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

**Homework 5**

1. [**AES-CTR: implementation**] Write a program, which encrypts a message “Pay $2000 to Bob” (without quotes) using the AES cipher in the randomized CTR mode. Your program must print the following on the screen: the plaintext in text and in hex, the key, nonce, and the resulting ciphertext, all in hex. In a practical application, the key and nonce are chosen at random. In this assignment, for simplicity, they will be hardcoded into your program. The key and nonce will be selected as follows: take the last two letters of your student ID, for **example**, if your student ID is 12345678, then the last two letters are “78”. The key will be “0x78..78” and the nonce will be “0x87..87”. The ciphertext (in hex) will be saved into a text file “ciphertext.txt”. Next, your program will decrypt the ciphertext and print the resulting plaintext to the screen. Note that in all the above operations, you will use the same key and nonce, which are chosen as described above.

**IMPORTANT NOTE:** Make sure to select the key and nonce as described above (they will be different for all students). An incorrect selection will result in reduced grades.

**The output screenshot:**

**A screen shot of a computer program

Description automatically generated**

1. [**AES-CTR is malleable: implementation**] Copy the file “ciphertext.txt” (which was produced in the previous question) to a file “ciphertext1.txt”. Then manually modify it so that the decryption would result in “Pay $7000 to Bob”. Print the modified “ciphertext1.txt” to the screen (in hex). Run your decryption program from the previous question and confirm that the decrypted plaintext is now modified as expected. Explain how you modified the ciphertext and why your attack was successful.

**The output screenshot:**

**A black screen with white text

Description automatically generated**

Well, I manually modified the 6th byte of the original ciphertext (a71898bf0ada1796ee912cfb19cf9a01) i.e. **da** to **df** making modified ciphertext as a71898bf0adf1796ee912cfb19cf9a01. Using the following equation

Keystream = ciphertext XOR plaintext = 0xda XOR 0x32 =0xe8

New ciphertext = keystream XOR new plaintext = 0xe8 XOR 0x37 = 0xdf

The attack was successful because AES ciphers in CBC mode does not provide the integrity protection. In CBC mode, each plaintext block is XORed with the previous ciphertext block before encryption. This means that modifying a byte in a previous ciphertext block directly alters the corresponding plaintext byte in the next block upon decryption. Flipping the ciphertext's bits, decrypted plaintext can be manipulated without knowing the encryption key. By changing the byte which corresponds to ‘5’ to ‘7’ without a key, we altered the decrypted plaintext.

1. [**AES-CBC: implementation**] Perform the same steps as in Question 1, but now implement encryption/decryption of the message “Pay $2000 to Bob” on the AES cipher in the randomized CBC mode.

**The output screenshot:**

**A screenshot of a computer program

Description automatically generated**

1. [**Plaintext Padding**] When using the block cipher, a plaintext must be padded to ensure that its length is a multiple of the block length. Suppose that the AES cipher with a 128-bit key is used and that PKCS #7 (see <https://www.rfc-editor.org/rfc/rfc5652#section-6.3>) is used for padding.   
   Now, let the plaintext be 0123456789abcdef0123456789ab in hex.  
   Task: write the plaintext (in hex) which is padded according to PKCS #7.

**Answer:**

Given plaintext in hex:0123456789abcdef0123456789ab which is 28 hex characters (14 bytes)

For AES, we need blocks of 16 bytes ( multiple of the block length).

So, lets calculate the padding length according to PKS #7 for 128-bit key of AES cipher.

K k . . . . k k --- if lth mod k = 0

For that,

K – (lth mod k) = 16- (14 mod 16) = 16 – 14 = 2

That means the padding to given plaintext consists of two bytes of 02.

Therefore,

The plaintext (in hex) which is padded according to PKS #7 is:

**0123456789abcdef0123456789ab0202**

1. [Ciphertext Stealing] Consider the ciphertext stealing mode presented at the end of Lecture 5-2. Explain why the decryption algorithm works correctly even though it may seem that the fragment Cn-1\*\* is not included in the ciphertext.

**Answer:**

In the CBC-CS1 (Cipher Block Chaining with Ciphertext Stealing) mode, the decryption algorithm works correctly despite the apparent absence of the fragment Cn-1\*\* in the ciphertext because:

During encryption, CBS-CS1 avoids padding by rearranging ciphertext blocks. The penultimate ciphertext block Cn−1 is truncated, and the final plaintext block Pn is joined with its truncated component (Cn−1∗∗) to create the final ciphertext block Cn.

i.e.

Pn′​=Pn​∥(Cn−1∗∗​⊕ Pn​) and encrypt Pn′​⊕Cn−1 for final ciphertext block Cn.

Here, Cn-1\*\*  is embedded into Cn via XOR but not explicitly sent which gives us the ciphertext as

C1​,C2​,…,Cn−1′​ (truncated), Cn

During decryption, the algorithm starts by processing while both the truncated piece Cn−1∗∗ and the final plaintext fragment Pn are recovered when decrypting Cn. The complete penultimate ciphertext block Cn−1 is then rebuilt using Cn−1 ∗∗, which is decoded and added to the preceding block to retrieve Pn−1.

i.e.

D(Cn) = P’n ​⊕Cn-1

P’n = D(Cn) ⊕ Cn-1 ( to isolate P’n)

While splitting P’n into original plaintext (Pn) and Cn-1\*\*  where Cn-1\*\*  = P’n ⊕ Pn

Now, Cn−1​=Cn−1′​∥Cn−1∗∗ then we can decrypt it to get Pn-1

Here we can see, even though Cn-1\*\*  is not explicitly transmitted, it is embedded within Cn.

Therefore, the decryption algorithm works correctly due to the cyphertext stealing mechanism embeds critical fragments (Cn-1\*\*  ) within the ciphertext block.

1. [**CBC with Predictable IV**] Let us consider why a predictable IV is a problem for CPA security in the CBC mode. Suppose that in the semantic security game with many-time key (a.k.a. the CPA security game), the adversary knows IV in advance (that is before deciding on the plaintexts m0 and m1). Show the adversary who wins the CPA security game. Explain your answer.

Note: We assume that IV is still chosen at random for each ciphertext. Also, the IV is selected from a large set, so it is very unlikely to repeat.

**Answer:**

The use of a predictable Initialization Vector (IV) in CBC mode compromises the security of CPA (Chosen Plaintext Attack) since it enables an adversary to control the encryption procedure and distinguish between encrypted communications.

Before encryption in CBC mode, each block of plaintext is XORed with the block of ciphertext that came before it. The initial block of plaintext is XORed with the IV. The adversary can guess this value if the IV is predictable.

Suppose the adversary knows the IV that will be applied to the challenge ciphertext, which is either m0 or m1 encrypted. The attacker can create m0;

Such that IV ⊕ m0 = the value they already encrypted using the encryption oracle.

For example, the adversary knows that C1 = E (IV1 ⊕ P) if they have already encrypted a plaintext P with a random IV (IV1 ​).

Then adversary performs XOR for m0 as;

m0 = IV ⊕ (IV1⊕ P).

This guarantees that;

IV ⊕ m0 = IV1 ⊕P = C1, therefore encrypting m0 will result in C1 once more.

After generating the challenge ciphertext (encrypting either m0 or m1), the adversary can verify whether the first ciphertext block corresponds to C1.

If it matches, adversary can conclude m0 is encrypted, otherwise m1 is encrypted.

This is how the adversary can win the CPA security game if they know IV in advance before deciding on m0 and m1.