Assignment 5

COMP 7401 - Feistel Cipher with CBC

Filip Gutica - A00781910
Shahin Nikvand - A00809275

Table of Contents

Introduction

The purpose of this assignment is the explore the Feistel cipher and also to help us understand how DES and Triple DES function. It teaches us about multi round ciphers but also the importance of the algorithm and method used to encrypt data for easy encryption and decryption with the right key but making it near impossible to reverse.

Constraints

We will be limiting our development to the following:

- Python as the development language
- 8 Rounds of the feistel cipher
- Using the same key for all the rounds for ECB
- Do subkey generation for CBC
- Being able to encrypt and decrypt a few paragraphs of plain text

Action Plan

Development Plans are as follows:

1. Create a simple “scramble” function which will hide the data while following the rules of the feistel cipher and be easy enough to encrypt and decrypt
2. Execute the encrypt by hand on paper to verify the validity of the approach and that indeed is reversible and functional
3. Write Pseudo of how we initially intend to execute the idea
    a. Please note that our actual code does differ from this but this was the initial approach to the application

Pseudo Code

This is our initial plan of approach to writing the application. As we continued our development it resulted in slight changes from this to better follow along with the code.

Additionally we are assuming that efficiency and speed are not top priority so we will be using array which will cause additional data. The garbage collector should handle the removal of the data but this is a statement that there are better ways to execute this code.

Encryption

Assumptions

- 8 rounds
- 64 bit blocks (8 bytes)
- Use sub key generation for CBC

Formulas:

1. f(x, k) = [(2i * k)^x] % (2^32 -1)
2. Li = Ri - 1
3. Ri = L[i-1] XOR f(R[i-1], ki)

```
Function Encrypt(Plaintext, key)
        Key initial = key
        Initialize empty string Ciphertext
        For every 64 bit block of the plaintext
                Initialize array L[8]
                Initialize array R[8]
                Set L[0] to first 32 bits of plaintext block
                Set R[0] to second 32 bits of plaintext block
                For i = 1 to max rounds
                        L[i] = R[ i- 1]
                        If mode is cbc
                                If first round
                                        Key = key initial
                                Else key = subkeygen(L[i], key initial)
                        R[i] = L[i - 1] XOR Scramble(R[i - 1], i, key)

                // Add together the final results of each L and R (Lfin and Rfin)
                //Append to the Ciphertext
                Ciphertext += (L[8] + R[8])

        Return Ciphertext

Function Scramble(x, i, k)
        Return ((xi * k)^i) % (2^32-1)

Function SubkeyGen(s1, s2)
        sha256(s1 + s2)
```

Decryption

```
Function Decrypt(Ciphertext, Key)
        Key_initial = key
        Initialize empty string
        For every 64 bit block of the ciphertext
                Array L[8]
                Array R[8]
                L[0] first 32 bit
                R[0] second 32 bit
                For i = 8 to 1
                        If mode is cbc
                                Key = subkeygen(L[i], key_initial)
                                If i is 1
                                        Key =  key_initial
                        R[i+1] = L[i]
                        L[i+1] = R[i] Scramble(L[i ], i, key)
                Ciphertext = L[8] + R[8]
        Return ciphertext


Function Scramble(x, i, k)
        Return ((xi * k)^i) % (2^32-1)

Function SubkeyGen(s1, s2)
        sha256(s1 + s2)
```

Instructions

We have two modules: feistel.py and feistel-decrypt.py

Feistel.py is run using python version 2.

To encrypt a file please run:

Feistel.py -e -m <cbc,ecb> -t <plaintext file> -k <key> -o <ciphertext file>

Feistel-decrypt.py is run using python version2.

To decrypt a file please run:

Feistel.py -d -m <cbc,ecb> -t <ciphertext file> -k <key> -o <resulting plaintext file>

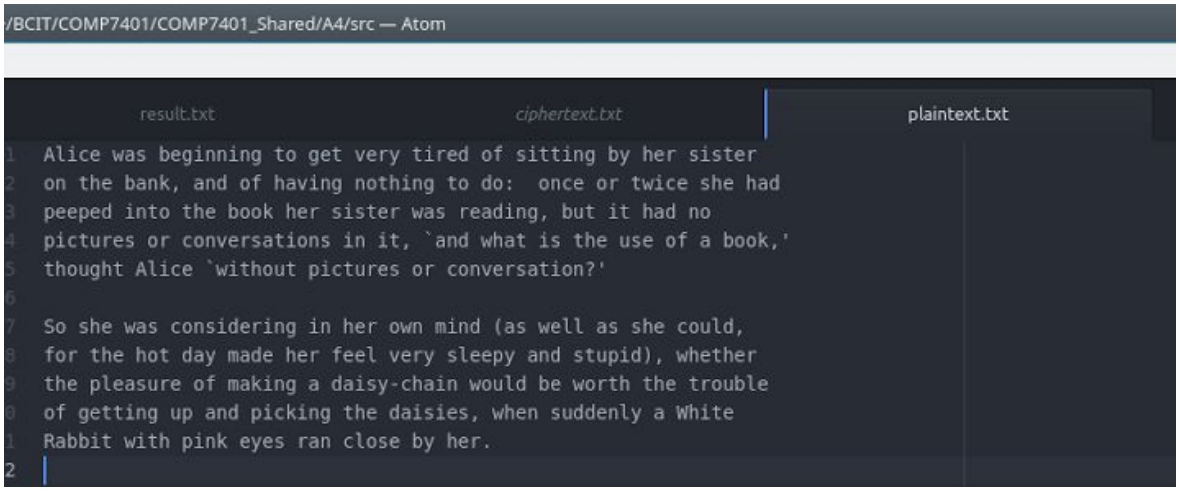
Please make sure you have the input files for both feistel.py and feistel-decrypt.py in the same directory as the script.

Testing

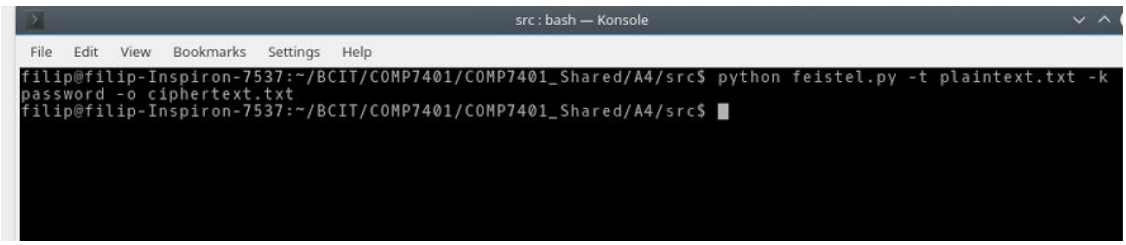
Test	Test Case	Expected Result	Actual Result	Pass/Fail
1.	Run fesistel.py with key: password, on file plaintext.txt with output file ciphertext.txt	Run with no errors, produce ciphertext file: ciphertext.txt	Ran with no errors. Produced ciphertext file: ciphertext.txt	Pass
2.	Run feistel-decrypt.py with key: password, on file ciphertext.txt with output file result.txt	Run with no errors, produce the original plaintext to result.txt	Ran with no errors, produce the original plaintext to result.txt	Pass
3.	Run feistel-decrypt.py with wrong key: passwordp, on file ciphertext.txt with output file result.txt	Run with no errors, produce result.txt file still encrypted	Run with no errors, produce result.txt file still encrypted	Pass

Plaintext used:

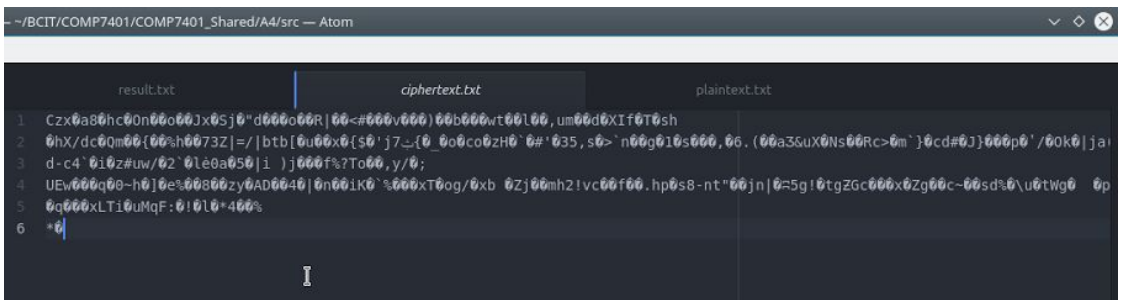
Test 1



Run festeil.py

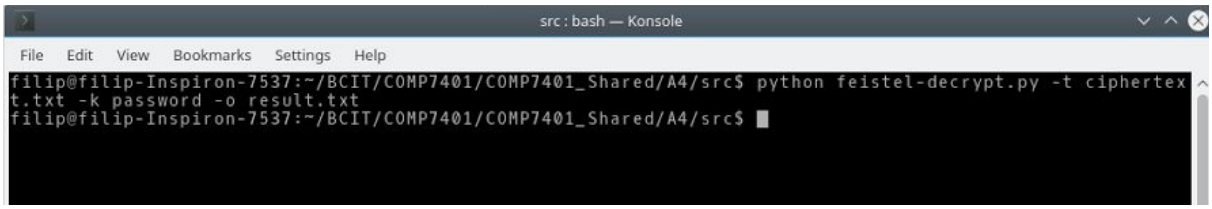


No errors. Resulting Ciphertext:

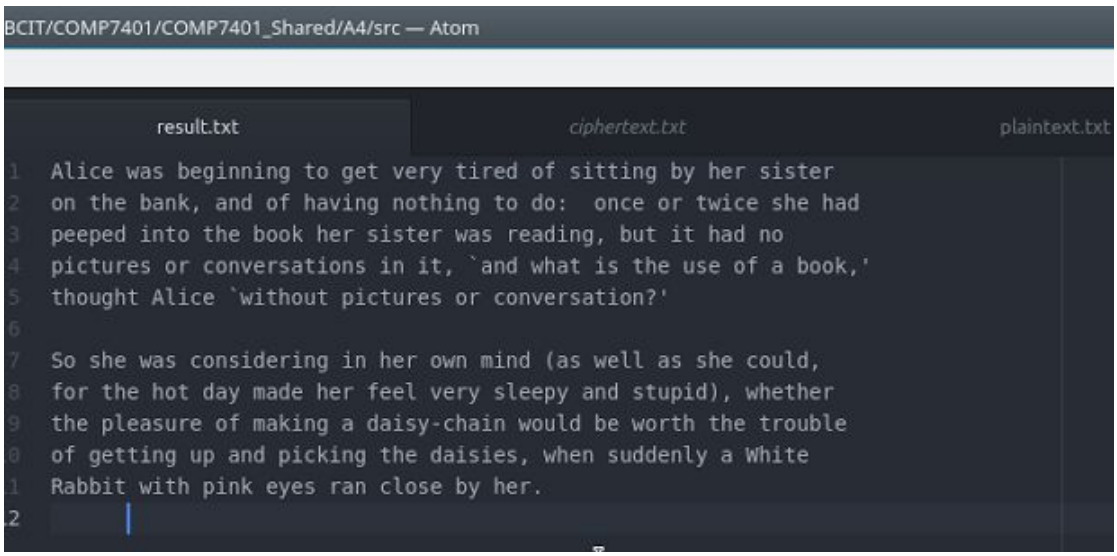


Test 2

Run feistel-decrypt.py

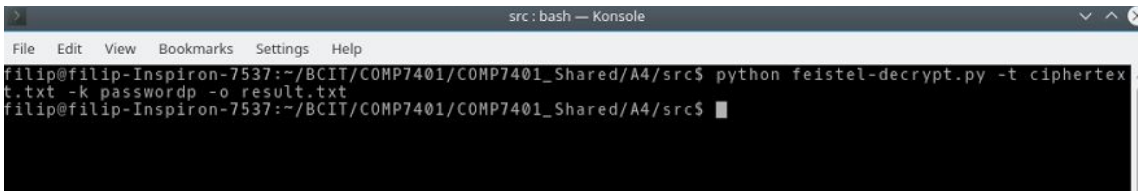


No Errors, Resulting plaintext:

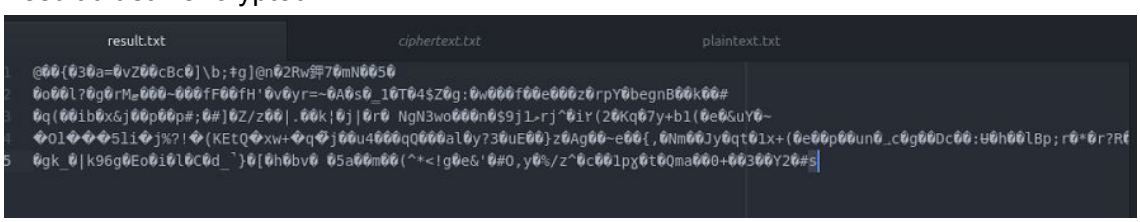


Test 3

Run decrypt with wrong password



Result.txt still encrypted:



Summary

Writing the code for the feistel cipher made us realize how brilliant yet simple the cipher is. It’s incredible to see how you have achieve such complexity and security with such a simple code. CBC is very powerful and provides great security at the loss of error propegation.