**Problem 1**

**a) Assume a message m is encrypted using the Shift cipher and the resulting ciphertext is as follows?**

**JBCRCLQRWCRVNBJENBWRWN**

**What would be the values of m and key k in this case? Please justify your answer. (Hint: you need to do an exhaustive search over 26 possible keys. Extra credit if you write your own code to perform the analysis.)**

Shift -9

ASTITCHINTIMESAVESNINE

**Code**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class CaesarCipher {

private final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

public String decrypt(String cipherText, int shiftKey)

{

cipherText = cipherText.toLowerCase();

String plainText="";

for(int i=0;i<cipherText.length();i++)

{

int charPosition = this.ALPHABET.indexOf(cipherText.charAt(i));

int keyVal = (charPosition-shiftKey)%26;

if(keyVal<0)

{

keyVal = this.ALPHABET.length() + keyVal;

}

char replaceVal = this.ALPHABET.charAt(keyVal);

plainText += replaceVal;

}

return plainText;

}

}

class CaesarDemo {

public static void main(String args[])

{

String plainText = "studentitzone";

int shiftKey=9;

CaesarCipher cc = new CaesarCipher();

String cipherText = "JBCRCLQRWCRVNBJENBWRWN";

System.out.println("Your Cipher Text :" + cipherText);

String cPlainText = cc.decrypt(cipherText,shiftKey);

System.out.println("Your Plain Text :" + cPlainText);

}

}

**Output:**

Your Cipher Text: JBCRCLQRWCRVNBJENBWRWN

Your Plain Text: astitchintimesavesnine

**b)**

**2.7) Prove that Definition 2.1 implies Definition 2.4. (Hint: Use Exercise 2.6 to argue that perfect secrecy holds for the uniform distribution over any two plaintexts (and in particular, the two messages output by A in the experiment). Then apply Lemma 2.3.**

From the exercise 2.6, we know that for every message m € M with non –zero probability, the scheme is perfectly secret.

That is for any two plan texts m0 € M and m1 € M, the scheme generates a cipher text c with probability greater than 1.

That is, Pr[M=m0|C=c1] =Pr[C=c1]

                Pr[M=m1|C=c2]=Pr[C=c2]

If the two plain texts m0=m1, then the scheme produce the same value c=c1=c2.

That is, Pr[C=c1]=Pr[C=c2]

From the lemma 2.3 ,

Pr[C=c|M=m0]= Pr[C=c|M=m1]

Assume the scheme is perfectly secret. That is, the for a message with non-zero probability Pr[M=m0] produces a cipher that is fixed with probability Pr[C=c1]

Similarly Pr[M=m1] , scheme produces Pr[C=c2].

Thus, the generating cihper value is depends on the message probability.

Pr[M=m0|C=c1]=Pr[M=m0]

And

Pr[M=m1|C=c2]=Pr[M=m1]

But if m0=m1, Pr[M=m0|C=c1]= Pr[M=m1|C=c2] and c1=c2

Therefore, for every message over M, Pr[M=m|C=c]=pr[M=m]

**Problem 2**

**a)**

**3.22 Show that the CBC, OFB, and counter modes of encryption do not yield CCA secure encryption schemes.**

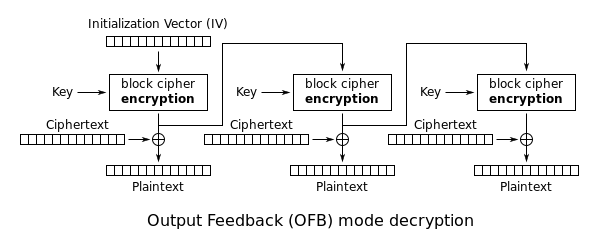
Proof The adversary A is given input 1n and oracle access to Enck(·) and Deck(·), then it outputs m0 = 0n and m1 = 1n to challenge these schemes

• Cipher Block Chaining (CBC) mode

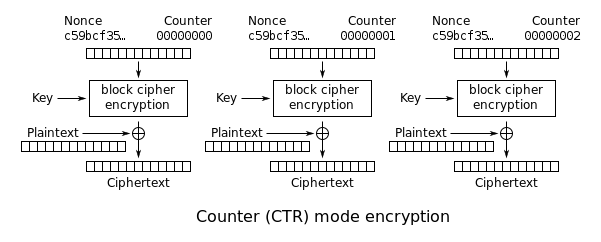


Upon reception of a challenge ciphertext (c, IV), the adversary flips the first bit of c and invokes the decryption oracle to acquire m’. Since Fk is a pseudorandom permutation m’ should differ from either m0 or m1 in only one bit. Such that the adversary can easily determine which message is encrypted.

• Output Feedback (OFB) mode



Counter (CTR) mode



The adversary flips the first bit of the challenge ciphertext c and invokes the decryption oracle to acquire m’ which should differ from either m0 or m1 in only one bit. Then A can easily tell which message is encrypted.

Therefore none of these modes yields chosen-ciphertext secure encryption schemes

**Problem 3**

**a)**

**4.8 Show that the basic CBC-MAC construction is not secure when used to authenticate messages of different lengths:**

This is the basic theorem for basic CBC-MAC

Let F be a pseudorandom function, and fix a length function ℓ. The basic CBC-MAC construction is as follows:

• Gen: on input 1n , choose k ← {0, 1}n uniformly at random.

• Mac: on input a key k ∈ {0, 1}n and a message m of length ℓ(n)·n, do the following (we set ℓ = ℓ(n) in what follows):

1. Parse m as m = m1, . . . , mℓ where each mi is of length n.

2. Set t0 := 0n . Then, for i = 1 to ℓ:

Set ti := Fk(ti−1 ⊕ mi).

Output tℓ as the tag.

• Verify: on input a key k ∈ {0, 1}n , a message m, and a tag t, do: If m is not of length ℓ(n) · n then output 0. Otherwise, output 1 if and only if t ?= Mack(m).

There is the main know exploit for CBC-MAC when the length of the message is not fixed:

First, get a MAC *T* on a message M1. Now XOR the tag *T* into the first block of some arbitrary second message M2, and get a MAC on the modified version of M2. The resulting tag T' turns out to be a valid MAC for the combined message (M1 || M2).

This is because an attacker who knows the correct message-tag (i.e. CBC-MAC) pairs for two messages (m, t) and (m', t') can generate a third message m''whose CBC-MAC will also be  t'. This is simply done by XORing the first block of m' with t and then concatenating m with this modified m'; i.e., by making m'' = m \| [(m_1' \oplus t) \| m_2' \| \dots \| m_x']. When computing the MAC for the message m'', it follows that we compute the MAC for m in the usual manner as t, but when this value is chained forwards to the stage computing E_{K_\text{MAC}}(m_1' \oplus t) we will perform an exclusive OR operation with the value derived for the MAC of the first message. The presence of that tag in the new message means it will cancel, leaving no contribution to the MAC from the blocks of plain text in the first message m: E_{K_\text{MAC}}(m_1' \oplus t \oplus t) = E_{K_\text{MAC}}(m_1') and thus the tag for m'' is t'.

**Problem 4**

**a) Suppose you want to verify the integrity of certain \_les, say system \_les or software packages downloaded from online. Explain how you can achieve this using MAC and hash functions separately. Also, describe the advantages and disadvantages in using each of this technique with respect to the verification of file integrity.**

With MAC we use the message + secret key and that produces the tag which then it is used to verify the integrity of the file. So in this case we can generate our tag and then when the message is downloaded verify that by using the secret key with the downloaded message we get the same tag. If so, that will verify the message integrity.

With HASH we can use the Hash function and calculate a digest, then after the transmission of the message is complete we can calculate the digest again, and compare both digests the one before and the one after, and if they are the same then the message integrity has been verified

Contrary to hash functions where everything is known and attackers are fighting against mathematics, MAC make sense in models where there are entities with knowledge of a secret. What we expect from a good MAC is unforgeability: it should be infeasible to compute a pair message+MAC value which successfully verifies with a given key *K* without knowing *K* exactly and in its entirety.

Hash functions and MAC are thus distinct kind of algorithms with distinct properties and used in really distinct situations.

**b) Can we use a hash function that is not collision resistant to construct a MAC scheme?**

No, because it is not secure. If the function is not collision resistant then an adversary that fins m0 != m1 s.t H(m0) = H(m1)

Then Sbig is insecure under a 1-chosen message attack, because the adversary:

First will ask for t <- S(k, m0)  
and then output (m1, t) as forgery.

This is why we must always used collision resistan hash functions to construct a MAC scheme