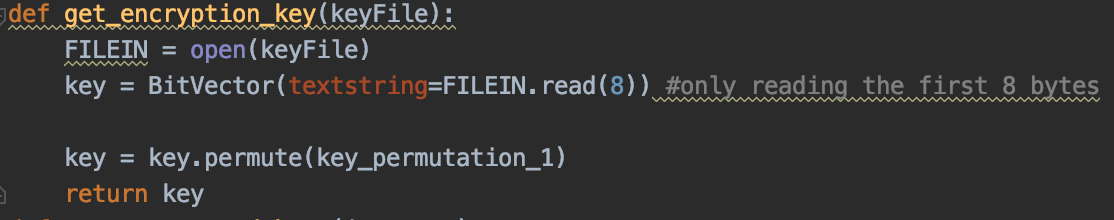
HW02 Summary

**Problem # 1:**

Code:

Note: substitute, get\_encryption\_key, and generate\_round\_keys lightly (or not) modified

Given Functions:



A screen shot of a computer

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Encrypt function:

A screenshot of text

Description automatically generated

Decrypt function:

A screenshot of a cell phone

Description automatically generated

Test Code:

A screenshot of a cell phone

Description automatically generated

Result: (output from encryption, and decryption)

A screenshot of a cell phone

Description automatically generated

<Hexstring file after encryption>

A close up of text on a black background

Description automatically generated

<Decrypted file after decrypting the above hexstring file>

Code Summary: (explanation)

After ensuring that there are enough input arguments and the second is either “-e” or “-d”, the script will begin by assigning the string of input files to their own variables. Assuming that we begin by encrypting a file, each of these variables will be passed through to the encrypt function where the encryption key and the round keys are found through two different dedicated functions. The “get\_encryption\_key” function simply reads the key file and permutes the retrieved bit-key so that it returns a 56-bit key. And the “get\_round\_keys” function takes the 56-bit key as a parameter and does an operation on it (splits, rotates, combines, and permutates) 16 different time to return a list of 16 different round keys.

Next, the plaintext file is opened as a file-object within a bit vector, so that it can be iteratively read in a while-loop until the bit-vector gets to the end of the file. Inside this while-loop, blocks of 64 bits are read at a time. These blocks are checked for being 64 bits (if not they’re padded), then split into a left-half and right-half, and the right half proceeds through a sequence of operations (expansion, XORed w/ round key, substitution, permuted, and XORed w/ the left-half). The result of these operations is then assigned to the next left-half block for the next round, while the unaltered right-half is then assigned to the next right-half block (Note: This process still happens after the last round, but it is immediately reversed). And each block is writte directly to the output file as a hexstring after it is encrypted.

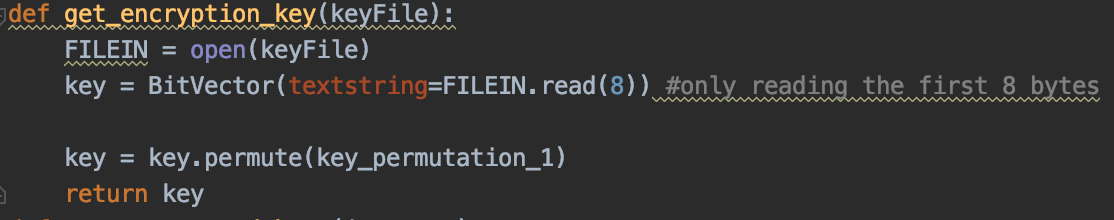
In decryption, the process is the same up until the time of reading from the file. To read from the file I decided to open it with a file pointer to help in facilitating reading the file as a hexstring with the bit-vector object. Because of the previous difference, the decrypting uses a for loop to iterate through the encrypted bit-vector (rather than iterating through the file). Further, the decryption is the same process as the encryption. And when the block is decrypted it it writte to the output file as a text string.

**Problem # 2:**

Code:

Note: substitute, get\_encryption\_key, and generate\_round\_keys lightly (or not) modified

Given Functions:



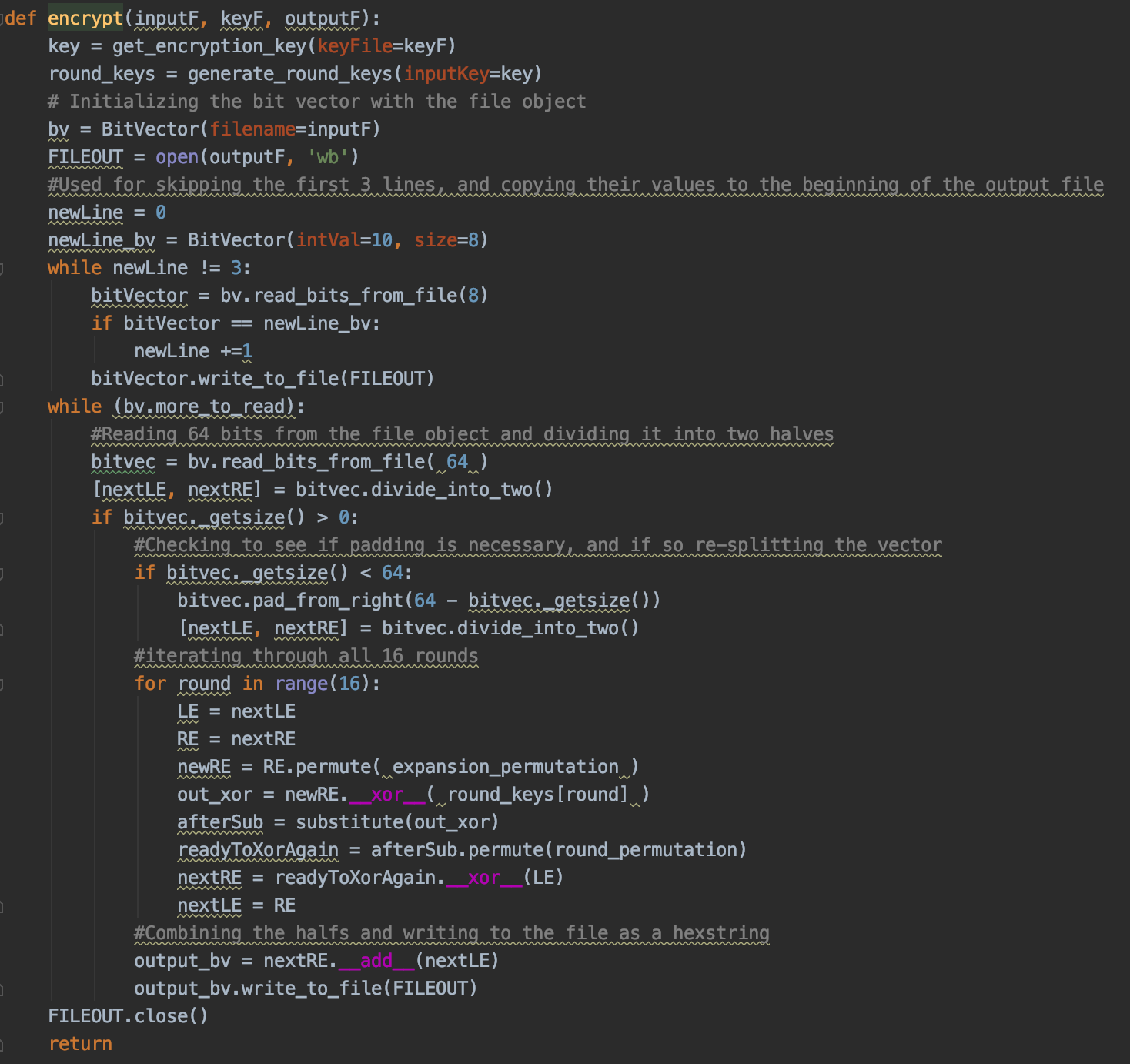
A screen shot of a computer

Description automatically generated

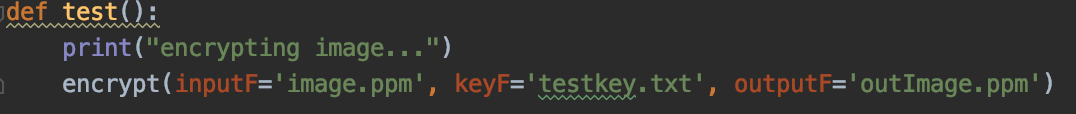
A screenshot of a cell phone

Description automatically generated

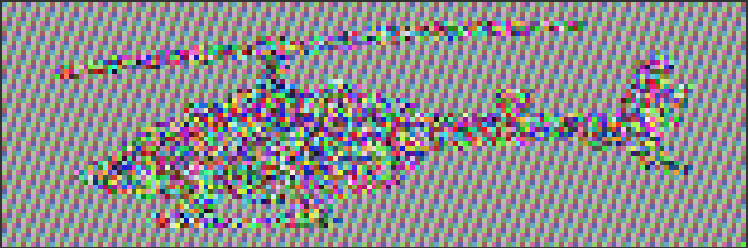
Encrypt function:



Test Code:



Result: (output from encryption, and decryption)



<Image after encryption>

Code Summary: (explanation)

The encryption of the image is largely the same as the process for a text file. The only differences were:

1. The output file is opened as a binary file, to allow binary writing
2. The input file specially iterated 3 lines to copy the ppm header to the output without encrypting it
3. The bit-vector object (after encryption) is directly used to write the output file