CS6500 - Network Security
Even Sem. 2021, Dr. Manikantan Srinivasan
Assignment 1 : Cryptanalysis of RC4 encryption algorithm
Due date: March 1, 2021, 11:59 PM, On Moodle

Extension: 15 % penalty for each 24-hr period; Max. of 48-hrs past the original deadline

February 19, 2021

The objective of this assignment is to do some simple cryptanalysis of the RC4 encryption algorithm. RC4 is a stream cipher, internally generating a stream of random bits which is then **XOR-ed** with the **plaintext**. The secret key initializes the random number generator. However, on observation there appears to be a weakness in RC4. Perhaps two keys that are very similar will result in similar random numbers coming out of RC4. In this assignment you will implement RC4, make some differential measurements of randomness for related keys, and you will graph your results. For details about RC4 you can refer wiki RC4.

Credits\Courtesy -
https://www.cs.rice.edu/ dwallach/courses/comp527_s99/ass4.html

1 Assignment Details

1.1 The Basics

You are going to first implement the RC4 algorithm. You can find some code at the RC4 in three lines of Perl site (which also includes a full, readable C version). This assignment is a form of related-key cryptanalysis. With a stream cipher like RC4, it should be the case that, if you toggle exactly one bit in the key, 50% of output bits should toggle. If you look closely at the key setup, it doesn't look like this is the case. It appears that single-bit differences in the key will result in fairly small differences in the internal state of RC4. After a sufficient amount of output, this small change will eventually propagate and then you would expect 50% of the output bits to toggle.

1.2 Implement the cipher

Don't worry about XOR-ing the output bits with a message you're going to keep secret. We're only going to study the random bits. Make sure you can accept keys of length up to 2048 bits.

1.3 Capture the output bits from two runs and XOR them

Generate a random key of 2048 bits (Unix's **random(3C)** is good enough for now). Collect some output from RC4. Toggle one or more of the key bits. Reinitialize RC4 and collect the output again. XOR them together. This gives you the **difference** of the two streams. If RC4 were "perfect", this difference stream would be "perfectly" random.

1.4 Analyze the differential output bits for randomness

There are many tests for randomness of numbers. You're going to implement a simple frequency counting test. Create an array of counters of length equal to a power of two (say, 256). Now, say the test data is a sequence of bits $(b_0, b_1, b_2, b_3...b_N)$. You can look at each sequence of 8 bits $(b_0, b_1, b_2, b_3...b_9)$ as a number and increment the appropriate counter in the array. When you're done you would expect that the counters would be approximately equal. If the two original bit streams were very similar, you would expect the counters for lots of zeros to have higher values than the counters for lots of ones.

You can compute 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. You might consider using different numbers of counters and see if your randomness measure changes very much

1.5 Run this randomness test lots of times and collect the results

Measure the randomness on outputs ranging from short through long (i.e., 2 bytes, 4 bytes, 8 bytes, 32 bytes, 128 bytes, 1024 bytes). For each of these, you need to consider the effect of toggling one bit in the key, two bits in the key, and so forth through 32 bits. For each of these pairs, you should make at least 20 measurements of the randomness and average the results. Your results will be more accurate if you make several thousand measurements for each pair.

1.6 Graph the results

You should produce a graph with one line for each output length. The Y axis is the randomness (higher values imply less randomness). The X axis is the number of bits you toggled. You would expect each line to start somewhere above zero and, as you toggle more bits, approach zero quickly. You would also expect to see higher values for the lines corresponding to shorter runs of data.

You can either use Unix tools like gnuplot or a spreadsheet like Excel to generate your graph.

2 What to Submit

Create a tar-gz file with name: Assignment1-RollNo1.tgz (e.g. Assignment1-CS17B099.tgz) that will contain a directory named Assignment1-RollNo1 with all relevant files.

The directory should contain the following files:

- Source Files

- A Makefile which generates all your executables
- A technical REPORT (in PDF format) with help of screen shots showing the working of your implementation (Graphs). Report your observations and analyze the results, in 1-2 paragraphs. The report should include your name, roll number, assignments.

  Also, give a brief summary as to what you learnt in this experiment and how much beneficial you feel the experiment was.
- a README file containing instructions to compile, run and test your program. README file should be written such that a TA must be able to run your code without your presence/help.

3 Help

1. Ask questions EARLY and start your work NOW. Take advantage of the help of the TAs and the instructor.
2. Submissions PAST the extended deadline SHOULD NOT be mailed to the TAs. Only submissions approved by the instructor or uploaded to Moodle within the deadline will be graded.
3. Demonstration of code execution to the TAs MUST be done using the student's code uploaded on Moodle.
4. NO sharing of code between students, submission of downloaded code (from the Internet, Campus LAN, or anywhere else) is allowed. Code copying will result in a 'U' Course Grade. Students may also be reported to the Campus Disciplinary Committee, which can impose additional penalties.
5. Please protect your Moodle account password. Do not share it with ANYONE. Do not share your academic disk drive space on the Campus LAN.
6. Implement the solutions, step by step. Trying to write the program in one setting may lead to frustration and errors.

4 Grading

- Code : 40 points
- Report: 40 points
- Viva voce: 20 points