Breaking AES using DPA and CNN Techniques

A Project Report

submitted by

Devendra Vamsi Korikana
(EE13B022)

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Breaking AES using DPA and CNN Techniques**, submitted by **Devendra Vamsi Korikana**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof.Chester Rebeiro
Research Guide
Professor
Dept. of Computer Science
IIT-Madras, 600036

Prof.Andrew Thangaraj
Research Co-Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600036

Place: Chennai
Date:

ACKNOWLEDGEMENTS

This work would not have been possible without the guidance and the help of several people. I take this opportunity to extend my sincere gratitude to all those who made this thesis possible.

First, I would like to thank all my teachers who bestowed me with good academic knowledge. I am indebted to my advisor Prof. Chester Rebeiro whose expertise, generous guidance and support made it possible for me to work on a topic that was of great interest to me.

I would like to thank my family for giving support and guidance all through my life. I would also like to thank all my friends and well-wishers for helping me in difficult times and being a good source of support and guidance.

Contents

Abstract

Template attack is the most common and powerful profiled side channel attack. It relies on a realistic assumption regarding the noise of the device under attack: the probability density function of the data is a multivariate Gaussian distribution. To relax this assumption, a recent line of research has investigated new profiling approaches mainly by applying machine learning techniques. The obtained results are commensurate, and in some particular cases better, compared to template attack. In this work, we propose to continue this recent line of research by applying more sophisticated profiling techniques based on deep learning.

1 Introduction

1.1 AES

The Advanced Encryption Standard, or AES, is a symmetric block cipher established by the U.S. National Institute of Standards and Technology (NIST) in 2001. It is based on a design principle known as a substitution-permutation network, a combination of both substitution and permutation, and is fast in both software and hardware. Following are the three major variants of AES in use

	Key length	No.of Rounds
AES-128	16 bytes	10
AES-192	24 bytes	12
AES-256	32 bytes	14

Table 1: Three Major Variants of AES in use

AES takes 16 bytes of plain text at once and arranges them in a 4×4 matrix of bytes, termed the state. All the operations are performed in the field GF(2^8). The field's irreducible polynomial is $x^8 + x^4 + x^3 + x + 1$. Each round consists of the following steps as shown in Figure 1.

- **Byte Substitution** - In this step each byte $a_{i,j}$ in the state matrix is replaced with a byte $S(a_{i,j})$ using an 8-bit substitution box, the Rijndael S-box[1]. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF(2^8), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points, i.e., $S(a_{i,j}) \neq a_{i,j}$and also any opposite fixed points, i.e., $S(a_{i,j}) \oplus a_{i,j} \neq FF_{16}$.

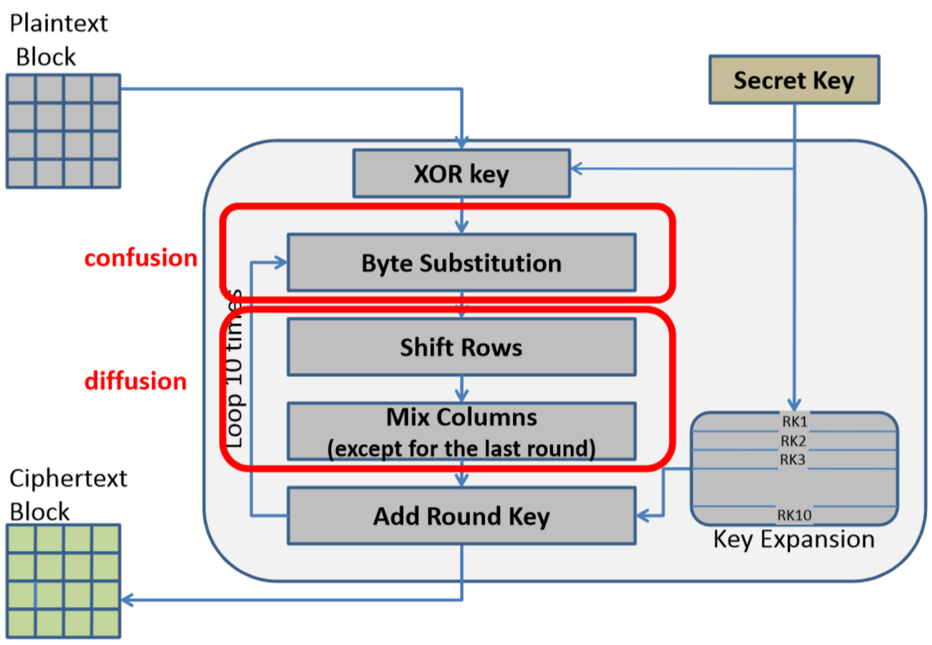

Figure 1: AES-128 Encryption

- **Shift Rows** - The Shift Rows step operates on the rows of the state, it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. Row n is shifted left circular by $n-1$ bytes. In this way, each column of the output state of the Shift Rows step is composed of bytes from each column of the input state. For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively. The importance of this step is to avoid the columns being linearly independent, in which case, AES degenerates into four independent block ciphers.

- **Mix columns** - In the Mix Columns step, the four bytes of each column of the state are combined using an invertible linear transformation. The Mix Columns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with Shift Rows, MixColumns provides diffusion in the cipher. During this operation, each column is transformed using a fixed MDS matrix. Matrix multiplication is composed of multiplication and addition of the entries. The MixColumns step can also be viewed as a multiplication by the below shown particular MDS matrix in the finite field GF(2^8).

$$\begin{array}{cccc} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{array}$$

This process is described further in the article Rijndael mix columns[2].

- **Add RoundKey** - In the Add RoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main key using Rijndael's key schedule[3], each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining the Byte Substitution and Shift Rows steps with the Mix Columns step by transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables, and utilizes a total of four kilobytes(4096 bytes) of memory - one kilobyte for each table. A round can then be done with 16 table lookups and 12 32-bit xor operations, followed by four 32-bit xor operations in the Add RoundKey step. If the resulting four-kilobyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit (i.e. 1 kilobyte) table by the use of circular rotates. Using a byte-oriented approach, it is possible to combine the Byte Substitution, Shift Rows, and Mix Columns steps into a single round operation[4].

1.2 Side Channel Attacks

Side Channel attacks(SCA) are nowadays well known and most designers of secure embedded systems are aware of them. They exploit information leaking from the physical implementations of cryptographic algorithms. Since, this leakage(e.g. the power consumption or the electromagnetic emanations) depends on the internally used secret key, the adversary may perform an efficient key-recovery attack to reveal these sensitive data. SCA attacks have been proven to be several orders of magnitude more effective than the conventional mathematical analysis based attacks and are much more practical to mount. Amongst side channel attacks, two classes may be distinguished.

- The so-called profiling SCA are the most powerful kind of SCA and consist of two steps. First, the adversary procures a copy of the target device and uses it to characterize the dependency between the manipulated data and the device behavior. Secondly, he performs a key-recovery attack on the target device. The set of profiled attacks includes Template attacks[5] and Stochastic cryptanalyses (aka Linear Regression Analyses) [16,47,48].
- The set of so-called non-profiling SCA corresponds to a much weaker adversary who has only access to the physical leakage captured on the target

device. To recover the secret key in use, he performs some statistical analyses to detect dependency between the leakage measurements and this sensitive variable. The set of non-profiled attacks includes Differential Power Analysis (DPA)[6], Correlation Power Analysis (CPA)[7] and Mutual Information Analysis (MIA)[8].

A recent line of works has investigated new profiling attacks based on Machine Learning (ML) techniques to defeat both unprotected and protected cryptographic implementations. These contributions focus mainly on two techniques: the Support Vector Machine (SVM)[9, 10]and the Random Forest (RF)[11]. Practical results on several data-sets have demonstrated the ability of these attacks to perform successful key recoveries. Over the past few years, there has been a resurgence of interest in using Deep Learning (DL) techniques which have been applied in several signal processing areas where they have produced interesting results. Deep learning is a parallel branch of machine learning which relies on sets of algorithms that attempt to model high-level abstractions in data by using model architectures with multiple processing layers, composed of a sequence of scalar products and non-linear transformations called activation functions. Several recent results have demonstrated that DL techniques have convincingly outperformed other existing machine learning approaches in image and automatic speech recognition. In this work, we propose to apply DL techniques in side channel context.

2 Overview on Techniques

2.1 DPA

Differential Power Analysis(DPA) attacks are the most popular type of power analysis attacks. This is due to the fact that DPA attacks do not require detailed knowledge about the attacked device. Futhermore, they can reveal the secret key of a device even if the recorded power traces are extremely noisy. The goal of DPA attacks is to reveal secret keys of cryptographic devices based on a large number of power traces that have been recorded while the device encrypt or decrypt different data blocks. In case of DPA attacks, the shape of the traces along the time axis is not so important. It analyzes how the power consumption at fixed points of time depends on the processed data. Hence, DPA attacks depend exclusively on the data dependency of the power traces. Following is the strategy followed in a DPA attack:

Step1: Choosing an Intermediate Result of the Executed algorithm. The first step of a DPA attack is to choose an intermediate result of the cryptographic algorithm that is executed by the attacked device. This intermediate result needs to be a function $f(d, k)$, where d is a known non-constant data value and k is a small part of the key. Intermediate resuts that fulfill this condition can be used to reveal k . In most attack scenarios, d is either the plaintext or the ciphertext.

Step2: Measuring the Power Consumption. The second step of a DPA attack is to measure the power consumption of the cryptographic device while it encrypts or decrypts D different data blocks. For each of these encryption or decryption runs, the attacker needs to know the corresponding data value d that is involved in the calculation of the intermediate result chosen in step1. We write these data values as vector $\mathbf{d}= (d_1, d_2, ..., d_D)$, where d_i denotes the data value in the i^{th} encryption or decyption run. During each of these runs the attacker records a power trace. We refer to the power trace that corresponds to data block d_i as $\mathrm{t}_i^{'} = (t_{i,1}, ...t_{i,T})$, where T denotes the length of the trace. The attacker measures a trace for each of the D data blocks, and hence, the traces can be written as matrix $\mathbf{T}$ of size $D \times T$. It is important for DPA attacks that the power traces are correctly aligned. This means that the power consumption values of each column $t_j$of the matrix $\mathbf{T}$ need to be caused by the same operation. In order to obtain aligned power traces, the trigger signal for the oscilloscope needs to be generated in such a way that the oscilloscope records the power consumption of exactly the same sequence of operations during each encryption or decryption run.

Step3: Calculating Hypothetical Intermediate Values. The next step of the attack is to calculate a hypothetical intermediate value for every possible choice of k. We write these possible choices as vector $\mathbf{k}= (k_1, ..., k_K)$, where K denotes the total number of possible choices for$k$. In the context of DPA attacks, we usually refer to the elements of this vector as key hypotheses. Given the data vector $\mathbf{d}$ and the key hypoteses $\mathbf{k}$, an attacker can easily calculate hypothetical intermedite values $f(d, k)$ for all D encryption runs and for all K key hypoteses. The following formula is used for this and it results in a matrix $\mathbf{V}$ of size $D \times K$.

$v_{i,j} = f(d_i, k_j)$ where $i = 1, ...D$ $j = 1, ...K$ Column j of $\mathbf{V}$ contains the intermediate results that have been calculated based on the key hypoteses k_j. It is clear that one column of $\mathbf{V}$ contains those intermediate values that have been calculated in the device during the Dencryption or decryption runs. $\mathbf{k}$ contains all possible choices for k. Hence, the value that is used in the device is an element of $\mathbf{k}$. We refer to the index of this element as ck. Hence, k_{ck} refers to the key of the device. The goal of DPA attacks is to find out which column of $\mathbf{V}$ has been processed during the D encryption or decryption runs.

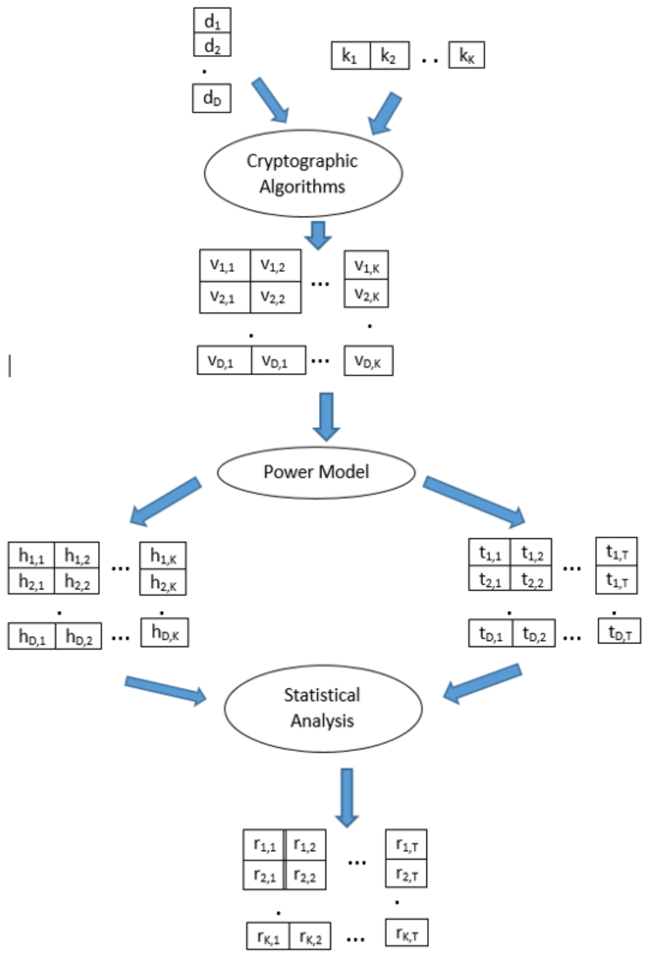

Figure 2: Block Diagram Illustrating the steps 3 to 5 of a DPA attack

Step4: Mapping Intermediate Values to Power Consumption values. The next step of the attack is to map hypothetical Intermediate values **V** to a matrix **H** of hypothetical power consumption values. For this purpose, the attacker uses the power models like Hamming-distance and Hamming-weight models. There are also many other ways to map data values to power consumption values, the mentioned are the most commonly used ones.

Step5: Comparing the Hypothetical Power consumption Values with the Power Traces. After having mapped **V** to **H**, the final step of a DPA attack can be performed. In this step, each column h_i of the matrix **H** is compared with each column $t_j$of the matrix **T**. The reult of this comparison is a matrix **R** of size $K \times T$, where each element $r_{i,j}$contains the result of the comparison between the columnsh_i and t_j. Higher the value $r_{i,j}$, the better the columns $h_i$and t_j match. The key of the attacked device can hence be revealed based on the following observation. The power traces correspond to the power consumption of the device while it executes a cryptographic algorithm using different data inputs. The intermediate result that has been chosen in step 1 is a part of this algorithm. Hence, the device needs to calculate v_{ck} during the different executions of the algorithm. Consequently, also the recorded traces depend on these intermediate values at some position. We refer to this position of the power traces as ct, i.e, the column t_{ct} contains the power consumption values that depend on the intermediate values v_{ck}.

The hypothetical power consumption values hck have been simulated by the attacker based on the values v_{ck}. Therefore the columns $h_{ck}$and $t_{ct}$are strongly related. In fact, these two columns lead to the highest value in **R**, i.e, the highest value of the matrix **R** is the value $r_{ck,ct}$. An attacker can hence reveal the index of the correct key ck and the moment of time ct by simply looking for the highest value in the matrix **R**. It can also happen that all values of **R** are approximately the same. In this case, the attacker has usually not measured enough power traces to estimate the relationship between the columns of **H** and **T**. The more traces an attacker measures, the more precisely the attacker can determine the relationship between the columns.

2.2 CNN

2.2.1 Perceptron

The perceptron is the simplest neural network model[12]. It is a linear classifier that uses a learning algorithm to tune its weights in order to minimize a so-called loss function[1] as described in below diagram. We detail hereafter how perceptron works to perform classification:

- first, an input vector $X = (x_1, ...x_n) \in R^n$ is presented as an entry to the perceptron.
- then, components of X are summed over the weights $w_i \in R$ of the perceptron connections (i.e. $w_0 + \sum_{i=1}^{n} w_i x_i$, with $w_0$being a bias).
- finally, the output of the perceptron is computed by passing the previously computed sum to an activation function[2] denoted f.

[1]The loss (aka cost, error) function quantifies in a supervised learning problem the compatibility between a prediction and the ground truth label (output). The loss function is typically defined as the negative log-likelihood or the mean squared error.

[2]In the case of the perceptron, the activation function is commonly a Heaviside function. In more complex models (e.g. the multilayer perceptron that we will describe in the next section), this function can be chosen to be a sigmoid function (tanh).

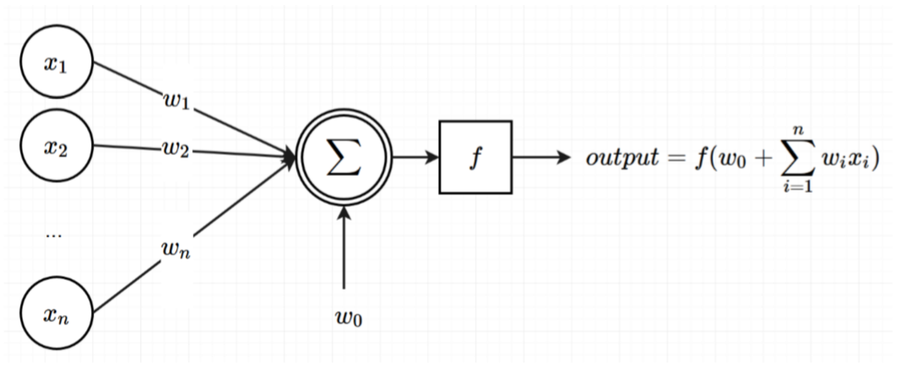

Figure 3: Representation of a Perceptron

During the training phase, the perceptron weights, initialized at zeros or small random values, are learned and adjusted according to the profiling dataset $(X^{(i)}, y_i)$. By e.g. applying a gradient descent algorithm, the goal is to find/learn the optimal connecting weights moving the perceptron outputs as close as possible to the correct labels/scores (e.g. to minimize the sum of squared differences between the labels $y_i$and the corresponding perceptron's output).

2.2.2 Multilayer Perceptron

A Multilayer Perceptron (MLP) is nothing more than a specific way to combine perceptrons in order to build a classifier for more complex data-sets. As shown in Figure 4, the information is propagated from the left to the right and each units (perceptrons) of a layer is connected to every unit of the previous layer in this model. This is called a fully connected network. Each neuron belongs to a layer and the number of layers is a parameter which has to be carefully chosen by the user.

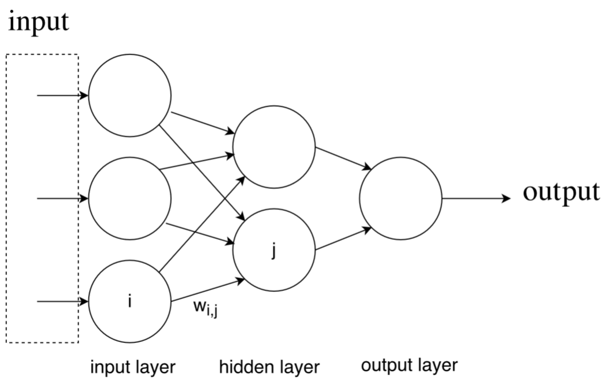

Figure 4: Example of MLP, where each node is a perceptron

An MLP is made of three different types of layers:

- Input Layer: in the traditional model, this layer is only an intermediate between the input data and the rest of the network. Thus the output of the neurons belonging to this layer is simply the input vector itself.
- Hidden layer: this layer aims at introducing some non-linearity in the model so that the MLP will be able to fit a non-linear separable data-set. Indeed, if the data that have to be learned are linearly separable, there is no need for any hidden layer. Depending on the non-linearity and the complexity of the data model that has to be fit, the number of neurons on the hidden layer or even the number of these layers can be increased. However, one hidden layer is sufficient for a large number of natural problems. Regarding the number of neurons on the hidden layers, it has been demonstrated that using a huge number of neurons can lead to over-fitting if the model that has to be learned is close to a linear one. It means that the algorithm is able to correctly learn weights leading to a perfect fit with the training data-set while these weights are not representative of the whole data. On the other hand, the opposite may happen: for a complex data-set, using too few neurons on the hidden layers may lead the gradient minimization approach to fail in returning an accurate solution.
- Output layer: this is the last layer of the network. The output of the nodes on this layer are directly mapped to classes that the user intends to predict.

Training a multilayer perceptron requires, for each layer, the learning of the weighting parameters minimizing the loss function. To do so, the so-called backpropagation[12] can be applied. It consists in computing the derivative of the loss function with respect to the weights, one layer after another, and then in modifying the corresponding weights by using the following formula:

$$\triangle w_{i,j} = -\frac{\partial E}{\partial w_{i,j}}$$

where E is the loss function and $w_{i,j}$denotes the weight of the connection between two neurons of indices (i, j).

2.2.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a specific kind of neural network built by stacking the following layers:

- A convolutional layer: On this layer, during the forward computation phase, the input data are convoluted with some filters. The output of the convolution is commonly called a feature map. It shows where the features detected by the filter can be found on the input data. Figure 5 shows an example of a convolutional layer where the input vector (X is represented as a matrix $X = (x_{i,j}) \in R^{t \times t}$ where t is smallest square integer greater than the size n of X viewed as a vector) and padded with zeros around the border. The output values can be expressed as $y_{i,j} = \sum_{a=1}^{m} \sum_{b=1}^{m} w_{a,b} x_{i+a,j+b}$,where $w_{a,b}$ denotes the weights of the filter viewed as an m-by-m matrix. During the backward computation, the filter weights are learned by trying to minimize the overall loss.

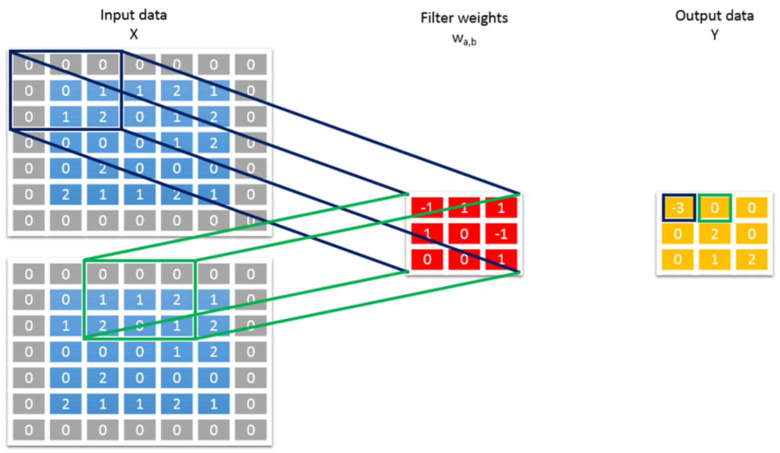

Figure 5: Example of Convolutional Layer where n=25, t=5 and m=3

- A Max Pooling layer: this is a sub-sampling layer. The feature map is divided into regions and the output of this layer is the concatenation of the maximum values of all these regions. Such layers can help reducing computation complexity and enhance the robustness of the model with respect to a translation of the input.
- A SoftMax layer: it is added on the top of the previous stacked layers. It converts scores from the previous layer to a probability distribution over the classes.

Learning the filters enables to extract high level features from the data. This step may therefore be used as a dimensionality reduction or a Points Of Interest (POI) selection technique (e.g. a PCA).

3 Experiment

A board, titled the "SASEBO GII" has been used to acquire the traces and its full hardware FPGA design is available. In the following section, we mention the procedure and analyse the results for DPA and CNN implementations.

3.1 DPA

Input to this attack is a set of power traces obtained during encryption of different plaintexts with the same key. In our context each trace is a file containg the power samples at different points of time during the process of encryption, plaintext and ciphertext.

3.1.1 Procedure -

The first step in the AES attack program is to split the 128-bit ciphertext message into byte long blocks. The AES-128 algorithm operates on each byte individually, which allows us to guess the 8-bit portion of the round key used for each byte individually. Focusing on one byte at a time, we take each byte of the ciphertext and run through the AES decryption method 256 times (once for each possible key). The result is an array of 256 potential values for the final round input.

Using the initial and final states we quantize the sensititve transition using the hamming weight model as $senstivetransition = initalstate \oplus finalstate$. We already have the power trace containing some samples. Now it is possible to determine a sample correlation between the power consumption and the Hamming distance data. We calculate the correlation between the power trace and the sensitive data using Pearson's sample correlation coefficient, given by:

$$r_{sb} = \frac{\sum_{i=1}^{n}(h_i^{(sb)} - \overline{h^{(sb)}})(p_i(t_j) - \overline{p(t_j)})}{(n-1)\sigma_h \sigma_{p(t_j)}}$$

The mean value across all traces of a point t_j is given as $\overline{p(t_j)}$ and the standard deviation of these values at a point t_j is given as $\sigma_{p(t_j)}$. For each trace i, there exists sensitive data which can be exploited. The sensitive data for a trace i is denoted as h_i, with the mean sensitive data over all traces $\overline{h_i}$, and standard deviation σ_h. Using this equation to correlate the power information to the sensitive data, We calculate this coefficient across every point(j=1 to 3253) in each power trace. Finally we get correlation coeficient between each of the 256 sensitive data values and the 3253 samples in power trace. This process is repeated using many power traces to build up the correlation coefficient. After a sufficient number of traces, there exists a peak in the correlation which corresponds to the correct key. Similarly every byte of the key can be obtained using the same procedure.

3.1.2 Results

Experiment is conducted with a set of 20,000 traces. All the 20,000 traces are obtained during encryption of different plain texts with the same key. We analyse 1^{st}, $8^{th}$and 16^{th}byte of the 10^{th}roundkey, although it can be done for any byte of any roundkey. Following are the results obtained.

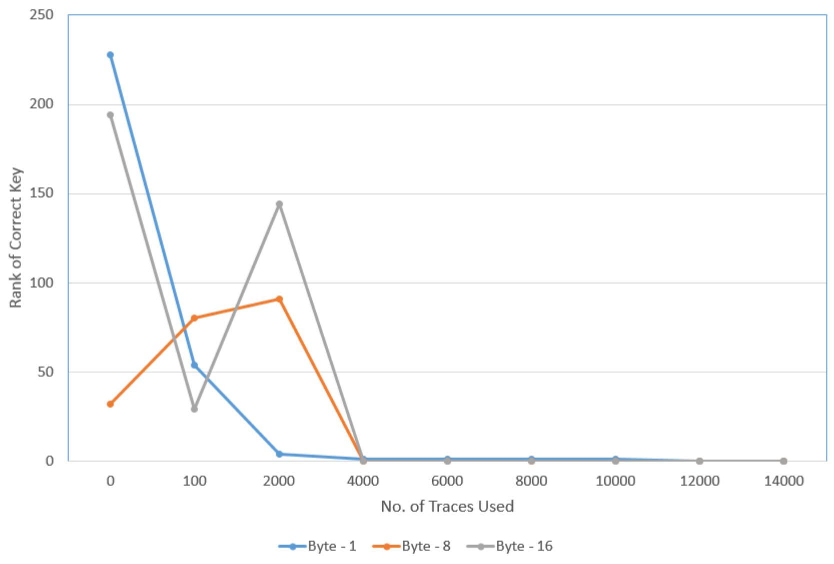

Figure 6: Evolution of Correct Key rank with increase in number of traces

On doing this experiment for different set of traces, we obtained the correct key(16 bytes) values for an average of 10000 traces, with 5700 being the minimum number of traces required, and 12000 being the maximum.

3.2 CNN based attack

Here we are dealing with profiled attack, we assume an attacker who has full control of a training device during the profiling phase and is able to measure the power consumption during the execution of a cryptographic algorithm. Then during the attack phase, the adversary aims at recovering the unknown secret key, processed by the same device, by collecting a new set of power consumption traces. To guarantee a fair and realistic attack comparison, we stress the fact that the training and the attack data-sets must be different. This attack contains two steps - Profiling phase and Attack phase. For profiling phase we give a large set of power traces with each trace containing 3253 samples, key, plain text and ciphertext. Large set of traces(in our case 10lakh traces) are required to obtain a better model. In attack phase we provide another set of power traces obtained during encryption of different plaintexts with the same key.

3.2.1 Procedure

Consider we are trying to find a particular byte of the key. And the operation, we are targeting on is AES SBox output at the end of 1st round and it is stored in variable $Z = Sbox[X \oplus k*]$ where X and $k*$ respectively denote the plaintext and the secret key. We motivate our choice towards targeting this non-linear operation by the fact that it is a common target in side channel analysis and that it has a high level of confusion.

Profiling phase - We create files like Z=0.txt, Z=1.txt,....Z=256.txt since there are the 256 different possible values of Z. For each power trace the value of $Z = Sbox[X \oplus k*]$ is calculated and the trace data is written into corresponding file. By the end of reading in all the power traces(in our case 10lakh traces) there is certain amount of data in each file(in our case 3000-4000 traces), the data in each of these 256 files are used for training corresponding model. In the end there will be 256 different models. The network used for training each model is shown in Figure 6. The input to the network is a powertrace of length 3253 and groudtruth key.

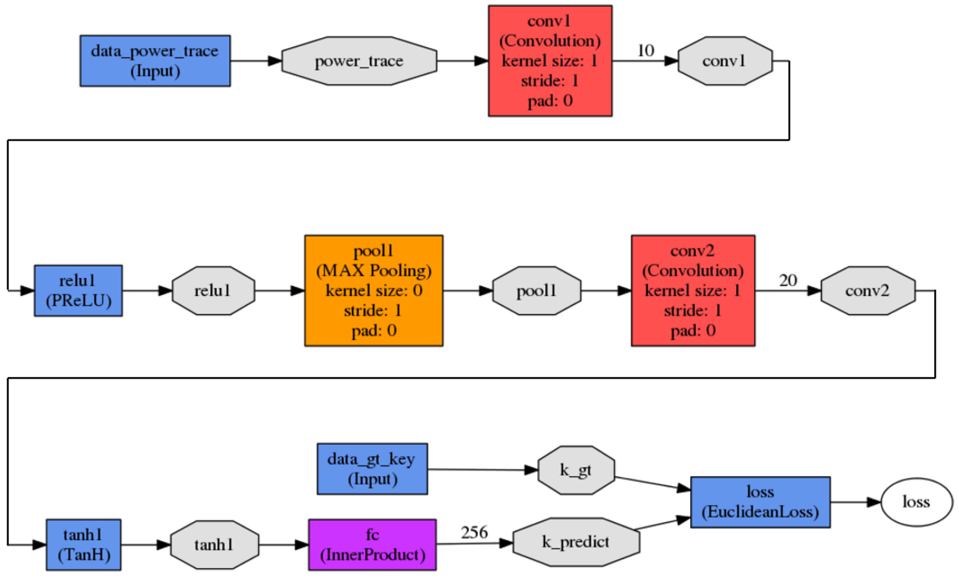

Figure 7: Network Model Used

Layer1: Convolutional Layer - It is used to get the local features within the powertrace. The CNN used is of kernel size 16 and there are 10 outputs coming out of it.

Layer2: ReLu - It is Rectified Linear Unit, used to introduce non-linearty in the fitting process which makes the network easier to fit complex functions.

Layer3: Pooling - It is used to reduce the complexity by reducing the effective length of powertrace. Max pooling of size 2 with stride 2 is used i.e. it takes the maximum of two adjacent values in the input and places it in the output, so the size reduces by half. It is used where there is no much dependence in the final output with respect to the adjacent values.

Layer4: Convolutional Layer - It is used to get the local features within the powertrace. The CNN used is of kernel size 10 and there are 20 outputs coming out of it.

Layer5: Tanh - It is Hyperbolic tangent function. It is used to introduce non-linearty in the fitting process which makes the network easier to fit complex functions.

Layer6: Inner Product - This layer takes the output obtained at the end of layer5 as input and maps those input values to 256 output values, with output values quantitatively representing the prob(k = 0), prob(k=1),.......prob(k=255).

Layer7: Loss - Loss between predicted key and groudtruth key is calculated and the loss is backpropagated to the previous layers to re-adjust the weights to reduce loss.

Attack Phase - In training phase we have obtained 256 models . Now in attack phase, we consider a trace, so we know plaintext x and we assume $k = 0, 1, ..., 255$. For each key assumption we calculate $Z = Sbox[X \oplus k*]$. Now we consider that particular Z model and obtain the probability of the same assumed key from the model. This way we get probability for each key assumption for a trace. We follow the same procedure for a set of traces obtained with same key. Now the final probabilities for each key asssumption is calculated by multiplying the corresponding key probabilities of all the traces. As the probabilities are small and might be negligble to compare, we take log of the probabilities and add them instead of multiplying the direct probabilities. Finally the result is an array of size 256, with each element proportionally representing the P(key = index of element). The index of the element with highest value is the predicted key. We can sort the array to find the rank of groundtruth key. This whole process can be repeated for predicting each byte of the key.

3.2.2 Results

In the experiment we are trying to predict the 1st byte of the key, although it can be repeated to obtain any byte of the key. Experiment is conducted with two sets of traces. One set contains power traces with 1^{st} byte of key = 0 and another set of traces with 1^{st}byte of key = 19. Following are the reults obtained:

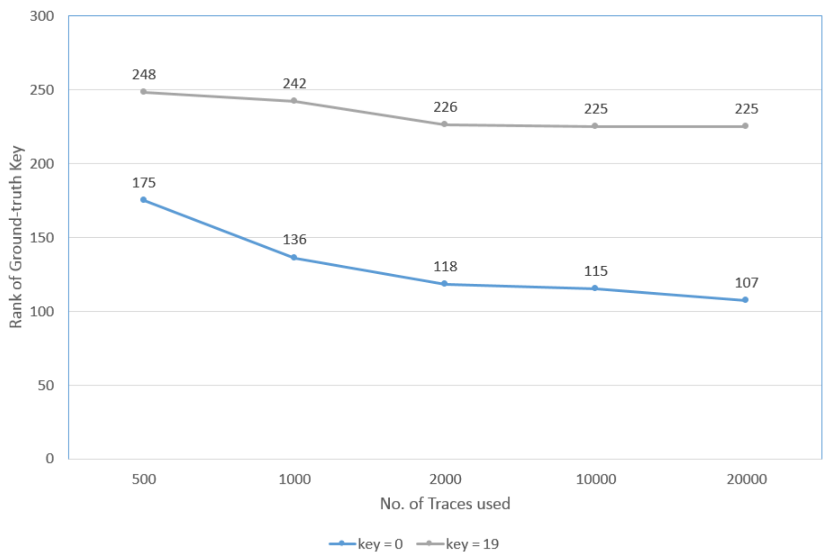

Figure 8: Evolution of Correct key rank with increase in number of traces

As we can see from the above graph that the rank of the groundtruth key is decreasing with increasing number of traces, but these are not the proper results. This happened because of choosing wrong parametric values like filter length, number of channels in convolutional layer etc. This model is predicting a key(although it is not equal to groundtruth key) for around 200 traces and the same key is obtained even after increasing the number of traces. This means that the groundtruth key can be predicted with around 200 traces if we set optimal parametric values. Profiling phase takes around 10-15 hours for those 10lakh training power traces, so it is difficult to find the optimal parameters by trail and error method. Techniques like evolutionary algorithms and genetic algorithms can be used and is considered as part of future work.

4 Conclusion and Future Work

In completing the project we were successfully able to recover the AES-128 cryptosystem keys by mounting a DPA attack. For AES-128, we were able to recover the subkey used with average of 10000 traces. From the success of our attack on the power traces from the DPA contest, it is clear that these types of side-channel attacks are very powerful when it comes to breaking a cryptosystem. What makes a DPA attack so powerful is that it can make plaintext or ciphertext only attacks, which greatly increase the versatility when attacking a target device. DPA-based attacks also perform much faster than other techniques such as exhaustive search. However, this type of side-channel attack requires access to the physical hardware in order to obtain the traces required for its execution.

As part of this project we study the application of deep learning techniques in the context of side channel attacks. The deep learning techniques are based on some nice features suitable to perform successful key recovery. The parametres like filter length, no.of channels etc. used for training models greatly effect the results, i.e, the predicted key doesn't match with the groundtruth key and even the rank of the groundtruth key in the obtained results will be high. The predicted key will not be equal to groundtruth key even if we use a large set of traces if the set of parameters chosen are not the optimal parameters.

Given the success of the attack on AES-128 implementations, there are a few directions in which this project could go towards future work. In current research, some of the only reliable models for power consumption to lead to a cryptosystems key have been the Hamming Weight and Hamming Distance models. One excellent way of improving our attack could come directly from a new power model or better utilization of the existing power models. This would allow higher correlations in the power traces and lead to more correct key guesses, resulting in the key space being narrowed down further to make exhaustive search techniques feasible. The way power traces are statistically correlated after the power model has been utilized is another source for additional work. More complex statistical methods could be explored in order to improve the correlations even further and allow for more information to be obtained from the power traces.

An alternative method to explore power analysis side-channel attacks for breaking cryptographic implementations would be template-based attacks[16]. Template-based attacks are considered to be one of the most powerful types of side-channel attacks. This is because the statistical functions involved capture a large amount of data from the power traces and utilize probability density functions in order to obtain correct key guesses.

Considering the CNN based attack, we can try to obtain a better model while profiling by choosing the optimal parametric values. One common technique to find the optimal parameters is to use evolutionary algorithms[13] and more precisely the so-called genetic algorithm[14]. These can be implemented as part of the code to find the optimal parameters, which would give proper results and also the key could be detected with less number of traces. We can even try different other deep learning techniques like Autoencoder(AE), Recurrent Neural Networks(RNN), Long and Short Term Memory Units(LSTM) for profiling and compare with the CNN method[15].

References

[1] Rijndael S-box . https://en.wikipedia.org/wiki/Rijndael_S-box.

[2] Rijndael mix columns.https://en.wikipedia.org/wiki/Rijndael_mix_columns.

[3] Rijndael key schedule. https://en.wikipedia.org/wiki/Rijndael_key_schedule.

[4] byte-oriented-aes – A public domain byte-oriented implementation of AES in C – Google Project Hosting. Code.google.com. Retrieved 2012-12-23.

[5] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In CHES, volume 2523 of LNCS, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.

[6] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In CRYPTO, volume 1666 of LNCS, pages pp 388–397. Springer, 1999.

[7] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In CHES, volume 3156 of LNCS, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.

[8] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In CHES, 10th International Workshop, volume 5154 of Lecture Notes in Computer Science, pages 426–442. Springer, August 10-13 2008. Washington, D.C., USA.

[9] C. Cortes and V. Vapnik. Support-vector networks. Mach. Learn., 20(3):273–297, Sept. 1995.

[10] J. Weston and C. Watkins. Multi-class support vector machines, 1998.

[11] L. Rokach and O. Maimon. Data Mining with Decision Trees: Theroy and Applications. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2008.

[12] C. M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, Inc., New York, NY, USA, 1995.

[13] A. E. Eiben and J. E. Smith. Introduction to Evolutionary Computing. SpringerVerlag, 2003.

[14] M. Mitchell. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA, 1998.

[15] Breaking Cryptographic Implementations Using Deep Learning Techniques. https://eprint.iacr.org/2016/921.pdf

[16] Template Attacks. https://wiki.newae.com/Template_Attacks.