Assignment 1: Random Number Generator

This project is due on **Thursday, February 15** at **11:59 p.m.**. Late submissions will be penalized by 10% per day. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your project early.

This is an individual project.

The code and other answers you submit must be entirely your own work. Undergraduate students are bound by the Honor System while graduate students are bound by the Graduate School's expectation of research integrity. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions must be submitted electronically via CS Dropbox.

Introduction

In this project, you will implement your own pseudo-random number generator (PRG) in Java.

We will give you some of the code you need, and we'll ask you to provide certain functions missing from the code we provide. You can download the code we are providing here: https://goo.gl/vtnGEJ.

We are giving you one fully implemented code file, on which you should build all of the crypto functionality you need to do this assignment. The file *PRF.java* gives you access to a pseudo-random function, as described in lecture. This is the only crypto primitive you are allowed to use—any other crypto you use must be built (by you) on top of this file. Specifically, you may not use any other crypto libraries, not even the ones that are part of the standard Java libraries.

In this assignment you will implement two facilities, by modifying two Java code files. You will modify *PRGen.java* to implement a pseudo-random generator class. You will modify *StreamCipher.java* to implement a stream cipher. In each case, we have provided you with a code file in which some parts are "stubbed out." You will replace the stubbed out pieces with code that actually works and provides the required security guarantee. We have put a comment saying *"IMPLEMENT THIS"* everywhere that you have to supply code.

Objectives

- Learn the common mistakes made in PRG design and implementation.
- Understand the risks these problems pose.
- Gain experience writing cryptographic programs.

PRGen

Your PRGen class should implement the following API:

```
public class PRGen extends Random {
    public PRGen(byte[] key)   //creates a new PRGen
    protected int next(int bits) //generates the next pseudorandom number
}
```

You do not need to implement any other methods of the Random class. The long key that is used by the parent Random class will not be used by your PRGen class.

As stated in the Random spec "The general contract of next is that it returns an int value and if the argument bits is between 1 and 32 (inclusive), then that many low-order bits of the returned value will be (approximately) independently chosen bit values, each of which is (approximately) equally likely to be 0 or 1." For example, if you call next(4), it will return an int between 0 and 15. If you call next(32), it will return an int between -2,147,483,648 and 2,147,483,647 (the highest order bit determines the sign).

Your PRGen must obey the following three properties:

- It must be **pseudorandom**, meaning that there is no (known) way to distinguish its output from that of a truly random generator, unless you know the key.
- It must be **deterministic**, meaning that if two programs create generators with the same seed, and then the two programs make the same sequence of calls to their generators, they should receive the same return values from all of those calls.
- It must be **backtracking-resistant**, meaning that if an adversary is able to observe the full state of the generator at some point in time, that adversary cannot reconstruct any of the output that was produced by previous calls to the generator. Note that the .next function in java.util.Random is not backtracking-resistant.

StreamCipher

Your StreamCipher class should implement the following API:

```
public class StreamCipher {
    public StreamCipher(byte[] key, byte[] nonceArr, int nonceOffset)
    public StreamCipher(byte[] key, byte[] nonce)
    public byte cryptByte(byte in)                    //encrypts the next byte
    public void cryptBytes(byte[] inBuf, int inOffset, //encrypts next numBytes
              byte[] outBuf, int outOffset,           //in the inBuf, writing
              int numBytes)                           //results to the outBuf
}
```

This class encrypts or decrypts a stream of bytes, using a stream cipher. (Recall that for a stream cipher, encryption and decryption are the same operation.)

Getting Started

Tips This list may grow in response to Piazza questions.

- Make sure you understand what a PRF is, and how you can use the PRF class to deterministically generate pseudo-random values. See the comments in PRF.java for examples.
- The spec is deliberately vague regarding how you should accomplish each task. There is a significant design component to each problem.
- The bulk of the work for this assignment will be in the design, not the implementation. It shouldn't take many additional lines of code to complete the classes. Our reference solutions are 75, 61 lines of code (the whole file, including everything, even comments, whitespace, brackets, etc.) for *PRGen*, *StreamCipher* respectively.

Advice on Testing Crypto Code As always, it's important to test your code. But you should be aware that crypto code presents different testing issues than other code does. Testing can sanity-check your code, but it can't verify that your code has the desired security properties. For example, if your code is encrypting data for confidentiality, you can test whether the ciphertext is the right size, and you can test whether the ciphertext looks kind of randomish, and you can test whether different plaintexts yield different ciphertexts. But you can't test whether there is a way for an adversary to recover the plaintext. So by all means test your code — if you don't, it's almost certain not to work — but remember that passing the tests is not enough.

Submission Checklist

Upload to CS Dropbox the files listed below. Make sure you have the proper filenames and behaviors.

PRGen

`PRGen.java`	A file containing your implementation of the *PRGen* class.

StreamCipher

`StreamCipher.java`	A file containing your implementation of the *StreamCipher* class.