COMP 125, Fall 2021
Homework Assignment 2

Due: 23:59, November 29 (Monday), 2021

Alice and Bob are two friends and want to talk to each other privately. Oscar is a bad guy who wants to hear what they say. Keeping their communication private is the job of cryptography systems. One of the oldest but widely used systems is RSA invented by Rivest, Shamir, and Adleman in 1977. The RSA system is based on the assumption that it is easy to multiply two large prime numbers but difficult to factor out the resulting large composite number. The RSA system is still being widely used by modern computers to encrypt and decrypt messages, for example, in electronic commerce protocols. The RSA system is believed to be secure if sufficiently large prime numbers are used. This system has three main steps:

1. **Key generation** generates the public and private keys (e, n) and (d, n). The public key (e, n) is publicly released to everyone (in our example, including the bad guy Oscar). The private key exponent (d, n) should be kept as private (in our example, it should be given to only Bob but not to the bad guy Oscar).

2. **Encryption** encrypts a given message x with respect to the public key (e, n). In our example, Alice encrypts her message using the public key and sends the encrypted message through the public channel (for example through the Internet) so everyone can hear the encrypted message.

3. **Decryption** recovers an encrypted message to the original one with respect to the private key (d, n). In our example, since Bob has this private key, he can easily recover the message. However, although the bad guy Oscar heard the encrypted message in the public channel, he cannot recover and understand it since Oscar does not have the private key.

In this homework assignment, you will implement a simplified version of the RSA system.

You will have the entire program in Part G. However, we will guide you in its design by explicitly asking you to implement its important steps as functions. We will give you the headers of these functions.

*****IMPORTANT: You MUST use these function headers as they are. We will use these headers to call your functions and test them. You are NOT allowed to modify these function headers. If you modify them, you will get no points from the corresponding part since we cannot call your functions properly, and thus, we cannot test and grade them.**

Through Part A to Part F, you will implement these functions. In Part G, you will combine these functions to come up with the entire program. For each function, we provide example outputs to show how the function should work for different inputs. Please test your functions with the provided examples and also additional test cases to make sure that it always works.

Similar to your first homework assignment, you should save the solution of each part in a separate py file. You MUST conform to the following naming convention to give a name to each of these py files. Otherwise, you will lose 5 points each time you are using another file name.

- For Part A, the file name should be **`hw2_part_a.py`**
- For Part B, the file name should be **`hw2_part_b.py`**
- For Part C, the file name should be **`hw2_part_c.py`**
- For Part D, the file name should be **`hw2_part_d.py`**
- For Part E, the file name should be **`hw2_part_e.py`**
- For Part F, the file name should be **`hw2_part_f.py`**
- For Part G, the file name should be **`hw2_part_g.py`**

Please put all these py files in a compressed file (zip, rar, etc) and submit it as a single file via Blackboard.

Please note that, the correctness of your program will, of course, affect your grade. However, in addition to this, the following points are important to get the full credit from each part.

- You MUST use meaningful names for the variables.
- You MUST write explanatory and clearly understandable comments.
- At the beginning of each file, you MUST write your name, your id, and your section as comments. After these, you MUST provide a very brief description about what the program does as additional comments.

For this assignment, you are allowed to use the codes given in the recommended textbooks and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any Python built-in functions except the **`input`**, **`print`**, **`range`**, and type cconversion functions. **Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.**

Before continuing further, please be sure that you carefully read all explanations given above and understood them very well.

PART A: (15 points)

Write two functions called **`is_prime`** and **`is_relatively_prime`** as explained below. You will use these functions in the implementation of the other functions (explained in other parts). You may use the codes given in the lecture slides to implement these functions.

You **MUST** use the following function headers.

```
# This function takes an integer as its parameter and returns True if the
# integer is prime and returns False otherwise. You may assume that its
# parameter is always a positive integer.
def is_prime(number):
    # Write your code here
    # ...


# This function takes two integers (first and second) as its parameters and
# returns True if these two integers are relatively prime and returns False
# otherwise. Remember that two integers are relatively prime if they share
# no common factors other than 1. You may assume that both of the parameters
# are always positive integers.
def is_relatively_prime(first, second):
    # Write your code here
    # ...
```

Below are example outputs.

```
In [1]: runfile('/Users/hw2_part_a.py', wdir='/Users')

In [2]: y1 = is_prime(5)
In [3]: print(y1)
True

In [4]: y2 = is_prime(1501)
In [5]: print(y2)
False

In [6]: y3 = is_prime(19)
In [7]: print(y3)
True

In [8]: r1 = is_relatively_prime(9, 8)
In [9]: print(r1)
True

In [10]: r2 = is_relatively_prime(9, 8)
In [11]: print(r2)
True

In [12]: r3 = is_relatively_prime(28, 49)
In [13]: print(r3)
False

In [14]: r4 = is_relatively_prime(101, 103)
In [15]: print(r4)
True

In [16]: r5 = is_relatively_prime(1501, 1513)
In [17]: print(r5)
True
```

Implement these functions in a file called **hw2_part_a.py**.

PART B: (20 points)

Write a function called **`display_menu`** that displays a menu for the choice of a user and returns the choice. The valid choices and the corresponding return values are

- g or G to generate the public and private keys (the function should return G for both)
- e or E to encrypt a message using the public key (the function should return E for both)
- d or D to decrypt a message using the private key (the function should return D for both)
- x or X to exit (the function should return X for both)

All other options are invalid and the user should be asked to enter her/his choice again until s/he enters a valid choice.

You **MUST** use the following function header.

```
# This function displays a menu for the choice of a user and returns this choice.
# Options are explained above. Please check the given output for the menu format.
def display_menu():
    # Write your code here
    # ...
```

Below are example outputs.

```
In [1]: runfile('/Users/hw2_part_bb.py', wdir='/Users')


In [2]: choice1 = display_menu()
g or G for key generation
e or E for encryption
d or D for decryption
x or X to exit

Please enter your choice: D

In [3]: print(choice1)
D

In [4]: choice2 = display_menu()
g or G for key generation
e or E for encryption
d or D for decryption
x or X to exit

Please enter your choice: X

In [5]: print(choice2)
X

In [6]: choice3 = display_menu()
g or G for key generation
e or E for encryption
d or D for decryption
x or X to exit

Please enter your choice: A

***Invalid option***

g or G for key generation
e or E for encryption
d or D for decryption
x or X to exit

Please enter your choice: encryption

***Invalid option***

g or G for key generation
```

```
e or E for encryption
d or D for decryption
x or X to exit

Please enter your choice: comp 125

***Invalid option***

g or G for key generation
e or E for encryption
d or D for decryption
x or X to exit

Please enter your choice: g

In [7]: print(choice3)
G
```

Implement the program in a file called **hw2_part_b.py**.

PART C: (15 points)

Write a function called **`calculate_n_and_phi`** that asks the user to enter two prime numbers p and q and returns their product and the totient of their product.

Your function should ask the user to enter these two prime numbers p and q one by one and continues asking if a number entered by the user is not prime or a non-positive integer.

The product n is defined as $n = p \cdot q$ and the totient ϕ of this product is defined as $\phi = (p-1) \cdot (q-1)$. Your function should return these two numbers n and ϕ. You will use these numbers to generate the public and private keys as well as to encrypt and decrypt messages.

You **MUST** use the following function header. You may call the functions you implement in Part A. For that you need to import **hw2_part_a.py.**

```
# This function asks the user to enter two prime numbers. It checks whether
# these numbers are prime and positive. If not, it continues asking the user
# to enter valid prime numbers. If both of them are prime, it calculates and
# returns their product as well as the totient of this product. You may
# assume that the user always enters integers (negative or positive).
def calculate_n_and_phi():
    # Write your code here
    # ...
```

Below are example outputs.

```
In [1]: runfile('/Users/hw2_part_c.py', wdir='/Users')


In [2]: n1, phi1 = calculate_n_and_phi()

First prime number: 61

Second prime number: 53

In [3]: print(n1, phi1)
3233 3120

In [4]: n2, phi2 = calculate_n_and_phi()

First prime number: 12

Invalid prime, enter again: -2

Invalid prime, enter again: 101

Second prime number: 108

Invalid prime, enter again: 103

In [5]: print(n2, phi2)
10403 10200
```

Implement the program in a file called **hw2_part_c.py**.

PART D: (10 points)

Write a function called **`generate_public_key`** that generates the public key exponent e according to the totient ϕ that you previously calculated in Part C. This public key exponent should be chosen as an integer e such that $1 < e < \phi$ and e is relatively prime to ϕ. This function then returns the public key (e, n) as two number e and n.

You **MUST** use the following function header. You may call the functions you implement in Part A. For that you need to import **hw2_part_a.py.**

```
# This function takes two integers as its parameters: the product n and the
# totient phi. It returns the public key as explained above. You may assume
# that the parameters are always valid.
def generate_public_key(n, phi):
    # Write your code here
    # ...
```

Below are example outputs.

```
In [1]: runfile('/Users/hw2_part_d.py', wdir='/Users')

In [2]: e1, n1 = generate_public_key(10403, 10200)
In [3]: print(e1, n1)
7 10403

In [4]: e2, n2 = generate_public_key(3233, 3120)
In [5]: print(e2, n2)
7 3233

In [6]: e3, n3 = generate_public_key(323, 288)
In [7]: print(e3, n3)
5 323
```

Implement the program in a file called **hw2_part_d.py**.

PART E: (15 points)

Write a function called **`generate_private_key`** that generates the private key exponent d according to the public key exponent e and the totient ϕ that you previously calculated in Part C and Part D. This private key exponent should be chosen as an integer d that satisfies $e \cdot d \equiv 1 \bmod \phi$.

Finding this integer d is equivalent to finding d that satisfies $e \cdot d = 1 + m \cdot \phi$ for some integer m. This equation can also be written as $d = (1 + m \cdot \phi)/\, e$. Thus, in your function, starting from 0, you will work through all possible values of m until you find an integer solution for d with the given values of e and ϕ.

Below is an illustrative example. Suppose that $e = 5$ and $\phi = 108$. To find the private key exponent d, you will try different m values as follows, until you find an integer solution for d:

$m = 0 \rightarrow d = (1 + 0 \cdot 108) \,/\, 5 = 1 \,/\, 5$ continue since it is not an integer solution
$m = 1 \rightarrow d = (1 + 1 \cdot 108) \,/\, 5 = 109 \,/\, 5$ continue since it is not an integer solution
$m = 2 \rightarrow d = (1 + 2 \cdot 108) \,/\, 5 = 217 \,/\, 5$ continue since it is not an integer solution
$m = 3 \rightarrow d = (1 + 3 \cdot 108) \,/\, 5 = 325 \,/\, 5$ stop since 325/5 = 65 is an integer solution

For this example, the private key exponent d is selected as 65.

This function then returns the private key (d, n) as two number d and n.

You **MUST** use the following function header.

```
# This function takes three integers as its parameters: the public key e,
# the product n, and the totient phi. It returns the private key as
# explained above. You may assume that the parameters are always valid.
def generate_private_key(e, n, phi):
    # Write your code here
    # ...
```

Below are example outputs.

```
In [1]: runfile('/Users/hw2_part_e.py', wdir='/Users')

In [2]: d1, n1 = generate_private_key(7, 10403, 10200)
In [3]: print(d1, n1)
8743 10403

In [4]: d2, n2 = generate_private_key(7, 3233, 3120)
In [5]: print(d2, n2)
1783 3233

In [6]: d3, n3 = generate_private_key(5, 323, 288)
In [7]: print(d3, n3)
173 323
```

Implement the program in a file called **hw2_part_e.py**.

PART F: (20 points)

Write two functions called **`encrypt`** and **`decrypt`** that encrypts and decrypts a given message according to the public and private keys as well as the product n, which was used to generate the keys. In this homework, you may assume that all messages are positive integers and less than the product n. The details are as follows.

Encryption: For a given message x (an integer x), the encrypted message y is calculated as $y = x^e \ mod \ n$, where e is the public key exponent and n is the given product.

The calculation of the exponent might be easier for small values of e, but it may lead to an "*overflow error*" for larger ones. For example, try to calculate 6201^{1876} where $x = 6201$ and $e = 1876$. Here use $\text{pow}(6201, 1876.0)$ instead of $\text{pow}(6201, 1876)$ to see the error.

However, you do not need to calculate the exact value of x^e, but the result of $x^e \ mod \ n$. Thus, you can make the calculation iteratively and apply the modulus operator after each iteration. Below is an illustrative example to calculate $6201^{1876} \bmod 10403$.

$$6201^1 \equiv 6201 \pmod{10403}$$
$$6201^2 = 6201^1 \cdot 6201 \equiv 2913 \pmod{10403}$$
$$6201^3 = 6201^2 \cdot 6201 \equiv 2913 \cdot 6201 \equiv 3905 \pmod{10403}$$
$$6201^4 = 6201^3 \cdot 6201 \equiv 3905 \cdot 6201 \equiv 7124 \pmod{10403}$$
$$6201^5 = 6201^4 \cdot 6201 \equiv 7124 \cdot 6201 \equiv 4786 \pmod{10403}$$
$$\ldots$$
$$6201^{1876} = 6201^{1875} \cdot 6201 \equiv 10 \cdot 6201 \equiv 9995 \pmod{10403}$$

As you see in this example, the result at the end of each iteration cannot be greater than the product n, since you apply the modulus operator, and thus, you will not receive the "*overflow error*".

Decryption: For a given encrypted message y (an integer y), the decrypted original message x is calculated as $x = y^d \ mod \ n$, where d is the private key exponent and n is the given product. Likewise, in order to calculate the exponent without encountering the "*overflow error*", you need to use the aforementioned iterative algorithm.

You **MUST** use the following function headers. In this part, you may write an additional function to calculate the exponent iteratively, as explained above, and call this function in the **`encrypt`** and **`decrypt`** functions.

```
# This function takes three integers as its parameters: x, e, and n, where
# x is the message to be encrypted, e is the public key exponent, and n is
# the product that was used to generate the keys. This function should
# return the encrypted message (as an integer). You may assume that the
# parameters are always valid.
def encrypt(x, e, n):
    # Write your code here
    # ...


# This function takes three integers as its parameters: y, d, and n, where
# y is the encrypted message to be decrypted, d is the private key exponent,
# and n is the product that was used to generate the keys. This function
# should return the decrypted original message (as an integer). You may
# assume that the parameters are always valid.
def decrypt(y, d, n):
    # Write your code here
    # ...
```

Below are example outputs.

```
In [1]: runfile('/Users/hw2_part_f.py', wdir='/Users')

In [2]: x1 = encrypt(4560, 7, 10403)
In [3]: print(x1)
200

In [4]: x2 = encrypt(5346, 7, 10403)
In [5]: print(x2)
2536

In [6]: y1 = decrypt(200, 8743, 10403)
In [7]: print(y1)
4560

In [8]: y2 = decrypt(2536, 8743, 10403)
In [9]: print(y2)
5346

In [10]: y3 = decrypt(x1, 8743, 10403)
In [11]: print(y3)
4560

In [12]: x3 = encrypt(46, 5, 323)
In [13]: print(x3)
88

In [14]: y4 = decrypt(88, 173, 323)
In [15]: print(y4)
46

In [16]: x4 = encrypt(182, 5, 323)
In [17]: print(x4)
292

In [18]: y5 = decrypt(292, 173, 323)
In [19]: print(y5)
182
```

Implement the program in a file called **hw2_part_f.py**.

PART G: (5 points)

Now you are ready to combine all these functions into the final program. Use the following main function to make the necessary calls. Put this main function in a file called **hw2_part_g.py**. Call the main function in this file as well. Additionally, do not forget to import the necessary modules, which you implement in the previous parts.

Run this simplified RSA algorithm for different inputs and enjoy!

```
# This is the main function to put everything together. Do not forget to
# import the necessary modules. Do not forget to add a function call line
# at the end of the submission file.
def main():
    keys_generated = False
    choice = hw2_part_b.display_menu()
    while choice != 'X':

        if choice == 'G':
            n, phi = hw2_part_c.calculate_n_and_phi()
            e, n = hw2_part_d.generate_public_key(n, phi)
            d, n = hw2_part_e.generate_private_key(e, n, phi)
            keys_generated = True
            print('Keys have been generated')
            print('Public key: (', e, ', ',  n, ')', sep = '')
            print('Private key: (', d, ', ', n, ')', '\n', sep = '')

        elif choice == 'E':
            if not keys_generated:
                print('You need to generate the keys before encryption\n')
            else:
                x = int(input('Enter the message to be encrypted: '))
                y = hw2_part_f.encrypt(x, e, n)
                print(x, 'is encrypted as', y, '\n')

        elif choice == 'D':
            if not keys_generated:
                print('You need to generate the keys before decryption', '\n')
            else:
                y = int(input('Enter the message to be decrypted: '))
                x = hw2_part_f.decrypt(y, d, n)
                print(y, 'is decrypted as', x, '\n')

        choice = hw2_part_b.display_menu()
```