Santa Clara University

Scholar Commons

Electrical and Computer Engineering Senior Theses

Engineering Senior Theses

Spring 2021

Attacking Logic Locked Circuits Using Reinforcement Learning

Allen Shelton

Jake Mellor

Follow this and additional works at: https://scholarcommons.scu.edu/elec_senior

Part of the Electrical and Computer Engineering Commons

SANTA CLARA UNIVERSITY

Department of Electrical Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY

Allen Shelton and Jake Mellor

ENTITLED

ATTACKING LOGIC LOCKED CIRCUITS USING REINFORCEMENT LEARNING

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

BACHELOR OF SCIENCE
IN
ELECTRICAL AND COMPUTER ENGINEERING

Sara Tehranipoor 06/02/2021

Thesis Advisor(s) Dr. Sara Tehranipoor date

Shoba Krishnan June 4th 2021

Department Chair(s) Dr. Shoba Krishnan date

ATTACKING LOGIC LOCKED CIRCUITS USING REINFORCEMENT LEARNING

By

Allen Shelton and Jake Mellor

SENIOR DESIGN PROJECT REPORT

Submitted to
the Department of Electrical Engineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements
for the degree of
Bachelor of Science in Electrical and Computer Engineering

Santa Clara, California

Spring 2021

Abstract

Logic Locking is an emerging form of hardware obfuscation that is intended to be a solution to many of the trust issues associated with the modern globalized IC supply chain. By inserting extra key-gates into a circuit, the functionality of the circuit can be locked until the correct order of bits or “key” is applied to the key gates. To assess the strength of new logic locking techniques, we propose a new attack that uses deep reinforcement learning. This attack aims to test logic locking as well as evaluate reinforcement learning as a possible attack against logic locking. By using a deep Q-learning neural network, Q-values can be approximated and the model can be trained much faster than using traditional Q-learning. By allowing the model to change a single bit in the key each timestep, simulations of the circuit with the key produced can be run and the outputs can be compared to that of the unlocked version of the circuit, called an oracle. A reward is calculated based on how many bits of the locked circuit are correct and is used to reinforce the learning of the model. During the training phases of the model, the relationship between a highly correct key and how correct the outputs are for some random inputs is not strong, causing the model to struggle to learn quickly.

Acknowledgements

We would like to acknowledge Dr. Sara Tehranipoor and Micheal Yue for their support and advice during the duration of this project.

Table of Contents

Introduction

In the age of globalized supply chains, electronics producers are in need of ways to obfuscate their designs from untrusted foundries. Many design houses have shifted to outsourcing the fabrication of their integrated circuits (IC) because of the economic feasibility of using an offshore foundry. The trade-off is the decrease in security and privacy. The use of these untrusted foundries has led to many security issues such as IC piracy, cloning, and overproduction. The security flaws in ICs have also allowed attackers to extract sensitive information from these systems. The ICs at this stage are vulnerable to Trojans, physical tampering, and logic attacks.This has prompted researchers to develop security mechanisms to include on circuits to counter these attacks.

Hardware obfuscation is the idea that the structure and logic of an IC are hidden to prevent adversaries from accessing it [1]. Many forms of hardware obfuscation have been devel-oped to hide the functionality of an IC. One form of hardware obfuscation is logic locking, shown in Figure 1. Logic locking aims to protect the IC by inserting key gates to hide the logic in a netlist. The Inserted key gate is usually in the form of a logic component like an XOR, XNOR, or MUX. These key gates are inserted between logic components of a netlist and have an input of a key bit. When multiple key gates are inserted into the design,only a correct key will unlock the circuit. The key gates actas a buffer when the proper key value is provided. Otherwise,the key gate will change the logic of the system and generate different output for the IC. This idea adds an extra layer of security by hiding away the logic and functionality of the design and only providing trusted entities with the correct keyvalue.

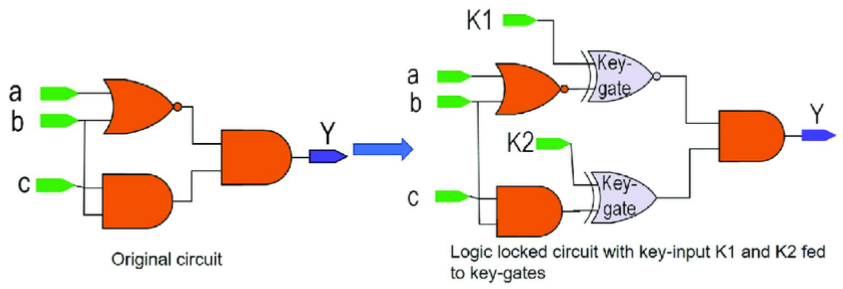

a
b
c
Y
Original circuit
K1
a
b
Key-gate
K2
c
Key-gate
Y
Logic locked circuit with key-input K1 and K2 fed to key-gates

Figure 1: Logic Locking Example

Logic locking provides extra security, but most of these schemes have been broken by more advanced attacks. Researchers have developed strong logic locking techniques in an effort to increase security in ICs. These techniques usually provide some type of key or have a golden circuit that represents the true design of the system. An attack on logic locking will try to decipher the key or the golden circuit.There are many types of attacks that have been developed against logic locking. An oracle refers to an unlocked circuit or a functionally correct circuit where an input would provide the correct output. Oracle-guided attacks use IO information to generate an algorithm while an oracle-less attack doesn't rely on an oracle. Structural attacks are based on using structural analysis to identify components of the system. The information gathered is used to create an algorithm for the attack. Functional attacks use functional analysis to determine various properties of the system. The basis of functional and structural analysis is used in many modern attacks on logiclocking. Over the past several years, various attacks have been applied to logic locking obfuscation techniques. These attacks are all unique, with some being simply functional like DoubleDIP [8] or are designed to observe the design structure like SAIL [6]. Multiple attacks combine both structural and functional concepts. For example, SURF [2] uses SAIL [6] to predicate a key that is used as the starting point of the key refinement algorithm. Similarly, FALL [9] uses comparators to identify key gates and their inputs for the algorithms that solve for the key. Other algorithms take less common approaches, like the Novel Bypass Attack [3], which uses an extra circuit to bypass the logic locking and fix corrupted outputs.

We seek to develop our own attack method to further the field of logic-locking attacks, which is a relatively new area of research. To the best of our knowledge, This is the first use of reinforcement learning being used in a logic-locking attack. We would like to explore the viability of reinforcement learning in exploiting hardware security measures in order to publish our work and allow other researchers to find more robust ways to secure hardware.

Problem and Objectives

Logic locking is a proposed solution to the trust issues associated with having a globalized IC supply chain. By having many different companies be involved in the entire process of designing and manufacturing IC's, there is opportunity for IP theft, overproduction, counterfeiting, and piracy. These trust issues end up taking millions from the profits of IC designers, so a solution is highly sought after. By using logic locking to essentially lock down the functionality of a circuit so that those who have the key can use it is a promising solution to the trust issues that are currently faced. As long as the keys are secure and their distribution is controlled, the IC's created with logic locking implemented should be unusable by unauthorized parties. That is unless they are able to determine the key of the given IC. Preventing this will require a strong and robust logic locking scheme, which needs to be found. Research on logic locking is split between developing new mechanisms and strategies for implementing key gates, and new ways to attack logic locked circuits. Eventually, this field will reach a point where a strong logic locking technique has been developed and is very difficult to defeat, much like RSA and AES in cryptography.

Our project aimed to help the field of hardware security and logic locking advance by evaluating a new kind of attack on logic locking as well as test the strength of logic locking and analyze its strengths and weaknesses. Our first objective was to review existing literature on logic locking attacks so we could get an idea of how attacks are usually executed and what works well and what does not, as well as performance metrics that we can use to evaluate our attack. Once we did this, we were able to write a paper surveying the attacks we researched and had it accepted into the International Conference on Consumer Electronics 2021.

Our next objective was to choose an attack that was new and hadn't been done before, and develop it so that we could get an idea of its performance. Our initial goal was for the attack to successfully extract the key from a logic locked circuit. We also wanted the attack to execute quickly. Knowing that reinforcement learning takes a lot of time to train, we didn't expect the execution time of the attack to be on the scale of minutes or seconds like other attacks we reviewed, but had set a target of 2 hours. Our final goal was to look at the data from our attack and reach some conclusions about it. We wanted to be able to decide if reinforcement learning is well suited as a logic locking attack and to see how strong logic locking is as a defense mechanism.

While we did not reach some performance oriented goals, we were able to reach our final goal of being able to come to a clear conclusion about the viability of reinforcement learning for this kind of application, and were able to make some decisions about how strong logic locking is currently.

Project Plan

As mentioned earlier, we will be using a reinforcement learning algorithm in order to carry out our hardware attack. Reinforcement learning is an area of machine learning that seeks to train intelligent agents to take actions in an environment that maximizes its cumulative reward. This iterative process is illustrated in Figure 2. At each time step, an agent receives an observation, which includes the current state and reward. Then the agent chooses an action from the set of available actions and it is sent to the environment. Then the environment changes in some way as a result of that action and moves to a new state and the next reward associated with that transition is determined. This feedback loop continues as the agent learns more information about its environment and how to make the best actions. For our project, we needed to define our problem, a logic locking hardware attack, in the context of a reinforcement learning model in order for this method to work, since reinforcement learning has never been used before in a hardware attack. At the very least, we needed to define our agent, our environment, what constitutes an action, how we define a state, and a reward function. These definitions are shown here:

Agent - Key bits (or the algorithm for finding the key bits)

Environment - Circuit trying to be deobfuscated

Action - Flipping 1 key bit

State - The values of the key bits

Reward - Output accuracy from a fixed number of inputs with the current key value compared to the correct outputs with correct key value

Our plan for this project was to design our own logic locking attack using reinforcement learning. We would then implement this attack on various benchmark circuits using an existing logic locking technique. For our attack, we were to design and train a reinforcement learning model, as well as an algorithm around it that will be able to extract the correct key bits from logic locked circuits. After developing this algorithm, we will need to fine tune the hyperparameters of this algorithm in order to increase its performance, both in terms of accuracy and time. We would then collect data from the attacks we run on these benchmarks and compare our results to other known logic-locking attacks. We hoped to achieve 100% key accuracy with this attack, as well as have an execution time that competes with other existing state-of-the-art attack methods. All of our code was written in Python.

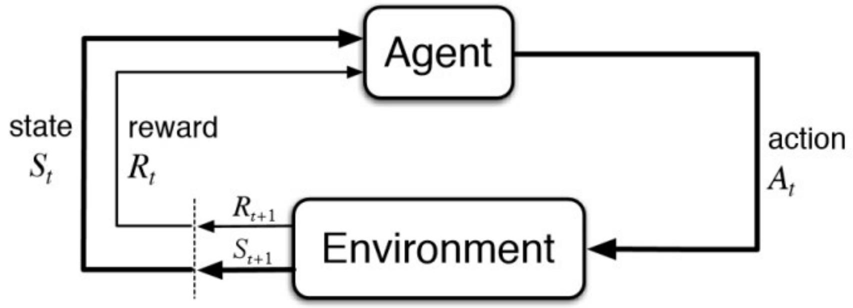


Figure 2: Reinforcement learning feedback loop

Our first task was to obtain the obfuscated benchmark circuits that we would be using to test the performance of our attack. These benchmarks were provided to us by a colleague, written in Verilog and already obfuscated. These benchmarks were obfuscated using random, logic cone analysis, and SLL methods [13].

Our algorithm required the creation of a number of different components that would all work together. The next task was to create a netlist parser component for our benchmarks. This netlist parser will be used to extract the input signals, output signals, and key bits from the netlist files. This parser will be used in conjunction with the simulation manager so that our algorithm can pass inputs into the benchmark circuits along with the key values that it generates and capture the outputs.

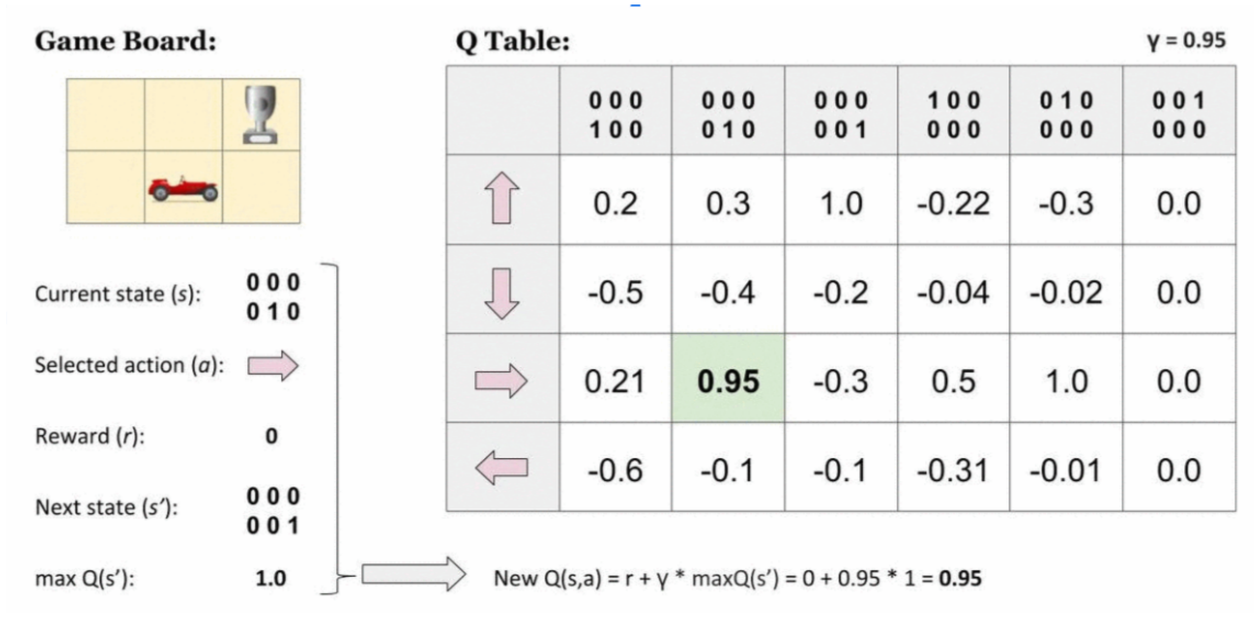
The next component was the simulation manager. As mentioned, this component acts as an intermediary between our attack algorithm and the netlist circuit. As our hardware attack continues to run, it will generate new key values that need to be tested on the benchmark circuit. The simulation manager will take this key value, along with a number of random inputs, and generate a Verilog testbench to pass these inputs into the circuit and capture the outputs.

The next component is the oracle comparison. This will take the circuit outputs that were obtained with a given set of inputs and the key generated from the algorithm and compare them to the outputs from those same set of inputs with the correct key value in order to generate a reward for our algorithm. The higher the percentage of output bits that are correct, the higher the reward will be. Since this initial reward scheme that was generated is a percentage, it’s value will be between 0 and 1. As you will see later, we slightly modified our reward scheme to try to improve the performance of our reinforcement learning model.

The next and most important component of our algorithm is our reinforcement learning model. The model will be trained by generating key values, deciding actions to take, generating rewards based on these actions, and passing these experiences into a neural network to learn how to find the correct key value. This part of the project was created using PyTorch, an open source machine learning library that allows users to create neural networks for AI applications. All data that was used by PyTorch's library functions have a special data type called a tensor. This data type is very similar to an array or matrix, and it's used to encode the inputs, outputs, and parameters of any model created.

There were three main sections of our reinforcement learning model, the deep Q-network, the epsilon greedy strategy, and the replay memory. In traditional reinforcement learning, specifically in the type used in our project called Q-learning, how an agent learns about its environment is by using what is known as a Q-table, where an example with a car moving through a grid is shown in Figure 3. In this example, one axis has all possible states, which in this example correspond to the car being in any of the six squares on the grid, and the other axis has all possible actions, which are the directions up, down, left, and right. As an agent gains experience by taking actions from different states, a Q-value is determined for each state-action pair, where a Q-value denotes the future return of taking an action from a given state. The values in this table will be iteratively updated as the agent continues to explore its environment and find the optimal path to achieve its goal, reaching the trophy square.

As the state space becomes too large, however, traditional Q-learning becomes infeasible. The benchmark circuit we're trying to attack has anywhere from 32 to 256 key bits. Even with 32 bits, that maps to 2^32 possible states, so trying to store a Q-table for this attack, if possible, would make it extremely slow and inefficient. So we have decided to use deep reinforcement learning, which combines deep learning and reinforcement learning. Instead of using a Q-table to store and update Q-values, a neural network is used to approximate the Q-function, and this network is trained on the experiences of the agent in its environment. The network takes in the current state, which is the key value, as input, and each output neuron will be the approximate Q-value for every action that can be taken from the state that was passed into it.



	000 100	000 010	000 001	100 000	010 000	001 000
⇧	0.2	0.3	1.0	-0.22	-0.3	0.0
⇩	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
⇨	0.21	**0.95**	-0.3	0.5	1.0	0.0
⇦	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Figure 3: Reinforcement learning example

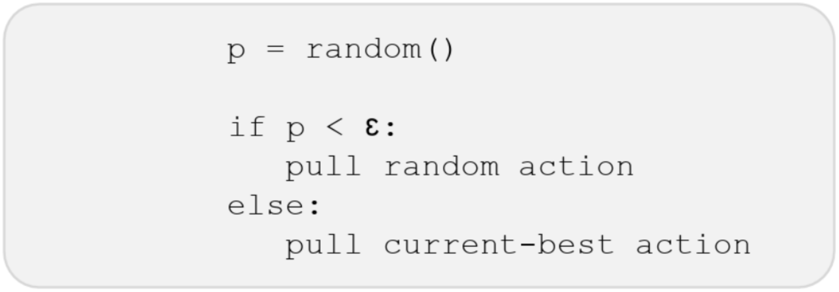
Note that in traditional Q-learning, the Q-table is updated by finding the loss between the calculated Q-value from the current state-action pair and the optimal Q-value, which is given by Equation 1:

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Equation 1

This equation can be interpreted as saying the optimal Q-value for a given state action pair is equal to the expected value of the sum of the immediate reward from that state-action pair and the scaled maximum Q-value of the next state among all possible next actions. This scaling factor is called a discount factor, which takes into account how much the reinforcement learning algorithm will take into account future returns, which is what the Q-value represents, in relation to immediate return. In deep reinforcement learning, the loss between the current and optimal Q-values is used to train the network weights through gradient descent and backpropagation. Both the current Q-value and the "optimal" Q-value require a pass through the Q-network in order to be calculated, since we do not actually know the optimal q-values ahead of time. Using the same neural network to find both values will cause network optimization to be chasing its own tail, since both the current Q-values and the optimal q-values are being changed at each training step. We want the optimal Q-values to be somewhat fixed. Therefore, we are using two neural networks for our reinforcement learning algorithm. The policy net will have weights that are trainable based on the calculated loss. The target net will have fixed weights and are used to find the optimal Q-values, and every certain number of timesteps, the target net weights will be updated to match the policy net weights. This removes the instability introduced in using only one network for both values.

We chose to incorporate an epsilon greedy strategy in order to mitigate how the agent chooses actions, where as mentioned earlier, an action is flipping one bit in the current key value. This strategy tries to find an optimal balance between two concepts of agents taking actions called exploration and exploitation. Exploration is the act of exploring the environment to find out information about it. Exploitation is the act of exploiting the information that is already known about the environment in order to maximize the return. Both are useful for the agent to navigate its environment, and using only one of them will not lead to good results. If only exploration is used, the agent wouldn’t be able to take advantage of the information learned to optimally achieve its goal, and if only exploitation is used, it might only seek immediate reward, even though its behavior is suboptimal in trying to achieve its ultimate goal. The epsilon greedy strategy incorporates both through a parameter called an exploration rate, and the pseudo code for the strategy is given in Figure 4. At every timestep of the algorithm, a random number is generated between 0 and 1. If the number is less than the exploration rate, a random action is chosen (exploration), else the action that promises the highest immediate return is chosen (exploitation). In our algorithm specifically, determining the action that gives the highest immediate reward is found by passing the current state into the policy net, and choosing the action that corresponds to the output neuron with the highest value. To reiterate, each output neuron in the policy net represents the approximate Q-value, or long-term return, of taking each action from a given input state.



```
p = random()

if p < ε:
   pull random action
else:
   pull current-best action
```

Figure 4: Action decision pseudocode

The last component of our reinforcement learning model is the replay memory. With a concept known as experience replay, the agent's experiences at each time step is stored in a data structure called the replay memory. An experience is defined as the tuple shown in Equation 2:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

Equation 2

where s_t represents the current state, a_t represents the action taken from that state, r_{t+1} represents the reward for taking that action, and s_{t+1} is the new state. A batch of these experiences will be sampled and used to train the Q-network at each timestep of the algorithm. The main advantage of using experience replay instead of providing the sequential experiences to the Q-network as they occur in the environment is to break the correlation between consecutive samples. If only consecutive samples are passed into the network, the sample would be highly correlated and lead to inefficient learning.

A complete flowchart of a single timestep of our algorithm is shown in Figure 5. At the start, the epsilon greedy strategy decides an action for the agent to take in its current state. Then, that action is executed, and both a reward and a new state are determined. This process constitutes an experience, and is stored in replay memory. After that, a random batch of experiences is sampled from replay memory. These experiences are passed into the policy net and target net in order to calculate the loss. This loss is then used to perform stochastic gradient descent and backpropagation to train the policy net. We then check if a certain number of timesteps have passed since our last update of the target net. If it has, we change the weights of the target net to match those of the policy net. We then move on to the next timestep.

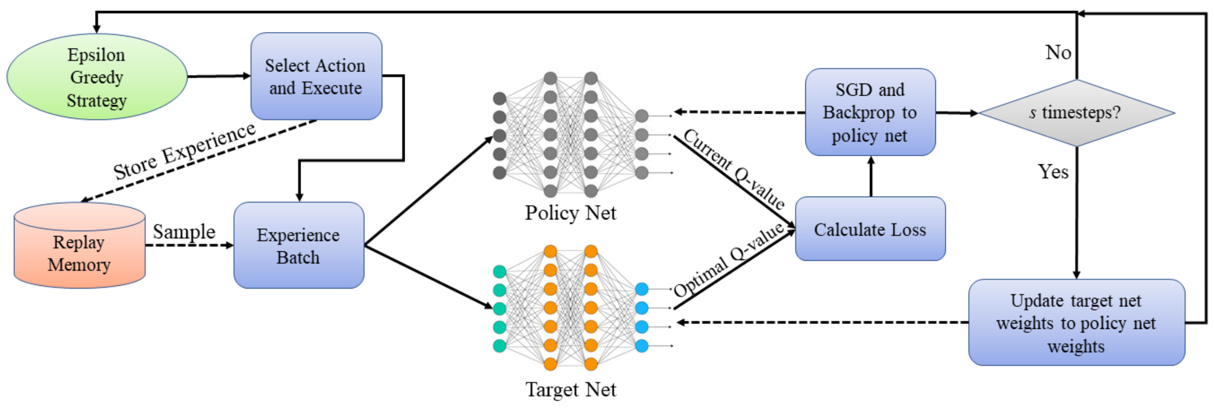


Figure 5: Deep Q-Learning training loop

Project Outcomes

One of the first things we did to prepare for this project was to read and synthesize several academic papers about the state-of-the-art hardware obfuscation attacks that exist in the hardware security industry today. Learning about these attacks not only gave us more knowledge about the hardware security industry, but many of the techniques used in these papers inspired aspects of our own attack. After reading about these different attacks, we wrote our own paper, a survey of all the logic-locking attacks we read about, and submitted this paper to the International Conference on Consumer Electronics (ICCE). Our paper was accepted to this conference, and we were able to present our work at this conference virtually in winter of 2021.

Once all of the individual components of our algorithm were made, they had to be combined together as a full working system. This process involved a bit of modification to how we initially coded our individual components. Because for the most part many of these components were created in parallel between each of us, there were some inconsistencies at first with the data types the modules we created accepted as input and the data types it gave as output. We tried to be in communication with each other as much as possible as we were creating these modules to try to avoid these issues. Eventually we were able to make sure the data being passed around to the different components of our algorithm were consistent and in a state where the components could actually perform operations on it, allowing us to be able to safely combine all components in our systems. The block diagram of our full design is shown in Figure 6. The diagram shows all the major components of our attack and the data that is being passed between them.

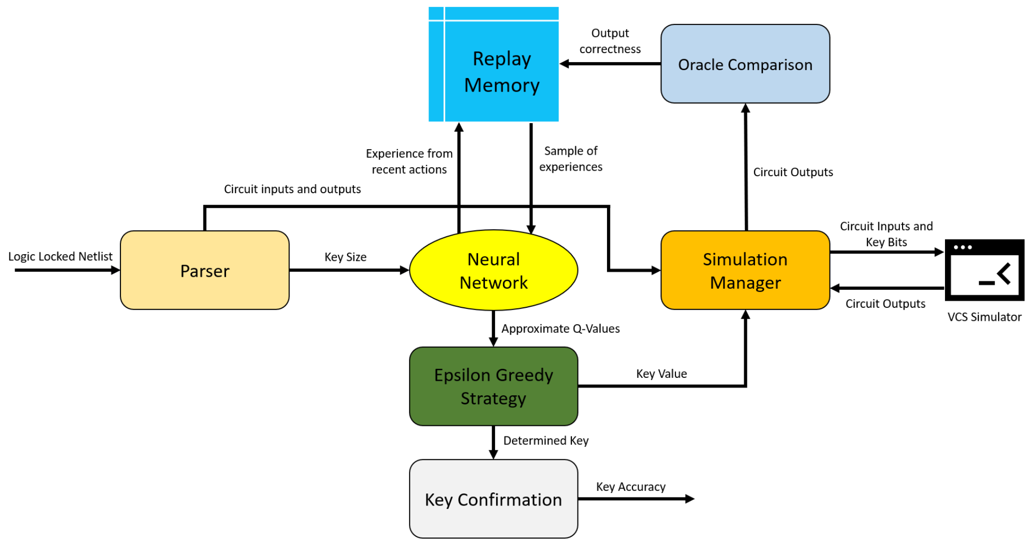


Figure 6: Block diagram of reinforcement learning attack

There are two phases to our attack: the training phase and the testing phase. The training phase is what we've already explained and requires all components of our system to function. Through training, the reinforcement learning model should gain more knowledge about the circuit and the key value as the agent continually performs actions and gains rewards, and this information is used to modify the network weights in a way that reinforces this knowledge. The testing phase requires only the deep Q-network and a key confirmation program that calculates the accuracy of a determined key relative to the correct key value of our benchmark circuits. The basic flow of the testing phase is shown in Figure 7. Once the policy net finishes training, we freeze its weights and generate a random starting state. We pass that state into the policy to get its outputs. We choose the action that corresponds to the output neuron with the highest value, or the action that has the highest Q-value, and perform that action on the state, which will flip a specific key bit and give a new state. This new state is passed back into the network, and this process continues. At each iteration, the key value is passed into the key confirmation program to test its accuracy, and the testing phase should stop automatically when the key is 100% accurate. A discussion of further improvements made to our algorithm and the results we obtained are discussed in the subsequent sections.

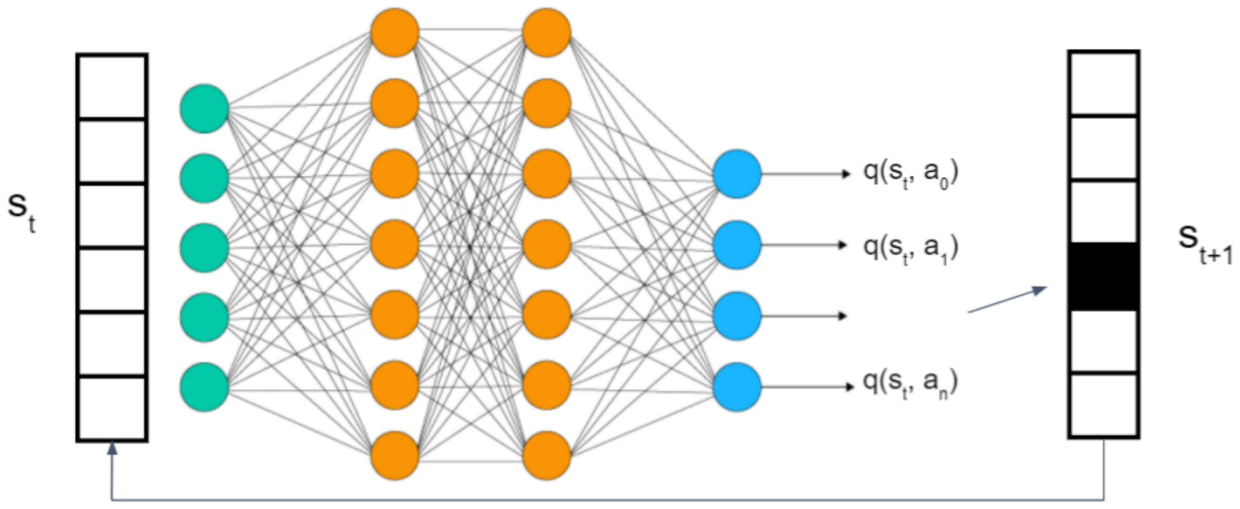


Figure 7: Testing phase loop

Final Design

We had made many improvements to our model and algorithm since when we first combined our code and initially developed the modules. While many of our improvements were made with the hopes of enabling our model to learn, they still had fundamental benefits and reasons for being added. Once we squashed all the initial bugs with our code, the first and most important improvement we made was changing our reward generation structure. Originally, the reward that reinforced the neural network was just the average percent of correct output bits from the batch of simulations that was run in each timestep. However, we noticed that the distribution of reward and how correct the current key is for that timestep had poor correlation. This can be seen below in Figure 8.

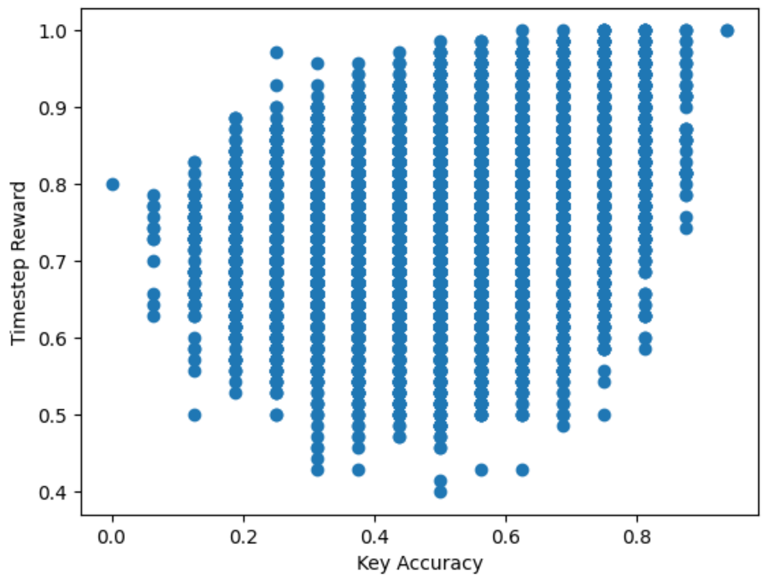


Figure 8: Key Accuracy vs Reward before new reward function

In order for the model to train effectively, we needed this reward scheme to have a stronger correlation between the model being in a state that is mostly correct and it receiving a strong reward. To accomplish this, we combined two new strategies. The first was to increase the number of simulations per time step. Since the input stimulus for each pair of unlocked and locked circuit simulations were random, the outputs would also be random. Therefore, for any simulation regardless of how correct the key being used is (unless that key is 100% correct) then the circuit's outputs can be correct to a widely varying degree, as shown by the data in the figure above. Increasing the number of simulations meant that each timestep would receive a reward that is closer to the true correctness of the key. However, this solution still allowed for the problem of a key that is not very correct, generating a reward that is almost as great as the reward generated by a key is almost entirely correct.

To mitigate this, we needed a reward scheme that would give out much greater rewards when the model was in a very correct state, and very small rewards for when the model was in a largely incorrect state. After some experimenting, we decided to use the following exponential function to generate a reward: $R = e^{6T_{acc}-3} - 0.05$, where R is the reward, and T_{acc} is the average accuracy of the outputs of the simulations in that timestep. This gave a curve that was more rewarding when the model was in a more correct state, and skewed rewards to be much greater as state accuracy increased. Below in Figure 9 shows the results of these changes.

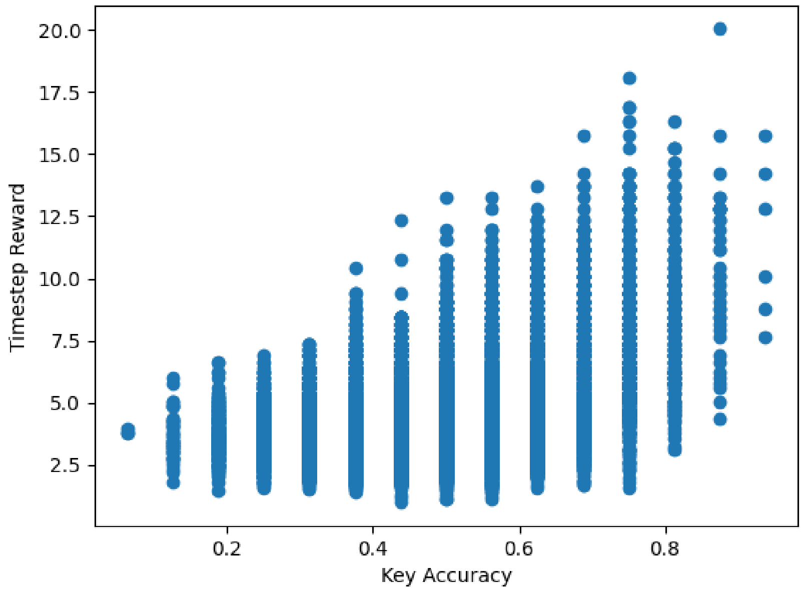


Figure 9: Key Accuracy vs Reward with new reward function

This data shows that while we did begin to get a better correlation between the state accuracy and the reward, we still were not getting exactly the curve we hoped for. One big issue is that the key-state that is almost entirely correct generates weak rewards. This is due to the randomness of the outputs. It could be mitigated with more simulations per time step but that would just add time to our already lengthy training process. We could also just increase the steepness of the exponential function we are using to generate the reward, but we are unsure what effect having such a wide range of reward values could have on the training process. But we believe that we will never get a truly strong correlation between key-state accuracy and the reward, due to logic locking and how it unpredictably impacts the circuit outputs. We believe that this may be one of the larger reasons our model was unable to train and believe that this is a major reason why the model couldn't learn.

The second major improvement was changing our activation functions. During our research we learned about something called the "Dying ReLU Problem". We originally had built our neural network using Rectified Linear Units (ReLU) for our activation functions. These units output a 0 for all values less than or equal to 0, and have a linear function for all values greater than 0. ReLU has become the most commonly used activation function for deep learning due to its capability to output a true 0 value. However, this can also cause certain paths of the network to "die" where they never get updated because they are often 0. This prevents backpropagation from updating the weights of these nodes and thus the model fails to learn. To remedy this issue, we changed our ReLU's to be Leaky ReLU's. Leaky ReLU's have a small negative slope for all values less than or equal to 0, which helps back propagation recover the dead nodes and help their weights update. In Figure 10 below these two activation functions are compared.

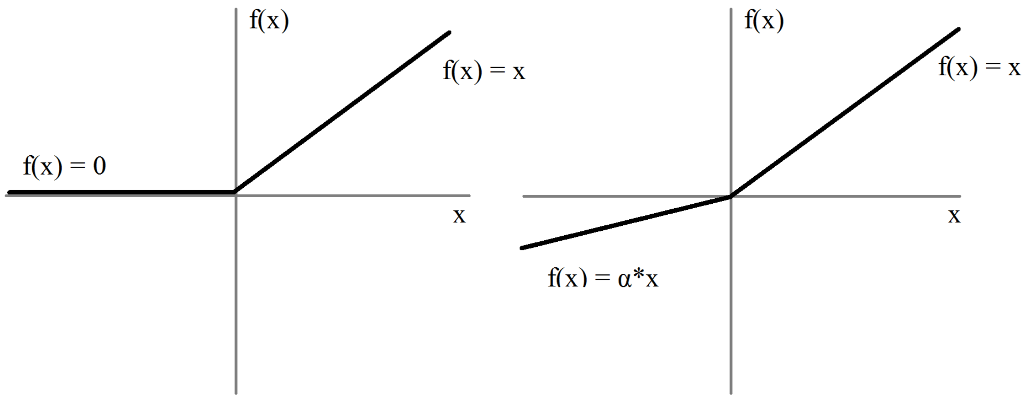


Figure 10: Left: ReLU Activation Function Right: Leaky ReLU Activation Function

Professional Issues and Constraints

Due to this project being done entirely in software with no hardware components at all except computers to train the model on, we were largely unconstrained. Our biggest issue was the lack of facetime due to COVID. Not being able to work together on the project slowed down progress. We encountered some other issues due to the nature of the project. Since our project is research based and is using deep learning in a way that has never been done before, finding help when things went wrong was difficult. Experts we know who are familiar with deep learning or reinforcement learning were unable to figure out why our model failed to learn. As neither of us were experts on reinforcement learning before this project, many of the issues we faced we had to figure out ourselves instead of finding the solution online.

Another issue we faced was the power of our hardware limiting how much we could train. We originally were using SCU’s Linux machines to train the model, but these machines were quite slow. We decided to try using one of our personal PC’s had a dedicated GPU with Tensor cores within it, and a higher speed processor than the Linux machines. Using GPU acceleration and overall faster PC, we saw a 47x speed up in training over the Linux workstations. This helped us save a lot of time with training, but reducing training time more would also have been helpful. There is always a possibility that with more training the model would eventually learn, but with the current hardware we have that would have taken far too long to be feasible.

In addition to needing faster hardware, a faster Verilog simulator could have also been of benefit. We originally started with Synopsys VCS to simulate the circuits, but this was severely limiting how fast we could train. When moving from the Linux workstations to our personal PC, we had to install a new Verilog simulator since we couldn’t install VCS locally without paying for a license. We found Icarus Verilog, which is an open-source Verilog simulator. This simulator was much faster than VCS and also helped in the massive speedup we saw over the Linux workstations. Even though Icarus Verilog did speed up the training process, it was still the limiting factor in how fast we could train. Per timestep of the algorithm, most of the time was spent waiting for the simulator to respond with circuit outputs, instead of doing operations that are training the model. Overall, this speedup was massively helpful in working on this project as it greatly reduced the amount of time we had to wait during training before we could analyze the data we logged and make improvements to try and get the model to function as desired.

There were no safety concerns associated with our project. Hardware obfuscation attacks like these are usually done entirely in software, and that is what we chose to do for our project. There was no use of any hardware for this project besides the computers we were running the code on, and so there were no real hazards associated with our work. None of the circuits we tested our attack on were fabricated ICs. They were all circuit netlists written in Verilog. In a real hardware attack, however, one would need to use the real ICs if that is all they had access to.

The entire field of hardware security is addressing ethical concerns. The field is seeking to stop malicious third parties with the intent to steal the intellectual property of chip designers using hardware attacks. The intent of the hardware attack we developed is purely for research purposes. Our goal was always to find vulnerabilities in logic locking techniques through our attack in order to publish those findings so that other researchers can develop more robust security measures that address those vulnerabilities. We are simply following in the footsteps of others who have developed strong logic-locking attacks for the purpose of further adding to the body of knowledge in the hardware security field, and not further complicating the field with malicious intent.

Conclusions

With this project, we sought to design and implement our own logic-locking circuit attack in order to assess the effectiveness of reinforcement learning on this type of hardware security technique. This was the first use of reinforcement learning as a means of implementing a hardware attack.

We learned current hardware attacks that have already been developed by researchers to become more knowledgeable on the field, and we compiled that knowledge into a survey paper that was accepted to ICCE 2021. Inspired by the papers we read, we designed a new hardware attack using reinforcement learning. We obtained obfuscated benchmark circuits to test our attack on, programmed the individual components of our algorithm, and combined them into a complete system. We spent a considerable amount of time fine-tuning the hyperparameters of our systems as well as making small fundamental improvements to the algorithm with the purpose of increasing its performance in regards to time and accuracy. Ultimately, this attack was unable to accurately extract the correct key value from any of our benchmark circuits, showing that logic-locking is an effective security measure against reinforcement learning-based hardware attacks.

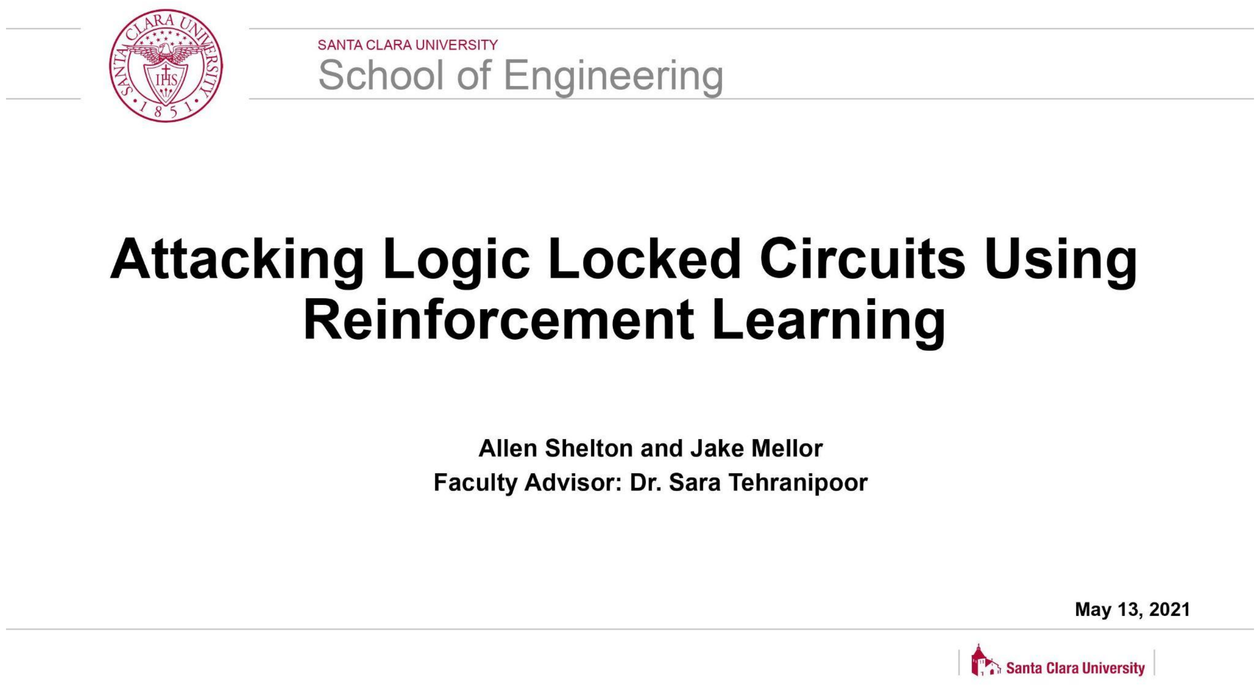
References

1. Forte, D., Bhunia, S. and Tehranipoor, M.M. eds., 2017. Hardware protection through obfuscation. Springer International Publishing.
2. P. Chakraborty, J. Cruz and S. Bhunia, "SURF: Joint Structural Functional Attack on Logic Locking," 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 2019, pp. 181-190, doi: 10.1109/HST.2019.8741028.
3. Xu, X., Shakya, B., Tehranipoor, M. and Forte, D., 2017. "Novel Bypass Attack And BDD-Based Tradeoff Analysis Against All Known Logic Locking Attacks," Lecture Notes in Computer Science, volume 10529.
4. Y. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," 2015 16th Latin-American Test Symposium (LATS), Puerto Vallarta, 2015, pp. 1-6, doi: 10.1109/LATW.2015.7102410.
5. K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2017, pp. 95-100, doi: 10.1109/HST.2017.7951805.
6. P. Chakraborty, J. Cruz and S. Bhunia, "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Hong Kong, 2018, pp. 56-61, doi: 10.1109/AsianHOST.2018.8607163.
7. Fatemeh Tehranipoor, Nima Karimian, Mehran Mozaffari Kermani, and Hamid Mahmoodi. 2019. Deep RNN-Oriented Paradigm Shift through BOCANet: Broken Obfuscated Circuit Attack. In Proceedings of the 2019 on Great Lakes Symposium on VLSI (GLSVLSI '19). Association for Computing Machinery, New York, NY, USA, 335–338. DOI:https://doi.org/10.1145/3299874.3318031
8. Yuanqi Shen and Hai Zhou. 2017. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In Proceedings of the on Great Lakes Symposium on VLSI 2017 (GLSVLSI '17). Association for Computing Machinery, New York, NY, USA, 179–184. DOI:https://doi.org/10.1145/3060403.3060469
9. D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," in IEEE

Transactions on Information Forensics and Security, vol. 15, pp. 2514-2527, 2020, doi: 10.1109/TIFS.2020.2968183.

10. F. Yang, M. Tang and O. Sinanoglu, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) – Unlocked," in IEEE Transactions on Information Forensics and Security, vol. 14, no. 10, pp. 2778-2786, Oct. 2019, doi: 10.1109/TIFS.2019.2904838.
11. M. Yasin, J. J. Rajendran, O. Sinanoglu and R. Karri, "On Improving the Security of Logic Locking," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 9, pp. 1411-1424, Sept. 2016, doi: 10.1109/TCAD.2015.2511144.
12. M. Yasin, B. Mazumdar, J. J. V. Rajendran and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2016, pp. 236-241, doi: 10.1109/HST.2016.7495588.
13. F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," in Proc. of the International Symposium on Circuits and Systems, 1985, pp. 663-698.

Appendix A: Senior Design Presentation Slides

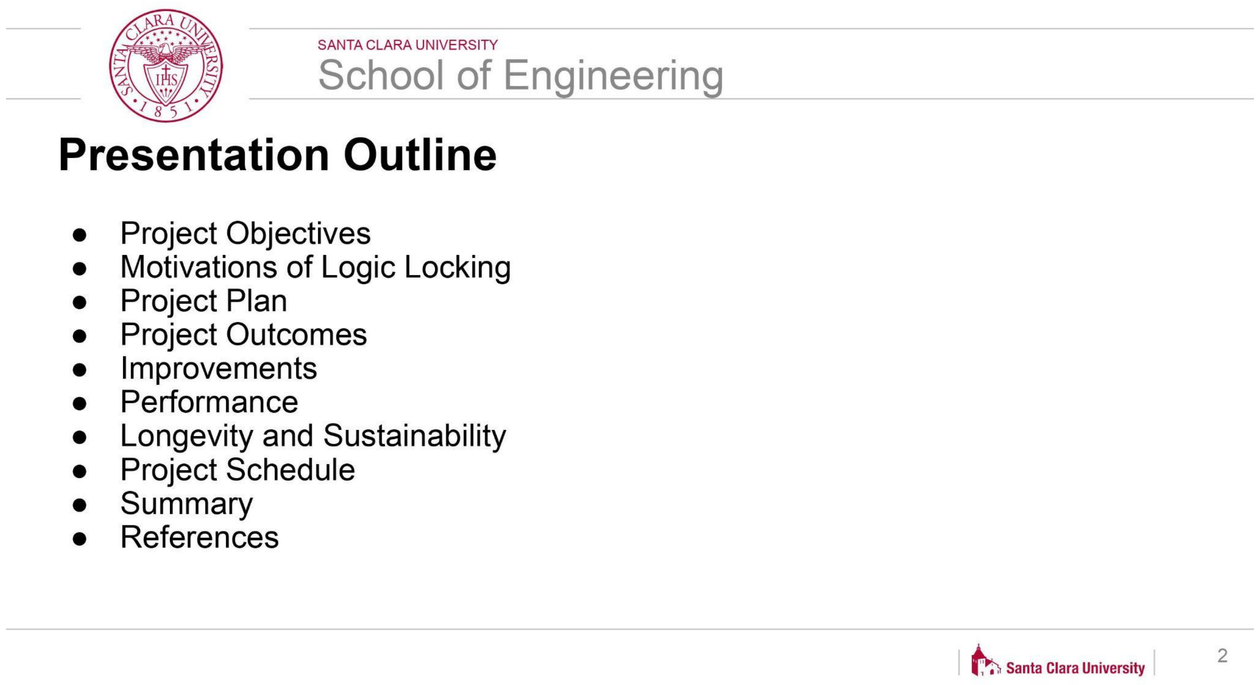

SANTA CLARA UNIVERSITY
School of Engineering

Attacking Logic Locked Circuits Using Reinforcement Learning

Allen Shelton and Jake Mellor
Faculty Advisor: Dr. Sara Tehranipoor

May 13, 2021

Santa Clara University


SANTA CLARA UNIVERSITY
School of Engineering

Presentation Outline

- Project Objectives
- Motivations of Logic Locking
- Project Plan
- Project Outcomes
- Improvements
- Performance
- Longevity and Sustainability
- Project Schedule
- Summary
- References

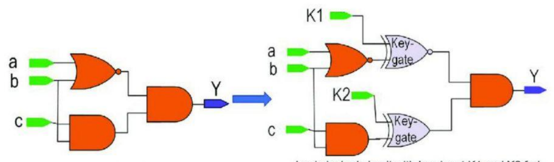
Santa Clara University 2

Project Objectives

- Conduct research and review existing literature on logic locking attacks
- Analyze existing vulnerabilities of logic locking circuits
- Develop a reinforcement learning attack
- Test our attack against various locked benchmark circuits
- Collect data
- Evaluate the performance of our attack

Motivation of Logic Locking

- Hardware Obfuscation is the answer to many of the trust issues associated with IC fabrication
- Logic Locking is a form of Hardware Obfuscation, but due to its recency its vulnerabilities have not been deeply explored
- We plan to evaluate the robustness of Logic Locking by trying to exploit its vulnerabilities



Original circuit

Logic locked circuit with key-input K1 and K2 fed to key-gates

Project Plan

- **Approach**
  - Implement logic locking on well known benchmark circuits using an already existing logic locking technique
  - Develop a reinforcement learning model that determine a highly accurate key for a logic locked circuit
  - Iteratively fine tune the model's parameters to get desired performance
  - Collect data from the model

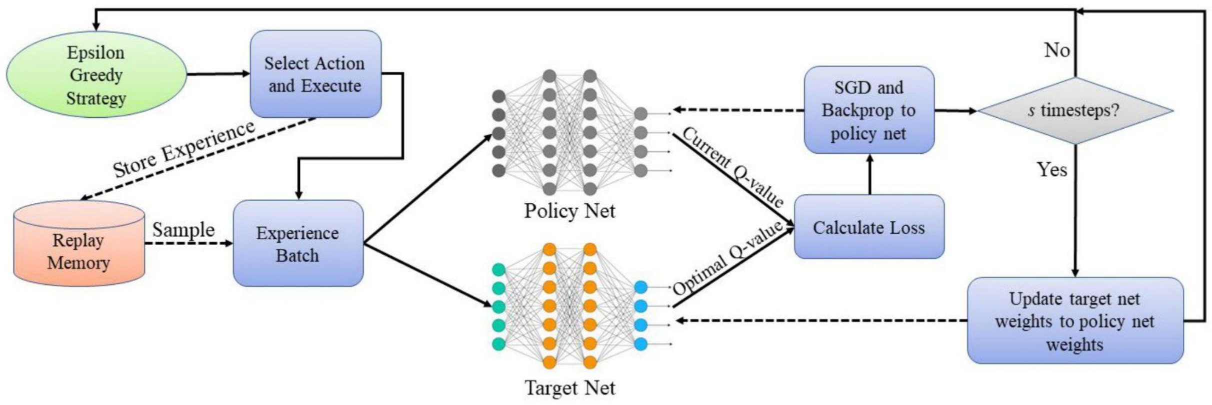
Project Plan Cont.

- **Engineering Tasks and Division of Labor**
  - Logic Locking Algorithm
    - Use already locked circuits with well known locking techniques
  - Netlist Parser (Jake)
    - Parses locked netlist and gets key size and net names
  - Simulation Manager (Jake)
    - Runs simulations of circuit with random inputs and specified key from algorithm
  - Oracle Comparison (Jake)
    - Compares locked circuit outputs to that of an unlocked circuit
  - Reinforcement Learning Model (Allen)
    - This will take a lot of time to first get to work, then to fine tune for desired performance
  - Key Confirmation (Allen)
    - Basic program that analyzes how many bits of the produced key are correct
  - Testing and Debugging (Jake and Allen)

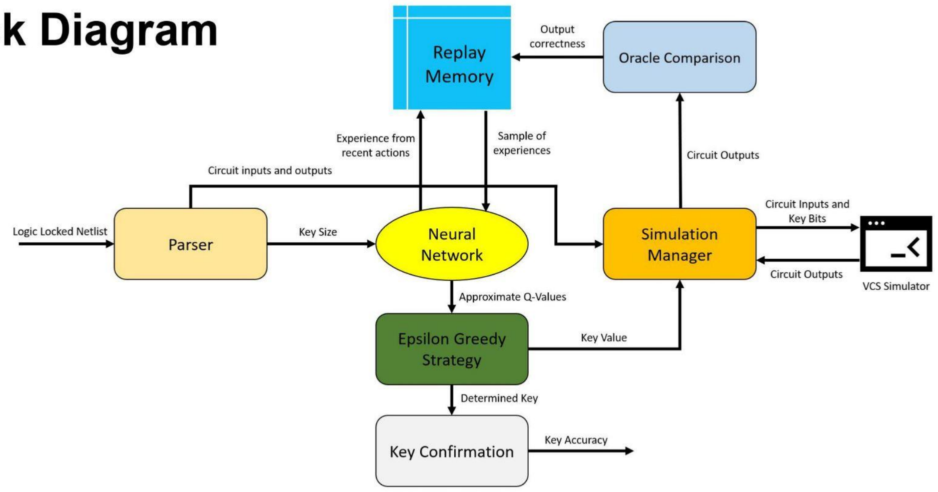
Project Plan Cont.

- **Anticipated Outcomes**
  - Functioning Reinforcement Learning Model
  - Successful key extraction
    - Aiming for 100% of key bits on average to be correct
  - Data that evaluates the performance of our proposed attack
    - Compare to other logic locking attacks
  - Algorithm takes less than 2 hours to complete with high accuracy

Reinforcement Learning Feedback Loop



Block Diagram

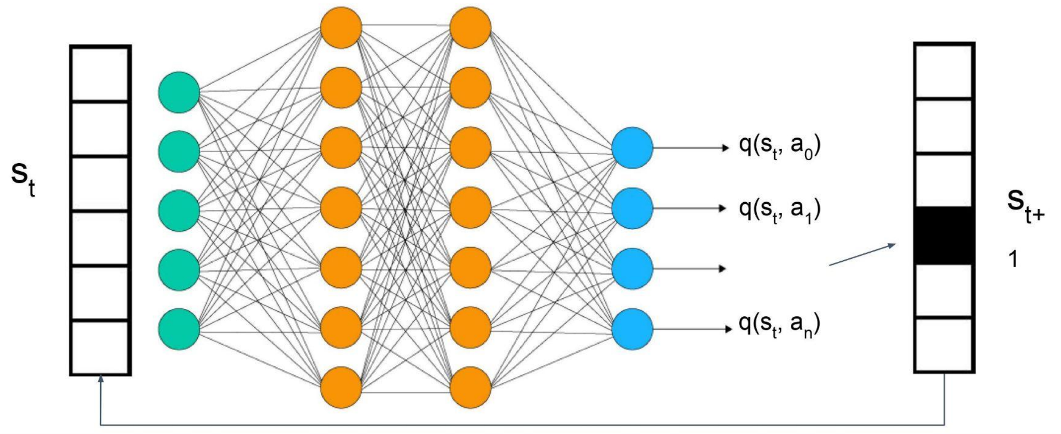


Hyperparameters

- Network depth
- Neurons in each layer
- Replay Memory size
- Batch size
- Epsilon decay
- Learn rate
- Target update
- Number of episodes

SANTA CLARA UNIVERSITY
School of Engineering

Testing



SANTA CLARA UNIVERSITY
School of Engineering

Project Outcomes

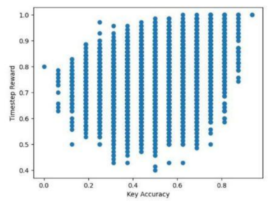
- Read current academic papers on logic locking and logic locking attacks
- Used what we learned to develop how we want our attack to work
- Wrote a paper surveying the papers we read
    - Our paper was accepted into International Conference on Consumer Electronics

Project Outcomes

- Developed all modules of algorithm with desired functionality as specified
  - Neural Network and Epsilon Greedy Strategy are able to decide on an action
  - Simulation Manager is able to take the key that results from the action and run simulations with that key using random input stimulus to circuit
  - Oracle Comparison can generate a reward based on how many of the outputs from the circuits were correct vs the Oracle circuit

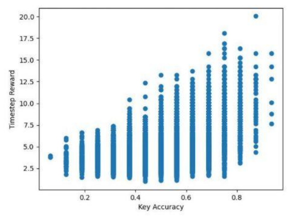
Improvements Made

- Initial bug with correct calculation of output % error
- Noticed a lack of relationship between the accuracy of key and output quality
- Shift in reward scheme because of non-linear effect of key accuracy on output accuracy
- Effect reliability of reward function
- Using exponential scheme slightly improved issue
- Reward and key accuracy relationship shows strength of logic locking



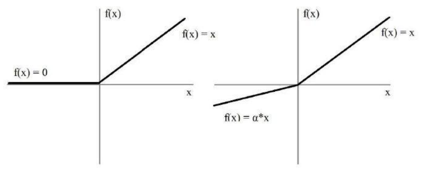
Top: Key Accuracy vs Reward Generated before new reward function

Bottom: Key Accuracy vs Reward Generated After new reward function



Improvements Made

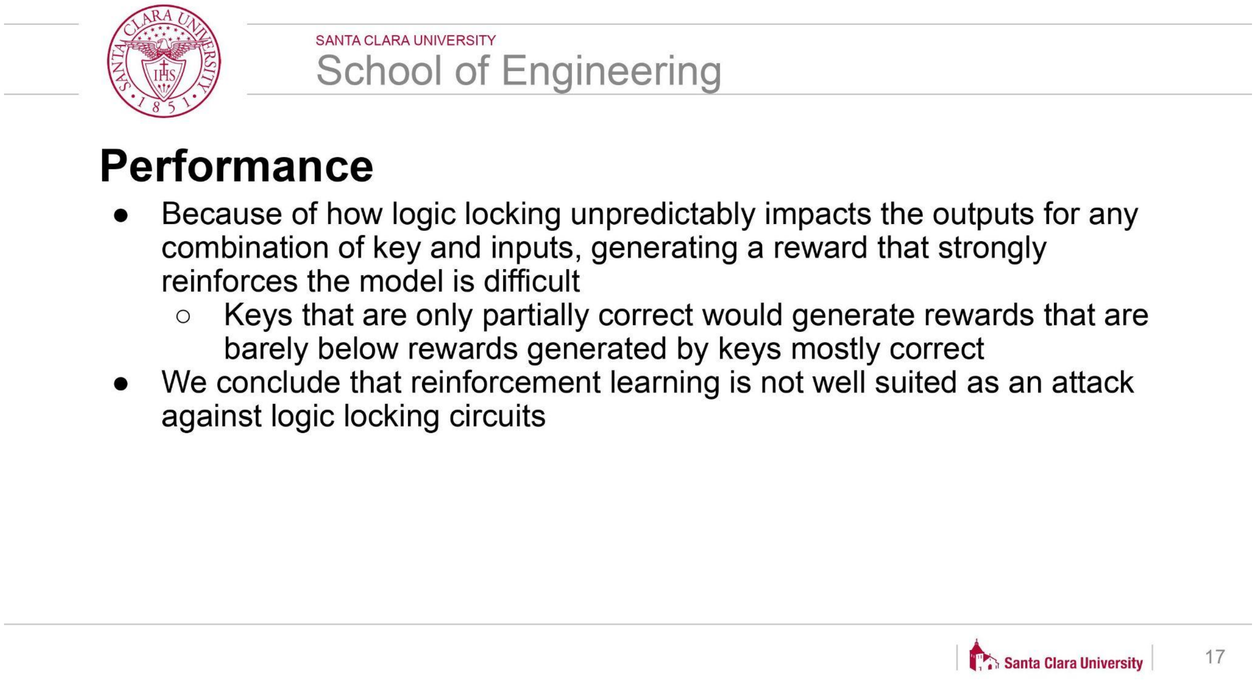
- Each of our fully connected layers in the neural network use Rectified Linear Units (ReLU)
  - Dying ReLU Problem
- Switched to using Leaky ReLU for each layer
  - Can help back propagation recover from dead neurons
- Added 4 layers to the network for a total of 7 layers and a final SoftMax layer for outputs



Left: ReLU Activation Function   Right: Leaky ReLU Activation Function

Performance

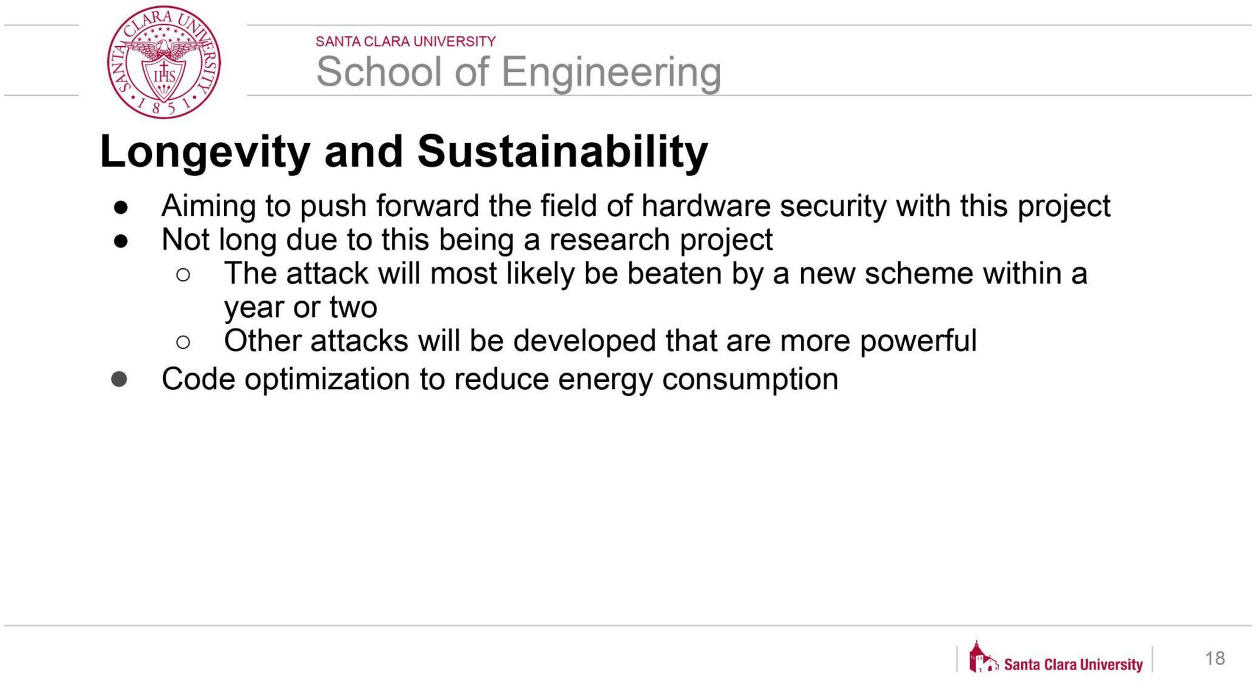
- The model fails to learn after 100,000 timesteps of training
- Cannot conclude accuracy for extracting a key from the circuit
- Training the model does take a considerable amount of time
  - The best logic locking attacks only take seconds to complete
  - May be able to reduce time with a faster Verilog Simulator
  - May also be able to reduce time with more compute power


SANTA CLARA UNIVERSITY
School of Engineering

Performance

- Because of how logic locking unpredictably impacts the outputs for any combination of key and inputs, generating a reward that strongly reinforces the model is difficult
  - Keys that are only partially correct would generate rewards that are barely below rewards generated by keys mostly correct
- We conclude that reinforcement learning is not well suited as an attack against logic locking circuits

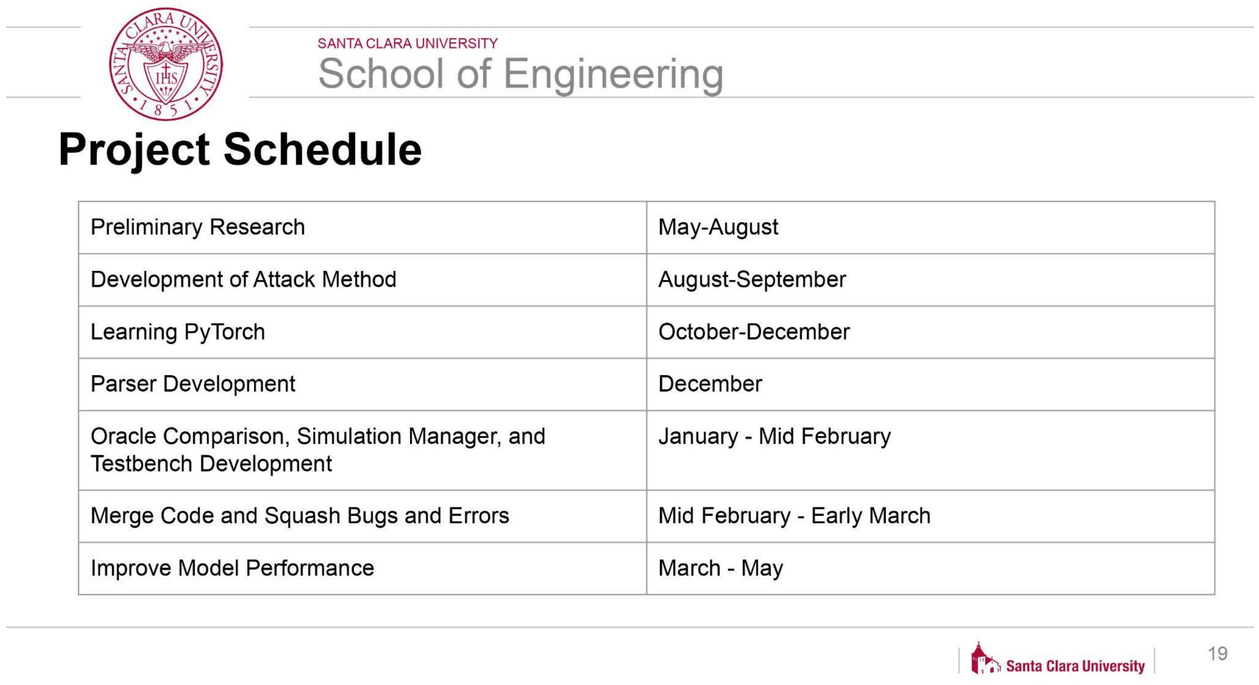
Santa Clara University 17


SANTA CLARA UNIVERSITY
School of Engineering

Longevity and Sustainability

- Aiming to push forward the field of hardware security with this project
- Not long due to this being a research project
  - The attack will most likely be beaten by a new scheme within a year or two
  - Other attacks will be developed that are more powerful
- Code optimization to reduce energy consumption

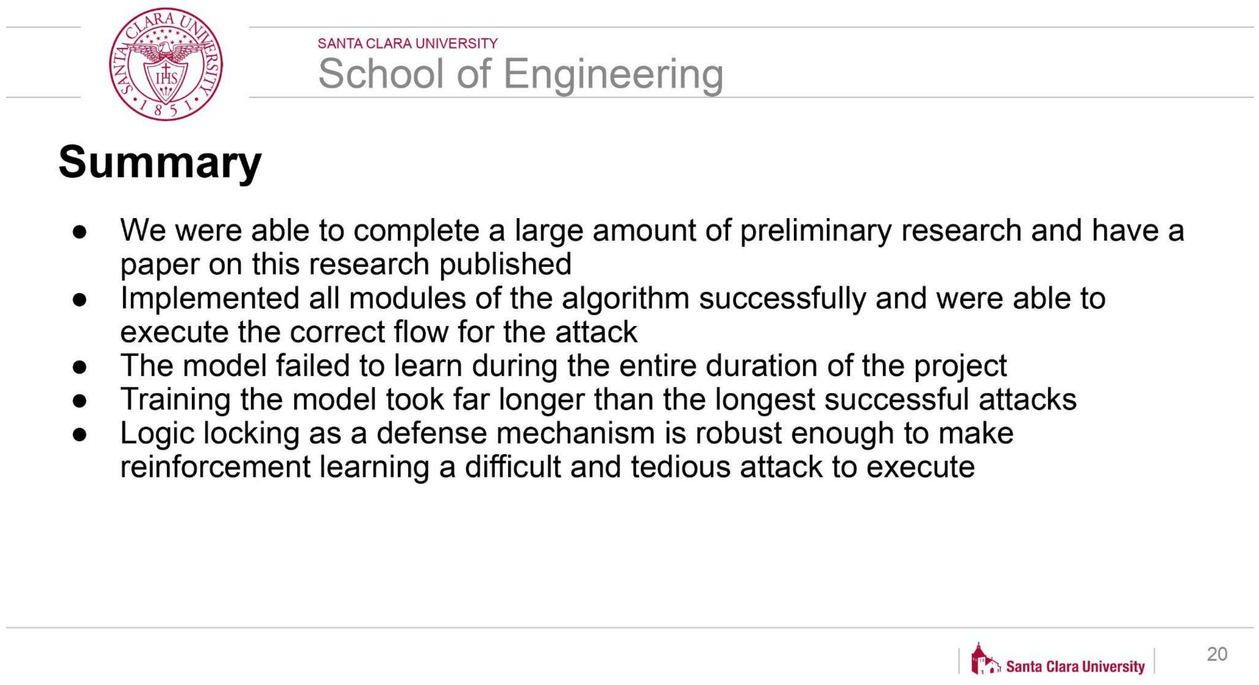
Santa Clara University 18


SANTA CLARA UNIVERSITY
School of Engineering

Project Schedule

Preliminary Research	May-August
Development of Attack Method	August-September
Learning PyTorch	October-December
Parser Development	December
Oracle Comparison, Simulation Manager, and Testbench Development	January - Mid February
Merge Code and Squash Bugs and Errors	Mid February - Early March
Improve Model Performance	March - May

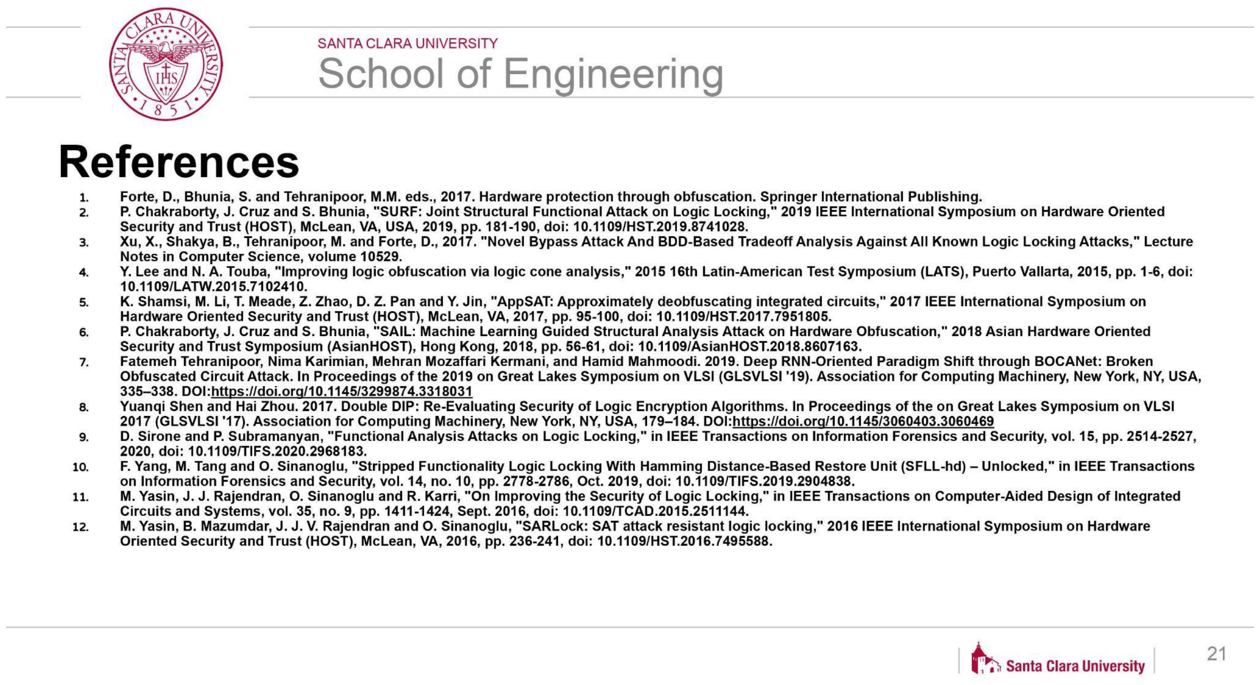
Santa Clara University 19


SANTA CLARA UNIVERSITY
School of Engineering

Summary

- We were able to complete a large amount of preliminary research and have a paper on this research published
- Implemented all modules of the algorithm successfully and were able to execute the correct flow for the attack
- The model failed to learn during the entire duration of the project
- Training the model took far longer than the longest successful attacks
- Logic locking as a defense mechanism is robust enough to make reinforcement learning a difficult and tedious attack to execute

Santa Clara University 20



SANTA CLARA UNIVERSITY

School of Engineering

References

1. Forte, D., Bhunia, S. and Tehranipoor, M.M. eds., 2017. Hardware protection through obfuscation. Springer International Publishing.
2. P. Chakraborty, J. Cruz and S. Bhunia, "SURF: Joint Structural Functional Attack on Logic Locking," 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 2019, pp. 181-190, doi: 10.1109/HST.2019.8741028.
3. Xu, X., Shakya, B., Tehranipoor, M. and Forte, D., 2017. "Novel Bypass Attack And BDD-Based Tradeoff Analysis Against All Known Logic Locking Attacks," Lecture Notes in Computer Science, volume 10529.
4. Y. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," 2015 16th Latin-American Test Symposium (LATS), Puerto Vallarta, 2015, pp. 1-6, doi: 10.1109/LATW.2015.7102410.
5. K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2017, pp. 95-100, doi: 10.1109/HST.2017.7951805.
6. P. Chakraborty, J. Cruz and S. Bhunia, "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Hong Kong, 2018, pp. 56-61, doi: 10.1109/AsianHOST.2018.8607163.
7. Fatemeh Tehranipoor, Nima Karimian, Mehran Mozaffari Kermani, and Hamid Mahmoodi. 2019. Deep RNN-Oriented Paradigm Shift through BOCANet: Broken Obfuscated Circuit Attack. In Proceedings of the 2019 on Great Lakes Symposium on VLSI (GLSVLSI '19). Association for Computing Machinery, New York, NY, USA, 335–338. DOI:https://doi.org/10.1145/3299874.3318031
8. Yuanqi Shen and Hai Zhou. 2017. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In Proceedings of the on Great Lakes Symposium on VLSI 2017 (GLSVLSI '17). Association for Computing Machinery, New York, NY, USA, 179–184. DOI:https://doi.org/10.1145/3060403.3060469
9. D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2514-2527, 2020, doi: 10.1109/TIFS.2020.2968183.
10. F. Yang, M. Tang and O. Sinanoglu, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) – Unlocked," in IEEE Transactions on Information Forensics and Security, vol. 14, no. 10, pp. 2778-2786, Oct. 2019, doi: 10.1109/TIFS.2019.2904838.
11. M. Yasin, J. J. Rajendran, O. Sinanoglu and R. Karri, "On Improving the Security of Logic Locking," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 9, pp. 1411-1424, Sept. 2016, doi: 10.1109/TCAD.2015.2511144.
12. M. Yasin, B. Mazumdar, J. J. V. Rajendran and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2016, pp. 236-241, doi: 10.1109/HST.2016.7495588.

Santa Clara University 21

Appendix B: Software Used

- PyCharm IDE
- Icarus Verilog
- PyTorch Library
- Subprocess Library
- CUDA Toolkit
- Synopsys VCS
- Python
- Verilog

Appendix C: Publications

J. Mellor, A. Shelton, M. Yue and F. Tehranipoor, "Attacks on Logic Locking Obfuscation Techniques," *2021 IEEE International Conference on Consumer Electronics (ICCE),* 2021, pp. 1-6, doi: 10.1109/ICCE50685.2021.9427730.
https://ieeexplore-ieee-org.libproxy.scu.edu/document/9427730