**CS 4732/57322 Homework #4**

***Due electronically by midnight July14th, 2025***.

For submission, if done on paper please scan and submit as a pdf. If done in word, please submit the .docx or .doc format.

**IMPORTANT**: Clearly indicate outside resources utilized and sign below. Failure to cite use of outside resources will be reported for appropriate disciplinary actions. Note that discussions with other students are encouraged; copying – with or without modifications – is unacceptable and will also be reported.

I discussed one or more problems with the following people:

I hereby certify that any outside resources utilized, other than the textbook and class materials, are clearly cited. All other material I provide for this homework submission is my own original work.

Jamie Harris

*Printed name*

1. (8 points) Explain what a meet-in-the-middle attack is in broad terms. Suppose we tried to increase the security of AES by encrypting twice, how much space do you think it would need (describe it in gigabytes).

A meet in the middle attack finds a key in a double-encryption security system by leveraging the fact that the decryption of the final ciphertext is equivalent to the encryption of the initial message, so it works from both sides at once to find potential key matches.

At its worst case, the AES algorithm needs 240 bytes for the key expansion, 16 bytes per round for 14 rounds for each key, 16 bytes for the plain text message, 16 bytes for the ciphertext, and 32 bytes for the original key. 240+16+16+32+16\*14=528 bytes. Multiply this by 2 for the second encryption, and a double-encrypted AES algorithm would require about 1056 bytes, or just over 1 gigabyte of space.

2. (12 points) While it is desirable to have a PRNG have a full period, that does not guarantee good randomness. Consider a linear congruential generator using the settings of a = 6, m = 13 and another one with a = 7, m = 13. For both cases, c=0. Write out the full two sequences out for their periods. After looking at the sequences, do you feel more comfortable with one or the other?

x1=(6\*1+0) mod 13 = 6

x2=(6\*6+0) mod 13 = 10

x3=(6\*10+0) mod 13 = 8

x4=(6\*8+0) mod 13 = 9

x5=(6\*9+0) mod 13 = 2

x6=(6\*2+0) mod 13 = 12

x7=(6\*12+0) mod 13 = 7

x8=(6\*7+0) mod 13 = 3

x9=(6\*3+0) mod 13 = 5

x10=(6\*3+0) mod 13 = 4

x11=(6\*4+0) mod 13 = 11

x12=(6\*11+0) mod 13 = 1

x1=(7\*1+0) mod 13 = 7

x2=(7\*7+0) mod 13 = 10

x3=(7\*10+0) mod 13 = 5

x4=(7\*5+0) mod 13 = 9

x5=(7\*9+0) mod 13 = 11

x6=(7\*11+0) mod 13 = 12

x7=(7\*12+0) mod 13 = 6

x8=(7\*6+0) mod 13 = 3

x9=(7\*3+0) mod 13 = 8

x10=(7\*8+0) mod 13 = 4

x11=(7\*4+0) mod 13 = 2

x12=(7\*2+0) mod 13 = 1

I would feel more comfortable with the settings of a=6, m=13, as a=7 leads to certain divisibility chains (12 to 6 to 3, 8 to 4 to 2 to 1, etc.)

3. (10 points) What is the difference between condition testing and health testing in regards to TRNGs. Give an example of a bitstream that would pass health testing but not conditioning and explain why.

Condition testing is used to determine if a generated string is sufficiently random. That is, that it is not deterministic or structured enough to be easily predicted. Health testing is used to make sure that as a number is being generated, there are no problems with the underlying algorithm or entropy source that could lead to failed condition tests.

An example that would pass health testing but not condition testing is 0010110100101101. This string has a uniform distribution of ones and zeros with no large chunks of either, but it has a fairly obvious structure to it, with 0010 being followed by 1101 and vice versa.

4. (8 points) In regards to whether or not a sequence is random, two criteria would be uniform distribution and independence. Define the two, then give me an example of a sequence (at least 10 bits long) that has a uniform distribution but violates independence. Relate to me the independence.

Uniform distribution means that there should be an approximately equal number of ones and zeros, and independence means that no substring should be able to be inferred from any other substring.

An example of a string that has uniform distribution but no independence is the string 101010101010… , since it can be inferred that if there is a “1”, there will be a “0” immediately after it.

5. (6 points) Is it possible to perform encryption on multiple blocks of plaintext in parallel in CBC mode? What about decryption? Justify your answer.

You would not be able to perform parallel encryption with CBC, because the plaintext for each block is exclusive-or’d with the resulting ciphertext from the previous block before going through the encryption function, therefore you need each resulting ciphertext before being able to proceed to the next block. However, decryption can be done in parallel, because the ciphertext required to decrypt each block is already available, so all blocks could be decrypted simultaneously.

6. (6 points) In transit a bit is flipped in a ciphertext that was created using a block cipher in CFB mode. What will be the result when we decrypt the text? Suppose that the message is 10 blocks in size, with the 4th block having a bit that was flipped.

When you decrypt the text, there will be two blocks that are erroneous. Because of a flipped bit in the 4th block, the plain text for block 4 and block 5 will have errors.

7. (10 points) Why is it a bad idea to reuse a stream cipher key (in this case, the stream cipher is just exclusive or’ing your key with the plaintext)? Explain with an example using two plaintext bitstreams of 8 bits involving the reuse of a key bitstream of also 8 bits. Then XOR the resulting two ciphertexts. What property does it have? This might be hard to see, but look at the two plaintexts in relation to it.

KEY: 00101101

TEXT1: 11011001

TEXT2: 01011011

CRYPT1: 11011001⊕00101101 = 11110010

CRYPT2: 01011011⊕00101101 = 01110000

CRYPT1⊕CRYPT2 =10000010=TEXT1⊕TEXT2

KEY: 10100001

TEXT1: 10101101

TEXT2: 01011010

CRYPT1: 10101101⊕10100001 = 00001100

CRYPT2: 01011010⊕10100001 = 11111011

CRYPT1⊕CRYPT2 =11110111=TEXT1⊕TEXT2

When you use a stream key in this manner, the exclusive-or of the two resulting ciphertexts will be the same as the exclusive-or between the initial messages, leading to the ability to easily find the key by sending in several messages through encryption and working out their exclusive-ors.