**USC UPSTATE**

**CSCI 455: Computer Security**

**Spring 2019**

**Lab 3**

**Problem 1**

To generate good pseudo random numbers, we need to start with something that is random; otherwise, the outcome will be quite predictable. Attach your code and screenshots of the output here.

1. Write a program that uses the current time as a seed for the pseudo random number generator. Here “current time” refer to the time as the number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC). In Java, System.currentTimeMillis() method returns the difference between the current time and midnight, Jan 1, 1970 UTC, in milliseconds(you need to divide by 1000 to convert to seconds).
2. Modify your program in part a) that it can generate a random keystream based on the seed. The size of the keystream should contain 16 characters and each character is an integer from 0 to 9. The size of the keystream will be 128 bits that can be used for AES encryption.
3. Suppose Alice used the scheme described above to encrypt a document. Bob somehow see the timestamp of Alice’s encrypted document, which is "2019-02-05 11:59:49 AM". He guessed that the key may be generated within a two-hour window before the file was created. Since the file is a PDF file, which has a header. The beginning part of the header is always the version number. Around the time when the file was created, PDF-1.5 was the most common version, i.e., the header starts with %PDF-1.5, which is 8 bytes of data. The next 8 bytes of the data are quite easy to predict as well. Therefore, Bob easily got the first 16 bytes of the plaintext. Based on the meta data of the encrypted file, he knows that the file is encrypted using aes-128-cbc. Since AES is a 128-bit cipher, the 16-byte plaintext consists of one block of plaintext, so Bob knows a block of plaintext and its matching ciphertext. Moreover, Bob also knows the Initial Vector (IV) from the encrypted file (IV is never encrypted). Can you help Bob to obtain the secrete keystream so that he can decrypt the file? Describe and Justify your method.

Yes. Since bob has part of the plaintext and the ciphertext, it can be reverse engineered to fill out the rest of the cipher. If we analyze the changes from m1 to c1 given the bits that we already know, we can intelligibly decode several parts of the ciphertext until we have uncovered the entire message and keystream.