Hacking AES-128

Timothy Chong
Stanford University
ctimothy@stanford.edu

Kostis Kaffes
Stanford University
kkaffes@stanford.edu

***Abstract*—Advanced Encryption Standard, commonly known as AES, is one the most well known encryption protocols. It is used in a large variety of applications ranging from encrypting classified documents in government agencies to allowing secure data transfer in everyday transfer protocols, such as HTTPS and FTPS. In this paper, we use template attack and Support Vector Machine to perform a side-channel attack on AES-128 and recover the cipher key from a power trace of a hardware encrypting device. Using SVM, we achieve an overall prediction accuracy of 35%, which is more than sufficient to retrieve the entire cipher key. In addition, we use a balanced SVM to counteract the skewing of our prediction classes.**

I. Introduction

The ubiquitous need for cryptography in a large number of tasks today has led to the development and usage of commodity cryptographic hardware modules. These modules are preferred to software implementations of cryptosystems due to their higher speed and lower power consumption. Most encryption algorithms today have been well studied and proved to be impervious to attacks in the information flow. However, the hardware that implements the encryption/decryption algorithms has to be studied more in depth as it can be vulnerable to side channel attacks. In cryptography, a side-channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms. For example, such an attack can exploit the power consumption of a cryptographic device which depends on the manipulated data and the executed instructions. The power consumption can be easily monitored on an oscilloscope by inserting a resistor in series with the ground. The obtained measurement is called a power trace. In this work we perform a side channel attack against the AES-128 encryption standard [1] using the power traces obtained from a large vendor's commercial hardware security module. We use statistical and machine learning techniques to extract information from the power trace that can be used to get the key. More particularly, our goal is to employ multinomial Gaussian distribution modeling and Support Vector Machine methods to get information about the output of a specific stage of the AES-128 encryption algorithm.

The rest of this paper is structured as follows. First, we present some relevant work on the topic. In section III we describe the AES-128 encryption algorithm. In section IV we discuss our attack methodology, the machine learning methods we use, and several adjustments to further improve them. In sections V and VI, we present the data set we use and the results of our attack. We conclude with some remarks and future directions.

II. Related Work

Template attacks, introduced by Chari et al. [2], deal with many of the shortcomings of the Simple and the Differential Power attacks although there are some countermeasures that can prevent them. Machine learning has been recently used extensively in side channel attacks since the better understanding of the physical behavior of a cryptosystem makes machine learning algorithms an important component of an attack strategy. Hospodar et al. [3] use the Least-Squares Support Vector Machine and compare between different feature selection methods and kernel parameters. However, they only consider a single S-Box look-up and do not perform actual key recovery. Lerman et al. [4] empirically evaluate the effectiveness of both several feature selection methods and of different supervised learning approaches such as Self Organizing Map, Support Vector Machine and Random Forest. Finally, He, Haffe, and Zou [5], after analyzing DES, conclude that the type of feature selection method used is not very important regarding key extraction.

III. AES-128

AES-128 uses a 128-bit cipher key to encrypt 128-bit input blocks. The encryption flow of the algorithm is shown in Figure 1.

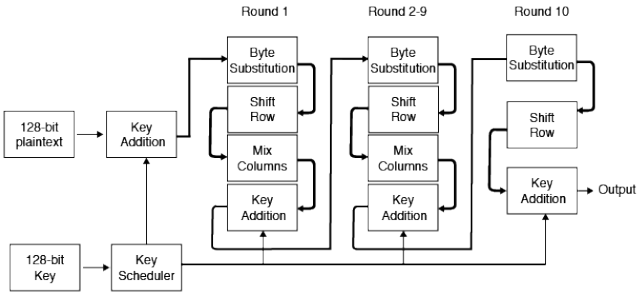

Fig. 1. Block diagram of AES encryption

In the beginning, the round key is added to the initial input text using bit-wise XOR. This operation is known as Key Addition, which is also used in subsequent phases. The encryption then enters a chain of 10 rounds of operations. The cipher key generates a different round key for each round using the Rijndael key schedule as well as for the beginning

key addition. Each of the next 9 rounds contains four phases, which includes Byte Substitution, Shift Row, Mix Columns, and Key Addition. In Byte Substitution operation, each byte of the matrix is substituted using an 8-bit substitution box, the Rijndael S-box. In Shift Row operation, the rows perform a left shift in an ascending order with the first row performing no shift at all and the fourth row shifting 3 positions. In Mix Columns, each column is transformed using the Rijndael mix columns function. The final round is identical to the previous rounds with the exception that Mix Columns operation is skipped. The output of each round is used as the input of the next round, while the output of the final round is the encrypted cipher text.

IV. Attack Methodology

In this work, our goal is to use machine learning techniques to predict the key used by an encryption device that implements AES-128 in hardware. We assume that the power trace P depends on the cipher key C and the input text I, i.e. $P = f(C, I)$. However, instead of trying to directly predict the key from the power trace, we target the hamming weight of each byte of the output of the Byte Substitution operation of the first round (OBFS) of the encryption algorithm (Figure 1). The term *hamming weight* refers to the number of 1's in the binary representation of a number. For example, the decimal number $9_{(10)}$ is $1001_{(2)}$ in binary, which has a hamming weight of 2 because it has two 1's. The hamming weight of the output of this particular phase of the encryption algorithm is a good target for AES side-channel power analysis for two main reasons. First, the power trace is highly correlated with the hamming weight of a value stored to the internal registers of a hardware device. In the AES case, the OBFS is the first time where registers are used to store some value related to the key, so it is a good target for our attack. Secondly, the input key is expanded and key additions are performed at the end of each round. If we wanted to target values in later rounds of the algorithm, the complexity of our side-channel analysis would grow because operations such as Shift Row and Key Addition would have to be taken into account. All operations in the AES algorithm run in 8-bit granularity, and each 8-bit sub-number is in-place until the first Shift Row operation. Therefore, with the hamming weight of each 8-bit sub-number in the output of the first Byte Substitution phase, one can retrieve the secret key since this hamming weight output is a function of the input secret key and the plain text. In the following section, we will describe the procedure by which we predict the hamming weight of the first 8-bit of the OFBS given the power trace, the first 8-bit of the input cipher text, and the first 8-bit of the plain text. It is important to note that the same analysis can be done on all subsequent 8-bit sub-numbers to retrieve the entire 128-bit key.

A. Feature Selection

Machine learning techniques usually require some kind of data processing before the training phase. In our case, each power trace consists of thousands of time-series values, most of which are not correlated with the cipher key. Thus, if every point of a power trace was considered a feature of the training set, the training phase would take too long and lead to overfitting and increased variance no matter which machine learning algorithm is used. To address that, we use point of interest selection and principal component analysis (PCA) to reduce the dimensionality of our input data.

1) Point of interest selection: We use two methods in order to identify d points of interest in our long power traces, which are then used for template attack as described in Section IV-B.

In the first method, we calculate the average for each point among all traces associated with a specific OFBS hamming weight. Since a 8-bit output has only 9 possible hamming weights, we obtain 9 mean power traces. We then calculate the sum of the absolute pairwise difference among all combinations of these 9 traces to obtain only one "trace". Finally, we select the d highest peaks in this trace as our points of interest.

In the second method, the Pearson correlation between each point in the power trace and the OFBS hamming weight is calculated. This metric represents the linear relationship between two data sets. We choose the d points with the highest Pearson correlation to be our point of interest.

2) Principal component analysis: PCA tries to identify the subspace in which the input data lie and to get rid of redundant dimensions. First, we need to preprocess the data by zeroing out the mean and rescaling each coordinate to have zero variance. If we assume that we want to project the data to only one dimension then we need to find a unit vector u so that when the data is projected onto u's direction the variance of the projected data is maximized. i.e. we need to maximize:

$$u^T(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T})u$$

Given the constraint $||u||_2 = 1$, the minimizer is the principal eigenvector of $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T}$. In order to get the d-dimensional subspace of the data that maximizes variance, we choose $u_1, \ldots, u_d$ to be the top d eigenvectors of Σ and we get:

$$y^{(i)} = \begin{pmatrix} u_1^T x^{(i)} \\ \ldots \\ u_d^T x^{(i)} \end{pmatrix}$$

B. Template Attack

Template attacks [2] are a very successful type of profiling side-channel attacks. The method uses a multinomial Gaussian distribution as its underlying model. It assumes that given the OFBS hamming weight of a given power trace, the power values (x) of the d points of interest (IV-A1) are going to vary according to a d dimensional Gaussian distribution with probability:

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

where $\mu \in \mathbf{R}^d$ is the mean vector, and $\Sigma \in \mathbf{S}_{++}^d$ is the covariance matrix.

To generate the model for a particular hamming weight class, we collect the power values of the d points of interest pertaining to that hamming weight. We can then generate a mean vector and a co-variance matrix for power values of that hamming weight class at these points of interest. With 9 possible hamming weight, we generate 9 such Gaussian distributions, each of d dimensions. To make a prediction given an input power trace, we simply calculate the probability of the d points of interest with our generated models assuming that they are generated from each of the 9 models. The model with the highest probability is chosen to be our hamming weight prediction.

C. Support Vector Machine

The Support Vector Machine (SVM) family of machine learning algorithms performs classification by finding a decision boundary that maximizes the geometric margin. In other words, given training vectors $x_i \in \mathbb{R}^p, i = 1, ..., m$, in two classes, and a vector $y \in \{1, -1\}^m$, we train the SVM by solving the following primal linear programming problem:

$$min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \zeta_i$$

subject to $y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0, i = 1, \ldots, m$.

One of the problems with any SVM method is that it is designed to perform binary classification. There are two basic approaches on how one can perform multi-class classification using SVM, "one-against-one" and "one-against-all"[6]. In the "one-against-one" approach a classifier is constructed for each pair of classes and the class which receives the most votes is selected. In the event of a tie, the class with the highest aggregate classification score, i.e. the sum of the pair-wise classification confidence levels for all the binary classifiers, is selected. For this method if we have n classes, $\frac{n*(n-1)}{2}$ classifiers are required. In the "one-against-all" approach a classifier is constructed for each class, i.e. n classifiers and makes a binary decision between that class and the rest of the class. The class that has the decision function with the highest value is selected. We use the "one-against-one" method because we have a relatively small number of classes (9) and the $O(n^2)$ classifier number is manageable. Furthermore, Hsu and Lin [7] have shown that it provides higher accuracy for a wide range of problems and data sets.

Another problem that we have to deal with is that the distribution of hamming weights for numbers with a set length is inherently skewed. For example, for an 8-bit binary number, there is only one number with hamming weight 0 ($0_{(10)}$) and one number with hamming weight 8 ($255_{(10)}$), whereas the frequency of other hamming weight classes is much higher. This observation drives us to adjust our algorithm to ensure balanced precision among hamming weight classes by adding weights to the samples. Thus, for each binary classifier that we train, we assign weights inversely proportional to class frequencies in the input data.

Our first step is to do a design space exploration for different kernel functions and parameters. An initial attempt to use a simple linear or polynomial kernel is quickly abandoned since the data set is clearly not separable by such kinds of functions. After fine tuning of the parameters of the SVM, the radial basis function kernel (rbf), $K(x_1, x_2) = exp\{\frac{-||x_1 - x_2||_2^2}{\sigma^2}\}$ proofs to be successful. The parameters that we use are C and $\gamma = \frac{1}{\sigma^2}$. The C parameter trades off misclassification of training examples against simplicity of the decision surface. Thus, the lower the C parameter, the smoother the decision surface while a high value of C means that the labels of all training data points, even outliers, are taken strictly into account. The γ parameter defines how far the extent of the influence of each training example. Low values mean "far" and high values mean "close". In our experiments we have chosen $C = 100$ and $\gamma = 10^{-6}$, values that seem to provide the highest prediction accuracy. Intuitively, the high value of C and low value of γ we have chosen mean that every data point (trace) is important and decisive for the outcome of the prediction.

In order to avoid the issue of the skewed hamming weight distribution altogether, we also try to predict the value of the OBFS directly. More specifically, we target the first bit of the OBFS and use both balanced and unbalanced SVM with linear and rbf kernels.

D. Data Set and Evaluation

Our data set consists of 20k power traces collected from a hardware security module[1] when it is actively performing AES encryption. Each trace contains 80k power values along the time axis. We divide our data set into a 17.5k training set and a 2.5k testing set. All traces have the same key and random plain texts, but our algorithm should work either random keys because the OFBS is a function of both the plain text and key.

We implement both template attack and Support Vector Machine in Python using packages such as numpy and sklearn. Points of interest selection method is used for template attack, while Principal Component Analysis is used for Support Vector Machine.

V. Results

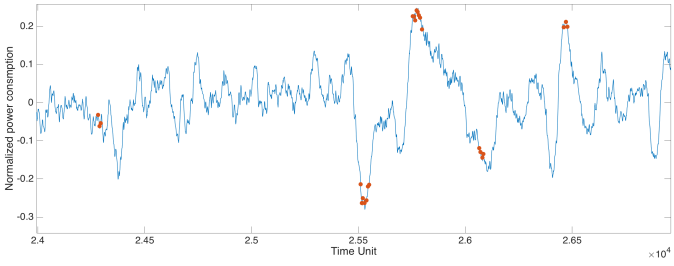

Fig. 2. 25 Points of interest example

Figure. 2 shows an example plot of the points of interest selected by maximizing the differences across OFBS hamming weight classes. This particular plot shows the normalized

[1]The exact model and specifications of the module is unknown due to non-disclosure agreement from Google Inc., where the data set was collected.

power trace of the class with hamming weight 4 and 25 points of interest. The plot is zoomed into the region where the points of interest are selected for clarity. As one may have expected, the points of interest are found to be located around the peaks and valleys of the power values because these points can correspond to power numbers when values are being stored to registers, which deviate the most for different hamming weight classes.

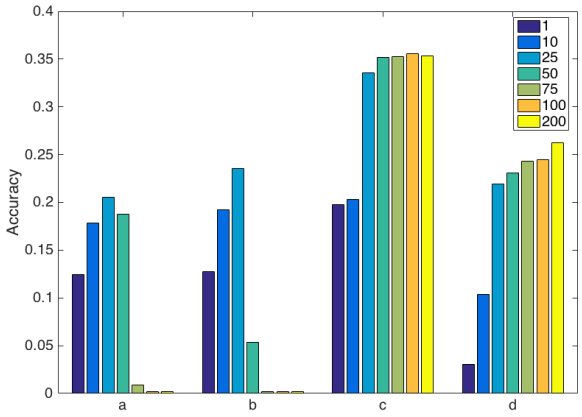

Fig. 3. Overall accuracy of our algorithms with different number of points of interest/ feature dimension. a) Template attack with point of interest selection method 1. b) Template attack with point of interest selection method 2. c) SVM with PCA. d) balanced SVM with PCA.

The overall accuracy of our algorithm is shown in Figure 3. The left two sub-bars represent template attack with different number of points of interest along the horizontal axis, while the right two sub-bars represent the SVM results with the number of PCA features along the horizontal axis.

Looking at template attack alone, the two point of interest selection method seem to have similar overall performance, with accuracy at around 20% to 25% with 25 points of interest. SVM in general gives better, although not significantly higher, accuracy. Perhaps due to over-fitting, the accuracy of template attack becomes abysmal as we further increase the number of points of interest beyond 50, whereas the performance of SVM continues to increase as we increase the number of PCA features to 200, even though the accuracy seems to plateau at higher number of features.

It is important to note that even though the accuracy is less than 40%, the actual key retrieval involves the prediction of hamming weight with a number of input traces of the identical key and random plain texts. As long as our classifier performs better than a random guess (1/9), we should still be able to guess the correct key given a sufficient number of power traces.

The precision (true positive divided by the sum of true positive and false negative) across the 9 classes of OBFS hamming weights is shown in figure 4. The two point of interest selection methods seem to have similar precision across different hamming weights. On the other hand, the precision for the unbalanced SVM is much higher for hamming weight classes 1 and 4, while the other classes are doing very poorly. In the balanced SVM case, however, the precision performance is more balanced among all the different classes. The balanced case sacrifices overall accuracy but retains higher overall precision across different hamming weight classes, which can be helpful if we want to sustain reasonable accuracy for all classes.

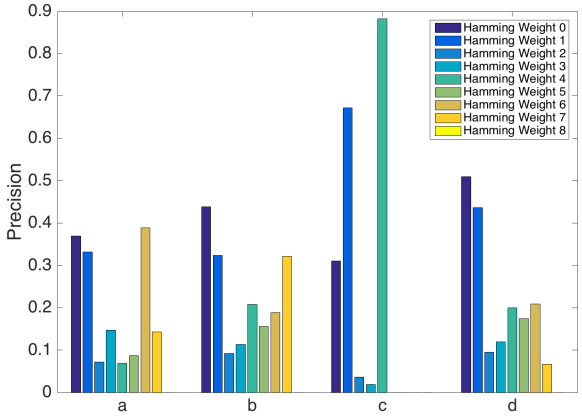

Fig. 4. Precision of OBFS hamming weight prediction. a) Template attack with 25 points of interest selected by method 1. b) Template attack with 25 point of interest selected by method 2. c) SVM with 200 PCA features. d) balanced SVM with 200 PCA features.

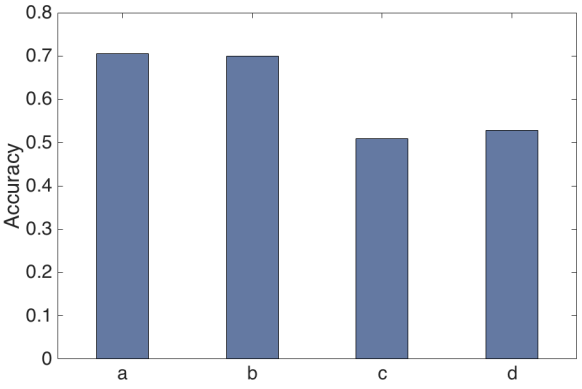

Fig. 5. One-bit prediction accuracy. a) Un-balanced SVM with RBF kernel. b) Balanced SVM with RBF kernel. c) Un-balanced SVM with linear kernel. d) Balanced SVM with linear kernel.

The accuracy of the one-bit prediction with Support Vector Machine is shown in Figure. 5. Since we are using SVM as a binary classifier, we expect there is no significant difference between the balanced and unbalanced implementation. SVM with linear kernel performs nothing better than a random classifier, with an accuracy of 50%, possibly because our data set is not linearly separable. The RBF kernel case gives an accuracy of 70% probably due to the infinite numbers of

features it uses. Such an accuracy allows the prediction of the key if enough attack traces are available.

VI. Conclusion and Future Work

In our work we use multinomial Gaussian distribution modeling and Support Vector Machine to predict either the hamming weight or directly the value of the OBFS. To our knowledge, this is the first time that a direct prediction of the hamming weight is attempted. We have shown that it is not a trivial task to predict the hamming weight using a single attack trace. It is worth noting that the inherent skewing of the hamming weight distribution makes it harder to sustain high accuracy while taking into account low frequency hamming weights. To alleviate this problem, we implemented the one-bit predictor which does not have to deal with hamming weights. The performance of this prediction has proved to be on par with similar attacks against the DES encryption standard [5].

This work can be expanded to two different directions: improving the accuracy of the prediction of the hamming weight and exploring other methods of attack. The former includes the isolation of the part of the power trace corresponding to the phase of the encryption algorithm we are attacking as well as the use of other machine learning techniques in addition to SVM such as Random Decision Forests. Another promising direction could employ neural networks where each trace could be associated with a neuron, and the network would be organized to group similar traces together. Finally, it would be interesting to test the methods proposed in this paper against side-channel resistant hardware encryption devices. Countermeasures implemented in such devices include, but are not limited to, adding random noise, shifting substitution box and using constant weight code so that any value written in the registers has always the same hamming weight.

Acknowledgment

The authors would like to thank Kevin Nicholas Kiningham for the initial idea for the project and for providing us the trace data set.

References

[1] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[2] S. Chari, J. R. Rao, and P. Rohatgi, *Template Attacks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 13–28.

[3] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, no. 4, p. 293, 2011. [Online]. Available: http://dx.doi.org/10.1007/s13389-011-0023-x

[4] L. Lerman, G. Bontempi, and O. Markowitch, "Power analysis attack: An approach based on machine learning," *Int. J. Appl. Cryptol.*, vol. 3, no. 2, pp. 97–115, Jun. 2014. [Online]. Available: http://dx.doi.org/10.1504/IJACT.2014.062722

[5] H. He, J. Jaffe, and L. Zou, "Side channel cryptanalysis using machine learning," *CS229 Project*.

[6] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[7] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Trans. Neur. Netw.*, vol. 13, no. 2, pp. 415–425, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/72.991427