Breaking AES-128: Machine Learning-Based SCA under Different Scenarios and Devices

Sara Tehranipoor and Nima Karimian
Lane Department of Computer Science and Electrical Engineering
West Virginia University
West Virginia, USA 26506-6109
Email: sara.tehranipoor, nima.karimian@mail.wvu.edu

Jack Edmonds
Department of Electrical and Computer Engineering
Santa Clara University
Santa Clara, USA
Email: jsedmonds@alumni.scu.edu

***Abstract*—Machine learning-based side-channel attacks (MLSCAs) have demonstrated the capability to extract secret keys from AES by learning the correlation between leakages from power traces or timing of AES execution. Previous work has focused on unmasked AES, the captured power traces for profiling and testing have been collected from the same device, and they are primarily implemented on microcontrollers. In this paper, we present a comprehensive MLSCA that considers both masked and unmasked AES running on software and hardware with a side-channel leakage model under four scenarios involving two target boards (Artix-7 XC7A100T FPGAs and STM32F415 microcontrollers) and different keys for training and testing the model. Our implementation results indicate that support vector machines outperformed other machine learning techniques on masked software and unmasked software AES with only 4 traces. Long short-term memory networks were found to outperform other techniques on unmasked hardware AES (FPGA) with only 283 power traces.**

***Index Terms*—Side-channel analysis, machine learning, AES-128, Cryptography, FPGA.**

I. Introduction and Related Works

Some of the cryptographic algorithms unintentionally leak information about the processed data and such attacks that exploit such knowledge in order to recover cryptographic secrets are called side-channel attacks (SCAs). SCAs extract secrets from physical devices by exploiting vulnerabilities in the implementation by finding the correlation between the physical measurements (power consumption, electromagnetic leaks, timing information, etc.) taken at various points during the computation and the internal state of the processing device. An attacker can deduce the internal state of a device and then retrieve the related sensitive information.

AES-128 algorithm is a symmetric encryption algorithm that has been shown to be vulnerable against SCAs. In publications including [11], [10], [17], and [1], studies involving machine/deep learning classifiers are discussed in the context of attacking AES, and the authors of [1] have provided AES-128 side-channel data collected from a microcontroller in their ASCAD database. In [5] and [6], cross-device attack results using deep neural networks are presented for AES-128 on microcontrollers. In [18], the authors "apply Deep Learning techniques in a Non-Profiled context, where an attacker can only collect a limited number of side-channel traces for a fixed unknown key value from a closed device." In [4], the authors use a "supervised learning-based approach for inferring applications executing on an android platform based on features extracted from EM side-channel emissions and software exposed dynamic voltage frequency scaling (DVFS) states." In [20], instead of using a convolutional neural network purely as an attacking algorithm, the authors present results of a side-channel attack on an FPGA-based convolutional neural network to recover images. In [12], machine learning based side-channel attack (MLSCA) is applied to evaluate the elliptic-curve cryptography algorithm, and in [2] to evaluate RSA. Other papers provide overviews of the topic include [16] and [7]. However, the majority of previous work considered breaking AES-128 using same key where the power traces collected in the same device during profiling and tested on same key using same device. While in the ideal scenario, the device that has been used for profiling phase should not be considered in testing phase. Additionally, most of the literature works do not consider different key and different devices with both unmasked/masked software/hardware. In this work, we will expand upon existing works such as these by providing a single set of experimental results of cross-device attacks (involving 2 keys) on 3 different implementations of AES-128 (unmasked software, masked software, and unmasked hardware) across two device types (MCU and FPGA) with 8 different machine learning classifiers.

To summarize our work, we evaluated the resistance of AES-128 implemented on a pair of STM32F415 MCUs and XC7A100T FPGAs in four different scenarios against MLSCA, i.e. profiling one target and key and attacking the same target and key, profiling one target and key and attacking a different target with the same key, profiling one target and key and attacking a different target with the same key, and profiling one target and key and attacking a different target with a different key. Eight machine learning classifiers including both classical and deep learning algorithms were used in combination with a side-channel leakage model to do this. We found that without implementing SCA-specific countermeasures (such as masking in our case), secret encryption keys can be vulnerable to this form of attack. Our main contributions are summarized as follows:

(1). Experimental results of MLSCA attacks on the STM32F415 running AES-128 in four scenarios involving two

MCUs and two keys, showing that unmasked software AES-128 is extremely vulnerable to attacks with the 8 machine learning classifiers used, but that masked software AES-128 can prevent a successful attack when the attacked key is different than the profiled key.
(2). Our experimental results on the FPGA XC7A100T running AES in four scenarios involving two FPGAs and two keys, showing that while not as vulnerable as unmasked software AES-128 on the MCUs (as more traces were required for a correct prediction), a successful attack can still be performed in all four scenarios evaluated with 8 machine learning classifiers used.
(3). We collected new power traces using both XC7A100T FPGA and STM32F415 microcontroller for both masked and unmasked AES-128. The database along with a machine learning code on Jupyter Notebook will be publicly available to allow for reproducibility and expansion upon our work hosted on GitHub.

The rest of the paper is organized as follows. In section II, we provide background information on side-channel analysis, machine learning, and AES-128, In section III, we provide an overview of our evaluation methodology, and in IV we discuss our results. In section V, we conclude the paper.

II. Background

A. The Advanced Encryption Standard

The Advanced Encryption Standard (AES) is currently one of the most extensively utilized symmetric-key algorithms. It is widely used in the private sector and is also the primary encryption and decryption method employed by the US government for sensitive data. AES can operate with 128, 192, or 256-bit keys, which is a significant step up from what is commonly regarded as its predecessor, DES (the Data Encryption Standard), which was developed by IBM in the 1970s. AES is a block cipher that processes chunks of bits together instead of processing each bit separately. It works on 16-byte blocks of data (128 bits in total) regardless of the key size. A set of operations is carried out on the 4x4 state array of this data for a specific number of rounds, depending on the key size. Specifically, AES-128 has 10 rounds, AES-192 has 12 rounds, and AES-256 has 14 rounds as shown in Figure 1.

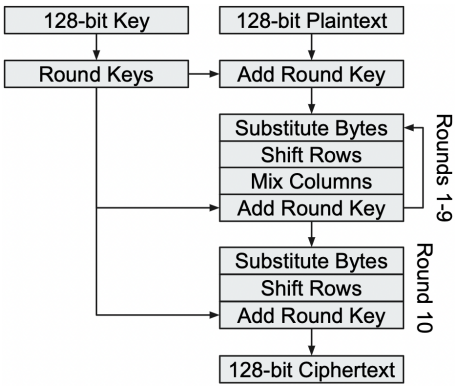


Fig. 1. Encryption flow of AES-128.

During the **"add round key"** operation, the state array bytes (which hold the plaintext byte values along with any required padding) are XORed with the corresponding round key bytes. The **"substitute bytes"** or S-box operation substitutes each of the 16 bytes in the state array with a value in the Rijndael S-box. The values of the bytes prior to the substitution are used as indices to determine what values to substitute them with from the S-box lookup table [1]. The table itself is static across AES implementations and there are 256 values in it because a byte has 2 to the 8 = 256 possible values, thus allowing for indices from 0 to 255. One countermeasure commonly used to protect cryptographic algorithms against SCA is masking, and this technique was applied in one of the two AES-128 implementations we evaluated. Specifically, the masked AES-128 implementation XORed each of the S-box output bytes with an additional mask value in an attempt to disrupt the first-order relationship between the power consumption spike resulting from this operation and the S-box output byte values. XOR is invertible, and the mask can later be removed so long as its value can be recalled. The **"shift rows"** operation involves circularly shifting the rows of the state array to the left. The first row remains unshifted, the second row is shifted once to the left, the third row is shifted twice to the left, and the fourth row is shifted three times to the left. Because it is circular, a value on the far left of a given row will become the value on the far right after the next shift. The **"mix columns"** operation involves computing the matrix-vector product of an MDS (Maximum Distance Separable) circulant matrix and each column of the state array. A circulant matrix that is MDS is intentionally constructed to conceal the relationship between the plaintext and ciphertext. It should be noted that this operation does not show up in the final round, as it was determined when the algorithm was developed that including it would not significantly add to the algorithm's security in the final round.

To create **"round keys"** AES employs a key schedule routine that expands the primary key into 10 additional 128-bit subkeys. To obtain the first column of the first round key's array, the far-right column of the original key array is taken, shifted upward by one, and then subjected to an S-box substitution of the bytes. The resulting column is XORed with the first column of the key array, and then XORed with a 4x1 vector that has a predefined constant for its first value, which depends on the round number and can be found in the AES round constant table, and zero for its second, third, and fourth values. The result of the second XOR is the first column of the first round key's array, which can be envisioned as being placed in a new fifth column of the key array. The next three columns that make up the remainder of the first round key's array are generated similarly, but the column to the right of the first column of the original array plays the role that the first column played in generating the second column of the first round key's array, and the newly generated column takes

the place of the far-right column of the original key array in its generation. In short, the index of the columns used in the subkey column generation routine will continue to increment by one until all four columns of all round key arrays have been generated. Note that there are actually 11 instances of the "add round key" operation in AES-128, or the number of rounds plus one in general. The first of these "add round key" operations typically use the original key itself, which is often referred to as the zeroth round key.

B. Side-Channel Analysis

Side-channel analysis (SCA) is a method of attacking cryptographic algorithms that involves exploiting hardware vulnerabilities by analyzing device information associated with the internal workings of the algorithm. This involves analyzing things like device power consumption, electromagnetic radiation, timing, and sound to learn information about a system and reveal information that was intended to remain unknown to unauthorized parties. Side-channel information can be thought of as the side effects of a device's operation. While SCA has been around for several years, only in the past couple of decades has there been a surge of interest in its potential as a cheap, non-invasive, or semi-invasive method of attack. Our research has focused on pairing SCA with machine learning, specifically using device power consumption as our chosen type of side-channel information. Our approach, which is based on machine learning, belongs to the subcategory of side-channel template attacks. In this type of attack, the attacker creates a "profile" of a device they have full control over, and then uses the information obtained from that device to attack other similar devices [3].

III. Methodology

A. Experimental Setup

In order to capture power traces from the target board, a shunt resistor was installed in the power supply rail of each target board (XC7A100T FPGAs and STM32F415 MCUs) to monitor and record voltage readings. The primary interface connection between the target devices and the workstation consisted of the CW1173 ChipWhisperer-Lite and the CW308 UFO Board. The collected data pertained to the encryption executed by the two target devices, as documented in the reference [13]. Our hardware setups have been depicted Figures 2 and 3. We also employed various Python packages such as ChipWhisperer, Keras/TensorFlow, Scikit-learn, NumPy, Pandas, and Matplotlib libraries for data evaluation. For both MCU and FPGA devices, we conducted evaluations in four scenarios. These scenarios involved profiling a single target and key, and subsequently attacking the same target using the same key. Additionally, we profiled a single target and key, and then attacked the same target using a different key. Furthermore, we profiled a single target and key, and performed an attack on a different target using the same key. Lastly, we profiled a single target and key, and carried out an attack on a different target using a different key.

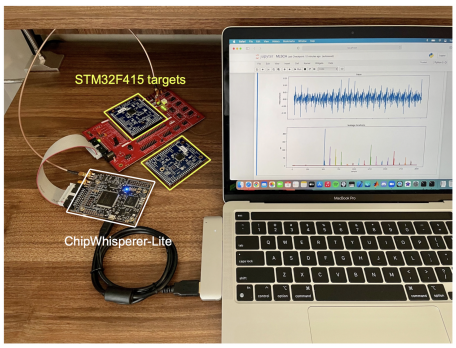


Fig. 2. Hardware setup: recording power traces from STM32F415 microcontroller board using CW308 UFO board.

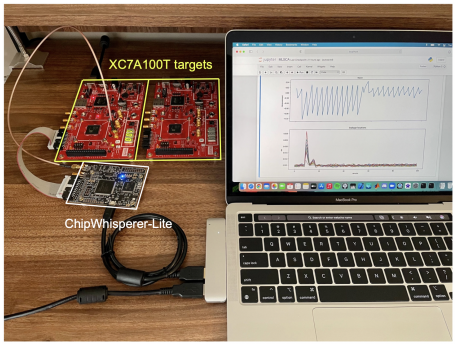


Fig. 3. Hardware setup: recording power traces from XC7A100T FPGA board using ChipWhisperer-Lite

B. Evaluation Procedure

*1) **STM32F415 MCU**:* To assess the machine learning-based side-channel analysis (SCA), we initially focused on the STM32F415 MCU. For this evaluation, we gathered a dataset of 2,000 samples, representing 60,000 distinct encryptions performed on randomly generated 128-bit plaintext using a fixed 128-bit key. These encryptions were triggered by a sequence of encryption operations. During our investigation, we discovered that a dataset of 2,000 samples adequately captured the operation of interest to us in the initial round of encryption. In order to conduct the evaluation tests, we collected an additional set of 10,000 power traces from the same target devices. These power traces were obtained after profiling the target devices with one set of encryptions using the same key as the profiling set, and another set using a different key. To ensure a fair evaluation, we incorporated the concept of unseen board and unseen key during the attack phase. This means that we conducted attacks using a different board and a different key than those used during the profiling phase. By employing different boards and keys in the profiling and attack stages, we aimed to assess the robustness and generalization capabilities of our approach. After collecting the data, we proceeded to analyze the profiling set using a side-channel leakage model. This model helped us identify the samples within our collected traces that corresponded to the operation of interest as attackers. Specifically, for both the

unmasked and masked software implementations of AES-128 on the MCUs, the operation we focused on was the "substitute bytes" or S-box operation in the first round. Studies have demonstrated that the power consumption associated with the S-box operation exhibits a correlation with its output values. If an attacker can accurately predict these values, it becomes feasible to reverse-engineer the secret key that was employed in the encryption process. However, this assumption relies on the attacker having knowledge of the specific random plaintext used for each encryption. In reality, there is only one value out of 256 possible values for the first key byte that, when XORed with the first plaintext byte, will yield the predicted first output byte of the S-box operation. This pattern continues for subsequent bytes of the key and plaintext. Figure 5 shows a general flow of the AES operations involved in this leakage model.

We also performed a signal-to-noise ratio (SNR) calculation on information from the profiling set to identify the samples of interest to us that corresponded to the processing of the 16 bytes. Figure 4 (a) displays the result of the SNR calculation for the unmasked software profiling set. The peaks in the bottom plot indicate which samples are related to the 16 S-box output bytes in the first round, while the top plot shows one of the 60,000 collected power consumption traces. Figure 4 (b) shows the result of the masked implementation.

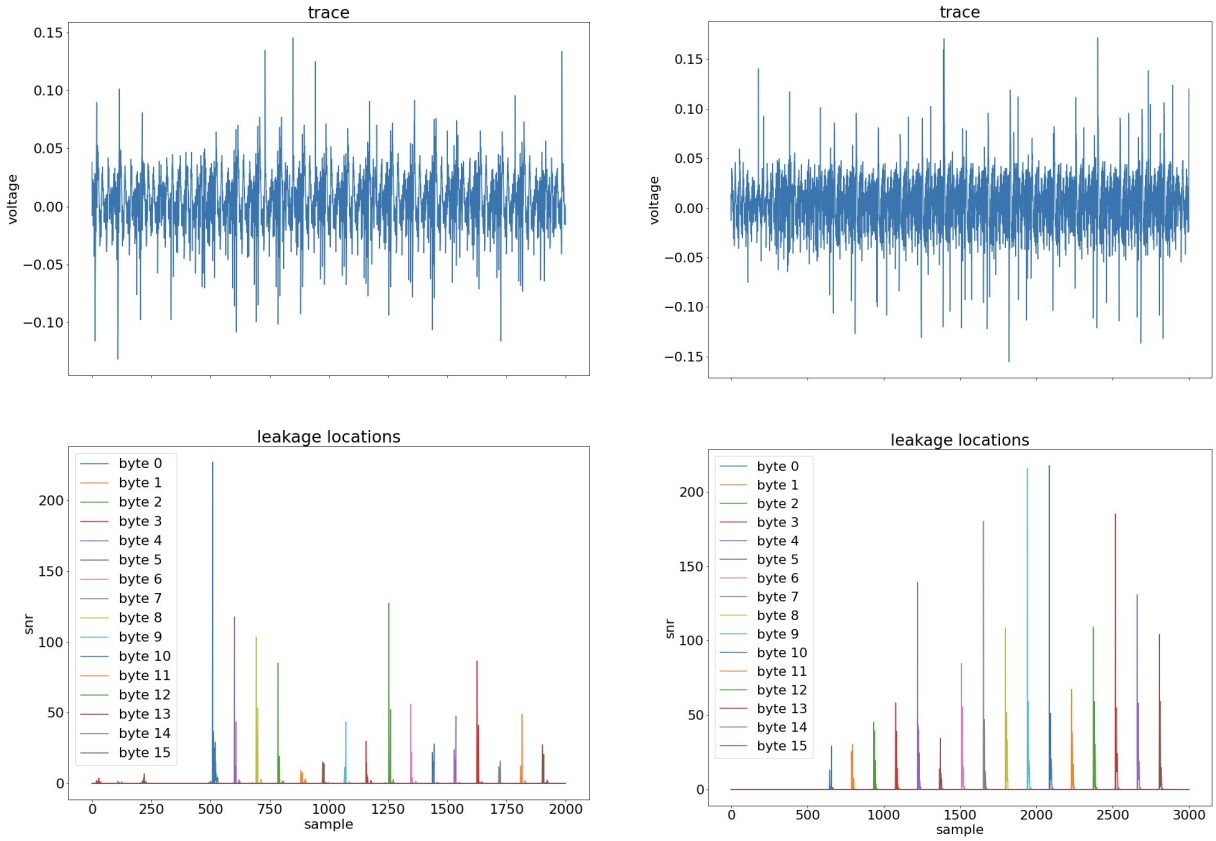
After identifying the 16 sample values of interest based on the peaks found with our leakage model's SNR calculation, we created 16 different training and test sets from our profiling and attack sets, which were examined by 12 different machine learning classifiers. For the first byte's training set, we zoomed in on the sample of interest in each of the 60,000 profiling traces (or in some cases several samples centered around the peak SNR index) and used that voltage value or group of voltage values as the predictor in the training set for each observation, with each predictor labeled with the correct S-box output value. In the corresponding test set, the predictor was also the voltage value(s) associated with the first S-box output byte in each of the 10,000 attack traces. While we labeled each of them to check our classifiers' prediction accuracy for evaluation purposes, in reality, we would assume an attack set would not be labeled as we wouldn't know the key ahead of time to check our test predictions against. For the masked software implementation of AES-128, there are only two key differences in the process described above. Firstly, 3,000 samples were collected per trace to capture all S-box substitutions in the first round. Secondly, an additional mask value was XORed with the S-box output bytes to disrupt any first-order relationship between the measured voltage value associated with the S-box operation and the values trained by our machine learning classifiers.

*2) **XC7A100T FPGA**:* When evaluating the FPGAs, we obtained an equivalent number of traces for both the profiling and attack sets, matching the number used for the MCUs. However, due to the faster execution speed of the hardware implementation of AES-128 on the FPGAs, we only captured 100 samples per trace. This was sufficient to capture the entirety of full encryption. Despite our initial efforts to focus on the S-box operation in the first round, similar to what we did for the MCUs, we encountered greater difficulty in establishing a correlation between the measured voltage and the S-box output values when targeting the FPGAs. Although we achieved partial key recovery in certain cases, our attempts to retrieve complete keys using our machine learning classifiers through this approach were unsuccessful. Moreover, unlike the software implementations on the MCUs, the S-box outputs were not stored in memory on the FPGAs, leading to less detail in each collected power trace in terms of identifiable patterns and certain operations being performed. Therefore, we ended up using a different leakage model for the FPGAs, targeting the end of the AES-128 encryption process instead. Specifically, our chosen model attempted to find a correlation between the voltage measurements and the difference between the input and output of the last round, where "difference" referred to the result of the XOR between the two states.

C. Machine Learning Classifiers

We employed 8 distinct machine learning classification algorithms along with a side-channel leakage model to classify power traces in this study.

Convolutional Neural Network: We employed the Convolutional Neural Network (ConvNet/CNN) algorithm, a deep learning technique capable of recognizing and assigning significance to various objects or features within an input image using learnable weights and biases. For our implementation, we utilized Keras/TensorFlow's built-in CNN. Our model was comprised of a 1D convolutional layer featuring 64 filters, an 11-unit kernel size, a Rectified Linear Unit (ReLU) activation function, and 'same' padding; a 1D average pooling layer with a 2-unit pool size and 2-unit stride; a flatten layer; a dense layer with 256 units and a ReLU activation; a batch normalization; a 0.2 dropout rate; and an output layer with 256 units and a softmax activation. We compiled the model using Adam optimization with a learning rate of 0.001 and a categorical cross-entropy loss function, trained with a batch size of 100 across 5 epochs. **Gaussian Naive Bayes:** We employed the Gaussian Naive Bayes (GNB) classifier, which is a supervised learning algorithm based on the assumption of conditional independence between every pair of features given the value of the class variable, using a Gaussian likelihood function. To implement the GNB classifier, we used the implementation provided by sklearn with a variance smoothing value of 1e-09. **K-Nearest Neighbors:** Neighbors-based classification is a non-generalizing learning method that relies on instance-based learning. It does not try to build a general model but rather stores instances of training data. The classification is based on the majority vote of the nearest neighbors of each point: the class with the highest number of representatives among the nearest neighbors is assigned to a query point. For our KNN classifier, we utilized sklearn's implementation with 1024 neighbors, uniform' weights, and the minkowski' distance metric. **Linear Discriminant Analysis:** We employed



(a) (b)

Fig. 4. (a) Result of a leakage model SNR calculation for revealing samples associated with the processing of the 16 S-box output bytes on the (a) unmasked software AES-128 profiling set and (b) masked software AES-128 profiling set.

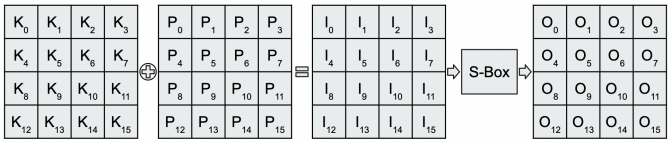


Fig. 5. Flow of the "add round key" and "substitute bytes" operations in the first round of unmasked AES-128.

a linear discriminant analysis (LDA) classifier, which generates a linear decision boundary by fitting class conditional densities to the data and applying Bayes' rule. In this model, a Gaussian density is fitted to each class, with the assumption that all classes share the same covariance matrix. We used the 'svd' solver implementation provided by sklearn. **Logistic Regression:** Logistic regression is a type of linear model used for classification tasks instead of regression. It can also be referred to as logit regression, maximum-entropy classification, or the log-linear classifier. In this model, a logistic function is used to model the probabilities of the potential outcomes of a single trial. We utilized sklearn's implementation of logistic regression (LR) with an l2' penalty, lbfgs' solver, and an inverse regularization strength of 4096. **Long Short-Term Memory Network:** Our LSTM model used Keras/TensorFlow's implementation. The model comprised of two initial LSTM layers with 10 units each that were set to return sequences. Each layer was followed by a dropout of 0.2. We added one more LSTM layer with 10 units that were not set to return sequences, followed by another dropout of 0.2, and an output dense layer of 256 units with a softmax activation. We compiled the model using Adam optimization with a learning rate of 0.001 and a categorical cross-entropy loss function. The training was performed with a batch size of 100 in 10 epochs. LSTM networks are a type of recurrent neural network that can learn order dependence in sequence prediction problems. **Multilayer Perceptron Network:** A multilayer perceptron (MLP) is an artificial neural network (ANN) that consists of three or more layers of nodes: an input layer, a hidden layer, and an output layer. Each node, except the input nodes, is a neuron that employs a nonlinear activation function. We utilized Keras/TensorFlow's implementation for our MLP, which began with a dense layer of 256 units and ReLU activation. It was followed by batch normalization and dropout of 0.2. The output layer also had 256 units with a softmax activation. We compiled the model using Adam optimization with a learning rate of 0.001 and a categorical cross-entropy loss function. We trained the model with a batch size of 100 in 5 epochs. **Support Vector Machine:** Our SVM classifier

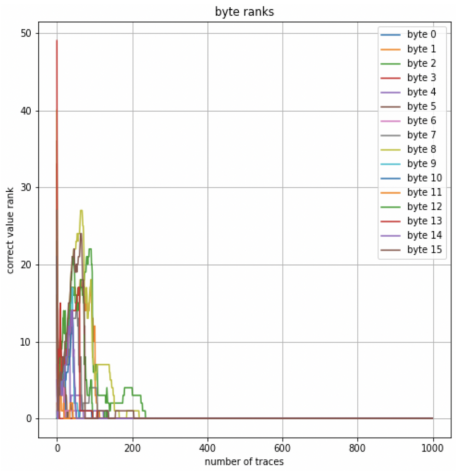


Fig. 6. Byte rank plot showing a successful attack, as all 16 bytes converge to a rank of 0 before running out of attack traces.

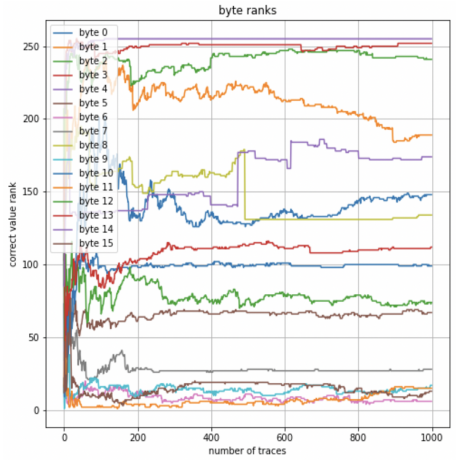


Fig. 7. Byte rank plot showing a failed attack, as all 16 bytes do not converge to a rank of 0 before running out of attack traces.

was implemented using sklearn. The SVM algorithm aims to identify a hyperplane in an N-dimensional space (where N is the number of features) that can effectively separate data points into distinct classes. We used sklearn's implementation with a regularization strength of 1, an 'rbf' kernel, a degree of 3, and default parameters for all other settings.

D. Rank Metric

To assess the effectiveness of our machine learning classifier in predicting the desired key, we employed the rank or byte rank metric. This metric served as a measure of success in our evaluation process. Like our evaluation of MCUs, assigning the correct S-box output bytes for each encryption to the training and test sets for the 16 bytes did not lead to reliable predictions based on a single trace. However, we observed that incorporating predictions from multiple traces enhanced our ability to determine the correct key.

To ascertain the correct key byte value associated with each potential S-box value for a byte, we utilized machine learning classifiers that produced a probability array of 256 values. We iterated through the probability arrays of all traces and calculated the probability for each of the 256 possible key byte values corresponding to the S-box value, using the corresponding plaintext value for each trace. These probabilities were then accumulated in a sums array, which represented the probabilities for each key byte value. The primary goal as an attacker was to aggregate the probabilities across all traces and identify the key byte with the highest cumulative value. Additionally, throughout this process, we maintained the rank of the correct value relative to the 255 incorrect values. We visualized the changes in rank over time and the number of trace predictions required to achieve the top rank or highest likelihood.We designated the highest rank as 0 and the lowest rank as 255. Figures 6 and 7 illustrate example results of rank calculations for a successful and failed attack, presented in a plot format. We employed the rank metric in a similar manner for the FPGAs, where the labels for both the profiling and test sets were the last round state difference values.

IV. Experimental Results

A. Software based AES-128 on STM32F415

During the evaluation of the unmasked software implementation of AES-128 on the MCUs, we made an interesting observation. In all four attack scenarios, which involved two targets and two keys, we discovered that all eight of our selected machine learning classifiers, when combined with a first-round S-box leakage model, successfully recovered the complete encryption key. Remarkably, this achievement was attained with minimal or no tuning, indicating that this specific combination of device and AES variant is highly vulnerable to this type of attack. Our investigation of the masked software implementation yielded intriguing findings. In two out of the four attack scenarios, the machine learning classifiers, when coupled with the same leakage model, successfully recovered the secret key. However, in the remaining two scenarios that involved a different attack key from the profiling key, the classifiers were unable to predict the key. Although we occasionally managed to retrieve partial keys, we never achieved complete recovery of all 16 bytes. Notably, these two scenarios, which involved attacking a different target with a different key or the same target with a different key, are considered to be the most realistic scenarios. Consequently, we concluded that masking serves as an effective countermeasure against machine learning-based side-channel attacks (MLSCA).

B. Hardware based AES-128 on XC7A100T

we were able to accurately recovering the complete hardware based AES-128 XC7A100T FPGA using machine learning classifiers, with one notable exception. We found that the four bytes comprising the first column of the AES state array displayed a higher level of predictability compared to the remaining bytes, although the exact reason for this phenomenon remains elusive. However, we effectively resolved this issue by utilizing a single model that accurately predicted the corresponding byte, thereby allowing us to predict

Classifier Rank Comparison for the STM32F415 and XC7A100T												
Machine Learning Classifier	Unmasked Software AES-128 (MCU) Average Number of Attack Traces to Rank 0				Masked Software AES-128 (MCU) Average Number of Attack Traces to Rank 0				Unmasked Hardware AES-128 (FPGA) Average Number of Attack Traces to Rank 0			
	SD & SK	SD & DK	DD & SK	DD & DK	SD & SK	SD & DK	DD & SK	DD & DK	SD & SK	SD & DK	DD & SK	DD & DK
Convolutional Neural Network	83	83	7	8	13	N/A	17	N/A	304	573	414	585
Gaussian Naive Bayes	88	129	15	13	20	N/A	37	N/A	449	583	527	712
K-Nearest Neighbors	8	7	3	4	5	N/A	7	N/A	1484	1849	1161	3147
Linear Discriminant Analysis	118	136	6	4	8	N/A	19	N/A	251	508	312	524
Logistic Regression	109	98	6	5	7	N/A	15	N/A	249	505	312	522
Long Short-Term Memory Network	15	13	6	5	7	N/A	11	N/A	283	448	293	472
Multilayer Perceptron Network	110	133	12	8	11	N/A	26	N/A	574	996	594	1695
Support Vector Machine	7	9	5	4	4	N/A	16	N/A	314	612	316	672

Fig. 8. For the MCUs, the average byte rank 0 convergence was observed when using classifiers trained with 20,000 profiling traces and tested with 1,000 attack traces. Similarly, for the FPGAs, the convergence occurred with classifiers trained on 60,000 profiling traces and tested with 10,000 attack traces. The abbreviations SD, SK, DD, and DK represent the following scenarios: SD (same device/target profiled with same key), SK (same key profiled with same device/target), DD (different device/target profiled with same key), and DK (different key profiled with different device/target).

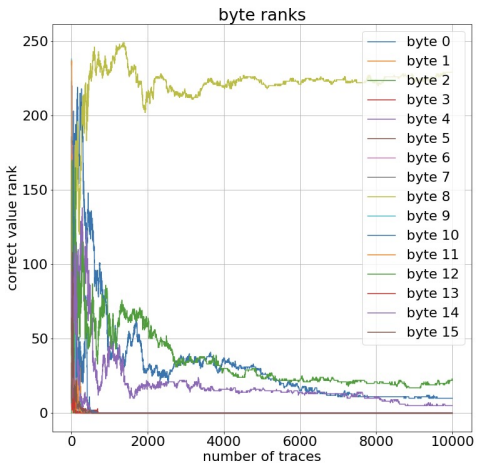


Fig. 9. Partial key recovery for the XC7A100T FPGA in a differing profiling and attack key scenario, with separate machine learning classifiers used for each byte.

the remaining 15 bytes as well. This finding suggested that training separate models for each byte was unnecessary and potentially impeded the attack process. Although utilizing a single model necessitated an increased number of traces during the prediction stage for the 15 untrained bytes, it ultimately resulted in a faster evaluation process.

C. Classifier Comparison

The average number of attack traces required per byte before reaching a rank of 0 for all 8 machine learning classifiers in various scenarios is shown in the table in Figure 8. It is essential to emphasize that the performance of a classifier can vary depending on the specific circumstances. Therefore, these tables should not be considered definitive in determining the optimal classifier for machine learning-based side-channel attacks (MLSCA). When evaluating the STM32F415, we observed that the long short-term memory, K-nearest neighbors, and support vector machine classifiers tended to require the fewest number of traces to accurately rank the correct key byte values as the most probable. In the case of the XC7A100T, the convolutional neural network, linear discriminant analysis, logistic regression, long short-term memory, and support vector machine classifiers exhibited similar performance levels. However, the support-vector machine classifier took considerably longer to run than the others, so while it produced low-rank convergence, it was not as time-efficient. Additionally, the ensemble/tree-based machine learning algorithms failed to predict keys for the FPGAs, as did the restricted Boltzmann machine classifier.

For the successful attacks on masked software AES-128 on the STM32F415, the average rank 0 convergence was lower when the attacked target was not the same as what was profiled, which was unexpected. This could be attributed to process variation, as each device behaves differently and may generate noise that affects the collected power traces. Investigating why some devices are more vulnerable to MLSCA even with the same scenario could be a topic for future work.

D. Comparison with Existing Works

Table I compares our work to several prior studies [19], [9], [11], [14], [8], [15] in terms of experimental scenarios and key rank results. It is noteworthy that most previous studies only used the same device and key for profiling and testing, whereas our work demonstrated the feasibility of retrieving keys under four scenarios. Additionally, most studies in the existing literature only considered masked software, unmasked software, or unmasked hardware, while we analyzed all scenarios, including masked/unmasked software and unmasked hardware, for both FPGAs and microcontrollers.

V. Conclusions and Future Works

This paper presents our research on evaluating the resistance of unmasked and masked software AES-128 implemented on STM32F415 MCUs and unmasked hardware AES-128 on XC7A100T FPGAs against machine learning-based side-channel attacks. We used a side-channel leakage model targeting either the first round S-box operation or the last round state difference in combination with 8 machine learning classifiers. Our evaluation revealed that unmasked software AES-128 on

Work	Hardware	AES-128 Implementation	Scenario	ML Algorithm(s)	Number Traces
This work	STM32F415 MCU and XC7A100T FPGA	Unmasked Software Masked Software Unmasked Hardware	SD & SK SD & DK DD & SK DD & DK	CNN, GNB, KNN LR, LDA, LSTM MLP, SVM	16, 18, 5 10, 8, 20 692, 792
[37]	ATXmega	Unmasked Software	SD & DK DD & DK	CNN	160, 400
[20]	ASIC	Unmasked Hardware	SD & DK	CNN	1300
[22]	Virtex-5 FPGA	Unmasked Hardware and Masked Software	SD & DK	AE (autoencoder) CNN, MLP, LSTM	200, 500
[25]	Virtex-5 FPGA	Unmasked Hardware	SD & DK	MLP, CNN	2500
[18]	Atmel AVR ASCAD database	Unmasked Hardware and Masked Software	SD & DK SD & SK	CNN	10, 25000 3, 300
[26]	Artix-7 FGA	Unmasked Hardware	SD & DK	LSTM-AE	3700

TABLE I
RELATED WORKS COMPARISON (SD = SAME DEVICE/TARGET THAT WAS PROFILED, SK = SAME KEY THAT WAS PROFILED, DD = DIFFERENT DEVICE/TARGET THAN WHAT WAS PROFILED, DK = DIFFERENT KEY THAN WHAT WAS PROFILED).

the MCUs was highly susceptible to attack, but masking can protect against it when the attacked key is different than the profiled one. We also found that while all 8 machine learning classifiers were capable of predicting the secret key when at least one of them was successful, some required more power consumption traces than others to do so. Further investigation could focus on optimizing the machine learning classifiers, exploring alternate forms of attack on the power consumption data, and better understanding the factors affecting rank 0 convergence for unmasked AES-128 when attacking a second MCU that was not profiled as compared to attacking the same MCU that was profiled.

REFERENCES

[1] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Study of deep learning techniques for side-channel analysis and introduction to ascad database," *ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am*, vol. 22, p. 2018, 2018.

[2] M. Carbone, V. Conin, M.-A. Cornelie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, and A. Venelli, "Deep learning to evaluate secure rsa implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 132–161, 2019.

[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 13–28.

[4] N. Chawla, A. Singh, M. Kar, and S. Mukhopadhyay, "Application inference using machine learning based side channel analysis," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[5] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-deepsca: Cross-device deep learning side channel attack," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[6] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury, "Practical approaches toward deep-learning-based cross-device power side-channel attack," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2720–2733, 2019.

[7] S. Jin, S. Kim, H. Kim, and S. Hong, "Recent advances in deep learning-based side-channel analysis," *ETRI Journal*, vol. 42, no. 2, pp. 292–304, 2020.

[8] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 148–179, 2019.

[9] T. Kubota, K. Yoshida, M. Shiozaki, and T. Fujino, "Deep learning side-channel attack against hardware implementations of aes," *Microprocessors and Microsystems*, p. 103383, 2020.

[10] H. Maghrebi, "Deep learning based side channel attacks in practice." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 578, 2019.

[11] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016, pp. 3–26.

[12] N. Mukhtar, M. A. Mehrabi, Y. Kong, and A. Anjum, "Machine-learning-based side-channel evaluation of elliptic-curve cryptographic fpga processor," *Applied Sciences*, vol. 9, no. 1, p. 64, 2019.

[13] C. O'flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

[14] S. Picek, I. P. Samiotis, J. Kim, A. Heuser, S. Bhasin, and A. Legay, "On the performance of convolutional neural networks for side-channel analysis," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2018, pp. 157–176.

[15] K. Ramezanpour, P. Ampadu, and W. Diehl, "Scaul: Power side-channel analysis with unsupervised learning," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1626–1638, 2020.

[16] S. Song, K. Chen, and Y. Zhang, "Overview of side channel cipher analysis based on deep learning," in *Journal of Physics: Conference Series*, vol. 1213, no. 2. IOP Publishing, 2019, p. 022013.

[17] B. Sönmez, A. A. Sarıkaya, and Ş. Bahtiyar, "Machine learning based side channel selection for time-driven cache attacks on aes," in *2019 4th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2019, pp. 1–5.

[18] B. Timon, "Non-profiled deep learning-based side-channel attacks with sensitivity analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 107–131, 2019.

[19] H. Wang, "Side-channel analysis of aes based on deep learning," 2019.

[20] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 393–406.