CS6500 - Network Security

Assignment 1 : Cryptanalysis of RC4 encryption algorithm

Name: Sandip Saha
Roll number: CS20S044

---

1. Description

In this assignment we have to implement RC4 cipher and perform cryptanalysis with similar keys. RC4 is a popular output feedback cipher designed by Ron Rivest in 1987. In his Assignment we will study the effect of similar keys in the generated key stream used for encryption.

2. Implementation

I have implemented the RC4 cipher in C language. Then used python to do the cryptanalysis.
The "rc4_init" initialize the State array S.

```
void rc4_init(unsigned char *key, size_t key_len, unsigned char *S)
{
    for (size_t i = 0; i < SIZE; i++)
        S[i] = i;

    size_t j = 0;
    for (size_t i = 0; i < SIZE; i++)
    {
        j = (j + S[i] + key[i % key_len]) % SIZE;
        swap(&S[i], &S[j]);
    }
}
```

the function key_stream_gen is used to generate the key stream.

```
void key_stream_gen(unsigned char *S, int plain_text_len, unsigned char *key_stream)
{
    size_t i = 0, j = 0;

    for (size_t n = 0; n < plain_text_len; n++)
    {
        i = (i + 1) % SIZE;
        j = (j + S[i]) % SIZE;
        swap(&S[i], &S[j]);

        key_stream[n] = S[(S[i] + S[j]) % SIZE];
    }
}
```

And RC4 is the driver code. for the purpose of this experiment we are generating the key stream and storing it in a file called 'byte', which is read in the python code to do cryptanalysis.

```
void RC4(unsigned char *key, int key_len,int plain_text_len)
{
    unsigned char S[SIZE];
```

```
    unsigned char key_stream[plain_text_len];
    rc4_init(key, key_len, S);
    key_stream_gen(S, plain_text_len, key_stream);
    FILE *f;

    if ((f = fopen("./bytes", "wb")) != NULL)
    {
        fwrite(key_stream, 1, plain_text_len, f);
        fclose(f);
    }
    else
    {
        printf("Error writing the bytes");
        exit(0);
    }
}
```

Other function for encryption and decryption are also implemented.

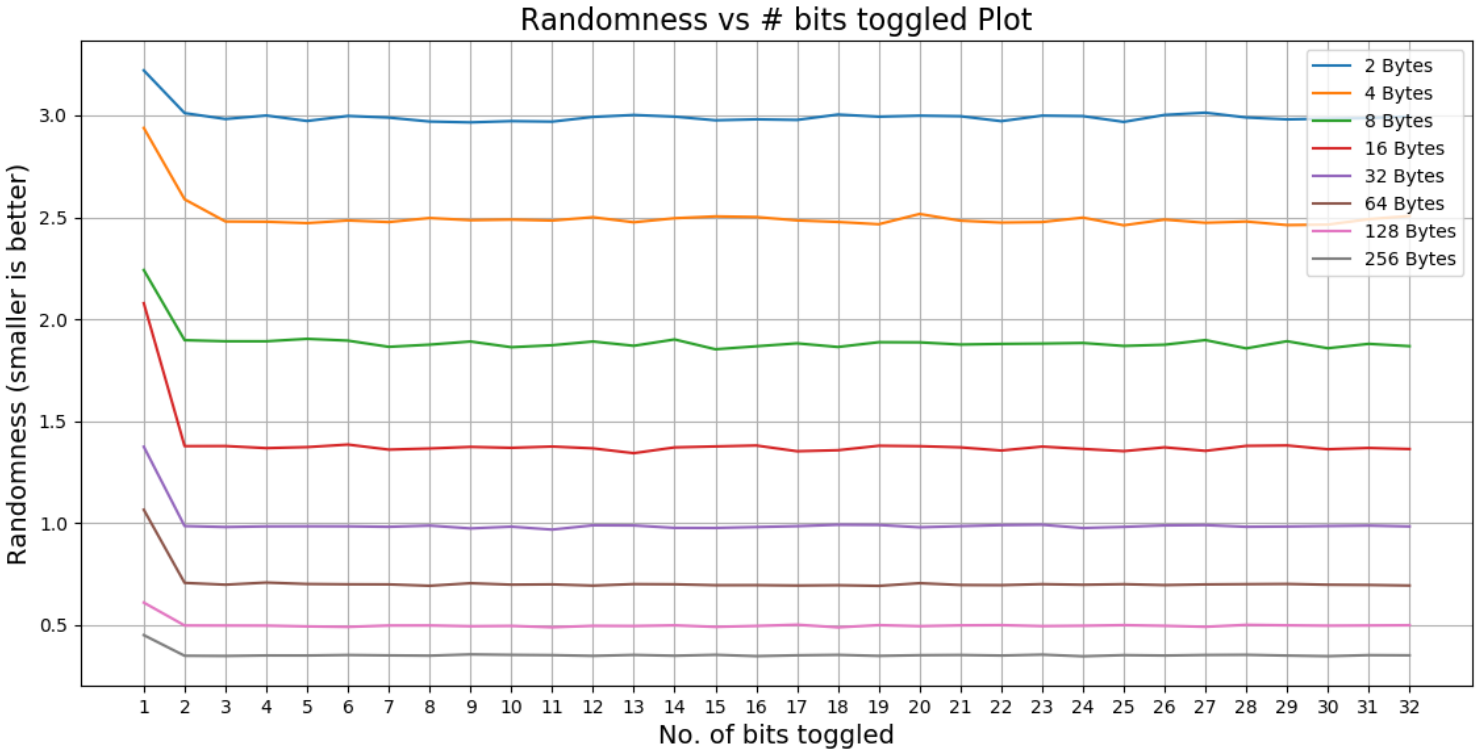
3. Experiment

I generate a random 256 byte key and store it in a file. Once generated I will use the same key to perform the whole experiment. I will refer this key as original key to explain my experiment. The same experiment is performed for various plain text length like $2, 4, 8, \ldots 256$. For a fixed plain text length the experiment steps are as follows

1. First I generate a key stream using the Original Key and store it in a byte array called **br1**.
2. Next I 150 times randomly toggle one bit from original key to get a modified key and do the following
3. The modified key used to generate another byte stream called **br2**.
4. **br1** and **br2** is exor'ed, lets say bin_str is the corresponding binary string of the exor result.
5. Then for each 8 bit sequence in the bin_str, Interpret it as an unsigned integer and increment the counter array corresponding to that value.
6. Then I calculate numerical measure of the randomness as $R = (D * C) \,/\, N$, where
   (a) D is standard deviation of counter values.
   (b) C is the number of counters
   (c) N is the number of samples i.e. number of 8 bit sequence in the bin_str.
7. Finally, the average over 150 different R is taken for plotting.
8. Next I randomly toggle two, three, ... up to 32 bits in the original key and perform the steps from 3 to 7.

4. Results

After generating the data I used matplotlib to plot the **numerical measure of the randomness vs number of bits flipped** for various plain text length $(2, 4, 8, \ldots, 256)$.



5. Observations

The observations of the experiments are as follows

1. randomness of the key stream increases with the with increase in number of bit flips from 1 to 2 or 2 to 4. But randomness of the key stream is not proportional to the number of bits flipped. for a particular plain text length randomness of the key stream is almost constant for more than 4 bits flipped.
2. randomness of the key stream is dependent on the length of the plain text, if we are generating a long key stream say (256 byte long) then the randomness is pretty high.