University of Padua
Department of Information Engineering

Information Security Report
Laboratory Session 1

Implementation and linear cryptanalysis of a Feistel cipher

Author:
Luca Badin
Arturo Bellin
Karakuchi Chidananda Nikhil
Simone Favaro
Filippo Gianbartolomei
Fawad Ul Haq

Teacher:
Nicola Laurenti

5th November 2020

Solution

Our solution of laboratory 1 is entirely implemented using Python. In particular we made use of the Numpy library in order to easly manipulate vectors and quickly compute operations beetween them. The solution is composed by three python files: `main.py` contains all the encryptors implementation and attack results, `attack.py` contains functions necessary for the attacks accomplishment, and `hexutils.py` contains two functions for string to binary numpy vector conversion and vice versa.

Task 1

We implement all the Feistel encryptors using the same function called `encrypt`, which takes different input parametes and functions based on the cypher type (*linear*, *nearly linear* and *non linear*): they are the plaintext `u`, the initial key `k`, the number of rounds `r`, the message length `l` and the round function `f`.
In fact, even if those may differ for each implementation, the way in which the three transformation are computed is always the same: they are *substitution*, *linear tf* and *transposition*.
Concerning the subkey generation and the round functions, `key_gen` takes in input the initial key `k` and the specific round `i` in order to output the subkey for that round, while there are three different round functions, one for each Feistel encryptor.

As far the linear Feister encryptor riquired in task 1, we just used the `encrypt` function with the linear round function `lin_f`, which uses the round subkey `k`, the round ordinality `i` and the `y` input to compute the so called `w` value.

Task 2

Since in Feistel cyphers the encryptor and decryptor are the same but the order in which the subkeys are used inside the round functions, we made a `decrypt` function which works with the same parameters used in `encrypt`, just swapping plaintext `u` with cyphertext `x`, where the subkeys order used for the round function computations is reversed. In this way we preserved the `f_lin` function the way it is avoiding the rewrite of a very similar code.

Task 3

In order to find the two matrixes A and B which indentify the linear relationship for the linear Feister cypher, we create the `find_mat` function, which uses the encryption function `encrypt`, the round function `f` and the message length `l`. The way it works is just an implementation of the methodology presented in *Appendix 1* of the lab instructions, using numpy arrays.

Task 4

The linear cryptanalysis KPA against the linear Feister cypher is done using the `find_key_kpa` function, which takes as input the two matrixes we found in task 3 `a`, `b` and the plaintext-cyphertext pair `u`, `x`. The function uses numpy arrays and numpy operations in order to quickly retrive $k = A^{-1}(x + Bu)$. Moreover, the A^{-1} computing is done following the *Appendix 2* of the lab instructions.

To carry out the cryptanalysis on the five (u, x) pairs inside the document provided called `KPApairsVanouver_linear.hex`, we firstly used the `find_mat` function to find the A and B matrixes (must be the same for all five pairs since the encryption method is the same). They are:

$$A = \begin{bmatrix}
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
\end{bmatrix}$$

$$B = \begin{bmatrix}
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
\end{bmatrix}$$

Then, using `find_key_kpa` function for each pair, we convince ourself that the guessed key is correct, since it is always the same and it maps each plaintext to the corresponding cyphertext. That is $\hat{k} = k =$`55D9F224`

Task 5

Concerning the nearly linear Feistel cypher, we just use the `encrypt` function with a different round function implementation, called `near_lin_f`. Same thing for `decrypt`.

Task 6

topkek.txt

Task 7

Similar to task 5, the non linear Feistel cypher implementation required just a different round function called `non_lin_f`. The `encrypt` and `decrypt` function stays the same.

Task 8

We implemented the Meet in the middle attack against the concatenation of two non linear Feistel cyphers using an approach based on *Appendix 3*, of the lab instructions.
The `meet_in_the_middle` function takes as input parameters the cardinalities of random guesses `n1`, `n2` for $\hat{k'}$ and $\hat{k''}$, encryption and decryption functions `enc`, `dec`, the plaintext-cyphertext pair `u`, `x`, the round function `f` and the length of the message `l`.
After generating the k' and k'' random guesses using the numpy *random.randomint* function (which create random binary arrays of length `l`), for each guess $\hat{k'}_i$ we store inside a list the couples $(\hat{k'}_i, \hat{x'}_i = E'_{\hat{k'}_i}(u))$. Same thing is done for couples $(\hat{k''}_i, \hat{u''}_i = D''_{\hat{k''}_i}(x))$. Then we use numpy function *intersect1d* to retrieve all possible matches between the $\hat{x'}_i$ and $\hat{u''}_j$, so that the corresponding keys pairs $\hat{k'}_i$ and $\hat{k''}_j$ are stored in a list and returned by the function.

In order to get the key pair guess regarding the KPA attack on the five plaintex-cyphertext pairs in the `KPApairsVancouver_non_linear.hex` document, we initially ran the function for only the first (u, x) pair, using really high `n1` and `n2` parameters, finding some matches. Since at the end of the day we just need the one more probable guess, we trimmed all the matches, checking if a certain matching key pair worked also for some of the other four (u, x) pairs provided in the document: this was done checking if $x = E'_{\hat{k}'_i}(E''_{\hat{k}''_j}(u))$.
Clearly, if a key pair works for each of the five (u, x) pairs, it is most likely the correct guess (even though it is still not certain), so when we found a $(\hat{k'}_i, \hat{k''}_j)$ that satisfies that last requirement we assume that $\hat{k'}_i = k'_i$ and $\hat{k''}_j = k''_j$. In particular, our guess is $\hat{k'}_i = k'_i =$ `30C0` and $\hat{k''}_j = k''_j =$ `564D`.