Assignment 3 Report

Akhilesh (CS1170323) Rajat Singh (CSZ208507)

Running the code:

We have implemented this assignment in python3 and used matplotlib library to plot the graph.

We have used the following code files:

* assignment3.py – to run the randomness test for individual N.
* assignment3\_all.py – to run the randomness test on all N in one go.
* Assignment3\_vendor.py – to run the test for finding out the ideal number of output bits to throw out from RC4 to let the key bits mix properly.

To run the code, use the following command

**python3 assignment3.py <number of output bytes> <Number of runs> <name of plot>**

**python3 assignment3\_all.py <Number of runs> <name of plot>**

**python3 assignment3\_vendor.py <Number of runs> <name of plot>**

Examples:

**python3 assignment3.py 2 60 plot\_2\_60.png**

**python3 assignment3\_all.py 60 all.png**

**python3 assignment3\_vendor.py 60 vendor.png**

Plots and Observations:

Chart, histogram

Description automatically generated

Fig 1: Plot for randomness using N=2bytes over 60 runs.

Chart, histogram

Description automatically generated

Fig 2: Plot for randomness using N=4bytes over 60 runs.

Chart, histogram

Description automatically generated

Fig 3: Plot for randomness using N=8bytes over 60 runs.

Chart

Description automatically generated

Fig 4: Plot for randomness using N=32bytes over 60 runs.

Chart, histogram

Description automatically generated

Fig 5: Plot for randomness using N=128bytes over 60 runs.

Chart, histogram

Description automatically generated

Fig 6: Plot for randomness using N=1024bytes over 60 runs.

Chart, line chart

Description automatically generated

Fig 7: Plot for randomness using all the values of N over 60 runs.

Ques: If the two original bitstreams were very similar, you would expect the counters for lots of zeros to have higher values than the counters for lots of ones.

Ans: After comparing the outputs for every iterations we found out that we are getting 5010 % of bits, where the bits are different in two equal length output and this is due to the reason that 50% of the bits should be different for random value in an ideal case. (Output shown in Fig8 for N=4Bytes)

Text

Description automatically generated

Fig 8 : Showing the number of different bits between two outputs of 4bytes (32 bits)

Ques: Numerical measure of the randomness.

Ans: We can see the following observations from the above plots.

1. As we can see from the above plots the randomness on outputs ranging from short through long (i.e., 2 bytes, 4 bytes, 8 bytes, 32 bytes, 128 bytes, 1024 bytes) is increasing as the value of R is decreasing. This is due to the fact that in the **pseudoRandomGeneration()** function we are shuffling the bits with more number of time for 1024bytes as compared to 2bytes.
2. Another observation we can see from the above plots is that if we increase the number of flip bits then the randomness is increasing but after a certain number of flips it will not vary much and seems stagnant.
3. From fig7 we can conclude that as the values of N is increasing the plot is more closer to each other as compared to the lower values of N. This can be justified by the reason that as the value of N increases the randomness increase and after a certain value of N there will not be very drastic change in randomness thus the values lies close to each other.
4. Lastly we observed that if we decrease the number of runs (say 10) then the plot we are getting was very jumbled and if we increase the number of runs (say 60+) then the plot we are getting is quiet intuitive and values are not any zig-zag pattern.

Ques: How many bits need to be flipped (on average) before the differential bitstream looks random?

Ans: For fig1, fig 2, and fig 3 we can conclude that we are getting good amount of randomness after flipping at least 3 random bits. For fig4, fig5 and fig6 we can conclude that we are getting good amount of randomness after flipping at least 2 random bits. We assume this trend is correct as for fig4, fig5 and fig6 we are swapping the outputs by 32, 128 and 1024 times respectively which means we will achieve randomness quiet early and vice versa for fig1, firg2, and fig3.

Ques: If a vendor wished to ship a product using RC4 to encrypt short messages (50 bytes max), perhaps they should throw out some of the initial output from RC4 to let the key bits mix properly. How much?

Ans:

Chart, line chart

Description automatically generated

Fig 9: Plot of randomness vs Initial output to throw from RC4

Mironov in [1] argued that the weakness of RC4 is the improper swap function which occurs generally in the first 3\*256 Bytes. In this part we are trying to find the approximate number of bytes that we can throw from initial output to let the keys mix properly. We found out that it mostly lies around 1000Bytes. Here in the above figure it is around 850Bytes.

Another observation we can make form the above graph is that the value of R is oscillating around a particular value of R. It looks like that it is following some cyclic property.

Lastly we can argue that throwing more and more initial bits does not improve the randomness to a large extent but it will oscillating to a particular value of R.

Ques: Rather than randomness tests, you might observe that two RC4 systems initialized with similar keys may initially generate identical values but will eventually diverge. Try to measure the average length (in bytes) of identical output as a function of the number of bits you change in the key.

Ans:

References:

[1] Mironov I 2002 (Not So) Random shuffles of RC4. In: Proceedings of Advances in Cryptology (CRYPTO), California. LNCS.