WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Input/output Locking Security System with Encryption/Decryption Module using Deep Neural Network Model

Dinesh Kumar Cholkar[1], Jitendra Ahir[2], Laxmi Singh[3], Sanjeev Kumar Gupta[4]

[1]PhD. Scholar, Department of Electronics & Communication Engineering, Rabindranath Tagore University (RNTU), Bhopal
[2,3,4] Professor, Department of Electronics & Communication Engineering, Rabindranath Tagore University (RNTU) , Bhopal

ABSTRACT

Technology has increased dramatically in the effectiveness of the attacks. There are different measures, at the logic lock (LL) level and at the chip level etc. The emergence of satisfaction (SAT)-based functional queries, and 3-D at the board level at the board level forced attacks at the board level. The Encryption/Decryption Module can improve overall security of the strategy. To determine the accurate key for a particular blocked circuit attacker will try. If the target is to attack attempts to output a circuit that calculates function. One way the two goals relate to is that a critical recovery of a particular blocking circuit means recovery of the circuit. However, the opposite may not be true. The researchers propose locking mechanisms in response to attacks on previous mechanisms, with another investigator having immediate impact. This input design applies to opened ICs and buttons that are avoided from off base yield. The calculation proceeds to discover exceptional input designs and kills untrue keys until the fitting key is found. The vital values for these entryways are "0" or 1. An inverter can be included to the same flag line as turning the key esteem. The thought of this approach is to anticipate the other party from speculating key values based on the entryway sort, as they don't know in case the over is portion of the first circuit or on the off chance that it has been included within the handle of rationale locking.

Index Term: Security, RE, Delay, overheads, Logic locking, PCB,

I Introduction

One way to make integrated circuits (ICs) more secure is through a method called hardware obfuscation. This method changes how a circuit is built or described to make it harder for someone trying to break into it. This process can be done in different stages of the design, like the register-transfer level (RTL), gate level, and layout level. At the RTL level, a design is usually made using a programming language called HDL, such as Verilog or Verilog.This language helps create a high-level version of the circuit. Hackers might want to figure out the original RTL code to copy the design. The RTL code can be turned into a secret form using a formula, and then changed back using a special key. Without the key, an attacker can't get the original design. Also, the RTL code can be shown as a graph with a state machine. By adding more states, it's harder to understand how the circuit works. Only the correct key will make the design work properly; otherwise, it may stop working or go into the wrong state.At the gate level, a circuit is made using standard logic parts. One method used here is logic locking. This involves adding extra gates to the circuit, so it won't work unless the right key is used. The key is a special binary code that matches parts of the circuit. Using gates like XOR or XNOR as key gates, the correct key makes them act like a regular wire, and the rest of the circuit works. If a wrong key is used, the gates will give wrong values, making the circuit not function. As shown in a diagram, these extra gates are placed between other parts of the circuit. Adding these gates adds a little extra work to the circuit but makes it more secure. These kinds of security issues have made hardware security a big topic, and designers are more concerned than ever.

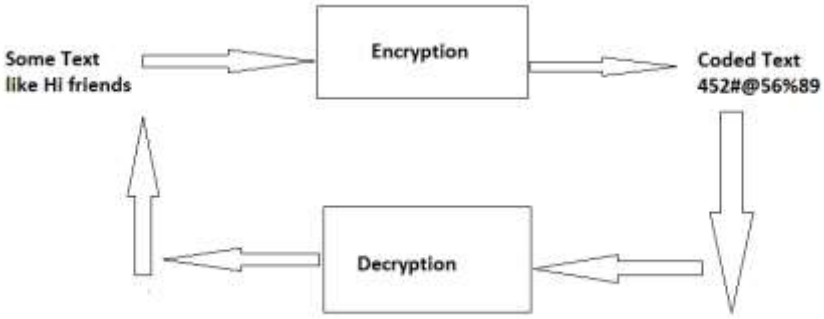
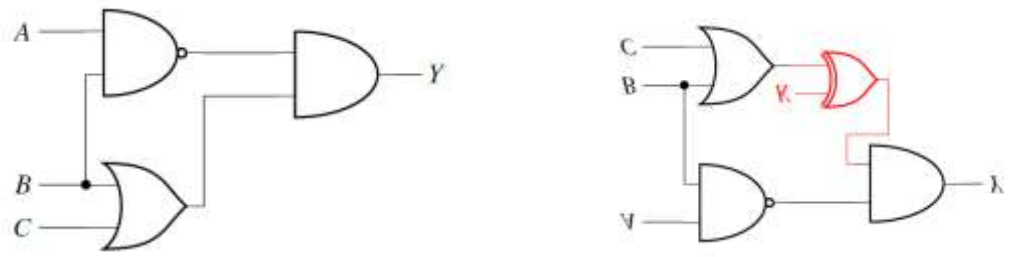


Figure 1: Encryption [3]



(a) (b)

Figure 2: An Example of a Logic Locking Circuit.

Another technique at the gate level is using finite state machines (FSMs) and authentication. An FSM can be added to the circuit to increase security. By adding more states than originally planned, it becomes harder for someone to understand the circuit. Only those with the right input pattern can move through the correct states. If the wrong input is given, the machine might get stuck in a loop and behave incorrectly. A diagram shows how only those with the right state can move through the added parts into the main design. Authentication can also stop people from making fake copies of the circuit. Only allowed people have access to a special key.

II Reverse Engineering

The method adopted are input/output logic locking for Protection Against Reverse Engineering Attacks using Artificial Intelligence. Used encryption and decryption techniques that use logic locking. It uses the layered structure of deep neural networks and the balanced behavior of the ReLU function to greatly reduce this types of problems. The method also includes a strict way to check and fix the key to make sure it's accurate. Testing shows that this method works well with big neural networks and different types of network designs.Deep neural networks (DNNs) have been very successful in many areas. However, these models are facing problems like security risks. Hackers want to steal the parameters of a DNN model for two main reasons. First, training a new model is costly. This is because deep learning needs a lot of data, which can be private or hard to get. Also, experts are needed to design the model, choose the right learning methods, and set up the right parameters. Plus, training takes a lot of computer power and time. Second, if the model's parameters are leaked, it can cause serious security issues and lose the model's secrets. If someone gets access to the model, they can carry out attacks like evasion or data poisoning to trick or harm the model. They can also use inversion attacks to get back the original training data from the model. In reverse engineering process used the XOR gate and utilize and mention the proper key for the security. Number of steps are their which present in the flowcharts.

DNN model

A DNN model can be represented as a function

$f : \mathrm{X} \rightarrow \mathrm{Y}$

Which takes inputs from the input space

$\mathrm{X} \subseteq \mathrm{R}^{P}$

and returns outputs to the output space

$\mathrm{Y} \subseteq \mathrm{R}Q$.

A k-layer deep neural network [6]

f= an alternating sequence of linear transformations

Total of k non-linear activation functions:

$f = f_{k+1} \circ \sigma \circ f_k \circ \cdot\ \cdot\ \cdot \circ \sigma \circ f_1$.

In this equation, the i-th hidden layer is given by a linear transformation fi followed by an element-wise ReLU activation function σ.

Specifically,

$fi\ (xi{-}1) = A^{(i)}xi{-}1{+}b^{(i)}$

is an affine transformation, in which the post activation hidden state

$x_{i-1} \in \mathrm{R}^{di-1} = d_{i-1}$

dimensional vector, the weights

$A^{(i)} \in \mathrm{R}^{di \times di-1} = di$ by d_{i-1} matrix

the biases $b_i \in R^{di} = di$-dimensional vector.

" Notice that the final output layer $fk+1$ is not followed by activation functions. All parameters, represented as $A(i)$ and $b(i)$ for $i \in 1, \cdots, k+1$, can be updated during training".

The above setup assumes that all the hidden layers are fully connected layers. However, even with these, the network can still be seen as a series of local linear transformations. Also, a softmax layer can be added to the output layer.

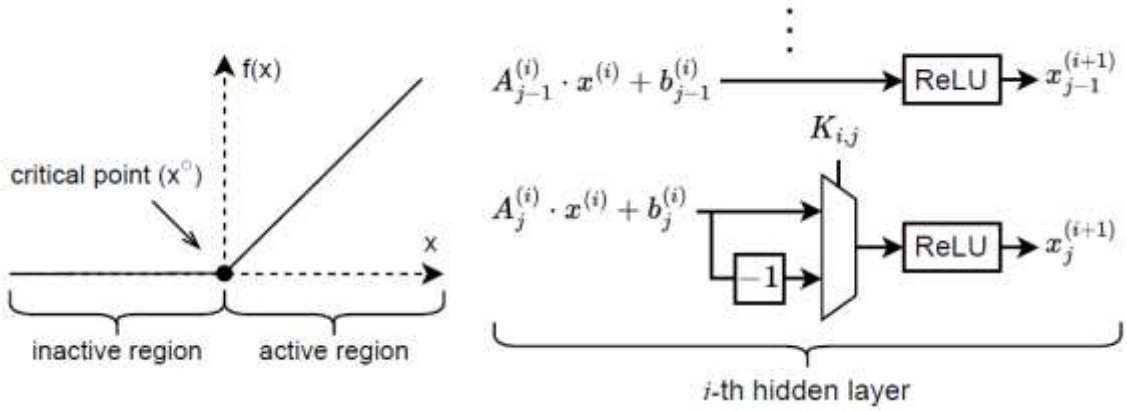


Figure 3: (a) The Rectified Linear Unit (ReLU) activation function. (b) An unprotected neuron (upper) and a protected neuron (lower) of HPNN.

Figure 3 shows that the Rectified Linear Unit (ReLU) activation function. The function is at the critical point if its input equals 0 and second figure shows that the An unprotected neuron (upper) and a protected neuron (lower) of HPNN. The inserted flipping unit negates the pre-activation value of the protected neuron when the key value equals 1.

In this case, given an input example,

$x \in X$

we call

x_{k+1}= the logits of $f(x)$

y = softmax(x_{k+1})

the output vector of $f(x)$.

Each component of σ is a ReLU activation function [7] defined as

$\phi(z) = \max(z, 0)$.

ReLU has established itself as the default choice for deep learning because DNNs with ReLUs can be optimized more efficiently [1].

As shown in Figure 1(a), a ReLU is a piecewise linear function. We denote the j-th neuron in the i-th hidden layer as $\eta i, j$.

It computes

$xi, j = \phi(A^{(i)}{}_j\, xi{-}1 + b^{(i)}{}_j)$,

where

$A^{(i)}j$ is the j-th row of $A^{(i)}$ and $b^{(i)}j$ is the j-th element of $b^{(i)}$.

So

$zi, j = A^{(i)}j\, xi{-}1 + b^{(i)}j$

As the pre-activation value of $\eta i, j$ then $\eta i, j$ is at its critical point if $zi, j = 0$.

Also, a neuron is inactive

If $zi, j \leq 0$ and is active if $zi, j > 0$.

III Bit Interference key functions Algorithm

Algorithm 1 steps: DNN Encryption Algorithm

*1.Input:Netwok white box f, Network Oracle O, Protected key neuron ηi, j , decrypted key values for preceding layer K*1 ……..K*i-1.*

*2. Output : K*i,j*

3. x◦ ← search_critical_point (ηi, j)

4.observe $m^1, \ldots \ldots m^{i-1}$ *with a forward pass from x◦.*

5. Compute as a mention in A^(i)

6. ei,j ← the j-th standard basis vector in R^di

7. find least square A^(i)vi,j=ei,j.

8.return if vi,j not exist.

9.if $o(x^{\circ}) \neq o\left(x^{\circ}+\in . v_{i,j}\right)$ *then return 0.*

10. if $o(x^{\circ}) \neq o\left(x^{\circ}-\in . v_{i,j}\right)$ *then return 1.*

11. Return

Algorithm 1 Details how we implement the key bit inference procedure.

- It starts by finding a critical point $x◦$ of the targeted neuron (Line 3).
- Then it computes the product weight matrix associated with the level-i linear region where $x◦$ is located (Line 4-5).
- Given the product weight matrix $A^(i)$, we are able to compute the pre-image vector $vi, j \in RP$ for ei, j .
- In practice, find the pre-image with least squares (Line 7), which is a built-in function provided by statistics and deep learning frameworks such as SciPy [29] and PyTorch.
- Finally, the algorithm determines the key bit through queries to the oracle network (Line 9-10). Notice that the statements on Line 9 and Line 10 are contra-positive to their counterparts in Lemma.
- In some rare cases, all the three values obtained from the oracle queries are close to each other.In that circumstance, we attempt to find another $x◦$ and start over the entire procedure.

IV Simulation and Results

To performing attacks used a well-known Python library. The algorithm showed how models are susceptible to attacks.The way of our experiments was by creating Python functions that take percentage and generate a functions.The process started by loading the content. The entire work at different percentages all at once and then split the work into training and testing sets.The iterative bit by bit, increasing by 1% each time until the required percentage is reached.Figure 4, the experiments showed that all techniques had similar effects in terms of how much the performance dropped due to poisoning.The reason is that a smaller impact is because of how the technique works. When using entire attracks are selected, and their labels are changed. Unlike other methods, looks at all points within a together rather than looking at individual points. Because of this, points are more likely to be surrounded by other points from the same node, which have similar features and labels. This natural group structure within nodes makes the overall impact of the poisoning attack less severe.

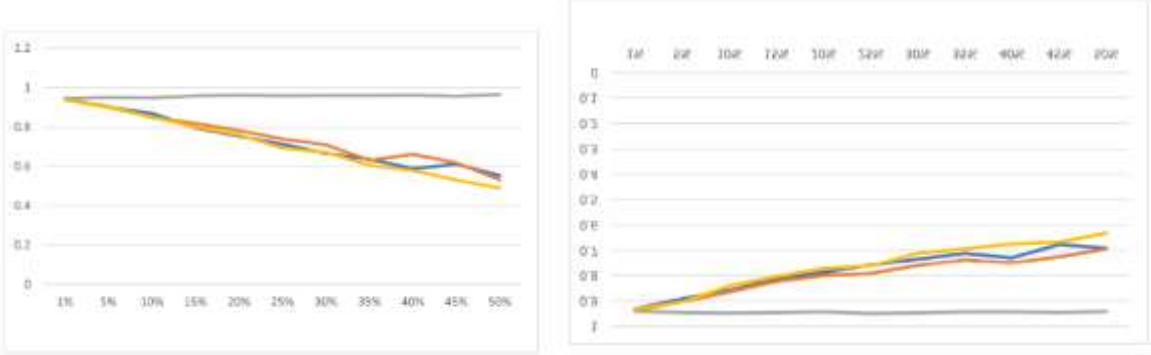


Figure4: Results for same nodes and iterative

In the divided and iterative case, some interesting things happen.The random method has almost no effect on the results, showing little influence. For both DNN methods, performance slowly decreases but stays steady, reaching around 0.76 score. This stability is because the poisoning is done step by step.In the case of k for the divided and iterative setup, the F1 score moves between about 0.93 and 0. 95. This behavior is because k is not always the same. It randomly chooses the starting points, but with the same node, similar clusters form each time. As poisoning is applied in steps, flipping one node and then flipping it back, the performance changes because of the way the clusters are formed. This back and forth leads to varying levels of misrepresentation and confusion, causing the performance to fluctuate.

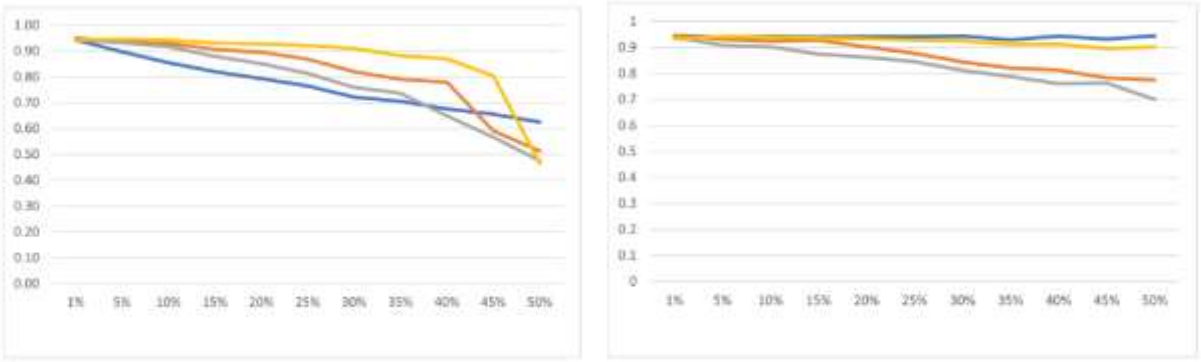


Figure 5: Results for divided nodes and iterative.

In summary, when the node is split, there are different effects compared to when the whole node is poisoned.Poisoning has a smaller overall effect, with random and DNN showing a slow decrease in performance, while k shows big changes.

V. Conclusion

An AI-based approach for identifying and stopping hardware attrack during different stages of IC design. AI-embedded routing optimization approach reduces the risk of signal leakage and makes the system more resistant to electromagnetic attacks.The key results are find infected nodes with up to 95% accuracy, which is better than other machine learning models used for different tasks. This method has a True Positive Rate of 97%. Federated Learning-based Security Frameworks allow secure checking of ICs without sharing design details, improving supply chain security.AI-assisted side-channel analysis: Using Deep Neural Networks helps detect power leaks and electromagnetic signals caused by attracks.Logic Locking Techniques which combines logic locking and layout hardening, increases the average success rate by 45% compared to other prevention methods.A complete hardware detection system using AI, which offers cost-effective detection and location.The AI models, including their accuracy, scalability, and ability to resist attacks.

References

[1] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, “Machine learning resistant strong PUF: Possible or a pipe dream?” in Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST), May 2016, pp. 19–24.

[2] Q. Guo, J. Ye, Y. Gong, Y. Hu, and X. Li, “Efficient attack on non-linear current mirror PUF with genetic algorithm,” in Proc. IEEE 25th Asian Test Symp. (ATS), Nov. 2016, pp. 49–54.

[3] J. Ye, Y. Hu, and X. Li, “POSTER: Attack on non-linear physical unclonable function,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 1751–1753.

[4] M. Majzoobi, M. Rostami, F. Koushanfar, D. S.Wallach, and S. Devadas, “Slender PUF protocol: A lightweight, robust, and secure authentication by substring matching,” in Proc. IEEE Symp. Secur. Privacy Workshops, May 2012, pp. 33–44.

[5] M. Rostami, M. Majzoobi, F. Koushanfar, D. S.Wallach, and S. Devadas, “Robust and reverse-engineering resilient PUF authentication and keyexchange by substring matching,” IEEE Trans. Emerg. Topics Comput., vol. 2, no. 1, pp. 37–49, Mar. 2014.

[6] M.-D. Yu, D. M’Raihi, I. Verbauwhede, and S. Devadas, “A noise bifurcation architecture for linear additive physical functions,” in Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST), May 2014, pp. 124–129.

[7] J. Ye, Y. Hu, and X. Li, “RPUF: Physical unclonable function with randomized challenge to resist modeling attack,” in Proc. IEEE Asian Hardw.-Oriented Secur. Trust (AsianHOST), Dec. 2016, pp. 1–6.

[8] J. Ye, Y. Hu, and X. Li, “OPUF: Obfuscation logic based physical unclonable function,” in Proc. IEEE 21st Int. On-Line Test. Symp. (IOLTS), Jul. 2015, pp. 156–161.

[9] G. T. Becker, “On the pitfalls of using arbiter-PUFs as building blocks,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 34, no. 8, pp. 1295–1307, Aug. 2015.

[10] J. Ye, Q. Guo, Y. Hu, H. Li, and X. Li, “Modeling attacks on strong physical unclonable functions strengthened by random number and weak PUF,” in Proc. IEEE 36th VLSI Test Symp. (VTS), Apr. 2018, pp. 1–6.