

Technical Report
CS6500 Network Security : Programming assignment 1
Name : Kolan Nikshith Reddy
Roll No : CS20m030

Cryptanalysis of Stream Cipher RC4:

RC4 (Rivest Cipher 4) is a Stream Cipher, while it is remarkable for its simplicity and speed in software, multiple vulnerabilities have been discovered in RC4, rendering it insecure. It is especially vulnerable when the beginning of the output keystream is not discarded, or when non-random or related keys are used.

After analysing the differential output bits for randomness using the given formula using a simple frequency counting test and computing a numerical measure of the randomness like so:

- N = number of samples.
- C = number of counters.
- D = standard deviation of counter values.
- $R = (D * C) / N$;

The closer the randomness (R) is to zero, the more random the data.

In my implementation, I checked the RC4 cryptanalysis for 4 cases with the help of 2 parameters.

Parameter 1: Whether the bits toggled are from 1 to 2048 position or the toggled bits are in the last portion of the key.

Parameter 2: Using two types of counters. One changes with every iteration and one Changes only once for each bit toggled.

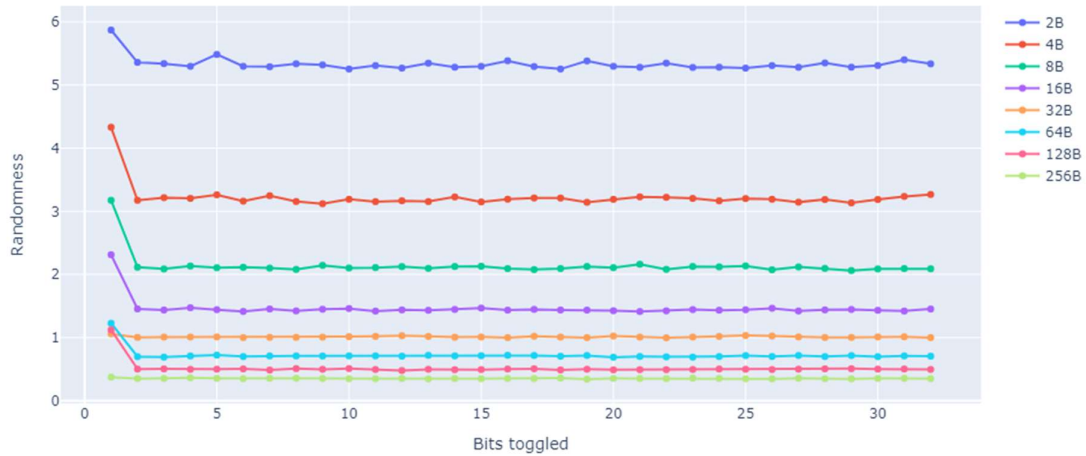
1. Counter is initialised to zero once for every iteration of 40 iterations and bits toggled can be in range 1 to 2048
2. Counter is initialised to zero once for every iteration of 40 iterations and bits toggled can be in range 1500 to 2048
3. Counter is initialised to zero once for each bit toggled and bits toggled can be in range 1 to 2048
4. Counter is initialised to zero once for each bit toggled and bits toggled can be in range 1500 to 2048

Observations:

1. By Measuring the randomness on outputs ranging from short through long (i.e from 2 B to 256 B) in the above experiment, I found out that,
 - a. Randomness is getting close to zero with moving from 2B to 256B. That is randomness of data is increasing with the increase in size.

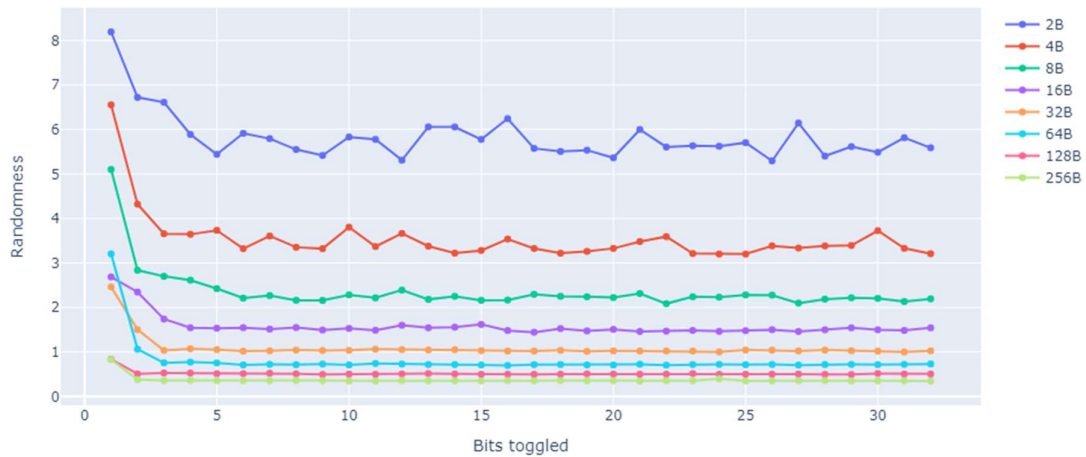
Case 1: When bits are toggled from the first bit itself, we got a neat curve of randomness. i.e., Randomness is increasing with increase in no of bits toggled.

RC4 Cryptanalysis: Case 1: Counter is zero for every iteration and Bits toggled are in range 1 to 2048



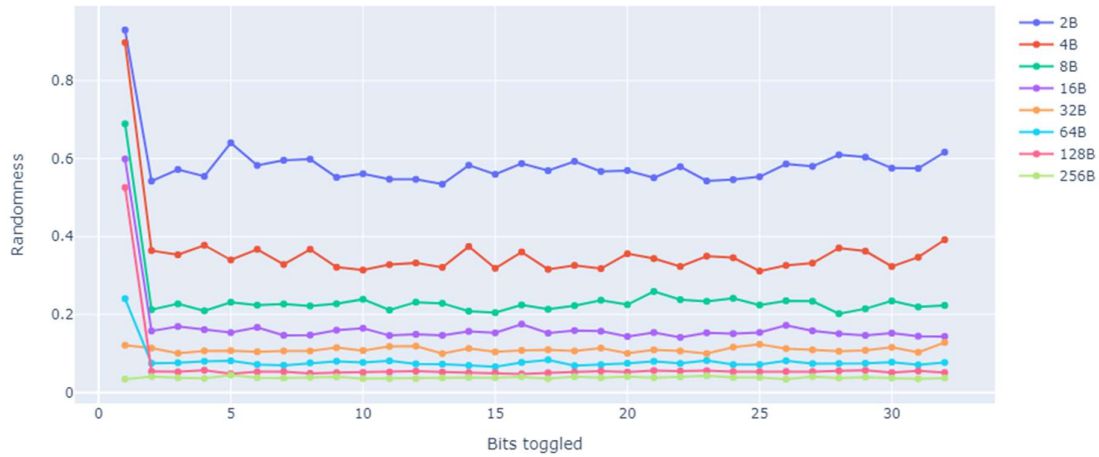
case 2: When the bits toggled are in the last portion of the key, The randomness is not neat. That is we have similarities in the cipher text

RC4 Cryptanalysis: Case 2: Counter is zero for every iteration and Bits toggled are in range 1500 to 2048



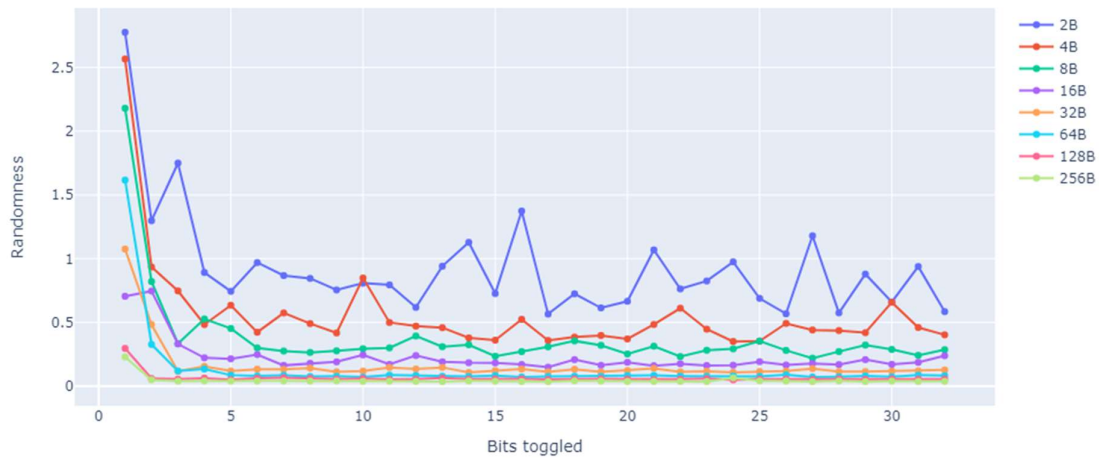
Case 3: Even with Different counter, When bits are toggled from the first bit itself, we got a neat curve of randomness. i.e, Randomness is increasing with increase in no of bits toggled

RC4 Cryptanalysis: Case 3: Bits toggled are in range 1 to 2048



Case 4: Even with Different counter, When the bits toggled are in the last portion of the key, The randomness is not neat. That is we have similarities in the cipher text.

RC4 Cryptanalysis: Case 4: Bits toggled are in range 1500 to 2048



Observation 2:

Key 1 : 2048 bit key is considered

Key 2 : only last one bit of key is altered from key 1

Key 3 : only first one bit is altered from key 1

1. When 1 bit is changed in the end of the key, we can see cipher text is almost similar and is not random.
2. When 1 bit is changed in the start of the key, we can see cipher text is completely different and is random.

```
Windows PowerShell
PS C:\Users\nkred\OneDrive\Desktop> python ex.py
Text:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus ante est, tempor ac mauris sit amet, interdum luctus sapien. Vivamus posuere dolor sed ris
us sagittis.

key 1 (2048 bit): 6A576E5A7234753778217A25432A462D4A614E645267556B58703273357638792F423F4428472B486250655368566D597133743677397A244326462948404D635166546A576E
5A7234753778214125442A472D48614E645267556B58703273357638792F423F4528482B4D6251655368566D597133743677397A24432646294A404E635266556A576E5A7234753778214125442A47
2D486150645367566B59703273357638792F423F4528482B4D6251655468576D5A7134743677397A24432646294A404E635266556A586E327235753878214125442A472D486150645367566B597033
733676397924423F4528482B4D6251655468576D5A713474377217A

key 2 (2048 bit): 6A576E5A7234753778217A25432A462D4A614E645267556B58703273357638792F423F4428472B486250655368566D597133743677397A244326462948404D635166546A576E
5A7234753778214125442A472D48614E645267556B58703273357638792F423F4528482B4D6251655368566D597133743677397A24432646294A404E635266556A576E5A7234753778214125442A47
2D486150645367566B59703273357638792F423F4528482B4D6251655468576D5A7134743677397A24432646294A404E635266556A586E327235753878214125442A472D486150645367566B597033
733676397924423F4528482B4D6251655468576D5A713474377217B

key 3 (2048 bit): 7A576E5A7234753778217A25432A462D4A614E645267556B58703273357638792F423F4428472B486250655368566D597133743677397A244326462948404D635166546A576E
5A7234753778214125442A472D48614E645267556B58703273357638792F423F4528482B4D6251655368566D597133743677397A24432646294A404E635266556A576E5A7234753778214125442A47
2D486150645367566B59703273357638792F423F4528482B4D6251655468576D5A7134743677397A24432646294A404E635266556A586E327235753878214125442A472D486150645367566B597033
733676397924423F4528482B4D6251655468576D5A713474377217A

key 1 cipher text:
ec2473270f561cd87ac79a9bd5467f925df84981fca45dea3e4f408966f8da57f407ff607ccd22cb3b0c76529d6e76bc9536cf8d38367003ae80e96a8c1950902c31dc3f65bba86e375103dd1373a7
2e5a67237151ceac761e526e0d85f5da92dee39ff9b7d3376ef25cb7af43e2af7415d821ad5c196786331ed62c62663867397624a45790bdd38094fd003db57425278bd9630721bafab7472f8e82
0b8c1e469ee852c6a801649

key 2 cipher text:
ec2473270f561cd87ac79a9bd5467f925df84981fca45dea3e4f408966f8da57f407ff607ccd22cb3b0c76529d6e76bc9536cf8d38367003ae80e96a871950902c31dc3f65bba86e375103dd1373a7
2e5a67237151ceac761e526e0d85f5da92dee39ff9b7d3376ef25cb7af43e2af7415d821ad5c196786331ed62c62663867397624a45790bdd38094fd003db57425278bd9630721bafab801effa29b
967f0b6ff7e466f3730cab58

key 3 cipher text:
d1ac25ded2eeeba2807b86e09a578d7dbc24bf4a87f202711c07d96769241a0adad3be2735037549731c3045ab7303f439e91f6666e8b12970a8e0b8b131bcc5475b3faf2632dd996c63a594d33ed
3d024744953eeec123bf6faf5773eaa9b4bd743c84f1dee7fa4cc62daa6a917ec1edd9269d40b2180593a6263f4e13533afa7f2b8f8bdca090c0105cf58646e7c94ba231ea9593dea0749d82eb42f28
839c9b6aa99f6874e8275519
PS C:\Users\nkred\OneDrive\Desktop>
```

Observation 3:

When we are using a long key, it is better to generate some initial dummy keystream which we can drop. Here we are also checking using a long stream, so we can perform the above as we can generate some initial dummy keystream and drop it.

With this, We can improve randomness. That is, from 2B to 256B we improved randomness.

Learnings:

1. the most important weakness of RC4 comes from the insufficient key schedule; the first bytes of output reveal information about the key. This can be corrected by simply discarding some initial portion of the output stream.
2. RC4 lacks authentication and if not used together with a strong message authentication code (MAC), then encryption is vulnerable to a bit-flipping attack
3. RC4 stream ciphers are simple to use, fast, easy to implement.
4. RC4 stream ciphers are implemented on large streams of data.

From this experiment I got a good exposure of how various ciphers work and Got hands on experience with implementing a cipher and in analysing the results using various graphs.

I tried toggling random bits and found out with trial and error that that bits that need to be toggled are the ones in the starting portion of the key, in order to generate a complete random cipher text.