**Applications of Cryptography CSCE 4050/5050 (Spring 2025)**

**Homework 4**

1. [**AES S-boxes**] For AES, compute the respective S-box outputs for the following inputs:
   1. 0xc3
   2. 0x7e

**Tip:** The description of the AES S-box may be found, e.g., here: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>   
You may simply use the table representation of the S-box to find answers. Remember to confirm correctness of your answers using the inverse S-box table (provided later in the document).

[10 points] Rubric: 5 points per item.

**Answer:**

Input = 0xc3

In S-Box, row c, column 3 = 2e which maps back to c3 in InvsBox table

Therefore, output of 0xc3 = 0x2e

Input = 0x7e

In S-Box, row 7, column e = f3 which maps back to 7e in InvSBox table

Therefore, output of 0x7e = 0xf3

1. [**AES-128 Key Schedule**] When writing “AES-128”, we are referring to AES with 128-bit key.  
   Suppose that the AES-128 key K is 0123456789abcdef0123456789abcdef (in hex). Compute the round key K1 according to the AES key schedule (note that K0=K for AES-128). Explain your answer.

**Tip:** You may find this tutorial useful:   
<https://braincoke.fr/blog/2020/08/the-aes-key-schedule-explained/>

**Answer:**

Given, K0 = 0123456789abcdef0123456789abcde

Splitting K0 into 32-bits words

w0 = 01234567

w1=89abcdef

w2​=01234567

w3​=89abcdef

1. RotWord on w3​= 89abcdef

Output = abcdef89

1. SubWord:

ab→62, cd→bd, cd→bd, ef→df, 89→a7

1. Rcon:

62bddfa7⊕01000000=63bddfa7

Now,

w4​= 63bddfa7 ⊕ w0 = 63bddfa7⊕01234567 =629e9ac0

w5​=w1​⊕w4​**=** 89abcdef ⊕ 629e9ac0 = eb35572f

w6​=w2​⊕w5​ **=** 01234567⊕ eb35572f = ea161248

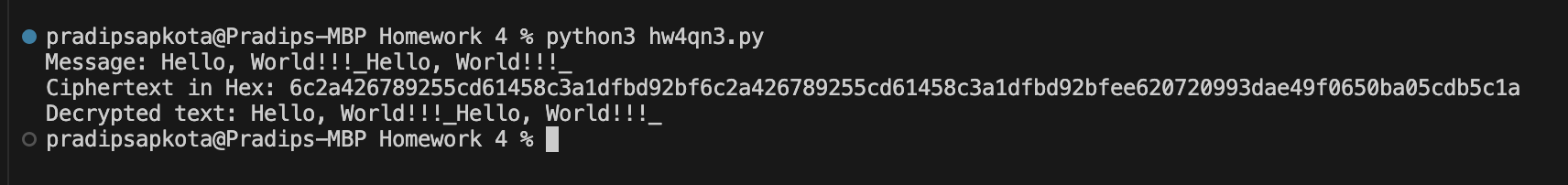
w7=w3​⊕w6​ **=** 89abcdef ⊕ ea161248 = 63bddfa7

K1 = [w4, w5, w5, w6] = [629e9ac0, eb35572f, ea161248, 63bddfa7]

K1 = 629e9ac0eb35572fea16124863bddfa7

1. [**AES-ECB-128 encryption: implementation**] Write a program that encrypts the following plaintext: “Hello, World!!!\_Hello, World!!!\_”. (The goal is to have a message consisting of two identical   
   16-byte blocks, so make sure to use an encoding with one byte per character.) Encrypt this plaintext using AES-128 in the electronic codebook (ECB) mode. The key will be hardcoded into the program. (Normally, it will be chosen randomly, but it is not required for this assignment.) For certainty, you may set the key as your Student ID in hex repeated four times. For example, if your Student ID is 12345678, set the key as 0x12345678123456781234567812345678. Print the message (in text) and the resulting ciphertext (in hex) to the screen. Decrypt the ciphertext and print the resulting plaintext to the screen (of course, it must match the original plaintext).

**Ans:**

****

1. [**More on** **AES-ECB-128 encryption: implementation**] Encrypt the following plaintext:  
   “Hello, World!!!\_Hello, Alice!!\_”. (Now, the goal is to have a message consisting of two 16-byte blocks which are not identical. Use the same encoding as in the previous question.) The key must also be the same as in the previous question. Print the resulting ciphertexts to the screen. Explain your observations.

**ANS:**

**A computer screen with white text

Description automatically generated**

**Observation:**

First, different ciphertext blocks are produced with shorter length in Q4 than in Q3 (where ciphertext blocks are also identical). This is because the message in Q4 is 31 bytes where PKCS7 padding added 1 byte to make length of 32 bytes (2 blocks) to make a multiple of AES block size (16 bytes) whereas in Q3, plaintext was 32 bytes so, 16 bytes of padding were added that results ciphertext in 48 bytes (3 blocks) automatically making the ciphertext of bigger length as shown in the above screenshot.

|  |  |  |  |
| --- | --- | --- | --- |
| Message | Plaintext bytes | Ciphertext | Ciphertext bytes |
| Hello, World!!!\_Hello, World!!!\_ | 32 | 6c2a426789255cd61458c3a1dfbd92bf 6c2a426789255cd61458c3a1dfbd92bfee620720993dae49f0650ba05cdb5c1a  (identical blocks) | 48 |
| Hello, World!!!\_Hello, Alice!!\_ | 31 | 6c2a426789255cd61458c3a1dfbd92bf9b5babad0d14987535adbdd1fef70150  (different blocks) | 32 |

1. [**AES-ECB: semantic security?**] Answer the following question: Is AES in the ECB mode semantically secure? Explain your answer. Note that in this question, we focus on messages which are longer than one block. In your answer, you must consider the notion of semantic security for one-time keys from Lecture 3-2.

**Hint:** You may find the ciphertexts produced in the two previous questions useful.

**Answer:**

AES in ECB mode is not semantically secure because When blocks of plaintext are identical, it shows patterns that let attackers identify ciphertexts and learn more about the plaintext. The same key is used to independently encrypt each 16-byte block in ECB mode. The identical ciphertext block will be generated if the same plaintext block occurs more than once in a message. An attacker can use the patterns found in the plaintext to deduce details about the original message without having to decrypt it.

In previous questions 3 and 4, first two 16-blocks are identical in ciphertext because of the two identical 16-block plaintexts (“Hello, World!!!”). **6c2a426789255cd61458c3a1dfbd92b** in ciphertext in Q3 is repeated up to 32 hex characters whereas the blocks in Q4 ciphertext (**6c2a426789255cd61458c3a1dfbd92bf9b5babad0d14987535adbdd1fef70150**) are different.

1. [**Pseudorandom generator based on AES: implementation**] Write a program which generates a pseudorandom sequence of 256 bytes using AES as a PRF as described in the end of Lecture 4-2. Your program will test the quality of this sequence using the Frequency (Monobit) Test from NIST SP 800-22 (rev. 1a): <https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762>  
   (see Sec. 2.1) – the same test as you ran in Homework 3, Question 2. Print the values computed as described in that section. Use the decision rule at the 1% level (see Sec. 2.1.5) to conclude if the sequence is random or not. Explain your decision.

**ANS:**

**A black screen with white text

Description automatically generated**

The NIST Frequency Test determines if the keystream's 1s and 0s are balanced, as would be expected in a random sequence. The P-value, in this case 0.536102, is determined by the test and compared with the significance level of 1% (0.01). Since 0.536102 >= 0.01, we can conclude the sequence as random that means the pseudorandom sequence generated using AES as a PRF passes the NIST Frequency Test at the 1% significance level.

1. [**Insecure PRG based on PRF**] Assume that AES-128 is a secure PRF. A software engineer Alice needs to generate a pseudorandom sequence “s” of 129 bits. She knows that AES(k,m) for an arbitrary key “k” and plaintext “m” will give her 128 pseudorandom bits. Instead of using an approach from the above question, she decides to make the following shortcut: compute an XOR of those 128 bits to obtain the 129-th bit. Specifically, for AES(k,m) = s[1..128], set s[129] = s[1] ⊕ s[2] ⊕ … ⊕ s[128].

Task: Show that the above PRG is insecure.

**Tip:** You will consider security of a PRG which is based on PRF. Hence, you will construct a distinguisher for this PRG and compute its advantage.

**Answer:**

Alice’s PRG is not secure because the 129th bit is not random; it is the deterministic XOR of the first 128 bits. Hence there is a linear dependency such that the last bit is predictable if the first 128 bits are known. Theoretically, each of those bits should have no statistical correlation with any of the other bits in a truly random 129-bit string but because of the way Alice method of decryption works, this independence is broken which makes the output distinguishable from random.

Here, 128 pseudorandom bit using AES as a PRF

AES(k,m) = s[1]s[2]….s[128]

S[129] = s[1] ⊕ s[2] ⊕….⊕s[128]

And output becomes s = s[1]s[2]……………s[129]

**A Distinguisher:**

For given Alice’s pseudorandom 129-bit sequence ‘s’,

Let’s check ‘s’ with s[129] = s[1] ⊕ s[2] ⊕….⊕s[128]

If the condition is true, the sequence is likely from Alice’s PRG (output 1)

If the condition is false, the sequence is likely random (output 0)

**Advantage:**

For the Alice’s PRG output, Pr[D(PRG)=1]=1

For random sequence, Pr[D(r)=1] =1/2

AdvPRG[D] = | Pr[D(PRG)=1]=1 - Pr[D(r)=1] | = |1-1/2| = 1/2

Since the advantage = 1/2 , it is non-negligible which makes Alice’s PRG not secure.