Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 13, Number 1, pages 29–37, January 2024

Empirical Hardness of AES Cipher

1st Kanat Alimanov
dept. of Computer Science
Nazarbayev University
Astana, Japan
kanat.alimanov@alumni.nu.edu.kz

2nd Martin Lukac, Shinobu Nagayama
dept. of Computer Networks and Architecture
Hiroshima City University
Hiroshima, Japan
malu@hiroshima-cu.ac.jp,
s_naga@hiroshima-cu.ac.jp

***Abstract*—The AES symmetric cipher is widely used as a standard encryption method in various network protocols. Although it has proven resistant to most direct attacks, it hasn't been extensively studied from the perspective of modern neurocryptanalysis and big data. Therefore, this paper aims to comprehensively analyze the components of the AES protocol, empirically determine their learnability, and provide empirical results that demonstrate the hardness of the cipher. Our analysis involves evaluating the ability of various models to learn the different components of the AES cryptosystem, as well as their combinations. Furthermore, we examine the overall ability of these models to recover a network that estimates the operation of adding the secret key to the input data. Through our research, we show that AES is indeed resistant to machine learning attacks. However, under certain configurations of the data and the AES cipher, it is possible to recover the decrypting network.**

***Index Terms*—AES, Machine Learning, Neural Networks**

I. Introduction

Symmetric cryptography is a vital tool used to secure data and facilitate secure communication in modern networks. Numerous symmetric cryptographic primitives exist, such as AES, IDEA, DES, and Blowfish, among others [1]. On one hand, symmetric cryptography is designed to offer high throughput, and on the other hand, it is desired for the cipher to be scalable, accommodating an increased number of inputs or complexity. AES serves as an example of such a cipher, as it is fast and based on simple discrete mathematical principles, including XORing the signal with the key, reversible substitution, and reversible modulo diffusion operations [2].

AES is a block cipher that exhibits ideal scalability by providing various modes to handle increasing input sizes. Additionally, the encryption strength of AES increases with more iterations of the AES round.

Because of its high usage and standard availability various attacks have been applied to AES. The first succesful atack was a side channel attack [3]. Another attack based on key relation allows in theory to fully break a ten round AES cryptographic system [4]. Standard approaches such as differential cryptanalysis or brute force have been all shown to be only partially successful however the authors of [5] have shown a method based on the man-in-the-middle attack that can recover a whole encryption key.

Though direct attacks have been demonstrated as possible, they currently remain computationally infeasible. Theoretical analyses assessing the security of random S-boxes and the complexity of AES rounds have shown resistance to various attacks [6], [7].

Recently, with convolutional and deep neural networks being applied to real-world problems, several attempts have investigated the cryptanalysis of AES and related block ciphers using machine learning approaches. While neural approaches have succeeded in tackling simplified or older ciphers like the Caesar cipher or DES, results for AES have been predominantly negative [8]–[11]. This discrepancy occurs despite arithmetic operations being proven to be learnable to some extent [12], [13].

To comprehend the complexities of learning AES, we conducted a thorough analysis of the cipher and examined existing theoretical results on the hardness of AES in relation to machine learning paradigms. Specifically, we investigated the potential learnability of AES or its components, the necessary conditions for such learning, and the overall difficulty of AES for machine learning-based deep neural models. The primary objective of this paper is to train a set of neural models that can decipher an AES-encrypted message without directly recovering the secret key.

The main findings of our research are as follows: Through extensive experimental evaluation, we demonstrate that AES remains secure against deep neural models used in machine learning. Furthermore, we show that it is possible to recover the encryption/decryption network for small portions of the AES system. Finally, we identify the vanishing gradient, accelerated by the diffusion operators in the AES cipher, as the primary obstacle in training such deciphering networks.

II. Previous Work

The hardness of AES has undergone extensive mathematical analysis, and multiple studies indicate its resistance against machine learning approaches. However, it's important to acknowledge that these mathematical formulations often come with strict bounds and assumptions to enable rigorous theorem proving. For example, in a recent publication [7], the authors demonstrated that given a random S-box, AES remains secure against deciphering after four blocks. Similarly, another study [6] adopted a similar approach and showed that the output of the AES cipher closely resembles a uniform distribution. Consequently, this finding makes it extremely difficult to learn AES, as most machine learning methods have demonstrated an inability to learn from random data.

From a learning perspective, various authors have made attempts to break AES using machine learning and neural network-based methods. In a study by the authors of [8], they successfully demonstrated the ability to learn the Caesar cipher and the DES cipher through simple end-to-end learning. Another study [9] focused on learning the FEISTEL cipher, although the approach was not extensively evaluated with real-world ciphers. Generally, the research on AES deciphering using neurocryptanalysis has been primarily limited to reduced problems or ciphers with reduced AES strength [11].

III. Background

The AES cipher is a symmetric encryption algorithm commonly employed for high-throughput data encryption. It operates on blocks of 128 bits of input data. The input is structured as a four-by-four matrix, where each entry represents a byte. The AES cipher performs a series of discrete steps on this two-dimensional array of bits.

The first step involves performing an XOR operation between the secret key and the input message. This step is referred to as "AddRoundKey" because the key used for the XOR operation is derived from the initial secret key using the round index of the AES algorithm. We can represent this mapping mathematically as $\mathcal{E} : 2^{128} \to 2^{128}$. Although this mapping can be reduced in size due to the bitwise XOR operation, it remains a one-to-one mapping.

The second operation in the AES cipher is the S-box substitution. It involves the use of a reversible mapping denoted as $\mathcal{S} : 2^8 \to 2^8$, which operates on GF(2) (Galois Field). The S-box is a well-known component that functions as a standalone substitution cipher and is also utilized in various cryptographic methods. In the context of the AES cipher, the S-box serves as a byte-wise diffusion operator, effectively mixing the bits within individual bytes.

The third operation in an AES round is a reversible transformation known as the ShiftRows step. It involves shifting the bytes in the 2D byte ordering to the left by a certain number of positions based on their row index. Specifically, the first row (index 0) remains unchanged, the second row is shifted one position to the left, the third row is shifted two positions to the left, and the fourth row is shifted three positions to the left. This transformation can be represented as $\mathcal{B} : 2^{128} \to 2^{128}$.

The last operator in an AES round is the MixColumns operation, which is a reversible transformation based on Galois field arithmetic. It operates column-wise on four bytes at a time, performing a mapping denoted as $\mathcal{M} : 2^{32} \to 2^{32}$. This operation is also reversible and maintains a one-to-one mapping between the input and output.

Encryption in AES is achieved by iterating over the four blocks of data in a specific order. Each iteration over the four operations is called a round and the number of iterations differs based on the AES variant: ten for AES128, twelve for AES192, and thirteen for AES256.

Apart from the number of iterations, there are other differences between these modes. One distinction is the length of the encryption key used, with AES128 using a 128-bit key, AES192 using a 192-bit key, and AES256 using a 256-bit key. Additionally, the key is transformed into the round key more times in AES192 and AES256 compared to AES128.

IV. Proposed Method

An artificial neural network (ANN) is a class of algorithms that utilize a distributed computing approach. It consists of multiple layers, denoted as $\mathbb{L} = l_1, \ldots, l_m$, where each layer contains a set of artificial neurons. These neurons perform the summation of weighted input signals and apply a function to the summed inputs to generate an output. The layers of the ANN are connected by weights, represented by a weight matrix $\mathbb{W}_{kl}$ that connects layers l_k and l_l. These weights contain the information about the function being approximated by the network.

This description represents a type of ANN commonly referred to as a feed-forward network because it propagates inputs from one side through the layers, and producing outputs at the other side, output terminals. The ANN uses learning to approximate a desired function, typically specified by a dataset $\mathbb{D} = \{(i_1, t_1), \ldots, (i_k, t_k)\}$ consisting of k input-output pairs.

The specific type of feed-forward ANN that we are focusing on here employs backpropagation for learning. During back-propagation learning, each input sample, denoted as i, from the training dataset $\mathbb{D}$ is fed into the ANN. This allows the information to propagate forward from the input layers to the output layers. The output of the network, represented as o_i, is then compared to the corresponding output pair t_i from the dataset to compute an error E. Subsequently, this error is propagated backward through the network, updating the weights in the process. These weight updates aim to refine the network's learned knowledge.

In this work, neural models are applied to learn various levels of the AES cipher. Let's represent a single round of AES as $\mathcal{A} : I \to O$, where $\mathcal{A} = \mathcal{E} \circ \mathcal{B} \circ \mathcal{S} \circ \mathcal{M}$. We study the AES from three different mappings. We start with the learning of the full AES cipher denoted as $\mathcal{A}^j$, where j indicates the number of rounds in the AES algorithm. Then we proceed to learn and analyse individual rounds. Finally the individual components of each round of the AES cipher. The decomposition and various points of analysis are illustrated in Figure 1.

The simplest AES implementation contains 10 rounds for 128-bit key, 12 rounds for 192-bit key and 14 rounds for 256-bit key.

In this work we are focusing on AES with $j = 10$ (ten rounds), using 128-bit key length in the ECB mode. This is the simplest possible mode and serves as baseline for the AES difficulty evaluation.

For learning the decryption using the neural models we created two types of datasets. The first one will be referred to as $\mathbb{D}_r$ and represents a dataset created randomly on the bit level. A second dataset referred to as $\mathbb{D}_w$ represents the text downloaded from Wikipedia. The text is not topic specific but is used as is as a comparison to the randomly generated data.

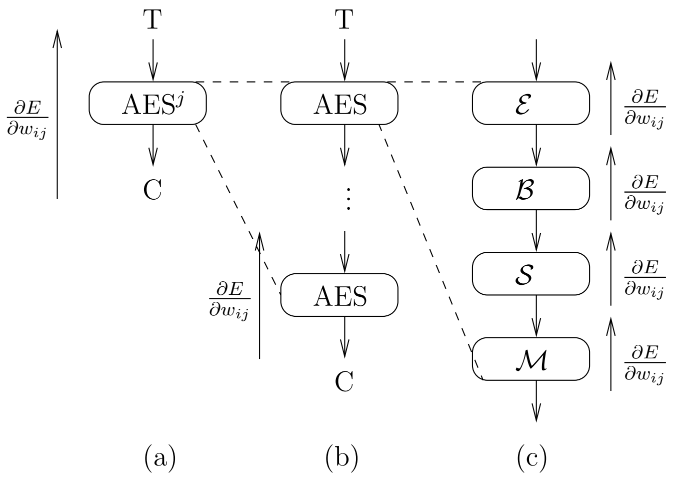


Fig. 1: The various points of analysis of AES

Each dataset is split into two parts, a train and a test dataset. The train dataset size varies from 2^{18} to 2^{26} training samples and the test dataset contains 2^{16} testing samples. Each dataset is encrypted using the AES 128 in the ECB mode.

To determine if an ANN has learned the correct function, we utilize a standard accuracy measure defined as $acc(\mathbb{D}) = \frac{\sum_{i=1}^{|\mathbb{D}|} \mathbf{I}(o_i^T == o_i)}{|\mathbb{D}|}$, where $\mathbf{I}$ denotes the indicator function. This accuracy measure calculates the ratio of correctly predicted outputs o_i to the total number of samples in the dataset $|\mathbb{D}|$. However, it is important to note that in some experiments where the learning outcome is negative, the accuracy may not be reported. Instead, the focus may shift towards observing the convergence of the learning process.

V. Experiments and Results

To analyze the AES we performed a set of experiments according to a workflow. The workflow is shown in Figure 2.

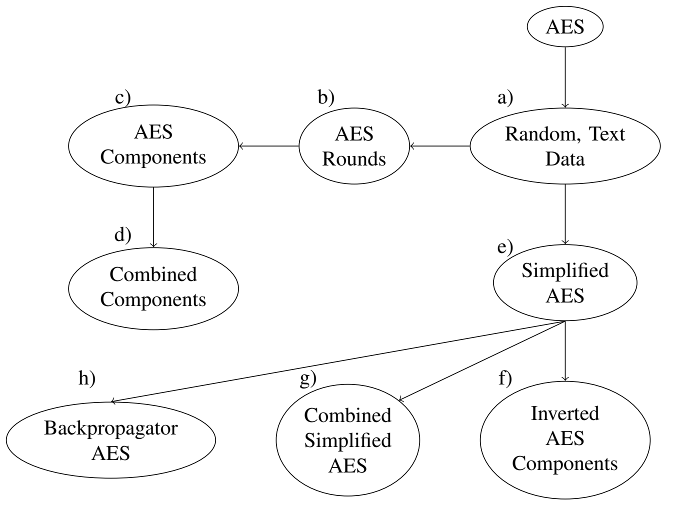


Fig. 2: Chart of the various experiments run as a part of understanding the learning and learning difficulty of the AES cipher.

At first we analyze the AES as a block algorithm (Figure 2a)) by both learning from random and from text data, then we look in details into the individual rounds of AES (Figure 2b)) and experiment with various configurations of the individual AES algorithm components (Figure 2c) and 2d)). While the studying the individual rounds and its components, we also investigated a set of simplified AES algorithms to determine what are the boundaries of learning the AES. Therefore we investigate various simplified versions of the AES rounds (Figure 2(e)) and build various alternative models such as Inverted components of AES rounds or AES backpropgators (Figure 2f) and 2h)).

Recall that our target is to learn a model for various components or even the whole AES: as such we are not looking to recover the key but simply to substitute the AES components by ANN models learned from input-output samples.

A. Learning the Cipher or Learning the Mapping

The first experiment conducted aimed to verify and determine the extent to which the AES cipher can be learned independently. Specifically, the focus was on learning the mapping $\mathcal{M}^{-1} : C \to T$ (as shown in Figure 1(a)) in a text-independent manner. To achieve this, we evaluated two different models in order to identify the most cost-effective approach that potentially provides advantages in the learning process.

The first model is a small convolutional neural network with three convolutional layers, each having 64 convolutional input filters of size 8 and three fully connected layers generating the output. The second model is a fully connected three layered Multi-Layer Perceptron with two hidden layers each with 1000 neurons.

The data prepared was based on the $\mathbb{D}_w$. In order to force the neural network into a data independent learning the initial dataset $\mathbb{D}_w$ was either configured to a fixed size and denoted by $\mathbb{D}_w$ or it was transformed into $\mathbb{D}_w^\dagger$, a dataset that never repeats any sample. This means that when using the dataset $\mathbb{D}_w^\dagger$ the network would never see the same data twice during training. The purpose of this experiment was to determine how much of learning and how much of memorising a network is doing while learning the AES cipher.

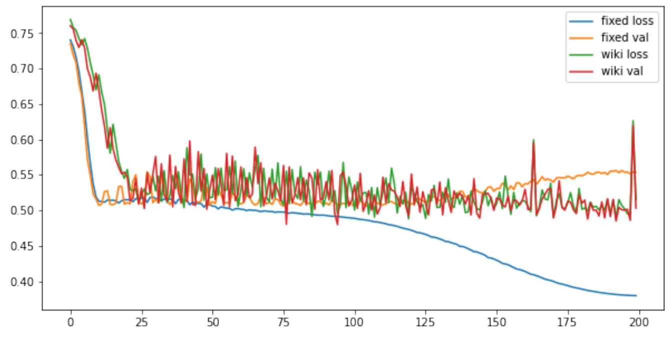


Fig. 3: The learning loss during training a ANN on growing wiki data and fixed wiki data.

Figure 3 shows the results of training a MLP on $\mathcal{D}_w$ and $\mathcal{D}^{\dagger}_w$. Note that both the loss and the evaluation accuracy remain flat and without convergence. The loss and accuracy for fixed amount of data shows some converging trends but the accuracy of classification remains under 0.6. These results are similar when learning the AES with purely random data from the $\mathbb{D}_r$ dataset.

The results presented in this section clearly demonstrate that none of the evaluated networks, regardless of the configuration, were able to effectively learn the AES. The observed outcomes can be classified into two categories: either the model converged to complete memorization of the training dataset, or the model failed to converge altogether.

B. Learning AES by Rounds

As the AES cipher could not be learned in its entirety, the analysis shifted towards learning individual rounds. For each round within AES128, a new dataset, denoted as $\mathbb{D}^i_w$, was generated from the original dataset $\mathbb{D}_w$. The index i corresponds to the round of AES128 for which the dataset was created. For example, $\mathbb{D}^0_w$ contains samples with inputs corresponding to the outputs of the first round of the AES cipher and the plain text. However, it is noteworthy that the learning in this experiment did not converge regardless of the number of rounds used.

C. Breaking AES Rounds to components

Considering the lack of positive results in learning individual rounds independently, the next phase involved analyzing the individual components of the AES rounds. The initial step was to determine the learnability of each of the mappings: $\mathcal{A}$, $\mathcal{B}$, $\mathcal{S}$, and $\mathcal{M}$. From a numerical standpoint, all these mappings exhibit a one-to-one relationship, and there are no apparent reasons why an ANN should not be capable of learning these components of the AES rounds.

Before proceeding we looked at certain properties of each of the four mappings. At first we looked at the cycles, that is if any of the operation is its self inverse. For this purpose we determine for each mapping the exponent leading to itself under the following formulation $\mathcal{U} = \mathcal{U}^n$. For $\mathcal{B}$ $n = 138591$, for $\mathcal{S}$ $n = 8$, for $\mathcal{M}$ $n = 4$ and for $\mathcal{E}$ $n = 22$.

The cyclicity of each AES operation is intended to serve as indicator of its complexity in the learning process. We expect that higher exponent n is obtained the more difficult the learning would be. We also expect that the cyclicity of each operation will not be completely sufficient information to estimate the learnability as for instance it has been shown that learning cyclic arithmetic functions is quite difficult [?]. Therefore the cyclicity should at least partially be informative.

1) Add Round Key: The first investigation was to determine the learnability of the $\mathcal{E}$ mapping. Conversely to the results on its cyclicity , $\mathcal{E}$ can be learned from relatively small amount of data to almost a 100% accuracy: an MLP with a single hidden layer with 128 neurons is enough. To determine more precisely its learnability we tested the needed amount of data for training the neural model. The results are shown in Table I.

TABLE I: Results of training a simple fully connected neural network in a dataset size depending manner for learning the mapping $\mathcal{E}$.

Dataset Size	Accuracy
65536	0.9999994039596155
32768	0.9999980131987183
16384	0.9999980131987183
8192	0.9999819201083363
4096	0.9999455616448807
2048	0.9995315122577692
1024	0.9949042520726311
512	0.9497373051345303

2) Shift Rows: Similarly to $\mathcal{E}$ the row shifting $\mathcal{S}$ can be learned using a single 128 neuron hidden layer without the requirements of any special learning settings. Interestingly and as expected, the simplest model of the AES round that contains $\mathcal{E} \circ \mathcal{S}$ can be learned perfectly up to 100% of accuracy as well. The low cyclicity of the mapping $\mathcal{S}$ again seems to be correlated to the high accuracy of learning and to the small amount of data required.

3) S-Box: The sub-bytes also known as the Rijndael S-box is a byte-wise substitution cipher by itself with both inputs and outputs being polynomials over the GF(2) Galois field. Similarly to the shift bytes the S-box acts individually on every byte of the text and transforms it by a value read from a look-up table. The S-box transformation can be seen a a but-wise diffusion operator on a byte-wise domain. The $\mathcal{B}$ is the one component of the AES round with the highest cyclicity.

The S-box was experimentally determined to be learnable using a MLP with a single hidden layer of 1000 neurons. The model is a mapping from $2^8 \to 2^8$ on the outputs only.

4) Mix Columns: The mix-column operation is the one that experimentally proved to be the hardest to learn. This is despite the fact that is has the cyclicity of only $n = 4$. The most interesting obsevration is that while learning the $\mathcal{E}$ or the $\mathcal{B}$ was succesful, the learning of $\mathcal{M}$ did not work well. The specific about mix-columns is that it does not need to be learned for all 128 bits but rather for each column wise transformation independently resulting in a reduced mapping $\mathcal{M} : 2^{32} \to 2^{32}$.

The model used for learning mix columns for the transformation of a single column is a convolutional neural network with the first layer being a 1d convolution containing 64 filters, each of size 9 and a second layer containing 64 filters of kernel size 3. In addition it has one hidden fully connected layer with 128 neurons and the output layer of either 8 or 32 outputs. The first network evaluated was used to learn to output one byte from 32 bits inputs and the second network was to learn the full mapping $\mathcal{M} : 2^{32} \to 2^{32}$. Each of the model was trained and evaluated with the resulting accuracy of 0.9999 for eight bits and 0.9530 for 32 bits.

A more optimized network with a size 8 of the kernels in the first layer resulted in a higher accuracy of up 0.9999. However, and interestingly the $\mathcal{M}^{-1} : O \to I$ could not be learned successfully.

D. Building AES Rounds From Bottom Up

The models for the various components of the AES round being available, the next step was to design a fully neural model of the AES so that the learning of the $\mathcal{E}$ can be evaluated. Because the expected difficulty and the obtained negative results of end-to-end learning of the AES we decided to start evaluating weaker AES rounds in order to understand where the learning is failing. In addition, because both the MixColumns mapping $\mathcal{M}^{-1}$ as well as SubByte mapping $\mathcal{B}^{-1}$ were not directly learnable and the only $\mathcal{M}$ can be learned we decided to determine if we can simply learn the encryption of the AES.

In addition, to understand the problem of the gradient backpropagation we investigate first the general backpropagation of the gradient in a much simpler size AES. For these experiment s the target was to learn the encryption rather than decryption. The reason was to determine if there is a difference between encryption and decryption learning and if the result of backpropagation would have similar shortfalls.

For the purpose of this study we built a fully neural model of the AES round. Each of the modules $\mathcal{M}$, $\mathcal{B}$ and $\mathcal{S}$ are trained ahead of the time and verified for their accuracy. Then only the $\mathcal{E}$ is to be learned directly from the data generated (Figure 1(c)) using backpropagation of the error through the pre-learned models of $\mathcal{B}$, $\mathcal{S}$ and $\mathcal{M}$ to update the model of $\mathcal{E}$.

1) Neural AES Model Verification: As a first step we decided to evaluate if a learned model would work at all. Therefore we combined pre-learned models for each mapping and evaluated its accuracy. The result for a full ten rounds AES128 implemented from neural blocks in the task of encryption and decryption is 0.9997.

2) Full Round Learning: The initial experiment is set up to directly verify the learnability of single round of AES by evaluating a set of composed neural models:

$\mathcal{A}1$: Input$\rightarrow\mathcal{E}\rightarrow\mathcal{B}\rightarrow\mathcal{S}\rightarrow\mathcal{M}\rightarrow\mathcal{E}\rightarrow$Output
$\mathcal{A}2$: Input$\rightarrow\mathcal{E}\rightarrow\mathcal{B}\rightarrow\mathcal{S}\rightarrow\mathcal{M}\rightarrow$Output
$\mathcal{A}3$: Input$\rightarrow\mathcal{E}\rightarrow\mathcal{B}\rightarrow$Output
$\mathcal{A}4$: Input$\rightarrow\mathcal{E}\rightarrow\mathcal{M}\rightarrow$Output
$\mathcal{A}5$: Input$\rightarrow\mathcal{E}\rightarrow\mathcal{S}\rightarrow$Output

The accuracy of the learning for the mappings $\mathcal{A}1$ to $\mathcal{A}4$ did not converge and their evaluation accuracy remained $\approx 50\%$. The only model that was able to learn the target $\mathcal{E}$ mapping was $\mathcal{A}5$ with 0.9999 accuracy.

Each neural model of the components of the AES round has binary inputs and binary outputs. Such setting is however problematic because of the learning: while each component can be learned independently, forcing such a segmentation on the model is however counterproductive.

This can be shown by instead of training a model $\mathcal{A}_3$ one can directly train a model $\mathcal{A}_3'$: Input $\rightarrow$ $\mathcal{E} \circ \mathcal{B}$ $\rightarrow$ Output with $\mathcal{E} \circ \mathcal{B}$ being trained as a single neural model. The result of accuracy when evaluating the learned model $\mathcal{A}_3'$ becomes 0.99.

One of the possible problems is that the training data is purely binary while the backpropagation is in general using floating point numbers. During inference however, each of the pre-trained model requires an output thresholding so that the correct binary outputs are obtained. Without such thresholding in between the learning seems to be possible at least partially.

E. Backpropagation Analysis

At this stage the most important observed problem is the related to the gradient backpropagation. While for $\mathcal{E} \rightarrow \mathcal{S}$ the learning occurs, when using any of the two other diffusion operators in a pre-trained form the learning does not happen.

Therefore in order to understand the learning dynamics we sstart the analysis by constructing a set of models with emphasis on the learning target location. Similarly to previous experiments we used nueral blocks for each of the AES components.

For each trained model, the $\mathcal{B}$ was pre-trained and multiple rounds of AES were used to learn the $\mathcal{E}$. First, models were constructed such that the learnable network block $\mathcal{E}$ is closest to the target output.

The first model $\mathcal{A}_f$ was built using as the following order of components: Text $\rightarrow\mathcal{B}$ $\rightarrow\mathcal{S}$ $\rightarrow\mathcal{E}$ $\rightarrow$Cipher (with $\mathcal{B}$ and $\mathcal{S}$ being pre-trained) resulted in an accuracy of 99.99%. This is an interesting result but the main merit of this shows that as long as the error is directly behind the output of $\mathcal{E}$, the learning is very efficient.

The second evaluated model $\mathcal{A}_{0,f}$ is given by Text $\rightarrow\mathcal{E}_0$ $\rightarrow$($\mathcal{B}$ $\rightarrow\mathcal{S}$ $\rightarrow\mathcal{E}_0$) $\rightarrow$Cipher. The maximum accuracy of learning obtained was 68.66%.

Finally when the last model $\mathcal{A}_f^2$ used was Text $\rightarrow$($\mathcal{B}$ $\rightarrow\mathcal{S}$ $\rightarrow\mathcal{E}_i$) x 2 rounds $\rightarrow$Cipher, the accuracy obtained was 70.73%.

Of course when the $\mathcal{B}$ was not present the learning was much more successful as seen on the following model Text $\rightarrow$($\mathcal{S}$ $\rightarrow\mathcal{E}_i$) x 3 $\rightarrow$Cipher with the accuracy of 99.84%.

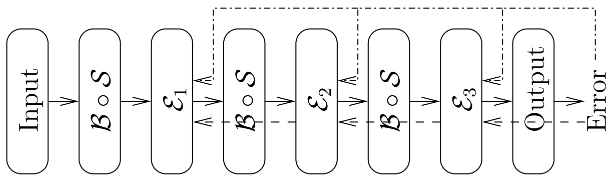
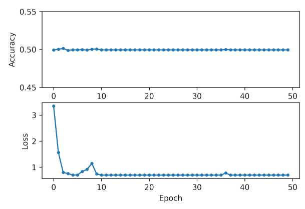
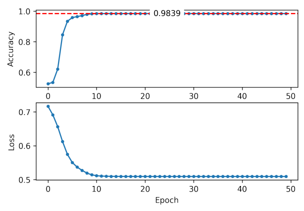

Input
$\mathcal{B} \circ \mathcal{S}$
$\mathcal{E}_1$
$\mathcal{B} \circ \mathcal{S}$
$\mathcal{E}_2$
$\mathcal{B} \circ \mathcal{S}$
$\mathcal{E}_3$
Output
Error

Fig. 4: General schematic of providing each learnable mapping $\mathcal{E}_j$ two feedbacks for learning.

1) Round Key Conditional Independence: Before delving even deeper in the gradient propagation study, we decided to determine if the conditional independence of the round key would help. For this purpose we assume that the individual round keys k_j are conditionally independent on the initial key k. Therefore for two rounds i and j the keys would have the property $(k_i \perp k_j|k)$. While this is a strong assumption and it is expected to be incorrect it allows to provide more data to the backpropagation of the error. Using a key k, being a 128 bit string. Each round key k_j is obtained from k by the scheduler. As a result each learnable network for $\mathcal{E}_j$ receives the error that is backropagated to it as well as the error obtained at the end of the network chain. The schematic of this approach is shown in Figure 4.



(a) $\mathcal{E}$ is the first component in the network



(b) $\mathcal{E}$ is the last component in the network

2) Inversion Models: The backpropagation of the gradient has several possible shortcomings. One of the standard problem is the vanishing gradient [14]. The vanishing gradient is appears as a result of successive reduction of gradient magnitude as a result of many derivatives during the backpropagation.

In order to determine if the vanishing gradient is the problem of the backpropagation, we implemented a set of inverse network models. That is, instead of learning the target mapping, we learned the inverse mapping. This implies that when the backpropagation would occur across the neural blocks, the gradient would be feed-forwarded as is rather than backpropagated.

The usual model for neural computation and network error backpropagation requires that the network itself is able to generate floating point values on its inputs, when backpropagating values not intended for its modification. However most of the pre-trained models used such as ANN for $\mathcal{B}$ are not well prepared for such a task as they are trained for strictly binary inputs and outputs. Therefore in order to deal with this problem we decided to implement the inverse model, with the sole purpose of propagating backwards the gradients.

Let a backpropagated error be of the form $\frac{\partial E}{\partial w_{ij}}$ on the output of a neural network model simulating the $\mathcal{E} \circ \mathcal{B}$ mappings. In order to get the gradient to the outputs of $\mathcal{E}$ model it must be backpropagated through the fixed model $\mathcal{B}$. Instead of backpropagating the gradients, we build alternate neural network performing the mapping shown in eq. 1

$$\mathcal{B}_{grad} : \frac{\partial E}{\partial w_{ij}} \to \mathcal{B}^{-1} \circ \frac{\partial E}{\partial w_{ij}} \tag{1}$$

The evaluated model called $\mathcal{A}_0^{-1}$ is constructed from the following order of components: Text $\to\mathcal{E}_0$ $\to$InverseMC $\to$InverseShift $\to$InverseSB$\to$Cipher. As expected however this model does not learn at all. Variations on learning and changing parameters did not have any effect on the learning result. Therefore we consider that the problem of backpropagation is not due to backpropagation itself but rather due to the nature of the gradient and of the error function.

The first approach was only working again when the $\mathcal{E}$ was the last component in the model. The result of learning can be seen in Figure 5a and Figure 5b.

3) Local Gradients: The failure of conditionally independent gradient backpropagation and of the inverse models implies that the backpropagated information is not correct. Therefore to determine the extent of the gradient loss through the diffusion blocks, we performed a set of experiments where we provide a basic gradient-guide to each $\mathcal{E}$ block. This is performed as follows. Let $\frac{\partial E}{\partial w_{ij}}$ be the gradient computed given the error E obtained on the output of the studied block. For instance, for the model $M_0 : \mathcal{E}_0 \to \mathcal{M}$ error E is computed at the output of the $\mathcal{E}$ block. In addition let E_i be the error computed directly at the i^{th} block $\mathcal{E}_i$. For the model M_0 the error E_0 is computed directly at the output of the $\mathcal{E}_0$ block. Then the gradient computed at the output of $\mathcal{E}_0$ as $\frac{\partial E}{\partial w_{ij}} + \frac{\partial E_0}{\partial w_{ij}}$. The experiments shows that if the local data $\mathbb{D}^i$ would be

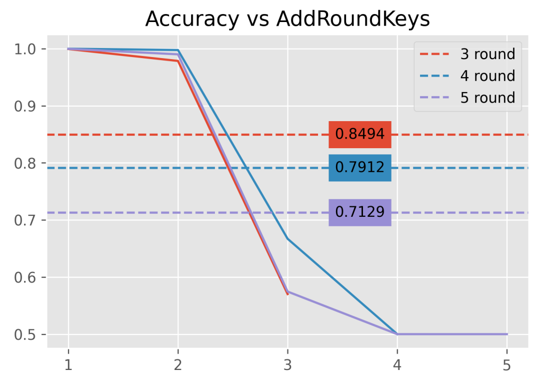


Fig. 5: Learning AES rounds with local gradients.

available the backproagation problem persists and after five rounds the learning remains still problematic such as shown in Figure 4 (dotted dashed line). Figure 5 shows the results of learning AES rounds. Each round is a full AES round $\mathcal{A} : \mathcal{E} \circ \mathcal{S} \circ \mathcal{B} \circ \mathcal{M}$. The learning experiment with the gradient-guide was performed for three, four and five $\mathcal{A}$ rounds. The dotted lines show the maximum accuracy that was achieved by the learning and the full line show the evolution of the loss during learning. As can be seen after five rounds, even with the locally correct result, the learning failed completely with accuracy of 50%.

F. Mixed Columns Backpropagation Analysis

The results of previous experiments confirmed that when learning the AES deciphering or even encryption using the ANNs suffers greatly when propagating the gradients necessary for weights updates across the network $\mathcal{M}$ and $\mathcal{B}$. In addition we also demonstrated that even with the gradient-guide the learning fails after five rounds. This means that indeed the backpropagation is suffering heavily during learning. In addition the learning was most affected when the $\mathcal{M}$ mapping was used in the AES round and therefore we reduced the AES round to a simplest model.

We investigate the $\mathcal{A}_{0M}$ model that contains only $\mathcal{E} \circ \mathcal{M}$ and the $\mathcal{A}_{0M1}$ model containing $\mathcal{E}_0 \circ \mathcal{M} \circ \mathcal{E}_1$. At first we decided to determine the difference between the gradients obtained using the backpropagation through $\mathcal{M}$ and compare them with the gradients calculated directly at the output of the $\mathcal{E}$.

Figure 6 shows the difference between the gradients for the model $\mathcal{A}_{0M}$. The graph shows the averaged difference between

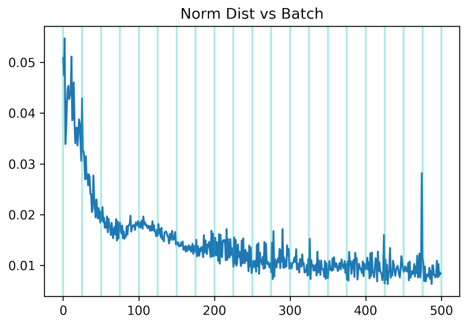
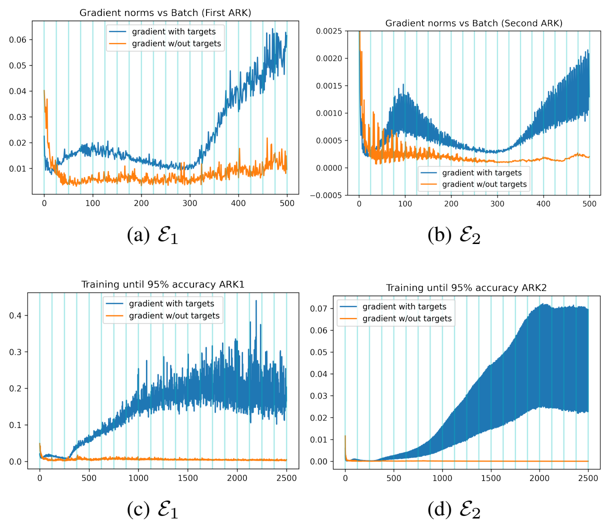


Fig. 6: Difference between gradients $\frac{\partial E_l}{\partial w}$ and $\frac{\partial E}{\partial w}$

$\frac{\partial E_0}{\partial w}$ and $\frac{\partial E}{\partial w}$ for the first 500 learning epochs. As can be seen the difference between the gradients slowly decreases to 0.01 which however given the gradients magnitude is a considerable error. In fact during the learning the difference is anywhere between 50% to 10% of the maximum values of the averaged gradients. This illustrates that the backpropagated gradient not only fails to properly backpropagate the values but is also strongly reduced in magnitude.

Similarly when we compared the gradients from the model $\mathcal{A}_{0M0}$ the difference was visible from the beginning. Figure 7 shows the gradient differences up to epoch 500 Figure 7a and 7b for the first and second network learning the $\mathcal{E}$ mapping.



(a) $\mathcal{E}_1$ (b) $\mathcal{E}_2$ (c) $\mathcal{E}_1$ (d) $\mathcal{E}_2$

Fig. 7: Difference between gradients $\frac{\partial E_l}{\partial w}$ and $\frac{\partial E}{\partial w}$ in an ARK-M-ARK network system for the first ARK network (a) and (c) and the second ARK network in (b) and (d) respectively.

The difference between the gradients magnitude becomes much more visible when we plot the difference until the point where the model using the local gradients converges to a validation accuracy of 95%. This is shown in Figure 7c and 7d. This result points to a failure of machine learning in this model; the gradients backpropagated through the $\mathcal{M}$ are simply attenuated to almost zero magnitude and therefore the backpropagintg information is not reaching its goal.

This result is however very unexpected because the error of the network is still high and the neural model for $\mathcal{M}$ is not changing and is relatively small. Therefore the vanishing gradient should not be happening at this stage.

As an illustration of this observation, Figure 8 shows the gradients on every weight generated at 135 epoch of training for each of the errors. The left column shows the gradients magnitude on the each weight of the output layer of the ARK network. The right columns shows the magnitude of the weights. The top row shows the gradients and weights obtained from backpropagation through the $\mathcal{M}$ network while the bottom row shows the same data obtained directly from local errors. The most notable difference is a) difference of the magnitude between the two sets of gradients and b) the difference between the magnitude of the weights. In both cases the magnitude is higher when calculated directly from the local error.

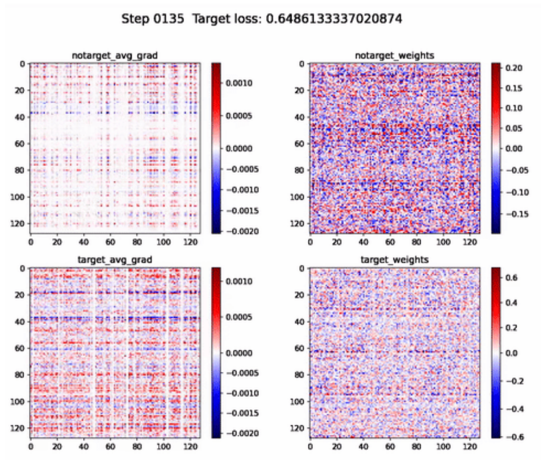


Fig. 8: Comparison on weight individual gradients during one backpropagation during the epoch 135. Note that the gradients obtained through backpropagation through $\mathcal{M}$ have much lower magnitude and are very close to zero.

1) On Purpose Trained Backpropagation: To solve this issue of vanishing gradients we learned the backpropagator in a feedforward manner: this means that we train a specific neural network to backpropagate the gradient across the pre-trained blocks in the studied model. Note this is quite different from the inverse model. The inverse model was simply an inverse operation of the AES component algorithm. Here the network is specifically trained on backpropagation data during the learning of using standard pre-trained AES component algorithms.

First, the analysis of the distribution of the gradient's values is shown in the histogram shown in Figure 10. The Figure 10a shows the histogram of gradients obtained by the local error and Figure 10b shows the gradients obtained as a result of backpropagation. The plot shows the actual gradients (not averaged values) of individual weights. The most important

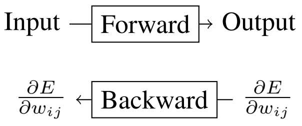
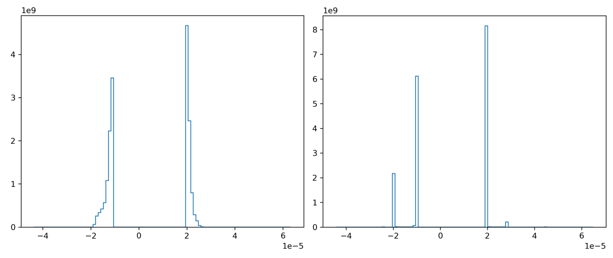

Input
Forward
Output
$\frac{\partial E}{\partial w_{ij}}$
Backward
$\frac{\partial E}{\partial w_{ij}}$

Fig. 9: The on-purpose trained backpropagator of the AES gradients.



(a) $\mathcal{E}_0$ (b) $\mathcal{E}$

Fig. 10: The histograms of gradienst a) from local error and b) from backpropagation through the $\mathcal{M}$ network.

observation is the very low diversity and relatively well grouped values of the gradients.

The results of training a network model on the gradient data directly resulted at maximum in an accuracy of $\approx 38\%$. We also attempted to train a model on a binned gradients so that there are at most for classes of available outputs and reduced the network model to a 32×32 input output mapping. The result improved up to $\approx 47\%$ for four classes. However the main surprising result is that similarly to the case of the model that contains $\mathcal{E} \circ \mathcal{B}$, a model containing two networks $\mathcal{E} \circ \mathcal{M}$ can learn the decryption with relatively high accuracy when both models are subject to training. In such a case the resulting accuracy is 83% as compared to $\approx 50\%$ when training only the ARK model with $\mathcal{M}$ having its weights frozen.

G. Data Types

The final set of experiments are focused on the learning data. As was shown previously training with randomly generated data does not forces the network to learn the decryption efficiently. In most cases, when learning larger AES blocks (one full round or more) the results remain random. This however should be expected because the purpose of the whole AES cryptosystem is to generate the outputs as randomly looking as possible and therefore minimizing the ability of an attacker to gain any significant information from it.

Therefore we decided to learn the simplest components of the AES system with text data scraped from various publicly available web pages. Training a network for learning the gradients distribution across the $\mathcal{M}$ model resulted in much higher scores this time.

The first seet of experiments are The various results obtained for the ANN used are shown in Table II. The network evaluated this time was a ResNet-50 trained form scratch on the scraped text data from wikipedia.

TABLE II: Evaluation of learning the gradients backpropagation using various data and various keys

Dataset	Accuracy
Train Set $\mathbb{D}_w$	94.59%
Eval Set $\mathbb{D}_w$	76.26%
Eval Set $\mathbb{D}_r$	60.26%
Eval Set $\mathbb{D}'_w$	58.24%
Eval Set $\mathbb{D}'_r$	62.21%

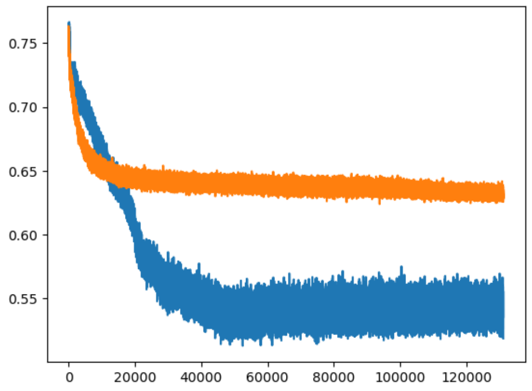


Fig. 11: LEarning the ARK→M network with pretrained backpropagator (blue) and using backpropagation (orange)

The results in Table II show that indeed the learning of even the gradients backpropagation are dataset and key dependent. The highest accuracy obtained is when the trained gradient backpropagator is evaluated on the same type of data encrypted with the same key (76.26%). The lowest accuracy is when the same data is encrypted with different key (58.24%). Using the model for backpropagatio instead of backpropagation works for learning the encryption on the same key with wikipedia data(128→ARK→MC→128 model). The result of learning is shown in Figure 11. The orange line shows the loss during learning using backropagation and blue line shows the learning loss when using the model for propagating the gradients. The validation accuracy of the model trained with backpropagation resulted in validation error of 32% while using the backpropagation network the resulting validation accuracy is 92%. However, using the model while trying to learn random data encrypted with another key, the loss does not improve, validation error still stuck at 50

VI. Conclusion

In this paper we showed empirically the limits of neural cryptanalysis on the AES 128 encryption system. The experiments have shown that despite the deep learning models having achieved great advancements on the processing of real-world and noisy data the learning of artificial systems with specific properties remains a challenge. The evaluated methodology has shown that as it stands the AES128 cipher is resistant due to specific components such as mainly the $\mathcal{M}$ and less due to the $\mathcal{B}$ components that prevent the required backpropagation for learning. These difusion operators operating in the Galois Field are difficult to simulate for neural networks but as has been shown the decription can be learned for single round of AES when started from pre-trained models and adapts all of the components.

References

[1] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*. Berlin, Heidelberg: Springer-Verlag, 2002.

[2] J. Daemen, “Aes proposal : Rijndael,” 1998.

[3] A. Biryukov and D. Khovratovich, “Related-key cryptanalysis of the full aes-192 and aes-256,” in *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT ’09, (Berlin, Heidelberg), p. 1–18, Springer-Verlag, 2009.

[4] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, “Key recovery attacks of practical complexity on aes variants with up to 10 rounds,” in *IACR Cryptology ePrint Archive*, 2010.

[5] A. Bogdanov, D. Khovratovich, and C. Rechberger, “Biclique cryptanalysis of the full aes,” in *International Conference on the Theory and Application of Cryptology and Information Security*, 2011.

[6] T. Liu, S. Tessaro, and V. Vaikuntanathan, “The t-wise independence of substitution-permutation networks.” Cryptology ePrint Archive, Paper 2021/507, 2021. https://eprint.iacr.org/2021/507.

[7] B. Cogliati, Y. Dodis, J. Katz, J. Lee, J. P. Steinberger, A. Thiruvengadam, and Z. Zhang, “Provable security of (tweakable) block ciphers based on substitution-permutation networks,” in *Advances in Cryptology – CRYPTO 2018*, vol. 10991 of *Lecture Notes in Computer Science*, pp. 722–753, Springer, 2018.

[8] H. Kimura, K. Emura, T. Isobe, R. Ito, K. Ogawa, and T. Ohigashi, “Output prediction attacks on block ciphers using deep learning.” Cryptology ePrint Archive, Paper 2021/401, 2021. https://eprint.iacr.org/2021/401.

[9] M. F. Idris, J. S. Teh, J. L. S. Yan, and W.-Z. Yeoh, “A deep learning approach for active s-box prediction of lightweight generalized feistel block ciphers.” Cryptology ePrint Archive, Paper 2021/066, 2021. https://eprint.iacr.org/2021/066.

[10] Y. Xiao, Q. Hao, Danfeng, and Yao, “Neural cryptanalysis: Metrics, methodology, and applications in cps ciphers,” 2019.

[11] S. Baek and K. Kim, “Recent advances of neural attacks against block ciphers,” 2019.

[12] B. Nollet, M. Lefort, and F. Armetta, “Learning arithmetic operations with a multistep deep learning,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2020.

[13] K. Abe, T. Maehara, and I. Sato, “Abelian neural networks,” 2021.

[14] C. Schofield, *Optimising FORTRAN programs*. Chichester: Ellis Horwood Publishing, 1989.