Computing fundamentals
Cryptography homework

Tecnológico de Monterrey

Description

In this homework you will apply what you know of classical ciphers and black-box attacks to implement a machine learning based strategy to understand ciphertexts. You will simulate a known-plaintext attack on a system and, by using a neural network, you will replicate the cipher used as well as learn how to decrypt the ciphertext (even without explicitly knowing the key!).

This homework consists of different tasks and, in order to move on you need to complete all the previous ones. Then, failing to complete one of the tasks will invalidate the rest of the tasks, which will seriously affect your grade.

You can solve this homework in teams of, at most, three people. Working in teams is not mandatory, but recommended. Once you finish this homework, submit a Zip file that contains the following:

- A document (properly formatted) with the requested information (described below).
- All the codes and files generated (in the language or languages of your preference). You can use the codes distributed along with this document.

In all the cases, each one of the tasks must be clearly visible in the document and the codes.

Prepare the document
0 points

Prepare an empty document and be sure that your full names and student IDs are visible. You will include more information to this document on future tasks of the homework.

Documentation task 01

Prepare an empty document and be sure that your full names and student IDs are visible.

Although this task has no value for your grade, failing to provide this information will invalidate your submission.

Implementing the cipher

20 points

Remember that you should not rely on the secrecy of the algorithm to state that a system is secure. So, I will explain how the cipher works.

First, take into consideration that for this homework you will only work with binary strings for plaintexts and ciphertexts. On the other hand, keys will be formed by integer numbers. For all the cases, the length of the plaintexts (n) is always equal to the length of the ciphertext (no characters are added or removed during the encryption/decryption process). The keys contain always a number of integers equal to the length of the plaintext. Although the plaintext can be any binary string of length n, the key has one important restriction: it must always be a permutation of the numbers 0 to $n - 1$. If the key fails to satisfy this condition, the system will not work as expected.

To encrypt a plaintext, the following pseudocode is used:

```
Let P be the plaintext
Let C be the ciphertext
Let K be the key
Let n be the length of the plaintext
for (i = 0; i < n; i++) {
     C[i] = P[K[i]];
}
```

Let us analyze this with an example. Let us suppose that n equals 5 in this example. Then, a random binary string to be used as plaintext could be: "01010". To generate a valid key, we will shuffle the numbers 0, 1, 2, 3 and 4 (the numbers from 0 to $n - 1$). Let us suppose that the result of shuffling these numbers is 3, 4, 2, 0, 1, which will be represented as 3-4-2-0-1, from now on. Please look that the result is a permutation

and then, no repeated elements can appear in the list. A graphical representation of the encryption process is depicted in Figure 1.

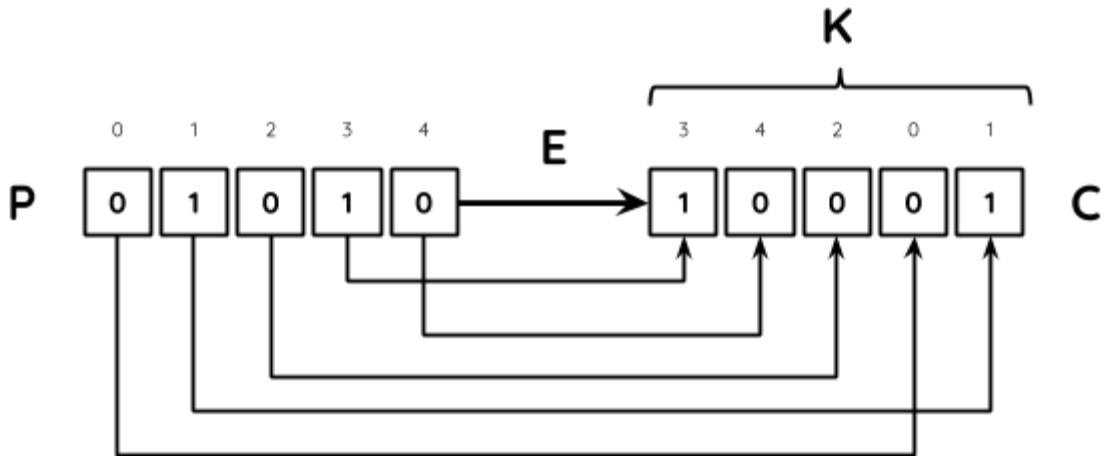

Figure 1. A graphical representation of the encryption process of the plaintext "01010" by using the key 3-4-2-0-1 with the proposed algorithm.

To decrypt a ciphertext, a similar process takes place:

```
Let P be the plaintext
Let C be the ciphertext
Let K be the key
Let n be the length of the ciphertext
for (i = 0; i < n; i++) {
     P[K[i]] = C[i];
}
```

A graphical representation of the decryption process is depicted in Figure 2.

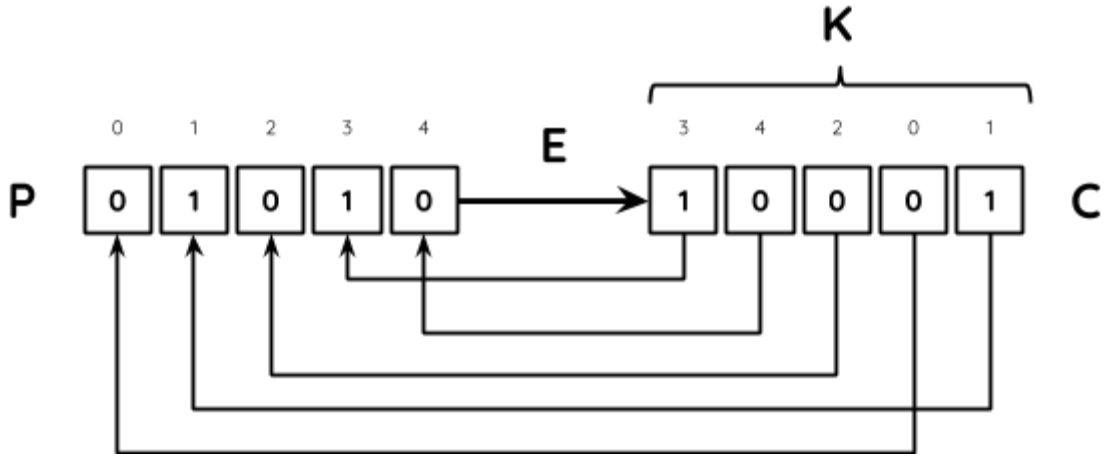

Figure 2. A graphical representation of the decryption process of the ciphertext "01010" by using the key 3-4-2-0-1 with the proposed algorithm.

Programming task 01

Your first programming task is to implement this cipher in the programming language of your preference (both encryption and decryption processes).

Data preparation

20 points

For the rest of this homework you will work with binary strings of length 25 ($n = 25$). The number of different strings you can produce by using this setting is 2^{25} (around 33 million of strings). The number of possible keys, given the constraints imposed by the algorithm is $25!$ (around 1.55×10^{25} different keys).

For this task you will generate a random key (a permutation of the numbers 0 to 24). Please keep this key since you will use it for all the remaining tasks in this homework. From this point on you are not allowed to change the key.

Documentation task 02

Document the key to be used in this homework.

Generate 10 thousand random plaintexts of length 25 and, by using the cipher implemented in the previous task and the key you just generated, obtain the corresponding ciphertexts. Save this information into a file named `data25-Train.csv` and repeat the process to generate another 10 thousand random palintexts and their corresponding ciphertexts. Save this information into a file named `data25-Test.csv`.

Since you will use code I provide, you must comply with the requested format. The file contains no header and each row in the file contains 50 characters separated by commas (25 corresponding to the random plaintext and the remaining 25 corresponding to the ciphertexts). For a reference, look at the file `data25-Sample.csv` distributed along with this document. Of course, this file is provided only as a reference and for this reason it only contains 10 rows (your file will contain 10 thousand rows).

Programming task 02

Be sure to include the files `data25-Train.csv` and `data25-Test.csv` within your Zip file.

Neural networks as ciphers: encryption

25 points

By using the Matlab codes distributed along with this document (or any other codes you prepare), train a perceptron with hard limit activation functions to see if such a neural network is capable of replicating the work of the cipher for the encryption process. In other words, you will train a neural network to receive plaintexts as input and produce the corresponding ciphertext (as it was produced by the algorithm you implemented before).

Neural networks are computing systems somehow inspired by the biological neural networks that constitute animal brains. The basic processing unit is the neuron, which can be seen as an adder: it sums all the inputs and, if the resulting value is larger than a bias, the neuron fires. The way the neuron fires depends on the activation function. Then different activation functions produce different outputs (even under the same inputs and bias).

A neural network is, in few words, a collection of neurons. The code distributed along with this document contains various functions to train and use perceptrons. Please note that the perceptron is one of the most simple neural networks but, given the simplicity of the cipher, it works perfectly for this task. The perceptron contains only one layer of neurons (one neuron per output) and each input is connected to all the neurons.

If you have no previous experience with neural networks, please make an appointment at jcobayliss@tec.mx as soon as possible to give you a quick introduction to the topic.

In order to use the code distributed along with this document you are suggested to consult Table 1, which contains a step by step process to train and test the neural network.

Table 1. Step by step process to train and test the neural network for the encryption process.

Instruction	Purpose
`training = load('data/data25-Train.csv');`	Loads the training data.
`P = training(:, 1:25);`	Selects the columns that contain the inputs for training. The first 25
`T = training(:, 26:50);`	Selects the columns that contain the expected outputs for training.
`[W, b] = train(P, T, @hardlim, 5000, 12345);`	Trains the perceptron. This instruction returns both the weight matrix and the biases. This information defines the behaviour of the perceptron. The last argument is the seed for al the random operations.
`testing = load('data/data25-Test.csv');`	Loads the testing data.
`P = testing(:, 1:25);`	Selects the columns that contain the inputs for testing.
`T = testing(:, 26:50);`	Selects the columns that contain the expected outputs for testing.
`accuracy(P, T, @hardlim, W, b)`	Tests the perceptron. The accuracy indicates the percentage of cases where there was no error in the output. In other words, the closer its value to 1, the better the performance of the network.

Programming task 03

By following the steps from Table 1, generate and test 10 different neural networks to encrypt (run the code 10 times by using different seeds to train the networks). Register the accuracy of each of the runs.

Documentation task 03

- Present the accuracy of the ten runs. You can use a table or a plot for this task.
- What can you conclude from the results? Is it possible to replicate the behaviour of this cipher (to encrypt) by using a neural network?

Neural networks as ciphers: decryption

25 points

By using the Matlab codes distributed along with this document (or any other codes you prepare), train a perceptron with hard limit activation functions to see if such a neural network is capable of replicating the work of the cipher for the decryption process. In other words, you will train a neural network to receive ciphertexts as input and produce the corresponding plaintext (as if it was decrypted by the algorithm you implemented before).

Please remember that, in the previous task, the inputs were the plaintexts and the outputs were the ciphertexts. This time, all you need to do is to change the order: the inputs for the neural network will be the ciphertexts and the outputs will be the plaintexts. You can also use Table 1 as a reference for this task but remember to consider the change of the input and output order when P and T are generated.

Programming task 04

Generate and test 10 different neural networks to decrypt (run the code 10 times by using different seeds to train the networks). Register the accuracy of each of the runs.

Documentation task 04

- Present the accuracy of the ten runs. You can use a table or a plot for this task.
- What can you conclude from the results? Is it possible to replicate the behaviour of this cipher (to decrypt) by using a neural network?

Neural networks as ciphers: where is the key?

10 points

At this point you might be wondering what happened to the key and how it is possible for the neural network to replicate the behaviour of the cipher without even knowing the key. The answer is simple: the key is coded within the neural network. Now, can you figure out how to extract it?

For this task, you are requested to train a neural network to work as the cipher described before and, once trained, you will extract the information of the key used for the process. Of course, it would not be a challenge if you already knew the key used. For this reason, you will find the key used on the files **data25-Train-Challenge.csv** and **data25-Test-Challenge.csv**. These files are distributed along with the Matlab codes.

Documentation task 05

- Write the key used for encryption/decryption of files **data25-Train-Challenge.csv** and **data25-Test-Challenge.csv**. Please note that you are not requested to write code for this task (but you can do it if you think it will help you).
- Clearly describe how you found out the key based on the information in the weight matrix.