Problem Set #3

CMSC/Math 456
Instructor: Daniel Gottesman

Due on Gradscope, Thursday, Sep. 21, noon (before start of class)

Overview: The goal of this problem is to break a poorly designed stream cipher. You will write Python scripts to attack the stream cipher in three different contexts.

Reminder: If you collaborate or use any outside resources, remember to cite them as a comment in your submission.

Instructions: You must upload to Gradescope a single Python file named "attack.py" containing all the functions with the specifications given below in the 3 problems. The file attack.py provided to you contains Python functions that implement the Init and Next functions of the stream cipher and the Enc and Dec functions of the pseudo one-time pad based on this stream cipher. It also includes named functions without contents for you to fill in the functions needed. You may use these functions to analyze the stream cipher, to test your attacks, and as subroutines in the attack. You may also reuse code from them in your scripts.

Do not rely on uploading your file at the last minute. If there is a problem with your upload or a bug in your code that prevents the autograder from working, you will be left with a 0 score.

Scoring: Gradescope will pass your files through an autograder to give you a score and some feedback right away. Note that Gradescope has a timeout of 10 minutes for the whole process and the autograder will be running over 50 trials at various lengths, so if your attacks are time-consuming, it could timeout and you will get an overall score of 0. You can, however, resubmit as many times as you like to try to improve your score. (Note that you should be getting the same trial instances each time, however.) However, I strongly recommend you do testing on your own system rather than try it repeatedly on the autograder, as it will be faster and consumes less communal resources.

Your score will be determined by running a sequence of tests as described below. It is possible to pass all tests 100% of the time, but you can also get partial credit if you come up with an attack that works for some lengths of input but not others.

Also, note that if we discover a bug in the autograder after the assignment is opened, it is possible we will have to rerun the autograder, in which case scores could potentially change.

The stream cipher: The stream cipher is somewhat similar to RC4. The state passed between calls of the stream cipher consists of list of 256 bytes (i.e., 256 numbers, each between 0 and 255), plus one additional number between 0 and 255, which can be viewed as a pointer into the list. (This is a difference from RC4, which has 2 pointers.). Another difference is that this stream cipher takes an initial value, which must be in the form of a list of 256 bytes.

Below is a pseudocode description of the stream cipher. If you have some doubts about its meaning, you can consult the actual code in attack.py, which is the same as the code which will be used to generate the stream cipher that you are breaking.

The Init and Next functions are substantially simpler than RC4. Init takes initial value IV (a list of 256 bytes) and key k (a list of some number of bytes) as input and does the following:

1. state = IV
2. for j running up to the length of k, let k_j be the jth byte of k, and update state:
   - new state[i] = old state[$i + k_j$ mod 256]
3. $i = 0$

Init outputs (i, state).

The Next algorithm takes internal state (i, state) as input and does the following:

1. $x = $ state[i]
2. $i = i + $ state[$(i + 1)$ mod 256]

It outputs the byte x as the output of the stream and the updated internal state (i, state).

The Enc function provided to you takes as input IV (which is supposed to be chosen randomly at the time of encryption), a key k, and a message m (which is given as a list of bytes) and outputs ciphertext c, which begins with the IV and then follows with $m_j + x_j$ mod 256 for all j, where m_j is the jth message byte and x_j is the jth byte output from the stream cipher with the IV and k. Dec just inverts this given key k and ciphertext c.

The functions described above allow the key for the stream cipher can be any length, but you may assume that the keys are 16 bytes long for these problems.

Problem #1. Attack via the Pseudorandom Generator (40 points)

Define a function "PRG_attack(IV, x)" which takes as input IV (an initial value which is a list of 256 bytes) and x (a list of bytes of variable length). Your function should return 0 if x is an output of the stream cipher with that IV and some seed and 1 if x was generated by a random process.

Note: Do not return the key value, only if x was random or pseudorandom. If you have an attack which determines the key, that is fine; but it is also acceptable to have an attack which gives the right answer without learning the key.

Your attack will be tested on a number of instances with x being 10 bytes long and 100 bytes long. You can get half credit if your attack works for one of these lengths but not the other.

Problem #2. Attack with a Pair of Possible Messages (40 points)

Define *two* functions "EAV_choose(length)" and "EAV_attack (m0, m1, c)." Together these will implement the EAV security game we discussed in class.

"EAV_choose(length)" takes as input a length, which could in principle be any positive integer, but in practice in this problem set ranges from 10 to 1000. It should return a pair of two messages (m0,m1) which are each lists of bytes with a number of entries equal to the length. You should choose a pair (m0,m1) which you will find easy to distinguish.

"EAV_attack (m0, m1, c)" takes as input two messages (each a list of bytes) and a ciphertext (a longer list of bytes generated via the Enc function provided). You may assume that m0 and m1 are the messages you chose via "EAV_choose"; they are provided to make sure your attack has access to them. "EAV_attack" should return 0 if c is an encryption of m0 for some key and 1 if c is an encryption of m1 for some key.

Your attack will be tested on a number of instances with x being 10 bytes long and 100 bytes long. You can get half credit if your attack works for one of these lengths but not the other.

Problem #3. Attack with a List of Possible Messages (40 points)

Define a function "decrypt(m_list, c)" which takes as input a list of possible messages (each of which is itself a list of bytes) and a ciphertext (a list of bytes) which is an encryption of one of the messages using the stream cipher. The function should return an index into the list of messages to specify which message encrypts to give the ciphertext. The list of messages could in principle be any length, but the autograder will test your program with a list of 20 possible messages. The length of each message in the list will be the same, and again that length could in principle have any value.

Your attack will be tested on a number of instances with x being 10 bytes long and 100 bytes long. You can get half credit if your attack works for one of these lengths but not the other.