ALMOST: Adversarial Learning to Mitigate Oracle-less ML Attacks via Synthesis Tuning

Animesh B. Chowdhury*, Lilas Alrahis†, Luca Collini*, Johann Knechtel†,
Ramesh Karri*, Siddharth Garg*, Ozgur Sinanoglu†, Benjamin Tan‡
*New York University, USA, †New York University Abu Dhabi, UAE, ‡University of Calgary, Canada

***Abstract*—Oracle-less machine learning (ML) attacks have broken various logic locking schemes. Regular synthesis, which is tailored for area-power-delay optimization, yields netlists where key-gate localities are vulnerable to learning. Thus, we call for security-aware logic synthesis. We propose ALMOST, a framework for adversarial learning to mitigate oracle-less ML attacks via synthesis tuning. ALMOST uses a simulated-annealing-based synthesis recipe generator, employing adversarially trained models that can predict state-of-the-art attacks' accuracies over wide ranges of recipes and key-gate localities. Experiments on ISCAS benchmarks confirm the attacks' accuracies drops to around 50% for ALMOST-synthesized circuits, all while not undermining design optimization.**

I. Introduction

Setting: Machine learning (ML)-based structural attacks like [1], [2] can decipher the key-bits of logic-locked circuits without access to an *oracle*.[1] Such attacks explored and exploited the impact of logic synthesis on structural properties of locked netlists. For example, even with bubble pushing – which is a logic synthesis technique used by locking schemes to obfuscate the relationship between key-bits to key-gates – some discernible structures can arise. This is because, even though logic synthesis is complex and uses carefully devised power, performance, and area (PPA) heuristics, it is deterministic, yielding predictable structural transformations around key-gates, which can be learned on to infer key-bits.

Prior Art: Truly random logic locking (TRLL) [3] and UNSAIL [4] are assumed to thwart ML attacks – both schemes remain unbroken as of today. However, both schemes also rely on specific design properties, like certain structures of gates to be present in sufficient numbers in the appropriate places, restricting their general applicability. Both schemes operate on netlists handled by commercial synthesis tool, which are operating as block-box modules that optimize for PPA, not for security. Thus, both schemes needed to "work around" those tools in a smart way to enable resilient locking schemes.

Challenge: There is a need for logic-locking techniques that are a) resilient against ML-based, oracle-less attacks exploiting structural properties of locked netlists, b) generally applicable to various designs, and c) integrate with logic synthesis and do not undermine design optimization offered by synthesis.

Scope: We study the causal nexus between logic locking, logic synthesis, and resilience to ML-based attacks.

For the first time, we propose a logic-locking methodology that drives logic synthesis to makes the locked circuits resilient against ML-based attacks. Unlike TRLL or UNSAIL, we achieve resilient against ML-based attacks without dedicated locking strategies or crafted key-gate structures. We demonstrate that we can use the simplest and vulnerable locking scheme, i.e., random logic locking (RLL) [5], and still obtain ML-resilient designs.

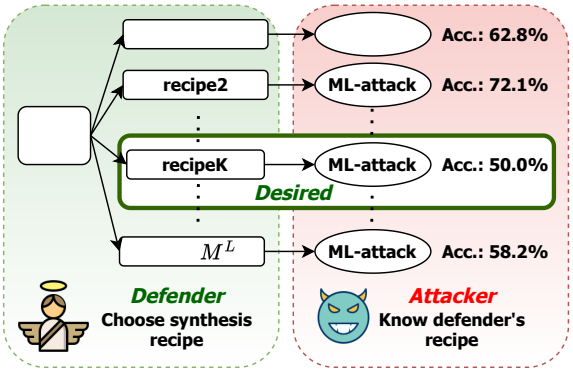

Fig. 1. Different synthesis recipes have different impacts on the resilience of a logic-locked design in the context of ML-based attacks.

However, achieving theis resilience requires tackling practical challenges. As shown in Fig. 1, even when starting from the same locked design, varying synthesis recipes will impact the resiliency against ML attacks differently. Finding synthesis recipes optimized for a specific metric, e.g., the accuracy of ML attacks[2], is Σ_p^2-hard [6]. Thus, our study addresses two key research challenges (RCs).

1) **RC1:** Can we explore the synthesis search space to find some recipe(s) that provide resilience against various state-of-the-art (SOTA) ML attacks?
2) **RC2:** Given any recipe, can we efficiently quantify its ML resiliency without running the various SOTA attacks?

Contributions: ALMOST is a security-centric framework to generate ML-resilient logic-locked designs using synthesis. It does not relying on security promises of the locking scheme. This is important as the value of such promises are subject to synthesis in the first place. Our contributions include:

1) A framework for security-aware synthesis-recipe generation using simulated annealing (SA). These recipes produce netlists that are ML attack resilient (RC1).

[1]An oracle provides black-box access to the correct functionality, e.g., through a functional chip obtained from the open market.

[2]Accuracy is an established ML metric. It is defined as (# correctly predicted key-bits) / (total # key-bits, i.e., key-size). For a key-size of 128 bits, when an attack correctly predicts 64-of-the-128 bits (and the remaining 64 bits are either incorrectly inferred or not inferred at all), the accuracy is 0.5 or 50%. As a defender, accuracy of 50% is desired. This equates to the attack is no better than a random guess.

2) An adversarially trained model to predict resiliency of locked designs post-synthesis against ML attacks (RC2).
3) Public release of source codes and artifacts.

Key Results: For ALMOST-synthesized ISCAS85 benchmarks, using RLL as an exemplary scheme that is otherwise fully vulnerable, SOTA ML attacks [1], [7] and non-ML structural attacks [8] are reduced to random guessing. It is important to note that ALMOST generates synthesis recipes that enforce/maintain such resiliency even when the ML-empowered attacker is fully aware of the recipe. At the same time, PPA overheads are marginal.

II. Oracle-Less Attacks on Logic Locking

Recent works show the potential of ML models for advanced attacks in the oracle-less scenario [2], [9], [1], [7].

SAIL [2] tackles XOR/XNOR locking. It learns the local, synthesis-induced changes around those key-gates, reverting the binding of key-bits: before bubble pushing, XOR is bound to '0' and XNOR to '1'. *SnapShot* [9] works by learning the local changes caused by key insertion and synthesis itself.

While SAIL and SnapShot use classic tensor-based models, *OMLA* [1] uses graph neural networks (GNNs), as a natural representation of gate-level netlists. OMLA extracts *localities*, i.e., the sub-circuit structures around key-gates, represents them as sub-graphs, and passes them to the GNN to predict the corresponding key-bits through subgraph classification.

These attacks employ self-referencing to train their ML models. The attacks apply some form of re-locking and re-synthesis to generate training datasets. The attacks know the synthesis recipe used by the defender.

SCOPE [7] extracts synthesis reports, e.g., on area and power, to decipher key-bits. Unlike the attacks above, SCOPE follows an unsupervised approach for learning the correlation between key-bit values and synthesis features.

Redundancy attack [8] is a non-ML structural attack. It assumes that the original design is fully testable. Accordingly, the attack infers the key-bits as those assignments that cause fewer untestable faults in the locked circuit.

III. ALMOST framework

We formulate security-aware synthesis as an optimization problem where the objective is to ensure that locked designs remain resilient post-synthesis. Thus, we search for synthesis recipes that structure the netlists such that SOTA attacks, be they ML or non-ML ones, can achieve only ∼50% accuracy.

Without loss of generality (wolog), we use the open-source synthesis suite *yosys* and *ABC* [10]. Unlike commercial tools, this suite allows for fine-grain tuning of synthesis recipes, also accounting for user-defined objectives as needed, i.e., for accuracy in this work. Like any synthesis tool, this suite takes register transfer level (RTL) and converts it to a gate-level netlist, by performing technology mapping using a technology library. Synthesis transformation and optimization steps are implemented using the and-inverter-graph (AIG) representation.

Next, we formulate the problem in detail. Then, we discuss the framework stages, which are also illustrated in Fig. 3.

A. Problem Formulation

General Problem. In a synthesis recipe $\mathbf{S}$ with $\mathbf{M}$ transformation steps, the number of recipes of length $\mathbf{L}$ is $\mathbf{M^L}$. Security-aware recipe generation can be formulated as:

$$\underset{\mathbf{S}}{\operatorname{argmin}} \, |Acc_{M_A^S}(\mathcal{G}(\mathbf{AIG}, \mathbf{S})) - 0.5| \tag{1}$$

where $\mathcal{G}$ is the synthesis function defined as $\mathcal{G} : \mathbf{AIG} \times \mathbf{S} \longrightarrow \mathbf{AIG}$. Accuracy (Acc) describes the prediction accuracy of an attacker's model M_A^S that is built-up using recipe $\mathbf{S}$. Acc evaluates the resilience of the locked netlist introduced through synthesis using $\mathbf{S}$; Acc values around 50% are the target.

Note that we do not explicitly consider PPA in the above formulation, but only security. Nevertheless, we observe empirically that (i) PPA optimization can follow-up on our security-aware synthesis without undermining the netlists' resilience and (b) PPA overheads are, on average, only marginal.

Model M_A^S and Naive Approach. We aim to solve the optimization problem (Eq. 1) using a black-box solver, wolog SA in this work. The challenge here is that, to accurately evaluate the effects of the attacker's model M_A^S,[3] we would need to separately train models M_A^S for the varying recipes S across every iteration. Fig. 1 outlines such naive approach, which seems computationally expansive.

Challenge for Transferability of M_A^S. While M_A^S models are demonstrated to predict key-bits very well, their accuracy would drop for locked designs that are synthesized using any other recipe, say S', i.e., accuracy(M_A^S) $\leq$ accuracy($M_A^{S'}$). This is because the range of structural transformations induced by S may not fully match with the range induced by S'.

To confirm this intuition, we run an experiment on the ISCAS85 circuit c5315 where we trained two attack models, $M_A^{S_1}$ and $M_A^{S_2}$, with training data covering the locked netlist T_{S_1} and T_{S_2} as synthesized using recipes S_1 and S_2, respectively. We observe that accuracy(T_{S_1},$M_A^{S_1}$) = 57.52%, whereas accuracy(T_{S_1},$M_A^{S_2}$) = 52.27%. Similarly, accuracy(T_{S_2},$M_A^{S_1}$) = 53.78% and accuracy(T_{S_2},$M_A^{S_2}$) = 58.91%. (We report the setup and more detailed results in Sec. IV-B.)

These accuracy mismatches clearly show challenges for transferring M_A^S for the evaluation of netlists locked with other recipes, thus re-iterating the need for training unique models for each iteration of optimization and exploration of the synthesis search space (Fig. 1). However, as indicated, training of separate models for every iteration seem too expensive.

[3]Typically, M_A^S is a binary classifier trained to minimize the loss function

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(M_A^S(\theta; x_i), y_i) := \mathcal{L}(X; Y, \theta). \tag{2}$$

where $\{x_i, y_i\}$, $i = 1, \ldots, n$ are entires of labeled dataset X, Y and $\hat{w}$ is a trainable parameter. For ML attacks, X denotes the feature embeddings of key-gates of a relocked and resynthesized design, and Y denotes their corresponding key-bit values. n denotes the total number of locations relocked in the design. The procedure for labeled dataset generation involves re-locking the locked netlist under attack and then re-synthesizing it using the defender's recipe S This approach is taken by the SOTA ML attacks (Sec. II); it is is based on the insight that doing so allows to accurately capture the structural transformations caused by S.

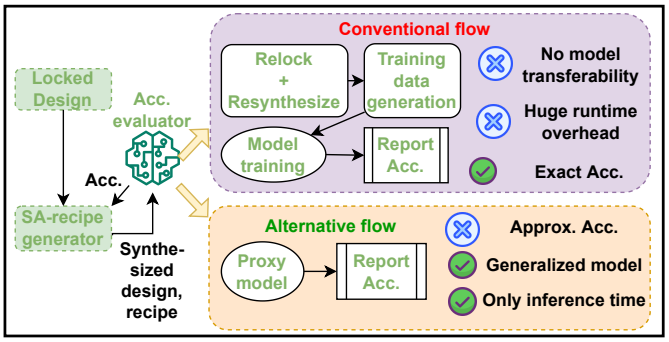

Fig. 2. Motivation for a proxy model.

ALGORITHM 1: Adversarial ML attack model training

Data: Re-synthesized locked netlist: $AIG_{initial}$, Epochs: N, Batchsize: B, Learning rate: γ, Periodicity: R
Result: Adversarial model $M_A^*(\cdot;\theta_{adv})$

Relock + Resynthesize $AIG_{initial}$ with random $L = 10$ length recipes.
Create $D_{training} = (X_{train}, Y_{train})$ using subgraph extraction from key-gates.
$\theta_0 \leftarrow$ He initialization, $t \leftarrow 0$
while $t < N$ **do**
  if $t\%R$ **then**
    $s^* \leftarrow$ SA (T_{init},$M_A^*(\cdot;\theta^t)$, N_{max},AIG$_{initial}$)
    Compute X^{s^*}, Y^{s^*} and augment $D_{training}$
  Compute $\Delta\theta^{(t)} = -\nabla\mathcal{L}(\theta^{(t)})$
  $\theta^{(t+1)} := \theta^{(t)} + \gamma\Delta\theta^{(t)}$
  return $M_A^*(\cdot;\theta_{adv})$

Proposed Solution. To enable a practical exploration of the search space, i.e., to tackle the stated RCs, we need an alternative in form of a proxy model M_A^* that yields good estimate of $M_A^S, \forall S \in [1, M^L]$. The challenge and motivation for such proxy model are also illustrated in Fig. 2.

For training such high-quality, transferable proxy model, the training data should contain a good number of structural transformations that are observed for a range of synthesis recipes. Next, we discuss how to train such proxy model.

B. Adversarially Trained Attacker's Model M_A^* (❶)

We adopt adversarial re-training [11], [12] to train a model that is exposed to a wide variation of localities, i.e., subgraph structures around key-gates. The motivation for adversarial retraining is as follows: we want to create adversarial subgraph embeddings where the attack model M_A^* mispredicts, whereupon we use these adversarial subgraphs to augment the training data and to subsequently learn a more robust model.

We create adversarial subgraph embeddings differently to conventional ℓ_2 or ℓ_{inf} perturbations in the image/vision domain [11]. Since logic synthesis offers no analytical closed-form function, we do gradient-free optimization. Similar to adding δ perturbations to original images, we seek for a synthesis recipe S_{adv} which transforms the re-locked netlist such that the subgraph embeddings are misclassified. We solve

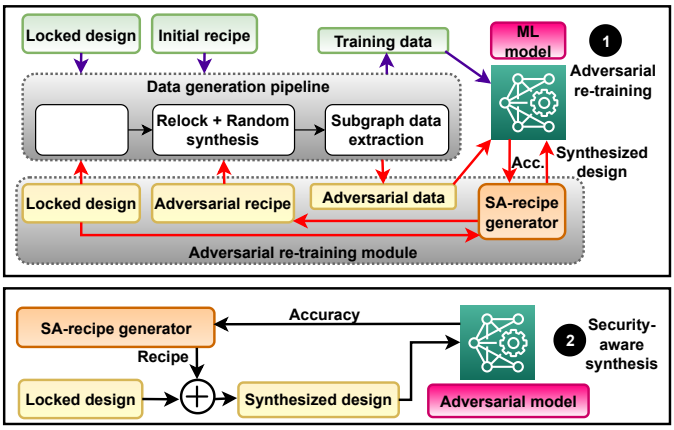

Fig. 3. ALMOST framework

an optimization problem for obtaining adversarial samples:

$$x^S = \arg\max_{\mathbf{S}} L(M_A^*(\theta;\hat{x}), y) \tag{3}$$

$$\text{where } \hat{x} = \textbf{READOUT}(h^k_{AIG^S}, k \in K), \tag{4}$$

$$AIG^S \leftarrow \mathcal{G}(AIG, S) \tag{5}$$

To solve this gradient-free optimization problem, we use simulated annealing (SA). Once the adversarial samples are generated, we augment the training data, rendering the model more robust and generalizable. Thus, we sample synthesis recipes that can provide diverse variants of the locked netlist under consideration. We solve the following min-max objective function for the adversarially trained model M_A^*.

$$\hat{\theta} = \min_{\theta} \max_{\mathbf{S}} \frac{1}{m} \sum_{i=1}^{m} L(M_A^*(\theta; x_i^S), y_i) \tag{6}$$

We outline more details in Algorithm 1.

C. Black-Box Optimization for Security-Aware Synthesis (❷)

While ALMOST may use a variety of black-box optimization approaches, like evolutionary algorithms, tree-search, etc., we use, wolog, a standard procedure for SA.

IV. Experimental Evaluation

Next we show how locked and ALMOST-synthesized designs are resilient against SOTA attacks, as the attack accuracy approaches random guessing.

A. Setup

Benchmarks. We evaluate ALMOST on the largest ISCAS85 combinational benchmarks. We initially lock them with RLL, considering key-sizes of 64 and 128. The resilience of ALMOST is demonstrated against OMLA, SCOPE, and the redundancy attacks (Sec. II).

Synthesis. As indicated, we use the synthesis suite *yosys* with *ABC* [10], for its flexibility of tuning synthesis recipes.

We select *resyn2* as baseline recipe, which is widely used for delay optimization. For fair comparison, we devise fixed-length synthesis recipe of $L = 10$. We employ seven synthesis transformations: `rewrite`, `re-substitute`, `refactor`, `rewrite -z`, `resub -z`, `refactor`

`-z`, and `balance`. For technology mapping, we use the NanGate 45 nm technology library.

Attack Model. As the OMLA framework is publicly available [1], we employ it for building the attacker's models. We set the network architecture, training configuration, and hyper-parameter settings as reported in [1]. For additional characterization of ALMOST, we apply recent oracle-less attacks, SCOPE [7] and Redundancy [8].

We study the effectiveness of adversarially trained M_A^* by comparing three variants:

- M_A^{resyn2} is the baseline model, where the attacker re-locks and re-synthesizes using the defender's baseline synthesis recipe, resyn2.
- M_A^{random} is trained on re-locked circuits that are re-synthesized using random recipes of length $L = 10$.
- M_A^* is trained using our adversarial data-augmentation-based re-training.

For **adversarial training**, we generate adversarial samples after every $R = 50$ epochs of training (Alg. 1). We start with 1,000 training data samples considering a 9:1 split for training and validation. We augment 200 adversarial samples at each SA iteration. We trained for 350 epochs in total.

B. Comparing Attack Models

Here, we first analyze the prediction accuracy of various attack models when attacking the original locked circuit synthesized using resyn2 (T_{resyn2}) and in attacking the locked circuit synthesized with 1000 random synthesis recipes (the "random set"). Table I shows the results, where the reported accuracy on the random set (*random* column) is the average achieved accuracy.

We observe a clear gap in accuracy when using M_A^{resyn2} to attack T_{resyn2} in comparison to its accuracy in attacking the random set. This shows that M_A^{resyn2} learns the structural changes resulting from the application of the defender's synthesis recipe. However, the model accuracy suffers severely in attacking the random set. We observe that the accuracy reduced drastically $\sim 1\% - 9\%$ with an average of 4.8%.

In contrast, the accuracy of M_A^{random} varies less in attacking T_{resyn2} compared to attacking the random set. Also, the accuracy when attacking the random set is better than the accuracy exhibited by the M_A^{resyn2} model.

The M_A^* model has more consistent accuracy when attacking T_{resyn2} and the random set ($0.18\% - 2.28\%$). M_A^* consistently achieves higher accuracy than the other models on the random set. This indicates good generalization of M_A^*, suggesting that it is a good fit as the accuracy evaluator in ALMOST's black-box optimization.

C. Generating ALMOST Synthesis Recipe

We generate S_{ALMOST} using the SA-based recipe generator such that the attack accuracy using M_A^{ALMOST} is $\sim$ 50%. We run SA for 100 iterations using an initial temperature of 120 and `acceptance=1.8`. Fig. 4 illustrates the SA-based recipe generation on ISCAS benchmarks. For showing the effectiveness of ALMOST using the M_A^* model, we compare our results to two other evaluators: M_A^{resyn2} and M_A^{random}. Blue line represents the attack accuracy estimated by M_A^* on the locked netlist synthesized with the recipe generated during simulated annealing. Orange and green represent the attack accuracy trend for M_A^{resyn2} and M_A^{random}, respectively.

There is a consistent trend in all plots: SA search using M_A^* requires more iterations to find a synthesis recipe where accuracy goes to $\sim 50\%$; with a clear pattern on benchmarks c2670, c3540, c5315, and c7552. This follows our intuition: M_A^* is trained with enough subgraph neighborhood diversity (netlist localities with key-gates 0 and 1). Hence, the SA recipe generator requires more time to find a synthesis recipe in the search space that is unfamiliar to M_A^*. In contrast, using M_A^{resyn2}, SA will quickly find a synthesis recipe where the accuracy falls to $\sim 50\%$. However, it is possible that there is a substantial accuracy gap between M_A^{resyn2} and M_A^S models for a particular recipe S generated by SA-based recipe generator which can give a false indication that the recipe will lead to general ML attack resilience. M_A^{random} performed similarly to M_A^*, however, there is a wide variation in accuracy obtained by M_A^{random} making it not a great choice for the defender's proxy model of an attacker. For c2670, c5315, and c7552 where accuracy could not reach $\sim 50\%$ within the budgeted iterations, we pick the synthesis recipe obtained at the end.

D. Comparing ALMOST Synthesis Recipe with Resyn2

After generating the recipes, we evaluate the efficacy of ALMOST synthesized circuits in thwarting the oracle-less machine learning attack (OMLA). Table II compares OMLA attacks on locked circuits synthesized using *resyn2* versus ALMOST generated synthesis recipe. On most benchmarks, our test accuracy is $\sim 50\%$. There is a $3\% - 12\%$ drop in accuracy which is substantial. The proposed approach, which uses a proxy model in the loop, found a synthesis recipe from the search space that can transform the netlist structure in a way that thwarts OMLA from obtaining high accuracy. We also report attack results from applying the redundancy attack [8] and SCOPE [7]. ALMOST synthesized circuits are more resilient than resyn2 synthesized ones.

E. What Happens if the Attacker Re-Synthesizes?

We assume the attacker can re-synthesize the ALMOST locked netlist with the aim to gain more information about how a design has been transformed. Hence, we run experiments to analyze whether an attacker can improve the accuracy of M_A^{ALMOST} by re-synthesizing a locked netlist again and training M_A^{ALMOST}. We re-synthesize the ALMOST locked circuit for area optimization and delay optimization, assuming that this is the "typical" goal for synthesis. As such, a defender wants to avoid any correlation between area and/or delay optimization and attack accuracy since this can be exploited by the attacker. We use the SA-based recipe generator with the ALMOST synthesized circuit as input and generate recipes targeting area/delay minimization. We take area and delay numbers of resyn2 as a baseline.

Fig. 5 shows the attack accuracy of M_A^* on re-synthesized circuits and corresponding area/delay. Blue denotes the attack accuracy of the ML model and orange denotes the normalized

TABLE I
PREDICTED ATTACK ACCURACY (%) FOR DIFFERENT ADVERSARIAL MODELS

Variant	Key-size	Benchmarks													
		c1355		c1908		c2670		c3540		c5315		c6288		c7552	
		resyn2	random	resyn2	random	resyn2	random	resyn2	random	resyn2	random	resyn2	random	resyn2	random
M_A^{resyn2}	64	57.52	54.21	59.01	50.57	58.00	51.17	59.63	52.26	62.62	57.46	52.21	53.32	66.33	58.46
	128	59.36	53.21	62.12	57.56	59.26	52.32	60.25	53.21	68.95	58.55	53.31	51.26	71.21	59.89
M_A^{random}	64	63.71	59.97	51.63	53.36	61.00	57.78	62.38	58.84	59.96	59.93	56.63	53.32	66.33	60.17
	128	54.58	55.67	57.85	58.55	62.21	57.85	54.20	55.87	65.41	61.25	52.26	51.88	65.25	61.56
M_A^*	64	61.94	61.05	53.27	55.55	63.00	62.50	62.38	61.37	61.61	61.35	53.09	54.14	63.36	63.18
	128	59.36	58.89	61.05	62.10	61.89	63.45	59.98	63.42	67.88	69.25	52.99	54.58	69.25	66.59

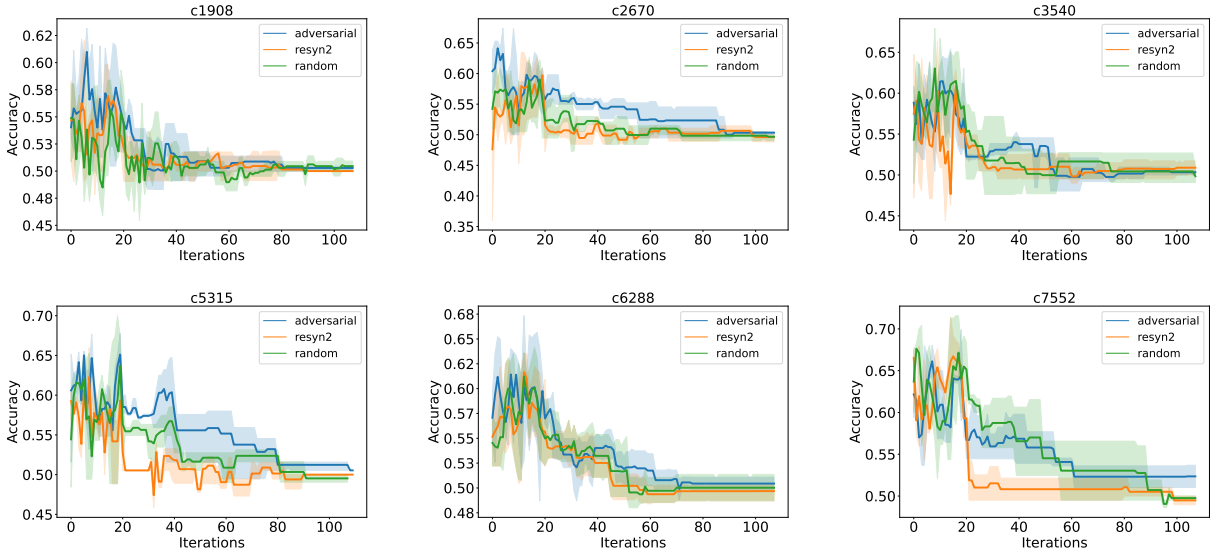

Fig. 4. Simulated annealing-based recipe search for minimizing attack accuracy to 50% or 0.5. M_A^* model consistently has better attack accuracy and thus SA takes more iterations to find a recipe resulting in accuracy $\sim 50\%$.

TABLE II
ATTACK ACCURACY (%) CONSIDERING SOTA ATTACKS

Attack	Keybits	Recipe	Benchmarks						
			c1355	c1908	c2670	c3540	c5315	c6288	c7552
OMLA	64	resyn2	57.52	59.01	58.01	59.63	62.62	52.51	66.33
		ALMOST	**54.18**	**47.80**	**49.78**	**46.57**	**49.78**	**49.88**	**55.55**
	128	resyn2	59.36	62.12	59.26	60.25	68.95	53.31	72.21
		ALMOST	**51.87**	**49.81**	**52.11**	**48.92**	**52.33**	**50.00**	**51.88**
SCOPE	64	resyn2	60.94	51.56	35.94	**34.38**	**45.31**	53.13	**40.63**
		ALMOST	**56.25**	**48.44**	**31.25**	37.50	57.81	**51.56**	43.75
	128	resyn2	51.56	46.09	**29.68**	36.71	**37.50**	59.37	46.09
		ALMOST	**50.78**	46.09	35.15	36.71	39.06	**53.91**	**45.31**
Redundancy	64	resyn2	**32.81**	37.50	**28.13**	50.00	50.00	34.38	35.94
		ALMOST	39.06	37.50	31.25	**45.31**	50.00	**31.25**	**32.81**
	128	resyn2	39.84	**35.93**	21.09	**41.40**	41.40	**31.25**	37.50
		ALMOST	**35.15**	42.96	**19.53**	44.53	**39.84**	34.38	**35.16**

area/delay (compared with resyn2). Recipe length for re-synthesis is $L = 10$ to match the length of resyn2. For delay minimization, SA generates recipes minimizing delay. However, there is no noticeable variation in the area except for c2670 and c7552. There is no clear correlation between area/delay minimization and improvement/decline in attack accuracy. An attacker who re-synthesizes the circuit will not know which synthesis recipe to use to improve attack accuracy.

F. Analyzing power-performance-area (PPA) metrics

To analyze the overhead ALMOST synthesized circuits bear for ML attack resilience, we present and analyze the PPA overhead of ALMOST synthesized designs by running Synopsys DC compiler in Table III. We consider two settings: (1) No optimization (-opt), and (2) Extreme optimization (+opt), where we enable ultra effort optimization along with the area recovery option. We use the PPA of the original locked netlist as a baseline. Area overhead varies in the range of $\sim \pm 3\%$. Similarly, power overhead also varies in the range of $\sim \pm 5\%$. For delay, circuits like c2670 have a relatively high overhead of around 18%. However, delay overhead is 15% lower for c7552. On average, ALMOST generates ML attack resilient circuits with low overhead.

V. CONCLUSION AND FUTURE WORK

ALMOST mitigates oracle-less ML-based attacks on logic locking. It uses synthesis tuning to make designs locked with a 100% vulnerable locking approach attack resilient using suitable synthesis recioes, with low impacts on PPA metrics. It applies to other locking techniques. Future research directions include investigating the impact of synthesis transformations

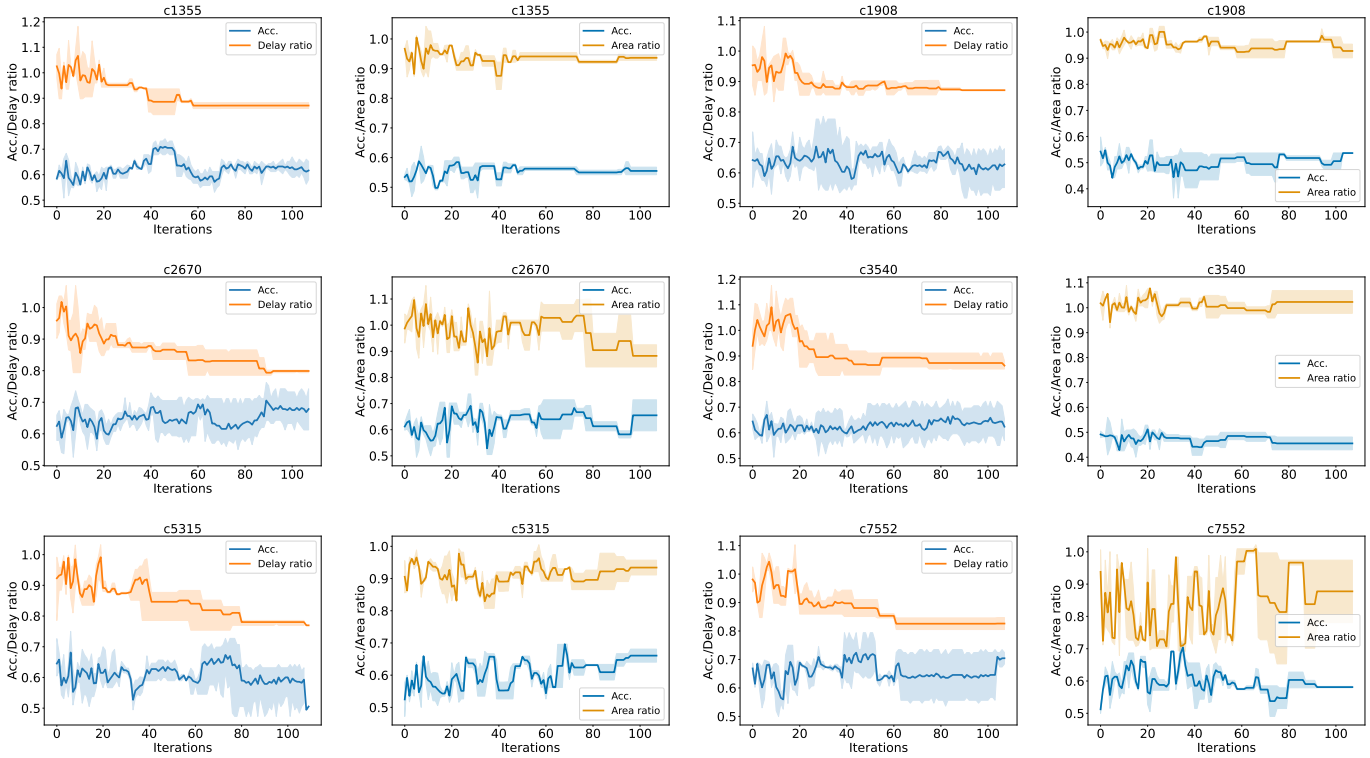

Fig. 5. SA-based recipe search minimizing delay, area after ALMOST-driven synthesis. Note: attack accuracy does not correlate with delay, area optimization.

TABLE III
POWER-PERFORMANCE-AREA (PPA) OVERHEAD (%) FOR ALMOST SYNTHESIZED CIRCUITS. -OPT: NO OPTIMIZATION, +OPT: EXTREME OPTIMIZATION

Variant	Key-size	Benchmarks													
		c1355		c1908		c2670		c3540		c5315		c6288		c7552	
		-opt	+opt	-opt	+opt	-opt	+opt	-opt	+opt	-opt	+opt	-opt	+opt	-opt	+opt
Area	64	+2.19	+0.89	-0.63	-0.95	-2.41	-2.89	+1.08	+0.73	+0.76	+0.53	+1.18	+0.98	+2.28	+2.19
	128	-0.05	-0.65	+2.32	+1.98	-0.38	-0.57	+0.94	+0.67	+0.04	-0.05	+3.08	+2.79	+0.84	+0.76
Delay	64	-3.45	-3.45	-4.95	-4.95	+18.31	+18.31	-0.46	-0.46	+5.00	+3.75	-0.93	-0.70	-15.24	-15.24
	128	+9.49	+4.47	-2.37	-1.42	+8.28	+8.28	+7.52	+7.52	-2.69	-2.69	-6.70	-6.49	-7.01	-7.01
Power	64	+3.36	+2.25	-0.28	-0.52	-3.64	-4.24	+3.49	+3.40	-0.12	-0.06	-0.36	-0.96	+1.17	+1.38
	128	-1.10	-1.37	+2.20	+2.05	-0.17	+0.12	-1.02	-1.16	+0.81	+0.63	+2.57	+2.28	+0.81	+0.48

in creating indistinguishable key-gate localities for ML attack resilient design, developing a generalized reinforcement learning-based synthesis engine to generate resilient designs, and jointly optimizing PPA and security metrics.

ACKNOWLEDGMENTS

This research was supported in part by NSF Award 1553419 and 2039607. The opinions, findings, and conclusions, or recommendations expressed are those of the author(s) and do not reflect the views of any sponsors.

REFERENCES

[1] L. Alrahis *et al.*, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE TCAS II*, vol. 69, no. 3, pp. 1602–1606, 2022.
[2] P. Chakraborty *et al.*, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *AsianHOST*, 2018, pp. 56–61.
[3] N. Limaye *et al.*, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE TCAD*, vol. 40, no. 9, pp. 1740–1753, 2020.
[4] L. Alrahis *et al.*, "Unsail: Thwarting oracle-less machine learning attacks on logic locking," *IEEE TIFS*, vol. 16, pp. 2508–2523, 2021.
[5] J. A. Roy *et al.*, "EPIC: Ending piracy of integrated circuits," in *DATE*, 2008, pp. 1069–1074.
[6] C. Umans, "Hardness of Approximating Σ_p^2 Minimization Problems," in *FOCS*, 1999, pp. 465–474.
[7] A. Alaql *et al.*, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE TVLSI*, vol. 29, no. 8, pp. 1529–1542, 2021.
[8] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *DATE*. IEEE, 2019, pp. 540–545.
[9] D. Sisejkovic *et al.*, "Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach," *ACM JETC*, vol. 17, no. 3, pp. 1–26, 2021.
[10] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *CAV*, 2010, pp. 24–40.
[11] A. Madry *et al.*, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.
[12] G. Jagatap *et al.*, "Adversarially robust learning via entropic regularization," *Frontiers in AI*, vol. 4, 2022.