ObfusGEM: Enhancing Processor Design Obfuscation Through Security-Aware On-Chip Memory and Data Path Design

Michael Zuzak
University of Maryland, College Park, MD
mzuzak@umd.edu

Ankur Srivastava
University of Maryland, College Park, MD
ankurs@umd.edu

ABSTRACT

A sizable body of work has identified the importance of architecture and application level security when using logic locking, a family of module level supply chain security techniques, to secure processor ICs. However, prior logic locking research proposes configuring logic locking using only module level considerations. To begin our work, we perform a systematic design space exploration of logic locking in modules throughout a processor IC. This exploration shows that locking with only module level considerations cannot guarantee architecture/application level security, regardless of the locking technique used. To remedy this, we propose a tool-driven security-aware approach to enhance the 2 most effective candidate locking locations, on-chip memory and data path. We show that through minor design modifications of the on-chip memory and data path architecture, one can exponentially improve the architecture/application level security of prior locking art with only a modest design overhead. Underlying our design space exploration and security-aware design approach is ObfusGEM, an open-source logic locking simulation framework released with this work to quantitatively evaluate the architectural effectiveness of logic locking in custom processor architecture configurations.

CCS CONCEPTS

• **Computer systems organization** → *Processors and memory architectures*; • **Security and privacy** → *Hardware security implementation*; Embedded systems security.

KEYWORDS

ObfusGEM, Processor Design Obfuscation, On-Chip Memory Design, Logic Locking, Untrusted Foundry, IP Piracy

ACM Reference Format:
Michael Zuzak and Ankur Srivastava. 2020. ObfusGEM: Enhancing Processor Design Obfuscation Through Security-Aware On-Chip Memory and Data Path Design. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3422575.3422798

MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA

ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422798

1 INTRODUCTION

The increasing cost of integrated circuit (IC) fabrication has driven chip design companies to adopt a *fabless* model. Fabless companies export their IC fabrication supply chain to an unaffiliated third party, known as an *untrusted foundry*. By doing so, they assume significant risk as untrusted foundries can pirate, counterfeit, or overproduce ICs [5, 28].

To mitigate these risks, researchers developed logic locking, a family of module level hardware security techniques that render IC functionality dependent on a secret key [9, 12, 17, 18, 20, 21, 24–26, 34, 36, 38, 39, 44, 45]. Without the correct secret key, logic locking deterministically injects multi-bit errors at the output of any locked module. Given a sufficient error injection rate (i.e. the number of inputs that produce corrupt output compared to the size of the input space), a locked IC becomes unusable. Therefore, by withholding the key from unauthorized users (e.g. an untrusted foundry), logic locking can prevent unauthorized IC use. Fundamentally, the goal of logic locking is two-fold: 1) **Error Severity**: injecting sufficient error to render an IC unusable for any wrong key and 2) **Attack Resilience**: resisting attacks against it.

Despite logic locking being proposed and implemented as a module level locking technique, it cannot be evaluated at this level. This is the case because to ensure error severity, the first goal of logic locking, module level error injection alone is insufficient. For error severity, module level error must critically impact the application being run on an IC. Therefore, any effective locking technique must induce sufficient module level error to thwart application level IC functionality. This has been noted by a substantial body of logic locking research which argues that locking cannot achieve security without considering an IC's unique architecture [6, 12, 15, 16, 23, 45]. However, despite the wide array of research which recognizes the importance of architectural considerations, there has been no systematic evaluation of locking or proposed design approach to achieve security at this level.

This motivates our work. The community has clearly shown that achieving security with logic locking is necessarily an architectural problem. Therefore, it is clear that architectural design decisions will impact the security of logic locking. In this work, we aim to both explore and quantify the ramifications of logic locking at this level. To do so, we narrow our focus to processor design obfuscation, one of the most commonly proposed obfuscation targets [6, 8, 12, 23, 40, 41, 45]. Based on a design space exploration of 2 processor ICs, we show that logic locking is severely limited when viewed at this level. As a result, we explore the possibility of a security-aware architecture design approach to enhance logic locking techniques. To this end, we direct our attention to the most commonly proposed candidate modules for logic locking: 1) the on-chip memory, such as cache controllers [6] or SRAM memory [45], and 2) the data path,

such as ALUs [12] or alternative compute units [15, 23]. Within these candidate locations, we propose and evaluate a quantitative, tool-driven design approach for both on-chip memory and data path architectures to achieve strong, application level supply chain security guarantees with logic locking.

1.1 Contributions

In this work, we present a first-of-its-kind quantitative design space exploration of logic locking at the application level using an x86 and ARM A53 processor. We show that cutting edge locking techniques are incapable of simultaneously achieving both error severity and attack resilience in either of these ICs when applied blindly at the module level (as is done in prior work [9, 17, 18, 20, 21, 25, 26, 34, 36, 38, 39]). To overcome these limitations, we explore a security-aware approach to design the on-chip memory or data path architecture of processor ICs. To evaluate this approach, we redesign components of both the on-chip memory and data path of our evaluated x86 and ARM A53 core. Each security-aware redesign is shown to enable existing locking techniques to achieve exponentially stronger error severity/attack resilience guarantees. Hence, with our security-aware design approach, a designer can construct memory and data path architectures capable of amplifying the application level supply chain security of existing logic locking art.

Underlying this work is a unique quantitative view of application level supply chain security that is absent from prior work. To provide this, we have developed ObfusGEM, a comprehensive logic locking simulation framework based on the GEM5 simulator [3]. Alongside this work, we have released ObfusGEM (available at: "*https://github.com/mzuzak/ObfusGEM*") as an open-source tool to not only enable our tool-driven security-aware on-chip memory and data path design approach, but also to aide in both the design and evaluation of logic locking techniques within these components at the application level. Our contributions are as follows:

(1) We show that logic locking cannot secure processors when applied blindly at the module level, regardless of configuration, through a design space exploration of an x86 and ARM A53 core.
(2) We propose a quantitative, security-aware design approach for processor on-chip memory and data path components capable of exponentially enhancing the application level supply chain security of a logic locked processor.
(3) Using our security-aware design approach, we demonstrate a variety of security-aware on-chip memory architectures that allow our x86 and ARM A53 core to achieve strong supply chain security with only a modest increase in overhead.
(4) We release ObfusGEM, an open-source logic locking simulation framework which quantitatively evaluates the application level security of logic locked processors. ObfusGEM is used to both design and evaluate the secure on-chip memory and data path architectures in our testbed ICs.

2 PRELIMINARIES AND PRIOR WORK

2.1 Logic Locking

Logic locking is a diverse family of combinational hardware security techniques aimed at preventing unauthorized IC use, such as piracy, counterfeiting, overproduction, and reverse engineering, by untrusted elements in an IC's fabrication supply chain [9, 12, 17, 18, 20, 21, 24–26, 34, 36, 38, 39, 44, 45]. It is characterized by the introduction of accessory logic into modules within an IC. This accessory logic is driven by both internal logic signals and an added set of primary inputs, known as key inputs, which are driven by a tamper-proof memory included in the design. Following fabrication, this tamper-proof memory is loaded with a user-defined value known as the *secret key* of the logic locking construction.

Through this construction, the functionality of a locked module becomes dependent on this secret key. This means that an IC will exhibit incorrect functionality (i.e. deterministic multi-bit error injections) whenever some key other than the correct secret key is applied. By sufficiently corrupting IC functionality, a locked IC becomes unusable. Hence, by withholding the secret key from any unauthorized user, logic locking will render the IC unusable for these entities, thereby restricting unauthorized use. In order to be successful, logic locking must achieve 2 primary goals: 1) **Error Severity**: injecting sufficient error to render an IC unusable for any wrong key and 2) **Attack Resilience**: resisting attacks against it. Error severity is generally related to the *wrong key error rate* of the logic locking construction, defined as the average number of inputs that produce errant outputs for a wrong key compared to the total possible input combinations. Attack resilience is usually defined as the time or number of attack iterations required to recover the secret key for a given locking construction. A generic example of a logic locked module is shown in Figure 1.

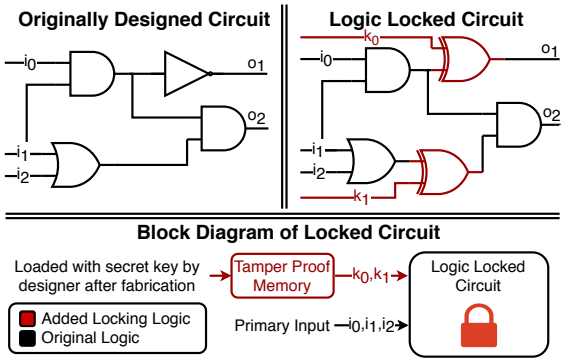

Figure 1: Configuration of a logic locked module.

2.2 State-of-the-Art Logic Locking Techniques

Initially, proposed logic locking techniques aimed to maximize error rate at the output of wrongly keyed modules [17, 21, 38]. Doing so ensured strong error severity guarantees. However, a Boolean satisfiability attack against logic locking, known as a SAT attack, was proposed to quickly unlock this early art [33]. The introduction of the SAT attack prompted the development of SAT resilient locking techniques [34, 36]. Both of these techniques achieved provable SAT attack resilience by carefully limiting the error injection rate of their locking constructions. Essentially, [34, 36] were the first techniques to sacrifice error severity to improve attack resilience. This trade-off is an underlying theme of recent logic locking research.

Currently, there exists a diverse array of state-of-the-art logic locking techniques [9, 12, 18, 20, 24–26, 39, 44, 45]. While these techniques differ in their specific constructions, they each place error severity and attack resilience into direct contention. Among these proposed techniques, SFLL [24, 25, 39] and SAS [12] define uniquely tunable constructions which enable locking configurations with any desired wrong key error rate. This tunability led the authors to provably quantify a direct, inverse relationship between the wrong key error rate (error severity) of their locking constructions and the SAT attack iterations necessary to unlock them (attack resilience). Later work generalized these derived trade-offs to apply to *any* combinational locking construction [12, 42–44]. We emphasize the importance of this trade-off for secure locking design. **To achieve security, a designer must balance error severity and attack resilience, ensuring that both are achieved regardless of locking construction.**

2.3 Architectural Security of Logic Locking

Logic locking is implemented with gate level locking structures that inject error at the module level. To ensure error severity, module level error injection alone is insufficient. For error severity, module level error must critically impact the application being run on an IC. Therefore, any effective locking technique must induce sufficient module level error to thwart architecture level IC functionality.

A substantial body of prior work on logic locking has highlighted the importance of these architectural considerations to secure ICs [6, 12, 15, 16, 23, 45]. However, despite the wide array of research that recognizes the importance of IC architecture for supply chain security, there has been no systematic evaluation of logic locking or design approach to achieve security at this level. For example, the work in [23] secures a large, multi-million gate chip with a focus on system-level impact, however, it performs only qualitative analysis to do so. Alternatively, [15, 16] merely recognize the need for architectural considerations, but do not delve into the specifics of locking at this level. On the other hand, [6, 12, 45] provide a limited evaluation of their proposed techniques in a specific architecture, but do not provide any methods to design or verify architecturally secure locking configurations. Therefore, while these works provide substantial evidence that architectural potency is necessary, they fall short of providing a quantitative understanding of architectural security or a method to reliably achieve it.

Throughout these works exploring the architectural efficacy of logic locking [6, 12, 15, 23, 40, 41, 45], 2 primary areas of processor architecture have been suggested as strong candidates for effective locking. These components are 1) on-chip memory, such as cache controllers [6] or SRAM memory [45], and 2) data path components, such as ALUs [12] or alternative compute units [15, 23]. We begin our work with a quantitative exploration of these candidate components. Following this, we propose and evaluate an architectural design approach for on-chip memory and data path components to amplify the supply chain security achievable by logic locking.

2.4 Attacker Model

In this work, we consider a SAT-capable adversary, common in recent research [9, 12, 18, 20, 24–26, 34, 36, 39, 44, 45], who takes some strategy using:

(1) A locked combinational netlist of an IC, which can be obtained by reverse engineering provided GDSII files.
(2) A correctly-keyed, black-box oracle IC. This can be obtained through either IC testing facilities or the open market. The attacker can query this oracle IC with an input to determine the correct corresponding output.

The goal of the attacker (e.g. an untrusted foundry) is to create an IC sufficient for sale or IP piracy. Because ICs are designed to run a set of specific applications, any successful defense strategy must ensure critical failures in these workloads for wrong keys. Therefore, the attacker's goal is to obtain an IC capable of running these specific applications. In this work, we quantify this with application failure rate and mean time to failure. A higher failure rate or shorter time to failure indicates more secure locking.

3 OBFUSGEM SIMULATION FRAMEWORK

We begin by introducing the ObfusGEM Simulation Framework[1] that we have developed for this work. ObfusGEM is an open-source tool-set which allows users to apply logic locking techniques to custom processor netlists, attack them with cutting edge attack methodologies, and then simulate custom workloads on the resulting ICs in a precise, cycle-accurate fashion. By observing any locking induced workload failures in these ICs, the application level security of logic locking can be quantified. Therefore, ObfusGEM enables the quantitative exploration of logic locking at the application level. As noted in Section 2.3, a significant body of research has identified the importance of architecture/application level considerations for logic locking. ObfusGEM serves as the first systematic way to explore these considerations, regardless of locking technique. Throughout this work, we will utilize the ObfusGEM framework to enable us to both design and evaluate secure on-chip memory and data path architectures.

3.1 ObfusGEM Supported Attacker Models

ObfusGEM is attacker model agnostic. This allows the user to evaluate any attacker model they consider realistic. Also, because ObfusGEM operates on real netlists of locked ICs, it can utilize any attack methodology or locking approach without modification.

3.2 Overview of the ObfusGEM Framework

To introduce ObfusGEM, we start with a brief overview. A block diagram of the process to quantify locking at the application level is in Figure 2. We discuss each step below.

(1) A netlist is selected and logic locked. Any number or combination of modules can be locked within an IC.
(2) Any attack (SAT/SMT [1, 33], structural [32, 35], removal [37], etc.) can be applied to the IC to locate a key. We note that a real netlist is used so any proposed attack can be applied without modification. This allows the effectiveness of specific attacks against logic locking to be quantified at the application level.
(3) The attacker's key is applied to each locked module and a fault analysis locates any corrupted input minterms within

[1] An open-source copy of the ObfusGEM Simulation Framework can be found at: "*https://github.com/mzuzak/ObfusGEM*".

any locked module. This essentially creates a truth table defining the functionality of each locked module.

(4) The on-chip memory/processor architecture, intended IC workloads, locking configuration, and any corrupted minterms for each module are specified in configuration files.

(5) The ObfusGEM simulator, described in Section 3.3, uses these configuration files to perform cycle-accurate simulations of a locked and an unlocked oracle processor running specified workloads. By tracking the divergence of these cores, the effects of locking are measured.

(6) Steps 2-5 are repeated in a Monte Carlo fashion, randomizing parameters including the simulated application, applied locking key, or adversarial attack methodology.

By aggregating the results of many Monte Carlo simulations, the application failure rate and the mean time to failure can be calculated for a locked processor. Additionally, because ObfusGEM is based on the GEM5 simulator [3], performance and power analysis details can also be obtained for the locked IC. Given sufficient Monte Carlo trials, these data points quantify the usability and overhead of a locked processor after attacked by an untrusted foundry. Because the attacker's goal is to produce an IC sufficient for open market sale or piracy, the more usable an IC, the more successful the attacker. Therefore, the data produced by ObfusGEM directly quantifies the architectural effectiveness of logic locking. We use ObfusGEM for the remainder of this work, first to systematically explore the logic locking design space for both on-chip memory and data path locking and then to enable the tool-driven security-aware design approach that we introduce for on-chip memory and data path design in Section 6.

3.3 Simulator Overview

Now, we discuss the simulator block of the ObfusGEM framework, displayed as step 5 of Figure 2. To construct this simulator, we relied upon stochastic fault injection research by the error resilience community [11, 13]. Building upon the outline laid out in [13], we implemented our own custom fault injection simulator with one critical difference: faults injected by logic locking are deterministic, not stochastic in nature. This means that locked primary inputs must always inject error when applied as input to a locked module. This mitigates the impact of error detection and many other error recovery procedures relied upon in the error resilience community.

Specifically, we perform application level simulations of a locked and an unlocked instance of an identical processor in GEM5 [3]. In the locked simulation instance, corrupted module output (located via a fault analysis of the locked netlist) are mapped to a deterministic error state. As errors are injected, their severity is classified by comparing the processor state of the locked and unlocked GEM5 simulation over a variable number of clock cycles. A divergence of the locked from the unlocked core which is not corrected or rendered latent in the variable clock cycle window is classified as an unrecoverable/critical application error. If the locked processor returns to a state exhibited by the unlocked processor at any point after the fault injection, we re-synchronize each processor trace (to reset the variable timing window and enable new faults to be analyzed) and consider the fault to be masked and therefore architecturally irrelevant.

3.4 Relationship to Prior Art

The concept of error resilience simulation is not new. It has been heavily studied by the research community, primarily focusing on either radiation-induced soft errors or manufacturing defect related errors. To facilitate soft error research, a series of error resilience simulators have been developed [7, 14]. While the ObfusGEM simulator relies heavily on the lessons learned by these tools, they are not interchangeable. This is due to the difference between the probabilistic, on-chip memory errors caused by cosmic rays which are modeled by soft error simulators and the deterministic, gate level errors caused by logic locking which are modeled by ObfusGEM. To effectively model soft errors, these simulators "fast-forward" through a random number of processor clock cycles, maintaining only the overall processor state. In the extremely unlikely probability of an error, the processor execution is halted, a small number of bit-flips are injected into on-chip memory, and a detailed simulation mode proceeds until the impact of the injected error is determined.

As described in Section 3.3, the functionality of ObfusGEM differs significantly. Because the errors injected by ObfusGEM are deterministic, rather than probabilistic, we are unable to use a "fast-forward" mode. Deterministic error injection requires that the current state of each module is maintained at all times and compared to any corrupted I/O during each clock cycle. Prior simulators lack the detailed tracking and comparison mechanisms necessary for deterministic error injection. Additionally, even if comparison logic was added, the detailed simulation mode would be necessary at all times. This would yield prohibitively slow performance.

Alternatively, there exists fault simulators focused on modeling errors related to manufacturing defects or degradation [4, 19]. While

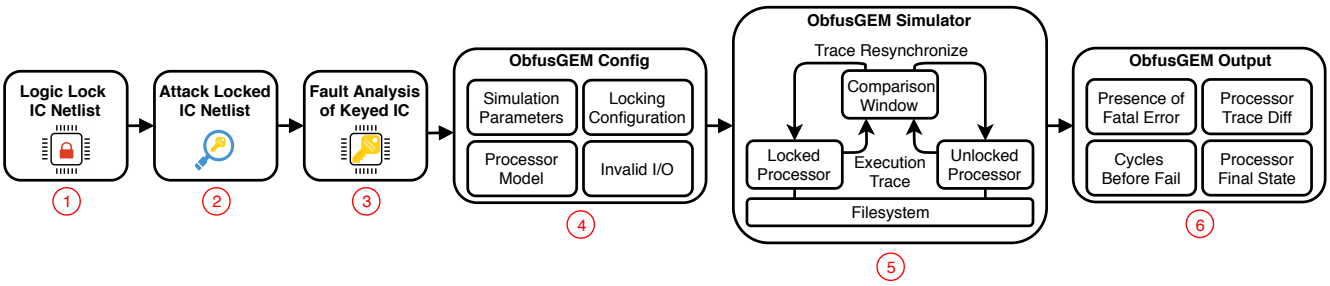

Figure 2: Block diagram of the ObfusGEM simulation framework.

these simulators do explore deterministic error injections, they rely on quite detailed netlist modeling to do so. In general, fault simulators utilize full-scale Verilog simulations of the IC under test. These detailed models make the simulation of operating systems or workloads, as is necessary for logic locking research, prohibitively slow. ObfusGEM, while based on a netlist representation, simulates only a functional model of the IC using GEM5. This not only reduces execution time, but also enables the user to easily model and modify IC architectures. By leveraging the customizability of the GEM5 simulator in ObfusGEM, one can easily explore the effect of on-chip memory design on supply chain security as is done in Section 6.

4 EXPLORING THE DESIGN SPACE OF PROCESSOR DESIGN OBFUSCATION

We continue by using ObfusGEM to perform a quantitative design space exploration of logic locking at the application level. To do so, we will model SFLL-Fault [24, 25, 39], as it is the most prevalent locking technique which remains unbroken[2]. As later discussed in Section 4.3, despite using SFLL-Fault for this exploration, our results are applicable not just to SFLL-Fault, but general to all logic locking, regardless of technique. Therefore, this experiment serves as an exploration of all combinational logic locking.

Specifically, we will incorporate a variety of locking configurations which sweep over the error injection rate of locking and quantify the corresponding architectural effectiveness (error severity) of each locking construction. As noted in Section 1, SAT attack resilience (attack resilience) is inversely related to the error injection rate of locking. Therefore, we can calculate the corresponding SAT resilience of each locking configuration based on error injection rate using results from [12, 42–44]. We note that this relationship between error injection rate and SAT attack resilience exists underlying all logic locking techniques, not just SFLL-Fault. Therefore, this experiment aims to identify locking configurations capable of inducing critical application failures (error severity) while maintaining SAT attack resilience within our processor testbeds. In other words, this experiment identifies secure locking constructions at the application level.

4.1 Experimental Methodology

4.1.1 Testbed Processors: An x86 (out-of-order, embedded core) and an ARM A53 processor were used. Note that due to the proprietary nature of both of these cores, we were forced to use software, rather than hardware, models. These models were functional representations developed within the GEM5 simulator. However, all methods and data presented could be performed equivalently on a hardware model of the core.

4.1.2 Locking Configuration: SFLL-Fault[25] based locking configurations sweeping over error injection rate were applied to each processor. To do so, SFLL-Fault was configured to lock a single, randomly selected input cube of varying length. By scaling the length of the locked input cube, a varying number of minterms could be corrupted, thereby scaling the error injection rate. Each of the 14 modules (see Figure 3) were locked and evaluated independently (i.e. one at a time). Note that the candidate locked modules were selected from 3 categories: 1) on-chip memory, 2) data path, and 3) control path. As we have noted before, on-chip memory [6, 45] and data path locking [12, 15, 23] have been the most commonly suggested locking candidates in prior literature. Therefore, we focused our attention on modules within the on-chip memory and data path.

4.1.3 Feasibility of Locking Configuration: We note that this locking configuration is consistent with state-of-the-art logic locking research. We make several observations regarding this:

(1) Recent work on architectural locking considers locking only a single module which contains the critical IP to be protected [6, 23, 25, 26, 39]. This is consistent with our decision to independently lock each module.
(2) Cutting edge locking, such as [6, 23, 25, 26, 34, 36, 39, 44], propose constructions which distribute error throughout a locked module's input space. This is consistent with our random cube selection approach. By doing so, we both maximize the SAT attack resilience of the locking construction and prevent cryptographic information leakage in the case that an adversary has knowledge of a module's input space (e.g. knowledge of "*protected input cubes*" for SFLL allows an adversary to recover the secret key in linear time [39]).
(3) Cutting edge techniques, such as [23, 24, 34, 36, 39], propose integrating their low error locking constructions alongside a high error locking technique, such as [38]. Taking this so-called "*compound*" approach allows a designer to achieve both high error severity and SAT attack resilience simultaneously. However, recent research has shown that high error locking techniques in compound locking constructions can be easily removed, leaving only low error locking within the module [27, 29–31]. This is consistent with our choice not to pair SFLL-Fault with a high error locking technique, as it could be easily removed.

4.1.4 ObfusGEM Configuration: Using the ObfusGEM simulator, 120 Monte Carlo simulations were performed for each locking configuration in each processor. A locking configuration consists of a locking location and an associated error injection rate (e.g. the x86 core adder locked by an SFLL-Fault configuration with an error rate of 0.01%). For each Monte Carlo iteration, a wrong key was randomly selected and 1 of 3 benchmarks from the PARSEC benchmark suite[3] [2] were simulated on the core. Because the error rate of SFLL-Fault is uniformly distributed across all wrong keys (i.e. each wrong key has the same error rate), each random key selection produces a locking configuration with an identical error rate and different locking-corrupted minterms. Note that this is the weakest possible attacker. The attacker randomly selects a key with no attempt to minimize the error injection rate or intuit the correct key. In total, this makes 50,400 Monte Carlo trials[4].

[2]Several works have shown that structural traces unique to SFLL can be exploited to recover the secret key [32, 35]. While these approaches have not successfully unlocked SFLL-Fault, they have unlocked earlier SFLL-based techniques, such as SFLL-HD, indicating some limitations of the SFLL family.

[3]PARSEC benchmarks [2] are designed to be a cross-section of common processor workloads, therefore, they serve as a good measurement of a locked processor's ability to do useful work at the application level.

[4]50,400 total trials = 2 ICs * 14 locking locations * 15 SFLL-Fault error rates * 120 Monte Carlo trials

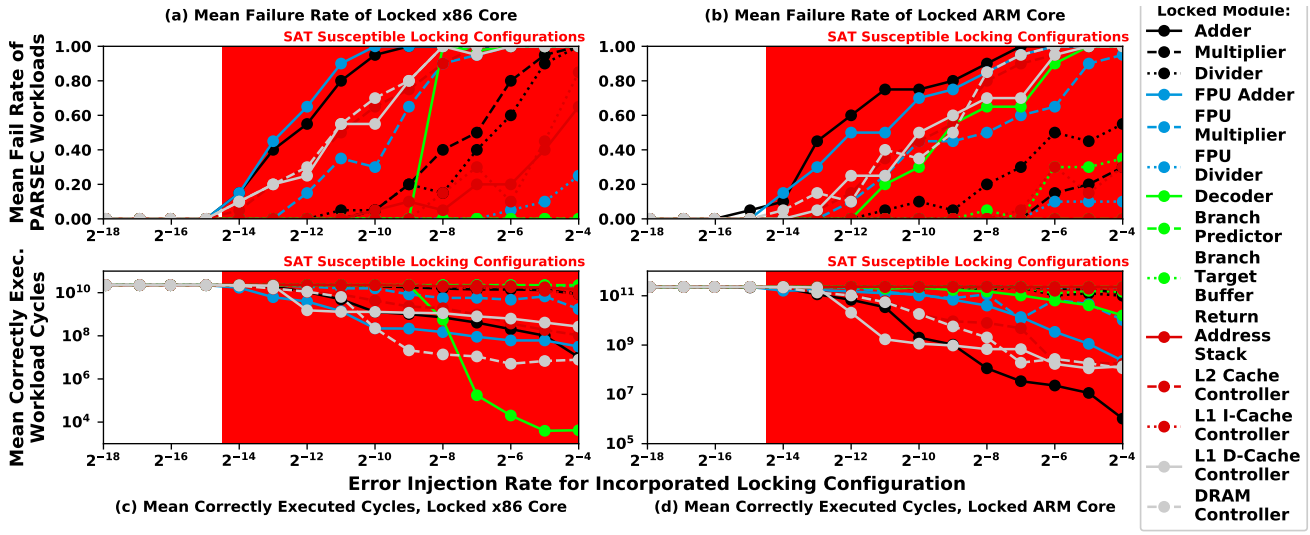

Figure 3: ObfusGEM results quantifying the application level security of locking in an x86 and ARM A53 core.

4.1.5 Monte Carlo Simulation Count: We selected the number of Monte Carlo trials performed for each locking configuration empirically. To do so, we performed 1,500 Monte Carlo trials for 4 error rates (2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}) applied to 4 locking locations (FPU Adder, Multiplier, L1 D-Cache Controller, Decoder) respectively. We considered the application failure rate after 1,500 iterations to be the *true* application failure rate of each locking construction. Based on this, we located the number of Monte Carlo trials in which each locking configuration converged to an error rate within ±5% of the *true* application failure rate. This was 120 Monte Carlo trials per configuration.

4.2 Quantifying SAT Attack Resilience

The work in [12, 42–44] defines an inverse mathematical relationship between the error injection rate of logic locking and the average number of SAT iterations necessary to unlock it. However, to be successful, a SAT attack must locate the correct key in a reasonable time. Because SAT attack runtime is both netlist and attack formulation dependent, the expected number of SAT attack iterations alone does not provide sufficient context to accurately characterize SAT susceptibility. Fortunately, the results presented in the work on SFLL-Fault [25, 39] provide a strong intuition for SAT runtime.

Yasin et al. provided an empirical analysis of SAT attack effectiveness against varying error rate SFLL constructions in a series of benchmark circuits. For their experiment, they incrementally lowered the error injection rate of SFLL and evaluated the corresponding SAT attack runtime. The lowest error injection rate SFLL configuration which could be successfully SAT attacked within 48 hours required 2^{14} SAT iterations. Therefore, we define any locking configuration unlocked in $\leq 2^{14}$ iterations as SAT susceptible.

4.3 Analysis of Design Space Exploration

We have aggregated the results of the experiment described in Section 4.1 in Figure 3. Within the figure, the region constituting SAT susceptible locking configurations have been shaded in red for clarity. Fundamentally, the goal of our design space exploration was to locate an architecturally secure locking configuration, defined as a locking configuration which simultaneously 1) induces critical application failures for any wrong key (error severity) and 2) maintains SAT attack resilience. In the figure, the first goal of logic locking (error severity) is quantified by the PARSEC benchmark failure rate. If a locking configuration with a given error rate induces a high failure rate for PARSEC benchmarks, the configuration has successfully derailed application functionality. The second goal of logic locking (maintaining SAT resilience) is quantified by the red-shaded SAT susceptible region. If a given error rate locking configuration resides outside of this region, it is deemed SAT resilient. Therefore, based on our results, there does not exist an SFLL-Fault configuration capable of simultaneously achieving both goals.

While each netlist was only locked with SFLL-Fault, this same trade-off between error severity and SAT resilience exists underlying *every* locking technique [12, 42–44]. Therefore, because all logic locking techniques restrict unauthorized use with the same fundamental functionality, namely deterministically corrupting the output corresponding to some portion of the input space, these results can be generalized to logic locking as a whole. Any alternative logic locking technique configured with a given, randomly distributed error injection rate can be expected to achieve a similar error severity to that of SFLL-Fault. Additionally, given the generalized nature of the results derived in [12, 42–44], this error severity can be expected to correspond to a similar number of SAT attack iterations necessary to unlock the locking construction, hence a similar SAT attack resilience. Therefore, the results of this design space exploration are not a limited example in which only SFLL-Fault was insecure when viewed at the application level, but a demonstration of the underlying limitations of logic locking when viewed at the application level.

This result is quite alarming. **Fundamentally, it indicates that state-of-the-art locking applied without application level considerations (as proposed by most cutting edge art [9, 17, 18,**

20, 21, 25, 26, 34, 36, 38, 39]) is inadequate to thwart an untrusted foundry attacker, regardless of locking location or configuration. To further exacerbate this result, we note that the weakest possible attacker model was utilized. The untrusted foundry simply selected a random wrong key with no attempt to apply cutting edge attacks or to minimize error within the locked IC. Even given this extremely weak attacker model, state-of-the-art logic locking was unable to achieve application level security within either IC, constituting a massive security risk.

Finally, we note that while no locking configuration was capable of achieving both error severity and attack resilience, the most successful locking locations resided in the on-chip memory (L1/L2 cache controller, memory controller) and the data path (adder, FPU adder) of each processor. This is rather unsurprising given the emphasis placed on on-chip memory [6, 45] and data path [12, 15, 23] locking in prior work. Therefore, based on these results, we narrow our focus to the on-chip memory and data path of the processor. With this focus, we aim to identify the reasons limiting the effectiveness of logic locking in these modules. Based on these limiting factors, we attempt to identify design decisions that can be made to amplify the supply chain security of on-chip memory and processor data path components. Ultimately, we hope to provide a quantitative, tool-driven approach for secure on-chip memory and data path design by leveraging ObfusGEM tooling.

5 FACTORS LIMITING SECURITY

As we move towards mitigating the identified security risks through secure on-chip memory and data path design, we first must understand the underlying causes. Based on our experiments, 2 primary factors limit the architectural effectiveness of locking, namely 1) module input space non-uniformity and 2) processor error resilience. We discuss each in turn.

5.1 Factors Limiting the Efficacy of Locking

5.1.1 Input Space Non-Uniformity: Within a processor, inputs to each module are generally heavily skewed towards a small subset of the input-space. This is due to the tendency of processors to repeatedly access the same set of data and resources, a heavily studied phenomenon referred to as the *principle of locality*. Additionally, because data and resource utilization is dictated by the application being run on a processor, the input-space of each module is not only skewed, but also application-specific. This means that locking applied independently of IC architecture/application (as is generally proposed) cannot account for the architecture/application-dependent input space of a module. Finally, as we have noted, the portion of the input space that is corrupted by logic locking must be limited to ensure SAT resilience [12, 42–44]. Therefore, it is unlikely that any of the tiny set of application-agnostic inputs corrupted by logic locking will ever actually occur within the skewed, application-dependent set of inputs applied to the locked module. This makes the likelihood of locking induced errors negligible, greatly limiting the efficacy of locking configured independently of IC architecture.

To empirically support the above claim, we have used ObfusGEM to characterize the input-space of an adder within our x86 core running 9 benchmarks from the PARSEC benchmark suite. Several observations from this experiment are below.

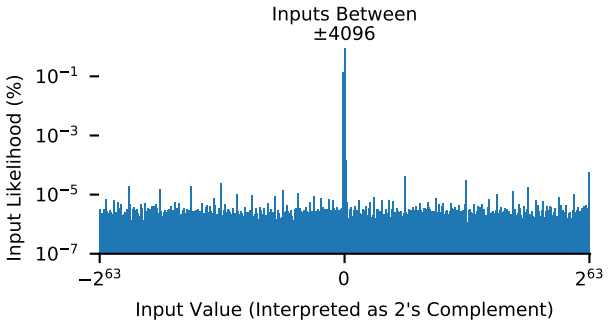

Figure 4: X86 core adder input utilization for Blackscholes.

(1) Workloads used 10^{-34}% to 10^{-31}% of the input space.
(2) A histogram of input utilization for the Blackscholes benchmark is in Figure 4. This input-space is skewed between ±4096, with > 95% of inputs in this range.
(3) Only ~13,000 inputs are shared between each workload.

These results support the claim that a small/skewed subset of a module's input space is used by the application. Additionally, the relatively small input overlap between benchmarks indicates that a module's input-space is quite application-dependent as well. Finally, we reiterate that to achieve SAT resilience, the number input minterms corrupted by locking must be severely limited. By combining these results with the 10^{-34}% to 10^{-31}% input space utilization identified empirically, we confirm that the probability of a corrupted minterm actually being applied to a locked module is indeed nearly negligible. Therefore, effective locking must account for both IC architecture and the applications being run on an IC to achieve security.

5.1.2 Processor Error Resilience: Substantial computer architecture research has shown that many ICs, especially processors, mask an overwhelming majority of module level errors when viewed architecturally [13, 22]. For example, the work in [22] showed that over 97% of random, radiation-induced soft errors vanished within a tested IBM POWER6 core. On top of the architectural error resilience of ICs, most common applications have been shown to be error resilient as well [7, 11]. For example, common media and AI benchmarks mask as much as 46% of module level errors injected when considering application output [11]. This means that even when logic locking induced errors occur, there still exists a sizable probability that this error will be simply masked and rendered architecturally irrelevant. Therefore, in addition to ensuring module level error is injected, effective locking must also ensure that injected error will derail IC functionality.

6 SECURITY-AWARE ARCHITECTURE DESIGN APPROACH

As noted in Section 1, all logic locking art is forced into a trade-off between error injection rate and SAT attack susceptibility [12, 42–44]. The experimental results aggregated in Figure 3 show that logic locking configurations with a wrong key error rate adequate for architectural security (error severity) are inherently SAT susceptible. On the other hand, each SAT resilient locking configuration was

unable to derail processor functionality. Because these bounds are theoretically derived [12, 42–44], minor tweaks and redesigns of similar art are unlikely to produce a viable solution. Therefore, instead of altering the locking techniques, we must explore other methods to improve the application level security of locking.

To this end, we look for ways to mitigate the factors limiting application level security. Given that these identified factors exist outside of the locked module itself, at the architecture level, we look to an IC's architecture to overcome the identified limitations. We explore the possibility of so-called *security-aware* architecture design to improve the security of logic locking, regardless of technique. Fundamentally, we are suggesting that architecture design decisions should consider not only traditional parameters (i.e. area, delay, power, performance, etc.), but also supply chain security. To this end, we propose and evaluate security-aware architecture design, a tool-driven approach, based upon the ObfusGEM simulator, to identify and evaluate minor architecture design modifications capable of improving the application level impact of locking. In particular, we look to the most effective locking candidates identified in Figure 3, namely the on-chip memory and data path, to improve supply chain security. For these components, we will attempt to quantify the goals of a design approach that favors supply chain security in these components. Then, we will apply this design approach to redesign candidate locking locations in each IC capable of exponentially enhancing supply chain security.

6.1 Security-Aware Architecture Approach

To both demonstrate and evaluate a security-aware architecture design approach, we implement it in our x86 and ARM A53 processor ICs. To this end, we proceed as follows.

(1) **Identify Factors Limiting Logic Locking:** Using the cycle-accurate, architectural data provided by ObfusGEM, we identify the factors limiting architectural security. This is highlighted in Section 5 for our tested ICs.
(2) **Identify Candidate Design Modifications:** Minor architectural design modifications for the on-chip memory and data path that are capable of mitigating any limiting factors must be identified. We perform this in Section 6.2 for our x86 and ARM A53 core.
(3) **Implement and Evaluate Security-Aware Changes:** Using the quantitative, architectural lens provided by ObfusGEM, the efficacy of each identified change must be quantified and tuned, ensuring that sufficient application level security guarantees are achieved within the IC. We perform this in Section 6.3 for our tested ICs.

Fundamentally, this approach relies on the quantitative lens provided by the ObfusGEM simulator to both identify and apply these security-driven modifications. Throughout the remainder of this work, we demonstrate how a security-focused approach to on-chip memory and data path design can exponentially improve security. This enables our previously insecure ICs to achieve strong application level security guarantees with locking.

6.2 Identifying Candidate Design Modifications

As shown in Section 5.1, the limitations of locking can be partially attributed to 1) input space non-uniformity and 2) processor error resilience. This means that any design decision which 1) increases the number of uses of (i.e. utilization) or unique input minterms applied to (i.e. diversity) a locked module or 2) amplifies the impact of locking induced errors at the application level will enhance security. We continue by identifying a series of architectural changes which achieve either of these security-focused design goals.

6.2.1 Increasing Locked Module Input Utilization/Diversity: When input utilization/diversity is increased, a larger percentage of a locked module's input space is used. Increased input space utilization increases the likelihood that a locked minterm will be applied to the module, increasing the likelihood of a locking induced fault injection. Many architectural decisions can achieve this:

- For cache controller locking, increasing cache associativity increases both the length and diversity of cache tags (increases input diversity of locked module).
- For memory controller locking, utilizing a write through (rather than write back) cache will increase write frequency, increasing memory controller use (increase utilization). However, we note that this change has a large number of side effects for an IC, likely making it unreasonable in practice.
- For functional unit (FU) locking, smart scheduler design can either favor locked FUs or ensure that corrupted I/O pairs are likely to be scheduled to locked FUs (increase utilization).
- The number of FUs (i.e. adder, FPU, etc.) can be increased and locked with different locking configurations corrupting different inputs (increases diversity of locked inputs).

6.2.2 Amplifying the Impact of Locking: By amplifying the impact of locking induced errors, locking is more likely to overcome architecture/application error resilience. Many design choices can achieve this, for example:

- Locking to ensure that unrecoverable faults are induced for incorrect keys. Examples include locking the FPU to throw a *divide by 0* exception (fatal error), or locking the branch predictor to force a branch to NULL (fatal security exception). Therefore, when a wrong key is applied, any logic locking induced fault injection will cause an error with a critical application impact.
- Locking to ensure fault propagation. For example, locking multiple cache controllers so that locking induced errors in low level caches trigger block write-back to high level caches/main memory. Increasing error propagation throughout memory reduces the odds of error masking.

6.3 Evaluating Security-Aware Design

Now that we have identified a series of security-aware design modifications for both on-chip memory and data path components, we continue by implementing and evaluating these design decisions. To this end, we leveraged ObfusGEM to design and evaluate security-aware modifications to our x86 and ARM A53 cores. As shown in Section 4, cutting edge locking was unable to protect either architecture. Therefore, for success, we must implement minor architectural design decisions within the on-chip memory and data path that sufficiently improve the application level security of incorporated logic locking art so that both error severity and attack resilience can be achieved simultaneously. For this section,

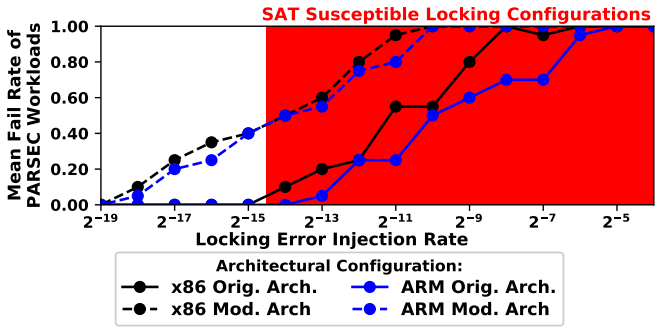

Figure 5: The effect of on-chip memory hierarchy redesign on the security of L1 D-cache controller locking.

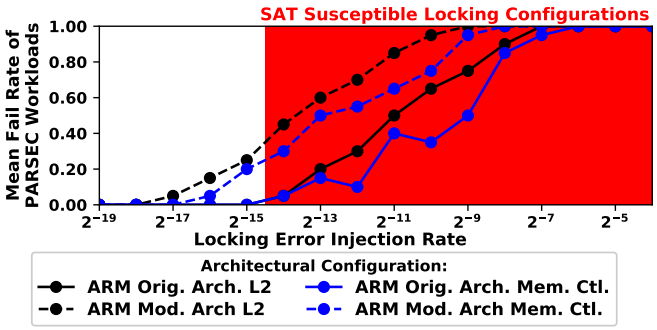

Figure 6: The effect of on-chip memory hierarchy redesign on the supply chain security of L2 cache controller and DRAM controller logic locking.

we targeted several on-chip memory and data path modules and developed security-aware architectures capable of amplifying supply chain security for each. The evaluation of each proposed design proceeded as follows.

(1) The location under test was locked with configurations identical to those in Section 4.1 (i.e. the same randomly selected input cube of varying length was locked). Therefore, for this experiment, only the architecture of the IC was modified compared to Section 4.1.
(2) Security-aware architecture modifications were applied.
(3) ObfusGEM compared the application level effects of locking within the modified and un-modified processor.

6.4 Experiment 1: Security-Aware On-Chip Memory Design

To evaluate the effectiveness of a security-aware design approach for the on-chip memory of a processor, we chose to redesign the L1 D-cache. To this end, we selected the locking configuration implemented within the L1 cache controller of each processor in Section 4 as our candidate locking configuration. We then redesigned the L1 D-cache of the IC to amplify supply chain security.

Specifically, we made 2 design decisions simultaneously. 1) The associativity of the cache was increased. This increases the cache tag length and the number of unique inputs applied to the cache controller's tag logic (input diversity). 2) The locking was designed to map locked minterms to fatal errors at the output of the cache controller. Both the x86 and ARM core were initially designed with 2-way set associative L1 D-caches. For this experiment, we increased the associativity of this cache to 8-way set associative. Additionally, the locking configuration was modified to produce an invalid tag whenever a locked input was applied to the cache controller. Other than these changes, all other aspects of the on-chip memory and locking configuration were fixed. ObfusGEM results for this experiment are in Figure 5.

A similar approach can be taken to enhance locking within higher level caches as well. To demonstrate this, we narrowed our focus to solely the ARM A53 processor testbed. The cache and DRAM control logic within this processor is more complex than the selected x86 core, thereby allowing more design modifications without significant redesign. For this experiment, we attempted to redesign the on-chip memory hierarchy to amplify both L2 cache controller locking and DRAM controller locking. To do so, we enabled hardware pre-fetching within both the L1 D-cache and L2 cache. By enabling pre-fetching, both the diversity and number of minterms applied to both the cache controller and DRAM controller can be increased. We also enabled speculative execution within the core, enabling the processor to execute instructions based on conditional branch predictions. Once again, this modification both increases the diversity and amount of traffic occurring within the on-chip memory system. We have aggregated ObfusGEM simulation results quantifying the supply chain security achieved by locking both the L2 cache controller and the DRAM controller in our ARM A53 core with both pre-fetching and speculative execution enabled in Figure 6.

6.5 Experiment 2: Security-Aware Data Path Design

To evaluate the effectiveness of a security-aware design approach for the data path of a processor, we chose to redesign the floating point unit. To this end, we selected the floating point adder locking configuration from Section 4 as our candidate locking configuration. We then redesigned the floating point unit of the IC to amplify the achievable supply chain security.

To improve floating point adder locking, we explored 2 architectural approaches. 1) We increased the number of floating point adder functional units (FUs) within the core. Each FU was then independently locked for a separate, randomly chosen input cube. 2) We implemented a *smart scheduler*, a redesigned version of each processor's out-of-order scheduler which favors locked FUs (only if that FU was available) for any operation on locked input minterms. For both the x86 and ARM processor, only a single FPU adder was included within the design. Therefore, for our evaluation, we increased the number of floating point adders to both 2 and 4 for both cores. ObfusGEM results for both designs (alongside the baseline from Section 4) are in Figure 7.

A similar approach can be applied to other data path modules as well. To demonstrate this, we performed the same experiment on the second best data path locking configuration, the integer adder. To amplify supply chain security in this module, we increased

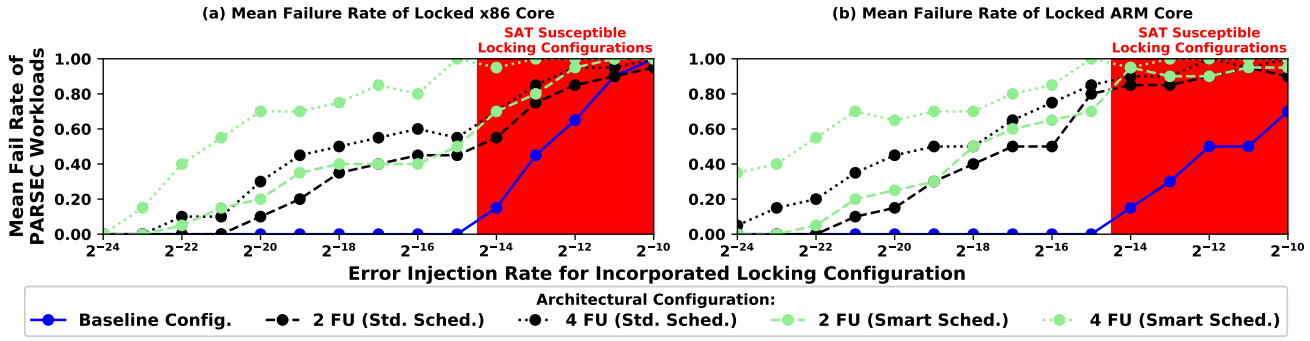

Figure 7: The effect of modified FU count and scheduler redesign on the application level security of FPU adder locking.

	X86 Core	ARM Core		
	L1 D-Cache	L1 D-Cache	L2 Cache Cont.	DRAM Cont.
Area	9.1%	5.5%	4.1%	4.3%
Peak Power	2.2%	1.2%	3.3%	3.4%
Runtime	-1.2%	-0.1%	0.7%	0.7%
Clock Rate	0.0%	0.0%	0.0%	0.0%

Table 1: Design overhead for x86 and ARM core redesigned with a security-aware on-chip memory architecture. Note that these numbers include locking overhead in addition to the overhead of any architectural redesign.

the number of integer adders in each core from 2 to 4. We then locked each of these additional adder circuits independently for a randomly chosen input cube. The ObfusGEM simulation results quantifying the impact of this design change on the supply chain security achieved by integer adder locking is included in Figure 8.

6.6 Experimental Design Overhead

By their very nature, the architectural changes we have implemented will impact the design parameters (i.e. area, delay, power, performance) of each device. While this is not ideal, we note that strong supply chain security guarantees are crucial to ensuring both the IP and integrity of a device. Therefore, we argue that strong supply chain security guarantees are a necessary component of IC design. We have aggregated the design overhead of the proposed architectural design modifications in Tables 1 and 2. Runtime was modeled with ObfusGEM through the runtime of PARSEC benchmarks. Processor power and area were estimated based on GEM5 data using the McPAT modeling framework [10] with a 32nm technology library. Note that the estimated overhead of logic locking each module with SFLL-Fault is included within the design overhead as well. To do so, we added the average area and power overhead of SFLL-Fault from [25] within the McPAT model of any locked design component. For each experiment, we fixed the clock rate for each architecture because scaling clock frequency would generally be considered a severe design modification. For this reason, no degradation was seen in the clock rate of any design.

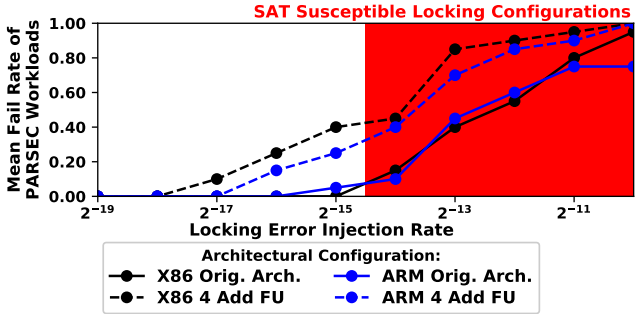

Figure 8: The effect of modified FU count on the application level security of integer adder locking.

6.7 Analysis of Security-Aware Designs

As seen in Figures 5-8, each proposed security-aware redesign yielded locking configurations that reliably derailed processor functionality (achieved error severity) with exponentially smaller error

Security-Aware X86 Data Path Redesign Overhead					
	Int. Adder Locking	FPU Adder Locking			
		No Smart Sched.		Smart Sched.	
	4 FU	2 FU	4 FU	2 FU	4 FU
Area	11.7%	12.1%	24.8%	12.1%	24.8%
Peak Power	9.3%	10.3%	20.7%	10.3%	20.7%
Runtime	-0.9%	-8.7%	-10.8%	-8.7%	-10.8%
Clock Rate	0.0%	0.0%	0.0%	0.0%	0.0%
Security-Aware ARM Data Path Redesign Overhead					
	Int. Adder Locking	FPU Adder Locking			
		No Smart Sched.		Smart Sched.	
	4 FU	2 FU	4 FU	2 FU	4 FU
Area	11.5%	12.3%	25.3%	12.3%	25.3%
Peak Power	9.2%	9.6%	20.2%	9.6%	20.2%
Runtime	-0.7%	-2.4%	-2.5%	-2.4%	-2.5%
Clock Rate	0.0%	0.0%	0.0%	0.0%	0.0%

Table 2: Design overhead for x86 and ARM core redesigned with a security-aware data path architecture. Note that these numbers include locking overhead in addition to the overhead of any architectural redesign.

injection rates. The trade-off identified in [39] proves that a linear decrease in error injection rate yields a linear increase in the SAT attack resilience of SFLL-Fault. This means that locking can obtain exponentially stronger SAT attack resilience while still maintaining equivalent error severity in these modified architectures. Because a secure locking configuration must achieve both error severity and attack resilience simultaneously, this constitutes an exponential improvement in the security of logic locking. In fact, through the security-aware design of both the on-chip memory and data path of the IC, logic locking critically impacted PARSEC benchmarks (achieved error severity) with error injection rates residing outside the red-shaded SAT susceptible region. Therefore, our security-aware design modifications enabled locking to simultaneously achieve both error severity and SAT attack resilience in both cores. In Section 4, the same locking configurations were unable to achieve security in each un-modified IC. Hence, each design approach was successful in allowing a designer to achieve supply chain security.

Despite this positive result in both cases, there are clear differences between on-chip memory hierarchy and data path redesign for this purpose. First, we note that the strongest supply chain security was achieved through our redesign of the FPU adder. In this case, non-zero application failure rates could be observed with error injection rates of 2^{-20} in all cases. However, this increase required a significant increase in the both the area and power consumption of the design, with over a 20% increase in the worst case. While it is possible that some processor designs could absorb this level of design overhead to achieve supply chain security, this additional overhead would likely be unfeasible in most cases. A similarly large overhead can be seen in the integer adder redesign as well. Hence, both evaluated data path designs induced substantial overhead.

Comparatively, the on-chip memory redesign yielded slightly smaller security improvements, but with substantially less design overhead. For example, our L1 D-cache redesign allowed L1 cache controller locking to achieve error severity with an error injection rate of 2^{-17}. This error injection rate resides well outside of the SAT susceptible region. However, this design required less than a 10% area overhead in the worst case and only a 1-2% increase in peak power. This level of design overhead is much more manageable.

We found that the design overhead of proposed on-chip memory modifications was much lower than data path modifications due to the more subtle changes available to the IC designer in memory hierarchy design. In our redesign of the data path, we relied on simply increasing the number of functional units within the device, an extremely coarse design modification. However, the on-chip memory hierarchy had a wide array of candidate design modifications, such as associativity, cache size, pre-fetching, or hierarchy organization. Each of these design modifications can be finely tuned to greatly impact the environment a cache or DRAM controller operates in. This makes each of these design changes ideal for a supply chain security-aware design approach. For example, by enabling hardware pre-fetching and speculative execution, 2 features already available within the hardware, we were able to increase DRAM controller traffic by 17.1%. This increase in traffic substantially increased the utilization and diversity of minterms applied to the locked module, exponentially improving supply chain security, while imposing minimal area/power/performance overhead.

Therefore, it appears that the on-chip memory hierarchy serves as a more viable candidate for logic locking. Unlike the data path, the complexity of the on-chip memory system enabled a diverse array of modifications capable of substantially enhancing supply chain security. While we obviously did not explore every aspect of locking within the on-chip memory hierarchy, our results demonstrate the promise of a memory-focused logic locking approach.

6.8 Summary of Security-Aware Design

To conclude, our on-chip memory and data path redesign in the x86 and ARM A53 core exponentially improved application level supply chain security. While the design overhead of the data path redesign approach was substantial in many cases, the on-chip memory redesign exhibited a much more modest increase in design overhead. The success of this approach supports the results of prior research noting the on-chip memory hierarchy as an ideal locking candidate [6, 45]. However, we note that achieving security in either of these components of an IC required some architectural tuning in both testbed processors. This result not only demonstrates the importance of IC architecture for supply chain security with logic locking, but also emphasizes the importance of a security-aware approach to architecture design.

Our security-aware approach was made possible by ObfusGEM, which both enabled us to identify the factors limiting logic locking and to design/evaluate changes to mitigate these factors. Therefore, while we have shown that an ObfusGEM-driven, security-aware design approach can achieve strong application level security within custom ICs, we also note that the presented results are just a small slice of the security-aware designs made possible by ObfusGEM. To this end, we have released the ObfusGEM simulator alongside this work to enable others in the research community to identify alternative security-aware design modifications and implementation methodologies capable of achieving the application level security currently missing from cutting edge logic locking approaches.

7 CONCLUSION

To begin our work, we performed a design space exploration of logic locking in 2 processors. Based on our exploration, we found that logic locking was unable to achieve application level security and attack resilience simultaneously. We identified input-space non-uniformity and processor error resilience as 2 of the factors which limited locking in these ICs. We then proposed security-aware architecture design to overcome these limitations. To facilitate security-aware design, we developed and released ObfusGEM[5], an open-source logic locking simulation framework to aide designers in both the design and evaluation of secure ICs. We used ObfusGEM to perform security-aware architecture design of the on-chip memory and data path of our 2 processor testbeds that were insecure with prior art. Our proposed security-aware on-chip memory and data path designs were shown to exponentially improve security. In the case of on-chip memory redesign, these exponential improvements incurred only a modest design overhead, serving as a viable approach to allow locking to achieve strong security guarantees in these previously insecure devices.

[5] ObfusGEM can be found at: "*https://github.com/mzuzak/ObfusGEM*".

ACKNOWLEDGMENTS

This work was supported by the ARCS Foundation, the National Science Foundation (NSF) Grant 1642424, and the Air Force Office of Scientific Research Grant FA9550-14-1-0351.

REFERENCES

[1] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. 2019. SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks. *Transactions on Cryptographic Hardware and Embedded Systems* (2019).

[2] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[4] Alberto Bosio and Giorgio Di Natale. 2008. LIFTING: A flexible open-source fault simulator. In *2008 17th Asian Test Symposium*. IEEE, 35–40.

[5] Abhishek Chakraborty, Nithyashankari Gummidipoondi Jayasankaran, Yuntao Liu, Jeyavijayan Rajendran, Ozgur Sinanoglu, Ankur Srivastava, Yang Xie, Muhammad Yasin, and Michael Zuzak. 2019. Keynote: A Disquisition on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (09 2019). https://doi.org/10.1109/TCAD.2019.2944586

[6] Abhishek Chakraborty, Yang Xie, and Ankur Srivastava. 2018. GPU obfuscation: attack and defense strategies. In *Design Automation Conference*.

[7] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2014. GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications. In *IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE.

[8] Nithyashankari Gummidipoondi Jayasankaran, Adriana Sanabria Borbon, Edgar Sanchez-Sinencio, Jiang Hu, and Jeyavijayan Rajendran. 2018. Towards provably-secure analog and mixed-signal locking against overproduction. In *Proceedings of the International Conference on Computer-Aided Design*. 7.

[9] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. 2019. Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks. In *Proceedings of the 56th Annual Design Automation Conference*.

[10] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM International Symposium on Microarchitecture*.

[11] Xuanhua Li and Donald Yeung. 2007. Application-level correctness and its impact on fault tolerance. In *2007 IEEE 13th International symposium on high performance computer architecture*. IEEE, 181–192.

[12] Yuntao Liu, Michael Zuzak, Yang Xie, Abhishek Chakraborty, and Ankur Srivastava. 2020. Strong Anti-SAT: Secure and Effective Logic Locking. In *International Symposium on Quality Electronic Design (ISQED)*.

[13] Shubhendu S Mukherjee, Joel Emer, and Steven K Reinhardt. 2005. The soft error problem: An architectural perspective. In *11th International Symposium on High-Performance Computer Architecture*. IEEE, 243–247.

[14] Prashant J Nair, David A Roberts, and Moinuddin K Qureshi. 2016. Fault Sim: A Fast, Configurable Memory-Reliability Simulator for Conventional and 3D-Stacked Systems. *ACM Transactions on Architecture and Code Optimization (TACO)* (2016).

[15] Christian Pilato, Francesco Regazzoni, Ramesh Karri, and Siddharth Garg. 2018. TAO: techniques for algorithm-level obfuscation during high-level synthesis. In *Design Automation Conference*.

[16] M Sazadur Rahman, Adib Nahiyan, Sarah Amir, Fahim Rahman, Farimah Farahmandi, Domenic Forte, and Mark Tehranipoor. 2019. Dynamically Obfuscated Scan Chain To Resist Oracle-Guided Attacks On Logic Locked Design. Cryptology ePrint Archive, Report 2019/946. https://eprint.iacr.org/2019/946.

[17] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. 2012. Security analysis of logic obfuscation. In *Proceedings of Design Automation Conference*.

[18] Amin Rezaei, You Li, Yuanqi Shen, Shuyu Kong, and Hai Zhou. 2019. CycSAT-unresolvable cyclic logic encryption using unreachable states. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 358–363.

[19] R Robache, J-F Boland, Claude Thibeault, and Yvon Savaria. 2013. A methodology for system-level fault injection based on gate-level faulty behavior. In *2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS)*. IEEE, 1–4.

[20] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. 2018. SRCLock: SAT-resistant cyclic logic locking for protecting the hardware. In *Great Lakes Symposium on VLSI*.

[21] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. 2008. EPIC: Ending piracy of integrated circuits. In *Conference on Design, automation and test in Europe*.

[22] Pia N Sanda, Jeffrey W Kellington, Prabhakar Kudva, Ronald Kalla, Ryan B McBeth, Jerry Ackaret, Ryan Lockwood, John Schumann, and Christopher R Jones. 2008. Soft-error resilience of the IBM POWER6 processor. *IBM Journal of Research and Development* (2008).

[23] Abhrajit Sengupta, Mohammed Ashraf, Mohammed Nabeel, and Ozgur Sinanoglu. 2018. Customized locking of IP blocks on a multi-million-gate SoC. In *International Conference on Computer-Aided Design*.

[24] Abhrajit Sengupta, Mohammed Nabeel, Nimisha Limaye, Mohammed Ashraf, and Ozgur Sinanoglu. 2020. Truly Stripping Functionality for Logic Locking: A Fault-based Perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[25] Abhrajit Sengupta, Mohammed Nabeel, Muhammad Yasin, and Ozgur Sinanoglu. 2018. ATPG-based cost-effective, secure logic locking. In *IEEE 36th VLSI Test Symposium (VTS)*. IEEE.

[26] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. 2020. CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 175–202.

[27] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. 2017. AppSAT: Approximately deobfuscating integrated circuits. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 95–100.

[28] Kaveh Shamsi, Meng Li, Kenneth Plaks, Saverio Fazzari, David Z Pan, and Yier Jin. 2019. IP Protection and Supply Chain Security through Logic Obfuscation: A Systematic Overview. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2019).

[29] Kaveh Shamsi, Travis Meade, Meng Li, David Z Pan, and Yier Jin. 2018. On the approximation resiliency of logic locking and IC camouflaging schemes. *IEEE Transactions on Information Forensics and Security* (2018).

[30] Kaveh Shamsi, David Z Pan, and Yier Jin. 2019. On the impossibility of approximation-resilient circuit locking. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 161–170.

[31] Yuanqi Shen and Hai Zhou. 2017. Double dip: Re-evaluating security of logic encryption algorithms. In *Great Lakes Symposium on VLSI 2017*.

[32] Deepak Sirone and Pramod Subramanyan. 2019. Functional analysis attacks on logic locking. In *Design, Automation & Test in Europe Conference & Exhibition*.

[33] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 137–143.

[34] Yang Xie and Ankur Srivastava. 2016. Mitigating sat attack on logic locking. In *Conference on Cryptographic Hardware and Embedded Systems*.

[35] Fangfei Yang, Ming Tang, and Ozgur Sinanoglu. 2019. Stripped Functionality Logic Locking with Hamming Distance Based Restore Unit (SFLL-hd)–Unlocked. *IEEE Transactions on Information Forensics and Security* (2019).

[36] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *Intl. Symposium on Hardware Oriented Security and Trust*.

[37] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. 2017. Removal attacks on logic locking and camouflaging techniques. *Transactions on Emerging Topics in Computing* (2017).

[38] Muhammad Yasin, Jeyavijayan JV Rajendran, Ozgur Sinanoglu, and Ramesh Karri. 2016. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2016).

[39] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. 2017. Provably-secure logic locking: From theory to practice. In *Conference on Computer and Communications Security*.

[40] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan JV Rajendran. 2017. What to lock?: Functional and parametric locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*.

[41] Monir Zaman, Abhrajit Sengupta, Danqing Liu, Ozgur Sinanoglu, Yiorgos Makris, and Jeyavijayan JV Rajendran. 2018. Towards provably-secure performance locking. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1592–1597.

[42] Hai Zhou. 2017. A Humble Theory and Application for Logic Encryption. *IACR Cryptology ePrint Archive* 2017 (2017), 696.

[43] Hai Zhou, Amin Rezaei, and Yuanqi Shen. 2019. Resolving the Trilemma in Logic Encryption. In *International Conference on Computer-Aided Design (ICCAD)*.

[44] Michael Zuzak, Yuntao Liu, and Ankur Srivastava. 2020. Trace Logic Locking: Improving the Parametric Space of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[45] Michael Zuzak and Ankur Srivastava. 2019. Memory Locking: An Automated Approach to Processor Design Obfuscation. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 541–546.