-guided attack, determines individual key bits by generating and applying patterns that sensitize them to the outputs. In [34], two key bits are considered pairwise-secure iff the sensitization of one key bit cannot be done without controlling the other key bit and vice versa. The same paper presents SLL that maximizes key bits that are all pairwise-secure. For example, key bits converging at a dominating gate are all pairwise-secure, if there is no input assignment to block any one of them before they reach the dominating gate.

Definition 2.3. A logic locking technique $\mathcal{L}$ is λ-secure against a sensitization attack iff λ key bits are all pairwise secure.

Removal attack resilience. Removal attack operates on a locked netlist and tries to isolate and remove the protection logic. The attack is a transformation $T : ckt_{lock} \rightarrow ckt_{rec} \mid \forall i \in I, ckt_{rec}(i) = F(i)$, irrespective of the key value. Note that for a removal attack $ckt_{rec}(p) \neq F(p), \forall p \in P$, where P denotes the set of protected patterns.

Definition 2.4. A logic locking technique $\mathcal{L}$ is λ-resilient against a removal attack, where λ denotes the cardinality of the set of protected input patterns P.

3 SFLL-HD

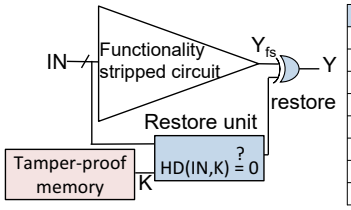
We first present SFLL-HD^h for general applications that will benefit from stripping an arbitrary part of the design functionality. We also show that SFLL-HD^h is a logic locking platform that provides controllable resilience against all known attacks. In SFLL-HD^h, all the protected input cubes are of the same Hamming Distance h from the secret key; though the set of protected cubes are thus restricted, a large number of cubes can be protected through a simple, scalable, and cost-effective hardware.

3.1 SFLL-HD^0

We first explain SFLL-HD^h for the special case of $h = 0$; there is only one protected input cube, and it is the same as the secret key. In other words, SFLL-HD^0 is functionally the same as TTLock [61]. SFLL-HD^0 modifies a design to invert its output for one selected (protected) input pattern; this inversion is the manifestation of the built-in error. The functionality stripping can be effected via logic gate insertions/replacements; the security-aware synthesis module in SFLL-flex in Section 4.2 can also be used to strip functionality based on a set of protected input cubes. SFLL-HD^0 inverts the erroneous output only upon the application of the correct key to the *restore unit*, thereby, cancelling out the built-in error and recovering the correct output. Moreover, SFLL-HD^0 introduces one additional error into the design along with the inverted output for each incorrect key. Here, the secret key consists of the protected input cube selected by the designer.

SFLL-HD^0 has the following properties:

- It protects exactly one input cube.
- Each input pattern can eliminate *one and only one* incorrect key, thereby ensuring the SAT attack requires number of DIPs that is exponential in terms of the key-size.
- Removal attacks, when launched against SFLL-HD^0, only recover the (minimally) modified design that exhibits incorrect (but approximate) functionality.



Y_{fs}	IN	k0	k1	k2	k3	k4	k5	k6	k7
✔	0	✖	✔	✔	✔	✔	✔	✔	✔
✔	1	✔	✖	✔	✔	✔	✔	✔	✔
✔	2	✔	✔	✖	✔	✔	✔	✔	✔
✔	3	✔	✔	✔	✖	✔	✔	✔	✔
✔	4	✔	✔	✔	✔	✖	✔	✔	✔
✔	5	✔	✔	✔	✔	✔	✖	✔	✔
✖	6	✖	✖	✖	✖	✖	✖	✔	✖
✔	7	✔	✔	✔	✔	✔	✔	✔	✖

Figure 4: Proposed SFLL-HD^0 architecture for n = k = 3. The functionality-stripped circuit (FSC) is minimally different from the original function: they produce a different response for one protected input pattern (IN = 6). The restore unit cancels this error for the correct key k6 and introduces a second error for all incorrect keys. Errors are denoted by X's. Note that although SFLL-HD^0 is functionally the same as TTLock, the two are different in hardware architecture; TTLock uses a comparator whereas SFLL uses a Hamming Distance checker. Source: [61].

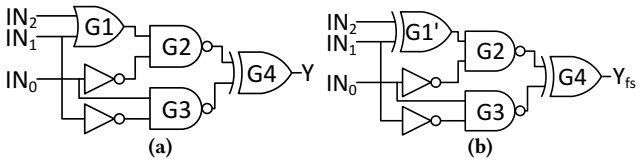


Figure 5: a) Original circuit, b) FSC. Gate G1 is replaced with G1' in the FSC, inverting the output Y_{fs} for IN=6. Source: [61].

3.1.1 Construction of SFLL-HD^0. The architecture of SFLL-HD^h consists of a restore unit and an XOR gate. The restore unit computes the Hamming distance between the key inputs and the primary inputs. In the special case of SFLL-HD^0, the Hamming distance between the primary inputs and the key is zero, implying that the *restore signal* is asserted only when the key inputs and the primary inputs match. Note that for $h = 0$, this restore unit can be reduced to a simple k-bit comparator rendering SFLL-HD^0 functionally equivalent to TTLock, as shown in Fig. 4.

We reuse the example circuit from [61] to illustrate the architecture of SFLL-HD^0 depicted in Fig. 5a. The circuit is protected by a three-bit key, n=k=3; the protected cube is an input pattern, as $n = k$ in this example. The original circuit is shown in Fig. 5a whereas the functionality-stripped circuit (FSC) is shown in Fig. 5b. The original and the functionality-stripped circuits produce a different output for only input pattern 6. Y_{fs} column in Fig. 4 shows the inversion (error) for this protected input pattern. This error is cancelled out by applying the correct key k_6 which asserts the restore signal for input pattern 6, and thus, recovering the desired output, as depicted in the table in Fig. 4. The table also illustrates that each incorrect key induces one extra error in the design, leading to two inversions in each column of the table except the one for the correct key.

3.1.2 Security analysis of SFLL-HD^0. We prove the following theorems which establish the security of SFLL-HD^0 against all known and anticipated attacks. We assume n inputs and k key bits, where $k \leq n$. SFLL-HD^0 delivers the same security properties as TTLock [61]. To establish the security properties of SFLL-HD^0, we develop a formal approach which was missing in [61].

SAT attack resilience. SFLL-HD^0 resilience against SAT attack is achieved by ensuring that the attack encounters its worst-case scenario. In each iteration, a DIP eliminates *exactly one* incorrect key, necessitating a number of iterations that is exponential in the key-size. In the example shown in Fig. 4, the attack requires $7=2^3$-1 iterations in the worst-case. However, if the attacker is fortuitous he/she may hit the protected input cube and eliminate all incorrect keys at once. In the same example, the protected input pattern IN=6 helps the attacker to eliminate all the incorrect keys immediately. However, as an attacker does not have any information about the protected cube, the probability of such a fortuitous hit is exponentially small in the number of key bits.

Theorem 3.1. *SFLL-HD^0 is k-secure against SAT attack.*

Proof. First, we classify all the input cubes into two sets, the set of protected cubes P and set of unprotected cubes $\widehat{P}$. Now, as SFLL-HD^0 only contains one protected input cube, P is a singleton set. Thus, $|P| = 1 \quad and \quad |\widehat{P}| = 2^k - 1$.

Now, an attacker can recover the secret key, and thus, the original functionality of the design if she can find a protected input cube in P. However, for a PPT attacker making only a polynomial number of queries $q(k)$ to the oracle, the probability of finding this cube is

$$\begin{aligned} & \frac{|P|}{2^k} + \frac{|P|}{2^k - 1} \cdots \frac{|P|}{2^k - q(k)} \\ = \quad & \frac{1}{2^k} + \frac{1}{2^k - 1} \cdots \frac{1}{2^k - q(k)} \\ \approx \quad & \frac{q(k)}{2^k} \end{aligned} \tag{2}$$

Note, without loss of generality, we consider the sampling as without replacement as the SAT attack does not repeat any DIP. So, from Definition 2.2, SFLL-HD^0 is k-secure against SAT attack. □

Sensitization attack resilience.

Theorem 3.2. *SFLL-HD^0 is k-secure against a sensitization attack.*

Proof. In SFLL-HD^0, all the k bits of the key converge within the comparator inside the restore unit to produce the *restore* signal. Therefore, sensitizing any key bit through the *restore* signal to the output requires controlling all the other key bits. All k bits are therefore pairwise-secure. SFLL-HD^0 is k-secure against sensitization attack. □

Removal attack resilience. Since the *restore* signal is highly skewed towards *zero*, it can be easily identified by a signal probability skew (SPS) attack. However, any removal attack would recover only the FSC, without leaking any information about the original design. As the FSC produces an erroneous response for the protected input cube, the design is resilient against removal attack.

Theorem 3.3. *SFLL-HD^0 is 2^{n-k}-resilient against removal attack.*

Proof. Suppose the attacker recovers a circuit ckt_{rec} by identifying and removing the restoration logic. Now, ckt_{rec} produces

an incorrect output for the set of protected input cubes, denoted as P. However, we know that each cube contains 2^{n-k} input patterns. Thus, if Γ denotes the set of all input patterns contained in P,

$$ckt_{rec}(i) \neq F(i), \quad \forall i \in \Gamma$$

$$\begin{aligned} |\Gamma| &= |P| \times 2^{n-k} \\ &= 1 \times 2^{n-k} \\ &= 2^{n-k} \end{aligned} \tag{3}$$

So, from Definition 2.4, SFLL-HD^0 is 2^{n-k}-resilient against a removal attack. □

3.2 SFLL-HD^h

In this section, we generalize for h; *SFLL-HD^h* can protect all input cubes that are of Hamming distance of h from the secret key. The number of protected input cubes is $\binom{k}{h}$.

3.2.1 Construction of SFLL-HD^h. With a HD of h, an input-size of n, and key-size of k, SFLL-HD^h inverts the FSC output Y_{fs} for $\binom{k}{h}$ input cubes, which contains $2^{n-k} \cdot \binom{k}{h}$ patterns. The restore unit, which comprises of k XOR gates and an adder to compute the Hamming distance, rectifies all these errors for the correct key, while it introduces a different but possibly overlapping set of errors for any incorrect key. Fig. 6 depicts the architecture of the proposed SFLL-HD^h along with an example where $n = k = 3$ and $h = 1$. As can be seen from the architecture, the implementation overhead of the restore unit is independent of h, which is a hard-coded (non-secret) constant that feeds the comparator inside the restore unit.

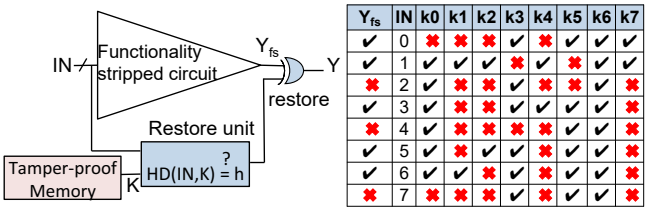
3.2.2 Security analysis of SFLL-HD^h. We assume n inputs and k key bits, $k \leq n$. Proofs of theorems are provided in Appendix B.1.

SAT attack resilience.

Theorem 3.4. *SFLL-HD^h is $(k - \lceil \log_2 \binom{k}{h} \rceil)$-secure against SAT attack.*

Sensitization attack resilience.

Theorem 3.5. *SFLL-HD^h is k-secure against sensitization attack.*



Y_{fs}	IN	k0	k1	k2	k3	k4	k5	k6	k7
✔	0	✖	✖	✖	✔	✖	✔	✔	✔
✔	1	✔	✔	✔	✖	✔	✖	✔	✔
✖	2	✔	✖	✖	✔	✖	✖	✔	✖
✔	3	✔	✖	✖	✔	✔	✔	✔	✖
✖	4	✔	✖	✖	✖	✖	✔	✔	✖
✔	5	✔	✖	✔	✔	✖	✔	✔	✖
✔	6	✔	✔	✖	✔	✖	✔	✔	✖
✖	7	✖	✖	✖	✔	✖	✔	✔	✖

Figure 6: SFLL-HD^h architecture for n=k=3 and h=1. Y_{fs} includes $\binom{k}{h}$ errors, denoted by X's. Restore unit rectifies all errors for the correct key k6. For the incorrect keys, restore unit introduces $\binom{k}{h}$ additional errors (at $\binom{k}{h}$ input patterns), which may possibly coincide and cancel errors in Y_{fs}.

Removal attack resilience.

Theorem 3.6. *SFLL-HD^h is $2^{n-k} \cdot \binom{k}{h}$-resilient against removal attack.*

As these theorems show, h can be adjusted to trade resilience to one attack for resilience to another. Values of h closer to either 0 or k deliver higher resilience to SAT and other key-pruning attacks, whereas resilience to the removal attack can be maximized by setting h=$k/2$.

4 SFLL-FLEX

In contrast to SFLL-HD^h, SFLL-flex$^{c \times k}$ allows the user to specify, and thus, protect the IP-critical input patterns; the restore unit stores the protected input patterns in a compact form, i.e., in the form of c input cubes, each with k specified bits. In this context, the input cubes can be conceived as the secret keys to be loaded onto the chip for the restore unit to recover the stripped functionality. We will use the terms "protected input cubes" and "secret keys" interchangeably for SFLL-flex$^{c \times k}$. The SFLL-flex$^{c \times k}$ framework is shown in Fig. 7; in this section, we elaborate on the individual processes in this framework.

In a design with multiple outputs, not every output needs protection; only the IP-critical part of the design has to be protected to control the cost of logic locking, which is at the discretion of the designer. SFLL-flex$^{c \times k}$ enables the outputs to be selectively flipped (and restored) for the protected input cubes; a *flip vector* associated with each protected input cube holds information regarding which outputs are to be flipped for the protected input cube.

Example. Fig. 8 presents an overview of SFLL-flex$^{c \times k}$. The FSC differs from the original circuit for two protected input cubes $x01x1$ and $x10x1$, collectively representing 8 input patterns. The restore unit stores the two input cubes and the corresponding flip vectors. In this example, only three out of five outputs are protected.

4.1 Architecture

The restore unit of SFLL-flex$^{c \times k}$ consists of a tamper-proof [50] look-up table (LUT) and XOR gates. The LUT stores c k-bit input cubes along with the corresponding f-bit flip vectors (for protecting f out of m outputs) that dictate the functionality stripped from the circuit.[2] When the input matches an entry in the LUT, the associated flip vector is retrieved from the table and XORed with the outputs to restore the original functionality.

Cost. The cost of SFLL-flex$^{c \times k}$ is proportional to the size of the LUT, in addition to f XOR gates inserted at the outputs of the FSC. The cost of the LUT is denoted as $c \times (k + f)$, where f is a designer-defined parameter. Cost minimization requires the minimization of c and k. Additionally, functionality stripping can be used as an opportunity to reduce implementation cost as will be discussed in Section 4.3. Thus, the net cost of SFLL-flex$^{c \times k}$ is this saving subtracted from the LUT cost.

[2] One extreme case of SFLL-flex$^{c \times k}$ is to strip the entire functionality of the design; such an approach would incur a prohibitive overhead, as the entire truth table, whose size is exponential in the number of inputs, needs to be stored on-chip.

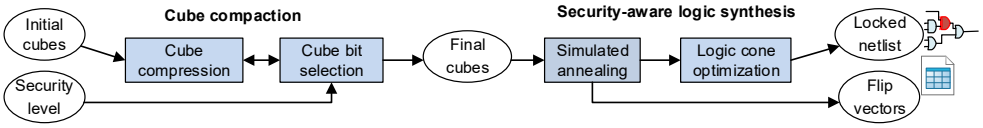


Figure 7: Proposed SFLL-flex$^{c\times k}$ logic locking. The user-specified *input cubes* are compressed to reduce the on-chip storage requirements. Security-aware logic synthesis is then used to strip functionality from the circuit based on compressed cubes.

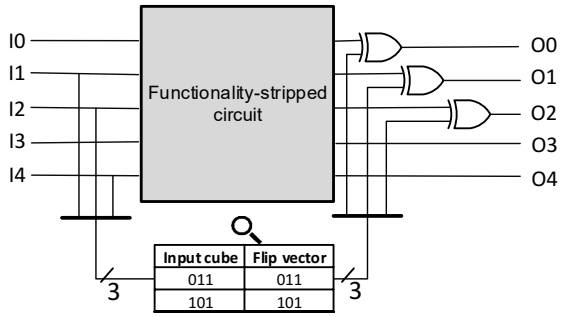


Figure 8: SFLL-flex$^{2\times3}$ for a circuit with five inputs and five outputs. k=3 in the example as only three bits of the protected input cubes are specified by the designer. f=3 as three outputs are protected. The circuit is re-synthesized to strip the functionality from the original design based on the input cubes; these cubes are used along with the flip vectors to restore the original functionality.

4.2 Challenges

As already pointed out, the applicability of SFLL-flex depends on addressing the following research problems:

- How can a minimal set of protected input cubes be produced from those provided by the designer to reduce the storage cost while ensuring the desired security level?
- Can the functionality strip be used as an opportunity to reduce the implementation cost?

We address these problems in the following section.

4.3 Optimization framework for SFLL-flex$^{c\times k}$

Given a desired security level s and a set of input cubes (or input patterns) C_{init} to be protected, both provided by the designer for a netlist N, the proposed stripped-functionality logic locking should be implemented at minimal cost: we should minimize $Cost_{sf} + c \times k$, where $Cost_{sf}$ is the implementation cost of the functionality-stripped netlist N_{sf}, and $c \times k$ is the implementation cost of the LUT. This is an optimization problem that can be formulated as:

$$\text{minimize} \quad Cost_{sf} + c \times k \quad \text{such that} \quad k - log_2 c \geq s$$

where $k - log_2 c$ is the security level attained against SAT attacks, as proved later in this section.

We break this optimization problem down to two smaller processes. In the first process, we compress the input cubes (or input patterns) to minimize the LUT cost = $c \times k$, producing the resulting keys in the process, while honoring the security constraint. In the second process, we re-synthesize the logic of the protected outputs based on the keys obtained from the first process with the goal of minimizing $Cost_{sf}$. Such a sequential approach where the output of the first process is fed into the second process may fail to deliver the overall optimum solution, which rather necessitates a holistic and intertwined approach. In this work, however, we choose to follow a computationally-tractable approach for simplicity at the expense of sub-optimality.

4.3.1 Process 1: Cube compression. In this process, our objective is to reduce the LUT cost $c \times k$, the major component of overall implementation cost, thus, reducing our optimization objective to:

$$\text{minimize} \quad c \times k \quad \text{such that} \quad k - log_2 c \geq s$$

There are a couple of strategies that can be followed to solve this optimization problem. In one strategy, we can create the keys that will flip at least one output for *every* pattern in every cube in C_{init}. The problem then is finding minimum cubes that collectively *cover* each cube in C_{init}; this is the classical problem of minimum-cube cover in 2-level logic optimization [30], and any synthesis tool can be utilized to solve this problem.

In another strategy, we can create the keys that will flip at least one output for *at least one* input pattern in every cube in C_{init}. In this case, the problem is to find minimum cubes that, this time, collectively intersect each cube in C_{init}. To solve this problem, we provide a heuristic approach, as described in Algorithm 1. The first step of the algorithm is cube compression wherein *compatible* cubes are merged to reduce c. To achieve the required security level $s = k - \log_2 c$, we may not need to consider all the k bits in a cube, reducing k. The second step of the algorithm is to eliminate (or turn into x's) the bits that are conflicting among the cubes, while adhering to security level s. This second step may further reduce c, as certain cubes become compatible for merging.

Example. Consider c17 ISCAS benchmark circuit shown in Fig. 9, a set of four 5-bit initial cubes, and security level $s = 3$, as specified by the designer. The two initial cubes $0x100$ and $x1x00$ can be merged into one cube 01100, reducing c to three. Next, we can reduce k to four by eliminating the rightmost bit in all the cubes. Elimination of bits in conflict also leads to further reduction in c to two, as more cubes can now be merged; the achieved security level $s = 3$. Thus, compared to initial $4 \times 5 = 20$ bits, only $2 \times 4 = 8$ bits need to be stored on-chip.

4.3.2 Process 2: Security-aware synthesis. If the designer explicitly specifies which output is flipped for each cube, then the flip vectors are already determined. Such a rigid scheme does not offer any opportunity for optimization; the selected output functions are flipped for the corresponding input patterns included in the protected input cubes. Any logic synthesis tool can be used for this purpose. On the other hand, if the designer chooses not to specify

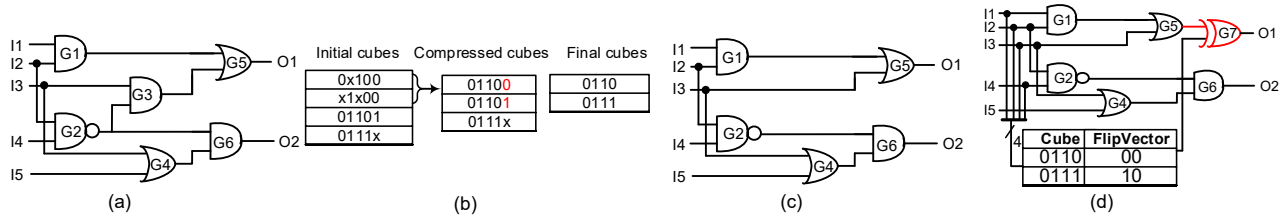


Figure 9: Application of SFLL-flex to c17 ISCAS circuit. a) Original circuit. b) Cube compression. c) FSC. d) Locked circuit.

Algorithm 1: Cube compression algorithm

Input : Initial cubes C_{init}, Security level s
Output: Final cubes C

$C \leftarrow$ merge_compatible_cubes(C_{init})
$s_{new} \leftarrow k - \log_2 c$
while $s_{new} \geq s$ **do**
  $C \leftarrow$ eliminate_conflicting_bit(C)
  $C \leftarrow$ merge_compatible_cubes(C)
  $s_{new} \leftarrow$ update_security_level(c, k)
end

Algorithm 2: Security-aware synthesis algorithm

Input : Original netlist N, Final cubes C
Output: Functionality-stripped netlist N_{sf}, Flip vector V

$V \leftarrow$ init_flip_vector(N)
$N_{sf} \leftarrow$rand_soln(N, C)
$cost_{sf} \leftarrow$ cost(N_{sf})
$T = 1.0, T_{min} = 0.00001, \alpha = 0.9$
while $T > T_{min}$ **do**
  for $i = 1$ *to* 200 **do**
    $N_{new} \leftarrow$neighbor(N_{sf}, C)
    $cost_{new} \leftarrow$ cost(N_{new})
    if $Rand(0,1) < exp(\frac{cost_{new} - cost_{sf}}{T})$ **then**
      $N_{sf} \leftarrow N_{new}$
      $cost_{sf} \leftarrow cost_{new}$
      $V \leftarrow$ update_flip_vector(N_{sf}, po)
    end
  end
  $T = T \times \alpha$
end

the flip vectors, a security-aware synthesis process can leverage this flexibility to minimize implementation cost of the functionality-stripped design N_{sf} without compromising security. The process also produces the flip vectors, denoted by V, as described in Algorithm 2.

Algorithm 2 starts with the original netlist N and a set of cubes C. Initially, a random solution N_{sf} with the associated cost $cost_{sf}$ is generated by initializing the flip vector V with a random value. From this random solution, simulated annealing starts optimization by selecting a neighboring solution at each iteration. A new solution N_{new} is generated by changing a random bit in the flip vector V, which leads to inclusion/exclusion of the corresponding cube for a particular output. The solution N_{new} is accepted if it yields cost savings, i.e. $cost_{new} < cost_{sf}$. An inferior solution may be accepted with a probability of $exp(\frac{cost_{opt} - cost_{new}}{T})$. This is a key feature of simulated annealing for exploring a larger search space without getting stuck at a local optimum.

Example. Let us consider the application of security-aware synthesis to the c17 circuit in Fig. 9. Algorithm 2 operates on the original c17 netlist and the final cubes produced by Algorithm 1, and produces the FSC; AND gate G3 is removed from the logic cone O1. The flip vector 10 will restore the stripped functionality for logic cone O1 by flipping its output for the cube 0110x.

4.4 Security analysis for SFLL-flex$^{c \times k}$

In this section, we discuss the resilience of SFLL-flex$^{c \times k}$ against the state-of-the-art attacks. The proofs of theorems below are provided in Appendix B.2.

SAT attack resilience. An attacker, following a SAT-based or a random guess attack model, must identify *all* input patterns of the protected input cubes in SFLL-flex$^{c \times k}$ to be able to recover the correct functionality of the original design from the on-chip implementation; in contrast to SFLL-HD^h, the protected input cubes can be arbitrary in SFLL-flex$^{c \times k}$, and one cube does not infer another. This requires the retrieval of the content of the entire LUT that represents the stripped functionality. Nevertheless, we assess the security strength of SFLL conservatively; attack success is defined by the attacker's ability to retrieve *any* input pattern that belongs to one of the protected input cubes. The following theorem establishes the resilience of SFLL-flex$^{c \times k}$ against SAT attack.

Theorem 4.1. *SFLL-flex$^{c \times k}$ is $(k - \lceil \log_2 c \rceil)$-secure against SAT attack.*

Sensitization attack resilience.

Theorem 4.2. *SFLL-flex$^{c \times k}$ is k-secure against sensitization attack.*

Removal attack resilience.

Theorem 4.3. *SFLL-flex$^{c \times k}$ is $c \cdot 2^{n-k}$-resilient against removal attack.*

As these theorems show, the number and the size of the protected input cubes, denoted by c and k respectively, dictate the trade-off between resilience to oracle-guided and removal attacks.

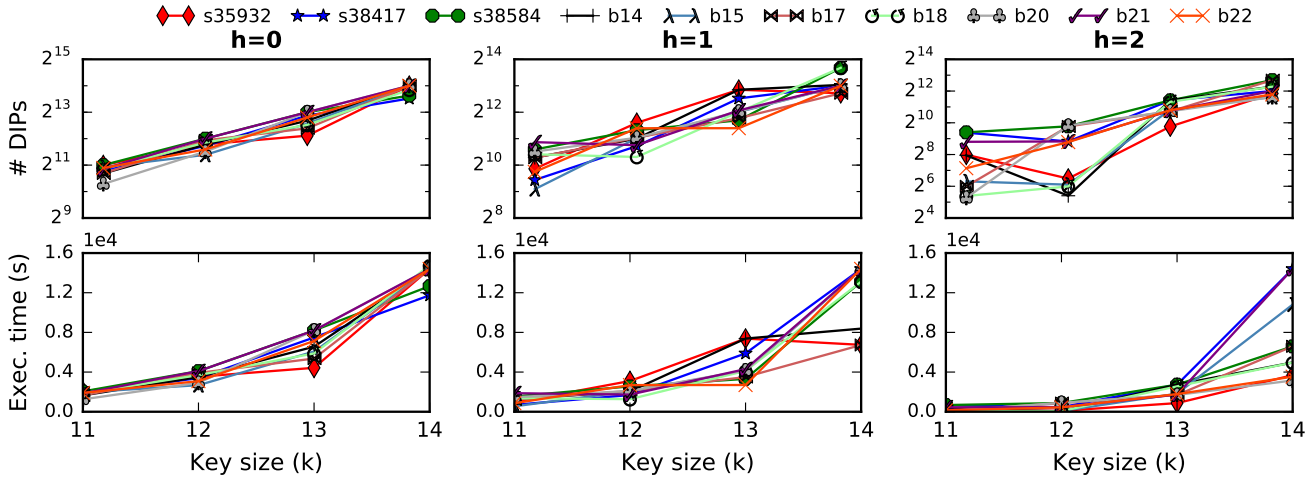


Figure 10: Results for SFLL-HD^h for h={0,1,2}, and k={11,12,13,14}; the number of DIPs required for the SAT attack [44], and the corresponding execution time in seconds.

5 SIMULATION RESULTS

5.1 Experimental setup

In this section, we present the experimental results to validate the security expectations and demonstrate the effectiveness of the proposed SFLL techniques. The experiments are executed on a 28-core Intel Xeon processors running at 2GHz with 128 GB of RAM. We lock the combinational part of the sequential benchmark circuits from the ISCAS'89 [7] and ITC'99 [11] suites in our experiments. Table 3 shows the statistics for the circuits; the largest circuit b18 has >100K gates. The area, power, and delay (APD) overhead for SFLL-HD and SFLL-flex versions are obtained using Synopsys Design Compiler along with Global Foundries 65nm LPe library. We present the results of security analysis wherein different variants of the SAT attack are launched on various versions of SFLL-HD and SFLL-flex. In particular, we launch the SAT attack [44] and the AppSAT [40] against the proposed techniques. Each attack experiment is repeated ten times to improve the statistical significance; average results of the ten runs are reported.

5.2 SFLL-HD^h

5.2.1 Security analysis. The resilience of SFLL-HD^h is dictated by the key-size k and h, which together dictate the number of protected input cubes $\binom{k}{h}$. In SFLL-HD experiments, we protect the largest logic cone in each circuit. The number of DIPs required for the SAT attack to succeed on SFLL-HD^h circuits, and the corresponding execution time are presented in Fig. 10 for k={11, 12, 13, 14} and h={0, 1, 2}. Although the actual security levels required in a practical setting are much larger (e.g., 64-bit or 128-bit), we cannot empirically assess the security of SFLL for such high values due to computational limitations. For the sake of understanding the trends, we experiment with these small key-sizes.

Table 3: Statistics for the largest ITC'99 [11] and ISCAS'89 [7] benchmarks. LLC denotes the largest logic cone.

	Benchmark	Functionality	Inputs	Outputs	Gate count	LLC inputs
Small	s35932	N/A	1763	2048	12,204	195
	s38417	N/A	1664	1742	8709	99
	s38584	N/A	1464	1731	11448	147
	b14	Viper processor	277	299	9,767	218
	b15	80386 processor	485	519	8,367	306
Large	b17	3× b15	1452	1512	30,777	308
	b18	2× b14 + 2× b17	3357	3343	111,241	271
	b20	2× modified b14	522	512	19,682	282
	b21	2× b14	522	512	20,027	282
	b22	3× modified b14	767	757	29,162	283

Impact of key size k. Fig. 10 demonstrates that the number of DIPs required for the SAT attack to succeed grows exponentially in k, confirming our theoretical expectation. For instance, the expected number of DIPs required to break SFLL-HD^0 is 2^{k-1}. The same trend holds for SFLL-HD^1 and SFLL-HD^2 as well, except for a few cases, where an attacker may be fortuitous and the attack terminates earlier, reducing the average number of DIPs.

The execution time of the SAT attack is proportional to the number of DIPs, although there is a slight variation of 3× to 4× across the benchmark circuits; the execution time grows exponentially in k.

Impact of Hamming distance h. SFLL-HD^h is $(k - \lceil \log_2 \binom{k}{h} \rceil)$-secure. Thus, an increase in h leads to a significant change in the security level and the expected number of DIPs required for the SAT attack. For example, the average number of DIPs for the circuit s38584 for h={0, 1, 2} and k=14 is 15K, 10K, and 5K, respectively, as shown in Fig. 10.

5.2.2 APD overhead. The APD overhead is obtained using Synopsys DC Compiler using Global Foundries 65nm LPe library [43] and is shown in Fig. 11 for k=128. The overhead for SFLL-HD can be attributed to two sources: the restore unit and the functionality-stripped circuit. SFLL-HD^h restore unit comprises a single k-bit comparator along with an adder unit, where the overhead is anticipated to increase linearly in k but to remain constant for h, which is a hard-coded constant (as it need not be a secret). The 128-bit comparator and adder blocks incur a significant area, power, and delay overhead on small-sized circuits; for the smallest five benchmarks (˜10K gates), area, power, and delay overhead are 28%, 50%, -2%, respectively. For larger-sized circuits, however, the overhead of the restore unit is amortized; for the largest five benchmarks, the

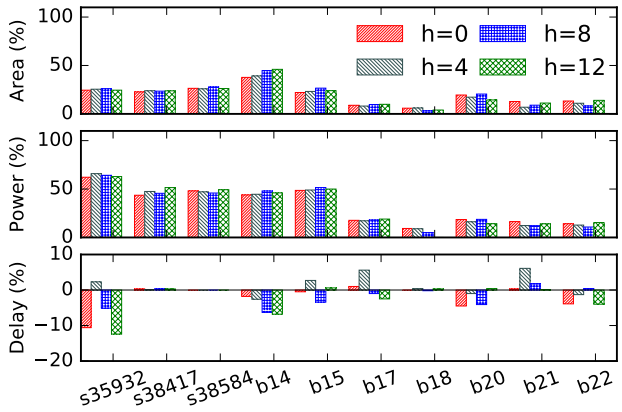


Figure 11: Area, power, and delay overhead for SFLL-HD^{h} for k=128 and h=$\{0,4,8,12\}$. The APD overhead for the larger five benchmarks are 10%, 6%, and -5%, respectively; for the smaller five benchmarks, they are 28%, 50%, and -2%, respectively.

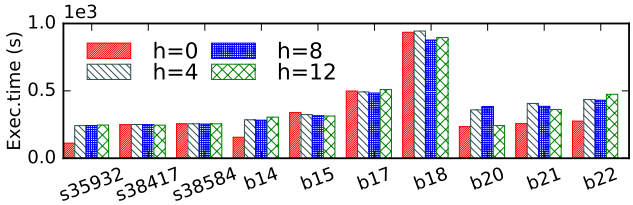


Figure 12: Execution time of SFLL-HD^{h} for k=128.

average area, power and delay overhead are only 10%, 6%, and -5%, respectively, boding well for even larger-sized industrial circuits.

5.2.3 Scalability. SFLL-HD^{h} algorithm operates on the RT-level circuit. Fig. 12 shows that the execution time of the SFLL-HD algorithm is only a few minutes, irrespective of h. For b18 circuit with more than 100K gates, the execution time is only about 15 minutes, *confirming the scalability* of the proposed SFLL-HD^{h}.

5.3 SFLL-flex$^{c\times k}$

5.3.1 Security analysis. To validate the security of SFLL-flex, we launch the SAT attack [44] and AppSAT [40] attack on circuits locked using SFLL-flex for $c = \{1, 2, 3\}$ and $k = \{11, 12, 13, 14\}$. The results shown in Fig. 13 demonstrate that the number of DIPs for SFLL-flex is exponential in k. With increasing c, we observe a logarithmic decrease in the number of DIPs, in line with our theoretical expectation. The trends for the execution time is similar to that for DIPs, except that the increase in execution time is more prominent. While the DIPs double for each increment in k, the execution time may increase by 3-5×. The AppSAT [40] attack on SFLL-flex again fails in 100% of the cases.

5.3.2 Cube compression. The savings for the cube compression technique are presented in Table 4. In our experiments, we generate test cubes for randomly selected c_{init} stuck-at faults by using Atalanta test pattern generation tool [25], and treat these test cubes

Table 4: Cube compression ratio R for SFLL-flex$^{c\times k}$.

Bench	s=64		s=128	
	c = 32	c = 64	c = 32	c = 64
s35932	867.9	1735.9	437.3	874.7
s38417	403.4	806.8	136.5	409.6
s38584	354.9	1441.5	180.2	360.4
b14	26.5	52.9	6.7	14.9
b15	238.8	115.8	120.3	79.6
b17	352.0	469.3	59.1	70.4
b18	813.8	3305.4	832.7	234.3
b20	126.5	61.4	31.9	42.5
b21	49.9	99.7	31.9	36.4
b22	91.6	183.2	62.9	74.9
Average	332.5	827.2	190.0	219.8

as the designer-provided input cubes C_{init}. The compression ratio R is computed as the ratio of the initial number of key bits to be stored $c_{init} \times k_{init}$ to that of compressed key bits $c_{final} \times k_{final}$; k_{init} equals the number of inputs n. The results are presented for two different security levels s = 64 and 128 and for two different numbers of initial cubes c = 32 and 64. On average, a compression level of 400× is achieved, while still maintaining the desired security level. These compression levels directly translate to a reduction in implementation cost for the restore unit. It can be noted that a lower security level (s = 64) enables a higher compression level.

5.3.3 Security-aware synthesis. We report the area, power, and delay (APD) overhead separately for 1) the "optimal-cost" FSC (without the restore unit) and 2) the overall circuit (with the restore unit comprising the LUT and the surrounding combinational logic). The APD overhead is shown in Fig. 14a and Fig. 14b for target security levels s = 64 bits and 128, respectively. The simulated-annealing based optimization is accomplished using area as the primary cost metric. The ABC [6] synthesis tool is used to convert a design to And-Invert-Graph and the gate count is taken as the cost metric. It can be inferred that security-aware synthesis incurs only a minimal *overall* overhead of 5%, 4% and 2% for area, power, and delay for a security level s = 64, and 11%, 8% and -1% for a security level s = 128. In these figures, negative values denote a reduction in APD when compared to the original circuit due to the functionality-strip operation; e.g., this can be seen for the circuit s35932 in its area footprint. However, due to the overhead of the restore unit comprising mostly sequential elements, the overall overhead is positive. In majority of the cases, the delay overhead is almost negligible (˜0%). This is due to the fact that adding the restore unit does not actually affect the delay of the critical path, thus, incurring no significant performance penalty.

The combined execution time for cube compression and security-aware synthesis is presented in Fig. 15. The execution time for cube compression is in the order of a few seconds. The execution time for security-aware synthesis is directly determined by the simulated annealing parameters (the temperature T and the gradient α) and the size (number of gates) of a circuit. As can be seen from the figure, even for large circuits such as b18 with >100K gates, the synthesis is completed in about two hours. Our empirical results indicate that the execution time remains independent of the security level s

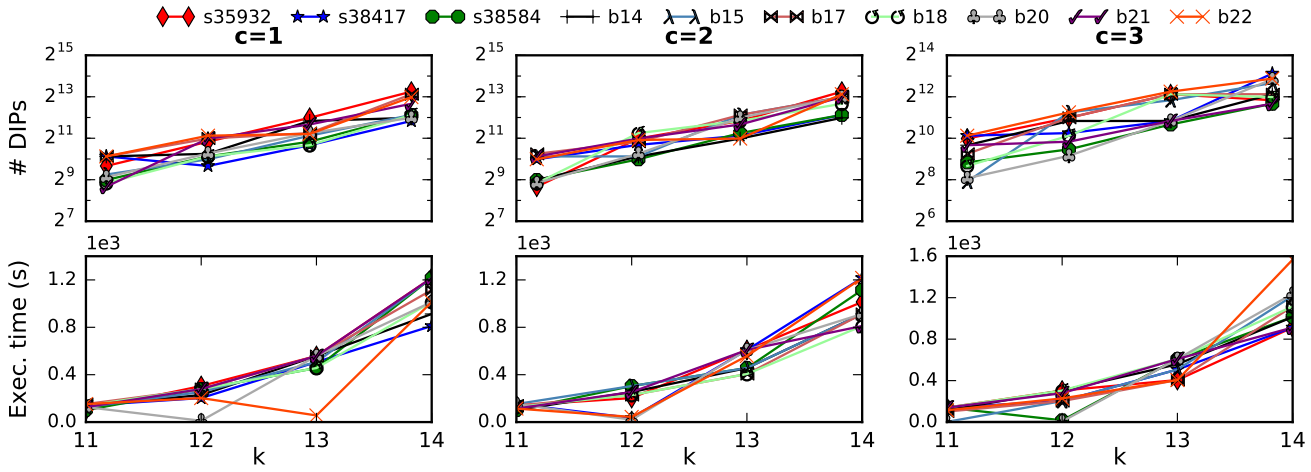


Figure 13: Results for SFLL-flex$^{c\times k}$ for c={1,2,3} and k={11,12,13,14}; the number of DIPs required for the SAT attack [44] and the corresponding execution time (seconds).

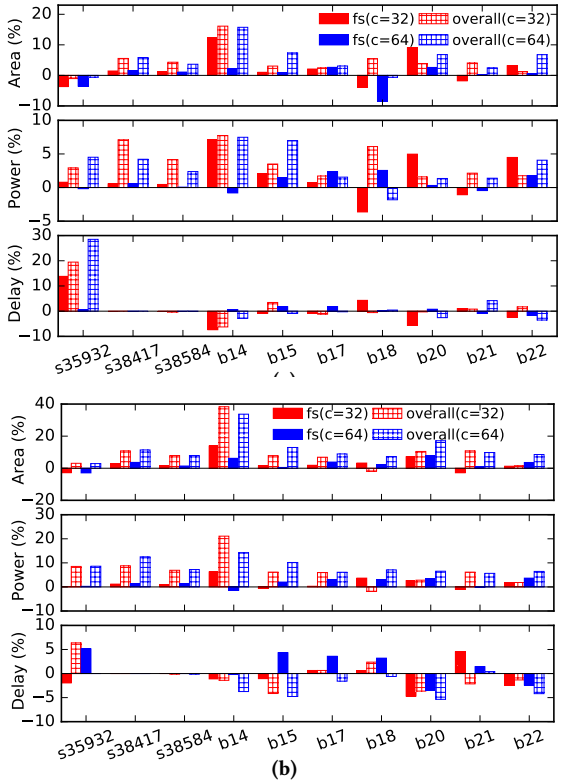


Figure 14: Area, power, and delay overhead for SFLL-flex$^{c\times k}$ for number of initial cubes c=32 and c=64 and for security level (a) s=64 and (b) s=128. The overhead of the functionality stripped circuit, denoted as fs, is shown with dark shade and overhead of the overall circuit including restore unit is shown with light shade. The average overhead for the largest five benchmarks are 6%, 4%, and -1.5% for area, power, and delay, respectively; for the smallest five benchmarks, the numbers are 10%, 8%, and 1.5%, respectively.

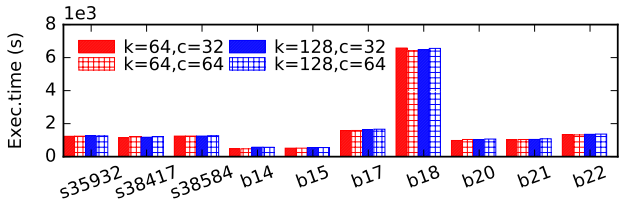


Figure 15: Combined execution time of cube compression and security-aware synthesis for SFLL-flex$^{c\times k}$.

and the number of protected cubes k, *confirming the scalability* of the proposed SFLL-flex$^{c\times k}$.

5.4 Double-DIP/AppSAT attack results

While the SAT attack terminates only upon retrieving the correct key, the AppSAT [40] and Double-DIP [41] attacks may (*counter-intuitively*) terminate earlier returning an incorrect key value, which results in an approximate netlist [40]. The termination criteria for AppsAT is dictated by an error rate specified by the attacker, whereas, Double-DIP terminates when it can no longer find DIPs that eliminate at least two incorrect keys.

Double-DIP. Each of the 2-DIPs employed by the Double-DIP attack can eliminate at least two incorrect keys. Since no such 2-DIPs exist for SFLL-HD^0 and SFLL-flex$^{1\times k}$, the attack will terminate immediately, recovering an approximate netlist. For larger h and c values, each input pattern is a 2-DIP, leading to scalability issues for the Double-DIP attack. As illustrated in Fig. 16, the attack then behaves similarly to the SAT attack, except that the execution time of the two attacks may vary depending on the DIPs employed by the two attacks.

AppSAT. In our first set of AppSAT experiments, we used the default AppSAT parameters as reported in [40], i.e., 50 random queries to the oracle were employed at every 12^{th} iteration of the attack. We observed that estimating the error rate using such a small number of patterns can be misleading and result in premature termination of the AppSAT attack, even for circuit with

Table 5: AppSAT [40] attack results (with default AppSAT setting) against SFLL-HD^h and SFLL-flex$^{c\times k}$. Only 50 random queries are applied as per the default AppSAT settings [40]. The attack fails to retrieve the correct key, and thus, we report it as failure.

Benchmark	s35932	s38584	s38417	b14	b15	b17	b18	b20	b21	b22
Success/failure	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail

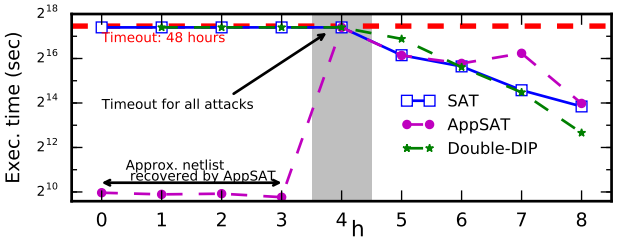


Figure 16: Execution time of the SAT, AppSAT, and Double-DIP attack on (k=) 32-bit s38417 circuit plotted as a function of h. 1000 random queries are applied after every 12 AppSAT iterations. In the shaded region, all the three attacks time out. Double-DIP is not applicable for h=0.

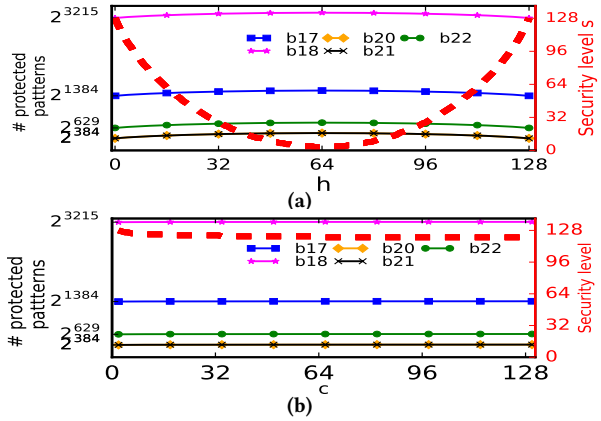


Figure 17: SAT attack resilience vs. removal attack resilience; k=128. (a) SFLL-HD^h and (b) SFLL-flex$^{c\times k}$.

high corruptibility. Table 5 reports that the "default" AppSAT attack terminates erroneously for all of the SFLL circuits, failing to retrieve the correct netlist.

For more realistic corruptibility estimates, we repeated the experiments on s38417 SFLL-HD circuit with 32 key bits. 1000 random queries were applied after every 12 iterations. Fig. 16 shows that for $h \leq 3$, the attack terminated quickly, recovering an approximate netlist. However, for the same h values, the SAT attack failed to complete within the time limit of 48 hours. Moreover, for the larger values of h, representing higher corruptibility, AppSAT behaves exactly like the SAT attack, failing to retrieve an approximate netlist. For example, for $h = 4$ (implying security level of $32-\lceil\log_2 \binom{32}{4}\rceil$=15 bits), both AppSAT and the SAT attack fail to succeed within the time limit of 48 hours. Note that due to the inclusion of the random queries (and additional clauses in the SAT formula), the execution time of AppSAT may be occasionally higher than that of the SAT attack. See Section 7.3 for further discussion.

5.5 Trade-off: resilience to different attacks

Fig. 17 illustrates the wide spectrum of solutions offered by the proposed SFLL-HD and SFLL-flex techniques; it shows the trade-off between the removal attack resilience (in terms of the number of protected input patterns) and the security level s against oracle-guide (e.g., SAT) attacks for the largest five benchmark circuits. We observe that for SFLL-HD^h, the security-level s attained against SAT attacks varies polynomially with h ($h \in [0, k]$); the larger the number of protected patterns, the lower the security level. The security level depends only on k and h, irrespective of the circuit. For the maximum number of protected patterns, i.e., h=$n/2$, the security level s is minimal. The security level is at its maximum at $h = 0$ or $h = k$.

For SFLL-flex$^{c\times k}$, however, s decreases only logarithmically with c ($s = k - \lceil\log_2 c\rceil$). As an example, for $c = 128$ cubes, the security level s attained is 121, irrespective of the circuit. The number of protected patterns increases linearly with c. For example, for the circuit b20, the number of protected patterns increases from 2^{384} for c=1 to 2^{391} for c=128.

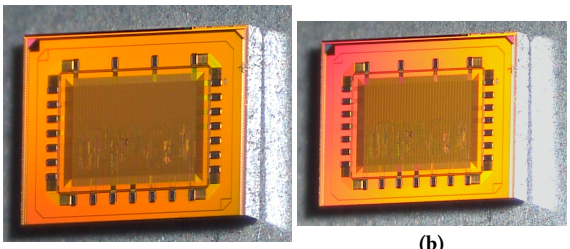
Both variants of SFLL enable the protection of a large number of input patterns. While SFLL-HD allows the designer to choose only the secret key value and the Hamming distance h, SFLL-flex allows him/her to specify the complete set of input cubes to be protected.

6 CASE STUDY: SILICON IMPLEMENTATION OF SFLL-HD^0 ON ARM CORTEX-M0 PROCESSOR

With the objective of deploying SFLL for IoT applications, we present the details of silicon implementation of SFLL on an *in-house* designed microcontroller using ARM Cortex-M0 microprocessor [2]. For accurate comparisons, we fabricated both the baseline and the SFLL-locked microcontroller. Cortex-M0 belongs to the family of Cortex-M 32-bit RISC processor series from ARM, suitable for a variety of low-cost microcontrollers. The microcontroller includes ARM AHB-Lite as its BUS, UART interface, and 64 KB of SRAM.

6.1 Logic locking on ARM Cortex-M0

The baseline ARM Cortex-M0 is locked using 128-bit SFLL-HD^0 along with 128-bit FLL [36]. FLL is a technique to achieve high output corruptibility, while SFLL ensures security against any SAT-based attack. Due to tight scheduling constraints, we were able to tape-out only one version with SFLL-HD^0. In our implementation, we chose to lock the program counter (PC) to safeguard against unauthorized execution. This ensures that an attacker with an incorrect key would end up with an incorrect execution due to the corrupted PC. Ideally, the secret key is stored in a tamper-proof memory, such as one-time programmable fuse ROM. However, in our implementation, the 256-bit key for the locked processor is stored in a *write-only* configuration register. The locked processor



(a) (b)

Figure 18: Top-view of the fabricated silicon chips for ARM Cortex-M0. (a) Baseline version and (b) Locked version.

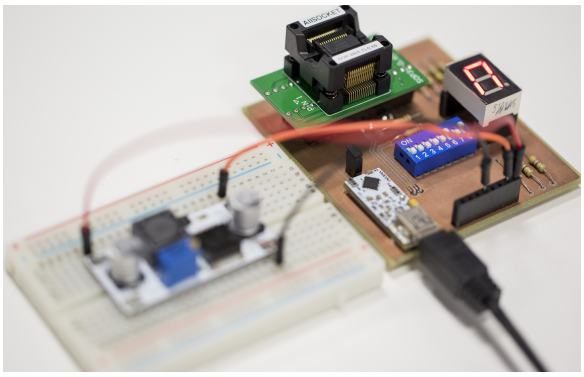


Figure 19: Test set-up for the baseline as well as the locked processor. The program is loaded onto the processor via UART interface.

is activated by loading the secret key onto the configuration register through UART.

Chip design/fabrication flow. We used Synopsys VCS for simulation, Synopsys Design Compiler for RTL synthesis, Synopsis IC Compiler for back-end implementation, Synopsys Prime Time for static timing analysis, Synopsys Formality for logical equivalence checking, PrimeRail for IR drop analysis, and Cadence PVS for physical verification. The baseline and the locked versions with the maximum frequency of 100 MHz have been fabricated using Global Foundries 65*nm* LPe process. A microscopic view of the bare dies for the baseline and the locked versions are shown in Fig. 18a and Fig. 18b, respectively. Fig. 19 illustrates the test setup for the chip.

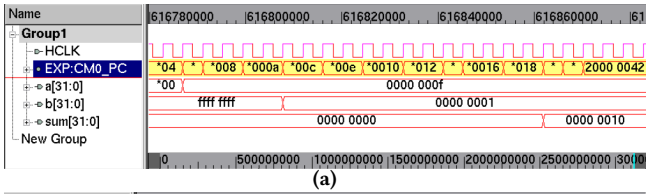
6.2 Implementation overhead

The APD overhead along with other parameters for the baseline and locked processors are shown in Table 6. The proposed 128-bit FLL+128-bit SFLL-HD^0 incurs a minimal overhead of 2.16%, 5.62%, and 5.38% for area, power and delay, respectively, when compared to the baseline design.

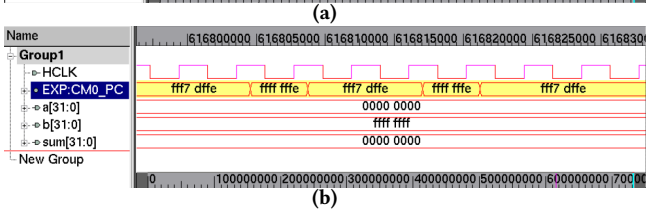
A brief look at other implementation parameters, such as RAM size, combinational/sequential area or wirelength demonstrates that the two versions of the processor are quite similar. The most significant difference is in the combinational area, which is about 15.2%. This increase in area for the locked processor can be attributed to the key gates introduced by FLL, and the restore unit introduced by SFLL. The additional routing resources required for the additional logic translate into a wirelength overhead of 7.6%

Table 6: Baseline ARM Cortex-M0 vs. locked ARM Cortex-M0 (128-bit FLL + 128-bit SFLL-HD^0).

	Baseline	Locked	Overhead (%)
Gate count	46800	51397	9.82
RAM area (μm^2)	349240	349240	0
Combinational area (μm^2)	61404	70765	15.24
Sequential area (μm^2)	36876	37169	0.79
IO pads (μm^2)	150000	150000	0
Wirelength (μm)	985233	1060502	7.64
Overall area (μm^2)	597521	607175	**1.62**
Power (μW)	6.66	7.03	**5.62**
Delay (ns)	8.00	8.43	**5.38**



(a)



(b)

Figure 20: Execution of the SFLL-locked ARM Cortex-M0 with the (a) correct key and (b) incorrect key. The content of the program counter EXP:CM0_PC is highlighted to show the difference in execution.

6.3 Security analysis

The locked processor protects against all oracle-guided attacks. The sensitization attack [34] terminates in a few minutes but without returning the correct key. When the SAT attack [44] is launched on the locked processor, the attack does not terminate within the specified time limit of 48 hours. Since we implement compound logic locking (SFLL+FLL) on the processor, the AppSAT attack [40] would be able to reduce the compound logic locking problem to SFLL alone; indeed the AppSAT attack on the locked processor terminates after 46 hours, but *fails to identify the SFLL key.*

6.4 Operation of the locked processor

We use the simple example code below that performs one addition operation to explain the impact of logic locking (i.e., hardware-level protection) on processor operations (i.e., software execution).

```
int a,b;
GPCFG->SPARE0=0x0000000F;
GPCFG->SPARE1=0x00000001;
a=GPCFG->SPARE0;
b=GPCFG->SPARE1;
GPCFG->GPTACFG=a+b;
```

This C code is compiled for the ARM Cortex-M0 using ARM IAR Embedded Workbench and the corresponding binary images are loaded onto the SRAM via the UART interface. The activated processor (that has the secret key loaded on the chip) executes

the code correctly as shown in Fig. 20a; the addition of 0x01 and 0x0F produces 0x10 as expected. On the other hand, the locked processor (with an incorrect key loaded) cannot execute the code correctly, as shown in Fig. 20b, as the program counter is corrupted. An exception handler is then called, resetting the PC to the default value of 0xFFF7_FFFE, causing the execution to go into an infinite loop.

7 DISCUSSION

7.1 Comparative security analysis

Table 7 presents a comparison of SFLL-HD and SFLL-flex with other logic locking techniques. Existing SAT attack resilient techniques such as SARLock and Anti-SAT are vulnerable to removal attacks. The proposed SFLL thwarts all known attacks on logic locking. Further, it allows a designer to cost-effectively explore the trade-off between resilience to SAT and removal attacks.

7.2 Choosing between SFLL-HD and SFLL-flex

While SFLL-HD is suitable for generic applications where the main requirement is to protect a large number of input patterns with minimal overhead, SFLL-flex allows a designer to protect specific input cubes. The capability to specify IP-critical cubes to protect, even a small number of them, may be very beneficial for applications such as microprocessors with IP-critical controllers, digital signal processing applications with IP-critical coefficients, etc. The flexibility required in SFLL-flex necessitates a slightly more expensive restore unit mainly due to the LUT, compared to SFLL-HD, which has a generic, simple, and scalable restore unit. In either case, the security-aware synthesis framework enables the designer to attain the desired security level.

7.3 Resilience against the derivative attacks

In this section, we discuss the security of SFLL against the attacks that have been derived from the SAT attack, namely AppSAT [40], Double-DIP [41], and Bypass [53]. These attacks mainly target compound (multi-layered) logic locking techniques. AppSAT and Double-DIP are approximate attacks as they only reduce a compound logic locking technique (e.g. SARLock+SLL) to a SAT attack resilient technique (e.g. SARLock). The Bypass attack, however, is an exact attack; the attack, if successful, returns a netlist functionally equivalent to the oracle (functional IC).

These attacks rely on the classification of compound logic locking key bits into two classes: key bits for RLL/SLL etc. that introduce high corruptibility and key bits for SARLock/Anti-SAT etc. that induce low corruptibility at the outputs. These attacks can quickly determine the correct values for the high corruptibility key bits. The AppSAT and Double-DIP attacks then assign a random value for the low corruptibility key bits, whereas, the Bypass attack introduces additional logic to fix the occasional corruption at the outputs. These attacks will not be effective against SFLL as all the key bits in SFLL incur uniform corruptibility and it is not feasible to partition the key search space into low/high corruptibility regions.

AppSAT. There are two main differences between AppSAT and the SAT attack. First, AppSAT is constructed by augmenting the SAT attack with random queries to the oracle at regular intervals. As reported in [40], AppSAT involves 50 random queries every 12 iterations of the attack. Second, AppSAT can terminate much earlier than the SAT attack, i.e., when the error rate (or Hamming distance at the outputs) is below a certain threshold (e.g., $\frac{1}{2^k}$). While the AppSAT attack can quickly recover an approximate netlist for low-corruptibility SFLL circuits (with low h or c), it behaves similarly to the SAT attack for high-corruptibility SFLL circuits since the early termination condition is not satisfied. Thus, SFLL resilience against AppSAT is similar to that against the SAT attack.

The 50 queries as per the default AppSAT settings are sufficient to separate the key bits into two classes in case of compound locking techniques. However, no such classes of key bits exist in SFLL where the corruptibility is uniform for all the key values. As reported in Section 5.2.1, when the attack is launched on SFLL circuits with varying corruptibility values (represented using h), the attack terminated erroneously even for high corruptibility circuits. The error is better estimated with 1000 random queries. The attack then quickly extracts the approximate netlist for the smaller values of h. For the larger h values, the attack performance is similar to that of the SAT attack.

Double-DIP. Compared to the SAT attack, the Double-DIP attack uses a larger miter circuit comprising four copies of the locked netlist [41]. The 2-DIPs computed by the attack eliminate at least two incorrect keys per DIP. The attack terminates when no more 2-DIPs can be found, implying that only DIPs that can eliminate at most one incorrect key remain in the search space. The attack returns an approximate netlist. While the attack can break compound logic locking techniques, it is not scalable, especially if the locked circuit has multiple correct keys.

Except for SFLL-HD^0 or SFLL-Flex$^{1\times k}$(whrere there are no 2-DIPs), the Double-DIP attack, when launched on SFLL circuits, will run into scalability issues as it will compute an exponential number of DIPs before it terminates. Rarely, when the attack is fortuitous and selects one of the protected patterns as a DIP (the protected pattern may be a 2-DIP), it can eliminate most of the incorrect keys in a single iteration of the attack. In such cases, the attack returns returns the exact netlist, similar to the basic SAT attack, but this is highly unlikely for large enough key sizes.

Bypass. The Bypass attack selects two random key values as constraints for the two locked netlists in the miter circuit [53]. The attack then computes all the DIPs that result in differing outputs for the two key values. For the traditional low corruptibility locking techniques such as SARLock, only a few DIPs will be extracted. The attacker then determines the correct key values for those DIPs from the output of functional IC. One of the two key values is designated as the secret key and an additional bypass circuit is added around the locked netlist to fix the output for the selected DIPs.

In SFLL, a protected input pattern produces the same incorrect output for most of the incorrect key values. Occasionally, the output may be correctly restored even for incorrect key values, as illustrated earlier in Figure 4. When applied to SFLL, the Bypass attack fails to compute the complete set of DIPs that lead to incorrect outputs for the two key values. Most of the DIPs yield exactly the same incorrect output for both incorrect keys, and as such, cannot be extracted using the miter construction employed by the Bypass attack. The bypass circuitry, when constructed using an incomplete set of DIPs, will be erroneous.

Table 7: Comparative security analysis of logic locking techniques against existing attacks. SFLL is secure against all attacks. Various versions of SFLL offer a trade-off between SAT attack resilience and removal attack resilience.

Attack/Defense	Anti-SAT [52]	SARLock [55]	TTLock [61]	SFLL-HD^h	SFLL-flex$^{c \times k}$
SAT	k-secure	k-secure	k-secure	$k - \lceil \log_2 \binom{k}{h} \rceil$-secure	$(k - \lceil \log_2 c \rceil)$-secure
Sensitization	k-secure	k-secure	k-secure	k-secure	k-secure
Removal	0-resilient	0-resilient	2^{n-k}-resilient	$\binom{k}{h} \cdot 2^{n-k}$-resilient	$c \cdot 2^{n-k}$-resilient

7.4 Limitations and future work

While SFLL offers resilience against all known and anticipated attacks in addition to offering trade-offs between attack resilience, it might be occasionally broken if the attacker is fortuitous enough to find one of the protected patterns. Currently, a linear increase in the number of protected patterns leads to a logarithmic decrease in SAT attack resilience. As part of our future work, we will refine SFLL algorithm so that the decrease in SAT attack resilience is sub-logarithmic with increasing number of protected patterns.

Our current SFLL implementation relies on the existing security-agnostic tools such as Synopsys DC Compiler and ABC [6] for synthesis. Netlists synthesized using these tools may leave behind traces for an attacker to exploit. In future, we plan to modify the ABC tool and offer security guarantees against synthesis-driven attacks [29].

Currently, SFLL relies on a designer to specify which parts of a design are security-critical. In future, we plan to automate this process by developing metrics that quantify the criticality of different circuit components. Another related research question is to identify the patterns-to-be-protected that incur the least implementation overhead, or can even yield area, power, or delay savings.

8 CONCLUSION

We propose stripped-functionality logic locking: a low-cost, secure, and scalable logic locking technique that provably thwarts all known and anticipated attacks. We quantify the resilience of any logic locking technique against a given attack in terms of the number and the size of the protected input cubes. Based on this finding, we develop a CAD framework that allows the designer to strip functionality from the hardware implementation of the design based on a set of input cubes to be protected; we also propose a security-aware synthesis process that can strip functionality with the objective of minimizing the cost of implementation. By adjusting the number and the size of the protected cubes, the designer can explore the trade-off between resilience to different attacks. The stripped functionality is hidden from untrusted entities, such as the foundry and the end-user (potential reverse-engineer). Only the secret key, i.e., the protected cubes, can successfully recover the stripped functionality through an on-chip restore operation.

Another flexibility that the proposed framework offers is that for general applications, it enables the designer to protect any number of a restricted set of cubes, leading to a simple and scalable architecture. It also supports specialized applications that require IP-critical input cubes to be protected. The designer can thus choose the solution that best fits the security needs of his/her application.

Upon implementing the proposed logic locking technique on large-sized benchmarks (> 100K gates) and launching all known attacks on them, we confirm that the proposed technique is secure and cost-efficient. For further validation, we also apply the proposed logic locking technique on an industry-strength microprocessor design that we then implement in silicon; the data obtained from the fabricated chips also confirm the practicality, security, and scalability of our technique. The proposed technique can be seamlessly integrated into the IC design flow to thwart IP piracy, reverse engineering, and overbuilding attacks.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation Computing and Communication Foundations (NSF/CCF) under Grant 1319841, the National Science Foundation, Division Of Computer and Network Systems (NSF/CNS), under Grant number 1652842; and the New York University/New York University Abu Dhabi (NYU/ NYUAD) Center for Cyber Security (CCS). A part of the research was carried out using Core Technology Platform resources at NYUAD. The authors would also like to thank James Weston and Nikolaos Giakoumidis for their support throughtout the project.

A OPTICAL/SEM IMAGING OF THE CHIPS

A microscopic view of the bare dies of the baseline and locked versions are shown in Fig. 18a and Fig. 18b, respectively, using an optical microscope. The I/O openings for the chips are clearly visible in the figure. We observe identical structures for both the baseline and the locked versions, yet there are minute differences between the two versions though not visible in the figure.

Also, a scanning electron microscope (SEM) image of the locked version is presented in Fig. 21 where a particular area of the chip is milled out to spot part of the protection unit (shown in the inset) used to lock the processor. The chip was sectioned using an FEI Scios focused ion beam (FIB) system. The first milling stage involved removing 5 microns of the top layer over a 100 by 100 micron area using the Galium beam at 30 kV/15 nA. Afterward, 250 nm, or thinner, slices were removed at a lower current 7 nA. Imaging was performed using the electron beam using both secondary and back-scattered electrons (ETD and T1) detectors. This mill-and-image process is our attempt in mimicking the reverse engineering capabilities of an attacker to obtain the netlist of the device. In fact, an attacker would rather use etching to delayer individual metal layers until he/she reaches the substrate layer. In our experiments, we use FIB-SEM to mill until the substrate layer is exposed and the gates used for logic locking are visible.

B PROOFS OF THEOREMS

We assume n inputs and k key bits, where $k \leq n$.

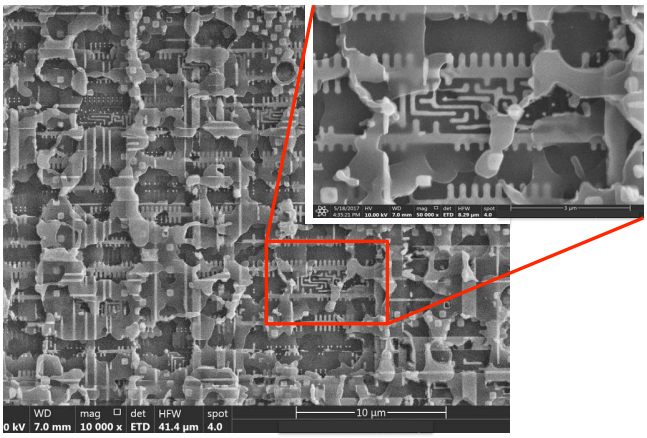


Figure 21: Scanning electron microscope (SEM) image of the milled chip showing the area where part of the protection unit was inserted.

B.1 SFLL-HD^h

Theorem 3.4. *SFLL-HD^h is $(k - \lceil \log_2 \binom{k}{h} \rceil)$-secure against SAT attack.*

Proof. Similar to Theorem 3.1, for SFLL-HD^h, $|P| = \binom{k}{h}$ and $|\widehat{P}| = 2^k - \binom{k}{h}$. Thus, for a PPT attacker oblivious to the protected input cubes, making only only polynomial number of queries $q(k)$, the success probability is given by Eq. 2,

$$\begin{aligned} & \frac{|P|}{2^k} + \frac{|P|}{2^k - 1} \cdots \frac{|P|}{2^k - q(k)} \\ = \; & \frac{\binom{k}{h}}{2^k} + \frac{\binom{k}{h}}{2^k - 1} \cdots \frac{\binom{k}{h}}{2^k - q(k)} \\ \approx \; & \frac{q(k) \cdot \binom{k}{h}}{2^k} \\ < \; & \frac{q(k)}{2^{k - \lceil \log_2 \binom{k}{h} \rceil}} \end{aligned}$$

So, from definition 2.2, we get SFLL-HD^h is $(k - \lceil \log_2 \binom{k}{h} \rceil)$-secure against the SAT attack. □

Theorem 3.5. *SFLL-HD^h is k-secure against sensitization attack.*

Proof. Similar to SFLL-HD^0, all the k bits of SFLL-HD^h converge within the comparator inside the restore unit to produce the *restore* signal. Therefore, sensitizing any key bit through the *restore* signal to the output requires controlling all the other key bits. All k bits are therefore pairwise-secure. SFLL-HD^h is k-secure against sensitization attack. □

Theorem 3.6. *SFLL-HD^h is $2^{n-k} \cdot \binom{k}{h}$-resilient against removal attack.*

Proof. As the *restore* signal is skewed towards 0, it can be identified by a signal probability skew (SPS) attack. The attacker is then able to recover the FSC, denoted as ckt_{rec} which produces erroneous output for the set of protect input patterns Γ. Similar to Theorem 3.3, from Eq. 3 we get,

$$\begin{aligned} |\Gamma| &= |P| \times 2^{n-k} \\ &= \binom{k}{h} \times 2^{n-k} \end{aligned}$$

So, from Definition 2.4, SFLL-HD^h is $2^{n-k} \cdot \binom{k}{h}$-resilient against a removal attack. □

B.2 SFLL-flex$^{c \times k}$

Theorem 4.1. *SFLL-flex$^{c \times k}$ is $(k - \lceil \log_2 c \rceil)$-secure against SAT attack.*

Proof. For SFLL-flex$^{c \times k}$, the cardinality of the set of protected cubes P is $|P| = c$. Thus, from Eq.2, the success probability of a PPT adversary making a polynomial number of queries $q(k)$ is given by

$$\begin{aligned} & \frac{|P|}{2^k} + \frac{|P|}{2^k - 1} \cdots \frac{|P|}{2^k - q(k)} \\ = \; & \frac{c}{2^k} + \frac{c}{2^k - 1} \cdots \frac{c}{2^k - q(k)} \\ \approx \; & \frac{q(k) \cdot c}{2^k} \\ < \; & \frac{q(k)}{2^{k - \lceil \log_2 c \rceil}} \end{aligned}$$

So, from definition 2.2 we get SFLL-HD$^{c \times k}$ is $(k - \lceil \log_2 c \rceil)$-secure against the SAT attack. □

Theorem 4.2. *SFLL-flex$^{c \times k}$ is k-secure against sensitization attack.*

Proof. All the k bits of SFLL-flex$^{c \times k}$ converge within the comparator inside the LUT to produce the signal that asserts the XOR vector operation between the flip vector and the outputs. Therefore, sensitizing any key bit through the LUT to any of the outputs requires controlling all the other key bits. All k bits are therefore pairwise-secure. SFLL-flex$^{c \times k}$ is k-secure against sensitization attack. □

Theorem 4.3. *SFLL-flex$^{c \times k}$ is $c \cdot 2^{n-k}$-resilient against removal attack.*

Proof. Even if the LUT along with its surrounding logic can be identified by a reverse-engineer, he/she can only recover the FSC denoted as ckt_{rec}. However, ckt_{rec} produces incorrect output for the protected input patterns Γ. Thus, similar to Theorem 3.3,

$$ckt_{rec}(i) \neq F(i), \quad \forall i \in \Gamma$$

$$\begin{aligned} |\Gamma| &= |P| \times 2^{n-k} \\ &= c \cdot 2^{n-k} \end{aligned}$$

So, from Definition 2.4, SFLL-HD$^{c \times k}$ is $c \cdot 2^{n-k}$-resilient against a removal attack. □

REFERENCES

[1] Y. Alkabani and F. Koushanfar. 2007. Active Hardware Metering for Intellectual Property Protection and Security. In *USENIX Security*. 291–306.

[2] ARM. 2013. Cortex-M0 Processor. (2013). https://www.arm.com/products/processors/cortex-m/cortex-m0.php

[3] J.P. Baukus, L.W. Chow, R.P. Cocchi, P.O., and B.J. Wang. 2012. Building Block for a Secure CMOS Logic Cell Library. (2012). US Patent no. 8111089.

[4] J.P. Baukus, L.W. Chow, R.P. Cocchi, and B.J. Wang. 2012. Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing. (2012). US Patent no. 20120139582.

[5] A. Baumgarten, A. Tyagi, and J. Zambreno. 2010. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Des. Test. Comput.* 27, 1 (2010), 66–75.

[6] R. Brayton and A. Mishchenko. 2010. ABC: An Academic Industrial-strength Verification Tool. In *International Conference on Computer Aided Verification*. Springer, 24–40.

[7] F. Brglez, D. Bryan, and K. Kozminski. 1989. Combinational Profiles of Sequential Benchmark Circuits. In *IEEE International Symposium on Circuits and Systems*. 1929–1934.

[8] Q. Chen, A. M. Azab, G. Ganesh, and P. Ning. 2017. PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks. In *ACM Asia Conference on Computer and Communications Security*. 167–178.

[9] Chipworks. 2012. Intel's 22-nm Tri-gate Transistors Exposed. http://www.chipworks.com/blog/technologyblog/2012/04/23/intels-22-nm-tri-gate-transistors-exposed/. (2012).

[10] S. Chu and C Burrus. 1984. Multirate filter designs using comb filters. *IEEE Transactions on Circuits and Systems* 31, 11 (1984), 913–924.

[11] S. Davidson. 1999. Notes on ITC'99 Benchmarks. http://www.cerc.utexas.edu/itc99-benchmarks/bendoc1.html. (1999).

[12] J. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin. 2007. NOC-centric security of reconfigurable SoC. In *IEEE First International Symposium on Networks-on-Chip*. 223–232.

[13] C. Helfmeier, D. Nedospasov, C. Tarnovsky, J.S. Krissler, C. Boit, and J.P. Seifert. 2013. Breaking and Entering through the Silicon. In *ACM SIGSAC Conference on Computer and Communications Security*. 733–744.

[14] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara. 2013. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. In *USENIX Conference on Security*. 495–510.

[15] Maxim Integrated. 2010. DeepCover Security Manager for Low-Voltage Operation with 1KB Secure Memory and Programmable Tamper Hierarchy. https://www.maximintegrated.com/en/products/power/supervisors-voltage-monitors-sequencers/DS3660.html/tb_tab0. (2010).

[16] R.W. Jarvis and M.G. McIntyre. 2007. Split Manufacturing Method for Advanced Semiconductor Circuits. (2007). US Patent 7,195,931.

[17] A.B. Kahng, J. Lach, W. H Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. 1998. Watermarking Techniques for Intellectual Property Protection. In *IEEE/ACM Design Automation Conference*. 776–781.

[18] A.B. Kahng, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, Huijuan Wang, and G. Wolfe. 1998. Robust IP watermarking methodologies for physical design. *Design Automation Conference* (1998), 782–787.

[19] M. Kammerstetter, M. Muellner, D. Burian, D. Platzer, and W. Kastner. 2014. Breaking Integrated Circuit Device Security Through Test Mode Silicon Reverse Engineering. In *ACM SIGSAC Conference on Computer and Communications Security*. 549–557.

[20] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. 2010. Trustworthy Hardware: Identifying and Classifying Hardware Trojans. *Computer* 43, 10 (2010), 39–46.

[21] D. Kirovski and M. Potkonjak. 2003. Local watermarks: methodology and application to behavioral synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 9 (2003), 1277–1283.

[22] F. Koushanfar. 2012. Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management. *IEEE Trans. Inf. Forensics Security* 7, 1 (2012), 51–63.

[23] F. Koushanfar and G. Qu. 2001. Hardware Metering. In *IEEE/ACM Design Automation Conference*. 490–493.

[24] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner. 2006. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *ACM SIGCOMM Computer Communication Review*, Vol. 36. 339–350.

[25] H.K. Lee and D.S. Ha. 1993. Atalanta: an Efficient ATPG for Combinational Circuits. In *Technical Report*.

[26] S. Leef. 2017. In Pursuit of Secure Silicon. http://textlab.io/doc/22959027/mr.-serge-leef--vp-new-ventures--mentor-graphics. (2017).

[27] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D.Z. Pan. 2016. Provably Secure Camouflaging Strategy for IC Protection. In *IEEE/ACM International Conference on Computer-Aided Design*. 28:1–28:8.

[28] M.E. Massad, S. Garg, and M.V. Tripunitara. 2015. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes. In *Network and Distributed System Security Symposium*.

[29] M.E. Massad, J. Zhang, S. Garg, and M.V. Tripunitara. 2017. Logic Locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism. *CoRR* abs/1703.10187 (2017). http://arxiv.org/abs/1703.10187

[30] E.J. McCluskey. 1956. Minimization of Boolean functions. *Bell System Technical Journal* 35, 6 (1956), 1417–1444.

[31] A.L. Oliveira. 1999. Robust Techniques for Watermarking Sequential Circuit Designs. In *IEEE/ACM Design Automation Conference*. 837–842.

[32] T. S. Perry. 2017. Why Hardware Engineers Have to Think Like Cybercriminals, and Why Engineers Are Easy to Fool. (2017). http://spectrum.ieee.org/view-from-the-valley/computing/embedded-systems/why-hardware-engineers-have-to-think-like-cybercriminals-and-why-engineers-are-easy-to-fool

[33] S.M. Plaza and I.L. Markov. 2015. Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking. *IEEE Transactions on CAD of Integrated Circuits and Systems* 34, 6 (2015), 961–971.

[34] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. 2012. Security Analysis of Logic Obfuscation. In *IEEE/ACM Design Automation Conference*. 83–89.

[35] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri. 2013. Security Analysis of Integrated Circuit Camouflaging. In *ACM/SIGSAC Conference on Computer & Communications Security*. 709–720.

[36] J. Rajendran, Huan Zhang, Chi Zhang, G.S. Rose, Youngok Pino, O. Sinanoglu, and R. Karri. 2015. Fault Analysis-Based Logic Encryption. *IEEE Transactions on Computer* 64, 2 (2015), 410–424.

[37] M. Rostami, F. Koushanfar, and R. Karri. 2014. A Primer on Hardware Security: Models, Methods, and Metrics. *IEEE* 102, 8 (2014), 1283–1295.

[38] J.A. Roy, F. Koushanfar, and Igor L Markov. 2010. Ending Piracy of Integrated Circuits. *IEEE Computer* 43, 10 (2010), 30–38.

[39] SEMI. 2008. Innovation is at Risk Losses of up to $4 Billion Annually due to IP Infringement. (2008). www.semi.org/en/Issues/IntellectualProperty/ssLINK/P043785 [June 10, 2015].

[40] K. Shamsi, M. Li, T. Meade, Z. Zhao, D.P. Z., and Y. Jin. 2017. AppSAT: Approximately Deobfuscating Integrated Circuits. In *to appear in IEEE International Symposium on Hardware Oriented Security and Trust*.

[41] Y. Shen and H. Zhou. 2017. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. Cryptology ePrint Archive, Report 2017/290. (2017). http://eprint.iacr.org/2017/290.

[42] J. P. Skudlarek, T. Katsioulas, and M. Chen. 2016. A Platform Solution for Secure Supply-Chain and Chip Life-Cycle Management. *Computer* 49, 8 (2016), 28–34.

[43] J.E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P. D Franzon, M. Bucher, S. Basavarajaiah, J. Oh, et al. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *IEEE International Conference on Microelectronic Systems Education*. 173–174.

[44] P. Subramanyan, S. Ray, and S. Malik. 2015. Evaluating the Security of Logic Encryption Algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust*. 137–143.

[45] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik. 2013. Reverse Engineering Digital Circuits Using Functional Analysis. *IEEE/ACM Design Automation and Test in Europe* (2013).

[46] SypherMedia. 2017. SypherMedia Library Circuit Camouflage Technology. http://www.smi.tv/syphermedia_library_circuit_camouflage_technology.html. (2017).

[47] TechInsights. 2017. Samsung Galaxy S8 (SM-G950W) Teardown. http://www.techinsights.com/about-techinsights/overview/blog/samsung-galaxy-s8-teardown. (2017).

[48] M. M. Tehranipoor, U. Guin, and S. Bhunia. 2017. Invasion of the Hardware Snatchers. *IEEE Spectrum* 54, 5 (2017), 36–41.

[49] R. Torrance and D. James. 2011. The State-of-the-Art in Semiconductor Reverse Engineering. In *IEEE/ACM Design Automation Conference*. 333–338.

[50] P. Tuyls, G. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters. 2006. Read-Proof Hardware from Protective Coatings. In *International Conference on Cryptographic Hardware and Embedded Systems*, Louis Goubin and Mitsuru Matsui (Eds.). 369–383.

[51] A. Vijayakumar, V.C. Patil, D.E. Holcomb, C. Paar, and S. Kundu. 2017. Physical Design Obfuscation of Hardware: A Comprehensive Investigation of Device and Logic-Level Techniques. *IEEE Transactions on Information Forensics and Security* 12, 1 (2017), 64–77.

[52] Y. Xie and A. Srivastava. 2016. Mitigating SAT Attack on Logic Locking. In *International Conference on Cryptographic Hardware and Embedded Systems*. 127–146.

[53] X. Xu, B. Shakya, M.M. Tehranipoor, and D. Forte. 2017. Novel Bypass Attack and BDD-based Tradeoff Analysis Against all Known Logic Locking Attacks. Cryptology ePrint Archive, Report 2017/621. (2017). http://eprint.iacr.org/2017/621.

[54] M. Yasin, B. Mazumdar, S. S. Ali, and O. Sinanoglu. 2015. Security Analysis of Logic Encryption against the Most Effective Side-Channel Attack: DPA. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. 97–102.

[55] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu. 2016. SARLock: SAT Attack Resistant Logic Locking. In *IEEE International Symposium on Hardware*

Oriented Security and Trust. 236–241.

[56] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. 2016. CamoPerturb: Secure IC Camouflaging for Minterm Protection. *IEEE/ACM International Conference on Computer-Aided Design*, 29:1–29:8.

[57] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. 2016. Security Analysis of Anti-SAT. *IEEE Asia and South Pacific Design Automation Conference* (2016), 342–347.

[58] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. 2017. Removal Attacks on Logic Locking and Camouflaging Techniques. *IEEE Transactions on Emerging Topics in Computing* 99, 0 (2017), PP.

[59] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri. 2016. On Improving the Security of Logic Locking. *IEEE Transactions on CAD of Integrated Circuits and Systems* 35, 9 (2016), 1411–1424.

[60] M. Yasin, S. M. Saeed, J. Rajendran, and O. Sinanoglu. 2016. Activation of Logic Encrypted Chips: Pre-test or Post-Test?. In *Design, Automation Test in Europe*. 139–144.

[61] M. Yasin, A. Sengupta, B.C. Schafer, Y. Makris, O. Sinanoglu, and J. Rajendran. 2017. What to Lock?: Functional and Parametric Locking. In *Great Lakes Symposium on VLSI*. 351–356.

[62] M. Yasin, O. Sinanoglu, and J. Rajendran. 2017. Testing the Trustworthiness of IC Testing: An Oracle-Less Attack on IC Camouflaging. *IEEE Transactions on Information Forensics and Security* 12, 11 (2017), 2668–2682.

[63] M. Yasin, T. Tekeste, H. Saleh, B. Mohammad, O. Sinanoglu, and M. Ismail. 2017. Ultra-Low Power, Secure IoT Platform for Predicting Cardiovascular Diseases. *IEEE Transactions on Circuits and Systems I: Regular Papers* PP, 99 (2017), 1–14.