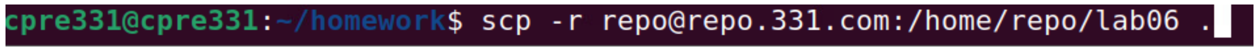
Lab 06 - Stream Ciphers in Software

Part 01

While ChaCha20 is a well-known modern stream cipher created by Daniel J. Bernstein, it isn't often used for encryption by itself. It is usually used as one of two components in the ChaCha20-Poly1305 encryption algorithm. However, it can be used for encryption as a stream cipher as you will see below. It is not recommended to be used without Poly1305 outside of lab. It is also commonly used as a pseudo-random number generator.

1. Copy the skeleton code from repo.331.com using the repo/repo credentials. You may need to adjust the permissions so you can write in the directories.

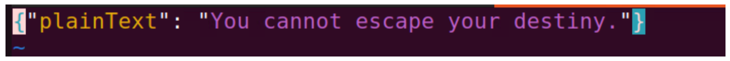


```
cpre331@cpre331:~/homework$ scp -r repo@repo.331.com:/home/repo/lab06 .
```

2. While there are multiple packages available that implement cryptographic primitives for python, we will be using PyCryptodomex (PyCryptodome is also an option) to work with ChaCha20 in part01.

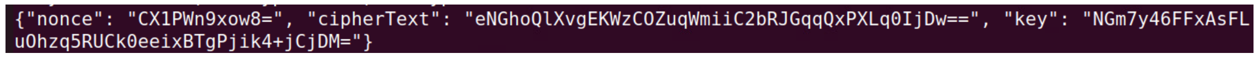
```
pip install pycryptodome
```

3. Modify the `part01_skel.py` code so it can encrypt the plaintext found in `part01_plainText.json` Minimal modification is needed to make the python code work. The heavy lifting has been done for you in this code. Look for the TODO: notifications. As shown below, there is only one entry in the plaintext json file. It will be read into a python dictionary with *plainText* as the key and *You cannot escape your destiny.* as the value. In python the dictionary is commonly referred to as a key:value pairing.



```
{"plainText": "You cannot escape your destiny."}
~
```

4. **<u>Include a screenshot of the json dumps result in your lab report.</u>** Because the key and the nonce are both randomly generated, your output will differ from the image below.



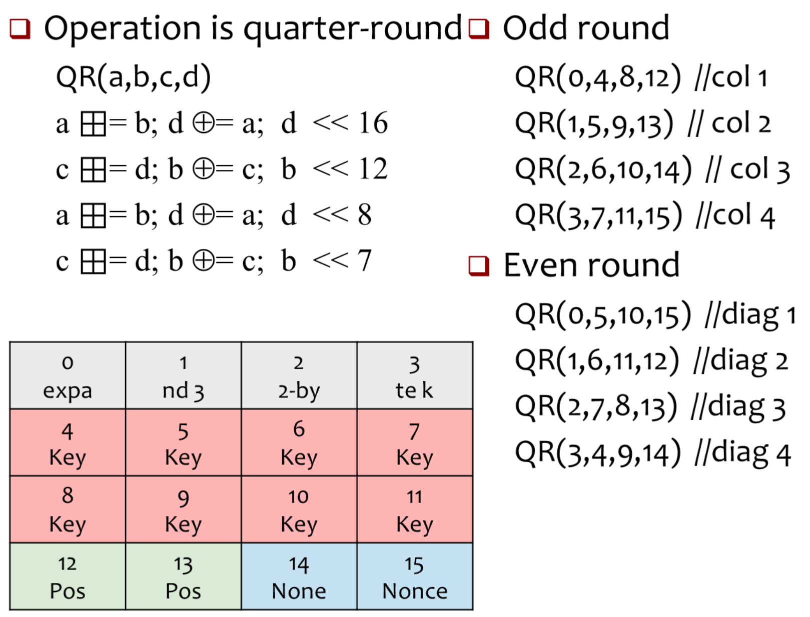
```
{"nonce": "CX1PWn9xow8=", "cipherText": "eNGhoQlXvgEKWzCOZuqWmiiC2bRJGqqQxPXLq0IjDw==", "key": "NGm7y46FFxAsFL
uOhzq5RUCk0eeixBTgPjik4+jCjDM="}
```

5. Use the `part01_skel.py` code to decrypt the ciphertext found in `part01_cipherText.json`. Again, there are key:value pairs that you will have to work with in the TODO: sections. **<u>Include a screenshot of the json dumps result in your lab report.</u>**

6. **<u>Upload your working part01_skel.py code as a separate file.</u>**

Part 02

As shown in the lecture, the generator of the keystream bits (or the random number bits) come from quarter-round operations. Each quarter-round is a combination of ARX (Addition, Rotation, XOR). `part02_skel.py` implements ChaCha20 with the details of the quarters shown. You will be using this code to **look at the diffusion of bits in just the first odd quarter round**.



❑ Operation is quarter-round

QR(a,b,c,d)

$a \boxplus= b;\ d \oplus= a;\ d << 16$

$c \boxplus= d;\ b \oplus= c;\ b << 12$

$a \boxplus= b;\ d \oplus= a;\ d << 8$

$c \boxplus= d;\ b \oplus= c;\ b << 7$

0 expa	1 nd 3	2 2-by	3 te k
4 Key	5 Key	6 Key	7 Key
8 Key	9 Key	10 Key	11 Key
12 Pos	13 Pos	14 None	15 Nonce

❑ Odd round

QR(0,4,8,12) //col 1

QR(1,5,9,13) // col 2

QR(2,6,10,14) // col 3

QR(3,7,11,15) //col 4

❑ Even round

QR(0,5,10,15) //diag 1

QR(1,6,11,12) //diag 2

QR(2,7,8,13) //diag 3

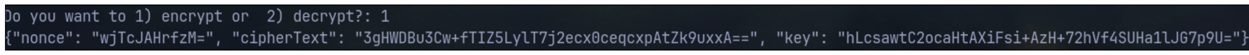
QR(3,4,9,14) //diag 4

1. Using the `part02_skel.py` code and `part02_input.json`, determine the diffusion between the **initial matrix** values and the output from the first odd quarter round **(odd qr 0**). You will calculate diffusion by
    a. Counting the number of bits changed in the four words used in the first odd quarter round.
    b. Providing the percentage of bits changed (number of bits changed/128) for just those bits used in the odd quarter round (odd qr 0). You arrive at 128 because 4 words x 32 bits per word.
    c. Finally, you will provide an overall percentage of change for the first odd quarter round (odd qr 0). Take the number of bits changed divided by the total number of bits (bits changed/512). You arrive at 512 because there are 16 words x 32 bits per word.
2. The input json file provides the key, the nonce, the plaintext and the ciphertext. You will only use the key, the nonce, and the plaintext when calculating the diffusion. The ciphertext is provided as a sanity check to ensure you don't accidentally modify something in the round functions.

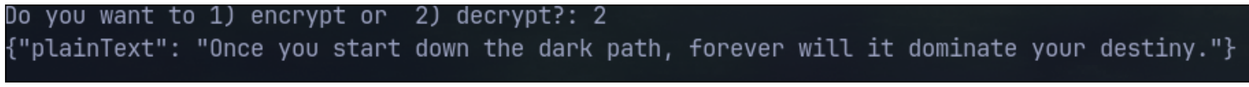
3. Remember the values stored in the matrix are 32 bit words and printed in this code in decimal format. You will have to do a little work in the skeleton code to get the binary values and then a function to compare the bits and a final percentage of how many bits changed in the first quarter round. You will also want to comment out quarter rounds you aren't interested in. It will just make it easier to read the output. **Include your bit counts and the percentages of diffusion you calculated in your lab report. Upload your code to Canvas as a separate file.**

Lab 06 Template

Part 01

1) **Include a screenshot in your lab report of the json dumps result for the ciphertext you created from the plaintext.json file.**
   (15 points)
   - 

```
Do you want to 1) encrypt or  2) decrypt?: 1
{"nonce": "wjTcJAHrfzM=", "cipherText": "3gHWDBu3Cw+fTIZ5LylT7j2ecx0ceqcxpAtZk9uxxA==", "key": "hLcsawtC2ocaHtAXiFsi+AzH+72hVf4SUHa1lJG7p9U="}
```

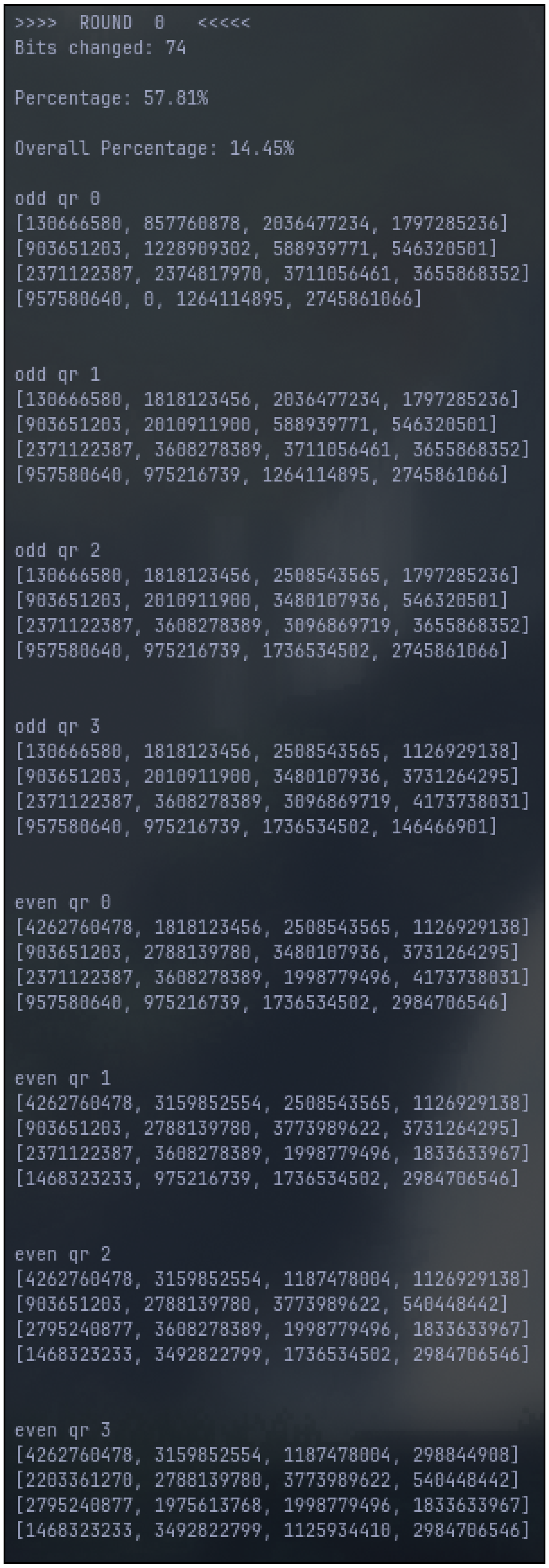
2) **Include a screenshot in your lab report of the json dumps result for the plaintext you decrypted from the ciphertext.json file.**
   (15 points)
   - 

```
Do you want to 1) encrypt or  2) decrypt?: 2
{"plainText": "Once you start down the dark path, forever will it dominate your destiny."}
```

3) **Upload your working part01_skel.py code.**
   (20 points)

Part 02

4) **Include the diffusion bit counts and percentage for odd quarter round 0. There are three values you need here. Changed bits, percentage in 4 words, percentage over 16 words.**
   (10 points)



```
>>>>  ROUND  0   <<<<<
Bits changed: 74

Percentage: 57.81%

Overall Percentage: 14.45%

odd qr 0
[130666580, 857760878, 2036477234, 1797285236]
[903651203, 1228909302, 588939771, 546320501]
[2371122387, 2374817970, 3711056461, 3655868352]
[957580640, 0, 1264114895, 2745861066]


odd qr 1
[130666580, 1818123456, 2036477234, 1797285236]
[903651203, 2010911900, 588939771, 546320501]
[2371122387, 3608278389, 3711056461, 3655868352]
[957580640, 975216739, 1264114895, 2745861066]


odd qr 2
[130666580, 1818123456, 2508543565, 1797285236]
[903651203, 2010911900, 3480107936, 546320501]
[2371122387, 3608278389, 3096869719, 3655868352]
[957580640, 975216739, 1736534502, 2745861066]


odd qr 3
[130666580, 1818123456, 2508543565, 1126929138]
[903651203, 2010911900, 3480107936, 3731264295]
[2371122387, 3608278389, 3096869719, 4173738031]
[957580640, 975216739, 1736534502, 146466901]


even qr 0
[4262760478, 1818123456, 2508543565, 1126929138]
[903651203, 2788139780, 3480107936, 3731264295]
[2371122387, 3608278389, 1998779496, 4173738031]
[957580640, 975216739, 1736534502, 2984706546]


even qr 1
[4262760478, 3159852554, 2508543565, 1126929138]
[903651203, 2788139780, 3773989622, 3731264295]
[2371122387, 3608278389, 1998779496, 1833633967]
[1468323233, 975216739, 1736534502, 2984706546]


even qr 2
[4262760478, 3159852554, 1187478004, 1126929138]
[903651203, 2788139780, 3773989622, 540448442]
[2795240877, 3608278389, 1998779496, 1833633967]
[1468323233, 3492822799, 1736534502, 2984706546]


even qr 3
[4262760478, 3159852554, 1187478004, 298844908]
[2203361270, 2788139780, 3773989622, 540448442]
[2795240877, 1975613768, 1998779496, 1833633967]
[1468323233, 3492822799, 1125934410, 2984706546]
```

-

5) **Upload your working part02_skel.py code that includes code for calculating diffusion bit counts and percentages.**
(40 points)