Stream Cipher Assignment

Start Assignment

Due 13 Apr by 17:00 **Points** 100 **Submitting** a file upload **File types** zip
Available 25 Mar at 12:00 - 13 Apr at 17:00 19 days

Purpose

The purpose of this assignment is that you will design a practical encryption application around a symmetric encryption algorithm. You will write programs for data encryption and decryption that use the encryption algorithm. Also, you will also design and evaluate your own random number generator.

The assignment is to implement **a stream cipher**. A stream cipher encrypts an arbitrary length of plaintext. It is described in Section 6.5.2 in Goodrich-Tamassia and the lecture notes of Symmetric Encryption I.

Background

Stream Ciphers

Stream ciphers can be divided into synchronous and self-synchronizing encryption systems. Synchronous stream ciphers need a key (or seed) for the random number generator. If parts of the stream are lost, then the communication needs to restart to decrypt the message stream.

A self-synchronizing stream cipher needs the previous *n* symbols to restart the decryption process.

Pseudo-random Number Generators

A simple PRNG (PseudoRandom Number Generator) is the *linear congruential generator*. It generates uniform random numbers by using modulo arithmetic and a recursive calculation:

$$x_{i+1} = ax_i + b \bmod m$$

We want the period, i.e., the length of the pseudo-random sequence before it repeats itself, to be as large as possible. The period is given by $\Phi(m)$. This means that the period is maximized if m is a prime, which gives the period $\Phi(m) = m-1$. To get the maximum period, a has to be a primitive root of m, that is, the multiplicative order of a should be $\mathrm{ord}_m(a) = \Phi(m)$. This will result in a pseudo-random sequence in the interval $[1, m-1]$, where all numbers are represented. Notice that 0 is not in the sequence and should not be given as a seed to x_0. The offset b does not change the statistical properties.

Statistical tests of randomness

The simplest method is to generate a bitmap (0 or 1 pixels) of say 300 x 300 and plot these as a bitmap. See example at **https://www.random.org/analysis/** (https://www.random.org/analysis/) . The human brain is very good at discovering patterns.

Other methods are the Chi-square test (**https://www.khanacademy.org/math/ap-statistics/chi-square-tests/chi-square-goodness-fit/v/chi-square-statistic** (https://www.khanacademy.org/math/ap-statistics/chi-square-tests/chi-square-goodness-fit/v/chi-square-statistic)) or run-test. As the mathematical statistics course is in the third year, you are not required to use these tests in the assignment.

RC4 ciphers

https://en.wikipedia.org/wiki/RC4 (https://en.wikipedia.org/wiki/RC4)

Implementation

In this assignment, a Stream Cipher implementation consists of one program:

- **StreamCipher** is a program that takes a plaintext message of a sequence of bytes and an encryption key as input**.** The output is a ciphertext consisting of a sequence of bytes.

StreamCipher takes three arguments and should be executed as follows:

```
$ javac StreamCipher.java
$ java StreamCipher <key> <infile> <outfile>
```

The first argument is the encryption key, given as an integer (decimal) number**. StreamCipher** will read the input from the file with the name given as the second argument <infile>, and write the resulting output to a file named <outfile>. All data files have the same format: binary files, i.e., sequences of bytes.

Task 1. Simple Synchronous Stream Cipher

The first task is to write the StreamCipher program that generates a pseudo-random stream of bytes and combines it with an input that consists of plaintext (for encryption) or of ciphertext (for decryption). In the first case, the output will be the ciphertext, and in the second case, the plaintext. You should use the standard PRNG in the Java programming language library (java.util.Random), and use <key> as the seed for the random number generator.

Use the standard PRNG to generate a stream of bytes, and compute the output as the byte-wise exclusive-or (XOR) of the input and the bytes from the PRNG. See the example in the lecture *Symmetric Key Encryption I*. To generate a random byte, use Random.NextInt(256).

Submission

A zip file called Stream1.zip with the following files:

- StreamCipher.java

Task 2. Pseudorandom Number Generator

For the second task, you will design and evaluate your own pseudo-random number generator. Define it as a subclass of the Java Random class (java.util.Random), and call the new class MyRandom. If you override the next() and setSeed() methods of the Random class, your class will inherit any other methods and can use them to, for instance, generate random floating point numbers.

Constructors	
`MyRandom()`	Creates a new random number generator
`MyRandom(long seed)`	Creates a new random number generator using a single long seed
Methods	
`int next(int bits)`	Generates the next pseudorandom number.
`void setSeed(long seed)`	Sets the seed of this random number generator using a single long seed.

A. Design your own pseudo-random number generator based on the Linear Congruential Generator presented in thelecture *Symmetric Key Encryption I*. Choose your own prime number and find suitable constants a, b and seed x_0. Implement it and compare that to the one in the class Random in Java. The background section gives some small tips on how to investigate the randomness of a pseudo-random number generator. Design and analysis of the PRNG is best done in mathematical software such as MATLAB or Mathematica. Describe and analyze your PRNG in a short report of 1-2 pages.
B. Implement StreamCipher with your own PRNG.

Note: Thanks to Java inheritance, you should be able to use your submission for task 1 here, and simply replace “Random” with “MyRandom”. No other changes should be needed in your StreamCipher program.

Submission

A zip file called Stream2.zip with the following files:

- StreamCipher.java
- MyRandom.java
- Report.pdf

Task 3. RC4

Solve task 1 using the RC4 pseudo-random number generator in the stream cipher. Note that you should implement the RC4 PRNG yourself – you cannot use a library implementation of the PRNG or RC4.

The interface to your PRNG is different here, compared to Task 2. To get the next 8-bit word from the PRNG, use MyRandom.next(8). This is the only way of using the PRNG that your implementation need to support, that is, that the next() method is called with the bits parameter set to 8.

Note: This implementation you are doing here is a proper implementation of the RC4 stream cipher. This means that for a given key and plaintext, the output ciphertext will be the same as from any other RC4 implementation.

Submission

A zip file called Stream3.zip with the following files:

- StreamCipher.java
- MyRandom.java

Environment

This assignment is a programming exercise, and you should use Java to solve it. You can use any Java development environment of your choice; however, it is a requirement that the code you finally submit runs on the course virtual machine. Furthermore, your submitted code should be possible to execute directly in a Linux shell, as described above in section "Implementation," *so it should not depend on any IDE tool such as NetBeans or Eclipse*.

Hints and Tips

Reading Input Data

Since you are dealing only with binary data in this assignment, you can use basic I/O streams in Java for reading and writing bytes. For example, you might find the different read/write methods of FileInputStream/FileOutputStream or BufferedInputStream/BufferedOutputStream to be suitable.

Error Handling

If the user gives incorrect parameters or the input data does not match the parameters, this should be detected, and an informative error message should be generated. (Java exceptions do not count as informative error messages.)

Moreover, your program should follow the standard convention that a program terminates (“exits”) with status 0 if the execution of the program is successful, otherwise the status is non-zero. Anything that prevents the program from completing its task is considered an error. In particular, invalid user input is an error, and should result in the program exiting with non-zero status.

The exit status is specified with the exit() system call. See the exit(int) method in java.lang.System.

Organization

This assignment is done individually.

Collaboration

We encourage you to collaborate and discuss with other students, but each student must produce and submit his/her own solution. We will check solutions for originality. If two students submit very similar solutions, it will be treated as a case of plagiarism.

Requirements and Points

For this assignment, you can get a maximum of 100 points. The grading scale is as follows:

- 10 points: your submission compiles correctly, without compilation errors, and your submission meets the requirement above.
- 50 points: Task 1 correctly implemented:
  - Takes a plaintext as a binary file and encrypts/decrypts it correctly.
- 70 points: Task 2 is correctly implemented, and a simple analysis is made.
- 80 points: Task 2 is mathematically analyzed.
- 100 points: Task 3 correctly implemented.

Note that there can be a deduction in points depending on the quality of your solution. If you aim for 100 points, for instance, but we assess that your solution only solves half of the challenges for Task 3 in a satisfactory manner, you might get 90 points.

Submission

Submit your solutions here by uploading the zip files specified under the heading **Submission** in each section above. In other words, upload one zip file for each task you have solved, where each file should be called "Stream*N*.zip", where *N* is the number of the task. For example, if you submit solutions for Task 1 and 2, you should submit the two files Stream1.zip and Stream2.zip.

Each zip file should contain the complete solution for the corresponding task. This means that there may be duplicates where you submit the same material in several files, but that is expected.

You can upload as many times you like before the due date.

Testing

The grading of Task 1, Task 2b and Task 3 of your submission will be made with the assistance of an automated grading tool.

Your implementation will be tested for basic input validation and error handling. It is important that you follow the error handling convention as described above, and exit with non-zero status in case of errors. The automated grading tool will check for this.

Here are some examples of tests:

- Input validation
  - Do the input files exist, and are they readable?
  - Are input values valid?
  - In particular, is the key argument a valid key?
- Can **StreamCipher** process large files (several megabytes)?