AFRL-RY-WP-TR-2020-0357

A CROSS-LAYER FRAMEWORK FOR COST-EFFECTIVE INTELLECTUAL PROPERTY (IP) PROTECTION

Farinaz Koushanfar
University of California, San Diego

Jeyavijayan (JV) Rajendran and Yiorgos Makris
University of Texas at Dallas

Ozgur Sinanoglu
New York University

FEBRUARY 2021
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2020-0357 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

ORLANDO.POMPEI.L.III.1085949701 Digitally signed by ORLANDO.POMPEI.L.III.1085949701 Date: 2021.01.27 15:28:45 -05'00'

POMPEI L. ORLANDO
Program Manager
Trusted Electronics Branch
Aerospace Components & Subsystems Division

LOCKHART.MARY.E.1380792784 Digitally signed by LOCKHART.MARY.E.1380792784 Date: 2021.01.28 08:52:48 -05'00'

MARY E. LOCKHART, Chief
Trusted Electronics Branch
Aerospace Components & Subsystems Division

BROOKS.ADAM.L.1270115205 Digitally signed by BROOKS.ADAM.L.1270115205 Date: 2021.02.03 17:28:03 -05'00'

ADAM L. BROOKS, Lt Col, USAF
Deputy
Aerospace Components & Subsystems Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
February2021	Final	7 June 2018 – 7 December 2019

4. TITLE AND SUBTITLE
A CROSS-LAYER FRAMEWORK FOR COST-EFFECTIVE INTELLECTUAL PROPERTY (IP) PROTECTION

5a. CONTRACT NUMBER
FA8650-18-1-7827

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62716E

6. AUTHOR(S)
Farinaz Koushanfar (University of California, San Diego)
Jeyavijayan (JV) Rajendran and Yiorgos Makris (University of Texas at Dallas)
Ozgur Sinanoglu (New York University)

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1RY

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California, San Diego
Office of Contract and Grant Administration
9500 Gilman Drive, Dept. 621
La Jolla, CA 92093-0621

University of Texas at Dallas

New York University

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH
45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency (DARPA/MTO)
675 North Randolph Street
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYDT

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2020-0357

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
This material is based on research sponsored by the Air Force Research Lab (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-1-7827. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon." "The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Labs (AFRL), the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Report contains color.

14. ABSTRACT
This report summarizes the progress on the Efficient Cross-Layered IP Protection Scheme (ECLIPSE) project. In addition to developing security metrics and proofs, we apply the proposed stripped-functionality logic locking (SFLL) technique and demonstrate it on a field-programmable gate array (FPGA) platform. The report also highlights our efforts towards protecting multiple outputs of circuit, unlocking circuits using machine learning, and developing a logic tool that implements different variants of logic locking. The report also summarizes the latest benchmarking and red-teaming efforts on our defense as part of Cybersecurity Awareness Worldwide 2019 competition, which was the first logic locking contest. The key takeaway is that our proposed locking technique, SFLL-rem, withstands the test-of-time

15. SUBJECT TERMS
intellectual property, stripped-functionality logic locking, threat model, integrated circuit fabrication

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Pompei Orlando
Unclassified	Unclassified	Unclassified	SAR	40	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

Table of Contents

Section **Page**

List of Figures

Figure **Page**

List of Tables

Table **Page**

1 INTRODUCTION

As the trend of outsourcing integrated circuit (IC) fabrication consolidates, the semiconductor industry faces new challenges. Reliance on off-site *untrusted* fabrication facilities, though economical, has given rise to several threats ranging from intellectual property (IP) piracy to hardware Trojans [1]. As the design houses share their valuable IP in the form of GDSII with the *untrusted* foundry, the foundry can reverse-engineer the gate-level netlist from it with malicious intentions. At the same time, lack of adequate state-of-the-art IP protection techniques has only made the problem worse. It is estimated that several billions of dollars are lost each year due to these threats [2]. Many government/military organizations along with leading semiconductor companies are currently trying to address these concerns [3-5].

To safeguard designs at the silicon layer, designers have resorted to several design-for-trust (DfT) techniques such as IC camouflaging [6], split manufacturing [7], watermarking [8], IC metering [9], and logic locking [10-13]. Recently, logic locking has gained popularity due to its ease of implementation and it being a holistic solution to multiple threats (fab, end-user, etc.). In logic locking, the original design is locked by introducing additional logic that expects a secret key. The key bits are securely fused in a *tamper-proof* [14] memory which needs to be correctly configured for the IC to be functional. The threat model for logic locking is illustrated in Figure 1.

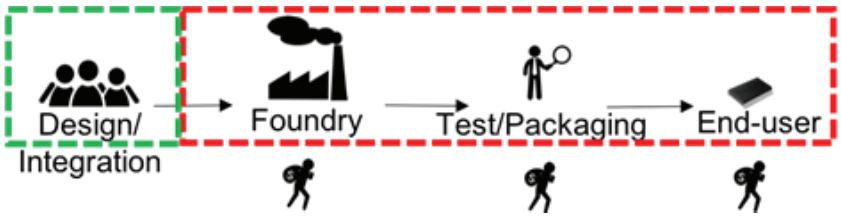

Figure 1: Threat Model for Logic Locking
Every entity in the IC supply chain except the design house can be untrusted

After the initial work [10], a plethora of works advanced the state-of-the-art in logic locking [11, 15]. Nevertheless, a Boolean satisfiability (SAT) based attack broke all the logic locking techniques existing at that time [16]. Though several SAT-resilient works were presented afterwards [17-19], most of them had structural flaws that were eventually exploited to break these schemes [20, 21]. Among these attacks, removal attack can isolate and remove the protection logic from the original design. Only recently, a set of works, known as *stripped-functionality logic locking* (SFLL), were presented that were shown to successfully thwart all the state-of-the-art attacks [12, 13].

1.1 Threat Models

A threat model defines the attacker's capabilities. We define several threat models for logic locking, as presented in Table 1. *In the first threat model*, we assume that the attacker has access to a working chip, which in the context of logic locking, is a chip that has the secret key loaded on its memory. In this case, the attacker can use the chip as an oracle, apply input patterns to it, and collect the responses from the chip. From these input-output pairs, the attacker can do an analysis to infer the logic locking key. In this threat model, a likely attacker with the working

chip would be the end-user; his/her approach would simply be applying inputs in a brute-force fashion possibly in addition to side-channel analysis.

In the second threat model, we assume that the attacker has access to the GDSII representation of the logic locked design IP. So, the attacker can reverse engineer the GDSII to obtain the locked netlist in an effort to pirate the design IP. The attacker's goal is to isolate the key logic to remove it, or from its structure, infer the secret key values [21]. The likely attacker in this threat model is the untrusted fab.

The third threat model is a union of the attacker capabilities in the first two threat models, assuming a powerful attacker who has access to not only the working chip with the key inside, but also the reverse engineered netlist that includes the key logic. Now the attacker can simulate the netlist to produce meaningful input patterns rather than brute-forcing and can apply these input patterns to the working chip to collect the responses. He/she can then figure out the secret key. In fact, many attacks assume this threat model; well-known examples include sensitization attack [15] and SAT attack [16].

Table 1. Different Threat Models showing Attacker's Capabilities

#	Attacker's capabilities		Objective
	RE netlist	**Working chip (oracle)**	
1	no	yes	Corrupt outputs for incorrect keys (high error rate (ER))
2	yes	no	Make key logic difficult to isolate and corrupt outputs for incorrect keys (high ER)
3	yes	yes	Both of the above and protect IP from all attacks (e.g., SAT: low ER)

1.2 Defense Objectives

The business model of a chip design company dictates the trusted and the untrusted entities, and thus, the threat model. There may also be features specific to the system on a chip (SoC) design or its blocks that guide the chip or block designer in choosing the appropriate threat model for the chip/block. Depending on the threat model, the designer then determines the objectives of the logic locking defense.

In the first threat model, the designer needs a logic locking solution to corrupt the chip outputs for wrong keys. This prevents an attacker from using the working chip as a black-box and collecting hints about the secret key every time he/she applies an input pattern on a chip.

Considering the *second threat model*, where the attacker has only the reverse-engineered but locked netlist, the proper defense would be to make the protection logic difficult to isolate in addition to creating corruption for incorrect keys (to still prevent black box usage of pirated IP). Actually, a better strategy would be to modify the IP design before implementing it on silicon, with a secret key loaded post-fabrication restoring the original functionality. This way, the reverse engineered netlist without the knowledge of the key would actually reveal the modified

IP, and not the original IP that the designer is trying to hide from the untrusted parties, e.g., the fab.

As for the third threat model, which is consistent with Kerckhoff's principle of assuming that the attacker knows everything but the key, the defense must meet all the objectives of the previous two threat models, and in addition, thwart all attacks, such as the sensitization and the SAT attack.

1.3 Metrics and Attack Resilience

The basic idea of SFLL is to change the original IP by removing and modifying logic, i.e., *stripping* away some functionality from it. This operation creates a discrepancy between the original and the modified logic function, which on a working chip is fixed by a restore unit controlled by the secret key as illustrated in Figure 2. The input patterns for which the original and the modified functions differ are referred to as the *protected input patterns* (PIPs). The modified function produces an error for the protected input patterns. *Error rate* (ER) is defined as the ratio of PIPs to all input patterns. The removal attack that simply isolates and removes the restore circuitry would be left with the modified function with these errors. The higher the ER, the more resilient the logic locking technique is against the removal attack. Intuitively, it can also be stated that a locked chip with a high ER is more useless as a black box.

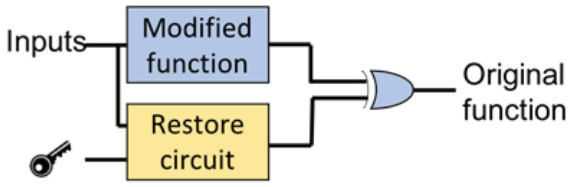

Figure 2: General Logic Locked Circuit: Modified Function along with the Restore Circuitry

Latest research in logic locking has shown that the SAT attack [16] works better when ER is higher [12]; the SAT solver learns more in every iteration when an incorrect key leads to many errors, resulting in quicker convergence for the attack. ER is therefore a fundamental metric that, when controlled, can enable a trade-off between conflicting logic locking objectives. A proper logic locking technique is one that can control ER and give the designer the control over resilience against all attacks. In SFLL-fault, the modified function is obtained from the original one by injecting a fault, and the failing patterns for this fault are the PIPs. By selecting the proper fault to inject into the design (and remove logic from the design), ER can be controlled.

Next we connect the security metric, namely ER, to resilience to different classes of attacks. Obviously, the *brute-force attack* that requires trying all keys blindly will have a complexity that is exponential in the size of the key. When the concern is *removal attacks or the black-box usage* of the locked chips, the resilience can be defined in terms of the output corruption level, which can be computed as the ratio of input patterns for which the outputs show error.[1]

Obviously, the outputs that are not locked will not contribute to the corruption, while the locked outputs will. *SAT attack* resilience of a locked output can be approximated to ER^{-1} on that output. The overall SAT attack resilience of the chip can be conservatively defined as the maximum ER^{-1} overall outputs, assuming that the attacker can target different outputs in parallel. This is a generous assumption in terms of attacker capabilities.

1.4 Organization

This report summarizes our research over the entire span of the Efficient Cross-Layered IP Protection SchemE (ECLIPSE) project. We organize the report into the following sections:

1. Security definitions and formal security analysis (Section 2).
2. Implementation of SFLL on the Defense Advanced Research Projects Agency (DARPA) Common Evaluation Platform (CEP) SoC. We report only on the latest and secure version of SFLL, referred to as SFLL-rem (Section 3).
3. Exploratory research on approximate logic unlocking using machine learning (Section 4).
4. Preliminary study on improving ER and protecting multiple outputs of a circuit (Section 5).
5. Development of a logic locking tool that inputs a netlist and outputs its locked version (Section 6).
6. Demonstration of logic locking on a field-programmable gate array (FPGA) (Section 7).
7. Cybersecurity Awareness Worldwide (CSAW) logic locking competition (Section 8).

[1]This would be the collective ER for all the outputs, while an alternative definition could be the average ER across all the outputs, or equivalently, the Hamming Distance between the corrupted and the correct outputs.

2 FORMAL SECURITY ANALYSIS

2.1 Security Definitions

In the following sections we prove the security of SFLL-rem with rigorous details. However, first we introduce the notations that is extensively used for the remainder of this section.

Notation. A set is defined as S, whereas its elements are denoted as $s \in S$. The event of drawing sample s uniformly randomly from the set S is written as $s \xleftarrow{\$} S$. Moreover, the cardinality of the set S is denoted by $|S|$. A combinational circuit is denoted by ckt, while ckt_{lock}, ckt_{actv}, and ckt_{rec} denote a logic-locked, an activated, and a recovered circuit, respectively. A^{Δ} denotes a *probabilistic polynomial time* (PPT) adversary A following an attack strategy Δ. Without loss of generality, we assume inputs of size n, outputs of size m, and key size of k.

Next, the definition of logic locking is presented. Note that the definition was introduced in [22], however, for the sake of reading, we repeat it below.

Definition 1. *A combinational circuit ckt is a netlist that implements a Boolean function $F : I \rightarrow O$, where $I \in \{0, 1\}^n$ and $O \in \{0, 1\}^m$ with n inputs and m outputs. A logic locking technique L can be viewed as a triplet of algorithms,*[2] (Gen, Lock, Activate)*, where:*

1. Gen *is a randomized key generation algorithm,* $z \xleftarrow{\$} \text{Gen}(1^k)$*, where k denotes the key-size,*

2. Lock *is the algorithm to lock a circuit's functionality,* $ckt_{lock} \leftarrow \text{Lock}_z(ckt)$*, and*

3. Activate *is a deterministic algorithm that activates the locked circuit,* $ckt_{actv} \leftarrow \text{Activate}_z(ckt_{lock})$ *such that* $\forall i \in I, ckt_{actv}(i) = F(i)$.

Next, we provide the definition of security for a logic locking technique L with the help of the following experiment. Note that attack-specific security definitions were introduced in [22, 23]. However, no generic notion of security exists, and thus adapting the notion from [23], we establish it with the help of Experiment 1.

[2]Note that we only provide the definition for combination circuits, but this can be readily extended for sequential designs.

Experiment 1: LL-attack$_{\mathcal{A}^{\mathbb{S}}(\mathcal{L})}(k)$

function INITIALIZE(ckt)
  $z^* \xleftarrow{\$} \mathtt{Gen}(1^k)$ ▷ Chosen by $\mathcal{L}$
  $ckt_{lock} \leftarrow \mathtt{Lock}_{z^*}(ckt)$
  $ckt_{actv} \leftarrow \mathtt{Activate}_{z^*}(ckt_{lock})$
  $win = 0$
end function

function ATTACK(ckt_{actv}, ckt_{lock})
  $S \leftarrow \phi$
  for $j \leftarrow 1$ *to* $q(k)$ **do**
    ▷ $q(k)$ is a polynomial in k
    $p_{\mathcal{L}} \leftarrow \mathcal{A}^{\mathbb{S}}(ckt_{lock}(\cdot), S)$ ▷ $p_{\mathcal{L}}$ is observable from ckt_{lock}
    $p'_{\mathcal{L}} \leftarrow \mathcal{A}^{\mathbb{S}}(ckt_{actv}(\cdot), S)$ ▷ $p'_{\mathcal{L}}$ is observable from ckt_{actv}
    $S \leftarrow \mathcal{A}^{\mathbb{S}}(p_{\mathcal{L}}, p'_{\mathcal{L}}, S)$
  end
  for $j \leftarrow 1$ *to* $q'(k)$ **do**
    ▷ $q'(k)$ is a polynomial in k
    $p_{\mathcal{L}} \leftarrow \mathcal{A}^{\mathbb{S}}(ckt_{lock}(\cdot), S)$
  end
  for $j \leftarrow 1$ *to* $q''(k)$ **do**
    ▷ $q''(k)$ is a polynomial in k
    $p'_{\mathcal{L}} \leftarrow \mathcal{A}^{\mathbb{S}}(ckt_{actv}(\cdot), S)$
  end
  $ckt_{rec} \leftarrow \mathcal{A}^{\mathbb{S}}(ckt_{lock}, ckt_{actv}, p_{\mathcal{L}}, p'_{\mathcal{L}})$
  return ckt_{rec}
end function

function SUCCESS(ckt, ckt_{rec})
  if $ckt(i) = ckt_{rec}(i), \quad \forall i \in I$ **then**
    $win = 1$
  end
  return win
end function

The INITIALIZE function picks the *k*-bit secret key z^* randomly from a uniform distribution over *k*-bit binary strings. After the selection of the key, the circuit *ckt* is locked using a logic locking algorithm *L* with the key z^*. Afterwards, the secret key is loaded on the memory of the chip to activate the circuit, denoted by ckt_{actv}. Next, we initialize a variable *win* = 0. The ATTACK(ckt_{lock}, ckt_{actv}) function takes two entities as input, a reverse-engineered locked netlist ckt_{lock}, and a working chip ckt_{actv} with correct key embedded onto its memory. The attacker $A^{\mathbb{S}}$ initializes the auxiliary information set *S*. Next, he/she analyzes the locked circuit, and records the observation in p_L. Further, he/she makes a query to the oracle, and records this information in p'_L which is gained from the physical implementation of *L* following strategy $\mathbb{S}$. For example, the observable for a SAT attack could be the outputs from the chip. Afterwards, the attacker updates the auxiliary information set *S* from the observations p_L, and p'_L. Note that as the attacker is computationally bounded, his inability to iterate an exponential number of queries forces him to make only a polynomial number of queries $q(k)$ to the oracle, *k* being the key size. The attacker is allowed to make further queries to the locked and activated circuits, and update the observable. Note that an attacker is allowed to choose different sets of queries $q'(k)$, and $q''(k)$ for the locked and the activated circuits, respectively. Finally, reinforced with all the information *S*, p_L and p'_L, the attacker returns ckt_{rec}.

A logic locking scheme is said to be secure if for any PPT attacker $A^{\mathbb{S}}$,

$$\Pr[\text{SUCCESS}_{\text{LL-attack}_{A^{\mathbb{S}}(\mathcal{L})}(k)} = 1] \leq \epsilon(k)$$

where $\epsilon(k)$ is a negligible quantity in k, k being the key size.[3] For the rest of the paper, we will abbreviate $\Pr[\text{SUCCESS}_{\text{LL-attack}_{A^{\mathbb{S}}(\mathcal{L})}(k)} = 1] \leq \epsilon(k)$ as $\Pr[\text{SUCCESS}_{A^{\mathbb{S}}}(k) = 1]$ for simplicity.

2.2 An Overview of SFLL-rem

In this section, we present a secure methodology for SFLL-rem that has minimal reliance on physical synthesis tools. Yet this technique delivers the same control over ER that the other versions of SFLL do.

To effect functionality stripping, we take a similar approach to SFLL-fault, where we work with stuck-at faults.[4] While SFLL-fault [13] must find and store all the failing patterns for a fault as keys, all we need is a fault f which has at least one failing pattern with a sufficiently large number of care bits. Our goal is to make this failing pattern $t_{secure} \in T_f$ the one and only secret key. The fault is injected and logic is removed from F to obtain F_f. Note that finding all the failing patterns for F_f is critical to have a logically equivalent locked circuit. However, a complete enumeration of all the failing patterns for F_f could be computationally infeasible. As the discrepancy between F and F_f includes not only t_{secure}, but all the other failing patterns of f as well ($\{T_f - t_{secure}\}$), we utilize formal equivalence checking to restore the functionality for all the other failing patterns.[5] Instead of explicitly storing all the failing patterns in LUTs, the formal equivalence checker essentially generates the logic which restores all the failing patterns, except for the protected pattern/secret key t_{secure}. The logic added to F_f through the formal equivalence checking process helps create an $F_{f'}$ that differs from F for only the correct key t_{secure}. It is $F_{f'}$ that is implemented on silicon along with the restore circuitry. This approach, in contrast to SFLL-fault, eliminates the need for a tamper-proof LUT, as only one test pattern t_{secure} needs to be stored as the secret key. The restore circuitry is a simple comparator that flips the output of $F_{f'}$ when the input pattern matches t_{secure}.

Example. The original IP is the c17 circuit from the ISCAS benchmark suite [25]. A stuck-at-0 fault is injected at the output of the inverter as shown in Figure 3a to modify F into F_f in Figure 3b. This causes the circuit to fail on the output O23 for the test patterns listed in Figure 3c. Out of these test patterns, we select the pattern x0111 to be t_{secure}, i.e., the key shows with the red box. Next, engineering change order (ECO) is performed to restore F for all the test vectors except for t_{secure} as shown in Figure 3d. This circuit implements $F_{f'}$. Note that the circuits in Figure 3a and Figure 3d only differ for t_{secure} x0111, which is the key for the locked circuit. The final locked circuit is shown in Figure 3d, which includes the restore logic.

[3] A function $\in$ is negligible if $\forall c \in N$, $\exists \gamma 0 \in N$ such that $\forall \gamma \geq \gamma 0$, $\in(\gamma) < \gamma^{-c}$.
[4] Note that our methodology is agnostic to the fault model used.
[5] Note that this type of design changes are typical for engineering change orders (ECOs) to minimize the turnaround time for IC fabrication [24].

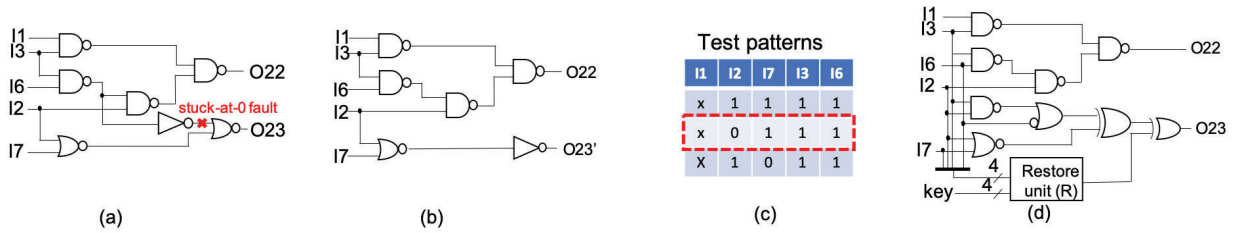

Figure 3: c17 Circuit with Stuck-at-0 Fault shown in Red (a), Fault-injected Circuit F_f (b), List of Test Patterns (c), and Final Locked Circuit with the Restore Unit (d)

2.3 SAT Attack Resilience

The following theorem establishes the security of SFLL-rem against SAT attacker.

Theorem 1. *Proposed SFLL-rem is secure against a PPT attacker following the SAT solving strategy [16], i.e.*

$$Pr[Success_{\mathcal{A}^{\mathrm{SAT}}}(k) = 1] \leq \epsilon(k)$$

Proof. From SFLL, we note that the success of SAT attack is determined by the probability of encountering a protected pattern, i.e., any pattern $i \in t_{secure}$. This probability is equal to $\frac{|t_{secure}|}{2^n}$. As previously stated t_{secure} is an n bit input pattern with k care bits, it contains $(n-k)$ don't care bits. Thus, by construction t_{secure} entails 2^{n-k} input patterns, i.e., $|t_{secure}| = 2^{n-k}$. Thus, for a PPT attacker making $q(k)$ queries to the oracle,

$$\begin{aligned} Pr[Success_{\mathcal{A}^{\mathrm{SAT}}}(k) = 1] &= q(k) \cdot \frac{|t_{secure}|}{2^n} \\ &= q(k) \cdot \frac{2^{n-k}}{2^n} \\ &= \frac{q(k)}{2^k} \\ &\leq \epsilon(k) \end{aligned}$$

where $\epsilon(k)$ is a negligible quantity for a large key size.[6] This concludes the proof.

[6]According to modern computing standard, a key size of 80-bits is considered secure.

2.4 Removal Attack Resilience

Finally, we discuss the resilience of SFLL-rem to the latest removal attacks that compromised SFLL hamming distance (SFLL-HD) and SFLL-flex [26, 27]. Note that SFLL-rem removes logic by injecting fault f into F, thus obtaining F_f, and subsequently adding logic through ECOs into F_f to limit the discrepancy to a single test pattern t_{secure}. Thus, contrary to the SFLL-HD and SFLL-flex instances of SFLL, a skewed signal that would guide the attacker in her netlist analysis is missing in SFLL-rem. The most recent removal attacks [26, 27] that SFLL-HD and SFLL-flex are vulnerable to can thus be thwarted by SFLL-rem. Even if the attacker could successfully identify and isolate the restore circuit, she is left with a circuit ckt_{rec} that implements $F_{f'}$ such that:

$$ckt_{rec}(i) \neq ckt(i) \quad \forall i \in t_{secure}$$
$$|t_{secure}| = 2^{n-k}$$

For example, a functional analysis based logic locking (FALL) attack was proposed which first identifies nodes that implement the functionality stripping by using structural traces and then analyzes the functional properties, called *unate* property, of these nodes to shortlist a small number of candidate locking keys [27]. Since SFLL-rem removes logic (functionality) corresponding to a fault instead of explicitly adding an AND-tree [22], the FALL attack fails during the functional analysis step.

Alternatively, an attacker can identify the traces for the logic added by ECOs, and remove this added logic, she can obtain F_f from $F_{f'}$. Recovering F from F_f, however, necessitates the attacker to guess the logic removed in injecting fault f. The following theorem establishes the security of SFLL-rem against such attack.

Theorem 2. SFLL-rem is secure against a PPT attacker following the removal attack strategy, i.e.

$$Pr[Success_{\mathcal{A}^{\mathrm{REM}}}(k) = 1] \leq \epsilon(k)$$

Proof. Suppose that an attacker is able to trace the logic added by the ECO, and remove this added logic, deriving F_f in the process. However, recovering F from F_f, necessitates the attacker to guess the logic removed in injecting fault f. Nevertheless, the missing (stripped) logic could implement one of 2^{2^k} different possible Boolean functions. For a PPT attacker completely oblivious to the functionality stripping, the probability of success is given by:

$$\begin{aligned} Pr[Success_{\mathcal{A}^{\mathrm{REM}}}(k) = 1] &= \frac{1}{2^{2^k}} \\ &\leq \epsilon(k) \end{aligned}$$

where $\epsilon(k)$ is a negligible quantity for a large key size. This concludes the proof.

3 CASE-STUDY ON COMMON EVALUATION PLATFORM

In this section, we demonstrate the application of logic locking at a granular level on a multi-million-gate SoC [28] provided by DARPA as a Common Evaluation Platform (CEP). An overview of the SoC is depicted in Figure 4. It is a one-master twelve-slave system. The master of the system is a version of the OpenRISC processor, OR1200, which runs its code from a 128KB static random access memory (SRAM). The SoC modules can be broadly categorized into three classes: 1) cryptographic (CRYPTO) blocks, 2) digital signal processing (DSP) blocks, and 3) a global positioning system (GPS) block.

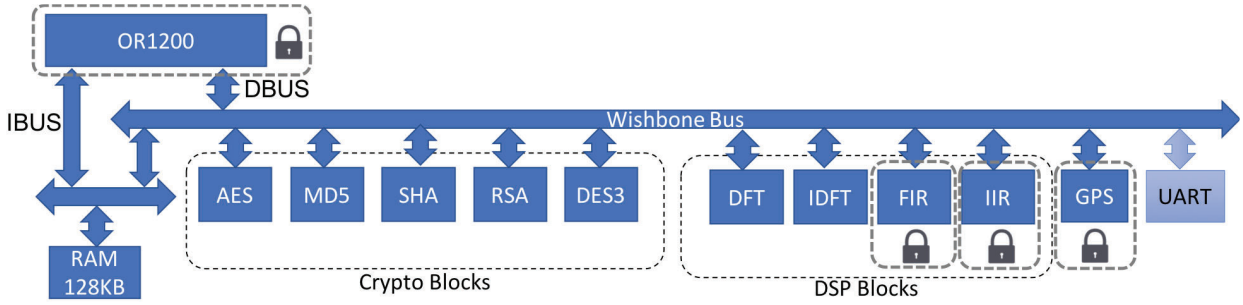

Figure 4: Architecture of CEP consisting of CRYPTO, DSP, and GPS blocks
We lock OR1200, GPS, finite impulse response (FIR), and infinite impulse response (IIR) blocks.

Protecting OR1200 and GPS. Note that only the OR1200 and GPS blocks need protection from IP piracy, as the rest of the blocks are extensively used in the industry, and, thus of public knowledge. In addition, there is full (scan) access to these blocks. We thus lock the OR1200 and the GPS blocks with SFLL-rem along with FLL with respect to threat model three in Table 2. SFLL-rem provides resilience against SAT attacks, and thus, IP piracy; FLL provides corruption and protects against black-box usage and removal attacks.

Table 2. How to Lock the Individual Blocks of SoC

Block	# Gates (K)	Type	Threat Model	Defense
OR1200	29	RISC processor	3	SFLL-rem + FLL
GPS	156	Custom	3	SFLL-rem + FLL
FIR	16	DFT	2	Coefficient locking
IIR	18	DFT	2	Coefficient locking
Overall SoC	15,000		Mixed	Mixed

Protecting DSP. Note that DSP blocks such as finite impulse response (FIR) filter are of public knowledge, and thus, the circuits do not require any protection against IP piracy. However, the secrecy and the working of these blocks lie on the careful choice of their coefficients, which need to be protected from piracy. Moreover, direct access to the block input/outputs (IOs) is not available to the attacker as they are embedded in the SoC; we thus assume threat model two in Table 2. Rather than locking the internal structure of the circuit, we focus on hiding the

coefficients so that an attacker is unable to meaningfully make use of the design. Thus, the coefficients in the DSP blocks constitute the secret key [29].

Limitations of logic locking. Note that the CRYPTO blocks are also of public knowledge that do not require any protection against IP piracy. Yet working oracles are readily available for the attacker, pointing to the third threat model. In these designs, the output bits are affected only by a small number of input bits. For example, in Advanced Encryption Standard (AES), the substitution operation is applied on one byte of data. No matter how these or any other blocks with small logic cones are locked, a high ER is guaranteed; an incorrect key will corrupt the output frequently, as the number of inputs that drive an output is small. SAT attack is thus guaranteed to be effective on designs with small logic cones, such as these CRYPTO blocks.

Overhead analysis. The power, performance, and area (PPA) overheads are plotted in Figure 5.

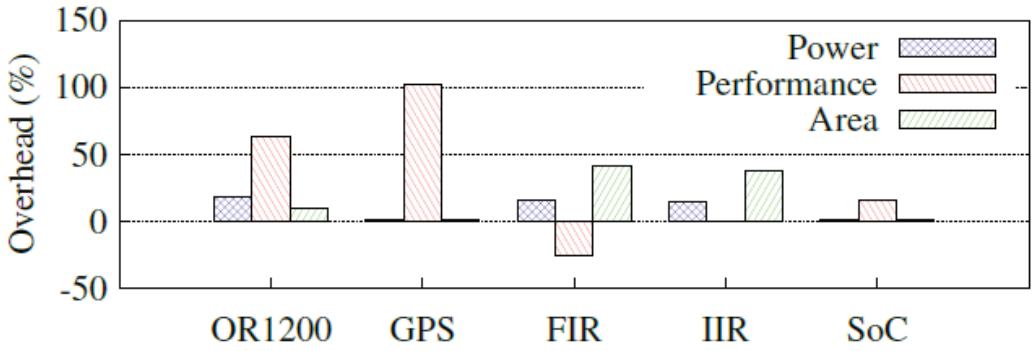

Figure 5: PPA Overhead for the Locked Blocks and the Entire SoC

- **OR1200 and GPS.** The overhead for implementing SFLL-rem+FLL for the OR1200 and GPS blocks is 17.5%, 1.0%, 9.6% and 1.1%, 101.4%, 1.2%, respectively for power, performance, and area. Note that all the blocks were synthesized with the global timing constraint of 4 ns (250 MHz). The critical path length for GPS post-locking was only 2.84 ns, much lower than the SoC timing constraint of 4 ns; as such, we did not pay any effort in optimizing timing in GPS further, and hence, the high performance overhead for the GPS block.

- **DSP.** We hide the coefficients for two of the DSP blocks, namely, FIR and IIR, to safeguard the designs. This comes at a large PPA overhead: 15.3%, -25.8%, and 40.9% for FIR and 14.7%, -0.5%, and 37.7% for IIR, for power, performance, and area, respectively. Without the locking of the coefficients, they are hardcoded in hardware (by replacing multipliers driven by constant values by shifters and adders); while this leads to optimized PPA, reverse-engineering can easily reveal the coefficient values from the logic gates that hardcode them. However, as we hide the coefficients through logic locking, the key in the tamper-proof memory drives the multipliers, disabling any logic optimization through design by contraction. This way, we protect the coefficients from reverse-engineering with a block-level impact on the area footprint, sometimes as high as 50%. However, at the same time, a multiplier in the locked design can sometimes replace

multiple instances of cascaded adders and shifters, thereby, reducing the depth of the circuit, and hence, improving timing.

- **Overall SoC.** The PPA overhead for the entire SoC is 0.45%, 15.3%, and 1.5%, respectively. Even if the overheads are quite high at the block level, we see that the impact of locking is quite reasonable at the SoC level.

- **Run-time.** The run-time to apply the SFLL-rem technique is 4632s and 7435s for OR1200 and GPS, respectively, demonstrating the practicality of our technique as it takes only a fraction of the IP design cycle. We performed coefficient locking for the DSP blocks at the RTL; this was a very straightforward process that did not incur any run-time cost.

- **Generating layout.** The SoC design netlist is taken through various stages of physical design flow in order to create a design rule check/layout versus schematic DRC/LVS-clean GDS for Global Foundries 65lpe technology. The place and route (PnR) was performed with Synopsys IC Compiler, whereas Cadence Virtuoso and Cadence PVS were used for the GDS creation and DRC/LVS checks, respectively. The layout of the SoC is shown in Figure 6 with the locked blocks highlighted in different colors.

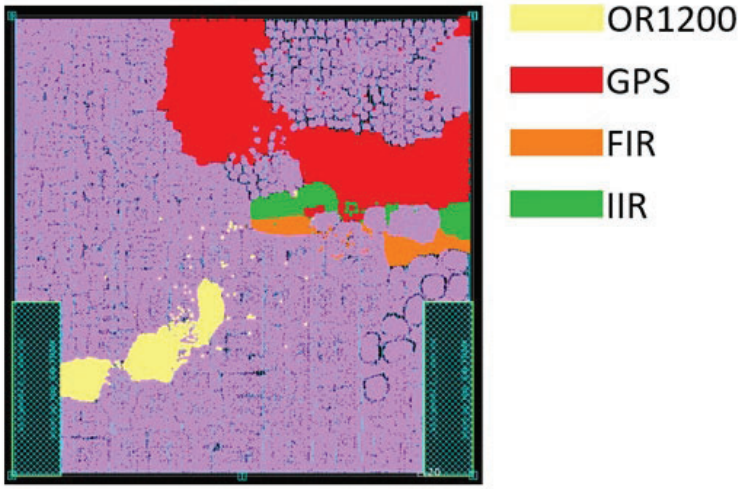

Figure 6: DRC/LVS-clean Layout of the CEP SoC
The locked blocks are highlighted in different colors, whereas the rest of the design is colored in purple.

4 APPROXIMATE LOGIC UNLOCKING USING MACHINE LEARNING

Logic locking inserts additional key gates to the original circuit for protecting the intellectual property of modern integrated circuits. Prior works have identified the vulnerability of logic locking to SAT-based attacks. However, SAT attacks are ineffective on circuits with SAT-hard structures. We develop GenUnlock, the first genetic algorithm-based logic unlocking attack framework to address the above limitation of SAT attacks. GenUnlock formulates logic unlocking (i.e., identifying the correct keys) as a *combinatorial optimization* problem and tackles it using genetic algorithms (GAs). Multiple key sequences form the individuals in the population and undergo the following main operations: circuit fitness evaluation, population selection, crossover, and mutation. The key sequences with high fitness scores 'survive' the selection and are transformed into the offspring. GenUnlock's evolutionary process of key searching features high scalability, exploration efficiency, and parallelizable fitness evaluation. We take an Algorithm/Software/Hardware co-design approach to optimize GenUnlock's runtime overhead. More specifically, we (i) Pipeline each computation stage by automatically constructing auxiliary circuitry for constraints checking, sorting, crossover, and mutation; (ii) Employ *hardware emulation* on programmable hardware for accelerating circuit fitness evaluation.

4.1 Problem Statement

Our objective is to design a systematic methodology for unlocking arbitrary unknown, encrypted circuit. We denote the original unlocked circuit and its encrypted version as C_o and C_e. The primary input, output vector, and the encryption key of the circuit are denoted as $\vec{I} \in \mathbb{B}^M, \vec{O} \in \mathbb{B}^N, \vec{K} \in \mathbb{B}^k$, respectively.

The functionality of the circuit is represented by the following *deterministic mapping*: $C_o(\vec{I}) = \vec{O}$ and $C_e(\vec{I}, \vec{K}) = \vec{O}$. The quality of a decryption key is quantified by the *output fidelity* (OF) that defines the probability of the output vector of C_e being consistent with the one of C_o given any input $\vec{I}$:

$$OF(\vec{K};\ C_o,\ C_e) = \underset{\forall \vec{I} \in \mathbb{B}^M}{Prob}\ [\ C_e(\ \vec{I},\ \vec{K}) = C_o\ (\vec{I})] \tag{1}$$

We consider logic unlocking as successful if the OF of the identified key is higher than the attacker-defined threshold OF > $(1 - \epsilon)$. Note that two different key sequences might result in the same circuit behavior (i.e., same mapping C_e). We define that $\vec{K}_1$ and $\vec{K}_2$ belong to the same equivalence class of keys [16] if the condition $C_e(\vec{I}, \vec{K}_1) = C_e(\vec{I}, \vec{K}_2)$ is satisfied for any $\vec{I} \in \mathbb{B}^M$.

Performance Metrics. We use *effectiveness* and *efficiency* as two main metrics to assess the performance of a logic unlocking scheme. These two metrics are quantified by the attack success rate (defined in Equation (1)) and the execution time, respectively. GenUnlock, for the first time, provides the trade-off between effectiveness and efficiency by generating a set of keys with evolving quality over time. In addition, we also use resource consumption as a metric to evaluate our hardware design.

Threat Model. We make the following assumptions about GenUnlock framework: *(i) The attacker has black-box access to the active IC*. We assume that the adversary can purchase the unlocked circuit from the market and obtains oracle access to it. As a result, the attacker is able to query the active IC with arbitrary input challenges and observe the corresponding outputs, which is the basis of GenUnlock's training data generation phase. *(ii) The attacker knows the netlist of the encrypted circuit.* We assume the attacker can reverse engineer the netlist of C_e from a physical circuit by performing depackaging, delayering, and imaging [30]. The obtained netlist is converted to conjunctive normal form (CNF) and used in circuit fitness evaluation.

Real-world Use Cases. Existing works focus on unlocking the circuit with perfect accuracy, thus may incur prohibitive runtime overhead to break large circuits. Here, we want to emphasize that *fast, approximate* decryption of the target circuit can be more threatening than slow, full decryption. This is particularly true for *fault-tolerant* applications. Let us consider block-chain mining as a real-world example where the signature of the cryptocurrency is extracted from AES and hashing operations [31] on the hardware miner. The resulting signature is continuously checked against the pre-defined template to determine whether the cryptocurrency is legitimate. As such, it is sufficient for the user to find a key that yields correct outputs with high probability in order to obtain financial benefits. Emerging application-specific integrated circuit (ASIC) accelerators for deep neural networks (DNNs) are also inherently fault-tolerant, which has been exploited for parameter quantization or pruning.

4.2 Attack Methodology

Figure 7 illustrates the global flow of GenUnlock. Our framework consists of two stages: (i) Offline pre-processing phase that generates training data for GA; and (ii) Key searching phase that performs key evolution. The one-time pre-processing phase is performed via oracle access while the key searching phase is accelerated using FPGA.

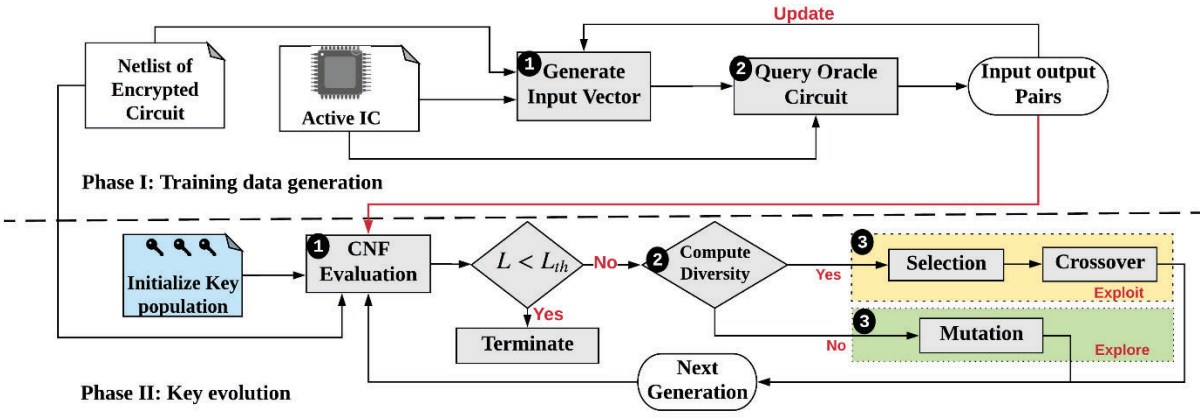

Figure 7: Global Flow of GenUnlock Framework for Logic Unlocking

Prior works have identified that there might be more than one correct keys to unlock the given circuit [16]. This is due to the fact that logic locking schemes, by default, do not guarantee the uniqueness of the decryption key. The collection of these key sequences is called *'equivalent class'* of the correct key sequence. GenUnlock leverages this fact and processes multiple keys representing different equivalence classes in each iteration, thus features higher efficiency for

space exploration. Note that GenUnlock is oblivious of the underlying encryption schemes used by the defender, thus is genetic and applicable to arbitrary ICs. We detail the two key phases of GenUnlock framework in the following of this section.

Phase I: Training data generation. This is an offline, one-time process, consisting of the following two tasks:

(1) Generate input vectors. Given the netlist of the encrypted circuit, we craft input vectors and filter the ones that result in the same circuit outputs when different keys are applied.
(2) Query active IC. The remaining input patterns from step 1 are then used to query the active IC. The collected (IO pairs form the training dataset for our logic unlocking.

Phase II: Key evolution. Once the training data for the target circuit is generated in Phase I, GenUnlock performs three subroutines during the key evolution phase as shown in the bottom of Figure 7:

(1) Circuit fitness evolution. Analogous to natural selection, the key sequences with higher fitness scores are maintained and transformed to offsprings at each iteration. The fitness of each key is evaluated by the ratio of output matching on the training dataset when the specific key is applied. We convert the netlist representation to CNF to facilitate fitness evaluation.
(2) Population diversity computation. GenUnlock separates genetic operations into two groups (*'exploitation'* or *'exploration'*) and determines which branch to take depending on the population diversity. Since key sequences are binary-valued in logic locking, we use the dispersion (i.e., variance) of the population as the measurement of diversity. The formula of computing diversity is given in Equation (2).

$$div(S_K) = \frac{1}{P}\sum_{i=1}^{P}\sqrt{\sum_{j=1}^{k}[S_K(i,j) - \bar{S_K}(j)]^2} \tag{2}$$

where $\bar{S}_K(j) = \frac{1}{P}\sum_{i=1}^{P} S_K(i,j)$ is the sample average of all individuals at j^{th} bit. Here, P is the population size, k is the key length, $S_K \in \mathbb{B}^{Pxk}$ is the current population, and $S_K(i,j)$ denotes the j^{th} bit of the i^{th} individual in the population S_K.
(3) Diversity-guided GA execution. Algorithm 2 outlines the steps of GenUnlock. We apply genetic operations on the current population based on the computed diversity. As opposed to traditional GAs that perform all genetic operations in each iteration, our *dynamic, diversity-aware* GA execution demonstrates better convergence.

Algorithm 2: Genetic Algorithm for Logic Unlocking.

Input : etlist of target encrypted circuit (C_e); Size of the encryption key (L); Training dataset (S_I, S_O); GA parameters, including the population size (P), maximum number of generations (G), number of high-fitness (h) and low-fitness individual (l) for selection, number of child (c) for each pair of parent, and mutation rate m; Diversity threshold (d_{low}, d_{high}); Error tolerance of the attack (ϵ).

Output: A set of feasible key values ($\{\vec{K}\}$) that can unlock the circuit C_e.

Initialization: $S_K = \vec{K}_1, ..., \vec{K}_P \leftarrow generate\_population(L,\ P)$. $i \leftarrow 0$
while $i < G$ *and* $F_K < 1 - \epsilon$ **do**
  $F_K \leftarrow evaluate\_population\_fitness(S_K,\ S_I,\ S_O)$.
  $div \leftarrow compute\_population\_diversity(S_K)$
  if $div < d_{low}$ **then**
    $GA\_mode \leftarrow$ '$explore$'
  else if $div > d_{high}$ **then**
    $GA\_mode \leftarrow$ '$exploit$'
  if $GA\_mode ==$ '$exploit$' **then**
    $S_K \leftarrow select\_next\_generation(S_K,\ F_K,\ h,\ l)$.
    $S_K \leftarrow crossover(S_K,\ c)$
  else if $GA\_mode ==$ '$explore$' **then**
    $S_K \leftarrow mutate\_population(S_K,\ m)$
  if $F_K > 1 - \epsilon$ **then**
    break `// Check termination condition`
  $i \leftarrow i + 1$
end

Return: Obtained a set of circuit deobfuscation keys S_K.

We detail the mechanism of core GA operations as follows:

- **Fitness Evaluation.** The definition of fitness is task-specific. Since our objective is to find (a set of) feasible decryption keys with high OF, we use the matching ratio of the specific key on the training data as the fitness measurement as shown in Equation (3). To facilitate the computation, GenUnlock first automatically constructs *auxiliary comparator components* that are added to the netlist of C_e, resulting in an evaluation netlist C_e^{aux}. Each comparator is implemented as an XNOR gate with two inputs where one of them comes from the ground-truth output in the training dataset. The auxiliary netlist is then converted to CNF to compute the fitness score based on Equation (3).

$$F_K = \frac{\#\ matched\ CNF\ clauses}{\#\ total\ CNF\ clauses} \tag{3}$$

- **Population Selection.** As a step of 'exploitation', the diversity of the population decreases after population selection. GenUnlock determines high-fitness individuals using the *tournament selection* technique [32]. A random subset of the current population is selected to participate in each round of the tournament. The individual with the highest fitness score is maintained in the next generation. Such a selection process repeats until the size of the resulting new generation reaches the desired number of high-fitness individuals (h). We also incorporates several (l) 'lucky' individuals with relatively low fitness in the next generation in order to increases the randomness and help GA escape local optima.
- **Crossover.** Crossover (also called 'breeding') is the other step in 'exploitation'. In this process, the 'genome' (encoding) of the parents are *recombined* to produce the offsprings. Crossover consists of the following two subroutines: (i) Parent pairing: given the current population, we randomly assign two individuals as a pair of parents without repeating the use of an individual. (ii) Offspring generation: each bit of the child sequence is obtained from a uniform random sampling of the corresponding bit from its parents.

- **Mutation.** As such, *mutation* is performed in the *'exploration'* mode of GenUnlock when the population diversity is lower than the pre-defined threshold. There are two key parameters in the mutation process: the chance of mutation and the level of mutation. The first parameter determines the probability that mutation occurs on a particular individual. The second parameter dictates how many bits in the key sequence will be *flipped* as a result of mutation. A high chance and/or a large magnitude of mutation will result in large fluctuation of the fitness scores of the population, making the GA training unstable.

4.3 Hardware Optimization for Attack Acceleration

We leverage an Algorithm/Software/Hardware approach to accelerate the key searching process for the target circuit. Figure 8 illustrates the overview of GenUnlock's hardware architecture consisted of a computing engine for circuit emulation and an auxiliary circuitry for genetic operations. We empirically identify that *circuit fitness evaluation* is the bottleneck of GenUnlock's execution time. To accelerate circuit evaluation, we deploy *circuit emulation* on the programmable hardware to obtain the response of the encrypted circuit (C_e) for the given input signals and the tested key. To reduce data communication between the off-chip DRAM and the FPGA, we perform all computations of key evolution on-chip. Note that we do not include a random number generator (RNG) in GenUnlock's hardware design. Instead, we store a set of random numbers pre-computed on a central processing unit (CPU) using the inherent variation of the operating system. The results of circuit emulation are used for computing fitness scores using Equation (3) during CNF evaluation. The clause checking process in CNF evaluation is parallelized by accommodating multiple Checking Engine (CE) in GenUnlock's design. The workload for each CE is partitioned evenly offline. Furthermore, GenUnlock automatically constructs the customized auxiliary circuitry to pipeline each computation stage and reduce the runtime. As shown in Figure 9, the ping-pong buffer enables pipelined execution of hardware emulation and CNF evaluation.

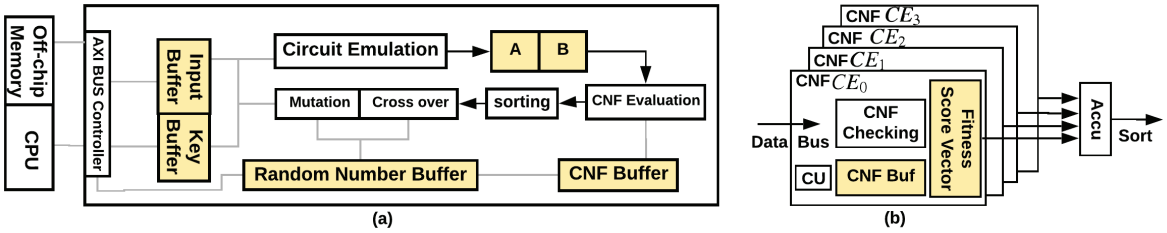

Figure 8: Overview of GenUnlock's Hardware Design
The overall layout of the hardware system (a) and the implementation of CNF Checking Engines (b) are shown.

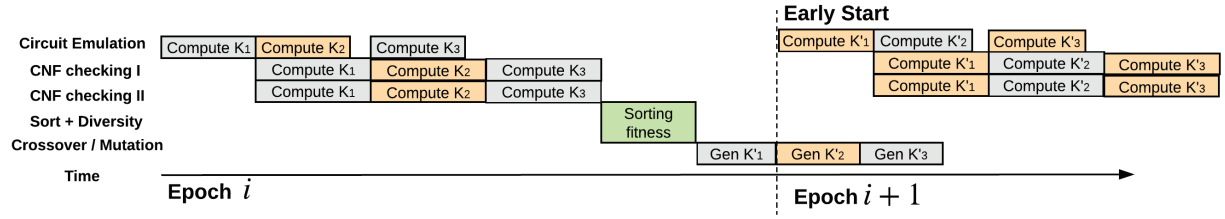

Figure 9: Pipelining Optimization deployed in GenUnlock's Genetic Algorithm Accelerator for Logic Unlocking

4.4 Experimental Results

Experimental Setup. We implement GenUnlock in Python and demonstrate its performance on various bench-marks, including ISCAS'85 and Microelectronics Center of North Carolina (MCNC) [33]. Experiments are run on an Intel i7-7700k processor with 32 GB of RAM and the energy consumption is measured using *pcm-monitor* utility. We use the open-sourced code of the SAT attack [16] as our baseline comparison. Note that [16] is implemented in C++ and tested on a more powerful CPU (Intel Xeon E31320). As such, our empirical results serve as a conservative *relative speedup* comparison.

Our FPGA prototype is implemented on Zynq ZC706 board using the high-level synthesize tool Xilinx SDx 2018.2. GenUnlock's CNF checking engine and the auxiliary GA accelerator discussed in Section 4.3 are implemented using high-level programming language. Our design is synthesized using a clock frequency of 100MHz. The power of FPGA is measure at the socket using a power meter during the execution of the GenUnlock. Throughout our experiments, we set the number of CEs to $N_{ce} = 16$ and the encryption overhead to 10% with [11] as our default setting. As for our GA, we use a key population size $P = 80$ and the total number of generations $G = 50$. The number of high-fitness and low-fitness individuals are set to $h = 54$ and $l = 6$ for selection. Each pair of parents produces $c = 4$ children during crossover. The mutation rate is set to 2%. We generate 50 input/output pairs from the active IC to construct the training data.

Figure 10 shows the comparison between GenUnlock's software/hardware implementation with the baseline [16]. Note that we use the average runtime on each benchmark to visualize the performance comparison in Figure 10. Several circuits cannot be decrypted by the baseline algorithm within 10 hours. In this case, we use 10 hours as the estimated runtime of [16] in Figure 10. With dedicated hardware design support, GenUnlock delivers on average 4.68x speedup compared to the baseline method. For SAT-hard circuits (such as c2670, c7552, *des*), our method engenders superior performance compared to SAT-based attacks, achieving 90x,13x, 2.1x speedup on CPU and 1014x, 153x, 31.2x speedup on the dedicated hardware. Besides the latency comparison, we also measure the power consumption of different circuit deobfuscation methods. The power consumption of 'GenUnlock+HW' on Zynq SoC is measured via the socket when the application is running. On average, GenUnlock with hardware optimization consumes 13.6W power while our software implementation consumes 53.3W power on CPU. Considering the runtime, the overall energy-efficiency of GenUnlock is 18.3x higher than the SAT-based method.

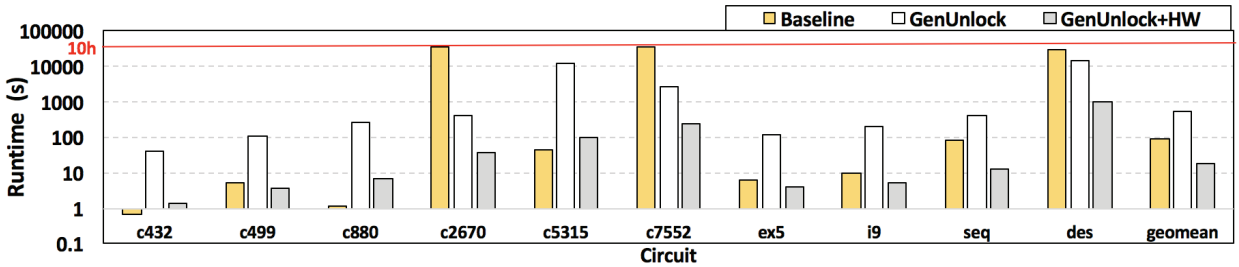

Figure 10: Average Runtime comparison between GenUnlock and the Baseline SAT Attack [16]

'GenUnlock' and 'GenUn-lock+HW' denotes the latency of our software implementation and accelerated FPGA implementation, respectively.

5 PROTECTING MULTIPLE OUTPUTS

5.1 Problem Description

We can protect multiple input patterns for increased output entropy of a design while still ensuring resilience to SAT attack. We first explore cost-effective hardware implementations; for SFLL-HD the modified logic cone will fail for multiple input patterns that are of a certain Hamming Distance *d* to the secret key and the restore logic will recover the output to its correct value when the Hamming Distance of an input pattern from the key equals *d*. Such an implementation will protect multiple patterns through a cost-effective use of a multi-point function. One of the options to increase output entropy and simultaneously protect a larger number of outputs is to protect a circuits multiple instances of the restore circuit. Accordingly, we study the impact of protecting a circuit using 1) a single restore unit and 2) two restore units. The core idea is explained in Figure 11. The circuit in Figure 11(b) is locked circuit with a single restore unit, whereas the circuit in Figure 11(c) is locked using two restore units. K_1 and K_2 are the two keys for two restore units, respectively. While traditionally SFLL protects only the primary outputs, we also explore protecting internal nodes in the circuit. To select the nodes-to-be-protected, i.e., the nodes where functionality-stripping is effected without impacting the desired security level, we deploy and compare the following three strategies:

1. **MaxPO** selects nodes that lead to the protection of the maximum number of outputs.
2. **MaxFI** is inspired by FLL [34] and uses the "fault impact (FI)" metric to select the node(s)-to-be-protected.
3. **MinCn** captures the influence of a node on the primary outputs using the notion of normalized controllability. The lower the normalized controllability of a node, the higher its influence on the outputs.

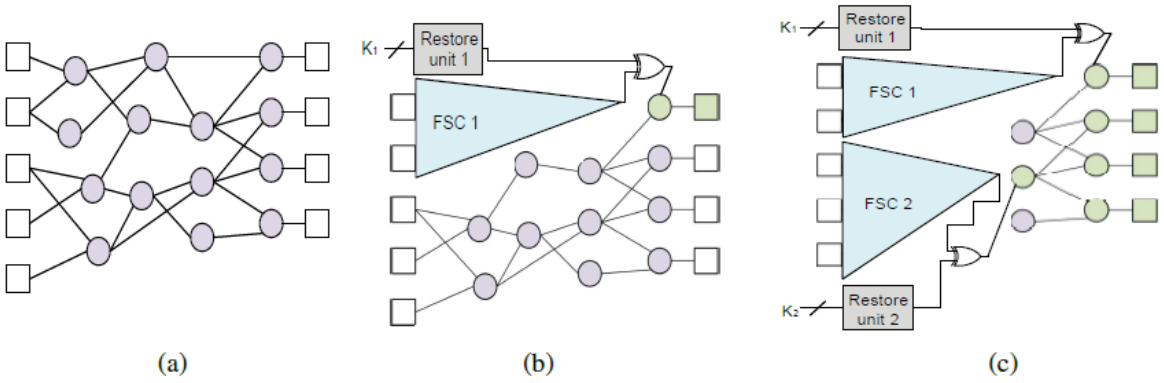

Figure 11: The Original Circuit (a), Traditional Application of SFLL (only one output is protected, which is marked in green) (b), and Proposed Protection using Multiple Restore Units (leading to the protection of multiple primary outputs as well as higher output entropy) (c)

5.2 Experimental Results

ER vs. # of protected outputs Figure 12(a) and Figure 12(b) report the output ER for one and two restore unit(s), whereas, Figure 12(c) reports the number of outputs that are protected. In this set of experiments, each restore unit has a key size of 32. The circuits we considered are controllers of UltraSPARC processor. Let's first compare the strategies in terms of the number of outputs protected. As expected, MaxPO almost always outperforms MaxPI and MinCn in this respect. However, we emphasize that for K=32, the maximum number of outputs any scheme could protect is only three. We attribute this to the "isolated" nature of the circuit graphs in the processor controllers. MaxPO may be a simple and effective strategy to maximize the number of protected outputs, especially in circuits with inherent logic sharing. MaxPO, however, fails to achieve high output ER.

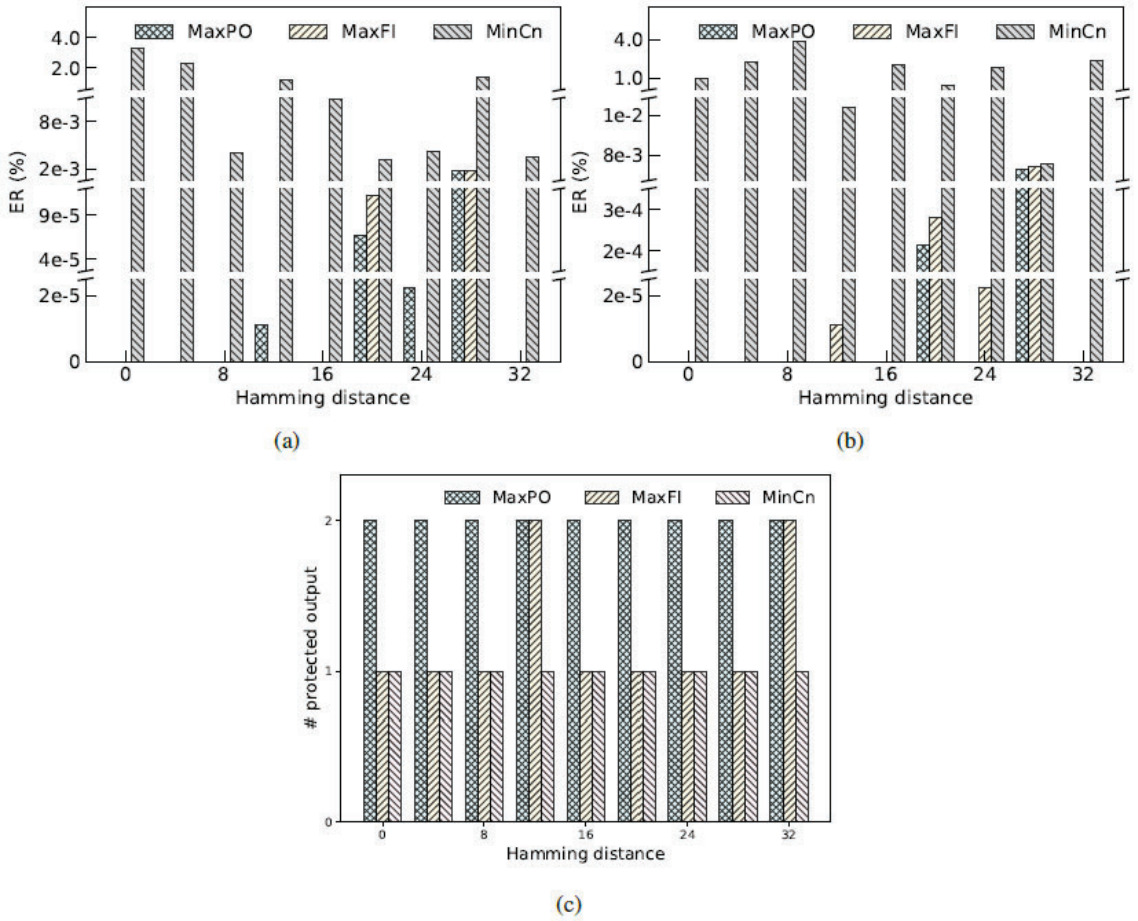

Figure 12: Output ER for One Restore Unit (a), Two Restore Units, for Three Node Selection Strategies (b), and the Number of Outputs Protected (c)

When it comes to output ER, MinCn clearly outperforms the other strategies. We observe that MinCn tends to choose nodes close to primary outputs. In many cases, it chooses a primary output, which implies that the error injected by SFLL appears as is on the circuit. MaxPO and MaxFI, however, tend to select internal nodes and exhibit much lower output ER. When

intermediate nodes are selected, the error injected may be masked by the logic between the protected node and the primary outputs.

Discussion: Most practical strategy In the aforementioned results, the key size is set to 32 for each restore unit. A consequence of key size is that the AC rate remain low for certain circuits and may be unacceptable for certain applications. In the following set of experiments, we report the AC upon changing both key size and HD. Two restore units, each with key size K, are inserted in the circuit. We compute the AC rate by randomly applying 100,000 test patterns to each locked circuit. Table 3 reports the configuration (K, HD) that leads to the highest AC rate for a given strategy and circuit. Note that the highest key size is achieved mostly for K=8. When we consider only the cases with K≥ 16, MinCn always achieves the highest output ER.

Table 3 also shows that when using the strategy MaxPO or MaxFI, a high AC is achieved for only small key sizes. Thus, these strategies can achieve a high AC albeit at the expense of the security level. The strategy MinCn, however, allows us achieve both a high AC rate and a reasonable security level. Recall that these are mainly the controllers of UltraSPARC processor. The optimal strategy may vary depending for other classes of circuits.

Table 3. Highest Average ER (AC) achieved and Configuration (K, HD) for Different Circuits and Strategies

		Single restore unit			Two restore units		
Circuit	**Strategy**	**AC (%)**	**K**	**HD**	**AC (%)**	**K**	**HD**
fpu_in	MaxPO	3.42×10^{-2}	8	4	0.187	8	4
	MaxFI	0.143	8	8	0.311	8	4
	MinCn	3.11	32	4	5.99	32	32
ifu_dcl	MaxPO	6.80×10^{-2}	8	8	0.105	8	8
	MaxFI	0.197	8	4	0.297	8	4
	MinCn	3.24	32	0	3.83	32	8
ifu_fqi	MaxPO	2.68×10^{-2}	8	8	4.12×10^{-2}	8	4
	MaxFI	8.77×10^{-2}	8	4	0.244	8	8
	MinCn	3.78	16	0	3.40	32	16
k2	MaxPO	0.136	8	8	0.334	8	8
	MaxFI	1.57	8	8	1.54	8	8
	MinCn	5.76	8	4	3.78	8	8
lsu_rw	MaxPO	3.52×10^{-2}	8	8	5.22×10^{-2}	8	4
	MaxFI	0.204	8	4	0.294	8	8
	MinCn	3.74	8	8	6.79	16	0
s5378	MaxPO	3.05×10^{-2}	8	4	4.66×10^{-2}	8	8
	MaxFI	0.337	8	4	0.699	8	4
	MinCn	3.89	8	8	2.48	16	12
seq	MaxPO	0.174	8	8	0.306	8	8
	MaxFI	0.172	8	8	0.428	8	4
	MinCn	5.34	16	16	5.35	16	12
tlu_mmu	MaxPO	1.55×10^{-2}	8	8	3.75×10^{-2}	8	8
	MaxFI	0.106	8	4	0.533	8	8
	MinCn	2.69	16	12	2.40	32	16

6 LOGIC LOCKING TOOL

Figure 13 presents an overview of the logic obfuscation tool that we developed. The tool can incorporate different variants of logic locking. The tool takes as input an RTL (Verilog/VHDL) file along with the relevant security parameters and outputs a locked/protected file along with the relevant security metrics and the implementation overhead. Note that the security parameters and metrics can be specific to the logic locking technique. This is further illustrated in Figure 14 that depicts the help screen of the current version of the tool.

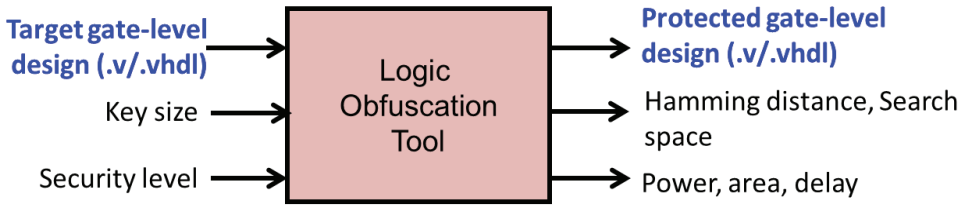

Figure 13: Interface to the SFLL Tool that Implements Three Variants of Stripped-Functionality Logic Locking

```
:: python SFLL.py --h
        Options for the SFLL tool:

        -i [str] ; Original circuit file (input)

        -o [str] ; Locked circuit file (output)

        -t [h SFLL-HD] [f flex] [fl fault]; Locking technique (default: SFLL-HD)

        -k [int] ; Target key size [-t h]

        -h [int] ; Hamming distance for SFLL-HD [-t h

        -n [int] ; Number of HD-units in SFLL-HD [-t h]

        -s [int] ; Target security level [-t f] [-t fl]

        -p [str] ; Protected pattern file [-t f]

        -m [int] ; Max number of (internally generated) protected patterns [-t f]

        -d [str] ; dc_shell options
```

Figure 14: Demonstration of the Features provided by the Current Version of the Logic Obfuscation Tool

7 FPGA DEMONSTRATION

The key objective for this demonstration is to implement SFLL-HD logic locking technique and quantify its effect using workloads running on the microprocessor. To that degree, we use instruction per cycle (IPC) to measure the overall throughput of the design. Given the correct key, the processor performs at its peak performance level.

However, with an incorrect key, the throughput is nearly halted, yielding an overly reduced IPC value. This new implementation for the logic locking technique is hereafter addressed as *performance locking*. In summary, the unique key provided by the authorized user *unlocks* the performance locking inserted inside the design.

We have implemented performance locking in the mor1kx-cappuccino microprocessor pipeline [35]. The highly parametrizable and in-order microprocessor is part of the CEP v1.2 SoC framework [28]. Figure 15 shows the 6 stage, single-issue pipeline of the mor1kx-cappuccino microprocessor. The fetch stage fetches a single instruction every clock cycle and the combinational decode stage generates different opcodes for the remainder of the pipeline modules. The execute module contains the arithmetic/logic unit (ALU) implementation for arithmetic and register file destination operations. Most of the core's functionality is controlled in the control unit which is responsible for majority of the pipeline's control signals. It also issues access to the special purpose register (SPR) and the debug unit. Finally, the memory and write-back module completes the instruction operation by finding and writing the result at the memory location.

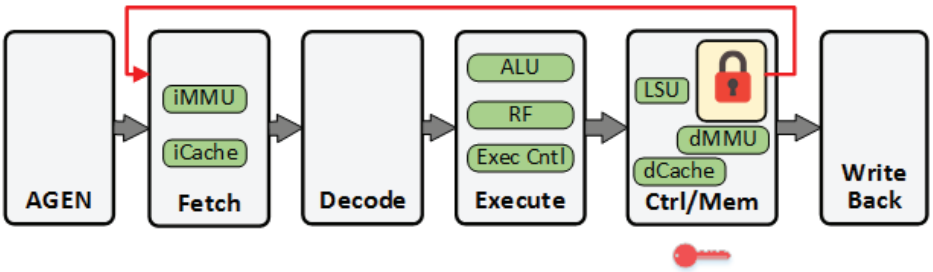

Figure 15: Performance Locking Implemented in the Control Unit of the mor1kx-cappuccino Microprocessor
A unique 128-bit key is required to unlock the performance.

The lock is inserted inside the control module of the microprocessor's pipeline. Specifically, we modify the operation of the new fetch instruction signal, and implement the locking such that, if the wrong key input is received and the protected input pattern appears at the module's input, the lock triggers and activates for *N* clock cycles. During this period, no new instruction is fetched, essentially inserting *bubbles* in the pipeline. The injected non-computing *bubbles* ensures that the overall IPC of the microprocessor is reduced significantly, resulting an effective performance locking.

7.1 Evaluation

7.1.1 Experimental Setup

The SoC with the performance locked mor1kx-cappuccino microprocessor is implemented on the Virtex-7 series FPGA evaluation board. The framework contains RAM, universal asynchronous receiver-transmitter (UART), clock generation, AES etc. modules, all connected using the AXI bus interface and runs different benchmarks supported by the OpenRISC toolchain. We have compiled three different benchmarks from the MiBench [36] suite using the or1k-gcc compiler. The CEP framework also provides script to convert the binary files into FPGA memory file which is used as the workload in the FPGA implementation. Each benchmark is profiled for first 5M instructions in order to find benchmark specific protected input patterns. These unique input patterns trigger the performance locking and reduces overall IPC for an incorrect key input.

We then obtain the baseline performance results, which is logged by executing all three benchmarks on the mor1kx-cappuccino microprocessor prior to implementing any performance locking. Next, we implement the performance locked control module in the pipeline stage and assess the degraded performance of the microprocessor. The program (.bit file) is loaded on the board using the onboard JTAG interface. The LCD unit is modified to display the instantaneous IPC value and the output of the benchmark is passed to the terminal via the UART interface. The functional equivalency between the locked and baseline design is ensured by comparing each benchmark's output displayed on the terminal. The overall setup for the FPGA implementation is shown in Figure 16.

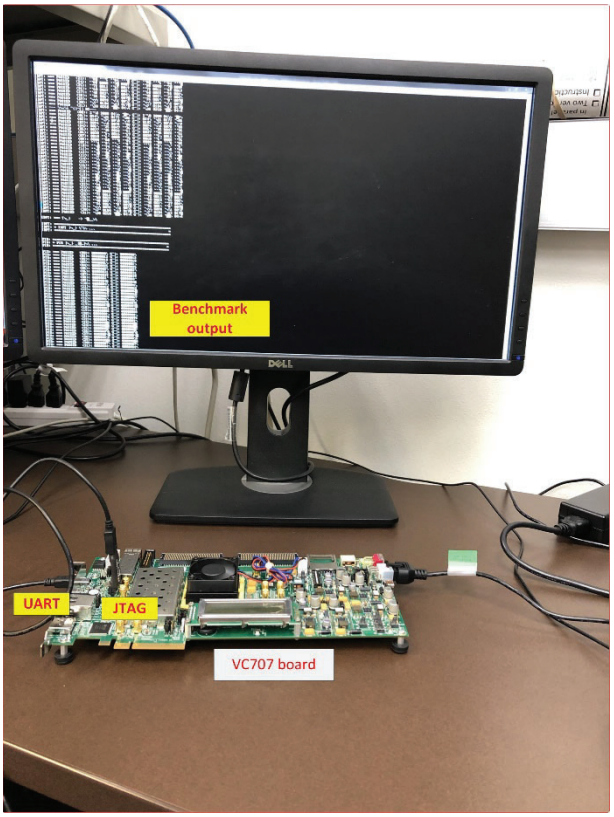

Figure 16: FPGA-based Hardware Implementation Setup for Performance Locking

7.1.2 Performance Locked Design Results

The effectiveness of the performance locking is evaluated for both 1K and 2K stall cycles for each benchmark against the baseline performance. Figure 17 shows the result for the performance locked design. On average, we achieve 44% and 58% performance degradation for 1K and 2K stall cycles, respectively.

Finally, the overhead analysis shows nominal (0.2%) power consumption and no fabric utilization increase by the FPGA for the performance locked design. This affirms the minimal power/area footprint by the SFLL-HD logic locking.

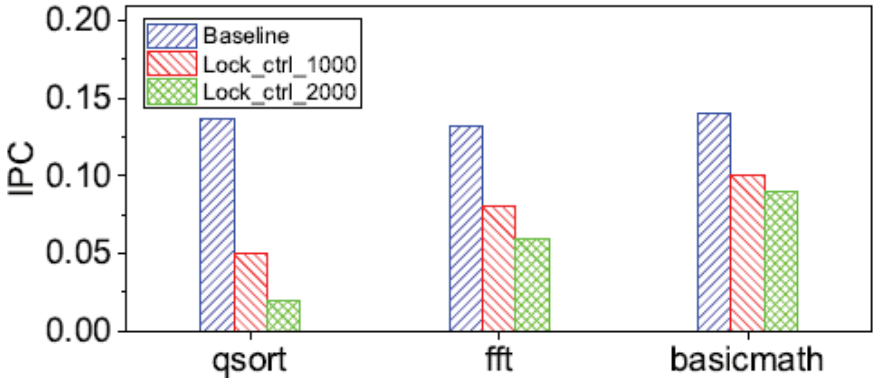

Figure 17: Overall IPC Degradation in FPGA Implementation

7.2 Results

SFLL-rem still stands.

8 NYU CSAW 2019: LOGIC LOCKING CONQUEST

In an effort to usher logic locking towards greater maturity, New York University (NYU) (led by Professor Karri) embarked on an initial community-guided benchmarking of logic locking. This took the form of a red team/blue team benchmarking ”competition” that aimed to establish a common framework for evaluating the security of locking techniques and efficacy of attacks on logic locking. By bringing together different research groups with interested parties from government agencies and private sector industry, this effort aimed to build capability towards something akin to a National Institute of Standards and Technology (NIST)-style standardization effort (for example, the currently ongoing post-quantum cryptography contest [37]).

In this competition, a third-party coordinator (Karri and his team) facilitated interaction between red teams and blue teams. First, the members of the community were invited to propose, discuss, and refine assessment criteria. Blue teams prepared sets of locked combinational and sequential circuits, with accompanying collateral as requested by the red teams. Then, the red teams were unleashed upon the common set of locked combinational and sequential circuits, where they applied various attack strategies. The results from this endeavor provide the community with a comparison of different techniques’ attack successes and a measure of the locking techniques’ resiliency in the face of such attacks.

The first ”Logic Locking Conquest” (LLC) was hosted by Karri’s team at NYU as part of the annual student-run NYU CSAW 2019, with 18 teams from 14 affiliations participating over the course of around 3 months.

Algorithm 3: Hamming Distance (HD)-based Attack

Data: A SFLL-fault cone $\mathcal{C}_{locked}$, oracle $\mathcal{O}$, parameter d
Result: Correct key key_c
$\mathcal{C}_{FSC} \leftarrow$ extract_FSC($\mathcal{C}_{locked}$) ;
$Reduced\_PIT \leftarrow$ extract_PI_table($\mathcal{C}_{FSC}$) ;
$PIP_{cand} \leftarrow \varnothing$;
for $pi \in Reduced\_PIT$ **do**
  $PIP_{cand} \leftarrow PIP_{cand} \cup \{p : HD(p, pi) \leq d\}$;
end
for $p \in PIP_{cand}$ **do**
  if $\mathcal{O}(p) \neq \mathcal{C}_{FSC}(p)$ **then**
    $key_c \leftarrow$ SAT_simulation($\mathcal{C}_{locked}, \mathcal{O}, p$);
    return key_c ;
  end
end

8.1 Combinational Logic Locking Defense in Competition: Our SFLL-rem [38]

Blue team for combinational logic locking was our (Sinanoglu) team from NYU. In this benchmarking effort, we adopted two techniques in tandem: Stripped Functionality Logic Locking (SFLL-rem) [38], which is a fully auto-mated variant of SFLL-fault [13] and has all its security properties, and random logic locking (RLL) [10]. The attackers need to circumvent both layers of defense to break the overall defense.

The first logic locking scheme is designed to mitigate the threat of Satisfiability-based attacks (SAT attack); SFLL-rem uses point-functions to increase the effort of the SAT attack by eliminating one key in each iteration out of a possible 2^N keys, where N is the number of bits in the key. While SAT attack resistant, this defense is vulnerable to approximate attacks.

Next, to thwart approximate attacks, we implemented RLL [10] on top of SFLL-rem to increase the output corruption when an incorrect key is used (i.e., to produce incorrect functionality for a partially recovered (approximate) key). RLL randomly inserts key gates into the netlist which may or may not at times provide 50% output corruption.

The benchmark circuit statistics used for the challenge are shown in Table 4. To prepare the combinational locking benchmark circuits, we apply SFLL-rem + RLL to 7 different circuits comprising two variants per reference design (i.e., one provided with an oracle, the other without) derived from academic benchmark circuits (taken from ITC '99), and a locked Cortex-M0 microprocessor (as the bonus challenge). The circuits are named small, medium, and large based on their relative gate counts. The original circuits from the ITC '99 website are first modified by changing the gate types of randomly selected gates. These modified versions prevent known-circuit based functionality recovery attack.

Table 4. Statistics of the Benchmarks used for the Combinational Logic Locking Challenge

Competition benchmark	Academic Benchmark	# Inputs	# Outputs	# Gates
small	b20_C	522	512	20226
medium	b22_C	767	757	29951
large	b17_C	1452	1445	32326

We choose key sizes for each of the small, medium, large, and bonus circuits, according to their gate count. For the small, medium, and large circuits, we choose 40, 60, and 80 bits of security, respectively, in part to make the competition approachable. The bonus circuit, however, is locked with 128-bit security. Thus, small benchmarks are locked with 40-bit RLL key and 40-bit SFLL-rem key; medium benchmarks are locked with 60-bit RLL key and 60-bit SFLL-rem key; and large benchmarks are locked with 80-bit RLL key and 80-bit SFLL-rem key. The Bonus circuit was locked with 128-bit RLL key and 128-bit SFLL-rem key. The locking process involves the following steps:

1. Lock the modified benchfiles with SFLL-rem.
2. Lock the SFLL-rem locked benchfiles with RLL.
3. Remove traces of the original benchfile, e.g. Input and output port names.
4. Convert this file to and-inverter graph (AIG) format to rename all the internal net names as well.[7]

[7]Note: We removed all the traces of the benchmark identity to prevent attacks based on similarity. Tools such as Cadence LEC can provide a patch file to recover the original functionality. As this threat model is not realistic, we chose this approach even for the red teams who would have launched a functionality recovery attack.

To prepare an oracle for the locked circuits, we then convert the modified benchfile to executable to only permit access to I/Os and not the internal circuitry; this mimics access to a working chip obtained from the market.

After locking, we verify the unlocking of the design using an open source LCMP equivalence checker [16]; it requires the locked netlist, the unlocked or original netlist, and the key value. Since we do not provide the original netlist, but only the oracle with access to only PIs and POs, the red teams cannot use this equivalence checker tool to verify their key bits. Hence, they report it back to us and we provide them with the analysis of how many key bits recovered are correct.

8.2 Red-Teams for Combinational Logic

Bit-Flipping (Oracle-guided attack) the team from Northwestern University used their previously devised Bit-Flipping attack [39] on the combinational locking benchmarks.

Automatic Test Pattern Generation (ATPG)-based (Oracle-guided attack) the team from Carnegie Mellon University applied a sensitization-based attack [11] on the combinational locking benchmarks.

Hamming Distance-based Attack (Oracle-guided attack) the team from Texas A&M University proposed a Divide and Conquer approach to attack the combinational locking.

Automated SAT (Oracle-guided attack) the team from University of Texas at Dallas also attempted a divide-and-conquer-style approach by trying to divide the circuit into smaller logic cones, starting with primary outputs and identifying the fan-in logic.

Redundancy (Oracle-less attack) the team from University of California San Diego used their Redundancy attack [40] to attack the combinational locking benchmarks.

Unit Function Search Attack (Oracle-less attack) the team at Auburn University proposed a Unit Function Search Attack on the combinational locking benchmarks.

Sub-circuit SAT (Oracle-guided attack) the team from the Indian Institute of Technology Guwahati proposed a Sub-circuit SAT attack on the combinational locking benchmarks.

8.3 Hamming Distance-based Attack

As part of this competition, the Texas A&M team has developed a divide-and-conquer approach to attack the combinational locking. There are three steps to perform this attack, which are identifying the type of key inputs, stripping partial RLL key inputs, and launching the HD-based attack.

For the locked netlist, we first split it into several individual logic cones (ILCs). Then by counting the number of key bits in each cone, we can tell the ILC's protection type and the type of each key input. Since we get all ILCs and each cone's protection type, we can get the valid RLL key value. We select all RLL cones and merge them and get a netlist with fewer primary

outputs. Then let us run the SAT attack on this netlist, and it returns a valid RLL key with partial RLL key inputs.

By applying this valid RLL key value on the initially locked netlist, it becomes a simplified locked circuit. Now, let us pick one cone which is only protected by SFLL-fault on this simplified locked circuit, as shown in Algorithm 3. First, we extract FSC from the locked cone. Then, it needs to be transformed into PLA format using ABC. With this PLA being the input file of ESPRESSO, we can get the reduced PI table for FSC. The tool ABC and ESPRESSO are both logic synthesis tools. ABC can change the circuit format from benchmark to PLA. PLA is a cover of the circuit written as a PI table, and the tool ESPRESSO aims at simplifying the PI table and returning one reduced PI table. Then, we collect all candidate PIPs who's HDs no greater than threshold *d* to at least one PI in FSC's reduced PI table. Then we verify if there exists a PIP that results in different outputs of oracle and FSC. If we can find one verified PIP, then we use this PIP as the first input pattern into the SAT-based attack and grab the correct key from the attack. This key is also valid for other locked cones.

We present the combinational locking attack results in Table 5. In the combinational locking attacks, several teams were able to recover the RLL key bits in their entirety. **No team was able to recover the complete SFLL key bits.** This competition helped us gain further confidence in our defense as it helped validate our theoretical findings and expectations.

Table 5. Combinational Locking Attack Results

This table shows the number of key bits recovered by the teams that are verified as correct. Where the result is stated as (–) the team did not provide any results. Where the result is stated as (x / y), the team report y bits recovered, of which only x are verified as correct.

Team	Approach	Small (40+40)		Medium (60+60)		Large (80+80)		Bonus (128+128)		Attack Scenario
		RLL	SFLL	RLL	SFLL	RLL	SFLL	RLL	SFLL	
CMU	Key Sensitization	40/40	-	60/60	-	80/80	-			Oracle
TAMU	Hamming Distance-based Attack	30/30	-	50/50	-	72/72	-			Oracle
UTD	Automated Analysis + SAT	11/18	-	31/50	-	10/34	-			Oracle
IITG	Sub-circuit SAT	17/17	-	29/29	-	-	-			Oracle
UCSD	Redundancy-based	28/28	4/12	35/35	23/28	45/45	0/51	66/66	8/27	Oracle-less
Northwestern	Bit-flipping Attack	40/40	-	60/60	-	80/80	-			Oracle
Auburn	Topology guided attack	15/32	-	19/50	-	36/73	-	75/108	-	Oracle-less

9 CONCLUSION

This report summarizes the progress on the ECLIPSE project. In addition to developing security metrics and proofs, we apply the proposed SFLL technique on DARPA CEP and demonstrate it on an FPGA platform. The report also highlights our efforts towards protecting multiple outputs of circuit, unlocking circuits using machine learning, and developing a logic tool that implements different variants of logic locking. The report also summarizes the latest benchmarking and red-teaming efforts on our defense as part of NYU CSAW 2019 competition, which was the first logic locking contest. The key takeaway is that our proposed locking technique, SFLL-rem, withstands the test-of-time.

10 REFERENCES

[1] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.
[2] SEMI, "Innovation is at Risk Losses of up to $4 Billion Annually due to IP Infringement," 2008, [June 10, 2015]. [Online]. Available: www.semi.org/en/Issues/IntellectualProperty/ssLINK/P043785
[3] J. P. Skudlarek, T. Katsioulas, and M. Chen, "A Platform Solution for Secure Supply-Chain and Chip Life-Cycle Management," *Computer*, vol. 49, no. 8, pp. 28–34, 2016.
[4] S. Leef, "In Pursuit of Secure Silicon," http://textlab.io/doc/22959027/mr.-serge-leef--vp-new-ventures--mentor-graphics, 2017.
[5] ARM, "Physical Security Solutions," https://www.arm.com/products/security-on-arm/ physical-security-solutions, 2017.
[6] SypherMedia, "Circuit camouflage technology," https://www.insidesecure.com/Products/Silicon-IP/ Circuit-Camouflage-Technology, 2017.
[7] R. Jarvis and M. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," 2007, US Patent 7,195,931.
[8] A. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking Techniques for Intellectual Property Protection," in *IEEE/ACM Design Automation Conference*, 1998, pp. 776–781.
[9] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *USENIX Security Symposium*, 2007, pp. 291–306.
[10] J. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[11] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," in *IEEE/ACM Design Automation Conference*, 2012, pp. 83–89.
[12] M. Yasin, A. Sengupta, M. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *ACM/SIGSAC Conference on Computer & Communications Security*, 2017, pp. 1601–1618.
[13] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "ATPG-based cost-effective, secure logic locking," in *2018 IEEE 36th VLSI Test Symposium (VTS)*, 2018, pp. 1–6.
[14] P. Tuyls, G. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-Proof Hardware from Protective Coatings," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2006, pp. 369–383.
[15] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.
[16] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2015, pp. 137–143.
[17] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016, pp. 127–146.
[18] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT Attack Resistant Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2016, pp. 236–241.

[19] H. Zhou, R. Jiang, and S. Kong, "CYCSAT: Sat-based attack on cyclic logic encryptions," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.
[20] X. Xu, B. Shakya, M. Tehranipoor, and D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2017, pp. 189–210.
[21] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," *IEEE Asia and South Pacific Design Automation Conference*, pp. 342–347, 2016.
[22] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1601–1618.
[23] A. Sengupta, B. Mazumdar, M. Yasin, and O. Sinanoglu, "Logic locking with provable security against power analysis attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 766–778, 2019.
[24] E. Sperling, "Engineering Change Orders Revisited," https://semiengineering.com/engineering-change-orders-revisited, 2012.
[25] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.
[26] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (SFLL-HD)–unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778– 2786, 2019.
[27] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.
[28] A. S. of Defense for Research and Engineering, "Common Evaluation Platform," https://github.com/mit-ll/CEP, 2018.
[29] M. Yasin, T. Tekeste, H. Saleh, B. Mohammad, O. Sinanoglu, and M. Ismail, "Ultra-low power, secure IOT platform for predicting cardiovascular diseases," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2624–2637, 2017.
[30] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.
[31] D.T. Heide, *A Closer Look At Ethereum Signatures*, Feb, 2018. [Online]. Available: https: //hackernoon.com/a-closer-look-at-ethereum-signatures-5784c14abecc
[32] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.
[33] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE international symposium on circuits and systems*, vol. 3, 1989, pp. 1929–1934.
[34] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic Encryption: A Fault Analysis Perspective," in *Design, Automation and Test in Europe*, 2012, pp. 953–958.
[35] GitHub, "mor1kx - an OpenRISC processor IP core," https://github.com/openrisc/mor1kx.
[36] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization (WWC-4)*, 2001.
[37] NIST, "Post-quantum cryptography: Round 1 submissions," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography, 2018.

[38] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.
[39] Y. Shen, A. Rezaei, and H. Zhou, "Sat-based bit-flipping attack on logic encryptions," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 629–632.
[40] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 540–545.

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

ACRONYM	DESCRIPTION
AES	Advanced Encryption Standard
AIG	and-inverter graph
ALU	arithmetic/logic unit
ASIC	application-specific integrated circuit
ATPG	automatic test pattern generation
C_e	encrypted circuit
CE	checking engine
CEP	Common Evaluation Platform
CNF	conjunctive normal form
CPU	central processing unit
CRYPTO	cryptographic
CSAW	Cybersecurity Awareness Worldwide
DARPA	Defense Advanced Research Projects Agency
DfT	design for trust
DNN	deep neural network
DRC	design rule check
DSP	digital signal processing
ECLIPSE	Efficient Cross-Layered IP Protection SchemE
ECO	engineering change order
ER	error rate
FALL	functional analysis based logic locking
FI	fault impact
FIR	finite impulse response
FPGA	field-programmable gate array
GA	genetic algorithm
GPS	global positioning system
HD	hamming distance
IC	integrated circuit
ILC	individual logic cone
IO	input/output
IP	intellectual property
IPC	instruction per cycle
IRR	infinite impulse response
LLC	Logic Locking Contest
LVS	layout versus schematic
MCNC	Microelectronics Center of North Carolina
NIST	National Institute of Standards and Technology
NYU	New York University
OF	output fidelity
PIP	protected input patterns
PnR	place and route
PPA	power, performance, and area
PPT	probabilistic polynomial time

ACRONYM	DESCRIPTION
RAM	random access memory
RLL	random logic locking
RNG	random number generator
SAT	satisfiability
SFLL	stripped-functionality logic locking
SoC	system on a chip
SPR	special purpose register
SRAM	static random access memory
UART	universal asynchronous receiver-transmitter