COP 4521 Spring 2021
Homework - 4

Total Points: 20
Due: Wednesday 03/31/2021

1 Objective

The objective for this assignment is to make sure

- You are familiar with the principles of Computer Security and several important ciphers.
- Given the description of the Security Algorithm, you can implement the Data Encryption Standard and RSA.
- You are familiar with the way data is stored in computers and can work with bitwise data.

2 General Rules

Python has a `cryptography` and a `des` module, and several third party libraries that implement the DES. However, the objective of this assignment is to make sure you understand the DES and RSA algorithms by implementing them yourself. To that end, you are restricted to the standard Python libraries, and are not allowed any of the Python Cryptography libraries or other third party libraries.
Also, the cryptographic algorithms have to work efficiently and close to the hardware storage of data to have any hope of working without taking a noticeably long time. To this end, your solution should works with the data bits, instead of storing each data bit in an integer in the form of a list or using string formatting or any other built-in library that uses internal lists. Doing so will result in a loss of 4 points.

3 DES (10 points)

3.1 Specifications

- Write a function called `encrypt`. This function will take 2 arguments, the plaintext and the key, and return the ciphertext (1 points).
- Write a function called `decrypt`. This function will take 2 arguments - the ciphertext and the key and return the plaintext (1 points).
- Write a function called `DES`. This function will take 2 arguments - a number and a key. It will them implement one iteration of DES, that includes the following:
    1. Key Scheduling using PC-2. PC-2 is a method of making a new 48 bit key from a 56 bit key using a predetermined sequence. For each round, do a cyclic left shift of the 56-bit key, and then apply PC-2. (1 points)
    2. Perform an initial permutation of the 64 bit text. (1 points)
    3. Apply 16 iterations of the round function. Each round consists of
        (a) Splitting the number into 32 bit halves.
        (b) The left half for the next round is the right half.
        (c) Then, apply an expansion permutation to the right half to expand it to 48 bits. (1 points)

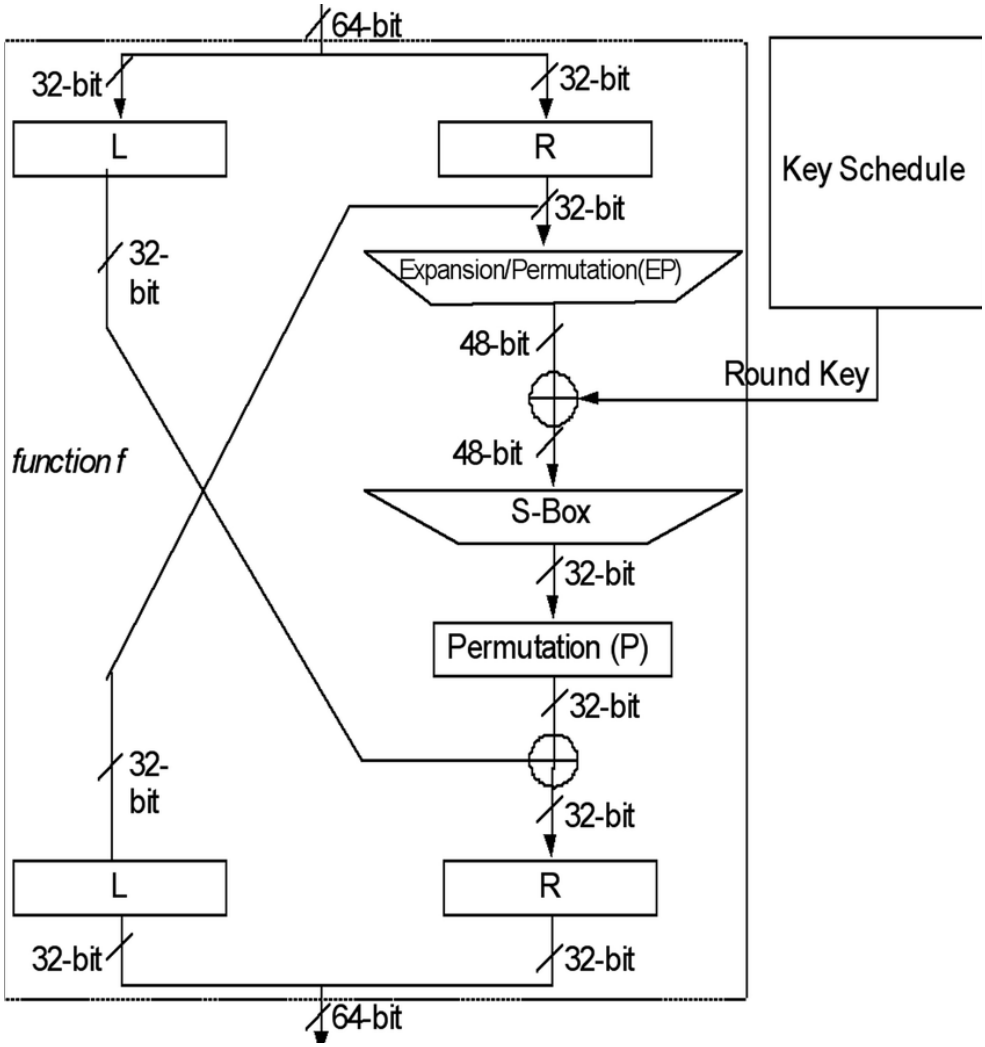
(d) XOR with the round key

(e) Use an S-box (given on slide 17 of the DES slides) to shrink it back down to 32 bits. (1 points)

(f) Use an intermediary permutation

(g) Finally, XOR with the left half to get the right half for the next round. (1 points)

The block diagram below shows the sequence of operations in each round:



4. Finally, exchange the left and right halves one last time and then perform a final permutation.
5. The details of the initial, final, intermediary permutations and the PC-2 for key scheduling can be found here: `https://en.wikipedia.org/wiki/DES_supplementary_material`
   We will not be using PC-1. We will use only 1 S-box instead of 8. We will only circular left shift the key by 1 bit (as a 56 bit value) for all rounds. We will not be using key scheduling.

- Both the `enrcypt` and `decrypt` functions should divide the text into 8 character (64 bit blocks) and then construct a 64 bit integer using the concatenated ASCII values of the substrings. Pad with 0's if the substring is not 8 characters long. (1 points). For example,
  "Hello Wo" Should become 0100100001100101011011000110110001101111001000000101011101101111

- Apply the `DES` function to each block. For each value returned by the `DES` function, separate each 64 bit number into 8 characters and concatenate them into a string using the 8 bit numbers as ASCII values. Then, put the substring together to get the result string. This should be done for both encryption and decryption.
- Return the result string. (1 points)
- In the main function, seed the random number generator using the current system time. (1 points)
- Ask for strings from the user. For each string, generate a 56 bit key, call the functions and print the ciphertext, and the decrypted plaintext. Repeat until the User types "Exit".
- The input will always be ASCII strings.
- Pad with 0's if the string length is not a multiple of 8. This would help on the decryption end, since you do not have to manually insert the end of string character.
- Make sure you follow Python coding conventions and add comments to your code.

3.2 Sample Run

```
DES Implementation:
Enter text to encrypt ("Exit" to quit): This is a sample DES test
Encrypted text:  'zè¢.J.òÃ.Õ#³×Ý..á.ÂM.x7ñ.Ö98f'
Decrypted text:  'This is a sample DES test'
Next text ("Exit" to quit): SmittyWerbenJeagerManJensen. He was number 1
Encrypted text: 'w....v"µ˜ôï.? T.ó.Ö.l.F¹n e* r0'
Decrypted text: 'SmittyWerbenJeagerManJensen. He was number 1'
Next text ("Exit to quit"): Exit
```

4 RSA (5 points)

The Rivest–Shamir–Adleman system is a simple public key cryptosystem that is based on prime numbers. Here's a very basic outline of how to encrypt using RSA

1. Generate 2 prime numbers `p` and `q`. `N = p*q`. `N` is the modulus.
2. The Totient Function λ(N) = (p-1)*(q-1).
3. Choose an integer `e` such that 1<e <$\lambda(N)$ and GCD(e,$\lambda(N)$) = 1.
4. Choose an integer `d` such that `e * d mod` λ`(N) = 1`. You need to use the Extended Euclid Algorithm to determine `d`
5. Now, you can encrypt a message by using c = m^e mod N
6. You can decrypt the ciphertext, using m = c^d mod N

You can find more information of RSA here: `https://en.wikipedia.org/wiki/RSA_(cryptosystem)`

4.1 Specifications

You will be writing an RSA encryption/decryption system that conforms to the following specifications.

- Write a class called `RSA` that will contain all of the functions in the program. In the constructor for `RSA` initialize an empty list and set variables `e` and `d` to 0. (1 points)

- Write a function called `inputFunc` that reads in the number of entries from the user. Then, read in those many values and add them to the list.
- Write a function called `printFunc` that takes in a number and prints "message is " followed by the number. (1 points)
- Write a generator function called `primeGen` that will take a minimum value as a parameter and then yield the next prime numbers. Please note that this input parameter will be quite large and you might want to use `long` if you're using Python 2. (1 points)
- Write a function called `keyGen`. This function will read in a minimum value from the user. Then, it will use the `primeGen` generator to get the next 2 prime numbers and generate the value `N` and the keys `e` and `d`. Print `e` and `N` but not the other values. (1 points)
- You would probably also need to write helper functions for the Lowest Common Multiple (LCM), Greatest Common Divisor (GCD) and the Totient function. (1 points)
- Write a function called `encrypt` that takes in a number as a parameter and returns the RSA encrypted value of the number. (1 points)
- Write a function called `decrypt` that takes in an encrypted number as a parameter and returns the RSA decrypted value. (1 points)
- Write a decorator function for `printFunc` that will print "The encrypted " before the printed message. (1 points)
- Write another decorator function for `printFunc` that will print "The decrypted " before the printed message. (1 points)
- Write a function called `messages` that calls `inputFunc` and `keyGen` and then, uses an iterator to iterate through the list and encrypts each of the numbers using the `encrypt` function. Store the results in another list. Then, go through the second list and print each encrypted number using the decorator for the encrypted message. (1 points)
- Verify your results be decrypting each of the encrypted messages and checking if you get the old value back. Print the decrypted values using the decorator for the decrypted message. (1 points)
- In main, create an RSA object and call the `messages` function.

4.2 Sample Output

This section shows the sample output for the program. Please note that the exceptions raised when the iterator hits the end of the list is not included here, since I checked for the exception while generating the output. Please make sure you do so as well. Also, you might have different values even if you have the same inputs, depending on how 'e' is chosen. As long your decrypted message matched your initial input, you should be fine.

We will also be testing for fairly large inputs, this is just a trivial example for demonstration. We will be testing for both large numbers of input and large values for N. While time is not a major concern, your program reproducing the given message is.

```
Enter the number of messages: 5
Enter the messages:
15
97
201
49
```

```
500
Enter the minimum value for the prime numbers: 20304
N is 413105621
e is 286049
The encrypted message is 106412658
The encrypted message is 367869867
The encrypted message is 393299060
The encrypted message is 70473606
The encrypted message is 409030235
The decrypted message is 15
The decrypted message is 97
The decrypted message is 201
The decrypted message is 49
The decrypted message is 500
```

5 General Instructions

- Add a comment with your name, FSUID, the due date and the line "The program in this file is the individual work of <your name>" on all files. Not having this in each of your files would result in a loss of 2 points.
- Call your files `DES.py` and `RSA.py`. These are the only files you will be turning in.
- If we have listed a specification and allocated point for it, you will lose points if that particular item is missing from your code, even if it is trivial.
- Your outputs will be different from mine depending on the random key.
- Your program should load and run without issues. Every interpretation error will result in a loss of 2 points each.
- You are restricted to standard Python (built-ins). Use of any other libraries would result in loss of 4 points per library.
- Testing your program thoroughly is a part of writing good code. We give you sample runs to make sure you match our output requirements and to get a general idea of how we would test your code.
- Only a file turned in through Canvas counts as a submission. A file on your computer, even if it has not been edited after the deadline, does not count.
- Please adhere to the Academic Honor Policy.
- The student is responsible for making sure they have turned in the right file(s). We will not accept any excuses about inadvertently modifying or deleting files, turning in incomplete tarballs, or turning in the wrong files.
- **Program submissions** should be done through the Canvas class page, under the assignments tab (if it's not there yet I'll create it soon.) Do not send program submissions through e-mail – e-mail attachments will not be accepted as valid submissions.
- The ONLY files you will submit via Canvas are `DES.py` and `RSA.py`.
- **General Advice** - always keep an untouched copy of your finished homework files in your email. These files will have a time-stamp which will show when they were last worked on and will serve as a backup in case you ever have legitimate problems with submitting files through Canvas. Do this for ALL programs.