Lab 1 Team Atlanta

Bardi Sara, Friso Luca, Kawas MHD Nazeer, Mari Daniele

November 2020

1 Task 1 & 2

In order to solve task 1 a Python class was implemented with re-usability concept in mind, in particular since the sub-key generation function and the round function are arguments of the constructor it becomes very easy to solve task 5 and 7 since just the new round function is needed.

These functions were implemented using the *numpy* library, since it provides efficient and convenient methods to deal with arrays (in this case binary ones). The only challenge in these two tasks is due to the indexes starting from 0 in python as opposed to the function definitions where they start from 1. This can be solved by manually computing the indexes, subtracting 1 from each of them and finally finding the corresponding *numpy* slice.

A general function was implemented both for encryption and decryption since the only difference between the two is that in the latter the sub-keys are fed in reverse order.

2 Task 3

The easiest way to solve task 3 is to define matrix C as the identity and to focus just on finding matrix A and matrix B . To compute these two matrices it is possible to consider the cipher as a black box and to extract A and B using the procedure described in *appendix 1* in the instructions PDF. It is possible to see the matrices obtained with this approach in figure 1a

3 Task 4

To complete Task 4 it is necessary to compute the inverse of matrix A found previously. For binary matrices it can be found using the formula $A^{-1} = A^{*}det(A)mod2$ (where A^{*} is the inverse in the real field) as specified in the *appendix 2* of the instructions PDF. After this, K can be easily obtained using as $k = A^{-1}(x + Bu)$. The key that was used to encrypt the message-cipher couples in the dataset was K=0xD091BB44

4 Task 5 & 6

The round function to be implemented to solve task 5 is

$$w_i(j) = \begin{cases} y_i(j) \oplus \{k_i(4j-3) \wedge [y_i(2j-1) \vee k_i(2j-1) \vee k_i(2j) \vee k_i(4j-2)]\}, & \text{if } 1 \leq j \leq l/2 \\ y_i(j) \oplus \{k_i(4j-2l) \wedge [k_i(4j-2l-1) \vee k_i(2j-1) \vee k_i(2j) \vee y_i(2j-l)]\}, & \text{if } l/2 < j \leq l \end{cases}$$

Also in this case the only difficulty is finding the correct indexes for python, but by using the procedure described for task 1 it is easy to implement the correct round function.

```
[0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0
 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0
 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0
 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1
 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1
 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0
 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1
 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1
 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0
 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1
 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1
 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0
 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0
 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0
 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0
 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0
 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0
 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1
 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0
 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0
 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0
 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0
 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1
 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0
 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0
 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1
 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1
 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0
 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1
 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0
 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0]
```

```
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

(a) A,B matrices

The non linearity is given by the *or* operation between bits of the sub-keys and of the intermediate ciphers, it is possible to see though that this part of the expression is evaluated to 1 unless all of its member are 0. Assuming a uniform distribution over the bits values the latter happens with probability $\frac{1}{16}$. Furthermore, even if this expression was wrongly evaluated to 1, the result could still be correct if the other member of the AND operation was equal to zero. So approximating the non linear part of the function to 1 allows us to obtain a cipher that is linear and that approximates well enough the one defined in task 5. It is easy to notice that the newly defined round function is in fact the same as the one in task 1. So the matrices A, B (C is the identity), that represent the cipher with round function equal to that used in task 1, can be used to linearly approximate the non linear cipher. This means that it is possible to find k with a probability far greater than $\frac{1}{2^{32}}$ (random guessing). Finding an analytical expression of the probability $P[Ak \oplus Bu \oplus Cx = 0]$ is far from trivial but it can be estimated with the following procedure

- Randomly sample messages $u_i; i = 1, 2, \dots, n$ and keys $k_i; i = 1, 2, \dots, n$
- Compute the ciphers $x_i = f_{k_i}(u_i)$
- Given x_i, u_i use A and B to compute $\hat{k}_i$
- Compare k_i and $\hat{k}_i$, if they are equal increase a counter

If n is big enough the ratio between the final value of the counter and n should represent a good estimate of the probability $P[Ak \oplus Bu \oplus Cx = 0]$. In this case using n = 100000 it is possible to see that this value is more or less 0.13 that is far bigger than the random guessing probability. Using this procedure though it was not possible to find the correct key for the data-set provided for the lab, but probably by slightly increasing the number of couples obtained with the KPA attack this algorithm would find it. Since it is possible to notice that many times the guessed key is very close (Hamming distance wise) to the correct key, it was worth trying to see if a key at distance 1 from the guessed one was actually the correct guess but this still didn't work.

5 Task 7 & 8

The implementation strategy for the round function needed for task 7 is the same as in task 1 and 5.

In order to solve task 8 the algorithm described in appendix 3 of the instructions PDF was implemented. In order to compute the success probability lets call $\hat{K}'$ the set of the guesses of k' and $\hat{K}''$ the set of the guesses of k'', here we can see that

$$P[success] = P[k' \in \hat{K}' \wedge k'' \in \hat{K}''] = P[k' \in \hat{K}']P[k'' \in \hat{K}'']$$

since the two events are independent. If the guesses are sampled with replacement from a uniform distribution over the keys space we get that this probability is:

$$P[k' \in \hat{K}']P[k'' \in \hat{K}''] = (1 - P[k' \notin \hat{K}'])(1 - P[k'' \notin \hat{K}'']) = \left(1 - \big(1 - \frac{1}{|\kappa|}\big)^{N'}\right)\left(1 - \big(1 - \frac{1}{|\kappa|}\big)^{N''}\right)$$

where κ is the key space, $N' = |\hat{K}'|, N'' = |\hat{K}''|$. For semplicity lets choose $N' = N''$ then we finally have

$$P[success] = P[k' \in \hat{K}' \wedge k'' \in \hat{K}''] = 1 + \big(1 - \frac{1}{|\kappa|}\big)^{2N'} - 2\big(1 - \frac{1}{|\kappa|}\big)^{N'}$$

It is possible to see that with $N' = 2^{15}$ the success probability is close to 15% and the complexity of the algorithm is much lower than the complexity of the brute force approach so by repeating multiple times the algorithm it is possible to find the correct keys. At this point given the message-cipher couples $(u_1, x_1), (u_2, x_2), \ldots$ some candidates key couples are selected using the meet in the middle attack on message u_1 and cipher x_1, all the wrong candidates are then removed by checking if they also correctly encrypt $u_i, i = 2, 3, \ldots$.

In particular for the provided KPA dataset it is possible to see that the keys that were used to encrypt the messages were $k' = \text{0xD091}, k'' = \text{0xE7E1}$