Здесь и далее используется AES с длиной ключа 128 бит.

## 1. Реализовать функцию на вашем языке программирования со следующим интерфесом:

byte[] AesBlockEncrypt(byte[] key, byte[] data, bool isFinalBLock, string padding), где

key - байтовое представление ключа блочного шифра

data - блок для шифрования

isFinalBLock - флаг того, что передан последний блок шифруемого открытого текста

padding - вид дополнения, принимает значение PKCS5.

В ходе реализации необходимо пользоваться стандартной или общеизвестной реализацией алгоритма AES. Если библиотера не поддерживает одноблочное шифрование AES, необходимо воспользоваться режимом ECB.

Аналогично реализовать функцию расшифрования AesBlockDecrypt.

Пример одноблочного шифрования на C#.

namespace AesExample  
{  
 using System;  
 using System.Security.Cryptography;  
 using System.Text;  
  
 internal class Program  
 {  
 private static void Main(string[] args)  
 {  
 // Your Key here, 16 bytes  
 byte[] key =  
 { 0x73, 0x69, 0x78, 0x74, 0x65, 0x65, 0x6e, 0x2d,  
 0x62, 0x79, 0x74, 0x65, 0x2d, 0x6b, 0x65, 0x79 };  
  
 // Plaintext, 16 bytes string, utf-8  
 string stringPt = "sixteen-byte-msg";  
  
 // Raw Plaintext, bytes  
 byte[] pt = Encoding.UTF8.GetBytes(stringPt);  
  
 // Resulted Ciphertext will be here  
 byte[] ct = new byte[16];  
  
 // Create new AES instance  
 using(Aes aes = new AesCryptoServiceProvider())  
 {  
 // Select Encryption mode  
 aes.Mode = CipherMode.ECB;  
  
 // Create encryptor with your key and zero IV  
 using (var aesEncryptor = aes.CreateEncryptor(key, new byte[16]))  
 {  
 // Transform one block  
 aesEncryptor.TransformBlock(pt, 0, 16, ct, 0);  
 }  
  
 // Get hex-string representation of Ciphertext  
 string hex = BitConverter.ToString(ct);  
 Console.WriteLine(hex.Replace("-", ""));  
 }  
 }  
 }  
}

c++

#include <stdio.h>   
#include <openssl/aes.h>   
  
static const unsigned char key[] = {  
 0x73, 0x69, 0x78, 0x74,   
 0x65, 0x65, 0x6e, 0x2d,  
 0x62, 0x79, 0x74, 0x65,  
 0x2d, 0x6b, 0x65, 0x79  
};  
  
int main()  
{  
 unsigned char text[]="sixteen-byte-msg";  
 unsigned char enc\_out[80];  
 unsigned char dec\_out[80];  
  
 AES\_KEY enc\_key;  
  
 AES\_set\_encrypt\_key(key, 128, &enc\_key);  
 AES\_encrypt(text, enc\_out, &enc\_key);   
  
 int i;  
  
 for(i=0;\*(enc\_out+i)!=0x00;i++)  
 {  
 printf("%X ",\*(enc\_out+i));  
 }  
 printf("\n");  
  
 return 0;  
}

python

from Crypto.Cipher import AES  
cipher = AES.new(b'sixteen-byte-key',AES.MODE\_ECB)  
cipher.encrypt(b'sixteen-byte-msg').hex()

## 2. Реализовать режимы ECB, CBC, CFB, OFB, CTR с использованием функции AesBlockEncrypt.

Реализовать интерфейс  
byte[] AesEncrypt(byte[] key, byte[] data, string mode, byte[] iv = null), где

key - байтовое представление ключа блочного шифра.

data - массив байт для шифрования

mode - режим шфирования, допустимые значения ECB, CBC, CFB, OFB, CTR. Может быть задан через Enum на c#.

iv - вектор инициализации или начальное заполнение счётчика в указанном режиме. Если значение не передано, или пуредано значение null (или пустой массив), но режим треюует использования IV или счётчика - значение должно быть сгенерировано (через отдельный метод, с использованием системного криптографически стойкого генератора).

Для генерации ключей и IV использовать стойкие стандартные генераторы. В качестве дополнения использовать PKCS5 padding (дополнение, см [википедия](https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS%235_and_PKCS%237))

**IV должен быть случайным для режимов CBC, CFB, OFB.**

**Выбор начального заполнения счётчика для режима CTR должен быть согласован с** [**rfc3686**](https://tools.ietf.org/html/rfc3686#page-7)**.**

При использовании IV (или счётчика) шифртекст дополняется им с начала сообщения.

c = IV || E(k, m)

## 2.5 Использовать реализации режима CBC в вашем языке программирвоания для валидации вашей реализации режима CBC.

## 3. Расшифровать следующие шифртексты:

Режим CBC и CTR. IV = 16 байт. IV добавлен к зашифрованному тексту в начале. PKCS5 padding

CBC key: 140b41b22a29beb4061bda66b6747e14  
CBC Ciphertext 1: 4ca00ff4c898d61e1edbf1800618fb2828a226d160dad07883d04e008a7897ee2e4b7465d5290d0c0e6c6822236e1daafb94ffe0c5da05d9476be028ad7c1d81  
  
CBC key: 140b41b22a29beb4061bda66b6747e14  
CBC Ciphertext 2: 5b68629feb8606f9a6667670b75b38a5b4832d0f26e1ab7da33249de7d4afc48e713ac646ace36e872ad5fb8a512428a6e21364b0c374df45503473c5242a253  
  
CTR key: 36f18357be4dbd77f050515c73fcf9f2  
CTR Ciphertext 1: 69dda8455c7dd4254bf353b773304eec0ec7702330098ce7f7520d1cbbb20fc388d1b0adb5054dbd7370849dbf0b88d393f252e764f1f5f7ad97ef79d59ce29f5f51eeca32eabedd9afa9329  
  
CTR key: 36f18357be4dbd77f050515c73fcf9f2  
CTR Ciphertext 2: 770b80259ec33beb2561358a9f2dc617e46218c0a53cbeca695ae45faa8952aa0e311bde9d4e01726d3184c34451

## 4. Для кажого режима шифрования зашифровать и расшифровать произвольный текст длины 2,5 блока.