Attacking Pseudo-randomness with Deep Learning

Jean-Baptiste Michel

School of Computer and Communication Sciences

Semester Project

June 2023

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Supervisor
Dr. Ritam Bhaumik
EPFL / LASEC

Abstract

In this work, we propose an approach that leverages the predicting power of deep learning to uncover hidden correlations in a pseudo-random environment. In cryptography, pseudo-random sequences are crucial as they determine the security of algorithms by introducing unpredictability, making it extremely challenging to anticipate these number sequences, and hence, crack the encryption. Most existing work involving deep learning focuses on differential cryptanalysis, round-reduced ciphers and key recovery attacks. This project experiments on a range of attacks leveraging popular deep learning techniques. We present an end-to-end deep-learning based framework for block cipher key recovery and plaintext recovery attacks against a range of block ciphers. One of our noteworthy contributions is a promising attack employing a feedforward neural network against Simplified AES (a vulnerable educative version of Advanced Encryption Standard), showing the potential of such techniques against weak encryption algorithms.

Contents

Chapter 1

Introduction

1.1 Background

With democratization of deep learning in recent years, it has become an evidence to apply this data-powered tool to all kinds of problems. Deep neural networks have the ability to learn a wide range of correlations in large collections of data. In cryptography, pseudo-randomness is a central concept, as the security of many systems relies on the fact that an algorithmically generated sequence of numbers is hardly distinguishable from a truly random sequence.

1.2 Motivation

By definition, we cannot predict randomness. But a classic question in science asks what is truly random? If a sequence of number is not truly random, can we then use an algorithm to predict it? Pseudo-random number generators have a great importance in cryptography. This work focuses on the pseudo-randomness question in a cryptography setting, because the ability to break pseudo-randomness in this field has a big impact. Reversing ciphers without a key, or recovering the key breaks the security of a cipher. The accessibility of larger amounts of data and compute power drives us to attack ciphers with deep learning techniques. Successful attacks using neural networks in previous works also gives hope of finding patterns in pseudo-random sequences.

1.3 Goal

The goal of this project is to perform deep learning-based attacks on cryptosystems using accessible data generation and neural networks. The objective is not to use any existing cryptanalysis techniques like differential attacks, or try to decompose the internals of a cipher. The goal is also to provide a modular framework to easily perform these kinds of attacks on several different ciphers. We want to perform key recovery and plaintext recovery attacks and verify whether or not popular ciphers like AES (Advanced Encryption Standard), Simplified AES and Speck are vulnerable to neural network-powered attacks.

1.4 Contributions

The core contribution of this work is a deep learning attack framework from data generation to training and testing. The data generation pipeline provides ready-to-use data for neural networks. For each cipher we want to attack, two types of datasets can be created. The first one is plaintext-ciphertext pairs as samples and keys as labels for known plaintext key-recovery attacks. The second one has ciphertexts as samples and plaintexts as labels for fixed key plaintext recovery attacks. The pipeline is modular and it is easy for a programmer to add a new cipher to attack. The data is automatically pre-processed to fit the neural network's formatting needs (if using the Keras deep learning library). The framework also provides the code to perform the attacks from the training to the testing of the deep learning model. The network architecture is at the discretion of the researcher, allowing easy modification for various needs. For testing in the case of a plaintext recovery attack, the framework can apply some corrections to the predictions assuming that the plaintexts are formed of ASCII-encoded text. The framework assesses the performance of the neural network using a classical accuracy metric and a custom byte-accuracy metric. All parts of the process are made modular and parametrizable.

The other contributions of this work are experiments using the framework. We perform attacks on three ciphers: Advanced Encryption Standard (AES), Simplified AES and Speck. For key-recovery attacks, our neural networks try to recover the first bit of the key. In the case of plaintext recovery attacks, the models try to predict a ciphertext given the plaintext. In each case, we train a new neural networks with hyper-parameters adapted to the problem (key size and block size). We perform time measurements for data generation and network training, and report our results. Then, we test the trained models with unseen data and assess the performance. For plaintext recovery attacks, we use both basic and more advanced neural networks. This paper aims to show a range of naive attacks based on deep learning and their efficacy, feasibility, and accuracy.

1.5 Related Work

The interaction between pseudo-randomness and neural networks is a subject that has recently sparked the interest of researchers. In 2018, researchers tried to demonstrate how much a neural network could learn from a pseudo-random noisy environment: in [4], Fan and Wang detected some correlation in the sequence of decimals of π.

This subject also attracted many researchers in the field of cryptography to attack block ciphers. In 2019, Gohr focused on applying deep learning on differential cryptanalysis to attack round-reduced Speck. The aim in [5] was to differentiate ciphertext-pairs with a predetermined plaintext difference from random pairs by creating and using a neural network distinguisher (or *neural distinguisher*). In a follow-up paper ([3]), Benamira, Gerault, Peyrin and Tan provided a more in-depth analysis and explanations about Gohr's neural distinguisher, and improved its accuracy. Between 2020 and 2023, other papers demonstrated deep learning-based differential attacks on lightweight ciphers, like [2], [9], [7] and [12]. Following this spark in differential-neural cryptanalysis, Gohr, Leander, and Neumann published [6] in 2022, comparing and commenting a multitude of so-called neural distinguishers. They also published the code for these attacks. These papers all used neural networks for a specific cryptanalysis technique, most often attacking lightweight and round-reduced block ciphers. Their research showed more precisely where the attacked ciphers lack pseudo-randomness.

In 2020, So published [14] in which he proposes a generic cryptanalysis model based on deep learning. For text-based keys, the author shows the feasibility of finding the key of block ciphers from known plaintext-ciphertext pairs. The attack succeeds for full rounds of Simplified DES, Speck and Simon, using a feed-forward deep neural network. The authors of the follow-up paper [10] improved the attacks by designing neural networks that take into account the underlying characteristics of the cryptographic algorithms. They used state-of-the art network topologies: residual neural networks and gated linear units. As for plaintext restoration from ciphertext without knowledge of the key, Hu and Zhao developed a feedforward neural network attacking the AES cipher in [8]. They were able to restore an entire byte with a probability larger than 0.4.

1.6 Outline

First, chapter 2 explains technical preliminaries. We describe in brief the ciphers we will attack, as well as the attack models. We also describe the deep learning models used, and provide basic precision about the different functions used to describe our models. In chapter 3, we explain how we generated the data, describe our deep learning framework and discuss in detail all attacks we performed. Our results and evaluation are presented in chapter 4. Finally, chapter 5 concludes the work, and we discuss possible future work for this project.

Chapter 2

Preliminaries

2.1 Cryptography

2.1.1 Pseudo-randomness

A sequence of numbers that are not truly random, but are generated in a way that they exhibit properties of randomness, such as having a uniform distribution of digits, is denoted as pseudo-random. A pseudo-random number generator (PRNG) produces a sequence of numbers determined by a deterministic algorithm that takes as input a starting value called a seed.

2.1.2 Ciphers

In this work, we will use block ciphers as PRNGs. Block ciphers are deterministic algorithms used in cryptographic protocols to encrypt data, having as input a secret key and a plaintext. The key and plaintext are the seeds, or sources of randomness, and the cipher is the function with a random-looking output. Encrypting a plaintext gives us a ciphertext, which is therefore a pseudo-random sequence of numbers.

AES

Advanced Encryption Standard (AES) is a widely used symmetric-key encryption algorithm. It is a block cipher that uses a fixed block size of 128 bits and supports key sizes of 128, 192, or 256 bits. In this work, we focus on AES-128 but our framework supports all three key sizes. We also explore a simplified version of AES called *Simplified AES* (S-AES) [11] which has a block size and key size of 16 bits.

Speck

Speck is a family of software-optimized block ciphers. It supports key sizes ranging from 64 bits to 256 bits and block sizes varying from 32 bits to 128 bits. In our experiments, we focus on Speck32/64 (32 bits blocks and 64 bits key). Again, our framework also supports other key and block sizes. It is considered to be a lightweight cipher and is thus often attacked in academic papers. In this work, we also study *round-reduced Speck*. Round-reduced Speck is a variant of the Speck cipher that has a reduced number of encryption rounds, making it a weaker target for cryptanalysis. We denote 1-round Speck (or 1-R Speck) as the variant of Speck with only one round.

2.1.3 Attacks

Key Recovery

We consider a known plaintext-ciphertext pair key recovery attack model. The adversary attempts to recover a secret key K used by the block cipher to encrypt a plaintext m. Let $c = ENC_K(m)$ be the encryption of plaintext m under the key K. The adversary has access to a number of plaintext-ciphertext pairs (m_i, c_i), all encrypted with different unknown keys K_i's.

Plaintext Recovery

We also consider a fixed key plaintext attack model. Here, the adversary attempts to recover the plaintext m corresponding to the ciphertext $c = ENC_K(m)$. In this model, the adversary only has access to ciphertexts c_i's corresponding to unknown plaintexts m_i's, all encrypted under the same key K.

2.2 Deep Learning

2.2.1 Neural Network

Deep learning utilizes artificial neural networks as non-linear algorithms. A neural network consists of a series of layers, with each layer containing multiple nodes (or neurons) that perform a weighted sum of inputs from the previous layer, followed by a non-linear activation function. By adjusting the weights between layers to minimize the loss in the output, neural networks can learn to classify, predict, or generate new data. Deep learning has shown remarkable success in a wide range of applications. In this work we apply such algorithms to cryptographic attacks. A neural network is trained with samples and labels. Its goal is to predict the correct label when given a sample.

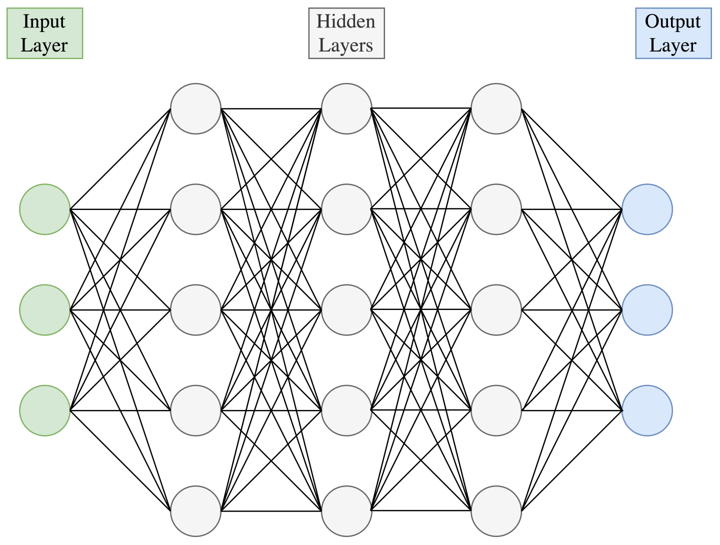


Figure 2.1: Feedforward neural network with `3-5-5-5-3` structure.

Network Architecture

There are many types of artificial neural networks, adapted to different tasks. We focus on two widely used architectures: *Feedforward Neural Networks* (FNN) and *Residual Neural Networks* (ResNet).

FNN's, also known as multi-layer perceptrons, are the most basic type of neural network, consisting of multiple layers of fully interconnected nodes, as shown on Figure 2.1. Information flows in a forward, sequential direction. They are widely used for classification, regression and pattern recognition tasks.

Residual networks address some challenges of training very deep neural networks (tens or hundreds of hidden layers), suffering from performance degradation when more layers are added. ResNets employ skip connections, which can bypass one or more layers. In this work, we build such networks using residual blocks pictured on Figure 2.2. This type of network is known to succeed in computer vision tasks, although their ability to propagate information deeper in a network by skipping layers can have applications in other fields.

Loss Function

The loss function compares the predicted labels y with the actual labels $\hat{y}$. When training a neural network, the goal is to minimize the loss function. In our experiments, we used two loss functions: *Mean Square Error* (MSE), which is one of the simplest and most popular loss functions in deep learning, and *Binary Cross Entropy* (BCE), also known as *log-loss*, which is popular for binary

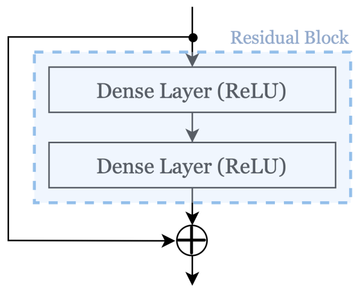


Figure 2.2: Single residual block.

classification problems. There are defined as such:

$$MSE = \sum_{i=1}^{D} (y_i - \hat{y}_i)^2$$

$$BCE = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Accuracy

The accuracy is a metric that serves to assess the effectiveness of a deep learning algorithm. When building models, we encouter different types of accuracy. First, the training accuracy, which is computed during training. It helps us understand how well the network has learned the training data provided and is computed at each epoch. The second is computed after training. The testing (or validation) accuracy of a neural network is calculated with a test set that is composed of sample-label pairs not present in the training set. Accuracy is computed by taking the number of correct predictions per output neuron divided by the total number of predictions. Note that in our designs, the output layer outputs, for each neuron, a probability. Therefore, the outputs are first scaled up to 1 if there are over 0.5 and down to 0 otherwise.

The previous accuracy metrics (training and testing) are general to neural networks but do not capture the essence of our problem very well. Therefore, we also define more representative accuracy metrics. The overall accuracy is the amount of correct predictions divided by the total amount of samples in the test set: $accuracy = \frac{\#correct\ predictions}{\#samples\ in\ test\ set}$. An accuracy of 1 would mean that the network always predicts the correct value, and 0 never. A prediction is correct if all output neurons predict the correct value. We also define and implement a custom metric adapted to our problem. Let the byte accuracy be the number of correctly predicted bytes divided the total number of bytes. This metric helps us quantify how many characters of a plaintext our neural network can recover.

Overfitting

Having a training accuracy significantly superior to the testing accuracy indicates overfitting. An overfitted model is too closely fit to the training set, resulting in poor predictive performance on unseen data. We say the model is unable to generalize.

Activation Function

In this work, we utilize two commonly employed activation functions in neural networks: *Rectified Linear Unit* (ReLU) and *Softmax*. Activation functions play a crucial role in introducing non-linearities to neural networks, to model complex relationships and make accurate predictions. ReLU is effective, simple, and widely used in deep learning. In this work, we use it as activation function in all hidden layers.

$$ReLU(z) = max(0, z)$$

For output layers, we use the Softmax function. It is also popular, for output layers of multi-class classification neural networks. We employ the Softmax activation function in the output layer to obtain probability distributions over multiple classes.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

Optimizer

The optimizer is used in a neural network to iteratively update the model's parameters during training, aiming to minimize the loss function and improve performance. In this work, we use the *Adam* (Adaptive Moment Estimation) optimizer, a popular choice in deep learning. It is known to be effective with noisy data.

2.2.2 Tensorflow/Keras

We use *Tensorflow/Keras*, a deep learning framework that provides a high-level API for building and training neural networks with the Python programming language. It provides an intuitive and user-friendly interface which is well suited for research. It allows quick prototyping and experimenting with various neural network architectures.

Chapter 3

Experiments

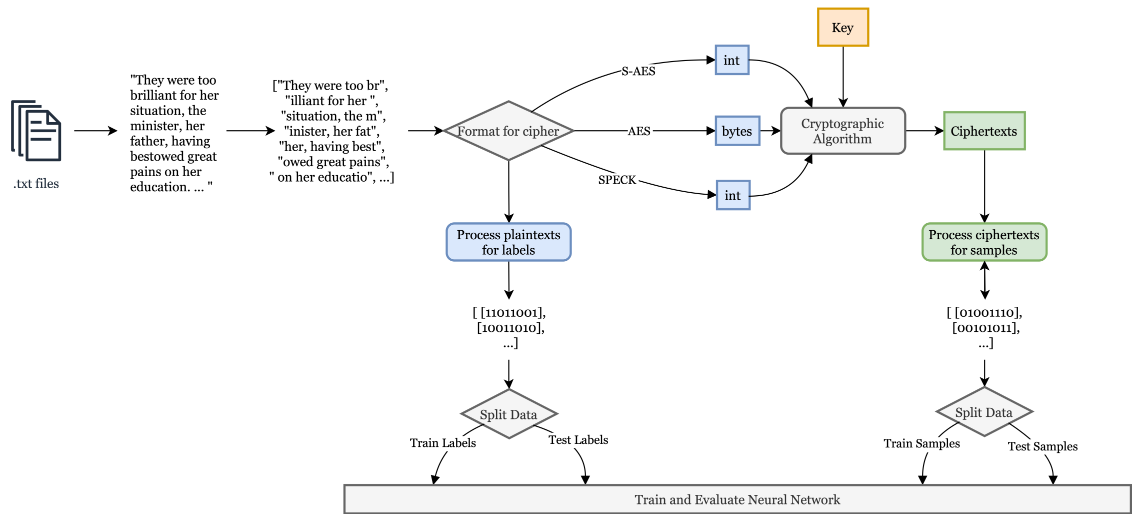
All experiments in this work were conducted on a single machine, a 2020 MacBook Pro with a 2 GHz Quad-Core Intel Core i5 processor, and 16 GB of memory, using macOS Ventura 13.3.1. We use Jupyter Notebook as a development environment, with a Python 3.11.2 kernel.

3.1 Proposed Framework

In this section, we describe the different components that were built to make the attack process modular.

3.1.1 Data Generation

Deep learning models are trained with data. During the training process, we feed it with sample-label pairs. For a known plaintext-ciphertext pair key recovery attack, the samples are the plaintext-ciphertext pairs and the labels are the keys. For a fixed key plaintext recovery attack, the samples are ciphertexts and the labels are plaintexts. In this section, we describe how we generated datasets for all experiments. For both types of attacks, plaintexts are generated from ASCII character strings, drawn from a collection of 82 English books (such as *The Confessions of Jean-Jacques Rousseau* or *The Rights of Man* by Tomas Paine). These books were downloaded from Project Gutenberg [13], an online library of free electronic books. The choice of having English text plaintexts was made to lower the entropy of plaintexts (as opposed to uniformly random bits). The `.txt` files are transformed to an array of string of desired sizes (usually the block size of the cipher). We implemented several possible splitting strategies: split every N bytes (N fixed, usually block size), split every punctuation, every group of words that will amount top approximately to block size, and split every word. All those strategies except the first require some padding. All experiments reported in this work use



.txt files
"They were too brilliant for her situation, the minister, her father, having bestowed great pains on her education. ... "
["They were too br", "illiant for her ", "situation, the m", "inister, her fat", "her, having best", "owed great pains", " on her educatio", ...]
Format for cipher
S-AES
AES
SPECK
int
bytes
int
Key
Cryptographic Algorithm
Ciphertexts
Process plaintexts for labels
Process ciphertexts for samples
[[11011001], [10011010], ...]
[[01001110], [00101011], ...]
Split Data
Train Labels
Test Labels
Split Data
Train Samples
Test Samples
Train and Evaluate Neural Network

Figure 3.1: Data generation and pre-processing pipeline.

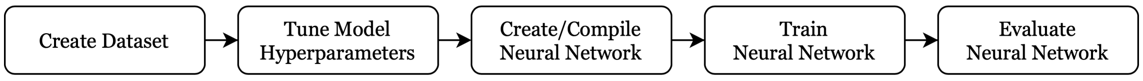


Figure 3.2: End-to-end attack process.

the first strategy (splitting the text every N characters). From those strings, the pre-processing pipeline transforms them inåto data types accepted by the encrypting function. The types differ between AES, Simplified AES or Speck. Plaintexts are also transformed to arrays of 1's and 0's, which is the type that will be fed to the neural networks. The plaintexts are encrypted using random keys (generated with Python's *secrets* module), and ciphertexts are also transformed to binary arrays. We made the choice to only encrypt one block, to keep data samples an acceptable size, as neural networks will be trained on a single machine. For key recovery, the binary plaintexts and ciphertexts are concatenated to create samples, and the keys are transformed to binary to create labels. For plaintext recovery, the plaintexts are encrypted using a single random fixed key, also generated with Python's *secrets* module. Both plaintexts (labels) and ciphertexts (samples) are transformed into binary arrays. The process is pictured on Figure 3.1. For each attack and cipher, the framework exposes an easy-to-use interface, where the programmer chooses between two dataset sizes. The interface returns training and testing datasets as *numpy* binary arrays ready to be fed to neural networks created with the Keras library. Note that for the round-reduced Speck experiments, data was generated by slightly modifying the code [1] for [6], to adapt it to our needs.

3.1.2 Deep Learning Pipeline

For the rest of the framework, a template *Jupyter Notebook* serves as a basis to train a model and evaluate the attack. The steps of the process performed on this template are picture on Figure 3.2.

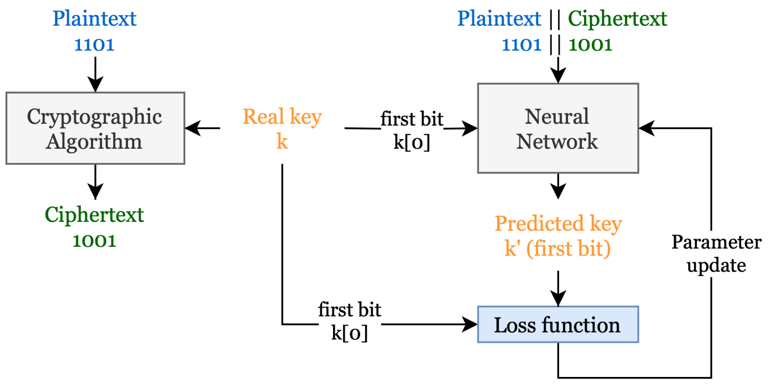


Figure 3.3: Diagram for key recovery method.

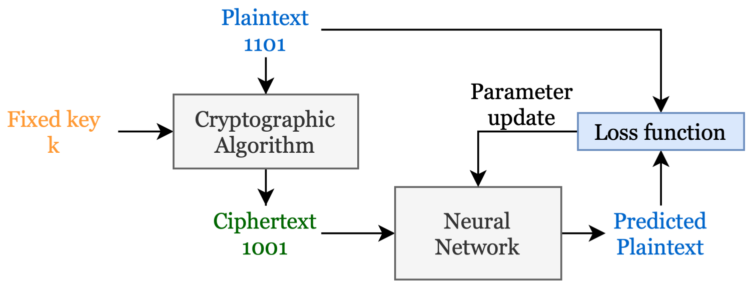


Figure 3.4: Diagram for plaintext recovery method.

The programmer creates the wanted dataset in a dedicated cell using the public facing interface of the data generation pipeline. When training a deep learning model, it is important to tune hyper-parameters, like the learning rate for the optimizer, the number of epochs, the batch size or the number of units per hidden layer. Then, the programmer can create any neural network using the Keras library. The training then takes place, and finally the programmer tests and evaluates the model, and can save it once it obtains satisfying accuracy. To evaluate a program, the Keras framework outputs the model accuracy. We also designed a custom per-character accuracy (byte accuracy) described in section 2.2.1. There is a possibility, when testing a model, to use ASCII-byte correction before testing, so that the framework corrects invalid ASCII bytes. This function is implemented by correcting an erroneous byte to the closest valid printable ASCII byte according to their Hamming distance. There are 95 printable ASCII bytes. This functionality is useful in the case where plaintexts are known to be ASCII-encoded characters, and enhances the model.

3.2 Attacks

In this section, we describe each different experiment we performed, for each type of attack. We trained a broad set of neural networks, with different parameters, ciphers, dataset sizes. The parameters and hyper-parameters are decided based on trial and error. We report the setting of the most relevant attacks: key size, block size, type of network, number of hidden layers, parameters, hyper-parameters, dataset sizes and training time. All networks are designed to be trainable in manageable time (typically less than 2 hours), using reasonable compute power. In this work, we experiment two types of neural networks, namely feedforward neural networks and residual neural networks. FNNs are widely used and easier to train, and therefore were adapted to the objective of this work. We also used ResNets, following their success in cryptographic tasks in [10], although these are much harder to train. In this section, we say a network *learns* something if its output is not fixed for all given inputs, i.e. it makes a decision according to the data it is fed and not the training set distribution.

3.2.1 Key Recovery

In the key recovery attacks, the adversary has a plaintext-ciphertext pair and tries to recover the corresponding key. In each of the following experiments, we try to recover the first bit of the secret key. A successful key recovery attack would imply (in our case) successfully recovering the first bit of the key. This choice was made to reduce the complexity of the neural networks created in this first set of experiments. In the following subsection, we design binary classifiers trained with plaintext-ciphertext pairs as training samples and random keys as labels. In this section, we always use binary cross-entropy as a loss function, as it is a standard for binary classifiers, and the Adam optimizer (as mentioned in section 2.2.1). Training a neural network is a long process, and hyperparameter tuning gets us to a satisfying model. We try to train all network in a way that they reach some degree of loss convergence, without getting *stuck* to outputting the most common label in the dataset (in this case always zero or always one). This way, we ensure that the model learns something. For all three attacked ciphers, attack parameters and metrics are reported on Table 3.1. In chapter 4, we analyze our results more in-depth.

Cipher	`AES`	`S-AES`	`SPECK`
Key Size/Block Size (bits)	128/128	16/16	64/32
Network Type	Feedforward	Feedforward	Feedforward
Hidden Layers	3	3	3
Units per Hidden Layer	512	256	128
Total Parameters	657’409	140’289	53’761
Learning Rate	0.001	0.001	0.001
Batch Size	1000	5000	5000
Epochs	150	50	50
Training Set Size	1’445’705	11’565’605	5’782’806
Testing Set Size	619’588	4’956’688	2’478’345
Training Time	66m	49m36s	13m58s

Table 3.1: Summary of each model parameters for first bit key recovery.

AES

The first experiment was to attack AES, the most robust, tested, and used cipher in our targets. Here, we use AES-128 (128 bits key), with 128 bits plaintexts. We train a sequential feedforward neural network with 512 units each. The network shape is as follows: `256-512-512-512-1`. The input samples are 256 bits, we used hidden layers twice as large as the input layer, and the output layer is a single node as the model is a binary classifier. During training, the loss went from 0.6939 to 0.49.

S-AES

Attacking Simplified AES allowed us to have a network larger than AES, proportionally to the sample size. We also had more data (as plaintexts are only 16 bits). The sample size is thus 32 bits. The network shape is the following: `32-256-256-256-1`. In this experiment, the loss went down from 0.6932 to 0.6752, then stayed to that level for some epochs.

Speck

Finally, we trained a model to attack the lightweight cipher Speck, with 64 bits keys and 32 bits plaintexts (Speck 32/64). The sample size was therefore 64 bits. The network has a `64-128-128-128-1` shape. Like Simplified AES, the training loss decreased only a little, going from 0.6933 to 0.6881, then reached convergence.

3.2.2 Plaintext Recovery

After attempting key recovery attack, we switch to fixed key known ciphertext plaintext recovery attacks. The setting is the following: for the ciphers we attack, we train neural networks to recover a plaintext from a ciphertext without knowledge of the key. The networks are trained with ciphertexts as samples and plaintexts as labels. All ciphertexts are encrypted plaintexts with a fixed key. In this section, we try different types of neural networks: sequential feedforward networks (with parameters and hyperparameters reported on Table 3.2) and residual neural networks (with parameters and hyperparameters reported on Table 3.3). As described in section 2.2.1, each residual block has two layers. Residual neural networks are harder to train than feedforward networks. For all different models, the loss function is mean square error (see section 2.2.1), and we use the Adam optimizer. Again, we train our models until their loss reach convergence, making sure they do not get stuck outputting the same value for any input.

Cipher	S-AES	AES	1-R SPECK	7-R SPECK	7-R SPECK
Key/Block Size (bits)	16/16	128/128	32/32	32/32	32/32
Hidden Layers	3	1	5	5	1
Units per Hidden Layer	256	2048	512	512	1024
Total Parameters	205’840	526’464	1’083’936	1’083’936	1’116’192
Learning Rate	0.001	0.001	0.001	0.001	0.001
Batch Size	100	5000	5000	5000	5000
Epochs	1000	500	70	100	2
Training Set Size	500	2’234’530	1’000’000	1’000’000	1’000’000
Testing Set Size	195’679	81’243	100’000	100’000	100’000
Training Time	55.4s	23m41s	29m27s	45m36s	55.6s

Table 3.2: Summary of each feedforward model parameters for plaintext recovery.

Cipher	AES	S-AES
Key Size/Block Size (bits)	128/128	16/16
Residual Blocks	5	5
Units per Hidden Layer	2048	2048
Total Parameters	18’632’832	37’908’560
Learning Rate	0.0001	0.001
Batch Size	5000	5000
Epochs	5	1000
Training Set Size	2’234’530	500
Testing Set Size	957’656	649’947
Training Time	68m39s	9m31s

Table 3.3: Summary of each residual model parameters for plaintext recovery.

AES

We first train a feedforward neural network designed to recover plaintext from corresponding ciphertexts. After trying several topologies, we settle on one large hidden layer, with 2048 neurons. The topology is the following: `128-2048-128`. In this type of attack, the output layer will have as many nodes as the plaintext size, as for every bit of the plaintext, it predicts whether its value is `1` or `0`. The training loss varies from 0.1784 to 0.1573.

We also train two different residual neural networks. In the first experiment, we use one residual block with 1024 neurons per layer (and a total of 3’424’896 parameters). The training loss stays the same, with a value of 0.4339. In the second experiment, we train a larger neural network, with more data. This time, we use 5 residual blocks with again 1024 neurons, for a total of 18’632’832 parameters. The loss is slightly lower, ranging from 0.4298 to 0.4296. In both cases, the models learned *something*, as they did not get stuck (output the same value for any input).

S-AES

When attacking Simplified AES with a fixed key, there is one thing we need to be careful about. There are 95 printable ASCII characters, and Simplified AES has a 16 bit block size (2 characters). Therefore, there are $95^2 = 9025$ different possible plaintexts, with some more frequent than others. Therefore, feeding too much data, i.e., more than 9025 samples, to the model would be equivalent to bruteforce the cipher. Therefore, we only feed 500 samples to the models we train.

The feedforward model we trained has the following structure: `16-256-256-256-16`. For this network, the training loss starts at 0.2327 and declines to 0.0018.

The residual neural network for this attack is the model with the most parameters we trained. Smaller networks did not learn anything as they outputted all zeros for all inputs. This model, with its 37 million parameters, decreased its loss from 0.3885 to 0.3776 during training.

Speck

Instead of attacking full-round Speck, we train models to attack round-reduced Speck. As a benchmark, we use 1-round Speck, and compare it with attacks on 7-round Speck.

Our feedforward model trained to attack 1-round Speck has 5 hidden layers with 512 neurons: `32-512-512-512-512-512-32`. Its training loss ranges from 0.1578 to 0.0996. We trained the same network topology with a 7-round Speck dataset. The loss slowly decreases from 0.25 to 0.2266. We also trained a different feedforward network for the same dataset, with a `32-1024-32` topology, that stayed at a 0.25 loss.

Chapter 4

Results and Evaluation

4.1 Data Generation

In this section, we report and comment on the performance of our data generation pipeline, by measuring the time to create the largest dataset available. On Table 4.1, we report results for each attack type and target cipher.

Cipher **Attack**	`AES` Key Rec.	`AES` Pt[1] Rec.	`S-AES` Key Rec.	`S-AES` Pt Rec.	`SPECK` Key Rec.	`SPECK` Pt Rec.
Dataset Size	3’192’186	3’192’186	25’537’173	25’537’173	3’192’186	3’192’186
Time	2m 34.8s	1m 36.7s	46m 47.3s	13m 50.3s	2m 51.3s	1m 54.5s

Table 4.1: Dataset generation time for largest option.

Simplified AES, with 16 bits plaintexts and keys, take the most time as more samples (over 25 million) are created and thus pre-processed, although it is useless to have that much samples for a plaintext recovery attack. AES and SPECK datasets are faster to create, as they have 128 bits block sizes, which results in less samples (over 3 million). For comparison, data generation of 1-round and 7-round SPECK (implemented in [1]) takes 0.2 seconds for 1’000’000 samples, although the samples aren’t created from English plaintexts, and directly created in the correct types for training neural networks.

[1] Pt stands for Plaintext.

4.2 Key Recovery

As we aim to recover the first bit of a random key, the models we train can output either `1` or `0`. To get a 0.5 accuracy, a model can output either the same value every time, or uniformly at random. Therefore, a successful model would have an accuracy larger than 0.5. It is important to note that the testing accuracy can also depend on the test set distribution. In our results, we state that a network learned something if the output depends on the input (and is not fixed for any input).

Target Cipher	`AES`	`S-AES`	`SPECK`
Model Accuracy	0.49867	0.51331	0.50101

Table 4.2: Model accuracy on test set for first bit key recovery attack.

4.2.1 AES

In the attack against AES, the trained model did learn something, but nothing useful for unknown data as the test accuracy was below 0.5. It is likely that the neural network overfitted on the training data, as the training accuracy reached 0.6956. It is also noteworthy that the training accuracy is higher than the models attacking S-AES and Speck.

4.2.2 S-AES

This attack reached the best accuracy: 0.51331. The training accuracy was at most 0.5730. We conclude that the model overfitted and that the result being slightly over 0.5 is not significant enough and was a result of luck.

4.2.3 Speck

The model attacking Speck (full-round), trained on our text dataset, also achieved an accuracy close to 0.5. Its training accuracy was 0.5171. We conclude that the network was unsuccessful at learning anything.

4.3 Plaintext Recovery

We report in this section the best performing neural network from the experiments conducted. For Simplified AES and AES, we also test the model accuracy after performing ASCII-byte correction on the outputs with 1000 test samples. (We do not apply this layer to round-reduced Speck as the plaintexts were not generated from English ASCII-formatted text).

Target Cipher	`AES`	`S-AES`	`1-R SPECK`	`7-R SPECK`	`7-R SPECK`
Testing Set Size	81’243	649’947	100’000	100’000	100’000
Model Accuracy	0.0127	0.49443	0.39945	0.03272	0.02121

Table 4.3: Accuracy of feedforward models for plaintext recovery.

Target Cipher	`AES`	`S-AES`
Correct Predictions	0	77
Prediction Accuracy	0.0	0.077
Total Bytes	16’000	2000
Correct Bytes	752	431
Byte Accuracy	0.047	0.2155

Table 4.4: Accuracy metrics of feedforward models for plaintext recovery after ASCII correction of 1000 predictions.

Target Cipher	`AES`	`S-AES`
Testing Set Size	957’656	649’947
Model Accuracy	0.00313	0.16864

Table 4.5: Accuracy of residual models for plaintext recovery.

4.3.1 AES

The feedforward network trained for the AES plaintext recovery attack achieves a testing accuracy of 0.0127, almost equal to the training accuracy (0.0125). During the ASCII-corrected test (with 1000 samples and thus 16’000 bytes), its byte-accuracy reached 0.047 as it predicted 752 bytes correctly (16’000 bytes in total). However, it did not recover any full sample correctly. The correctly recovered bytes were likely a result of the plaintext distribution. For example, if the character "e" was frequently in 4^{th} position in the plaintext, the model might also output an "e" in the same position because it minimized the loss. Therefore, if by luck, a test label also had an "e" in this position, we mark it as a correctly recovered byte. We conclude that eve though this network was able to learn *something* and did not overfit to the training data, it was not successful in attacking AES.

The residual network was much harder to train because of its size and complexity. It mostly overfitted to the data (although the accuracy is still far from good) as the accuracy at the end of the training was 0.015 and the test accuracy was 0.00313. It was unable to predict any bytes correctly. The feedforward network mentioned above had an accuracy four times better. We conclude that to have better results with a ResNet, we would need more layers, and thus more compute power and time, and hyper-parameter tuning would also be more complex.

Target Cipher	AES	S-AES
Correct Predictions	0	0
Prediction Accuracy	0.0	0.0
Total Bytes	16’000	2000
Correct Bytes	0	0
Byte Accuracy	0.0	0.0

Table 4.6: Accuracy metrics of residual models for plaintext recovery after ASCII correction.

4.3.2 S-AES

The feedforward neural network attacking S-AES was the most successful model we created. With 500 training samples, it reached a test accuracy of 0.3570, which is way larger than guessing at random, as there are 2^{32} possible outputs. In this case, the probability of having labels (plaintexts) that are present in both the training set and the testing set is very high, because some 2-letter strings may be very frequent in English text. For this reason, we filter out of the testing set the plaintext-ciphertext pairs that are present in the training set to avoid getting a biased accuracy. The following result show the ability of the neural network to correctly recover previously unseen labels. With the filtered testing set, the testing accuracy reached 0.32185, which is almost as good. After ASCII-correction on 1000 unseen testing sample outputs, it successfully recovered 77 plaintexts from their corresponding ciphertext (and 431 bytes over a total of $2'000$). This result shows the possibility for a neural network to express the relationship between ciphertexts and plaintexts without the knowledge of the key. The feasibility of scaling this attack to a full size and full round cipher remains unknown.

We also trained a residual network for this 500 samples dataset, and reached a test accuracy of 0.16864 (compared to the training accuracy of 0.1489 it converged to). However, it was unable to predict any correct plaintexts or bytes, with 0.0 byte accuracy.

4.3.3 Speck

The feedforward network we created to attack 1-round Speck gave us a successful accuracy: 0.39945. We do not develop further on this result as it wasn’t fully obtained with our framework, and we did not test its per-plaintext and per-byte accuracies. It although shows the possibility of training a model with a good accuracy (in this context) on 1-round Speck. Scaling it to more rounds remains a challenge.

Indeed, this attack did not scale for 7-round Speck, as the accuracy dropped to 0.02121 for a network with the same architecture. The best performing model for 7-round Speck achieved an accuracy of 0.03272, which is not significant enough to call this experiment a success.

4.4 Evaluation

4.4.1 Framework

The framework we developed was a necessary tool to be able to effectively experiment on this subject. Data creation and pre-processing is a tedious task. Each different cipher requires different formats and types, and it is important to think well about how we prepare the data for the neural network. Having this component completed allows faster implementation of a wide range of different experiments.

Deep learning is a very wide and complex subject with a steep learning curve, but thanks to the Keras library, we were able to build neural networks quickly. It is although important to understand each step of the model building process, especially network architectures and hyper-parameter tuning. The template we developed allowed us to reduce experiments to the strict minimum, namely designing the network, tuning the hyper-parameters, training, and testing. The framework abstracted away many parts of the end-to-end process and greatly reduced the cost of doing one experiment.

4.4.2 Attacks

Building a neural network is a difficult trial and error process, that requires attention to many details. The time of training a neural network is dependent on the size of the samples, the size of the dataset and the number of parameters of the model. An important decision for this project was building models with a realistic dimension to be trained on a single machine, without requiring external compute power. With time as a constraint, it was important to be able to iterate experiments fast. With these assumptions, we were able to conduct a wide range of attacks.

4.4.3 Limitations

Within our setting, we proved the difficulty and complexity of attacking certain model, but also the ability of a neural network to learn from symmetric key cryptosystems (namely Simplified AES and 1-round Speck). It is important to note that we were limited to basic deep learning techniques. This field is very wide and a lot of progress is being made on a regular basis. There are certainly more adapted architectures, (hyper)-parameters that could be applied to our problem. We could think of compute power as another limitation for this project. Deep learning network excel on large datasets, and therefore require a large amount of parameters. Such networks require more compute power and more cycles to train. These limitations would require more time, more expertise on deep learning and perhaps a multi-disciplinary team.

Chapter 5

Conclusion

5.1 Main Results

We are still in the experimental phase of implementing deep learning for plaintext restoration. A good neural network would express the relationship between inputs and outputs. Therefore, we could think of the model as being an encoding of both the key and the cipher. In this work, with basic deep learning techniques such as feedforward and residual neural networks, we showed the (un-)feasibility of key recovery and plaintext recovery attacks on different types of block ciphers. Experiments were conducted, thanks to an end-to-end framework we developed, to attack AES, Simplified AES, and full-round, 1-round and 7-round Speck. We demonstrated that a neural network can learn a relationship between ciphertexts and plaintexts to recover plaintexts from unseen ciphertexts in the case of Simplified AES and 1-round Speck. We attacked Simplified AES with a prediction accuracy of 0.077, and were able to correctly recover 431 plaintext bytes (or characters) out of 2000 from their corresponding ciphertexts, resulting in a byte accuracy of 0.2155. Other attacks were unsuccessful. Key recovery attacks were also unsuccessful. In a pseudo-random environment, networks are very hard to train and often converge to all zeros or fixed outputs, because it locally minimizes the loss. We observed overfitting to training samples, but the ability to still learn *something* shows some promise for further research in this area. As previous work focused on differential cryptanalysis or key-recovery attacks, this is, along with [8], one of the few works focusing on plaintext recovery attacks with deep learning.

5.2 Future Work

It is a fact that the larger a neural network is, the harder it is to train. Additionally, the training time becomes significantly larger. Future work could be trying giant neural networks, with larger amounts of data. It is important to keep in mind the total number of possibilities, because giving too much data to a neural network would make it become a bruteforce attack. But this process becomes extremely costly in compute power and time, which defies the purpose of a feasible attack. It is important to take into account, when discussing the success of an deep learning based attack, to compare it to the complexity of classical cryptanalysis. Also furthering the work on the deep learning side, it would also be interesting exploring more advanced types of neural networks.

During this research project, we also studied the possibility of adding a text correction layer to our system, although our results were not successful enough for this amelioration to be useful. Similar to our ASCII-correction function, it would help the model achieve higher accuracy with the assumption of plaintexts being extracted from English texts.

In the future, it would also be interesting to extend the successful attacks to longer plaintext with modes of operations like *Electronic Code Book* (ECB) or *Cipher block chaining* (CBC). Also, it would be interesting to experiment even smaller dataset sizes (less than 500) to measure how much data the S-AES plaintext recovery attack model (in subsection 4.3.2) needs to be accurate.

Finally, to focus the plaintext recovery attack research, an idea would be to adapt the well-designed code from [1] to plaintext recovery attacks instead of differential attacks. This repository contains the code for many round-reduced ciphers, like ChaCha, Speck and Simon.

Bibliography

[1] *An Assessment of Differential Neural Distinguishers, Source Code.* 2023. URL: `https://github.com/differential-neural/An-Assessment-of-Differential-Neural-Distinguishers/tree/main` (visited on 05/15/2023).

[2] Anubhab Baksi, Jakub Breier, Yi Chen, and Xiaoyang Dong. "Machine learning assisted differential distinguishers for lightweight ciphers (extended version)". In: *Cryptology ePrint Archive* (2020).

[3] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. *A Deeper Look at Machine Learning-Based Cryptanalysis.* Cryptology ePrint Archive, Paper 2021/287. `https://eprint.iacr.org/2021/287`. 2021. URL: `https://eprint.iacr.org/2021/287`.

[4] Fenglei Fan and Ge Wang. "Learning From Pseudo-Randomness With an Artificial Neural Network–Does God Play Pseudo-Dice?" In: *IEEE Access* 6 (2018), pp. 22987–22992. DOI: `10.1109/ACCESS.2018.2826448`.

[5] Aron Gohr. "Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning". In: *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II.* Santa Barbara, CA, USA: Springer-Verlag, 2019, pp. 150–179. ISBN: 978-3-030-26950-0. DOI: `10.1007/978-3-030-26951-7_6`. URL: `https://doi.org/10.1007/978-3-030-26951-7_6`.

[6] Aron Gohr, Gregor Leander, and Patrick Neumann. "An Assessment of Differential-Neural Distinguishers". In: *Cryptology ePrint Archive* (2022).

[7] Zezhou Hou, Jiongjiong Ren, and Shaozhen Chen. "Cryptanalysis of round-reduced SIMON32 based on deep learning". In: *Cryptology ePrint Archive* (2021).

[8] Xinyi Hu and Yaqun Zhao. "Research on plaintext restoration of AES based on neural network". In: *Security and Communication Networks* 2018 (2018), pp. 1–9.

[9] Aayush Jain, Varun Kohli, and Girish Mishra. "Deep learning based differential distinguisher for lightweight cipher PRESENT". In: *Cryptology ePrint Archive* (2020).

[10] Hyunji Kim, Sejin Lim, Yeajun Kang, Wonwoong Kim, and Hwajeong Seo. *Deep Learning based Cryptanalysis of Lightweight Block Ciphers, Revisited.* Cryptology ePrint Archive, Paper 2022/886. `https://eprint.iacr.org/2022/886`. 2022. URL: `https://eprint.iacr.org/2022/886`.

[11] Mohammad A. Musa, Edward F. Schaefer, and Stephen Wedig. "A SIMPLIFIED AES ALGORITHM AND ITS LINEAR AND DIFFERENTIAL CRYPTANALYSES". In: *Cryptologia* 27.2 (2003), pp. 148–177. DOI: 10.1080/0161-110391891838. eprint: https://doi.org/10.1080/0161-110391891838. URL: https://doi.org/10.1080/0161-110391891838.

[12] Debranjan Pal, Upasana Mandal, Abhijit Das, and Dipanwita Roy Chowdhury. "Deep Learning Based Differential Classifier of PRIDE and RC5". In: *Applications and Techniques in Information Security: 13th International Conference, ATIS 2022, Manipal, India, December 30–31, 2022, Revised Selected Papers*. Springer. 2023, pp. 46–58.

[13] *Project Gutenberg*. 2023. URL: https://www.gutenberg.org (visited on 05/15/2023).

[14] Jaewoo So. "Deep learning-based cryptanalysis of lightweight block ciphers". In: *Security and Communication Networks* 2020 (2020), pp. 1–11.

Appendix A

Source code

The source code for this project is available on:
https://gitlab.epfl.ch/jbmichel/attacking-pseudorandomness-with-deep-learning

The directory is organized as follows:

- **notes**: This folder contains daily notes of the advancement of the project, some notes about the related papers for this project, all files containing the results of the experiments and the LaTeX report source code.
- **src**: This folder contains source code for the framework, all *Jupyter* notebooks for the experiments and some saved models.
  - **dataset**
    - **assets**: This folder contains all text files for the large dataset.
    - **assets2**: This folder contains one text file for the small dataset.
    - `datasets.py`: Public facing interface with classes for each type of dataset (corresponding to an attack).
    - `encryption.py`: Encryption methods for AES and Speck.
    - `saes.py`: Implementation of S-AES.
    - `get_data_class.py`: Data generation and pre-processing code.
    - `README.md`: Information and instructions about the folder.
  - **dataset_rr**: This folder contains the code for round-reduced ciphers, Speck especially. Pulled and adapted from [1].
  - **attacks-key-recovery**: Folder with notebooks with models for key recovery attacks.
  - **attacks-aes**: Folder with notebooks with models for AES plaintext recovery attacks.

  - **attacks-s-aes**: Folder with notebooks with models for S-AES plaintext recovery attacks.
  - **attacks-speck**: Folder with notebooks with models for Speck plaintext recovery attacks.
  - **saved_models**: Folder with saved models in `.h5` format.
  - `attack_template.ipynb`: Notebook that serves as a template to train neural networks for any type of attack.
  - `pipeline.py`: Training/testing framework implementation and methods.
  - `README.md`: Information and instructions about the folder.

- `README.md`: Information about the project.