|  |  |
| --- | --- |
| Part 3 | |
| File Encryption And Disk Encryption For Maximum Security | |
|  |  |

Table of Contents

[Task A 3](#_Toc27687546)

[Task B 4](#_Toc27687548)

[Task C 4](#_Toc27687549)

[Part D 6](#_Toc27687550)

[Output 6](#_Toc27687551)

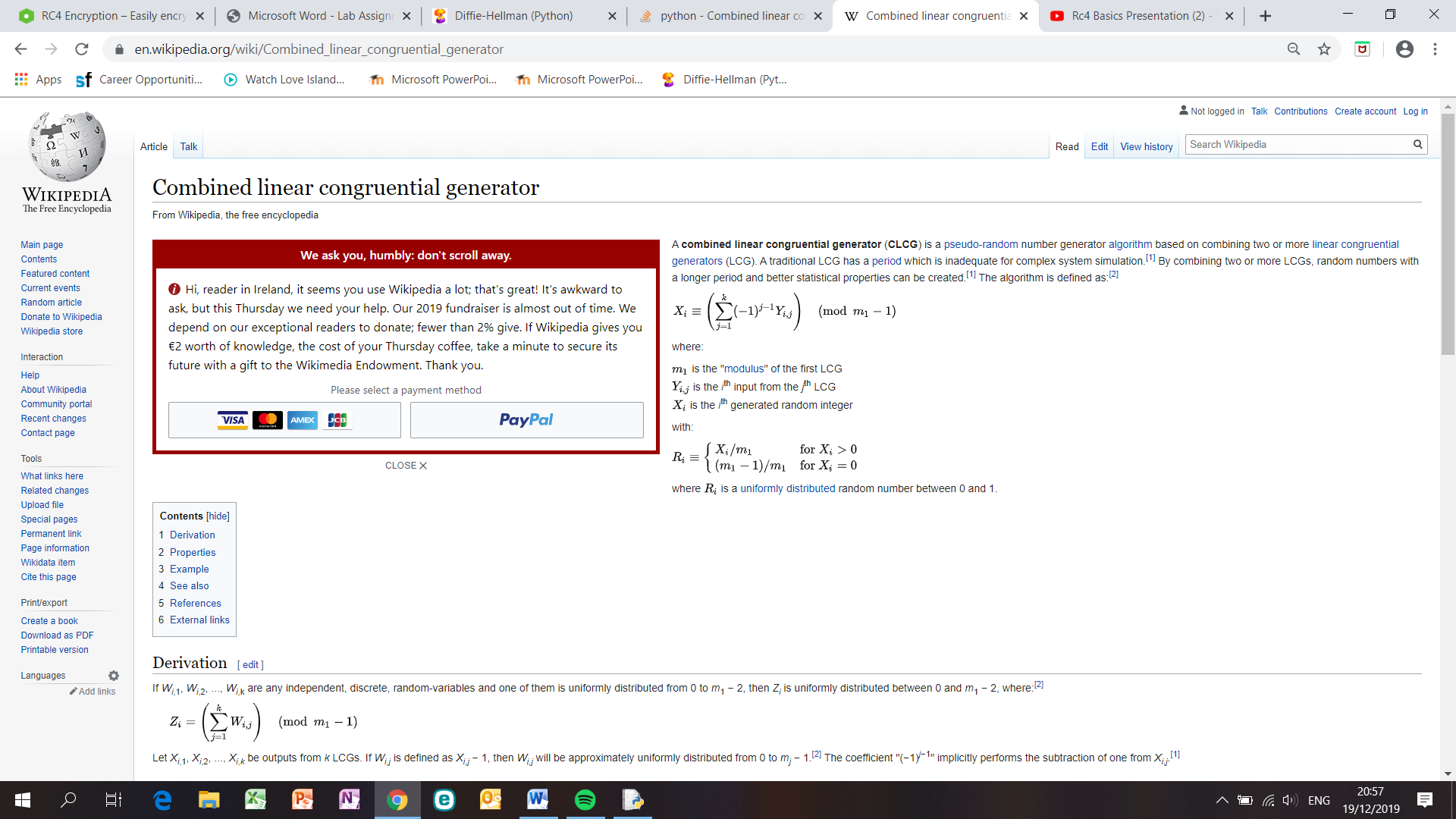
## Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. The two users, each is going to generates a one-time private key and calculates a public key. These public values, together with global public values for q and α, are stored in some central

## directory. Write a program in python and address the following requirements.

## Task A

Generate two random numbers as the one-time private keys for users A and B using combined linear congruential generator considering m1 = 2,147,483,642, a1 = 450, m2 = 2,147,483,423, a2 = 234. For Y0,1, generate a random number between [1, 2,147,483,641] and for Y0,2, generate a random number between [1, 2,147,483,422]. Use modulo operation to generate the random numbers in the range of 0 and 500. (For a new connection, new private keys should be generated).

The screenshot below is from Wikipedia and is the formula to generate two random numbers using combined linear congruential generator.



## The variable ‘r’ is an empty list which will store the two random values. The variables y1 generate a random number between [1, 2,147,483,641] and y2 generate a random number between [1, 2,147,483,422] using the random module function – ‘import random as rand’. The ‘for’ loop calculates the two random numbers by taking the variable y1 and multiply that by ‘a1’ and get (mod m1) and y2 multiplies ‘a2’ and gets (mod m2). The % (modulo) operator yields the remainder from the division of the first argument by the second. Then the variable ‘x’ subtracts y1 from y2 and gets modulo of m1.

If x is greater than 0 then x is divided by m1.

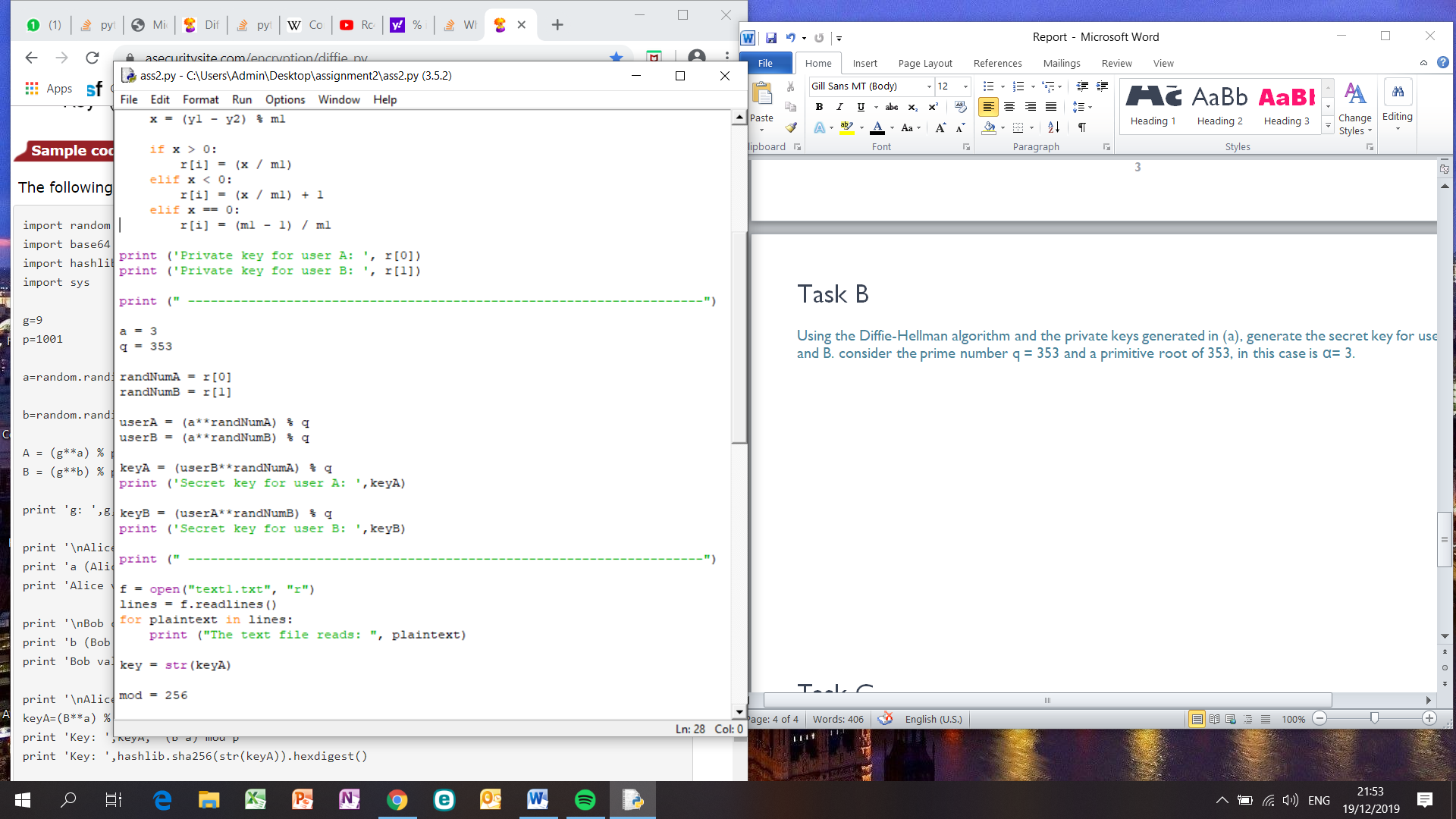
If x is less than 0 then x is divided by m1 and 1 is added to it.

If x is equals to 0 then 1 is subtracted from m1 and then it is divided by m1.

The first number of the list r[0] and second number r[1] is printed to screen as private keys for user A and B.

## Task B

Using the Diffie-Hellman algorithm and the private keys generated in (a), generate the secret key for users A and B. consider the prime number q = 353 and a primitive root of 353, in this case is α= 3.



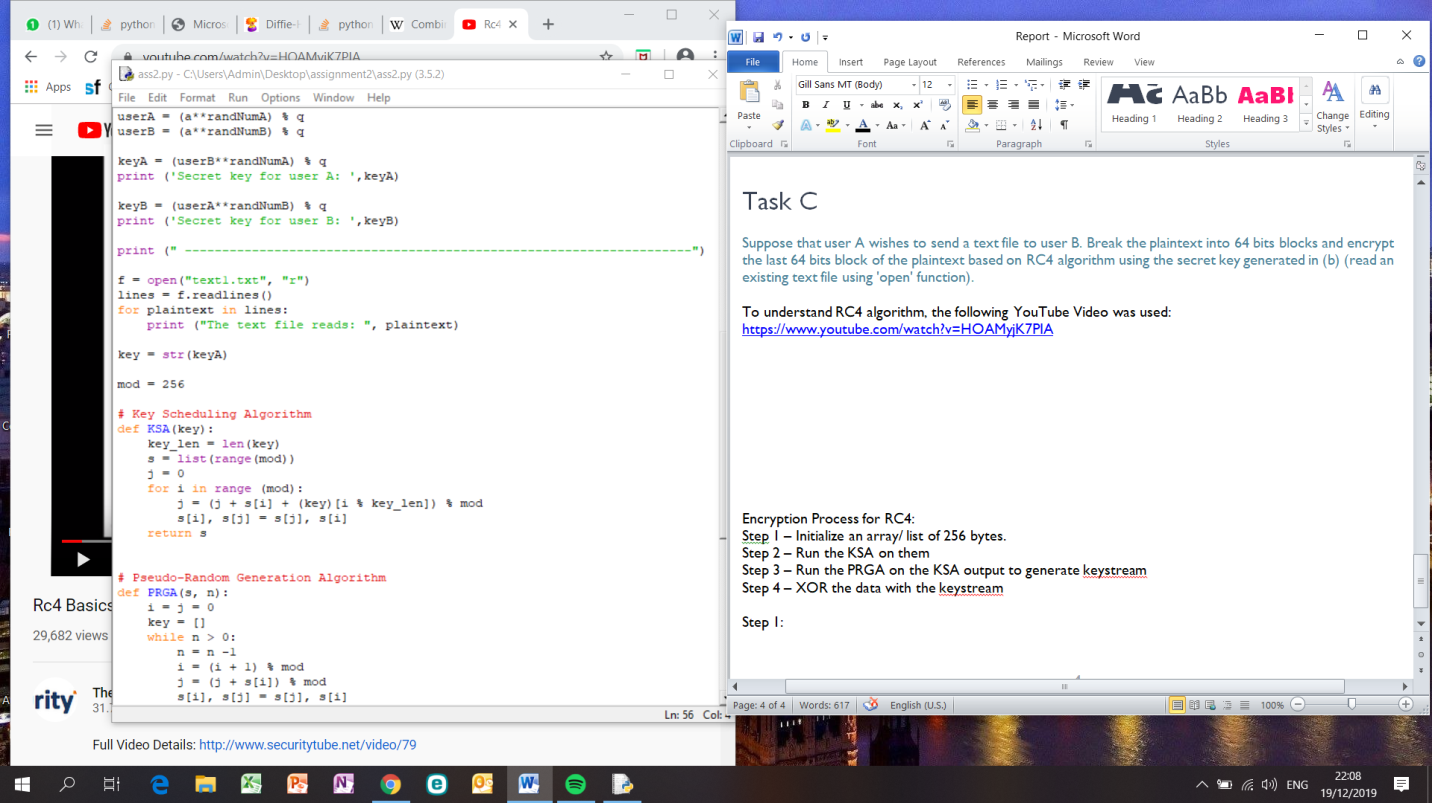
Variable ‘a’ is the primitive root value and prime number is variable ‘q’. The two random numbers generated in part a are used as ‘randNumA’ and ‘randNumB’. UserA variable calculates 3 to the power of randNumA mod 353. User A sends this publicly to User B. UserB variable calculates 3 to the power of randNumB mod 353. And sends this results to User A. Both users generate secret key by taking the results and calculating the result to the power of private numbers mod q to obtain the secret key. Both key are printed to screen.

Both have the same secret key. As both users did the same calculations.

## Task C

Suppose that user A wishes to send a text file to user B. Break the plaintext into 64 bits blocks and encrypt the last 64 bits block of the plaintext based on RC4 algorithm using the secret key generated in (b) (read an existing text file using 'open' function).

To understand RC4 algorithm, the following YouTube Video was used: <https://www.youtube.com/watch?v=HOAMyjK7PIA>



A text file called ‘text1.txt’ was opened for reading. Each line was read and the content was printed to the screen.

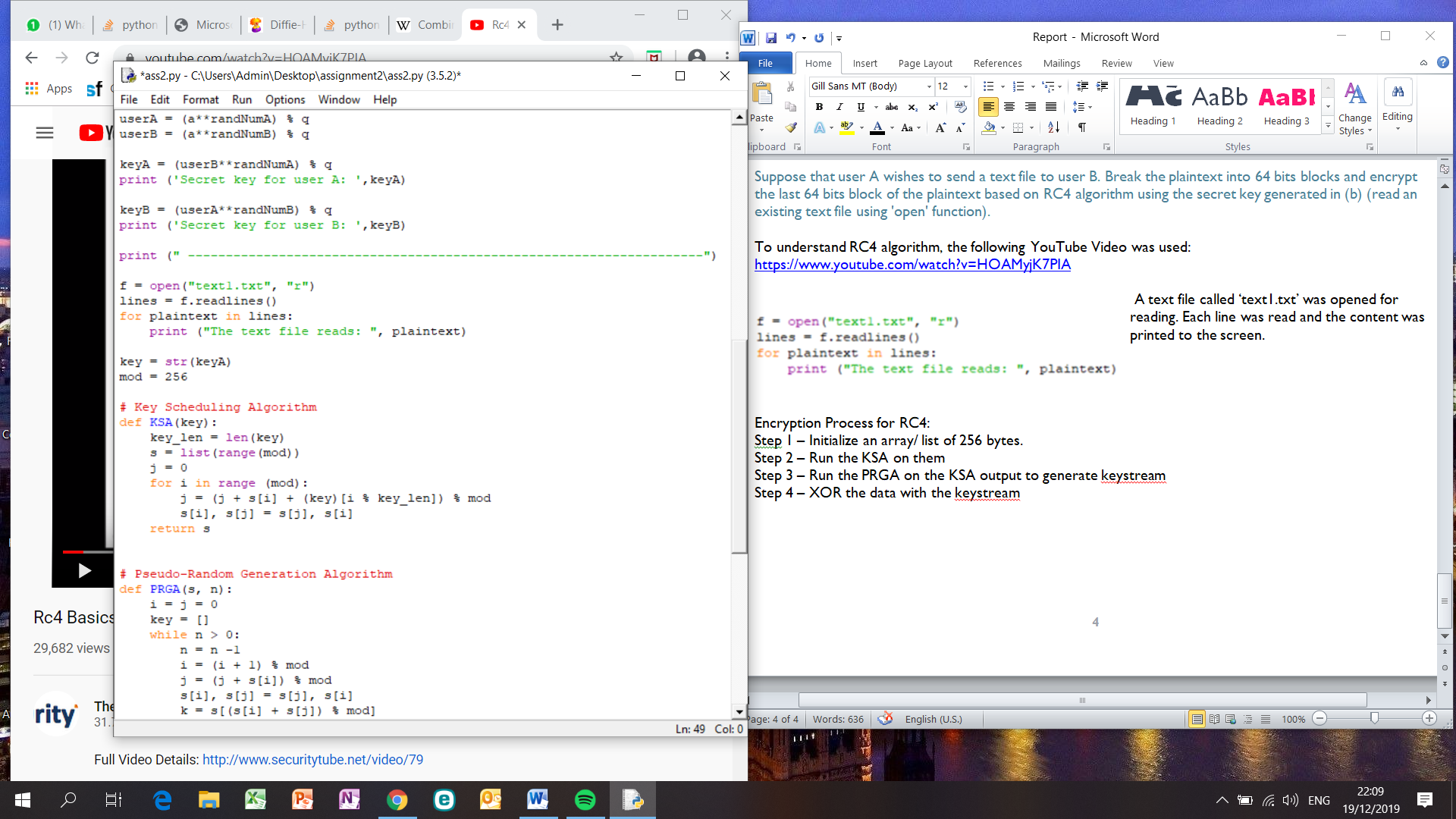
Encryption Process for RC4:

Step 1 – Initialize an array/ list of 256 bytes.

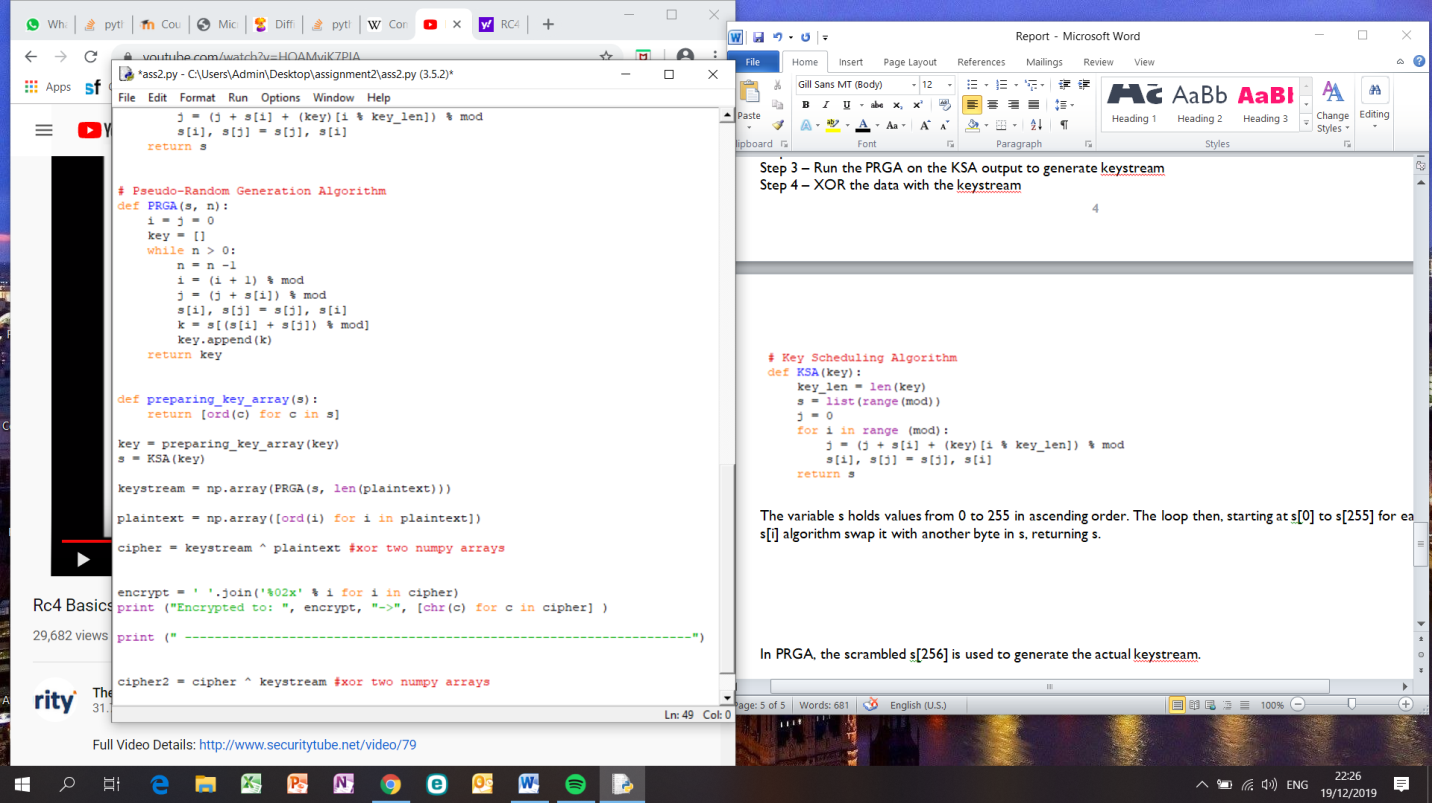
Step 2 – Run the KSA on them

Step 3 – Run the PRGA on the KSA output to generate keystream

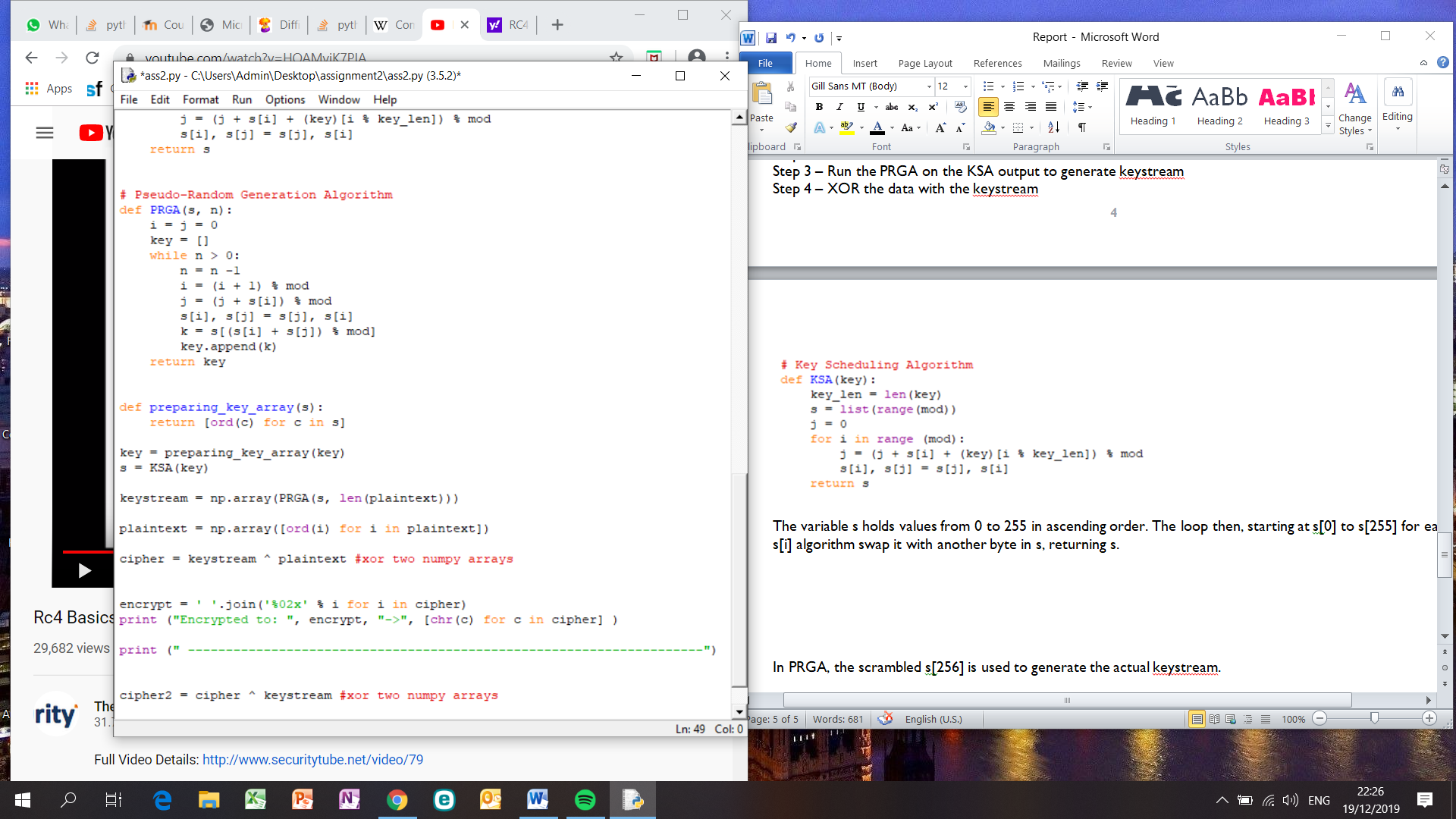
Step 4 – XOR the data with the keystream



The variable s holds values from 0 to 255 in ascending order. The loop then, starting at s[0] to s[255] for each s[i] algorithm swap it with another byte in s, returning s.



In PRGA, the scrambled s[256] is used to generate the actual keystream.

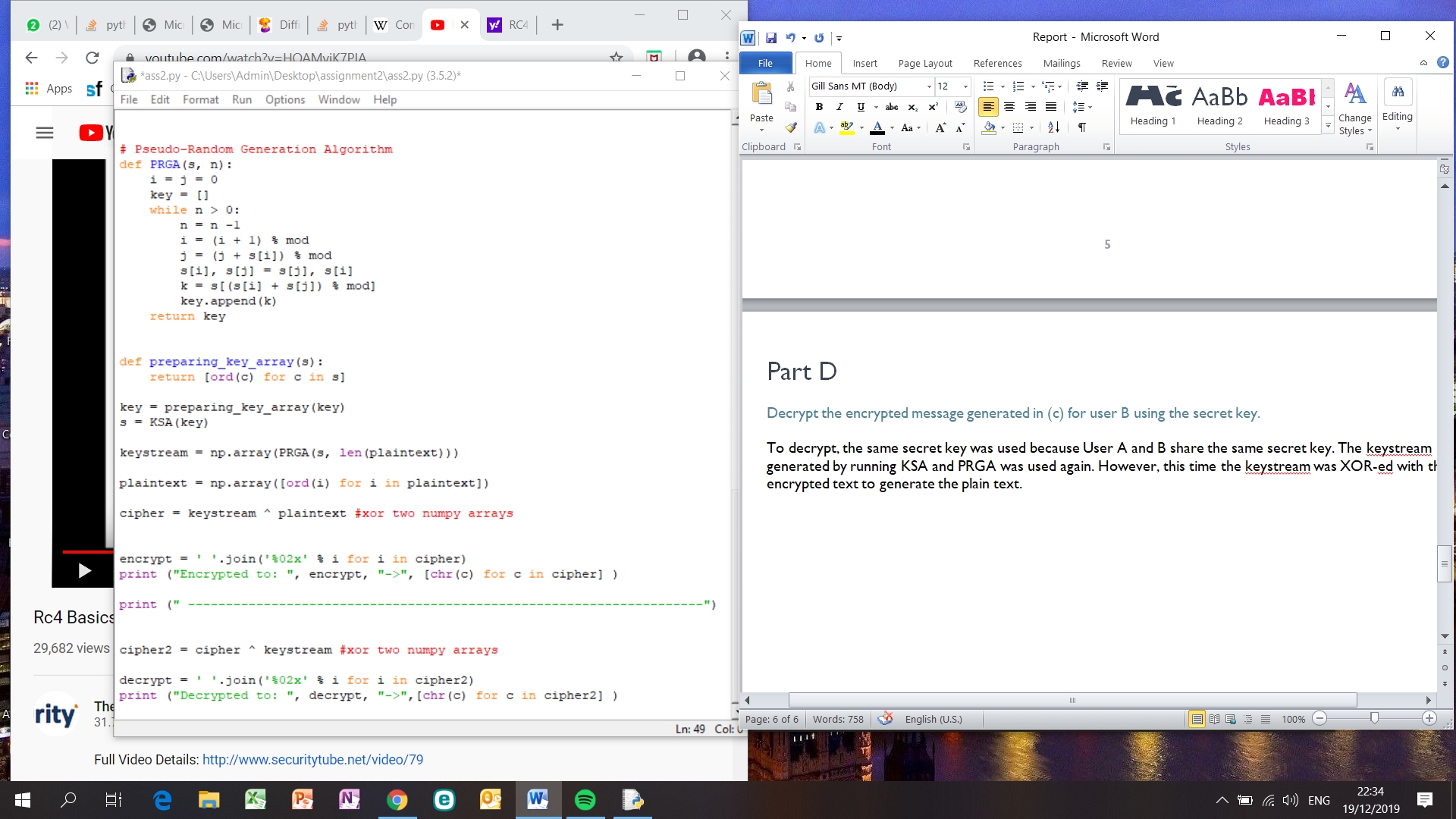


To encrypt the keystream was XOR-ed with the next byte of plaintext and printed to screen.

## Part D

Decrypt the encrypted message generated in (c) for user B using the secret key.

To decrypt, the same secret key was used because User A and B share the same secret key. The keystream generated by running KSA and PRGA was used again. However, this time the keystream was XOR-ed with the encrypted text to generate the plain text.



## Output

The output of the entire program from part a - d: 