Reverse Engineering of Neural Network Architectures Through Side-channel Information

Virtual OpenS3 Workshop, November 4, 2021

Outline

Outline

Machine Learning and Security

- Machine learning has become mainstream across industries.
- It is also widely used in security applications.
- Having strong ML models is an asset, on which many companies invest a significant amount of time and money to develop.
- How secure are such ML models against reverse engineering attacks?

Machine Learning and Security

- People investigate the leakage of sensitive information from machine learning models about individual data records.
- ML model provided by malicious attacker can give information about the training set.
- Reverse engineering of CNNs via timing and memory leakage.
- Exploits of the line buffer in a convolution layer of a CNN.

Neural Networks

- We consider neural networks: multilayer perceptron and convolutional neural networks.
- They are commonly used machine learning algorithms in modern applications.
- They consist of different types of layers that are also occurring in other architectures like recurrent neural networks.
- In the case of MLP, the layers are all identical, which makes it more difficult for SCA and could be consequently considered as the worst-case scenario.

Outline

Threat Model

- Recover the neural network architecture using only side-channel information.
- No assumption on the type of inputs or its source, as we work with real numbers.
- We assume that the implementation of the machine learning algorithm does not include any side-channel countermeasures.

Attacker's Capability

- The attacker in consideration is a passive one.
- Acquiring measurements of the device while operating "normally" and not interfering with its internal operations by evoking faulty computations.
- Attacker does not know the architecture of the used network but can feed random (and hence known) inputs to the architecture.
- Attacker is capable of measuring side-channel information leaked from the implementation of the targeted architecture.
- Targets are Atmel ATmega328P and ARM Cortex-M3.

Implementation Attacks and Side-channel Analysis

Implementation attacks

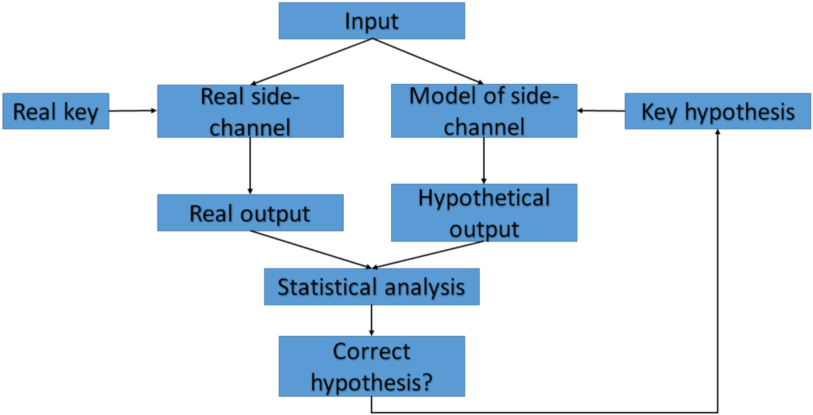
Implementation attacks do not aim at the weaknesses of the algorithm, but on its implementation.

- **Side-channel analysis** (SCAs) – passive, non-invasive attacks.
- SCA – one of the most powerful category of attacks on crypto devices.

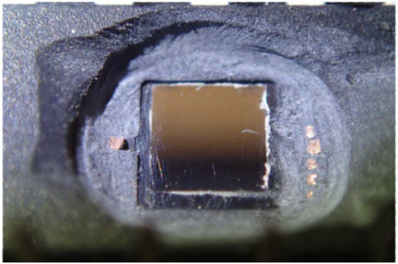
Side-channel Analysis

- Differential Power Analysis (DPA) (DEMA) is an advanced form of SCA, which applies statistical techniques to recover secret information from physical signatures.
- The attack normally tests for dependencies between actual physical signature (or measurements) and hypothetical physical signature, i.e., predictions on intermediate data. The hypothetical signature is based on a leakage model and key hypothesis.

Differential Power Analysis



Setup

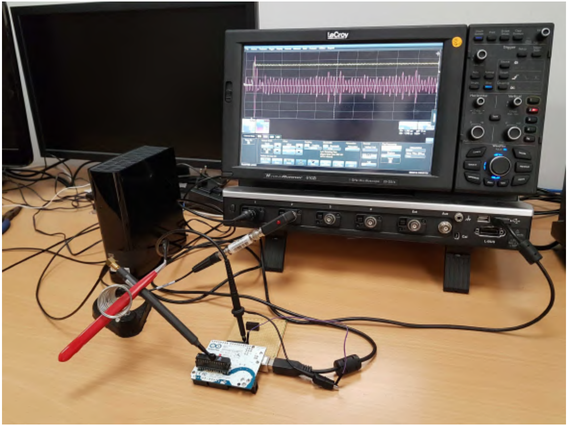


(a) Target 8-bit microcontroller mechanically decapsulated

(b) Langer RF-U 5-2 Near Field Electromagnetic passive Probe

Figure: Experimental Setup 1

Setup



(a) The complete measurement setup

Setup

- The exploited leakage model of the target device is the Hamming weight (HW) model.
- A microcontroller loads sensitive data to a data bus to perform indicated instructions.
- The training phase is conducted offline, and the trained network is then implemented in C language and compiled on the microcontroller.

$$HW(x) = \sum_{i=1}^{n} x_i \ , \tag{1}$$

What Do We Need

- Information about layers.
- Information about neurons.
- Information about activation functions.
- Information about weights.

Activation Functions

- An activation function of a node is a function f defining the output of a node given an input or set of inputs.

$$y = Activation(\sum(weight \cdot input) + bias). \qquad (2)$$

- Sigmoid, tanh, softmax, ReLU.

Activation Functions

$$f(x) = \frac{1}{1+e^{-x}}. \tag{3}$$

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1. \tag{4}$$

$$f(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}},\ for\ j = 1, \ldots, K. \tag{5}$$

$$f(x) = max(0, x). \tag{6}$$

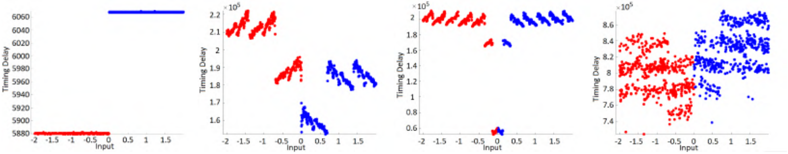
Reverse Engineering the Activation Functions

- The timing behavior can be observed directly on the EM trace.
- We collect EM traces and measure the timing of the activation function computation from the measurements.

Table: Minimum, Maximum, and Mean computation time (in *ns*) for different activation functions

Activation Function	Minimum	Maximum	Mean
ReLU	5 879	6 069	5 975
Sigmoid	152 155	222 102	189 144
Tanh	51 909	210 663	184 864
Softmax	724 366	877 194	813 712

Reverse Engineering the Activation Functions



(b) ReLU (c) Sigmoid (d) Tanh (e) Softmax

Figure: Timing behavior for different activation functions

Reverse Engineering the Multiplication Operation

- For the recovery of the weights, we use the Correlation Power Analysis (CPA) i.e., a variant of DPA using the Pearson's correlation as a statistical test.
- CPA targets the multiplication $m = x \cdot w$ of a known input x with a secret weight w.
- Using the HW model, the adversary correlates the activity of the predicted output m for all hypothesis of the weight.
- Thus, the attack computes $\rho(t, w)$, for all hypothesis of the weight w, where ρ is the Pearson correlation coefficient and t is the side-channel measurement.
- The correct value of the weight w will result in a higher correlation standing out from all other wrong hypotheses w^*, given enough measurements.

Reverse Engineering the Multiplication Operation

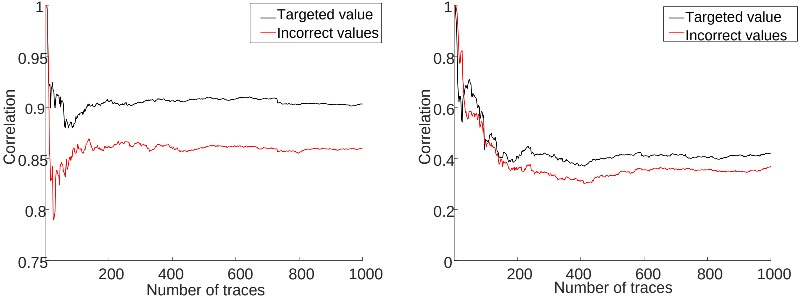
- We start by analyzing the way the compiler is handling floating-point operations for our target.
- The generated assembly confirms the usage of IEEE 754 compatible representation.
- Since the target device is an 8-bit microcontroller, the representation follows a 32-bit pattern ($b_{31} \ldots b_0$), being stored in 4 registers.
- The 32-bit consist of: 1 sign bit (b_{31}), 8 biased exponent bits ($b_{30} \ldots b_{23}$) and 23 mantissa (fractional) bits ($b_{22} \ldots b_0$).

$$(-1)^{b_{31}} \times 2^{(b_{30} \ldots b_{23})_2 - 127} \times (1.b_{22} \ldots b_0)_2.$$

Reverse Engineering the Multiplication Operation

- We target the result of the multiplication m of known input values x and unknown weight w.
- For every input, we assume different possibilities for weight values.
- We then perform the multiplication and estimate the IEEE 754 binary representation of the output.
- Then, we perform the recovery of the 23-bit mantissa of the weight.
- The sign and exponent could be recovered separately.

Reverse Engineering the Multiplication Operation



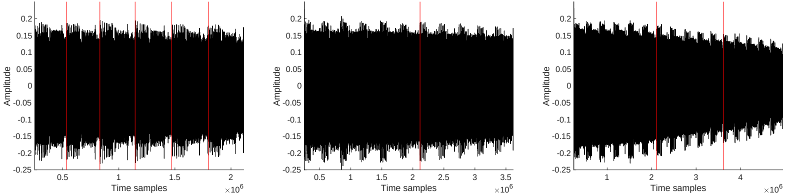
(a) First byte recovery (sign and 7-bit exponent)

(b) Second byte recovery (lsb exponent and mantissa)

Figure: Recovery of the weight

Reverse Engineering the Number of Neurons and Layers

- To perform the reverse engineering of the network structure, we first use SPA (SEMA).



(a) One hidden layer with 6 neurons

(b) 2 hidden layers (6 and 5 neurons each)

(c) 3 hidden layers (6,5,5 neurons each)

Figure: SEMA on hidden layers

Reverse Engineering the Number of Neurons and Layers

- To determine if the targeted neuron is in the same layer as previously attacked neurons, or in the next layer, we perform a weight recovery using two sets of data.
- Let us assume that we are targeting the first hidden layer (the same approach can be done on different layers as well).
- Assume that the input is a vector of length N_0, so the input x can be represented $x = \{x_1, ..., x_{N_0}\}$.
- For the targeted neuron y_n in the hidden layer, perform the weight recovery on 2 different hypotheses.

Reverse Engineering the Number of Neurons and Layers

- For the first hypothesis, assume that the y_n is in the first hidden layer. Perform weight recovery individually using x_i, for $1 \leq i \leq N_0$.
- For the second hypothesis, assume that y_n is in the next hidden layer (the second hidden layer).
- Perform weight recovery individually using y_i, for $1 \leq i \leq (n - i)$.
- For each hypothesis, record the maximum (absolute) correlation value, and compare both.
- Since the correlation depends on both inputs to the multiplication operation, the incorrect hypothesis will result in a lower correlation value.

Recovery of the Full Network Layout

- The combination of previously developed individual techniques can thereafter result in full reverse engineering of the network.
- The full network recovery is performed layer by layer, and for each layer, the weights for each neuron have to be recovered one at a time.
- The first step is to recover the weight w_{L_0} of each connection from the input layer (L_0) and the first hidden layer (L_1).
- In order to determine the output of the sum of the multiplications, the number of neurons in the layer must be known.
- Using the same set of traces, timing patterns for different inputs to the activation function can be built.
- The same steps are repeated in the subsequent layers $L_1, ..., L_{N-1}$.

Reverse Engineering the Number of Neurons and Layers

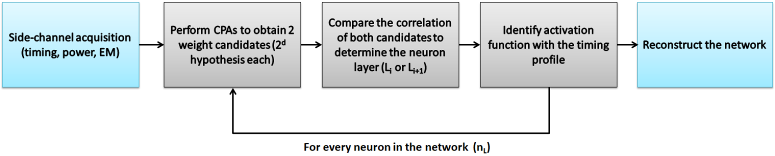
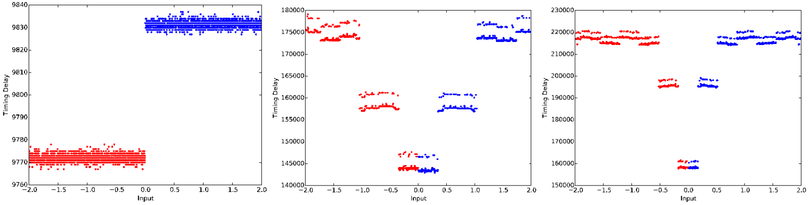


Figure: Methodology to reverse engineer the target neural network

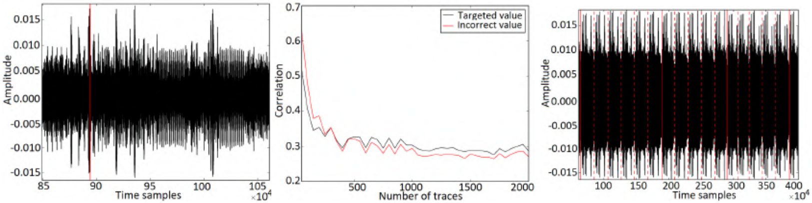
ARM Cortex M-3 and MLP



(a) ReLU (b) Sigmoid (c) Tanh

Figure: Timing behavior for different activation functions

ARM Cortex M-3 and MLP



(a) Observing pattern and timing of multiplication and activation function

(b) Correlation comparison between correct and incorrect mantissa for weight=2.453

(c) SEMA on hidden layers with 3 hidden layers (6,5,5 neurons each)

Figure: Analysis of an (6,5,5,) neural network

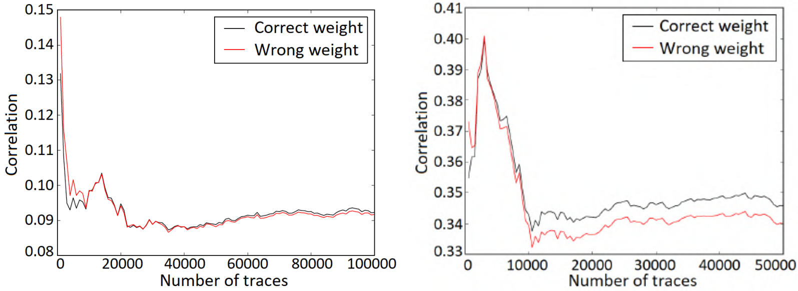
ARM Cortex M-3 and MLP

- Tests with MNIST and DPAv4 datasets.
- DPAv4: the original accuracy equals 60.9% and the accuracy of the reverse engineered network is 60.87%.
- MNIST: the accuracy of the original network is equal to 98.16% and the accuracy of the reverse engineered network equals 98.15%, with an average weight error converging to 0.0025.

ARM Cortex M-3 and CNN

- We target CIFAR-10 dataset.
- We choose as target the multiplication operation from the input with the weight, similar as in previous experiments.
- Differing from previous experiments, the operations on real values are here performed using fixed-point arithmetic.
- The numbers are stored using 8-bit data type – `int8` (q7).
- The resulting multiplication is stored in temporary `int` variable.
- The original accuracy of the CNN equals 78.47% and the accuracy of the recovered CNN is 78.11%.

ARM Cortex M-3 and CNN



(a) int scenario (b) int8 scenario

Figure: The correlation of correct and wrong weight hypotheses on different number of traces targeting the result of multiplication operation stored as different variable type: (a) `int`, (b) `int8`

Outline

Threat Model

- The underlying neural network architecture of the used network is public and all the weights are known.
- Attacker is capable of measuring side-channel information leaked from the implementation of the targeted architecture.
- The crucial information for this work are the weights of the first layer.
- Indeed, when MLP reads the input, it propagates it to all the nodes, performing basic arithmetic operations.
- This arithmetic operation with different weights and common unknown input leads to input recovery attack via side-channel.

Experimental Setup

- The training phase is conducted offline, and the trained network is then implemented in C language and compiled on the microcontroller.
- In our experiments, we consider MLP architectures consisting of a different number of layers and nodes in those layers.
- Note, we are only interested in the input layer where a higher number of neurons is beneficial for the attacker.

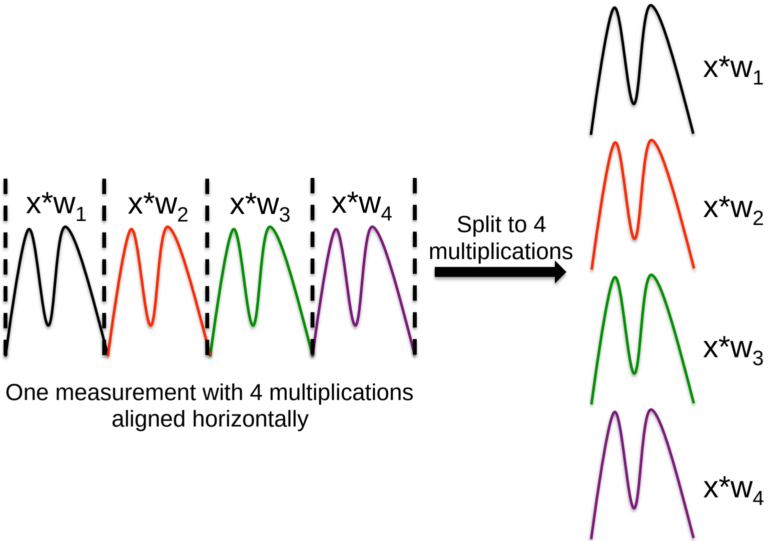
Results

- It can be extremely complex to recover the input by observing outputs from a known network.
- The proposed attack targets the multiplication operation in the first hidden layer.
- The main target for CPA is the multiplication $m = x \cdot w$ of a known weight w with a secret input x.
- As x changes from one measurement (input) to another, information learned from one measurement cannot be used with another measurement, preventing any statistical analysis over a set of different inputs.

Results

- To perform information exploitation over a single measurement, we perform a horizontal attack.
- The weights in the first hidden layer are all multiplied with the same input x, one after the other.
- M multiplications, corresponding to M different weights (or neurons) in the first hidden layer are isolated.
- A single trace is cut into M smaller traces, each one corresponding to one multiplication with a known associated weight.
- Next, the value of the input is statistically inferred by applying a standard CPA as explained before on the M smaller traces.

HPA


x*w1
x*w2
x*w3
x*w4
One measurement with 4 multiplications aligned horizontally
Split to 4 multiplications
x*w1
x*w2
x*w3
x*w4

Results on ATMega

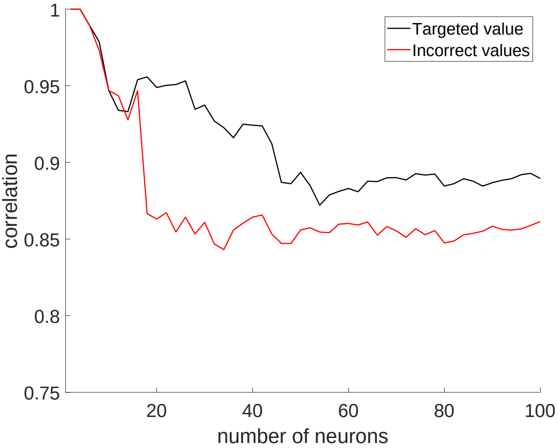


Figure: The first byte recovery (sign and 7-bit exponent).

Results

- The attack needs around 20 or more multiplications to reliably recover the input.
- In general, 70 multiplications are enough to recover all the bytes of the input, up to the desired precision of 2 decimal digits.
- This means that in the current setting, the proposed attack works very well on medium to large-sized networks, with at least 70 neurons in the first hidden layer, which is no issue in modern architectures used today.

Results on ARM Cortex M3

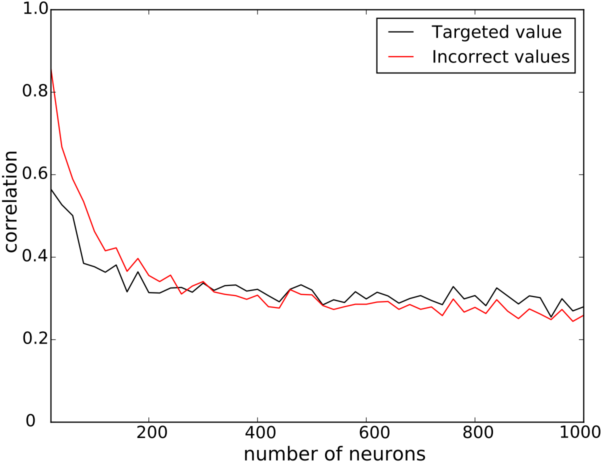


Figure: Correlation comparison between correct and incorrect inputs for target value 2.453.

Attack on MNIST Database

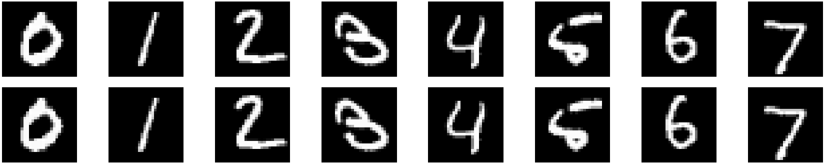


Figure: Original images (top) and recovered images with precision error (bottom).

Questions?

Thanks for your attention!

stjepan@computer.org Q?