Protecting ML models from Power Side Channel Attacks via Masking

A M. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Harshita Mishra
(234101016)

under the guidance of

Dr. Chandan Karfa

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Protecting ML models from Power Side Channel Attacks via Masking**" is a bonafide work of **Harshita Mishra** (**Roll No. 234101016**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Chandan Karfa**

Associate Professor,

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

Nov, 2024

Guwahati.

Acknowledgements

I would like to express my heartfelt gratitude to my thesis supervisor, Prof. Chandan Karfa, from the Department of Computer Science and Engineering, IIT Guwahati, for his invaluable guidance, unwavering support, and constant encouragement throughout this project. His insightful advice and dedicated efforts have provided me with invaluable lessons that have greatly enhanced my understanding and will undoubtedly benefit me in my future endeavors.

I also sincerely thank my parents and friends for their unwavering support and cooperation. Their encouragement and wisdom have been a pillar of strength throughout my journey. Lastly, I would like to once again thank Prof. Chandan Karfa, whose guidance made this project a reality and a significant milestone in my academic journey.

Contents

List of Figures

Chapter 1

Introduction

Physical side-channel attacks pose a major threat to the security of modern-day crypto-systems. Attacks like Differential Power Analysis(DPA) exploit the correlation between the secret key-dependent data and power consumption of the device[Koc99]. Until recently, these attacks were limited to cryptographic schemes, and hence, the side channel analysis research focused primarily on the protection of cryptographic implementations. But lately, ML models are also been targeted for physical side-channel attacks.

Edge-based ML accelerators are susceptible to these attacks as the adversary has physical access to the device and hence, applying physical side-channel attacks is comparatively easier. The differences between ML and cryptographic algorithms cause challenges when adapting side-channel defenses towards ML. My work uses a well-known defense technique known as Masking. Weights and biases are the critical parameters of an ML model that I will be aiming to protect via masking. This countermeasure splits the original parameter into multiple statistically independent shares to break the correlation of secret data with power consumption. These random shares are derived using random values drawn using uniform random distribution.

1.1 Background

1.1.1 Physical Side Channel Attacks

The training phase of the ML model is done in a trusted environment. However, the trained model is deployed to an edge device running on an untrusted environment. The adversary therefore gets physical access to the device and it is after the trained model parameters e.g., weights and biases. There are mainly two types of physical side-channel attacks :

- Power consumption: The CMOS power trace is affected by the data being processed. Hence by differential power analysis on the power traces, the sensitive information can be extracted by the attacker.
- Electromagnetic emanations-based attacks: Uses varying electromagnetic radiations depending on the computations of the sensitive information.

My work mainly addresses protecting the models from power-based side-channel attacks.

1.1.2 Masking

This technique primarily works on the principle of making side-channel information like power traces / electromagnetic emanations independent of the underlying data being processed. There are two main kinds of masking, depending on the method of randomization. First is Boolean Masking where the sensitive variables are protected by sealing it with the mask using bitwise operations. The variable is XORed with a random value to create shares. The computations then take place for the newly generated share rather than the variable itself. To get the original value the share is again XORed with the same mask. The second technique is Arithmetic Masking where the sensitive value is masked by adding a random value under a specific modulus.

1.2 Motivation

The growing shift of ML models to edge devices like surveillance cameras to directly provide models as a service to the customer for better performance and privacy makes them vulnerable to power-side channel attacks. To date, the defense techniques discussed to protect neural networks have mainly focused on masking the inputs of the neural network or the hardware components like adder and multiplexer[DAP$^+$22]. There is no countermeasure for masking of weights due to their huge size, hence masking all the weights throughout the network is unfeasible. One of the approaches can be the masking of the selective weights depending on the impact they have on the accuracy of the neural network.

1.3 Objective

Performing sensitivity analysis of weights which helps in ranking them based on their influence on the neural network's prediction by measuring changes in output when specific weights are perturbed. Then apply the most suitable masking technique on the higher-ranked weights.

1.4 Contributions

Based on the project's scope and achievements, the following are its contributions:

- Application of masking on partial weights in the neural network to enhance security against physical side-channel attacks.
- Conducted detailed experiment to rank the importance of weights based on their impact on the accuracy of the ML model.
- Two architectures for selective masking of weights in the first and second layers of neural networks are designed.

1.5 Report Organization

In Chapter 2, a brief survey of the Literature related to this work is presented. The chapter discusses the underlying theory of masking and its types. The chapter concludes by justifying the selection of secret variables as weights instead of inputs of the network. Chapter 3 presents the results obtained from the experiments performed on the weights obfuscation and proposes architectures of the masking procedure. Finally, in Chapter 4, I have summarized the report and also mentioned the future work that needs to be done for the implementation of the idea.

Chapter 2

Literature Survey

2.1 Boolean Masking

Masking uses secure multi-party computation to split the secrets into random shares and to de-correlate the statistical relation of secret-dependent computations to side channels (e.g., the power draw). The key terms of boolean masking are:

- Sensitive data: The data to be protected.
- Random mask: A randomly generated value that is XORed with the sensitive data to create masked data.
- Masked data: The result of XORing the sensitive data with the mask.

The XOR gate is fundamental in boolean masking because of its reversible property. If S is the sensitive data and M is the random mask, then masked data is represented as :
$S' = \text{S} \oplus M$.

For all the operations including the sensitive variable S, the masked variable and mask is propagated independently. It has to be made sure that there are no operations including the the mask and masked data simultaneously to avoid recovery of original data. To recover the original data S after operations, masked data can be XORed again with the mask as shown:
$\text{S} = S' \oplus M$.

XORing the masked data again with the same mask retrieves the original data, but without revealing any direct correlation between intermediate values and the sensitive data.

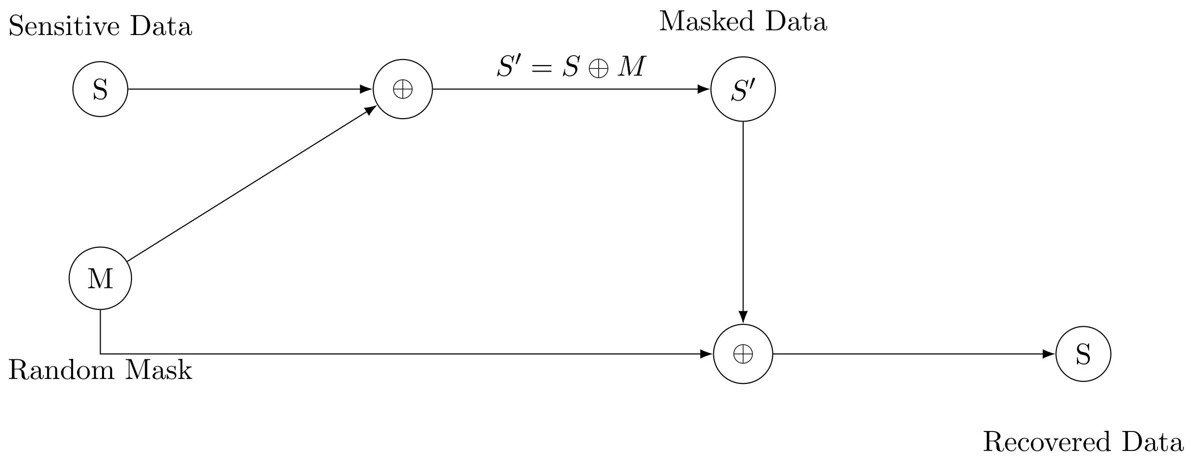


Fig. 2.1.1: Boolean Masking

2.2 Modular Arithmetic Masking

Neural network computations mainly revolve around arithmetic operations like addition and multiplication. Hence arithmetic masking is more feasible in such cases. Modular arithmetic masking works on modular addition of the sensitive variable with a random mask. Let S be the sensitive data, M a random mask and N being the modulus which is chosen based on the values of S. The masked data can be computed by: $S^{'}$ = S + M mod N. Similarly, by subtracting the mask from the masked data, original data can be recovered.

There are some challenges to this technique. Firstly, the modular operation shifts the negative activations to the positive side. Secondly, modulo folding causes a negative output score of the final layer to wrap around which results in incorrect predictions.[DAP+22].

2.3 VivadoHLS

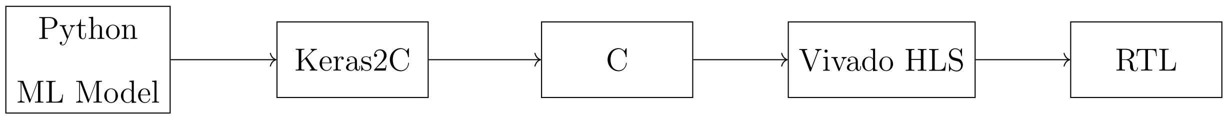
A highly popular High-level Synthesis(HLS) tool for generating low-level code such as Register Transfer Level(RTL) for implementing high-level models described in C and C++. This low-level code is then implemented on Field Programmable Gate Arrays(FPGA)[GCM+16]. The tool also provides a report giving insights about latency, resource utilization, and timing analysis. The report thus helps in comparing the baseline model with the masked model for performance evaluation.

2.4 Keras2C

Keras neural network models are stored in a .h5 file i.e. Python language. For generating the RTL(Register Transfer Level) code of the neural network we need the corresponding C code of the model. Keras2c is a library for converting Keras neural networks to C functions. Each Keras layer is performed as a pure C function. When we provide the Keras model to Keras2C it generates three files:

- .c file - source for neural network functions.
- .h file - header file for declarations.
- test-suite.c - automated testing file for the model.

After synthesizing the C file according to Vivado, these files can be run on Vivado to generate RTL and the respective report.



2.5 Masked Components of Neural Network [DAP+22]

Prior literature has implemented suitable masking based on the type of operation to be masked. They have mainly proposed to mask the inputs instead of weights due to area overhead. The following section discusses their technique.

- Masking of Weighted Summations:

  In an unmasked model, the input pixel(p_i) gets multiplied with the respective weight($w_{i,j}$) for all the input pixels and added throughout to calculate the weighted summation. To mask the input, arithmetic masking is used where a pixel is broken down to $(p_i - r_i)$ and (r_i) where r_i is a random mask. Weighted summation is computed on both the shares independently. Bias is added to one of the shares and sent to the activation function which is also masked.

- Masked Activation Function:

  An activation function has been defined as : BIN(z) = -sgn(z - K/2), where K is the modules. The masked design activation function deals with computing if the sum of two arithmetic shares received is greater than or less than K/2, which depends on the MSB of the sum. Hence, in the above formula z represents the sum of the arithmetic shares received from the weighted summation.

- Masked Output Layer:

  The output function receives the confidence scores from the activation function and by comparing gives the final inference result. Here boolean masking was used as there are bitwise manipulations involved.

2.6 Conclusion of Literature Survey

The Literature Survey helped me mainly conclude that masking of weights has never been explored yet due to area overheads. However, there is an alternate way discussed in further sections to reduce overhead as well as mask the critical parameters of the neural network. Also, Vivado HLS will be employed to compare the area/latency overhead of the baseline NN and the masked NN.

Chapter 3

Experimental Analysis and Proposed Architecture

3.1 Identification of partial weights to be masked

For large models, masking all the weights can lead to extremely huge overhead in terms of the area due to multiple reasons:

- Storage of random mask for each of the weights, which approximately doubles the storage memory
- Additional logic gates are needed for each weight operation for generating masked weight and again recovering the original weight.

Due to this overhead, generally masking has been performed on the inputs to relatively reduce the overhead.

Novelty in my work would be masking the partial weights. The weights are selected based on their impact on the accuracy of the model. The layer whose obfuscation of weights drops the accuracy the most is the layer whose weights are to be masked. I have worked on the MNIST which stands for Modified National Institute of Standards and Technology and is a database of handwritten digits used to train and test image processing and machine learning systems.

A model 784-256-32-10 denotes input layer has 784 neurons, 2 hidden layers with 256 and 32 neurons each, and the output layer with 10 neurons. There are 3 such models I have experimented on. The methods of obfuscating employed to compare are :

- Randomizing the weights of a single layer at a time.
- Replacing weights with new weights with a hamming distance of 2,4,6 or 8 with respect to original weights. As the weights of the neural network are float values, hamming distance can be calculated with respect to exponent as well as mantissa.

Table 3.1: Results of Model 784-256-64-10

Model	**Org. Accuracy**	**Randomized Layer**	**Accuracy**	**HD (Exponent)**			**HD (Mantissa)**		
				2	**4**	**8**	**2**	**4**	**8**
784-256-64-10	99.01	1	10.41	9.81	9.99	9.99	98.87	98.93	98.94
		2	10.28	9.90	10.07	9.83	98.93	98.98	98.94
		3	98.85	9.87	9.70	9.75	98.97	98.98	98.95

Table 3.2: Results of Model 784-512-128-10

Model	**Org. Accuracy**	**Randomized Layer**	**Accuracy**	**HD (Exponent)**			**HD (Mantissa)**		
				2	**4**	**8**	**2**	**4**	**8**
784-512-128-10	99.04	1	10.10	9.92	9.80	9.95	98.99	98.96	98.95
		2	10.27	9.98	9.01	9.99	98.98	98.96	98.95
		3	99.02	9.70	9.77	9.76	98.97	99.02	99.07

Table 3.3: Results of Model 784-256-32-10

Model	**Org. Accuracy**	**Randomized Layer**	**Accuracy**	**HD(Exponent)**			**HD (Mantissa)**		
				2	**4**	**8**	**2**	**4**	**8**
784-256-32-10	98.97	1	10.24	9.81	9.88	9.94	98.93	98.95	98.93
		2	10.27	9.90	10.07	9.83	98.94	98.95	98.94
		3	98.94	9.87	9.77	9.76	98.94	98.91	98.92

Accuracy trends with randomized weights and hamming distance weights depict that for each model configuration, randomizing weights of earlier layers(Layer 1) leads to a significant drop in accuracy and as you move towards randomizing weights in deeper layers i.e. layer 3. Earlier layers are critical for extracting basic features and heavily influence model performance. Replacing weights with new weights of hamming distance with respect to mantissa has no impact on the accuracy. However, exponent perturbations of weights again influence the accuracy on a large scale, be it any layer.

This experiment highlights the critical role of the weights in the earlier layers of the model, as obfuscating them leads to noticeable alteration in accuracy. This underscores the importance of securely masking these weights to preserve the model's performance and integrity.

3.2 Proposed Architecture

For the first layer of the model, weights are the only secret variable. As they are floating point numbers, arithmetic masking is used to divide them into two shares. A random mask is chosen and an arithmetic operation is performed between the weight and the mask. The result of the arithmetic operation is one of the shares and the mask itself is another share of the weight. Instead of weight, these two shares will be propagated inside the layer to perform all the operations. All the computations need to be performed independently as both shares together can leak the value of the secret variable i.e. weight.

The diagram below depicts the working inside the first layer of the proposed solution. In the below architecture, we assume that the weights are already masked and stored in tamper-proof memory along with the input and biases. A tamper-proof memory is a type of memory designed to prevent unnecessary access, modification, and destruction.

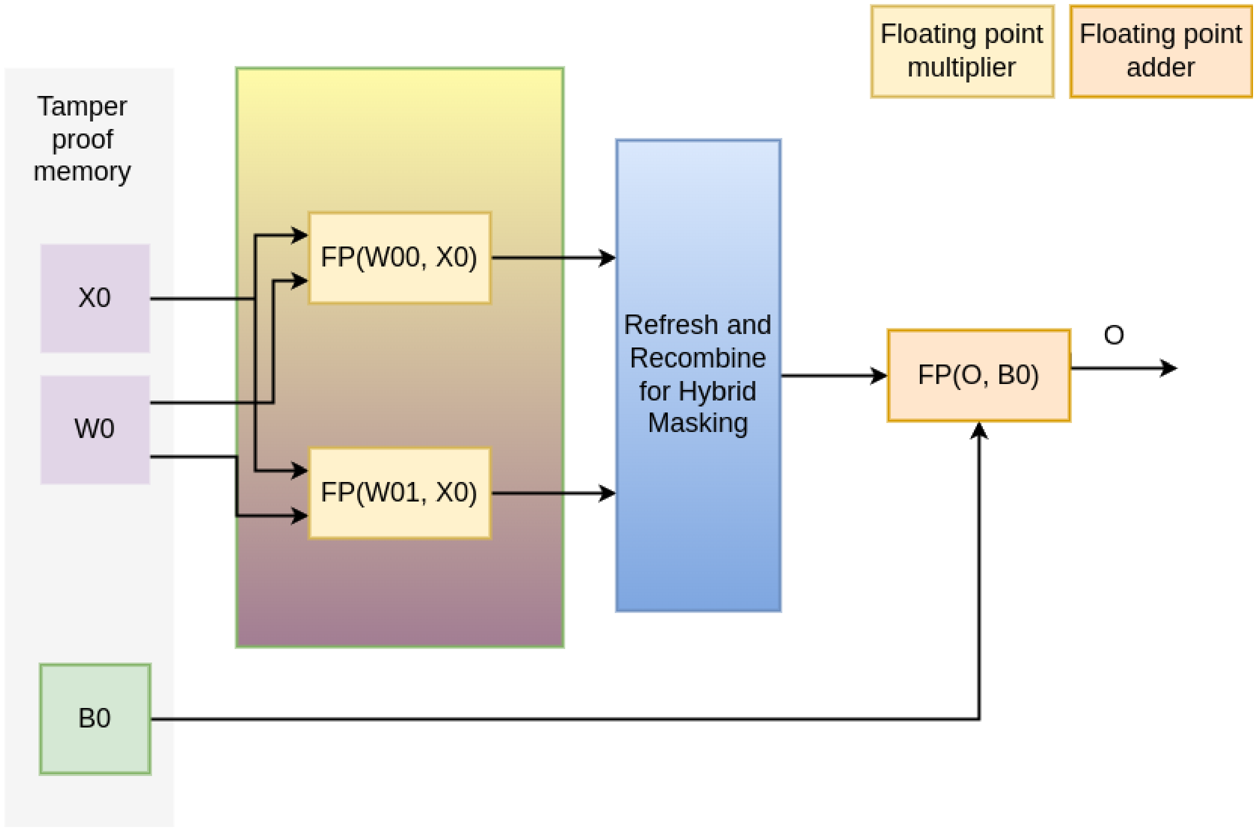


Fig. 3.2.1: Architecture for masking weights in 1st layer

The two shares are multiplied with the input independently in the floating point multiplier. In floating point multiplier design, the first step is that exponents of the input and share are made equal to the exponent of the input. The exponents will be added and adjusted in the final product at the end. Then the mantissa of input and the share is multiplied by treating the input to be constant as it is not secret in the first layer. The multiplication is done by the shifting method used in the case of multiplication by a constant. After multiplication, a recombine module is used to recover the product of the original weight and original input from the two products of input and shares using the recover method of arithmetic shares. Then the bias is added to the final product using a floating point adder.

The computed result is propagated to layer 2 of the model for further predictions. Here, the input is the output of the first-layer computations which consists of weights as a part of it, hence in the second layer the input also becomes the secret variable along with second-layer weights. Masking is therefore applied to inputs as well as the weights of layer 2. The architecture can be shown as follows:

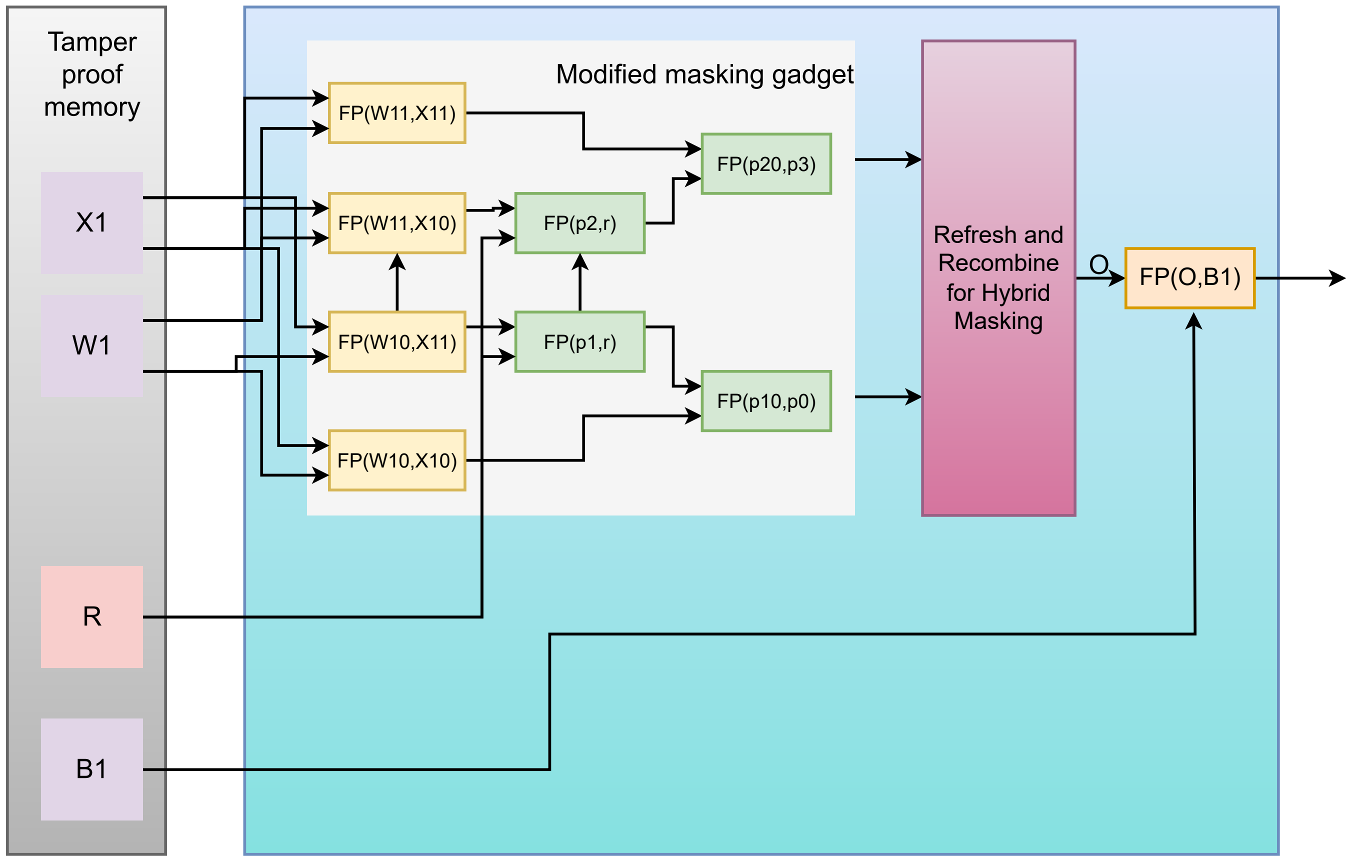


Fig. 3.2.2: Architecture for masking weights in 2nd layer

As there are two secret variables to be masked, there will be four shares created after masking, two for each variable. These shares along with bias are again assumed to be obtained from a tamper-proof memory. The four partial products of four shares are calculated using a floating point multiplier. The cross-domain partial products are again masked using the random mask to prevent the leakage of the secrets, as partial products are more vulnerable. The product of shares after respective multiplication is given to recombination model to recover the original. The output of the recovery model is the product of the original input and weight unmasked which is then added with the bias in a floating point adder.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

The current work I have done so far is to understand how the power side channel attacks hamper the security of edge based ML devices. The thorough survey of defense techniques employed till date to protect the critical parameters of the neural network drew my attention towards unexplored technique of masking the weights of the model instead of inputs. Most importantly masking the partial weights, as masking all weights leads to huge overhead. In order to decide which weights to be masked, I ranked them using several techniques. Then, after discovering various masking techniques and picking the most suitable one for my idea, I proposed two architectures of the masking procedure based on the experimental analysis and specific layers of the model.

4.2 Future Work

In the next phase, I plan to do the following:

- Implementing the proposed architecture on a machine learning model.
- Analyzing the overhead due to masking and minimizing it as much as possible.

References

[DAP$^+$22] Anuj Dubey, Afzal Ahmad, Muhammad Adeel Pasha, Rosario Cammarota, and Aydin Aysu. Modulonet: Neural networks meet modular arithmetic for efficient hardware masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 506–556, 2022.

[GCM$^+$16] Konstantinos Georgopoulos, Grigorios Chrysos, Pavlos Malakonakis, Antonis Nikitakis, Nikos Tampouratzis, Apostolos Dollas, Dionisios Pnevmatikatos, and Yannis Papaefstathiou. An evaluation of vivado hls for efficient system design. In *2016 International Symposium ELMAR*, pages 195–199, 2016.

[Koc99] P Kocher. Differential power analysis. In *Proc. Advances in Cryptology (CRYPTO'99)*, 1999.