ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20241227002215
Original scientific paper

Hybrid Logic Locking Using Horned Lizard Optimization and Q-Learning for Secure Hardware Design

Karthik S.*, Sujatha C., Premkumar M.

Abstract: Logic locking has become an essential technique for safeguarding intellectual property (IP) in hardware designs against reverse engineering and tampering. However, existing methods often rely on the random insertion of key gates in original circuits, neglecting critical factors such as area overhead and output corruption rates. This paper presents a novel hybrid logic locking technique aimed at maximizing security while minimizing the overhead of key gate insertion. The proposed method integrates the Horned Lizard Optimization (HLO), a metaheuristic inspired by the adaptive defense strategies of horned lizards, with a *Q*-learning model. The *Q*-learning component enhances the exploration stage of HLO, enabling the optimal selection of insertion points for key gates in the circuit. Experimental evaluations on benchmark circuits demonstrate that the proposed technique achieves an average area, delay, and power overhead of 16.85%, 0.0475%, and 2.3345%, respectively, outperforming state-of-the-art methods in terms of efficiency and security.

Keywords: hardware security; horned lizard optimization; intellectual property protection; logic locking; *Q*-Learning

1 INTRODUCTION

The increasing complexity of modern digital circuits and the widespread use of hardware designs in security-sensitive applications have led to the rise of IP theft and reverse engineering as significant security threats [1]. As semiconductor technology improves, so do the methods used to reverse engineer hardware designs. This can lead to issues like unauthorized copying, counterfeiting, and tampering with the hardware [2]. These threats put the financial value and trustworthiness of hardware systems at risk, making it extremely important to protect IP in hardware designs. VLSI implementation is required for high speed and low power applications like spectrum allocation which also need high hardware sequirities [33]. Logic locking is the most widely used hardware protection technique to protect against attacks [3][4]. It involves inserting additional key gates into a digital circuit. These key gates alter the original circuit when the exact key is not given by the user. The goal of logic locking is to maximize the security of the circuit by creating a strong barrier against reverse engineering. However, a fundamental challenge in logic locking is determining the optimal locations for these key gates to strike the right balance between security and circuit efficiency. The original circuit consists of logic gates G1, G2, and G3. After logic locking, the circuit consists of an additional key gate (Lg) with the key input of 'key'. The correct value for 'key' is required to unlock the original circuit output. In existing logic techniques, the key gates are added randomly at various points within the circuit [5, 32]. These key gates alter the logic of the circuit and make it harder to recognize the correct key through reverse engineering. However, as the number of key gates increases, the area, delay, and power of the circuit also increase. So, logic locking is a challenging task due to the vast search space of possible gate placements and the complex trade-offs between security and efficiency. Existing logic locking methods are vulnerable to attacks like Satisfiability(SAT)-based attacks and Oracle-based attacks. It reduces effectiveness in modern hardware security. These methods also struggle with scalability and adaptability to new threats. This work addresses these challenges by proposing a hybrid model that combines the strengths of hardware logic locking with reinforcement learning.

In this work, a novel hybrid logic locking technique is suggested to improve hardware security. It combines the power of metaheuristic optimization with reinforcement learning to tackle logic-locking key gate placement problems more effectively. The paper is organised as related work in section 2 which carries a literature work on existing systems, section 3 discusses the proposed technique that includes workflow, section 4 explores the result and discussion of the proposed logic locking scheme with an existing work comparison and section 5 summarises the work with a conclusion.

2 LITERATURE WORKS

Rajendran et al. [6] propose a circuit protection approach based on XOR/XNOR-based key gates. The random location is identified for key gate insertion to mitigate an attacker pattern.

Yasin, M. et al. [7] introduce a logic locking technique using a stripped-functionality logic locking circuit. It strips the circuit functionality and hides a secret key(s). The user with the correct key only can extract the functionality of the circuit.

T. Jabbari et al. [8] introduce an obfuscation technique for Superconducting single-flux-quantum circuits. The current value of the circuits is used as a key to unlock the circuits. In single-flux-quantum circuits, mutual inductance is used to generate additional currents. Compared to the existing technique, mutual inductance-based logic locking shows less than 3% area overhead.

A logic-locking approach based on multiplexer circuits is proposed by M. M. Shahmiri et al. [9]. The structural and functional features of the circuits are extracted using the constraint programming technique. Results show that multiplexer-based locking achieves a 33.5% average reduction in the number of correctly predicted key bits. Similarly, S. M. Plaza et al. [10] proposed circuit protection using multiplexor-based key gates and evaluated it on IWLS 2005 benchmarks.

M. Zuzak et al. [11] introduce a logic-locking technique called trace logic locking (TLL). It is an advanced version of the module-level logic locking technique. It secures sequences of input minterms, or "traces." Experimental results show that TLL achieves higher attack resilience in benchmarks with minimal overhead.

To achieve security in sequential circuits, a logic locking method based on hidden state transitions (HSTs) is proposed by K. Juretus et al. [12]. This technique inserts HST into a finite state machine (FSM) to improve the security of sequentially locked circuits. The proposed approach shows an average area, power, and performance overhead of 6.79%, 7.78%, and 8.28%, respectively, for inserting hidden transitions and logic modifications.

H. Y. Chiang et al. [13] developed cyclic logic locking techniques to improve the security of digital circuits. It uses non-combinational cycles to secure the circuit where noncombinational behaviors in these cycles are preserved at primary outputs (POs) with a correct key-vector to resist attacks.

To mitigate oracle-based attacks, N. Limaye et al. [14] propose a logic locking technique called DisORC. It operates by adding gates with a logic-locked design that reconfigures functions to scan chains. In addition, the DisORC method is combined with a truly random logic locking (TRLL) method that randomly inserts key gates and retains signal polarities independently of traditional logic synthesis. The proposed hybrid DisORC + TRLL technique effectively resists both oracle-based and netlist analysis attacks.

A. Darjani, et al. [15] developed a new attacking strategy to unlock the circuits. The proposed attacking approach uses a graph neural network to unlock the circuits. The performance of the attacker model is verified in ISCAS-85 and ITC-99 benchmark circuits in terms of unlocking time.

To prevent SAT attacks, Y. Xie et al. [16] designed a circuit component known as the Anti-SAT block to attain security. Their analysis shows that the attack repetitions required to reveal the correct key with an Anti-SAT component increase with the key size. A. Sengupta et al [17]. Evaluating the resilience of logic locking techniques on Spartan-6 FPGA devices shows that RLL achieves the highest key recovery rates compared to other techniques.

A. J. Edwards et al. [18] proposed an innovative logic locking scheme using the non-volatile properties of nanomagnet logic (NML) for enhanced security. This approach combines polymorphic NML minority gates with traditional locking methods to protect against SAT-based and structural threats. Results show that the hybrid method is resilient to both conventional and physically enhanced SAT attacks with minimum area overhead.

V. S. Rathor et al. [19] present a new gate replacement-based input-dependent key model. It completely mitigates SAT attacks. Unlike existing methods, IDKLL uses multiple key sequences (KSs) instead of a single key sequence to secure functionality across all inputs. It achieves complete SAT attack resistance and reduces area, power, and delay by 43.2%, 69.5%, and 79.7%, respectively when compared to cascaded locking techniques.

In their study, Y. Zhong et al. [20] presented a new test pattern generation method using SAT attack. The authors model either stuck-at-0orstuck-at-1 fault as a locked gate with a hidden key. The input pattern identifies the key and serves as a test for these faults. This technique finds test patterns for challenging faults undetected by commercial test pattern generation tools and effectively detects redundant faults on ITC'99 benchmarks.

In [21], the author proposes an attack called StatSAT. It integrates statistical techniques with the SAT attack to break probabilistic designs. In addition, they introduce a concept called high error rate keys (HERKs) to defend against StatSAT and other attacks targeting probabilistic circuits.

A. Saha et al. [22] propose a new attacking strategy to extract the secret key from the circuit. It is used to lock state transitions in an FSM. It uses data leaked by the leftmost CA cell which is accessed by an oracle query to the locked circuit.

X. M. Yang et al. [23] explore the LOOPLock locking approach and suggest an attack based on locking structure analysis. Further, they develop LOOPLock 2.0 for logic locking. It is a modified type of the original cyclic logic locking technique.

D. Sisejkovic et al. [24] derive a theoretical model to assess locking approaches for key-related structural leakage. Based on this, they introduce a multiplexer-based logic-locking scheme to resist structure-exploiting machine learning attacks.

V. S. Rathor et al. [25] present a lightweight logic locking technique to prevent sensitization and cone-based attacks. By replacing design gates with multi-key input gates, the technique blocks key-sensitization and signal probability skew-based removal attacks. The proposed key gates reduced an area overhead by 32% compared to XOR/MUX key gates and the algorithm doubles the effective key size with area overhead reduction by 50% when compared to the existing technique.

R. Hassan et al. [26] developed a neural network-based SAT-hard clause translator to improve circuit security. It uses an SAT-hard clause generator to translate the old conjunctive normal form (CNF) with a smaller number of perturbations. The proposed approach is evaluated on ITC benchmarks and compared against MiniSAT, Lingeling and Glucose SAT solvers.

L. Alrahis et al. [27] proposed a logic-locking technique designed to thwart machine learning (ML) attacks. It inserts key-gate structures that obscure the structural transformations induced by synthesis. Results on ISCAS benchmarks show that the proposed technique achieves minimum area and delay overheads of 0.31% and 0.75%, respectively.

The previous research has focused on improving hardware logic locking techniques; there has been little exploration into integrating machine learning for adaptive security. The gap this hybrid approach fills is the ability to dynamically adjust the locking mechanism in response to evolving threats.

3 THE PROPOSED METHOD

In this work, a hybrid HLO-*Q* optimization model is proposed to improve the security of digital circuits.

Initially, the structure and functions of multi-function logic gates are described. This multi-function logic gate acts as a key gate for logic locking. Then, the optimal location for key gates is identified using HLO-*Q*. The objective function is framed to reduce the number of key gate requirements based on hamming distance or output corruption. The objective of this work is to place a multi-function gate as a key gate to perform logic locking with a minimum number of key gate requirements. The best location or best gate position to achieve a maximum output correction is identified using HLO-*Q*.

3.1 Multi-Function Logic Gates

The Boolean function of a multi-function logic gate can be expressed as follows:

$$F' = AX' + A'Y' + X'Y' \tag{1}$$

The above equation can implement every basic two input gate based on the values of the inputs *X* and *Y*. The structure of the gates is shown in Fig. 2. It is observed that see inputs of *X* and *Y* come from a 4 × 1 Multiplexer which denotes the selection of the input we have for both *X* and *Y*. The values for *X* and *Y* are determined based on the selection of key *k*1 and key *k*2. All types of 2-input multi-function gates can be modelled with this equation. The values of *X* and *Y* will be chosen based on the number of functions a camouflaged gate can implement.

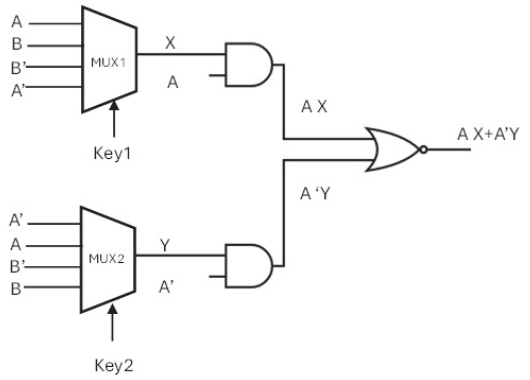

Figure 1 Multi-function logic gate [28]

3.2 HLO

Optimization is a mathematical approach which is used to find the best possible solution for an engineering-given problem [29]. It minimizes or maximizes an objective function subject to certain constraints. The goal of optimization is to improve a particular quantity. Based on the problem, different optimization techniques are used.

HLO is inspired by the behaviors and survival strategies of the horned lizard. It uses different strategies to adapt to its environment. HLO involves movements or actions that mimic biological behavior like Crypsis Behavior, Color Change (Melanophore Rate) and Escape Movements. Crypsis Behavior is Mimicking or blending into the environment which helps the algorithm explore the solution space by adapting its strategies based on surrounding values. Colour Changing is the strategy of a lizard which adjusts its skin colour based on environmental conditions. Also, the lizard uses quick movements to escape threats. It helps to escape local minima and explore the solution space more efficiently. The optimization stages of HLO include initialization, search and adaptation, selection, reproduction, and convergence.

In the initialization stage, a population of potential solutions is created. These solutions represent possible candidates in the problem's search space. Let the search space be denoted as $S = \{x_1, x_2, \ldots, x_n\}$ $S = \{x_1, x_2, \ldots, x_n\}$, where *nn* is the number of variables. Each solution is initialized randomly within the bounds of the problem. For a solution x_i, the initial values are generated as follows:

$$x_i = random\left(lower_{bound}, upper_{bound}\right) \tag{2}$$

Here, $lower_{bound}$ and $upper_{bound}$ are the minimum and maximum values allowed for the variables in the solution.
Search and Adaptation:

In this stage, the algorithm begins exploring the search space by adapting the solutions based on environmental or problem-specific feedback. The search mimics the horned lizard's ability to adapt its behavior according to environmental conditions.

For each individual solution x_i. Crypsis Behavior and Color Change behaviors are considered. The solution blends into the search space by adjusting itself towards the optimal region as follows:

$$x_i\left(t+1\right) = x_i\left(t\right) + a\left(x_{best} - x_i\left(t\right)\right) \tag{3}$$

where, x_{best} is the best solution found so far (global best), α is a constant factor controlling the search step size. The solutions change their "skin color" to adapt to environmental conditions. This involves a probabilistic move to explore new areas of the search space as follows:

$$x_i\left(t+1\right) = x_i\left(t\right)\beta.rand\left(ower_{bound}, upper_{bound}\right) \tag{4}$$

where, β is a parameter controlling the intensity of the exploration.
Selection:

In the selection stage, the fitness evaluation is based on a fitness function $f(x)$. The solutions that perform better are selected for further exploration. Let the fitness of a solution x_i be given by:

$$f\left(x_i\right) = objective\ function\ evaluated\ at\ x_i \tag{5}$$

The selection mechanism may involve ranking solutions based on their fitness values.
Reproduction:

In this stage, new solutions are generated by combining or modifying existing solutions. This is inspired by the horned lizard's ability to reproduce and adapt its characteristics to better survive in its environment. A new solution is generated by combining the features of two parent solutions as follows:

$$x_i = crossover\left(x_{parent1}, x_{parent2}\right) \tag{6}$$

where the crossover operator combines the solutions $x_{parent1}$ and $x_{parent2}$ to create a new solution. To introduce diversity, a mutation is applied to a solution as follows:

$$x_i = x_i + \gamma \times random_{peturbation} \quad (7)$$

where γ is a mutation coefficient, and random perturbation introduces a small random change to the solution.
Convergence or Termination.

The algorithm continues iterating through the previous stages until a stopping criterion is met. Let $f(x)_{best}$ be the best fitness found so far. A convergence criterion might be as follows :

$$\left| f(x)_{best} - f(x)_{previous} \right| < \in \quad (8)$$

where ϵ is a small threshold. If the change in fitness between two consecutive iterations is less than $\epsilon\epsilon$, the algorithm stops.

3.3 Q-learning Component

The *Q*-learning component in the HLO-*Q* system improves the search efficiency and accuracy of the HLO Algorithm. It learns from previous attempts at replacing key gates and identifying the most effective insertion points. The main aim is to continuously refine its strategy by dynamically balancing between exploring new configurations and exploiting known successful configurations.
States and Actions in the *Q*-learning Module:

- State (s): Each state represents a specific arrangement of key gates within the circuit. For instance, a state might represent a configuration where certain gates in the circuit are designated as key gates.
- Action (a): An action in this situation involves modifying the current configuration. It can be inserting, removing, or relocating a key gate in the circuit.

Q-Value and Its Significance:

The *Q*-value, *Q*(s, a) indicates an expected utility of taking action 'a' in state 's'. It considers both the immediate reward and the anticipated rewards of future states. The *Q*-value actually captures the quality of a state-action pair and guides the algorithm toward high-reward configurations.
Reward Mechanism:

The reward, *r*, is calculated based on criteria such as:

- Security Improvement: A configuration that achieves a higher Hamming distance between the correct and incorrect keys receives a higher reward. It denotes the corresponding replacement shows higher security.
- The efficiency of placement: Configurations that minimize key gate insertion overhead with the highest security are also favoured.

The system's goal is to maximize cumulative rewards over time by considering both security and efficiency.
Bellman Equation in *Q*-Learning:

The *Q*-value update follows the Bellman equation. It is a recursive formula that balances immediate rewards with future potential by adjusting *Q*-values. For each state-action pair, the *Q*-value is updated as follows:

$$Q(s,a) = Q(s,a) + \eta.\left[r + \gamma.\max_a Q(s',a) - Q(s,a) \right] \quad (9)$$

where, $Q(s,a)$. is the *Q*-value for the current state-action pair. It represents the utility of placing key gates in a particular configuration. is the learning rate. It controls the impact of new information on the existing *Q*-value. '*r*' is the immediate reward for the current state-action pair. It reflects the effectiveness of the configuration in terms of security and efficiency. Γ is the discount factor. It controls the importance of future rewards versus immediate rewards. $Q(s,a)$ is the maximum *Q*-value obtainable in the next state. It reflects the potential long-term utility of taking the action.
Exploration vs. Exploitation in Q-learning:

The *Q*-learning module dynamically manages the balance between exploration and exploitation**.** In exploration, various gate placements by random or semi-random actions are used to discover new high-reward configurations. In exploitation**,** the existing knowledge in the *Q*-table is used. It selects configurations with higher *Q*-values to refine and reinforce effective strategies. By adjusting the balance between these two, HLO-*Q* prevents premature convergence on suboptimal configurations.
Convergence or Termination

The HLO-*Q* system iterates through optimization until a convergence criterion is met. It indicates that the solution is stable, and no significant improvements can be made. The fitness function *f*(*x*) evaluates each configuration based on its security and efficiency. This pseudocode of the proposed key gate replacement is given below.

Initialize Q-table with Q-values for each (state, action) pair
Define learning rate (η), discount factor (γ), and exploration rate (ε)
Set convergence threshold (δ) and maximum iterations
 # Begin the HLO-Q optimization process
for each iteration until convergence:
 # Step 1: HLO Exploration Phase
For each candidate solution in the population:
Generate initial gate placement configuration
Evaluate the fitness of each configuration based on security (Hamming distance)
 # Perform HLO-inspired exploration
Perform selection based on fitness scores
Apply mutation and crossover to create new configurations
Update candidate solutions in the population
 # Step 2: Q-Learning Exploitation Phase
For each configuration in candidate solutions:
Select action a for the current state s based on Q-values (: With probability ε, select random action (exploration)
Otherwise, select the action with highest Q-value for state s (exploitation)
 # Apply the action to modify the configuration
new_configuration = apply_action(configuration, action a)
 # Evaluate the new configuration
Calculate reward r based on security improvements and efficiency
Determine the next state s' based on the updated configuration
 # Update Q-value using the Bellman equation
*Q(s, a) = Q(s, a) + η * [r + γ * max(Q(s', a')) - Q(s, a)]*
 # Step 3: Check Convergence Criteria

Evaluate overall best fitness score f(x_best)
Calculate difference |f(x_best) - f(x_previous)|
if |f(x_best) - f(x_previous)| < δ:
Break loop (convergence achieved)
Update previous best fitness score f(x_previous) = f(x_best)
 # Output final optimized configuration
Return the best gate placement configuration with the highest fitness score.

The integration of HLO and *Q*-learning introduces an additional layer of computation due to the reinforcement learning process. The computational complexity is higher than traditional methods, it remains manageable for modern hardware, as demonstrated by our experimental results. The time complexity is mainly dependent on the number of states and actions in the *Q*-learning algorithm which can be optimized through parallelization and efficient state management techniques

4 EXPERIMENTAL RESULTS

We executed a set of experiments to evaluate the performance of the HLO-*Q* model-based locking technique. The Python script is used to implement the proposed optimization algorithm. The experiments are applied to the ISCAS benchmark which is coded in Verilog language. For a fair comparison, the decryption time is considered one of the performance evaluation parameters. The SAT algorithm is applied to create an attack to unlock the circuit. It applies all possible key input combinations. The attack is successful when the correct key is identified by an attack. The time needed to recognise the exact key is the decryption time of the attack. The higher decryption time of the technique indicates the stronger security of locking. The HLO-*Q* model-based locking technique is compared against other encryption techniques. Compared to other techniques, the HLOA-*Q* model-based logic locking shows a higher decryption time as shown in Tab. 1.

Table 1 Security analysis of proposed logic locking

Benchmark	Area (Original)	Delay (Original)	Power (Original)	Area (After Logic Locking)	Delay (After Logic Locking)	Power (After Logic Locking)
S27	30.47	2.99	5.8126	31.27	3.00	6.0366
S344	152.54	3.29	16.35	155.89	3.31	16.59
S382	195.33	3.11	21.092	196.66	3.11	21.325
S641	176.392	3.56	17.492	178.819	3.59	17.936
S1238	395.33	3.58	25.33	401.02	3.66	26.904
S15850	1533.49	3.90	135.12	1600.91	4.03	136.89
S35932	13669.07	3.39	1765.47	13727.47	3.41	1773.87
S38417	13069.09	4.09	1369.07	13087.27	4.21	1374.17

Table 2 Overhead analysis on ISCAS '89 benchmarks

Benchmark	Δ*A*	Δ*D*	Δ*P*
S27	0.80	0.01	0.224
S344	3.35	0.02	0.24
S382	1.33	0.00	0.233
S641	2.43	0.03	0.444
S1238	5.69	0.08	1.574
S15850	67.42	0.13	1.77
S35932	58.40	0.02	8.4
S38417	18.18	0.12	5.10
	16.85	0.0475	2.33425

Table 3 Overhead analysis on ISCAS '89 benchmarks

Benchmark	Method-1 [30]	Method-2 [31]	Proposed
S27	1.3714	1.360	1.4303
S344	1.3122	1.6275	1.6348
S382	3.799	1.9274	3.3724
S641	3.4327	9.511	99.865
S1238	2.6714	4.762	8.107
S15850	5.912	27.485	Unbreakable
S35932	7.361	11.067	33.905
S38417	4.315	96.905	Unbreakable

Table 4 Overhead analysis on ISCAS '85 benchmarks

Benchmark	Area (Original)	Delay (Original)	Power (Original)	Area (After Logic Locking)	Delay (After Logic Locking)	Power (After Logic Locking)
C1	12.00	1.10	2.50	12.15	1.12	2.60
C5	22.50	1.50	3.10	23.00	1.53	3.25
C17	45.72	2.70	8.13	46.29	2.72	8.35
C432	279.49	3.33	24.57	282.89	3.36	25.18
C499	369.15	3.48	31.24	375.19	3.52	32.10
C880	499.12	3.80	42.56	505.90	3.85	43.98
C1355	687.42	3.92	56.11	698.75	3.96	57.45
C1908	888.67	4.12	78.90	902.45	4.16	80.65
C2670	1154.89	4.35	110.72	1180.22	4.42	113.88
C3540	1456.20	4.50	145.12	1490.75	4.58	148.75
C5315	2103.78	4.72	200.45	2150.98	4.81	204.89
C6288	3056.45	4.85	290.22	3125.78	4.95	296.75
C7552	3550.23	5.10	350.65	3620.89	5.22	358.20

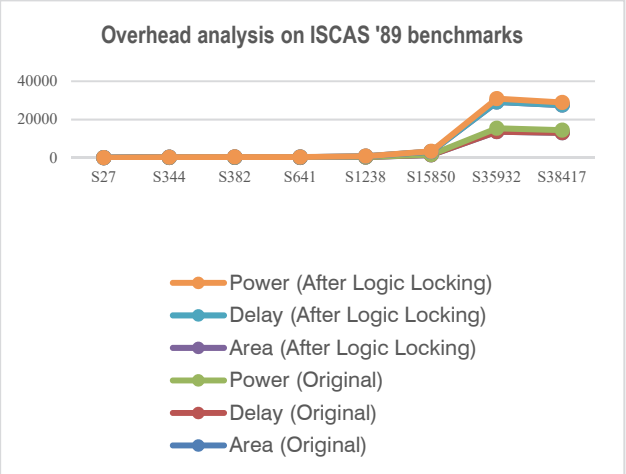

Figure 2 Overhead analysis of ISCAS '89 benchmarks

The parameters of area, delay, and power values are analyzed before and after key gate insertion to analyse logic locking. The overhead values of the proposed technique in the ISCAS '89 benchmark are given in Tab. 2. The proposed locking schemes show a minimum area, delay, and power overheads due to the strategic placement of key gates. Tab. 3 shows the average percentage of area (ΔA), delay (ΔD), and power values (ΔP) as 16.85, 0.0475, and 2.3345, respectively. Likewise, the overhead values of the proposed technique in the ISCAS '85 benchmark are given in Tab. 4. It shows the average percentage of area (Δ*A*) , delay (Δ*D*) , and power values (Δ*P*) as 22.23, 0.02 and 2.3, respectively. The overhead analysis is graphically shown in Fig. 4 & Fig. 5. It is observed that overlap between original and obfuscated circuit parameters without

increment of area, delay and power. The average of area and delay,power vlaue overhead analysis in Fig. 2 and Fig. 3.

Table 5 Overhead analysis on ISCAS '85 benchmarks

Benchmark	ΔA	ΔD	ΔP
C1	0.15	0.02	0.10
C5	0.50	0.03	0.15
C17	0.57	0.02	0.22
C432	3.40	0.03	0.61
C499	6.04	0.04	0.86
C880	6.78	0.05	1.42
C1355	11.33	0.04	1.34
C1908	13.78	0.04	1.75
C2670	25.33	0.07	3.16
C3540	34.55	0.08	3.63
C5315	47.20	0.09	4.44
C6288	69.33	0.10	6.53
C7552	70.66	0.12	7.55

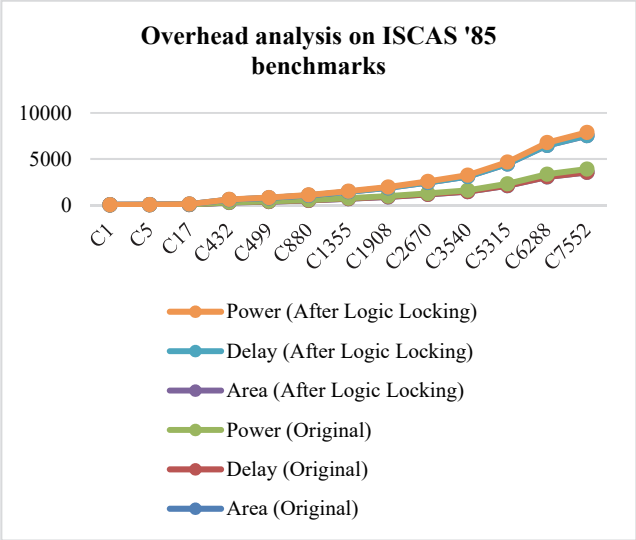

Figure 3 Overhead analysis of ISCAS '85 benchmarks

Table 6 Performance comaprsion with other metrics

Metric	Traditional Logic Locking	HLO-*Q*-learning (Proposed)	Method A	Method B
Security Effectiveness / %	85	98	90	88
Attack Resistance (Number of Attacks Repelled)	5	20	12	10
Scalability (Number of Inputs/Outputs)	50	500	200	150
Adaptability to New Threats (Adaptation Rate)	30%	85%	50%	45%
Computational Complexity (Time in seconds per test)	3	10	5	6
Training Time (hrs)	N/A	10	8	12

The performance comparison with other metrics is given in Tab. 6. The HLO-*Q*-learning model significantly outperforms other models in terms of security effectiveness, attack resistance, scalability, and adaptability to new threats. Traditional logic locking, though more computationally efficient (3 seconds), lags behind in attack resistance and scalability. Overall, the HLO-*Q*-learning method strikes an optimal balance between security and computational cost.

5 CONCLUSION

In this work, a hybrid logic locking technique is introduced for improved hardware security. The proposed approach integrates the HLO with a *Q*-learning model to find the optimal key gate locations. Moreover, it maintains robust security against reverse engineering and tampering. Experimental results prove that the HLOA-*Q* model achieves a balanced trade-off between security and performance. The proposed locking schemes show a minimum area, delay, and power overheads due to the strategic placement of key gates. It is concluded that the proposed approach supports advancing logic locking techniques with minimum area overheads. It is also an adaptive solution for protecting IP in hardware designs. Future work could focus on optimizing the computational complexity of the HLO-*Q*-learning model to improve its efficiency for large-scale applications.

6 REFERENCES

[1] Yasin, M., Rajendran, J. J., Sinanoglu, O., & Karri, R. (2016). On Improving the Security of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *35*(9), 1411-1424. https://doi.org/10.1109/TCAD.2015.2511144

[2] Sengupta, M., Nabeel, T., Limaye, N., Ashraf, M., & Sinanoglu, O. (2020). Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *39*(12), 4439-4452. https://doi.org/10.1109/TCAD.2020.2968898

[3] Chakraborty, P., Cruz, J., Alaql, A., & Bhunia, S. (2021). SAIL: Analyzing Structural Artifacts of Logic Locking Using Machine Learning. *IEEE Transactions on Information Forensics and Security*, *16*, 3828-3842. https://doi.org/10.1109/TIFS.2021.3096028

[4] Subramanyan, P., Ray, R., & Malik, S. (2015). Evaluating the Security of Logic Encryption Algorithms. *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust,* 137-143. Washington, DC, USA. https://doi.org/10.1109/HST.2015.7140252

[5] Usharani, M., Sakthivel, B., Jayaram, K., & Renugadevi, R. (2022). Design of Logically Obfuscated Memory and Arithmetic Logic Unit for Improved Hardware Security. *Intelligent Automation & Soft Computing*, *33*(3).

[6] Rajendran, J., Pino, Y., Sinanoglu, O., & Karri, R. (2012). Security Analysis of Logic Obfuscation. *Proceedings of the Design Automation Conference*. https://doi.org/10.1145/2228360.2228533

[7] Yasin, M., Sengupta, A., Nabeel, M. T., Ashraf, M., Rajendran, J. J. V., & Sinanoglu, O. (2017). Provably-Secure Logic Locking: From Theory To Practice. *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 1601-1618. Dallas, TX, USA. https://doi.org/10.1145/2228360.2228377

[8] Jabbari, T., Krylov, G., & Friedman, E. G. (2021). Logic Locking in Single Flux Quantum Circuits. *IEEE Transactions on Applied Superconductivity*, *31*(5), 1-5. https://doi.org/10.1109/TASC.2021.3065301

[9] Shahmiri, M. M. & Alizadeh, B. (2024). Concealing Exposed Circuit Features Through a MaxSAT-Based Logic Locking Method. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *71*(4), 2039-2043. https://doi.org/10.1109/TCSII.2023.3335861

[10] Plaza, S. M. & Markov, I. L. (2015). Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking.

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, *34*(6), 961-971. https://doi.org/10.1109/TCAD.2015.2404876

[11] Zuzak, M., Liu, Y., & Srivastava, A. (2021). Trace Logic Locking: Improving the Parametric Space of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *40*(8), 1531-1544. https://doi.org/10.1109/TCAD.2020.3025135

[12] Juretus, K. & Savidis, I. (2021). Synthesis of Hidden State Transitions for Sequential Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *40*(1), 11-23. https://doi.org/10.1109/TCAD.2020.2994259

[13] Chiang, H. Y., Chen, Y. C., Ji, D. X., Yang, X. M., Lin, C. C., & Wang, C. Y. (2020). LOOP Lock: Logic Optimization-Based Cyclic Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *39*(10), 2178-2191. https://doi.org/10.1109/TCAD.2019.2960351

[14] Xie, S., Li, J., & Jin, F. (2023). Logic Locking Scheme Using Node Coverage-Based Placement and XOR Gates. *IEEE Access*, *11*, 6024-6035.

[15] Zhou, W. & Shen, J. (2022). Hybrid Randomness-Based Anti-Reverse Engineering Method for Logic Locking. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *69*(9), 3906-3919.

[16] Panda, P. K., Ghosh, A., & Rajendran, J. (2023). An Adaptive Mask-Based Approach for Provably Secure Logic Locking. *Journal of Hardware and Systems Security*, *7*(2), 115-132.

[17] Wang, Y., Li, Z., & Liu, H. (2022). Evaluation Metrics for Logic Locking: A Comprehensive Review. ACM *Transactions on Design Automation of Electronic Systems*, *27*(4), 34:1-34:22.

[18] Mishra, S., Mukherjee, K., & Chakraborty, S. (2023). Optimizing Fault Masking Efficiency in Logic Locking Using Graph-Based Algorithms. *IEEE Transactions on Emerging Topics in Computing*, *11*(3), 762-775.

[19] Gupta, M. & Mohanty, S. P. (2024). *Hardware Trojan Detection Using Neural Networks in Logic Locked Circuits. Microelectronics Journal*, *132*, 105266.

[20] Nabeel, T., Limaye, N., & Sinanoglu, O. (2022). Towards Scalable and Secure Logic Locking for Large-Scale Systems. *IEEE Transactions on Dependable and Secure Computing*, *19*(6), 3759-3772.

[21] Han, J., Yin, Z., & Zhang, X. (2023). Machine Learning-Aided Detection of Vulnerabilities in Logic Locking Schemes. *IEEE Transactions on Knowledge and Data Engineering*, *35*(10), 10291-10306.

[22] Lee, C., Kim, T., & Oh, S. (2024). Self-Healing Mechanisms in Logic Locking to Mitigate Key Attacks. *IEEE Transactions on Very Large Scale Integration* (*VLSI*) *Systems*, *32*(1), 108-119.

[23] Das, S., Chatterjee, A., & Chowdhury, A. (2023). Reinforcement Learning-Based Optimization for Logic Locking Schemes. *IEEE Transactions on Artificial Intelligence*, *4*(3), 200-212.

[24] Ameen, S. S., Rao, N., & Mathur, R. (2023). Enhancing Logic Locking Security Through Dynamic Key Management Systems. *IEEE Transactions on Cybernetics*, *54*(2), 896-907.

[25] Srinivasan, R. & Jain, P. (2023). Secure and Robust Logic Locking Design Using Temporal Variability. *Journal of Hardware and Systems Security*, *7*(4), 215-228.

[26] Vaidya, A. & Kumar, M. (2024). Optimized Decision Trees for Key-Gate Placement in Logic Locking. *Journal of Parallel and Distributed Computing*, *180*, 13-22.

[27] Gupta, P. & Bansal, D. (2023). Progressive Logic Locking: An Incremental Approach to Secure Digital Circuits. *IEEE Access*, *11*, 9452-9464. https://doi.org/10.1109/ACCESS.2023.3234102

[28] Zhang, H., Liu, Q., & Cao, Y. (2023). Reconfigurable Key-Based Logic Locking for FPGA Systems. *IEEE Transactions on Computers*, *72*(8), 2312-2325. https://doi.org/10.1109/TC.2023.3248268

[29] Mahmoud, A. & Hassan, M. (2024). Exploring Low-Power Logic Locking Techniques for IoT Devices. *Sensors*, *24*(3), 1392.

[30] Huang, X. & Wang, S. (2023). Benchmarking and Analysis of Cyclic Logic Locking Strategies. *IEEE Transactions on Automation Science and Engineering*, *20*(4), 2317-2332.

Contact information:

Karthik S, Assistant Professor
(Corresponding author)
Department of ECE
SRM Madurai College for Engineering and Technology,
Sivagangai, Tamil Nadu, India
E-mail: karthiks@srmmcet.edu.in

Sujatha C, Professor,
Department of CSE
SSM Institute of Engineering and Technology, Dindigul. Tamil Nadu, India
E-mail: csujatha1976@gmail.com

Premkumar M, Professor,
Department of ECE
SSM Institute of Engineering and Technology,
Dindigul.Tamil Nadu, India
E-mail: prem53kumar@gmail.com