Lab1 RoC Information Security Report

Group: TCP

Cavasin Saverio - ID. 2044731 Cosuti Luca - ID. 2057061
De Faveri Francesco Luigi - ID. 2057069

A brief description of the report

All the tasks were implemented in Python and verified to prove correctness. For each cipher-related task, we verified that our Encryption and Decryption algorithms were correct through symmetry analysis. I.e. encrypting the plaintexts and computing them back after decryption.

Task 1-2: Encryption/Decryption

The code was tested using the inputs given in the lab slides.
The code of these tasks is divided into mainly three parts:

1. Variables definition.
2. Functions definition (key generation, sub-key sum, substitution, transposition, and linear transformation) and inverse function definition.
3. Encryption/Decryption results assessment.

Task 3: Cipher vulnerabilities

In Task 3, we decided to use the following matrices:

$$A = \begin{pmatrix} 9 & 0 & 1 & 6 & 0 & 0 & 1 & 10 \\ 0 & 8 & 6 & 2 & 2 & 9 & 0 & 0 \\ 0 & 6 & 0 & 8 & 3 & 10 & 0 & 0 \\ 6 & 0 & 0 & 8 & 0 & 1 & 6 & 6 \\ 2 & 0 & 1 & 10 & 0 & 0 & 1 & 3 \\ 0 & 1 & 8 & 4 & 9 & 6 & 0 & 0 \\ 0 & 10 & 0 & 5 & 7 & 6 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 1 & 4 & 8 \end{pmatrix} \qquad B = \begin{pmatrix} 6 & 0 & 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 6 & 3 & 0 & 0 & 3 & 0 & 0 \\ 0 & 3 & 6 & 0 & 0 & 0 & 3 & 0 \\ 3 & 0 & 0 & 6 & 0 & 0 & 0 & 3 \\ 5 & 0 & 0 & 0 & 4 & 0 & 0 & 8 \\ 0 & 5 & 0 & 0 & 0 & 4 & 8 & 0 \\ 0 & 0 & 5 & 0 & 0 & 8 & 4 & 0 \\ 0 & 0 & 0 & 5 & 8 & 0 & 0 & 4 \end{pmatrix}$$

The two matrices were computed solving the linear system:

$$AK + Bu = x = \text{Encryption}(u, k)$$

Solving with:

1. $K = \mathbb{I}$ and $u = \mathbf{0}$, to find the matrix A
2. $K = \mathbf{0}$ and $u = \mathbb{I}$, to find the matrix B

Task 4: Linear cryptanalysis

Our guess for $\hat{K}$ with respect to the plaintext/cyphertext is:

$$\hat{K} = [8, 10, 2, 7, 6, 4, 9, 1]$$

To find this key, and thus to compute the modular inverse matrix A^{-1}, we used the online software, Online Matrix Calculator to avoid numerical errors given by the `np.linalg.det` and the $\mod 11$ when computing the determinant of matrix A. Indeed while implementing different strategies with functions that performed the modular inverse, the main concern with this kind of task is the machine rounding issues. A wrong rounding or a forced flooring/ceiling can provide incorrect approximations and even create (nonexistent) singular matrices.

Task 5: "Nearly linear" simplified AES-like cipher

Task 5 again, followed the same logic and correctness assessment of tasks 1-2. After the requested cryptosystem was implemented, we performed the suggested cryptography and confirmed the functioning of the process. Moreover,

we tested the composition of the properties during encryption and decryption to ensure the appropriate symmetry and the system's reliability.

Task 6: Linear cryptanalysis of a "nearly linear" cipher

Unfortunately, we were not successful in implementing task 6. We tried different approaches to find a "close key", we tried to compute the linear approximation table from the s-box given the requested substitution function, but we were unsuccessful to understand how to implement it completely. Our strategy was leading to the usage of the bias of the boolean functions that can be built to use the "Piling-Up Lemma", by considering the inputs and outputs of the s-box as random variables. However, we missed understanding how to do it practically in our setup, given the modular field we are working on and the structure of our cryptosystem.

Task 7: Nonlinear simplified AES-like cipher

This time, we were able to use a function to compute the multiplicative inverse of a vector in the ciphering/deciphering phase while performing the substitution. Therefore, using the proper code structure as before, we successfully implemented the cryptosystem.

Task 8: Meet in the middle attack

Our guesses $\hat{K}$', $\hat{K}$" for the keys we used to encrypt the KPA pairs are:

$$\hat{K}' = \big[10, 7, 2, 2\big] \qquad\qquad \hat{K}'' = \big[5, 4, 6, 0\big]$$

We created two dictionaries: the first one associates *key_1, ciphertext* and the second *key_2, deciphered text*, and we checked for which couple *key_1, key_2* we had a collision. We worked with 2 sets of ~ 10000 randomly generated keys, and we were able to find a match via brute force.