TaintLock: Preventing IP Theft through Lightweight Dynamic Scan Encryption using Taint Bits*

Jonti Talukdar, Arjun Chaudhuri and Krishnendu Chakrabarty
Department of Electrical and Computer Engineering, Duke University, Durham, NC

2022 IEEE European Test Symposium (ETS) | 978-1-6654-6706-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ETS54262.2022.9810399

Abstract—**We propose TaintLock, a lightweight dynamic scan data authentication and encryption scheme that performs per-pattern authentication and encryption using taint and signature bits embedded within the test pattern. To prevent IP theft, we pair TaintLock with truly random logic locking (TRLL) to ensure resilience against both Oracle-guided and Oracle-free attacks, including scan deobfuscation attacks. TaintLock uses a substitution-permutation (SP) network to cryptographically authenticate each test pattern using embedded taint and signature bits. It further uses cryptographically generated keys to encrypt scan data for unauthenticated users dynamically. We show that it offers a low overhead, non-intrusive secure scan solution without impacting test coverage or test time while preventing IP theft.**

I. Introduction

The globalization of the integrated circuit (IC) supply chain poses significant risk to the security of intellectual property (IP). Of particular concern is the vulnerability to IP theft through attacks such as netlist reverse-engineering, counterfeiting, and IC overbuilding [1]. Logic locking offers protection against these attacks by locking the IP using combinational key-gates and/or sequential obfuscation states [2]. However, powerful SAT-based Oracle-guided attacks are capable of rapidly pruning the key-search space to extract the correct key [2].

Recent work has shown that pairing logic-locking schemes such as truly random logic locking (TRLL) with a secure scan chain can not only thwart Oracle-guided attacks but also achieve resilience against Oracle-free attacks [3]. Existing scan protection schemes either obfuscate scan data or block scan access altogether [4]. The strongest scan-obfuscation schemes achieve dynamic obfuscation, i.e., obfuscating the scan data dynamically for each pattern, using keys that are generated from an LFSR. However, such methods do not support scan authentication, thereby lacking the ability to determine if the scan chains are being accessed by a trusted user. Moreover, the use of linear obfuscation (using LFSRs) makes these methods vulnerable to attacks that remodel the dynamically obfuscated scan chains as a combinational logic-locked netlist [5], [6].

Pairing per-pattern authentication with dynamic scan obfuscation offers the ability to selectively validate the authenticity of each supplied pattern independently while obfuscating scan access for unauthorized users. This requires embedding the authentication information within the test pattern itself. In this work, we demonstrate the vulnerabilities associated with prior scan authentication methods under the latest (strongest) threat models used in the context of logic locking [7], [8]. We next propose **TaintLock**, a dynamic scan data authentication and encryption scheme that performs dynamic per-pattern authentication while achieving secure dynamic obfuscation (through encryption) for unauthorized users using *taint* bits embedded in the test pattern. TaintLock employs a challenge-response authentication mechanism using dynamically changing keys that are generated cryptographically and on the fly from taint bits embedded in each test pattern. The keys used for authentication and encryption change based on the pattern and on the order in which it is scanned in. As TaintLock is integrated with a low-overhead high output-corruptibility logic locking scheme (TRLL), it is resilient against not only Oracle-guided attacks but also against Oracle-free attacks, including removal/bypass attacks. In addition, its lightweight non-linear scan encryption scheme offers resilience against all existing attacks mounted on obfuscated scan chains, while incurring less than 0.2% area overhead for large circuits. The main contributions of this paper are as follows:

- We develop an attack that can extract, directly from test data, the scan authentication keys used in prior methods [7], [8].
- We present TaintLock, a dynamic scan data authentication and encryption scheme that uses substitution-permutation (SP) networks to authenticate each test pattern based on embedded taint and signature bits.
- We optimally allocate taint and signature locations for any set of test patterns and integrate it with the parametrized TaintLock architecture to generate authenticated test patterns.
- We show that TaintLock is adversarially indistinguishable, resilient against known- and chosen-plaintext attacks and any form of Oracle-guided attack, and secure against state-of-the-art scan de-obfuscation attacks.

II. Background and Related Prior Work

A. Secure Scan

Early scan-protection methods that rely on flipping scan bits or scrambling scan data were specifically developed to protect on-chip crypto cores from side-channel attacks, but not against IP theft [4]. Thus, these methods remain vulnerable to structural netlist reverse-engineering [2].

Secure scan is integrated with logic obfuscation to either obfuscate scan data or block scan access after the IP has been activated using functionally correct keys [9], [10]. Dynamic obfuscation is achieved by changing the keys dynamically using onboard pseudo-random number generators. However, as these methods use linear structures (embedded XOR gates and LFSRs), attacks such as DynUnlock [5] and ScanSAT [6] can apply the SAT attack to extract the obfuscation key (LFSR seed). Scan-obfuscation methods that utilize cryptographic macros to encrypt scan data suffer from large area overhead and test

*This research was supported in part by the Semiconductor Research Corporation (Task ID 2994.001) and the National Science Foundation under grant no. CNS-2011561.

times [11]. Scan-protection schemes that use secure cells and peripheral logic to block scan access in an activated IP [4] lack support for in-field functional debug, and they may also impact the timing profile of functional paths. These obfuscation schemes do not support per-pattern authentication, thereby limiting the ability to selectively screen untrusted end-users. Low-cost secure scan (LCSS) [7] supports static scan-data authentication by embedding the authentication key in dummy flip-flops. However, dummy flip-flops lead to increased test time and tester memory overhead. Another method [8] achieves dynamic scan data authentication by randomizing the test keys (SSTKR) that are generated using an on-board LFSR. The test keys are either embedded in dummy flip-flops or within the don't-care bits of each test pattern. However, this approach does not specify any selection criteria for embedding the test key across all test patterns.

B. Threat Model and Assumptions

The most pervasive threat model used in logic locking assumes that both the foundry and end-user are untrusted [2]. Thus, the attacker has access to the following: (1) *Locked Design:* An untrusted end-user or foundry can obtain the reverse-engineered (RE'd) netlist of the locked design (C_{lock}); (2) *Activated Chip:* The adversary has access to an activated chip (C_{or}) purchased from the open market; (3) *In-field Test Patterns:* The untrusted end-user has access to authenticated test patterns (P_{auth}) provided by the design house for in-field functional debug of the *activated chip*.

While the first two assumptions are common across all logic-locking techniques [2], the third assumption is relevant when a scan-authentication mechanism allows an end-user to apply authenticated test patterns to the activated device. These patterns may be supplied by the design house for in-field debug. Based on the above threat model, the attacker first uses the RE'd netlist to identify the type of security architecture used for logic and scan obfuscation. As the attacker has no control over the patterns included in P_{auth}, these patterns are not sufficient to mount an Oracle-guided attack. Thus, the attacker is forced to analyze P_{auth} to extract the scan authentication key to gain Oracle access. We do not consider invasive electrical probing attacks because these methods do not exploit the vulnerability of the obfuscation method itself, but rely on the shortcomings of the fabrication technology [1].

C. Reconstructing Authentication Keys from Test Data

We show that the patterns in P_{auth} are sufficient to reconstruct the static scan authentication key, k_{static}, in LCSS, and the set of dynamic authentication keys, $K_{dyn} = \{k_1, k_2, ..., k_n\}$, in SSTKR.

LCSS: Let the attacker choose a pattern $P \in P_{auth}$, where P_i denotes the i^{th} bit of P. Let b_j denote the bit location of the j^{th} dummy flip-flop in the scan chain. By analyzing C_{loc}, the attacker can generate the set $B_{loc} = b_j \ \forall j \in C_{loc}$, i.e., the set containing the bit locations of all dummy flip-flops in the design. It follows that k_{static} must contain $P_i \ \forall i \in B_{loc}$.

SSTKR: Let k_{seed} denote the LFSR seed used to generate the dynamic key $k_t \in K_{dyn}$. As the output of the LFSR is used as the authentication key, $k_{seed} = k_1$, i.e., the key used for the first pattern. The attacker can find all the bits in k_1 by analyzing the first pattern in P_{auth} along with B_{loc} as shown in the attack for LCSS. Given the i^{th} bit of the current authentication key, k_t^i, the corresponding key-bit in the subsequent sequence, k_{t+1}^i can be represented as $k_{t+1}^i = \bigoplus_{m=0}^{\lambda-1} c_m \cdot k_t^i$, where λ is the degree of the LFSR and c_m is the m^{th} feedback tap. The attacker can analyze C_{loc} to identify all feedback taps, thereby evaluating the tap coefficient $c_m, \forall m$ such that $0 \leq m \leq \lambda - 1$. Given k_{seed} and the LFSR structure, all keys in K_{dyn} can be computed by populating all the states of the LFSR.

As the authentication keys are not generated from specified bits embedded within the test pattern, the attacker needs only a single authenticated pattern to extract the authentication keys.

III. Description of TaintLock

A. Overview

TaintLock performs two major tasks, *authentication* and *encryption*. As both trusted and untrusted entities may have scan access, authentication is performed on a per-pattern basis. Scan responses for unauthenticated test patterns are encrypted.

Authentication: TaintLock employs a challenge-response mechanism to validate the authenticity of each test pattern. For each pattern P_i, there exists a corresponding signature value, S_i, computed using taint value, T_i, embedded in it. A Feistel structure-based substitution permutation (SP) network with a dynamic key schedule is used to compute $S_i = F_{auth}(T_i, K_i)$, where K_i is a dynamically changing key generated from a free-running reconfigurable block, and F_{auth} is a lightweight block cipher implemented by the SP-network. The same P_i can have a different S_i, depending on the order in which the pattern is scanned. Since an authenticated end-user is aware of the key-schedule, they can compute S_i for any P_i.

Encryption: TaintLock utilizes a stream cipher to dynamically encrypt scan data using XOR gates placed in the scan chain. The scan encryption key, K_{encr}, is generated by re-purposing F_{auth} as a stream cipher in counter mode [12]. Thus, $K_{encr} = F(T_i, K_i) \oplus \phi_i$, where ϕ_i is a dynamically changing key, also generated from a free-running reconfigurable block. The encrypted response is $R_{encr} = R_i \oplus K_{encr}$, where R_i is the original response.

While taint bits for each pattern are selected from the specified bits, signature bits are embedded within the test pattern by replacing carefully selected don't-care bits ($X's$). This allows for authentication support without impacting the test coverage. However, as the X locations used for embedding S_i may change depending on the pattern, we formulate a signature-bit selection scheme based on integer linear programming (ILP) that selects the minimum number of S_i locations to cover all ATPG-patterns in the test set. The number of bits used in T_i and S_i is determined by the architecture of F_{auth}, as discussed later in this section. Fig. 1 illustrates dynamic authentication using TaintLock. Note that the locations of the signature-bits change for each pattern.

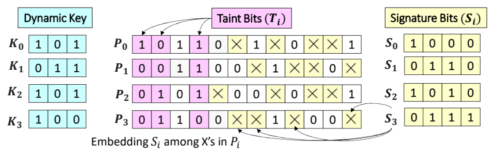
Dynamic Key
Taint Bits (T_i)
Signature Bits (S_i)
K_0 1 0 1
K_1 0 1 1
K_2 1 0 1
K_3 1 0 0
P_0 1 0 1 1 0 X 1 X 0 X X 1
P_1 0 0 1 1 0 0 X 1 X X 0 X
P_2 0 1 0 1 X 0 0 X 0 X X 1
P_3 0 1 1 0 0 X X 1 X 0 0 X
S_0 1 0 0 0
S_1 0 1 1 0
S_2 1 0 1 0
S_3 0 1 1 1
Embedding S_i among X's in P_i

Fig. 1: Dynamic scan authentication using taint bits.

B. System Architecture

TaintLock consists of the following components (Fig. 2a): (1) *Taint and Signature Cells:* These are the scan cells that store the taint and selection bits associated with each pattern. These cells are identical to regular scan cells; (2) *Selection MUXes:* Specify which signature cells contain the signature bits. If there exist r signature cells, with each signature S_i being q-bits wide, then there will exist at most q such $r:1$ MUXes. The MUX select signals are scanned into a separate register in parallel with the test pattern; (3) *Authentication Block (F_{auth}):* Contains the SP-network and computes the authentication signature using the taint bits and a dynamically generated key; (4) *Reconfigurable Block:* Responsible for generating the dynamic keys used for signature computation and scan encryption. It consists of four LFSRs with reconfigurable feedback. The LFSR seeds and feedback settings are supplied from tamper-proof memory and only known to authenticated users; (5) *Comparator:* Compares the authentication signature embedded in the test pattern with the one computed on-board using F_{auth}; (6) *Encryptor:* In case of mismatch between the computed and embedded signatures, it generates and stores the key, K_{encr}, used to encrypt the test response. During an authenticated test access, P_i is scanned in with the appropriate S_i embedded in it, leading to unobfuscated scan operation. As S_i can only be computed by a trusted user with access to the dynamic keys used for authentication and encryption, an untrusted user will be able to access only the encrypted responses for the supplied pattern.

C. Feistel Structure-based SP-Network Design

We design a lightweight authentication/encryption function based on Shannon's principles of *confusion* and *diffusion*. Similar to a Feistel structure-based SP-network, our function consists of five layers (Fig. 2b), alternating between permutation, key whitening, and non-linear diffusion layers. Although more such layers can be cascaded, we show (Sections IV–V) that five appropriately sized layers offer adequate security at low overhead. Existing crypto-cores used to encrypt scan data require multiple rounds of encryption at the cost of high area overhead and test time. Our architecture, being combinational in nature, offers a lightweight method to implement both authentication and encryption using dynamic keys without incurring any additional test time overhead. Our function supports two rounds of permutation (L1, L4) and key whitening (L2, L5) while performing one round of non-linear diffusion (L3) using a dynamic key schedule.

Layer Architecture and Key Sizes: The sizes of L1–L5 and k_1–k_4 are determined based on the number of signature bits (q bits per pattern) and the size of MUXes used in the permutation layers (2^{α}:1 MUX). Given q and α, the parametric values of the layer and key sizes are shown in Fig. 2. The permutation layers (L1, L4) use a network of 2:1 MUXes ($\alpha = 1$) to selectively propagate half the incoming bits to the next layer. Thus, each incoming bit has an equal probability of selection based on the dynamic key used to drive its corresponding MUX select line. The key whitening layers (L2, L5) are used to encrypt intermediate data and increase the size of the key space. The diffusion layer (L3) uses several 6×4 combinational S-Boxes to perform non-linear mapping between input and output. The number of S-Boxes is given by $q \times 2^{\alpha-2}$. As $\alpha = 1$ and we can only have an integer number of S-Boxes, q must be an even value. Finally, for $\alpha = 1$, the number of taint bits, $t = 6q$.

Key Schedule: The keys (k_1, k_2, k_3, k_4) driving the permutation and key-whitening layers are generated from on-chip LFSRs with reconfigurable feedback (RB). The key schedule is determined by the feedback configuration and seed of each RB. An RB of size λ consists of $\lambda - 1$ MUXes in its feedback path. The size of the seed for such an RB is $2\lambda - 1$. The key update frequency is determined by the clock signal of the RB. The clock input to the RB is driven by the $scan\_en$ signal, thereby updating the LFSR state, and consequently the keys, dynamically for each pattern (Fig. 2c). As the initialization seed of each RB can be changed for each IC/device, TaintLock supports a unique key sequence per device.

Scan Authentication & Encryption: Let k_1, k_2, k_3, and k_4 represent the dynamic keys used in different layers, $p_i(k_i, t)$ represent the output of the i^{th} permutation layer, and $s(t)$ represent the output of the substitution block. Then, the authentication signature is expressed as $s = k_4 \oplus p_2(k_3, s(k_2 \oplus p_1(k_1, t)))$. Scan responses are encrypted through XOR gates distributed uniformly across the scan chain. The encryption key is generated by re-purposing the block cipher to a stream cipher in counter mode, so that $k^{encr} = k1 \oplus k3 \oplus s$. The keys k_1 and k_3 are truncated to match s. If q is the size of encryption key, then the j^{th} bit of the encrypted response is encrypted with all the key bits following it, i.e., $r_j^{encr} = r_j \bigoplus_{i=j}^{q} k_j^{encr}$. A working example is presented in [?].

D. Taint and Signature Cell Allocation

We next formulate an ILP model that minimizes the number of signature cells, r, while covering all patterns in the test set. Let us consider a pattern set with n patterns, each being m bits wide. Without loss of generality, we assume a single scan chain. We use a binary decision variable x_{ij} such that $x_{ij} = 1$ if the j^{th} bit of the i^{th} pattern is selected, and $x_{ij} = 0$ if vice-versa. We next pre-compute the X-location matrix coefficients from the patterns in the test set, where $c_{ij} = 1$ if the j^{th} bit of the i^{th} pattern is an X and $c_{ij} = 0$ otherwise. The reward score is also computed for each bit position j, where $d_j = \sum_{i=1}^{n} c_j \; \forall 1 \leq j \leq m$. Larger the number of X's in a bit location, more likely is the selection of that location as a signature cell. We also determine χ, which is the minimum number of X's present among all patterns in the test set, also referred to as the X-budget. We impose the constraint that we must select exactly q X's, i.e., q signature bits per pattern, where $q \leq \chi$. Note that n such constraints exist, one for each pattern, $\sum_{j=1}^{m} x_{ij} = q \; \forall 1 \leq i \leq n$. Our objective is

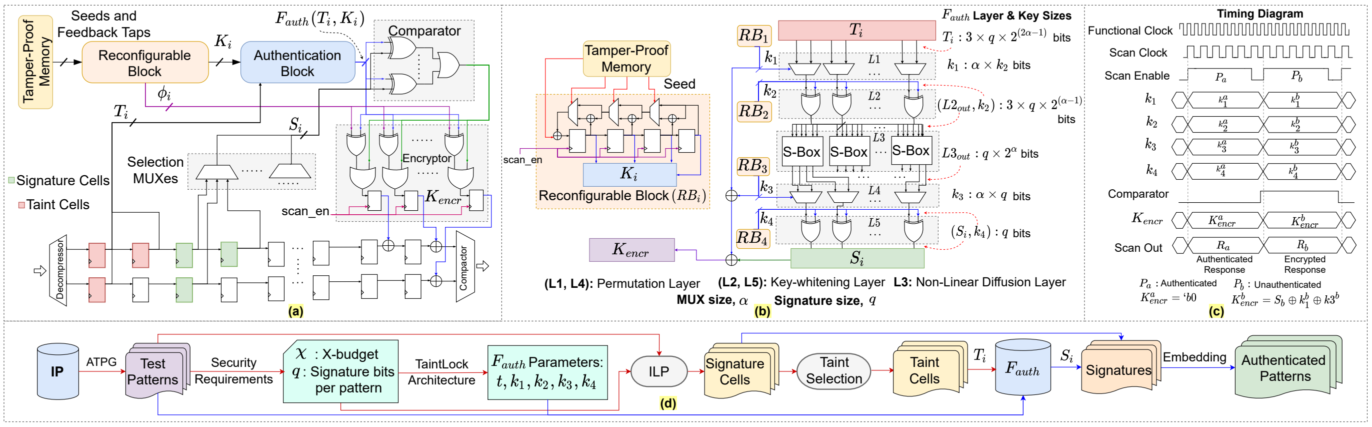

Fig. 2: (a) TaintLock system architecture. (b) SP-network used in authentication block (F_{auth}). (c) Timing diagram for authentication and encryption using Taintlock. (d) Methodology to integrate TaintLock with a generic IP and generate authenticated test patterns.

TABLE I: Evaluating the number of signature cells required to support different TaintLock versions across CEP benchmarks.

IP	Pattern Count	Pattern Length	Test Coverage (%)	χ	r (from ILP) q=12	r (from ILP) q=16
FIR	160	448	98.77	66	20	25
IIR	183	672	86.94	219	160	197
SHA256	306	1040	99.86	267	27	32
AES192	598	6854	98.97	4504	20	25
RocketCore	2183	43140	95.53	39856	23	23

q: # of signature bits, r: # of signature cells, t: # of taint cells, $t = 6q$, thus $t = 72$ and 96 for 12 and 16 bit signatures, respectively, χ: X-budget.

to maximize the reward function $\sum_{j=1}^{m} d_j \cdot \sum_{i=1}^{n} x_{ij} c_{ij}$. This objective function ensures that bit locations with larger number of X's per pattern are given priority. It also ensures that all X-bits in a column are selected during scoring.

After the r signature cells are allocated, the remaining $(m - r)$ cells associated with the specified bits in the test pattern are considered for taint cell allocation. The number of taint locations (t) is fixed for all patterns in the test set. It is also ensured that $(m - r) \geq t$, where $t = 6q$. For each remaining scan cell location, k, let a_k and b_k denote the number of 0's and 1's occurring at that location across all test patterns, respectively. As taint bits are used as inputs to the authentication block, we select taint cells that have similar numbers of 1's and 0's. This ensures that the input to the authentication block is balanced. Thus, for each k, we evaluate $\Delta_k = |a_k - b_k|$ and $\mu_k = (a_k + b_k)/2$. Subsequently, we choose k with the highest μ_k and lowest Δ_k, thereby ensuring a balanced selection of taint bits, leading to better authentication performance for the block cipher. Fig. 2d shows the overall methodology. We evaluate the ILP model on several IPs from the Common Evaluation Platform (CEP) [1] protected using different configurations of F_{auth}. We compute the number of signature cells for IPs containing F_{auth} with $q = 12$ bits and $q = 16$ bits, respectively. These results are shown in Table I.

IV. Security and Testability Analysis

A. *Encryption Quality and Adversarial Indistinguishability*

As TaintLock supports dynamic per-pattern authentication and encryption, the same pattern can have different signatures and encrypted responses based on the dynamic key schedule. If a test set contains n patterns, each pattern will have at least n different signatures and encrypted responses, respectively. For a given signature size and key schedule, we evaluate the *authentication quality* of a pattern P_i by analyzing the distribution of 1's and 0's associated with each signature bit location j $(1 \leq j \leq q)$ across all n signatures. For each P_i, let the pair $(\mu^j_{i,1}, \mu^j_{i,0})$ denote the average fraction of 1s and 0s associated with the j^{th} signature bit across all n signatures. We next evaluate the mean $(\mu^{sign}_{i,1}, \mu^{sign}_{i,0})$ and standard deviation $(\sigma^{sign}_{i,1}, \sigma^{sign}_{i,0})$ of these j ordered pairs. Here, $\mu^{sign}_{i,1}$ is the average fraction of 1's in all the signature bits for all n signatures. Similarly, $\sigma^{sign}_{i,1}$ is the standard deviation in the average fraction of 1's across the j signature bits for all n signatures. Note that because $\mu^{sign}_{i,1} = 1 - \mu^{sign}_{i,0}$, it follows that $\sigma^{sign}_{i,1} = \sigma^{sign}_{i,0}$. A value of $\mu^{sign}_{i,*}$ close to 0.5 and a very low value of $\sigma^{sign}_{i,*}$ are indicative of a balanced authentication function. Next, we use the Hamming distance between a scan response R_i, and its encrypted counterpart $R_{encr,i}$, to evaluate *encryption quality*. Given R_i, let $R^j_{i,encr}$ represent its j^{th} encrypted response such that $1 \leq j \leq n$. Then, μ^{HD}_i represents the average Hamming distance between the true response and all other n encrypted responses. A Hamming distance of 0.5 is indicative of a balanced encryption function. The authentication quality $(\mu^{sign}_{i,*}, \sigma^{sign}_{i,*})$ and encryption quality (μ^{HD}_i) values are averaged across all patterns in the test set and presented for IPs in Table II. We observe that TaintLock performs balanced authentication and encryption.

Using Table II, we show that TaintLock is adversarially indistinguishable, i.e., an adversary can learn no partial information about the plaintext, R_i, given the encrypted ciphertext, $R_{i,encr}$. For a given private-key encryption scheme Π, consider an experiment $PrivK_{\mathcal{A},\Pi}$ in which a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ outputs two messages m_0, m_1 out of which one of them is encrypted at random. We define $PrivK_{\mathcal{A},\Pi}$ as follows: (1) An adversary $\mathcal{A}$ is given an input 1^n and generates a pair of output messages m_0, m_1 with $|m_0| = |m_1|$. (2) A key is generated at random by running $Gen(1^n)$, and either message m_0 or m_1 is chosen at random depending on a uniform bit $b \in \{0, 1\}$. The challenge ciphertext $c \leftarrow Enc_k(m_b)$ is generated. (3) $PrivK_{\mathcal{A},\Pi} = 1$ if $\mathcal{A}$ identifies correct m_b for given c. We next introduce the following definitions [13].

TABLE II: Evaluating the authentication and obfuscation quality for 12- and 16-bit TaintLock versions across CEP benchmark IPs.

IP	12-bit			16-bit		
	$\mu_{avg,1}^{sign}$	$\sigma_{avg,1}^{sign}$	μ_{avg}^{HD}	$\mu_{avg,1}^{sign}$	$\sigma_{avg,1}^{sign}$	μ_{avg}^{HD}
FIR	0.506	0.030	0.485	0.497	0.036	0.509
IIR	0.494	0.049	0.480	0.505	0.032	0.492
SHA256	0.501	0.026	0.486	0.501	0.033	0.487
AES192	0.502	0.022	0.488	0.503	0.018	0.496
RocketCore	0.501	0.010	0.495	0.495	0.012	0.495

Definition 1. *A function f from the natural numbers to non-negative real numbers is negligible if for every positive polynomial p, there is an N such that $\forall\, n > N;\ f(n) < \frac{1}{p(n)}$.*

Definition 2. *A given private-key encryption scheme Π is considered adversarially indistinguishable if for a PPT adversary $\mathcal{A}$, there is a negligible function $negl$ such that for all n, $P[PrivK_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + negl(n)$.*

Based on the above definitions, we note that $\mu_{avg,1}^{HD}$ is an indirect measure of the adversarial indistinguishability of an encryption scheme. From Table II, we observe that $\mu_{avg,1}^{HD} \approx 0.5$ for all the IPs . This implies that a scan response bit has an equal probability of being encrypted or not, thereby indicating that TaintLock is adversarially indistinguishable.

B. Security Analysis

TaintLock offers a decentralized and non-intrusive method for cryptographically authenticating and encrypting scan access, thereby preventing Oracle access for untrusted users. We pair it with truly random logic-locking (TRLL), promising low overhead and high output corruptibility while being immune to existing netlist analysis-based removal attacks [3]. We quantify the resilience of TaintLock and demonstrate that it is secure against cryptanalysis attacks, Oracle-guided attacks, and Oracle-free attacks including state-of-the-art scan deobfuscation attacks.

1) Brute-force Resilience

Given q signature bits per pattern, an attacker can attempt to guess S_i for a given P_i. The probability of success $P(S_i|P_i)$ for the attacker is $1/2^q$. However, due to dynamic per-pattern authentication, S_i will change not only based on the pattern itself but also based on the order of patterns in the test set. For n patterns, the attacker must try all $n \times 2^q$ signature combinations to find the correct S_i for each P_i. Furthermore, this does not *unlock* the IP as the attacker will be forced to repeat this process in case of a new pattern set. Thus, the attacker is forced to guess the seed used to generate the correct keys, (k_1, k_2, k_3, k_4), to unlock the IP. If λ_i is the size of the i^{th} key, then the size of the RB seed used to generate that key is $2\lambda_i - 1$. Brute-force attack resilience is quantified by the number of attempts required by the attacker to uncover the correct key. For TaintLock, this is given by $t_{bf} = 2^{\lambda_{sum}}$, where $\lambda_{sum} = 2(\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4) - 4$ is the total seed size. The attack effort is prohibitively high for the different brute-force approaches discussed above (For e.g., $t_{bf} = 3.92 \times 10^{56}$ for $q = 12$).

2) Cryptographic Attacks

Given access to authenticated debug patterns, P_{auth}, the attacker may apply known-plaintext attacks [14]. However, TaintLock is resilient to both known-plaintext (KPA) and chosen-plaintext attacks (CPA). If the attacker can collect 2^m plaintext-ciphertext pairs, i.e., the number of patterns in P_{auth} is 2^m, the resilience of the system will be reduced to $t_{plaintext} = 2^{\lambda_{sum}-m}$, where $\epsilon = \lambda_{sum} - m$. However, given the small size of P_{auth}, $m << \lambda_{sum}$. This makes such attacks infeasible in practice. Additionally, given that TaintLock obscures the circuit test responses, the attacker does not have control over the original patterns being scanned out of the design, making chosen-plaintext attacks also infeasible.

3) Oracle-guided and Oracle-free Attacks

We show in [15] that a dysfunctional IP guarantees resilience against all forms of Oracle-guided attacks. As TaintLock encrypts test response data thereby making TaintLock resilient against these attacks.

A recent class of Oracle-free attacks called scan deobfuscation attacks, such as DynUnlock and ScanSAT, aim to extract the LFSR seed used for dynamic scan obfuscation. These attacks are successful because: (1) the attacker knows the LFSR feedback polynomial from the RE'd netlist, thereby allowing them to model the pattern transformation as a function of the LFSR seed; (2) the absence of non-linear blocks in the obfuscation logic that lack a combinational logic equivalent. TaintLock addresses both these weaknesses. The use of LFSRs with reconfigurable feedback (RBs) makes it impossible for the attacker to extract the correct feedback polynomial used for key generation from the RE'd netlist. Moreover, the attacks described above cannot model the encryption blocks used in the SP-network (permutation layers and S-boxes) into a combinational design logic-locked using the RB seed due to their non-linear nature [5], [6]. Finally, resetting the IP clears data in scan cells including signature values, triggering encryption.

C. Implications on Testability

Manufacturing Test: TaintLock supports scan-based manufacturing testing in an untrusted setting using dummy keys (deactivated IP without tamper-proof memory loaded). As taint bits are extracted directly from ATPG patterns and signature bits are embedded among the $X's$ in the pattern, there is no impact on test coverage. Since the scan chains remain unmodified, there is no impact on test time. Moreover, TaintLock supports for two-pattern delay tests such as launch-on-shift (LOS)/launch-on-capture(LOC) in encrypted mode. For these tests, the first pattern is applied with the embedded signature. However, the second pattern is obtained either from the circuit's response (LOC), or by shifting a single bit (LOS). Due to the lack of control over the bits in the second pattern, it is unlikely that the correct signature will be embedded for that pattern. Thus, the scan responses for LOC/LOS will be encrypted. However, because TaintLock uses a symmetric cryptographic scheme, a trusted user can decrypt the scan responses using the encryption key. Furthermore, the use of unique keys per pattern ensures that the encrypted response can be uniquely mapped to the expected response. This allows the user to pre-compute the expected fault-free encrypted responses for LOC/LOS tests.

IP Activation: After manufacturing test, the IP is shipped to a trusted site for activation. During activation, the design house may also apply other tests to ascertain the integrity of the IP. The objective of this step is to identify tampering within the IP, including scan chains, control logic, or hardware Trojans.

Test Compression: Unlike scan obfuscation methods that require bypassing of test compression [9], TaintLock is compatible with test compression. Although it fills carefully selected don't-care bits in the test pattern, it is clear from the results in Section III-C that even after the signature bits are embedded, many don't-care bits remain in each pattern to support compression. We were able to compress test patterns with embedded signature bits using Embedded Deterministic Test (EDT) [16].

V. Overhead Analysis

We evaluate the power, performance, and area (PPA) overhead of TaintLock integrated with different IPs in the CEP benchmark. We consider the 12-bit and 16-bit signature versions of TaintLock as they offer adequate security against known attacks. This is integrated with a 128-bit TRLL logic-locked netlist. All the benchmarks are synthesized at 500 MHz with Synopsys Design Compiler using the Nangate 45 nm standard cell library. We then run the place-and-route flow, together with parasitic extraction, using Cadence Innovus to generate the final design layout. We evaluate the impact on circuit timing and power consumption using Cadence Tempus.

The PPA results for five benchmark circuits (Table III) show that the area and timing impact of TaintLock are negligible, and the purely combinational authentication function does not impact the scan frequency and hence supports at-speed testing. As the TaintLock architecture has a fixed area for the given security configuration, both the area and power overhead decrease with an increase in the size of the circuit. Table IV compares TaintLock with related methods.

We also compare TaintLock with a lightweight cryptographic encryption scheme that may be re-purposed for scan authentication (PRESENT) [17]. Unlike PRESENT, TaintLock provides security against not only known- and chosen-plaintext attacks, but also against Oracle-guided attacks. Moreover, it does so at over 3× lower area overhead; see Table III. In addition, PRESENT requires multiple rounds of encryption that take up significant time, thereby adversely impacting the scan shift frequency. PRESENT also has a fixed block size, thereby is not parameterizable and hence not useful for small IP. Our results show that a lightweight scan authentication scheme using taint bits, such as TaintLock, can successfully offer security against state-of-the-art attacks.

VI. Conclusion

We have presented the vulnerabilities associated with existing scan data authentication techniques. We have proposed TaintLock, a low overhead method that uses taint and signature bits embedded within the test pattern for scan data authentication and encryption. We have shown the resilience of TaintLock against Oracle-guided attacks, Oracle-free attacks, and scan deobfuscation attacks.

TABLE III: PPA overheads associated with TaintLock and PRESENT.

IP	Security configuration	Cell area (μm^2)	Wirelength (μm)	Power consumption (mW)	Δ_{crit} (ns)
FIR	Baseline	6010.5	28103.1	2.88	1.98
	Δ_{prsnt} (%)	30.64	29.03	17.69	0.71
	Δ_{12} (%)	6.38	10.47	3.13	0.22
	Δ_{16} (%)	7.5	14.65	3.94	0.72
IIR	Baseline	9864.6	46255.5	7.02	1.98
	Δ_{prsnt} (%)	18.66	17.64	7.26	8.78
	Δ_{12} (%)	4.19	6.07	3.52	0.27
	Δ_{16} (%)	4.83	11.78	4.24	0.42
SHA256	Baseline	15077.6	114915.1	7.26	1.98
	Δ_{prsnt} (%)	12.21	7.1	7.02	0.05
	Δ_{12} (%)	2.64	5.42	2.16	0.12
	Δ_{16} (%)	3.57	7	4.4	0.05
AES192	Baseline	267278.4	3794342.7	71.91	1.99
	Δ_{prsnt} (%)	0.69	0.21	0.71	0.45
	Δ_{12} (%)	0.13	1.66	0.04	0.6
	Δ_{16} (%)	0.16	2.61	0.54	0.13
RocketCore	Baseline	375386.6	2979859.8	71.1	1.68
	Δ_{prsnt} (%)	0.49	0.28	0.72	0.45
	Δ_{12} (%)	0.11	2.05	0.06	0.85
	Δ_{16} (%)	0.13	2.03	0.54	0.42

Δ_{prsnt}: percentage overhead associated with the insertion of PRESENT; $\Delta_{12}(\Delta_{16})$: percentage overhead associated with the insertion of 12-bit (16-bit) signature version of TaintLock; Δ_{crit}: critical path delay.

TABLE IV: Comparing prior secure scan methods with TaintLock.

Metrics	LCSS [7]	SSTKR [8]	SEBC [11]	DOSC [9]	TaintLock
Scan deobfuscation attacks [5], [6]	✗	✗	✓	✗	✓
Removal Attacks [2]	✗	✗	✗	✗	✓
Dynamic Per-pattern Authentication	✗	✓	✗	✗	✓
Support Response Encryption	✗	✗	✓	✗	✓
Support LOC/LOS	✗	✗	✓	✓	✓
Support Test Compression	✗	✗	✓	✗	✓
Total Test-time Overhead (cycles)	$p \times d$	$p \times d$	$4N$	None	None

p: Pattern count; d: # of dummy flops; N: Round register size in [11].

References

[1] B. Tan et al., "Benchmarking at the frontier of hardware security: lessons from logic locking," *arXiv:2006.06806*, 2020.

[2] A. Chakraborty et al., "Keynote: A disquisition on logic locking," *IEEE TCAD*, vol. 39, pp. 1952–1972, 2019.

[3] N. Limaye et al., "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE TCAD*, 2020.

[4] K. Azar et al., "From cryptography to logic locking: A survey on the architecture evolution of secure scan chains," *IEEE Access*, 2021.

[5] N. Limaye et al., "DynUnlock: unlocking scan chains obfuscated using dynamic keys," in *DATE*, 2020, pp. 270–273.

[6] L. Alrahis et al., "ScanSAT: Unlocking static and dynamic scan obfuscation," *IEEE TETC*, 2019.

[7] J. Lee et al., "A low-cost solution for protecting IPs against scan-based side-channel attacks," in *IEEE VTS*, 2006.

[8] M. A. Razzaq et al., "SSTKR: Secure and testable scan design through test key randomization," in *IEEE ATS*, 2011, pp. 60–65.

[9] D. Zhang et al, "Dynamically obfuscated scan for protecting IPs against scan-based attacks throughout supply chain," in *IEEE VTS*, 2017.

[10] R. Karmakar et al., "A scan obfuscation guided design-for-security approach for sequential circuits," *IEEE TCAS*, pp. 546–550, 2019.

[11] M. Da Silva et al., "Preventing scan attacks on secure circuits through scan chain encryption," *IEEE TCAD*, vol. 38, no. 3, pp. 538–550, 2018.

[12] M. Dworkin, "Block cipher modes of operation: The CCM mode for authentication and confidentiality," *NIST Special Publication*, 2003.

[13] A. Sahai and B. Waters, "How to use indistinguishability obfuscation: deniable encryption, and more," in *Proc. Symp. Theory Comp.*, 2014.

[14] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Workshop on the Theory and App. of Cryptographic Tech.*, 1993, pp. 386–397.

[15] J. Talukdar et al., "A BIST-based dynamic Obfuscation scheme for resilience against removal and Oracle-guided attacks," in *ITC*, 2021.

[16] J. Rajski et al., "Embedded deterministic test," *IEEE TCAD*, vol. 23, no. 5, pp. 776–792, 2004.

[17] A. Bogdanov et al., "PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2007.