COMP 7402
Assignment 4

Rina Hong
Renato Montes

2019-03-05

Table of Contents

User Guide

General notes

- To run the program, please install python3.
- Please include the file extension when using file input, e.g. filename.txt
- Keyboard input type can be entered as kb or keyboard.
- File input type can be entered as for file.

Keyboard input

- If keyboard input is chosen, the program will prompt you to enter a plaintext. Once you enter the plaintext, the program prints an analysis of every round of encryption.
- Usage:
    - python3 feistel.py <inputType:{kb,keyboard}> -k <key:(integer)>
- Example:
    - `python3 feistel.py kb -k 1359876`

File input

To encode:

- Usage:
    - python3 feistel.py <inputType:{f,file}> -k <key:(integer)> -i <inputPlaintextFilename> -o <outputCiphertextFilename> -e
- Example:
    - `python3 feistel.py f -k 876 -i ptext.txt -o cipher.txt -e`

To decode:

- Usage:
    - python3 feistel.py <inputType:{f,file}> -k <key:(integer)> -i <inputCiphertextFilename> -o <outputPlaintextFilename> -d
- Example:
    - `python3 feistel.py f -k 876 -i cipher.txt -o decrypted.txt -d`

Report

Summary

The assignment was illuminating to cement understanding of Feistel ciphers in ECB mode.

In particular, it was found **confusion is much higher than diffusion**, as the key is used in every round during the encryption of every block, but a flipped bit-change in the plaintext affects its corresponding block only and does not propagate throughout the ciphertext. This confirms the observation that ECB-mode Feistel ciphers should not be used for longer messages.

It was also found that with the key function (used in every round) as currently implemented, eight rounds are not enough to be able to flip every bit at least once. A key function with a better distribution of produced 1-bits for XOR flipping could probably be written.

Diffusion vs Confusion

The diffusion in an algorithm that produces a ciphertext refers to the relationship between plaintext and its ciphertext. This can be examined by making slight one-bit changes to the plaintext and seeing how much the ciphertext changes while providing the same key.

Four pairs of texts were chosen to look at diffusion. Within each pair, the texts differed by only one bit. The files can be found in the data/dumps/diffusion_files/ directory of this assignment. The results obtained are shown in the following table.

Effect of changing one bit in the plaintext			
Plaintext pair	Key	Plaintext length / Ciphertext length	Difference between ciphertexts in the pair in number of bits
in1.txt, in2.txt different first byte	888	6 bytes / 9 bytes	32 bits
in1.txt, in3.txt different last byte	888	6 bytes / 9 bytes	31 bits
in4.txt, in5.txt different first byte	888	726 bytes / 729 bytes	34 bits (!)
in4.txt, in6.txt different last byte	888	726 bytes / 729 bytes	30 bits (!)

The results speak volumes of the great security problem of electronic codebooks: they should not be used to encrypt long messages because of their deterministic nature where different blocks are independent of each other.

The plaintexts of the last two rows in the table are 120 times longer than those in the first two rows, but the difference in bits in the ciphertexts after changing one plaintext bit is about the same! This is because only one block is affected: the block where the bit is changed.

Let us now turn to confusion. Confusion refers to the relationship between a key and the ciphertext. This can be examined by changing the key one bit and examining the ciphertexts produced with the same plaintexts.

Two pairs of texts were chosen, one longer than the other. Within each pair, the key was altered so that the binary representation of the key would differ by only one bit. The files can be found in the data/dumps/confusion_files/ directory of this assignment. The results obtained are shown in the following table.

Effect of changing one bit in the key			
Plaintext	Key	Plaintext length / Ciphertext length	Difference between ciphertexts in the pair in number of bits
in1.txt	4	6 bytes / 9 bytes	30 bits
in1.txt	5	6 bytes / 9 bytes	
in3.txt	4	726 bytes / 729 bytes	2908 bits (!)
in3.txt	5	726 bytes / 729 bytes	

The verdict is clear: ECB-mode Feistel encryption has significantly better confusion than diffusion.

This is what should naturally be expected, as the key is used in the encryption of every single plaintext block, therefore a change in one bit of the key should affect every block of the plaintext as it gets encrypted into the ciphertext.

Progressive bit flipping

After implementing the Feistel cipher in ECB-mode, it was found that more and more bits are flipped each round, but progressively less bits in each round. It was found that eight rounds are too few to flip every bit as currently implemented.

The key function used in each round to produce the mix-in that is XOR'd with half a block may be to blame. Experimentally, with the current function it takes a greater number rounds to start hitting complete flipping of all 64 bits. It is probably possible to write a function that produces at least one positive bit for bit-change in every position of the 64-bit block after less rounds.

The following table shows the progression of bit positions flipped at least once as a plaintext is encoded into its ciphertext in eight rounds. The program runs can be found in the data/dumps/bitflip/ directory.

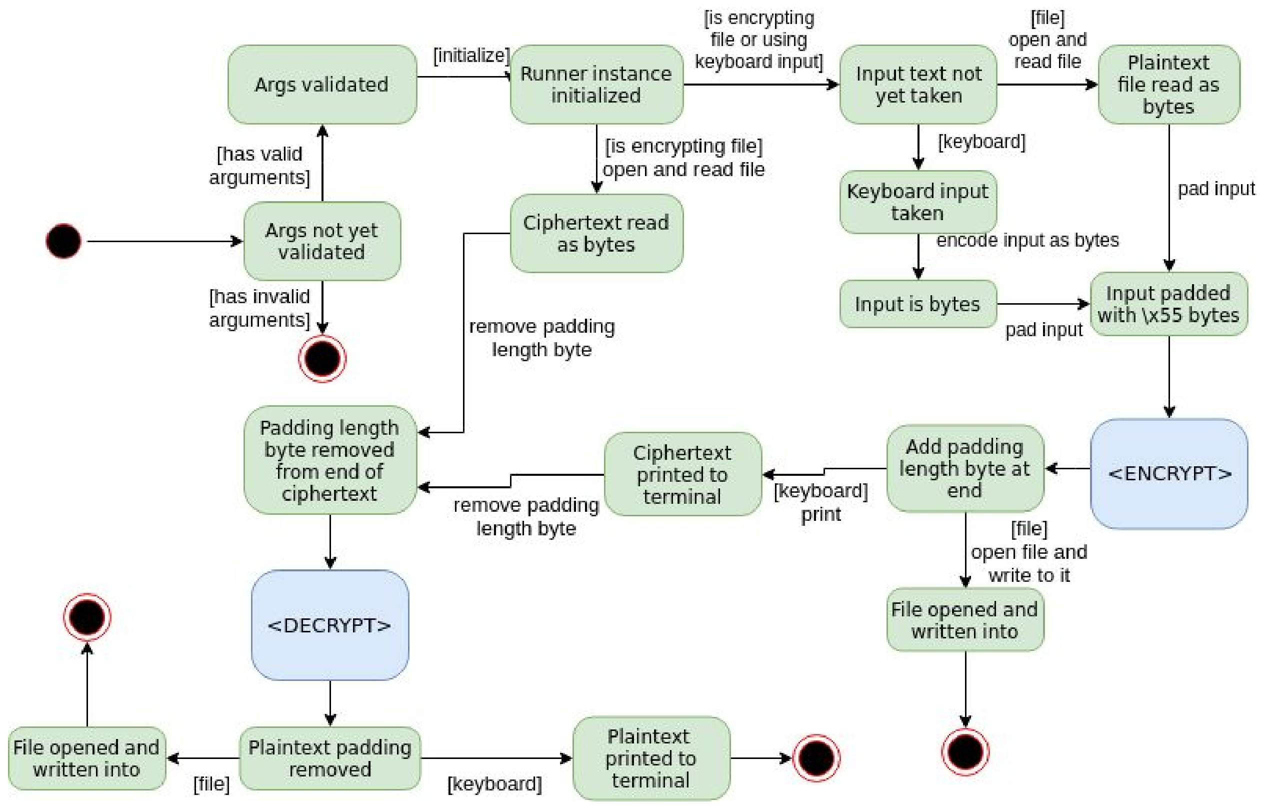
Bits flipped at least once during encryption up to a given round				
Round	Plaintext: folly Key: 6	Plaintext: follx Key: 6	Plaintext: folly Key 7	Plaintext: folly Key: 87654321
1	14	18	18	18
2	32	32	36	33
3	41	38	36	40
4	46	45	45	40
5	51	47	49	43
6	59	53	51	43
7	60	53	51	47
8	61	54	52	55

There is significant variation in the number of bit positions that get flipped to the opposite value at least once, depending on the plaintext and key provided. However, as it can be seen in the second column, it is sometimes possible to flip past 60 of the 64 bit positions.

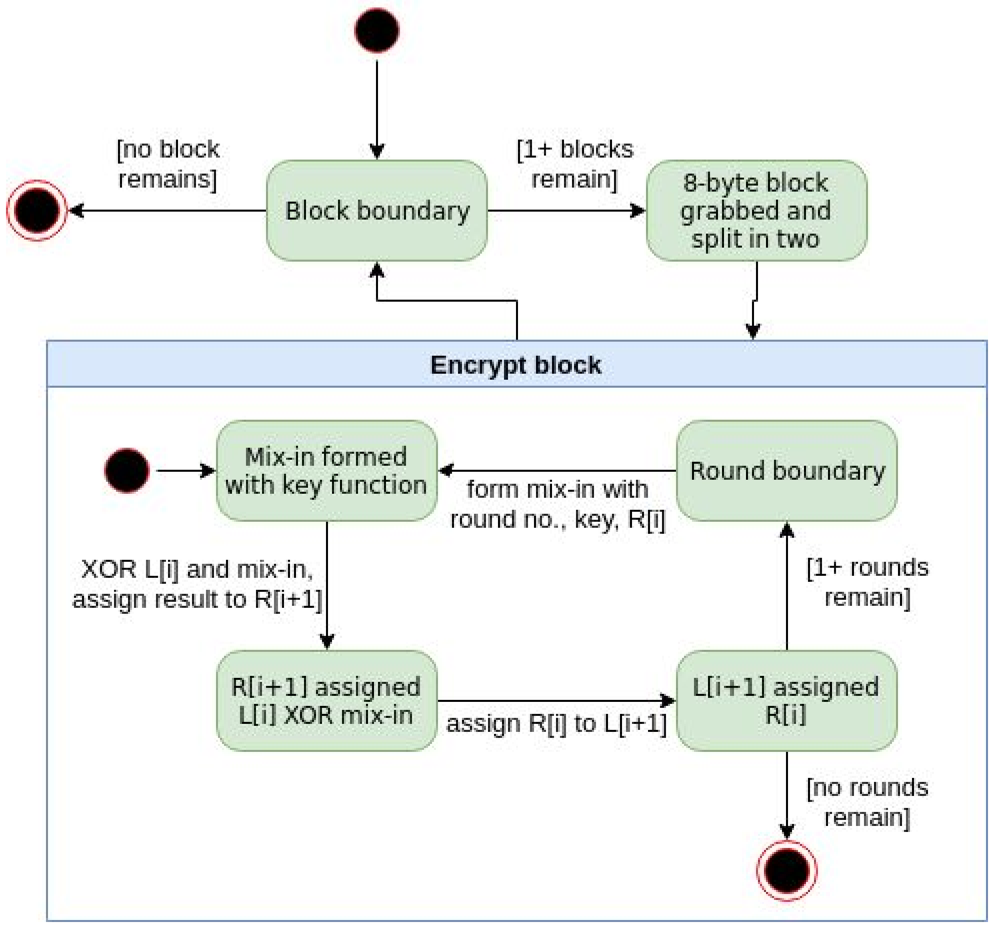
Design

State diagrams

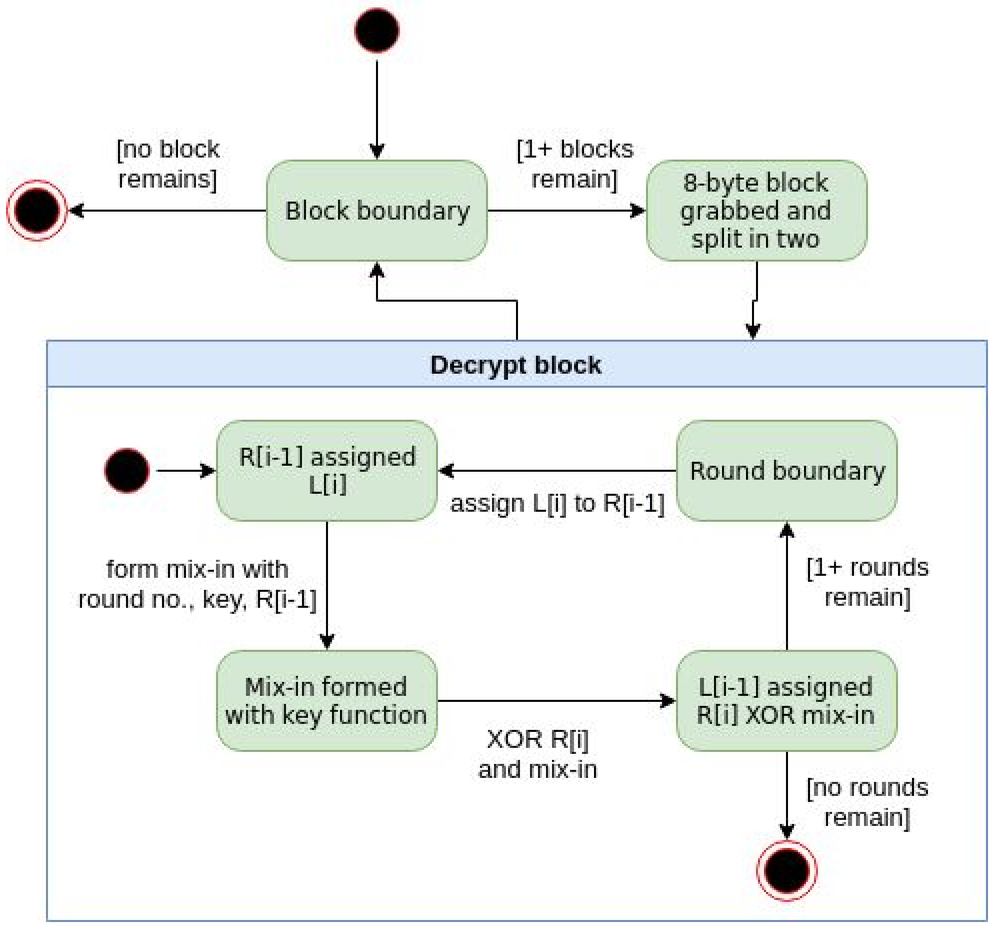
Overview


Args validated
[initialize]
Runner instance initialized
[is encrypting file or using keyboard input]
Input text not yet taken
[file] open and read file
Plaintext file read as bytes
[has valid arguments]
[is encrypting file] open and read file
[keyboard]
Keyboard input taken
pad input
Args not yet validated
Ciphertext read as bytes
encode input as bytes
Input is bytes
Input padded with \x55 bytes
pad input
[has invalid arguments]
remove padding length byte
Padding length byte removed from end of ciphertext
Ciphertext printed to terminal
Add padding length byte at end
<ENCRYPT>
remove padding length byte
[keyboard] print
[file] open file and write to it
<DECRYPT>
File opened and written into
File opened and written into
Plaintext padding removed
Plaintext printed to terminal
[file]
[keyboard]

<ENCRYPT>


[no block remains]
Block boundary
[1+ blocks remain]
8-byte block grabbed and split in two
Encrypt block
Mix-in formed with key function
form mix-in with round no., key, R[i]
Round boundary
XOR L[i] and mix-in, assign result to R[i+1]
[1+ rounds remain]
R[i+1] assigned L[i] XOR mix-in
assign R[i] to L[i+1]
L[i+1] assigned R[i]
[no rounds remain]

<DECRYPT>



File header

```
# feistel.py
#
# Assignment 4, COMP 7402, Winter 2019.
#
# Usage: feistel.py [-h] [-k KEY] [-i INPUT] [-o OUTPUT] [-e] [-d] inputtype
#
# Example uses:
#   python3 feistel.py kb -k 7
#   python3 feistel.py f -k 64820 -i in.txt -o out -e
#   python3 feistel.py f -k 64820 -i out -o decrypted.txt -d
#
# Date: 2019-03-02  implement file input
#       2019-03-03  implement keyboard input, argument validation
#       2019-03-05  add thorough comments to functions
# Designer: Rina Hong, Renato Montes
# Programmer: Rina Hong, Renato Montes
```

Pseudocode

Main

```
obtain command-line arguments
validate command-line arguments
if arguments are not valid, **end** program
initialize object that runs the Feistel algorithm with the arguments
if input type is keyboard,
    **encrypt input**, then **decrypt input**
if input type is file and action requested is encryption,
    **encrypt input**
if input type is file and action requested is decryption,
    **decrypt input**
**end** program
```

Encrypt input

```
get plaintext input (whether a file or something typed with the keyboard)
ensure plaintext is formatted in bytes
pad input so that number of bytes is multiple of 8 following Feistel cipher rules
while plaintext remains,
    grab block of 8 bytes of plaintext, then **encrypt block**,
    then save block to in-memory ciphertext and move onto the next block
add a byte with the padding length at the end
print ciphertext to file (if using a file) or terminal (if using keyboard input)
```

Encrypt block

```
initialize an array of 64 mutable booleans representing bit positions in the block
split 8-byte block into two halves (“left” and “right”)
for a specific number of rounds,
    store the right half value for later use,
    then use the round number, key argument and right half to generate a number,
    then XOR this generated number with the left half,
    then assign this new XOR’d number as the right half,
    then assign the saved old right half value to the left side,
    then show a short analysis of the bits that changed according to boolean array
return encrypted block
```

<u>Decrypt input</u>

```
get ciphertext input (whether a file or from the previous encryption run if using keyboard input)
take last byte out of the ciphertext and save its value containing the plaintext padding length
while ciphertext remains,
    grab block of 8 bytes of ciphertext, then decrypt block,
    then save block to in-memory plaintext and move onto the next block
remove the padding at the end of the plaintext
print plaintext to file (if using a file) or terminal (if using keyboard input)
```

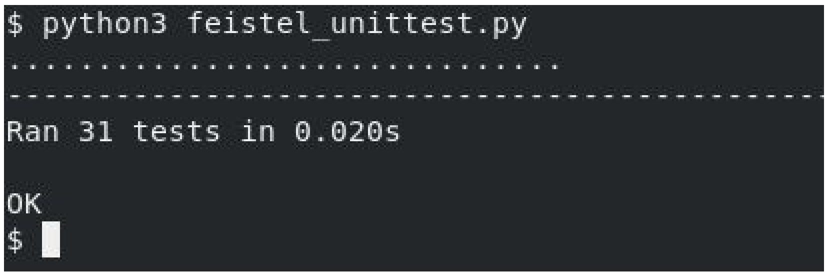
<u>Decrypt block</u>

```
split 8-byte block into two halves (“left” and “right”)
for a specific number of rounds,
    store the right half value for later use,
    then assign the left half value to the right half,
    then use the round number, key argument and new right half to generate a number,
    then XOR this generated number with the saved old right half value,
    then assign this new XOR’d number as the left half,
return decrypted block
```

Testing

Unit testing

- 31 unit tests were written. All of them are passing at the time of handing in the project.
- To run them, feistel.py and feistel_unittest.py must be in the same directory.

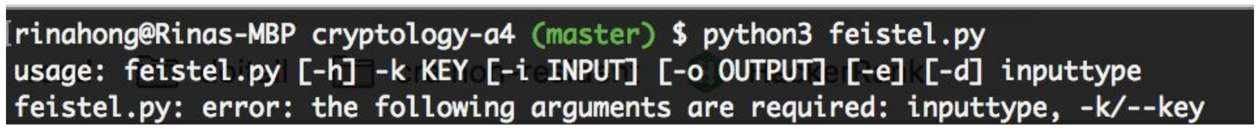


```
$ python3 feistel_unittest.py
...............................
----------------------------------------------------------------------
Ran 31 tests in 0.020s

OK
$
```

Blackbox testing

test: invoke program with no arguments	expected: validation kicks in, a message is shown, and the program exits	actual: [same as expected]

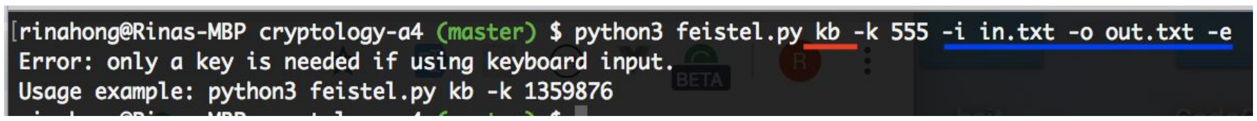
Screenshot:



```
[rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py
usage: feistel.py [-h] -k KEY [-i INPUT] [-o OUTPUT] [-e] [-d] inputtype
feistel.py: error: the following arguments are required: inputtype, -k/--key
```

test: (smoke test) When user selects keyboard mode and enter more arguments than required	expected: validation kicks in, a message is shown mentioning the problem, and the program exits	actual: [same as expected]

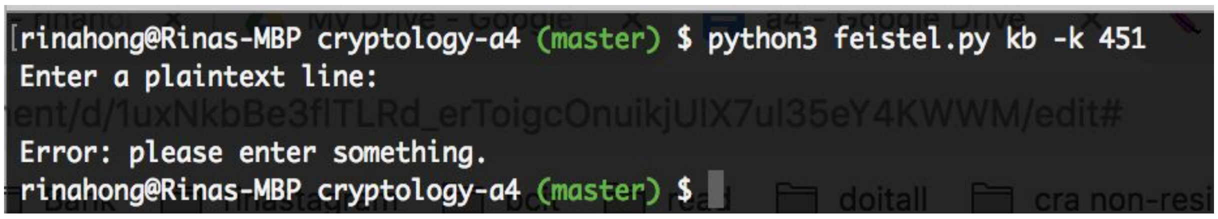
Screenshot:



```
[rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py kb -k 555 -i in.txt -o out.txt -e
Error: only a key is needed if using keyboard input.
Usage example: python3 feistel.py kb -k 1359876
```

test: (smoke test) When user selects keyboard mode and enters an empty string	expected: Display an error message and exit the program	actual: [same as expected]

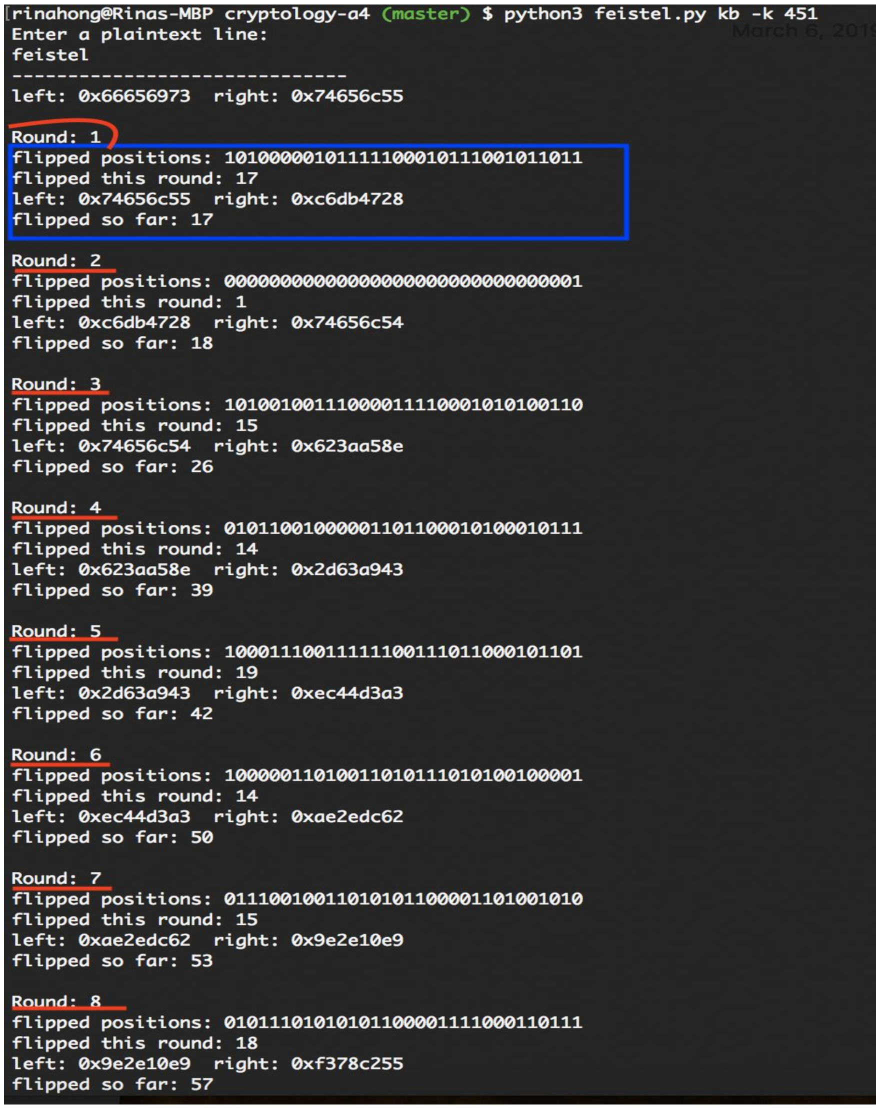
Screenshot:



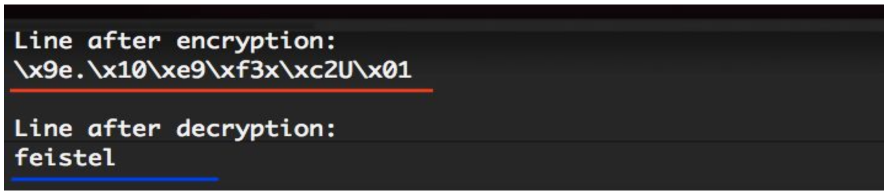
```
rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py kb -k 451
Enter a plaintext line:

Error: please enter something.
rinahong@Rinas-MBP cryptology-a4 (master) $ 
```

test: (smoke test) When user selects to keyboard mode and enters a plaintext line into the program correctly	expected: Display 8 rounds of encrypting report of each 64 bits block of the plain text. - Flipped positions - Flipped this round - Left, right - Flipped so far - Result from encryption Result from encryption, decryption	actual: [same as expected]

screenshot:



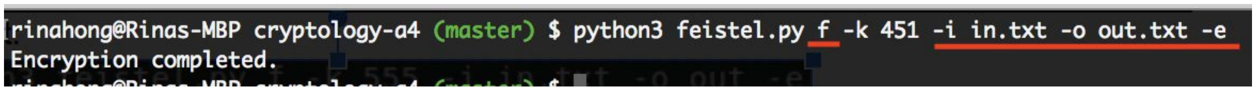
```
[rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py kb -k 451
Enter a plaintext line:
feistel
------------------------------
left: 0x66656973  right: 0x74656c55

Round: 1
flipped positions: 101000001011111100010111001011011
flipped this round: 17
left: 0x74656c55  right: 0xc6db4728
flipped so far: 17

Round: 2
flipped positions: 00000000000000000000000000000001
flipped this round: 1
left: 0xc6db4728  right: 0x74656c54
flipped so far: 18

Round: 3
flipped positions: 10100100111000011110001010100110
flipped this round: 15
left: 0x74656c54  right: 0x623aa58e
flipped so far: 26

Round: 4
flipped positions: 01011001000001101100010100010111
flipped this round: 14
left: 0x623aa58e  right: 0x2d63a943
flipped so far: 39

Round: 5
flipped positions: 10001110011111100111011000101101
flipped this round: 19
left: 0x2d63a943  right: 0xec44d3a3
flipped so far: 42

Round: 6
flipped positions: 10000011010011010111010100100001
flipped this round: 14
left: 0xec44d3a3  right: 0xae2edc62
flipped so far: 50

Round: 7
flipped positions: 01110010011010101100001101001010
flipped this round: 15
left: 0xae2edc62  right: 0x9e2e10e9
flipped so far: 53

Round: 8
flipped positions: 01011101010101100001111000110111
flipped this round: 18
left: 0x9e2e10e9  right: 0xf378c255
flipped so far: 57
```



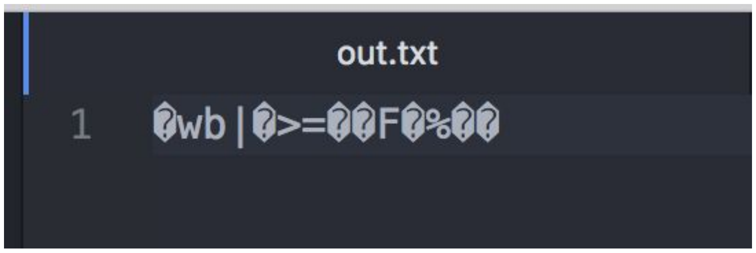
```
Line after encryption:
\x9e.\x10\xe9\xf3x\xc2U\x01

Line after decryption:
feistel
```

test: When user selects file mode with valid arguments correctly to encrypt a file	expected: message announcing successful encryption shown	actual: [same as expected]

screenshot:



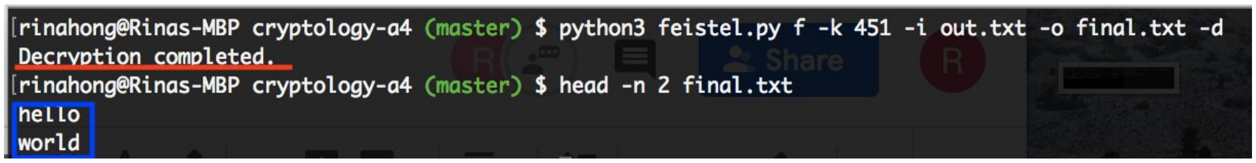
```
[rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py f -k 451 -i in.txt -o out.txt -e
Encryption completed.
```



```
out.txt
�wb|�>=��F�%��
```

test: When user selects file mode with valid arguments correctly to decrypt a file	expected: message announcing successful decryption shown, and the plaintext can be read	actual: [same as expected]

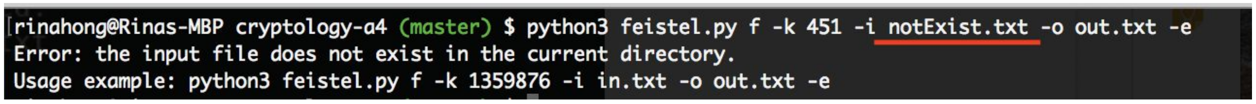
screenshot:



```
[rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py f -k 451 -i out.txt -o final.txt -d
Decryption completed.
[rinahong@Rinas-MBP cryptology-a4 (master) $ head -n 2 final.txt
hello
world
```

test: When user selects file mode with a non-existing file	expected: validation kicks in, a message is shown mentioning the problem, and the program exits	actual: [same as expected]

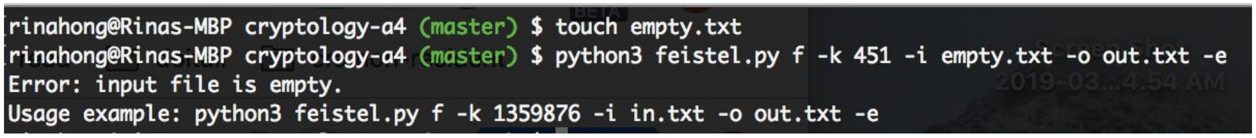
screenshot:



```
[rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py f -k 451 -i notExist.txt -o out.txt -e
Error: the input file does not exist in the current directory.
Usage example: python3 feistel.py f -k 1359876 -i in.txt -o out.txt -e
```

test: When user selects file mode with an empty file	expected: validation kicks in, a message is shown mentioning the problem, and the program exits	actual: [same as expected]

screenshot:



```
rinahong@Rinas-MBP cryptology-a4 (master) $ touch empty.txt
rinahong@Rinas-MBP cryptology-a4 (master) $ python3 feistel.py f -k 451 -i empty.txt -o out.txt -e
Error: input file is empty.
Usage example: python3 feistel.py f -k 1359876 -i in.txt -o out.txt -e
```