Dear Editor and Reviewers,

Thank you for your efforts in reviewing our TCAD submission. We are happy to see that in your opinion “The problem the work attempts to solve is very relevant and important. The paper is very clear and well written, providing sufficient background for readers. The evaluations are sound and holistic” and “The approach taken here is novel”.

We have fixed the handful of typos identified in the draft, added the reference suggested by the AE [9], clarified Algorithm 1, addressed suggestions found in reviews, as well as comments from the attendees of our ICCAD 2014 presentation. Following up on the vague suggestion that MUX-based locking has been explored in recent literature, we noticed a related paragraph in [9] under their detailed literature review. We double-checked the references, but did not find anything relevant that we have not already cited.

Additional experiments were performed in this revision — their results are reported in new tables and analyzed in the text. In particular, we examine the robustness of MUX-based locking against a general (not necessarily test-based) simulation attack, and compare the extent by which a MUX-locked and random XOR-locked circuit circuit deviate from a functionally correct circuit. We have also demonstrated a logic transformation by which XOR locks can be generalized into MUX locks, and discussed practical uses of proposed techniques in large circuits and with industry design flows. A side-by-side comparison with the ICCAD 2014 paper makes additional material obvious, including several new paragraphs, subsections, figures, experiments, tables and references. We prefer that the reviewers evaluate the new material in proper context rather than by enumeration.

We respectfully disagree with Reviewer 1’s claim that ”The ICCAD 2014 version appears to be neither a preliminary version of this submission nor is it substantially shorter.” We believe the ICCAD 2014 version satisfies the definition of the word “preliminary” as it was finalized before the journal submission, which extended it by several pages. Since Reviewer 1 was otherwise happy with the technical contribution and writing quality, we hope that the current, even longer revision, is now acceptable.

Reviewer 3 asked to explain how our approach remains practical in a situation where test vectors are not necessarily known in advance. To address this question, the current draft empirically evaluates a scenario where our proposed locking accounts for a larger number of test vectors than necessary. This allows one to select most appropriate test vectors in the field - the responses will be unaffected by circuit locking.

Reviewer 3 also asked if locking sites can be selected better than at random (which is what we did in our experiments). This question has been addressed in several prior works - for example, it is better not to insert multiple locks on the same path and not to slow down timing-critical paths. In fact, the draft reviewed by Reviewer 3 explained that.

Regarding Reviewer 3’s question about circuit size and whether our techniques scale to million-gate circuits. Our technique is purely combinational by construction and is meant to be applied to combinational sections of larger circuits. To this end, we do not foresee purely combinational million-gate circuits. For example, in a datapath with ten combinational stages, it would typically be sufficient to lock one or two stages - as long as they cannot be bypassed. Another possibility is to lock control logic, which is usually small. We have added this explanation to the manuscript.

The Authors
Solving the Third-Shift Problem in IC Piracy with Test-aware Logic Locking

Stephen M. Plaza, Igor L. Markov

Abstract—The increasing IC manufacturing cost encourages a business model where design houses outsource IC fabrication to remote foundries. Despite cost savings, this model exposes design houses to IC piracy as remote foundries can manufacture in excess to sell on the black market. Recent efforts in digital hardware security aim to thwart piracy by using XOR-based chip locking, cryptography, and active metering. To counter direct attacks and lower the exposure of unlocked circuits to the foundry, we introduce a multiplexor-based locking strategy that preserves test response allowing IC testing by an untrusted party before activation. We demonstrate a simple yet effective attack against a locked circuit that does not preserve test response, and validate the effectiveness of our locking strategy on IWLS 2005 benchmarks.

I. INTRODUCTION

Fighting counterfeit electronics has become a major challenge not only in engineering research, but also to ensuring the viability of the electronics industry and a broad variety of products that rely on integrated circuits. US Congressional hearings in November 2011 revealed that numerous substandard ICs were found in military electronics. They were featured on CNN and covered in depth by a US Department of Commerce study [5]. To this end, we seek to counteract unauthorized manufacturing (overbuilding) and support active-metering techniques. As explained in [10], the heavy costs of semiconductor manufacturing force IC supply chains to stretch across the Pacific. By outsourcing fabrication, the owner of IP rights invites unauthorized production of black-market ICs that undercut legitimate ICs (Figure 1) [20]. Whether such acts of IC piracy are condoned by the management of fabrication facility, or committed at a different facility, does not diminish the damage caused and does not help address the problem. Another threat model is to intentionally produce degraded or altered ICs looking like normal ICs, facilitate sabotage [20]. Despite significant interest from US DoD and Legislature, these challenges have been ignored by the EDA industry until recently. Yet, in the late 2013, major EDA companies started exploring business opportunities in the fight against IC piracy, specifically in the automotive and defense sectors.

Threat models in IC piracy and countermeasures are reviewed in [20]. A key countermeasure developed and analyzed in [1], [2], [8]–[10], [15], [18], [21] is active IC metering, which forces every new IC produced to be activated through real-time, interactive electronic communication with owners of IP rights. Otherwise, the IC will not work. Active-metering schemes [1] have recently attracted significant interest: in addition to suggesting improvements, [15] noted a step-ordering ambiguity in the DATE 2008 version of EPIC, clarified in the journal version [21]. Exploits for the DATE 2008 version of were also claimed by [18], along with antidotes.

We focus on a key feature of EPIC [21] — the need to activate the chip before circuit test, most likely at the fabrication facility. This unnecessarily exposes the activation protocol and the unlocked ICs, facilitating various attacks, as shown in [15], [21]. To reinforce this point, we develop a new algorithmic attack that uses test-patterns and observed responses. The algorithm performs a randomized local search with restarts, guided by the number of matching bits in the output response, that typically produces a correct key value. The (surprising) empirical success of this key-extraction attack hints at an underlying mathematical structure in the behavior of large combinational circuits.

To thwart the new attack and to rule out attacks suggested in [15], [18], we develop a methodology for combinational locking that supports post-manufacturing test of locked circuits before activation. Unlike prior methods that insert XOR gates (Figure 2), our new combinational locking inserts multiplexors. The insertion is based on functional simulation and tries to match logic covers of internal signals. The importance of performing IC test before IC activation as a security measure was recently articulated in [3]. Their strategy requires
significant infrastructure for remote test and on-chip logic to scramble test response, including cryptography and binary tags. In comparison, our proposal is lightweight.

The remaining part of this paper is structured as follows. Section II outlines relevant background on active IC metering, highlighting its key aspects considered in our research, then reviews recent literature and interaction with circuit test. It also articulates technical opportunities pursued in this paper. Section III introduces an algorithmic test-based attack on EPIC and partially motivates the development in Section IV of combinational locking to enable post-manufacture test of locked circuits. Empirical validation on IWLS 2005 benchmarks is reported in Section V, which also discusses embedding of proposed techniques in realistic design flows. Conclusions are given in Section VI. Readers confused by inconsistent terminology in recent literature may benefit from the Appendix.

II. BACKGROUND: ACTIVE METERING

Given recent interest in active metering [1], [2], [8], [9], [15], [18], [21], we illustrate it by combinational locking in the EPIC protocol [21] and related chip-locking schemes [2], [8], then review its cost-vulnerability tradeoffs (a formal description of EPIC can be found in [15], along with analysis and improvements). We also discuss attacks from [18].

**EPIC.** Building on prior work [1], [21] propose chip-locking and activation system for IC metering, while aiming to make “physical tampering unprofitable and attacks computationally infeasible.” In other words, defending against an omnipotent attacker with unbounded resources is not the goal — simpler attacks should be given priority.

For example, reverse-engineering (a part of) circuit layout is harder than running a live circuit on given inputs. Observing internal dynamic voltage levels in a live circuit is more difficult than observing static gates or wires. Modifying a circuit typically requires understanding some part of it. Since EPIC draws on unique process variations to ensure different responses in ICs produced from the same mask, an attack that requires work for each individual IC will require considerable resources. Due to low margins in the IC business, a per-chip cost increase can make mass-production unprofitable. Moreover, physical inspection and modification of ICs are becoming increasingly challenging at each new technology node due to smaller features.

To establish a combinational lock, EPIC [21] modifies a combinational circuit by adding XOR/XNOR gates with fanins connected to the bits of common key (CK) that unlocks the circuit, as shown in Figure 2. Correct key bits simplify the XOR/XNOR gates to wires, while incorrect key bits produce unintended inversions. A variety of equivalent circuits can be produced through randomized application of de Morgan’s rules for basic logic gates: $a \oplus b = \bar{a} \bar{b} + a b = a \oplus b, \overline{a b} = \bar{a} + \bar{b}$ and $a + b = \bar{a} \bar{b}$. Locked IC will fail post-manufacturing test, hence unlocking must occur at the fabrication facility before test. Care is taken to avoid circuit delay overhead on critical paths [21]. Simple removal of XOR/XNOR gates is ineffective if inversions are propagated through the circuit and especially if logic restructuring is performed after locking. The work in [2] is similar in principle to EPIC but uses LUT-based locks that hinder attempts to reverse-engineer functionality from the layout. It also illustrates how chip locking and active metering can be successfully adapted to FPGAs.

**Several attack vectors.** Both EPIC [21] and the analysis in [15] contemplate sophisticated attackers that obtained CK with some effort (guessing CK is shown difficult in [21]). They note that EPIC does not provide for direct entry of CK to unlock the circuit. Instead, the encrypted version of CK arriving from the owner of IP rights is protected by the RSA cryptosystem, which offers strong guarantees both in theory and in practice, though implementation-specific vulnerabilities exist [17]. The work in [18] assumes that the circuit has been reverse-engineered and develops attacks that simplify the search for CK, using unfortunate configurations of locking gates that may be created when inserting gates at random. Some of these configurations can be optimized during logic synthesis, and others are easy to avoid, e.g., as suggested in [18]. More critically, the authors of [18] consider an attack successful when CK is found and then focus on hiding CK better. In this context, recall that (i) reverse-engineering a large 22nm IC is going to be extremely difficult without access to the gate-level netlist, (ii) even if CK is found, entering it directly through mask modification demands advanced expertise and access to key infrastructure, increasing attacker’s costs and barriers. In comparison, attacks discussed in [15] can be perpetrated by intercepting communications during activation session.

In Section III, we describe a surprisingly effective CK-finding attack that uses either (i) a reverse-engineered or stolen gate-level circuit for simulation, or (ii) a live circuit with controllable inputs and accessible outputs. It does not make assumptions about how gates are inserted and is therefore completely unaffected by logic optimization or restructuring. The conclusion one should draw from this attack, as well as from those in [15], [18], is that chip activation should not be performed at the fabrication facility. On the other hand, as EPIC combines multiple layers of protection, viewing it as either secure or defeated would be misleading.
Observations and opportunities. Returning to the algorithmic description of EPIC, we note its interaction with circuit test. Activation is performed at the fabrication facility before circuit test because locked ICs will fail test. This exposes additional information to the manufacturer and enables additional attack vectors.

- The unlocked functionality is known after IC activation and during test, which may simplify and accelerate search for CK, including its encrypted form that can be entered directly,
- Including CK bits in post-manufacturing scan-test of locked circuits can significantly undermine EPIC if this facilitates entering CK bits directly [4], [19].

To improve IC security, testing and fabrication can be performed at separate locations by different commercial and legal entities. Such an arrangement can be described as split test. It has been explored in [3], at the cost of significant overhead. We pursue split test in a context that avoids significant changes in infrastructure by relying on more sophisticated ATPG algorithms. For further security, tamper-resistant packaging can deny access to activated chips available on the open market. On the other hand, ICs intended for sensitive military equipment (commonly mentioned as motivators for IC security research) should not be available on the open market.

Split test requires support in ATPG algorithms. To this end, we develop techniques to find patterns for testing both locked and unlocked ICs, so as to move activation from the fabrication facility without hampering yield optimization. Given the multiple concerns addressed by modern circuit test and its logistical complexity, it is important to maintain significant freedom for test-patterns.

III. AN EPIC ATTACK

We now introduce an algorithmic attack on EPIC that derives CK by simulating a set of test patterns. It can be executed on a stolen or reverse-engineered gate-level netlist, or on a physical circuit with direct access to CK inputs and circuit outputs (e.g., through scan-chains or maliciously inserted side-channels). Without the gate-level circuit, the attack additionally needs (i) test patterns and expected output responses, or (ii) an unlocked circuit that can produce correct responses on random inputs.\(^2\) On the other hand, since EPIC does not offer direct control of CK bits on an actual IC (Section II), using the results of the proposed attack would require malicious, although small, mask modification.

Our attack introduced in Algorithm 1 is iterative in nature: a random key candidate is gradually improved based on observed test responses. It uses hill-climbing search guided by the number of differences in the output response (for a key combination). At each iteration, randomly-selected key bits are toggled one by one. The function test() applies all test patterns to the current key combination. A key bit value of 0 or 1 is chosen to minimize differences between observed and expected responses. As described so far, the algorithm resets all key bits if no solution is found in one iteration. In practice,

\(^2\) Security pitfalls of scan-chains are well-known [19] and are being addressed through compression and encryption [4].

Input: Locked Circuit: ckt, Patterns: patterns
Expected Test Response: resp
Output: CK \(\in \{0,1\}^n\)

found = false;
while found do
    CK ← list random\{0,1\};
    foreach random ck in CK do
        resp1 = test(ckt(CK), patterns);
        diffs1 = diff(resp1,resp);
        // flip one random bit
        ck = !ck;
        resp2 = test(ckt(CK), patterns);
        diffs = diff(resp2, resp);
        if diffs1 < diffs then
            // original value of random bit
            ck = !ck;
            diffs = diffs1;
            if !diffs then
                if correct(ckt) then
                    found = true;
                    break
            end

end

Algorithm 1: Extracting CK bit values from a locked circuit given test input and output vectors using a hill-climbing algorithm that monitors output response.

This foreach loop can be run multiple times until a local minimum is reached, followed by a restart with a new random configuration. The function correct() is an oracle that checks whether the current key combination, which satisfies the test response, is equal to CK. In practice, the oracle could be implemented through more exhaustive validation with an unlocked circuit or other expected response. If correct() is called several times, we can save lock-down key bits (not shown in algorithm) that do not vary between each solution.

Unlike the work in [18], our proposed approach does not require isolated sensitization of key bits or even netlist access (assuming that the scan chain is exposed). The insight is that the output response often betrays a gradient toward the target configuration (this need not occur every time, but sufficiently often). A key combination with fewer errors indicates an improvement in key combination. To develop intuition, consider a circuit where each XOR lock impacts a circuit output not impacted by other XOR locks. For a random combination of key bits, the output will differ from the expected test response by \(M\) bits. Toggling one key bit produces a different \(M\) value, and of the two resulting key bit combinations, the one with fewer differences is closer to a correct combination. Repeating such steps from a random initial combination often leads to a combination that unlocks the circuit. If not, the process can be restarted from a different random initial combination. The same logic applies with random placement of XOR locks, as in [21], as well as alternative techniques.
The complexity of this algorithm, in terms of the number of iterations required, does not directly depend on the size of the circuit. A higher density of locks in the circuit could add complex correlations [18] and jeopardize our strategy. However, more sophisticated strategies may explicitly seek key bits that impact the output in a correlated way, involve randomization, and maintain a pool of candidate configurations not to get trapped in local minima. More advanced attacks would examine the circuit’s response to constrained stimuli. Among countermeasures against gradient-based attacks, we mention mapping CK key bits to pseudorandom locking combinations.

A Generalized Simulation-based Attack. The above technique can be extended to arbitrary stimuli, not just test input patterns. However, such attacks would require access to a functional circuit or at least its expected outputs. Also, as before, the attack would need to use test scan infrastructure to properly stimulate deep logic in the design. When working chips can be legally obtained from consumer electronics products, even test-aware locking is potentially susceptible to such attacks. These scenarios motivate more advanced protection, such as disguising or encoding CK key bits.

IV. Test-aware Locking

In this section, we introduce a strategy for locking a circuit that preserves test responses and is therefore immune to our proposed attack, as its response betrays no gradient. We advocate a lightweight locking approach (where the CK is encrypted) that allows a manufacturer to test the chip without a fully functional circuit. While our locking strategy does not rule out attacks that consider any circuit output response in the spirit of Algorithm 1, it significantly complicates such efforts because test responses are identical for all key combinations. An attack would require comparing outputs with an unlocked chip (or netlist) and potentially a prohibitive simulation to expose deep circuit state.

We first introduce an approach using logic signatures to find logic covers that preserve test response. Then we introduce an algorithm for inserting combinational locks in a circuit.

A. Test-Proof Locking using Logic Signatures

In [21], a lock at node $F$ consists of creating an alternative signal $F'$ created by adding an XOR (or XNOR) between $F$ and a key bit. An incorrect key bit value gives $F' \equiv \neg F$; the correct one gives $F' \equiv F$. $\neg F$ will not preserve the test response unless $F$ is redundant. Alternatively, we try to find (or synthesize) an $F'$ that preserves the test response as explained in the next few paragraphs.

A logic signature is a partial truth table that captures the function of a given circuit node [12]. For a circuit and its $K$ input vectors $X_1 ... X_K$, the logic signature of a functional node $F$ in the circuit is:

$$S_F = \{F(X_1), ..., F(X_K)\}$$

Evaluating all input combinations turns $S_F$ into a complete truth table. In practice, a set of random input vectors applied to a circuit can provide a useful mechanism for analyzing restructuring opportunities, such as identifying potential node equivalences [13], [16]. Since $S_F = S_G$ does not imply that $F = G$, such equivalences must be verified in general (more on this below). The time complexity of producing $K$-bit signatures for an $N$-node circuit is $O(NK)$. Signatures are generated quickly, as $K$ is typically small. We generate signatures using both random input vectors and test input vectors. Figure 3a illustrates a circuit stimulated with different test vectors. For instance, $S_{X_1} = \{0, 1, 1\}$. Given these signatures, we seek an alternative implementation of a signal (in this example $x_3$) that preserves test response. We can look first for a node in the circuit that has an equivalent signature to $S_{x_3}$. Since no such signature exists, we must synthesize a function. We use the signatures to identify nodes that cover $x_3$ (similar to the strategy in [11]). A logic cover $Y$ of $x_3$ is defined as:

$$S_{x_3} \subseteq S_Y \Rightarrow S_{x_3} \cup S_Y = S_{x_3}$$

(2)

(we say that $Y$ covers $x_3$ up to given test vectors). In Figure 3a, signature $S_{x_4}$ covers signature $S_{x_3}$ since $x_4 = 1$ every time $x_3 = 1$ for the input patterns that define the signatures.

Unlike previous work, we do not need to formally validate that a candidate logic cover or node equivalence exists for all possible input combinations. On the contrary, we must show that input combinations that are not test vectors violate the equivalences. Figure 3b shows that we can use random simulation to generate another set of signatures that disproves the logic covers previously found. In this example, $S_{x_4}$ does not cover $S_{x_3}$. Therefore, $x_4$ does not cover $x_3$. Any function that replaces $x_3$ with $x_3 \& x_4$ will preserve test response but will alter the circuit’s behavior in general. Figure 3c shows that a MUX gate can be added where the select is the locking key bit that chooses between the correct $x_4$ and $x_3 \& x_4$.

B. An Algorithm for Adding Key Locks

We outline our approach to circuit locking in Algorithm 2, which assumes the circuit to be locked and test patterns as inputs. It first generates two sets of signatures, one generated from random simulation and one from test-pattern simulation. The algorithm randomly traverses the netlist trying to find a logic cover for the selected signal. $\text{iscover()}$ is true if a cover is found but disproved by random simulation. If there is a cover, the function $\text{cover()}$ synthesizes it from $S_1$ and $S_2$ (or returns $S_2$ if it is equivalent to $S_1$). To ensure that the differences between $S_N$ and $S_1$ propagate to the output, we check its observability under random simulation. $\text{insertmux}$ replaces $S_1$ with the output of a MUX between $S_1$ and $S_N$. The algorithm terminates once $\text{numlock}$ locks are added.

In this approach, $\text{cover()}$ generates $S_N$ as a function of $S_1$. In general, any signal or combination of signals equal to $S_1$ under test simulation could be used. However, using $S_1$ is advantageous since it may be testable even with the wrong key bit value. Also, $S_1$ and $S_N$ would ideally be local to reduce violations of aggressive design constraints.

To guarantee that an incorrect key input leads to a difference at the outputs, we check the observability of each locked signal $\text{ckt.observable}$. Hence, incorrect key bits...
1. Apply Test Patterns

2. Find Candidate Cover

3. Apply Random Simulation

4. Disprove Candidate Cover

5. Add MUX lock

---

**Fig. 3.** Signatures are used to add locks to a circuit and preserve circuit-test properties: (a) signatures identify potential covers, (b) random simulation disproves the cover, (c) a MUX is inserted between the logic cover and the locked signal so as to preserve test response but not response to random simulation.

---

**Fig. 5.** Test-aware logic locking facilitates trustworthy activation of heterogeneous systems authorized by the holder of rights for the top-level system, while notifying right-holders of individual components.

result in a malfunctioning circuit. Our experiments check the observability of a locked signal with a set of random input patterns assuming that this is a subset of expected circuit payload (and that corresponding output values represent valid state). In more realistic settings, simulation patterns can be chosen among valid input states; output differences can be checked against valid output states. Two locked signals could theoretically cancel each other out, but even if such cases are not explicitly ruled out when positioning locks, this is astronomically improbable when the number of inserted MUXes (64 or 128) is small compared to circuit size.

A locked signal can be untestable given a wrong key bit value. In a circuit with \( N \) potentially untestable locked signals, we can test them by repeatedly applying test patterns with different key combinations. If there is little interference between locked circuit regions, the number of untested signals for \( t \) testing iterations is roughly governed by:

\[
N_{\text{untested}} \approx \frac{N}{2^t}
\]  

(3)

Notably, by ensuring that added locks are observable under random simulation, we miss potential locks that are rarely stimulated, which could enhance the resistance to simulation-based attacks. Also, we could make our approach visually harder to decipher (for an attacker with netlist access) by emphasizing locks that use local signals that are equivalent up to test vectors. Extracting the unlocking key-bit value is since it is determined by the order of the MUX inputs.

C. Proposed Methodology for IC Test

To minimize the impact of proposed testing-aware locking on typical chip-testing methodologies (Figure 4), two modifications can be made. To consider the first modification, recall that circuit test can use different subsets of test patterns
**Input:** Circuit: ckt, Patterns: patterns, Number of locks: numlock  
**Output:** Locked Circuit: ckt

```plaintext
randsigns = simulate(ckt);
testsigs = simulate(ckt, patterns);
foreach S1 in random({ckt.signals}) do
  foreach S2 in ckt.signals do
    if iscover(S1, S2) then
      SN = cover(S1, S2);
      if ckt.observable(SN) then
        newkey = random({0, 1});
        if newkey then
          ckt.insertmux(S1, S1, SN);
        else
          ckt.insertmux(S1, SN, S1);
        end
      end
      numlock--;
      break;
    end
  end
end
if numlock then
  break;
end
```

**Algorithm 2:** Inserts MUX locks in a circuit where key bits choose between the correct signal and a logic cover.

---

**D. Active Metering in Multichip Systems**

Our focus so far has been on the protection of a single IC and does not address weaknesses at the system level where the attacker can engage in chip replacement, tampering with components of an authorized system, or unauthorized reuse of stolen unlocked chips. As discussed in [6], such attacks are particularly critical against systems with FPGA components, which are more susceptible to reverse engineering and chip replacement. By facilitating activation after manufacturing, our locking strategy can be extended to protect multichip systems as illustrated in Figure 5. If some components are implemented using FPGAs, techniques from [2], [6] can be additionally used.

We now assume that each chip in the system is protected, as before, with a combination of test-aware locks and cryptography. In addition, the top-level assembly can generate random private and public keys, and supports cryptographic communication with each component IC (this functionality can be implemented using modern TPM chips or distributed throughout the system). The foundry tests each locked chip as before, and the components are assembled. At this point, activation at the system-level is performed by the end user. The activation request contains the system’s private key, along with those of individual chips. It is sent to the rights-holder of the system, who forwards it to rights-holders of individual component ICs. This prevents unauthorized unlocking of a whole system and allows rights-holders to determine whether their chips were activated individually or within a system.

**V. Empirical Validation**

Our experiments use circuits from the ISCAS89 and IWLS’05 [24] benchmark suites. We derive combinational circuits from sequential ones by removing sequential elements and exposing their inputs and outputs to the whole circuit. Test patterns are generated using ATALANTA [14] with default settings. IWLS benchmarks are structurally hashed using ABC [23]. Circuit simulation, lock insertion, and key combination attack algorithms are implemented in C++, compiled by g++4.6, and run on an Intel Core i7 workstation.

**A. Attacks using Logic Simulation**

We now demonstrate that locking strategies that impact test response are susceptible to the attack in Algorithm 1. In
particular, circuits that use XOR-based locking schemes can be unlocked by examining the gradient of output response.

As in [21], we randomly insert 32, 64, and 128 XOR key gates in each circuit (this is not a limitation of the experiment, but rather a representative strategy). Table I reports results for our attempts to determine CK from the provided test input and output response, as introduced in Algorithm 1. The results are averaged over 4 runs to produce detailed results, although in practice it is sufficient for a single run to succeed. The keys column indicates the number of different key combinations tried within our limits (1,000,000 combinations or two hours of simulation). For each key combination, test vecs is the number of test patterns applied.

We successfully unlock all circuits with 32 XORs, typically requiring only around 1000 key combinations. For circuits with 64 locks, we correctly determine the keys for all circuits except c880 and one run of usb_phy, as indicated by the column FRAC EXTRACTED. These circuits are more resistant to attacks due to their small size and the relatively large number of locking gates added. As outlined in Section III, the number of iterations in Algorithm 1 does not grow with the size of the circuit, but rather depends on the interactions between different locked signals. Also, the key combinations needed for usb_phy vary greatly as one run produced a circuit that was more resistant to attacks. Potentially, a more sophisticated simulation-based search strategy (rather than randomized hillclimbing) could better isolate interactions between different XOR key bits. When using 128 XOR locks, the interactions between them increase and hinder ability to discern the key combination. Despite this, we unlock spi in three of our trials. Furthermore, gradient descent quickly determines the majority of key bits for all circuits.

It is possible to find key combinations that satisfy the output response but do not unlock the circuit. This happens in over half of the circuits, as indicated by False Match. In other words, multiple global minima may exist. For a given circuit, if our gradient descent algorithm does not unlock the circuit after 10 random restarts, we analyze the common key bits between these key combinations (the randomness in our algorithm results in finding different minima). By removing the key bits common between these combinations, we effectively reduce the number of key bits to be examined. Then we re-solve the resulting smaller problem instance. As might be expected, larger circuits tend to have more global minima. Decreasing the observability of the locks increases the number of combinations that produce equal test response.

While our attack succeeds surprisingly often, less robust attacks may also cause heavy damage, especially when they can be repeated until success.

### B. Circuit Locking with MUXes

In this section, we validate the effectiveness of locking a circuit with MUXes so as to preserve test response. We demonstrate that there are several such transformations available in a circuit, and that, in general, these locks do not undermine fault diagnosability or create significant area overhead.

Table II describes the impact of randomly adding 64 logic locks using MUXes and logic covers. For the more realistic larger circuits, the gate area increase (shown under %OVERHEAD) was minimal. Unlike [3], we do not require additional logic to lock each scan chain. All circuits contain several locking opportunities (as indicated in %CANDIDATE). This column denotes the percentage of wires that have a candidate lock insertion to ensure that desired output bits are flipped on at least some inputs of interest. It is also important to note that

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>% OVERHEAD</th>
<th>% CANDIDATE</th>
<th>TIME(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>33.42</td>
<td>66.39</td>
<td>1.81</td>
</tr>
<tr>
<td>usb_phy</td>
<td>10.69</td>
<td>54.01</td>
<td>1.03</td>
</tr>
<tr>
<td>sasc</td>
<td>7.75</td>
<td>57.48</td>
<td>0.19</td>
</tr>
<tr>
<td>c3540</td>
<td>7.61</td>
<td>58.47</td>
<td>116.94</td>
</tr>
<tr>
<td>i2c</td>
<td>4.41</td>
<td>52.99</td>
<td>4.74</td>
</tr>
<tr>
<td>pci_spoiCtrl</td>
<td>3.67</td>
<td>73.36</td>
<td>12.28</td>
</tr>
<tr>
<td>systemcdes</td>
<td>1.42</td>
<td>66.97</td>
<td>6.94</td>
</tr>
<tr>
<td>spi</td>
<td>1.23</td>
<td>82.60</td>
<td>499.19</td>
</tr>
<tr>
<td>tv80</td>
<td>0.56</td>
<td>85.53</td>
<td>506.06</td>
</tr>
<tr>
<td>systemcaes</td>
<td>0.48</td>
<td>34.33</td>
<td>32.76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>MUX % locked outputs</th>
<th>MUX % bit diffs</th>
<th>XOR % locked outputs</th>
<th>XOR % bit diffs</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>47.36</td>
<td>1.76</td>
<td>54.09</td>
<td>28.08</td>
</tr>
<tr>
<td>usb_phy</td>
<td>21.09</td>
<td>0.39</td>
<td>31.88</td>
<td>8.55</td>
</tr>
<tr>
<td>sasc</td>
<td>19.73</td>
<td>0.35</td>
<td>25.69</td>
<td>8.32</td>
</tr>
<tr>
<td>c3540</td>
<td>17.00</td>
<td>1.18</td>
<td>86.08</td>
<td>27.12</td>
</tr>
<tr>
<td>i2c</td>
<td>15.67</td>
<td>0.12</td>
<td>25.29</td>
<td>6.42</td>
</tr>
<tr>
<td>pci_spoiCtrl</td>
<td>31.98</td>
<td>0.25</td>
<td>47.19</td>
<td>11.20</td>
</tr>
<tr>
<td>systemcdes</td>
<td>14.64</td>
<td>0.14</td>
<td>29.97</td>
<td>8.33</td>
</tr>
<tr>
<td>spi</td>
<td>9.49</td>
<td>0.02</td>
<td>42.41</td>
<td>4.45</td>
</tr>
<tr>
<td>tv80</td>
<td>3.24</td>
<td>0.004</td>
<td>28.65</td>
<td>1.37</td>
</tr>
<tr>
<td>systemcaes</td>
<td>7.75</td>
<td>0.03</td>
<td>23.91</td>
<td>2.60</td>
</tr>
</tbody>
</table>

| Average | 24.60                | 0.42           | 39.22                | 10.64          |
MUX-locking affects under 1% of output bits, whereas XOR locking typically affects around 10% of output bits as shown in the last two columns of the table.

**Circuit cracking.** MUX-based locking is resilient to simulation-based attacks that use test patterns since it preserves test response. However, an attacker with access to an activated circuit might determine chip keys using non-test inputs by comparing responses to the unlocked circuit. Table IV shows that MUX-based locking is less vulnerable to a random-simulation attack than XOR locking (but not invulnerable).

As in Table I, the results are averaged over 4 runs to produce detailed results, although in practice it is sufficient for a single run to succeed. The keys column indicates the number of different key combinations tried within our limits (1,000,000 combinations or two hours of simulation). As before, most circuits with only 32 locks are easily determined. The MUX-locked circuit appears harder to crack since fewer output differences are observable and the algorithm gets caught in local minima that do not unlock the design. Better input pattern selection might improve the effectiveness of this attack. In any event, our results further emphasize the need for a layered approach to chip security.

**Testability.** The addition of locking logic could limit the testability of the circuit. We briefly explore the coverage of gate output stuck-at faults for the MUX-locked designs (sans the locking logic, which is small). The first two columns of Table V show the fault coverage %COV and the number of untested faults #UNTEST in an unlocked circuit. The next two columns show results for the corresponding locked circuit with a random key combination. The coverage is nearly identical. This is due to (i) the number of locked sites being a small fraction of the design and (ii) synthesizing logic covers that include the locked logic such that it is still exposed to fault coverage. Notice that in some cases, like spi, the fault coverage actually improves. The added MUX logic can produce more sensitizing paths in the design, resulting in higher coverage.

Table V illustrates how the output response of a locked circuit compares to that of an unlocked circuit when a fault occurs. To aid circuit diagnosis, it is desirable for the locked circuit’s output response to match the unlocked circuit. While our locking algorithm does not explicitly consider the impact of locking on diagnosability, we observe that most locked circuits achieve equivalent output response for about 90% of the faults with a random key combination. Note that c880, the smallest circuit, only matches 60% of the faults. To improve diagnosability, the same test patterns can be applied with different key combinations, producing matching output response for at least one of the combinations. Five testing runs of c880 result in almost 90% of faults matching. The high percentage of matches with our locking strategy is due to (i) a relatively small amount of added locking logic and (ii) the locking strategy aiming to not change behavior under test in a functioning circuit.

**C. Use in Industrial Design-and-Test Flows**

Our proposed circuit locking technique is purely combinational and can be applied to combinational sections of larger designs. For example, in a ten-stage pipeline, it may be sufficient to lock only one or two stages, as long as they cannot be bypassed. Given sufficient control logic for the pipeline, it may offer an even better target for locking.

Industrial IC design flows are remarkable in their handling of multiple optimization objectives and constraints. When incorporating a new technique, there is danger of disturbing carefully optimized tradeoffs and the overall stability of the design process. In the context of circuit test, it is important to provision for yield optimization by process learning, which may alter or reduce the set of test patterns.

A commercially-viable combinational logic-locking strategy must incur minimal area overhead and negligible impact on the timing paths. The techniques we advocate offer several practical advantages. First, the number of locking sites is a small fraction of a large design — enough to scramble the output response while difficult to disable through mask modifications. Second, the large number of candidate covers shown in Table II reveal sufficient flexibility in common nets to avoid critical timing paths, which we used in our experiments (to preserve critical paths). In fact, heavily-optimized, deep logic designs with inserted control signals may enjoy many more candidate covers as the resulting logic signatures will be similar for most test patterns. The MUX insertion strategy is scalable as it only relies on the generation and comparison
TABLE IV

Extracting a 32 and 64 bit key in XOR and MUX-based locking using simulation with 1024 random input vectors for each circuit. Key indicates the number of keys tried. Frac Extracted is the fraction of random runs where the circuit was unlocked. Key is not reported when all trials fail to extract the chip keys. The timeout is two hours or one million combinations.

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>32 bit MUX</th>
<th>32 bit XOR</th>
<th>64 bit MUX</th>
<th>64 bit MUX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KEYS</td>
<td>FRAC EXTRACTED</td>
<td>KEYS</td>
<td>FRAC EXTRACTED</td>
</tr>
<tr>
<td>c880</td>
<td>430</td>
<td>1</td>
<td>13618</td>
<td>1</td>
</tr>
<tr>
<td>usb_phy</td>
<td>413</td>
<td>1</td>
<td>2193</td>
<td>1</td>
</tr>
<tr>
<td>sasc</td>
<td>186</td>
<td>1</td>
<td>83</td>
<td>1</td>
</tr>
<tr>
<td>c3540</td>
<td>1485</td>
<td>1</td>
<td>325</td>
<td>1</td>
</tr>
<tr>
<td>i2c</td>
<td>2060</td>
<td>1</td>
<td>1574</td>
<td>1</td>
</tr>
<tr>
<td>pci_spoi_ctrl</td>
<td>1028</td>
<td>1/4</td>
<td>3777</td>
<td>1</td>
</tr>
<tr>
<td>systemcdes</td>
<td>319</td>
<td>-</td>
<td>493</td>
<td>1</td>
</tr>
<tr>
<td>spi</td>
<td>10222</td>
<td>1/4</td>
<td>227</td>
<td>1</td>
</tr>
<tr>
<td>tv80</td>
<td>-</td>
<td>0</td>
<td>945</td>
<td>1</td>
</tr>
<tr>
<td>systemcaes</td>
<td>1753</td>
<td>1</td>
<td>125</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE V

Fault coverage of a circuit with test-aware locking (using gate output stuck-ats). The first two columns show coverage (%COV) and untested faults (%UNTEST) in unlocked circuits. The next two columns show coverage in locked circuits for random key assignments. The final five columns show how the output response of a locked circuit compares to an unlocked circuit in the presence of single faults in a circuit. A high percentage indicates that a high number of faults produce the same output in both locked and unlocked circuits. Checking the output response for different random key combinations increases the percentage of faults that have output response that matches an unlocked circuit.

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>Unlocked circuits</th>
<th>Locked circuits</th>
<th>% identical output response</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>%COV %UNTEST</td>
<td>%COV %UNTEST</td>
<td>(function of # random combinations)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>c880</td>
<td>100.0 0</td>
<td>99.5798 3</td>
<td>60.36 70.87 77.03 87.11 87.25</td>
</tr>
<tr>
<td>usb_phy</td>
<td>95.31 103</td>
<td>95.2641 104</td>
<td>86.38 92.30 92.71 95.99 96.58</td>
</tr>
<tr>
<td>sasc</td>
<td>98.65 41</td>
<td>98.458 47</td>
<td>91.63 96.10 97.54 98.16 98.46</td>
</tr>
<tr>
<td>c3540</td>
<td>96.93 101</td>
<td>96.8427 104</td>
<td>85.64 88.34 89.62 91.74 91.99</td>
</tr>
<tr>
<td>i2c</td>
<td>96.92 171</td>
<td>96.8457 175</td>
<td>94.65 96.61 97.17 97.67 97.69</td>
</tr>
<tr>
<td>pci_spoi_ctrl</td>
<td>87.84 832</td>
<td>87.9784 823</td>
<td>90.58 93.66 95.47 96.44 96.67</td>
</tr>
<tr>
<td>systemcdes</td>
<td>95.00 881</td>
<td>94.9819 885</td>
<td>84.68 91.32 94.81 95.51 95.84</td>
</tr>
<tr>
<td>spi</td>
<td>91.12 1753</td>
<td>91.1973 1738</td>
<td>92.54 93.37 94.04 94.68 95.49</td>
</tr>
<tr>
<td>tv80</td>
<td>90.78 4098</td>
<td>90.7896 4092</td>
<td>95.90 96.56 97.47 97.51 97.99</td>
</tr>
<tr>
<td>systemcaes</td>
<td>91.92 4207</td>
<td>91.9165 4211</td>
<td>94.99 96.67 98.85 99.09 99.22</td>
</tr>
</tbody>
</table>

TABLE VI

Impact of a larger pool of test vectors on the number of candidates for MUX locks. Locking sites were determined with each circuit having roughly twice the test vectors considered in Table II. %Candidate is the percentage of wires that have a candidate cover (the deviations with Table II are listed). %Fewer total Cands is the percentage of total candidates compared to the candidates found when using fewer test vectors. Total Cands is the total number of lock opportunities in the circuit.

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>%CANDIDATE</th>
<th>%FEWER TOTAL CANDS</th>
<th>TOTAL CANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>42.0 (-24.4)</td>
<td>26.9</td>
<td>576</td>
</tr>
<tr>
<td>usb_phy</td>
<td>33.7 (-20.3)</td>
<td>31.3</td>
<td>3228</td>
</tr>
<tr>
<td>sasc</td>
<td>23.3 (-34.2)</td>
<td>22.6</td>
<td>3261</td>
</tr>
<tr>
<td>c3540</td>
<td>45.2 (-13.3)</td>
<td>35.2</td>
<td>10173</td>
</tr>
<tr>
<td>i2c</td>
<td>34.1 (-18.9)</td>
<td>32.8</td>
<td>12567</td>
</tr>
<tr>
<td>pci_spoi_ctrl</td>
<td>51.5 (-21.8)</td>
<td>42.8</td>
<td>15404</td>
</tr>
<tr>
<td>systemcdes</td>
<td>34.3 (-32.7)</td>
<td>25.0</td>
<td>62361</td>
</tr>
<tr>
<td>spi</td>
<td>76.8 (-5.8)</td>
<td>32.3</td>
<td>215712</td>
</tr>
<tr>
<td>tv80</td>
<td>82.6 (-2.9)</td>
<td>41.2</td>
<td>842662</td>
</tr>
<tr>
<td>systemcaes</td>
<td>12.4 (-21.9)</td>
<td>19.3</td>
<td>44101</td>
</tr>
</tbody>
</table>

Optimization because a sizable variety of test patterns are supported, allowing one to add and remove individual patterns while process is optimized for yield. Thanks to numerous candidate locking sites, choosing locks that satisfy a very large number of test vectors upfront will allow testing different subsets of vectors without undermining the locking scheme. We validate this claim in Table VI where we show the number of wires with candidate MUX locks when doubling the number of test vectors applied to each circuit. Compared to Table II, the percentage of candidate wires (column one) decreases, but still represents a significant portion of the circuit. The next two columns indicate that despite fewer candidates, the number of opportunities far exceeds the number of locks needed.

VI. CONCLUSIONS

Counterfeiting poses serious yet hard-to-quantify risks to the semiconductor industry, whereas successful protection efforts may not lead to easily observable events. Thus, the industry and the research community face significant challenges but also enjoy a range of opportunities. To promote consistent implementation of countermeasures, recent standardization efforts, such as the IEEE 1149.1™-2013 standard [7], focus on chip authentication and detection of illegally produced chips. However, existing standards do not eliminate the exposure of IP to foundries, which can be viewed as a more fundamental challenge than detecting pirated chips.
Effective protection from IC piracy requires integrating several layers of security such as active metering and chip locking, as advocated in [1], [2], [8], [15], [18], [20], [21]. While the use of RSA-based preprocessing in EPIC [21] makes circuit-based key attacks less effective, even recent revisions of the EPIC protocol [15], [18] leave room for improvement. In this paper, we identify a weakness in combinational circuit-locking to attacks based on simulation and/or post-manufacturing test. To eliminate such weaknesses, we propose to restructure EPIC to support split test [3]. To avoid the overheads of [3], we use a novel, lightweight IC locking strategy invariant to test response. Compared to traditional combinational XOR-locking, we develop a MUX-locking scheme (compatible with LUT-based reconfigurable logic barriers from [2]). While not explicitly pursued in the paper, one way to view our contribution is through the identities $x \oplus y = x\bar{y} + \bar{x}y = (x \neq y)$. In other words, an XOR gate can be directly transformed into a 2-to-1 MUX gate with tied data inputs $y$ and $\bar{y}$. One could further untie the inputs by replacing $y$ with a fully equivalent internal signal $y''$ in the circuit, if such a signal is found. However, such opportunities are rare in highly optimized circuits. Hence we develop a more general technique to replace $y$ or $y''$ with only a partially equivalent internal signal $y''$.

Our detailed experiments on IWLS 2005 circuits demonstrate opportunities for proposed logic locks and confirm the scalability of this procedure to large IC designs. From the application perspective, the proposed restructuring of EPIC facilitates chip testing at the fab before the circuit is unlocked at a trusted facility by the owner of IP rights. This more advanced form of EPIC rules out several classes of attacks and makes contract IC manufacturing more secure. Considering possible attacks against our revised variant of EPIC, we note that a successful attack would also likely work against the previously known versions of EPIC [15], [21], for which attacks have been studied in depth and countermeasures are known [15], [18], [21]. We discuss extensions of proposed chip-activation protocols to multichip systems, which bring additional benefits.

REFERENCES


3Fabs with different business models (dedicated, pure play, etc) structure testing in different ways. However, yield learning is often an important part of the manufacturing process. To this end, our techniques allow some flexibility in the selection of test vectors. Such flexibility can be increased by inserting additional test points for controllability.


APPENDIX: LOCKING, OBFUSCATION, ENCRYPTION AND SCRAMBLING

Recent literature [9], [15], [18], [21] shows a surprising discord of terminology. While [21] and its conference version studied in [15], [18] describe proposed circuit modifications as combinational locking, the authors of [18] discuss obfuscation, and [10] uses both (following terminology introduced earlier in the FSM context). The analysis in [15] accurately follows original terminology. Yet, the work in [22] scrambles busses, and recently these methods have been summarized as circuit encryption, since [21] includes RSA cryptography.

Accounting for idiomatic usage in the English language, it is surprising to hear that “piracy is best fought with obfuscation,” given that locking and encryption are stronger words that carry concrete meanings and connotations. Representative concepts include house locks, scrambled cable TV signals, the obfuscated C contest, and Javascript obfuscators. To identify adequate terminology for [9], [10], [15], [18], [21], [22] recall that

- **locking**: preserves the structure, but renders the system temporarily unusable, until it is unlocked
- **obfuscation**: preserves the function, but renders the structure unintelligible, usually with no hope to restore it, hampering reverse engineering and modification
- **encryption**: is usually applied to data, rather than executable programs or live circuits, to render data useless.

---

**REFERENCES**


3Fabs with different business models (dedicated, pure play, etc) structure testing in different ways. However, yield learning is often an important part of the manufacturing process. To this end, our techniques allow some flexibility in the selection of test vectors. Such flexibility can be increased by inserting additional test points for controllability.
Like locking, encryption is reversible by definition, but it preserves neither function nor structure.

- **scrambling:** is close to encryption, but easier to undo.

Echoing the calls in [20], we hope that trustworthy terminology consistent with colloquial and technical usage of English words can be established in the field. This is particularly important to avoid confusion between new ideas and past misnomers.