Report

Name:- Kshitij Deogade
Roll no:- CS17B104

Since output cipher sizes are to varied across **[2,4,8,32,128,1024] bytes;** the way I have designed my cipher requires initial messages to be of size **[1,3,6,23,93,745]** bits. This is because given a message of size **m bits**, my output cipher has **11*m** bits.
So output of my messages are of sizes [11,33,66,253,1023,8195] bits, which are very close to the required values.

For each such message length, I am considering 50 different messages. After creating such a message using random bits, **for each of the 50 messages, I use 6 keys** of sizes (2**6,2**7,2**8,2**9,2**10,2**11).
Then, I take each such key, and test it on RC4 by toggling it **once, twice,...,32 times**. So if msg is the original message and key is the original key and key2 is the toggled key; cipher1=get_cipher_text(msg,key) and cipher2=**get_cipher_text(msg,key2)** are done, and randomness between cipher1 and cipher2 is got by **get_randomness(cipher1,cipher2).**

Thus, for a specific message length and a specific toggle size, 50*6=**300** observations are made on it. Thus, average of 300 observations is taken.

In get_cipher_text(msg,key), I consider **S** array to be of size **2048 bits**, to accomodate a key of max size 2048 bits. As key is padded with itself to create array T for processing with S, len(key) should be a factor of 2048.
RC4 algorithm is followed and len(msg) many bits are put one-by-one in keystream and enc is got, which is of size len(msg).
As values in enc are integers < 2048, I broke each integer into its binary, and put it within 11 bits length, and appended every such binary stream to create cipher_text of size **11*len(msg).**
For ex:- If 5 is a value at an index in enc, then, it is written as 00000000101.

As a test, I have also written code to decode this cipher text that I created, which is basically just following the steps of RC4, but with the cipher_text as input, and I am able to get the original message back, showing a successfule encoding.

In get_randomness(a,b), 2 cipher texts a and b are compared according to steps mentioned in assignment.
However,note that instead of keeping counter_array of size 256, I keep it as **2048.** This is because my cipher text's size is divisible by 11, so I am dividing XOR(a,b) into 11 bit chunks, and treating this as a single number. Since, 11 bit number can go till max value 2048, hence the size.

Observations:

1) In general, with increase in message size, randomness score decreases, showing more randomness is got by having larger message.
2) For message of size 1 bit, randomness score is same at all toggles. This is because only 1 number is getting incremented as cipher_text size=11 here, and I am considering 11 bit chunks. So, standard deviation is same for all.
3) From 1 to 5 toggles in key bits, randomness is clearly increasing, as can be seen in the sharp slope. However, after 5 bits, the effect is not visible so much,

as curve is almost flat.
However, the sharpness of the curve shows that within a few toggles, randomness is increasing massively, showcasing avalanche effect.

What I learnt:

I learnt about RC4 algorithm, and implemented the code by myself, in Python. I was successfully able to encrypt and decrypt data, and see how change of key can randomize the output bits got. I plotted a randomness score of output with number of key_toggles, and saw randomization increasing with key-bit toggles.