C# Sample code

```
namespace AES_Example
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {

                using (AesManaged myAes = new AesManaged())
                {
                    EncryptValue("Hello this is Cryptonite..",
Encoding.ASCII.GetBytes("bbC2H19lkVbQDfakxcrtNMQdd0FloLyw"),
Encoding.ASCII.GetBytes("gqLOHUioQ0QjhuvI"));
                }
                Console.ReadKey();
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: {0}", e.Message);
            }
        }
        static void EncryptValue(string plainText, byte[] Key, byte[] IV)
        {
            byte[] encrypted = EncryptStringToBytes_Aes(plainText, Key, IV);

            // Decrypt the bytes to a string.
            //string roundtrip = DecryptStringFromBytes_Aes(encrypted, Key, IV);

            //Display the original data and the decrypted data.
            Console.WriteLine("Encrypted:  {0}", Convert.ToBase64String(encrypted));
            //return (Convert.ToBase64String(encrypted));
            //Console.WriteLine("Original:   {0}", original);
            //Console.WriteLine("Round Trip: {0}", roundtrip);

        }
        static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, byte[]
IV)
        {
            // Check arguments.
            if (plainText == null || plainText.Length <= 0)
                throw new ArgumentNullException("plainText");
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");
            byte[] encrypted;
            // Create an AesManaged object
            // with the specified key and IV.
            using (AesManaged aesAlg = new AesManaged())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                // Create a decrytor to perform the stream transform.
                ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key,
aesAlg.IV);

                // Create the streams used for encryption.
```

```
                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt,
encryptor, CryptoStreamMode.Write))
                    {
                        using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                        {

                            //Write all data to the stream.
                            swEncrypt.Write(plainText);
                        }
                        encrypted = msEncrypt.ToArray();
                    }
                }
            }


            // Return the encrypted bytes from the memory stream.
            return encrypted;

        }

        static string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key, byte[]
IV)
        {
            // Check arguments.
            if (cipherText == null || cipherText.Length <= 0)
                throw new ArgumentNullException("cipherText");
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");

            // Declare the string used to hold
            // the decrypted text.
            string plaintext = null;

            // Create an AesManaged object
            // with the specified key and IV.
            using (AesManaged aesAlg = new AesManaged())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                // Create a decrytor to perform the stream transform.
                ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key,
aesAlg.IV);

                // Create the streams used for decryption.
                using (MemoryStream msDecrypt = new MemoryStream(cipherText))
                {
                    using (CryptoStream csDecrypt = new CryptoStream(msDecrypt,
decryptor, CryptoStreamMode.Read))
                    {
                        using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                        {

                            // Read the decrypted bytes from the decrypting stream
                            // and place them in a string.
                            plaintext = srDecrypt.ReadToEnd();
                        }
                    }
```

```
                }
            }
            return plaintext;
        }
    }
}
```

Android Sample code

```
public class CryptLib {

   /**
    * Encryption mode enumeration
    */
   private enum EncryptMode {
      ENCRYPT, DECRYPT;
   }

   // cipher to be used for encryption and decryption
   Cipher _cx;

   // encryption key and initialization vector
   byte[] _key, _iv;

   public CryptLib() throws NoSuchAlgorithmException, NoSuchPaddingException {
      // initialize the cipher with transformation AES/CBC/PKCS5Padding
      _cx = Cipher.getInstance("AES/CBC/PKCS5Padding");
      _key = new byte[32]; //256 bit key space
      _iv = new byte[16]; //128 bit IV
   }

   /**
    * Note: This function is no longer used.
    * This function generates md5 hash of the input string
    * @param inputString
    * @return md5 hash of the input string
```

```
 */
public static final String md5(final String inputString) {
   final String MD5 = "MD5";
   try {
      // Create MD5 Hash
      MessageDigest digest = java.security.MessageDigest
            .getInstance(MD5);
      digest.update(inputString.getBytes());
      byte messageDigest[] = digest.digest();

      // Create Hex String
      StringBuilder hexString = new StringBuilder();
      for (byte aMessageDigest : messageDigest) {
         String h = Integer.toHexString(0xFF & aMessageDigest);
         while (h.length() < 2)
            h = "0" + h;
         hexString.append(h);
      }
      return hexString.toString();

   } catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
   }
   return "";
}

/**
 *
```

```
* @param _inputText
*          Text to be encrypted or decrypted
* @param _encryptionKey
*          Encryption key to used for encryption / decryption
* @param _mode
*          specify the mode encryption / decryption
* @param _initVector
*         Initialization vector
* @return encrypted or decrypted string based on the mode
* @throws UnsupportedEncodingException
* @throws InvalidKeyException
* @throws InvalidAlgorithmParameterException
* @throws IllegalBlockSizeException
* @throws BadPaddingException
*/
private String encryptDecrypt(String _inputText, String _encryptionKey,
                        EncryptMode _mode, String _initVector) throws UnsupportedEncodingException,
      InvalidKeyException, InvalidAlgorithmParameterException,
      IllegalBlockSizeException, BadPaddingException {
   String _out = "";// output string
   //_encryptionKey = md5(_encryptionKey);
   //System.out.println("key="+_encryptionKey);

   int len = _encryptionKey.getBytes("UTF-8").length; // length of the key    provided

   if (_encryptionKey.getBytes("UTF-8").length > _key.length)
      len = _key.length;
```

```
int ivlen = _initVector.getBytes("UTF-8").length;

if(_initVector.getBytes("UTF-8").length > _iv.length)
   ivlen = _iv.length;

System.arraycopy(_encryptionKey.getBytes("UTF-8"), 0, _key, 0, len);
System.arraycopy(_initVector.getBytes("UTF-8"), 0, _iv, 0, ivlen);
//KeyGenerator _keyGen = KeyGenerator.getInstance("AES");
//_keyGen.init(128);

SecretKeySpec keySpec = new SecretKeySpec(_key, "AES"); // Create a new SecretKeySpec
// for the
// specified key
// data and
// algorithm
// name.

IvParameterSpec ivSpec = new IvParameterSpec(_iv); // Create a new
// IvParameterSpec
// instance with the
// bytes from the
// specified buffer
// iv used as
// initialization
// vector.

// encryption
if (_mode.equals(EncryptMode.ENCRYPT)) {
```

```
    // Potentially insecure random numbers on Android 4.3 and older.
    // Read
    // https://android-developers.blogspot.com/2013/08/some-securerandom-thoughts.html
    // for more info.
    _cx.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);// Initialize this cipher instance
    byte[] results = _cx.doFinal(_inputText.getBytes("UTF-8")); // Finish
    // multi-part
    // transformation
    // (encryption)
    _out = Base64.encodeToString(results, Base64.DEFAULT); // ciphertext
    // output
  }

  // decryption
  if (_mode.equals(EncryptMode.DECRYPT)) {
    _cx.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);// Initialize this ipher instance

    byte[] decodedValue = Base64.decode(_inputText.getBytes(),
        Base64.DEFAULT);
    byte[] decryptedVal = _cx.doFinal(decodedValue); // Finish
    // multi-part
    // transformation
    // (decryption)
    _out = new String(decryptedVal);
  }
  System.out.println(_out);
  return _out; // return encrypted/decrypted string
}
```

```
/***
 * This function computes the SHA256 hash of input string
 * @param text input text whose SHA256 hash has to be computed
 * @param length length of the text to be returned
 * @return returns SHA256 hash of input text
 * @throws NoSuchAlgorithmException
 * @throws UnsupportedEncodingException
 */
public static String SHA256 (String text, int length) throws NoSuchAlgorithmException,
UnsupportedEncodingException {

    String resultStr;
    MessageDigest md = MessageDigest.getInstance("SHA-256");

    md.update(text.getBytes("UTF-8"));
    byte[] digest = md.digest();

    StringBuffer result = new StringBuffer();
    for (byte b : digest) {
       result.append(String.format("%02x", b)); //convert to hex
    }
    //return result.toString();

    if(length > result.toString().length())
    {
       resultStr = result.toString();
    }
    else
```

```
    {
        resultStr = result.toString().substring(0, length);
    }

    return resultStr;

}

/***
 * This function encrypts the plain text to cipher text using the key
 * provided. You'll have to use the same key for decryption
 *
 * @param _plainText
 *          Plain text to be encrypted
 * @param _key
 *          Encryption Key. You'll have to use the same key for decryption
 * @param _iv
 *       initialization Vector
 * @return returns encrypted (cipher) text
 * @throws InvalidKeyException
 * @throws UnsupportedEncodingException
 * @throws InvalidAlgorithmParameterException
 * @throws IllegalBlockSizeException
 * @throws BadPaddingException
 */

public String encrypt(String _plainText, String _key, String _iv)
        throws InvalidKeyException, UnsupportedEncodingException,
```

```
        InvalidAlgorithmParameterException, IllegalBlockSizeException,
        BadPaddingException {
    return encryptDecrypt(_plainText, _key, EncryptMode.ENCRYPT, _iv);
}

/***
 * This funtion decrypts the encrypted text to plain text using the key
 * provided. You'll have to use the same key which you used during
 * encryprtion
 *
 * @param _encryptedText
 *          Encrypted/Cipher text to be decrypted
 * @param _key
 *          Encryption key which you used during encryption
 * @param _iv
 *        initialization Vector
 * @return encrypted value
 * @throws InvalidKeyException
 * @throws UnsupportedEncodingException
 * @throws InvalidAlgorithmParameterException
 * @throws IllegalBlockSizeException
 * @throws BadPaddingException
 */
public String decrypt(String _encryptedText, String _key, String _iv)
        throws InvalidKeyException, UnsupportedEncodingException,
        InvalidAlgorithmParameterException, IllegalBlockSizeException,
        BadPaddingException {
    return encryptDecrypt(_encryptedText, _key, EncryptMode.DECRYPT, _iv);
```

```
    }


    /**
     * this function generates random string for given length
     * @param length
     *            Desired length
     * * @return
     */
    public static String generateRandomIV(int length)
    {
        SecureRandom ranGen = new SecureRandom();
        byte[] aesKey = new byte[16];
        ranGen.nextBytes(aesKey);
        StringBuffer result = new StringBuffer();
        for (byte b : aesKey) {
            result.append(String.format("%02x", b)); //convert to hex
        }
        if(length> result.toString().length())
        {
            return result.toString();
        }
        else
        {
            return result.toString().substring(0, length);
        }
    }
}
```

Input

```
CryptLib _crypt = new CryptLib();

String output= "";

String plainText = "Hello this is Cryptonite..";

String key = "bbC2H19lkVbQDfakxcrtNMQdd0FloLyw";

String iv = "gqLOHUioQ0QjhuvI";

output = _crypt.encrypt(plainText, key, iv);

System.out.println("encrypted text=" + output);

output = _crypt.decrypt(output, key,iv);

System.out.println("decrypted text=" + output);
```

iOS (Swift) Sample code

```
extension String {

    func aesEncrypt(key: String, iv: String) throws -> String{

        let data = self.dataUsingEncoding(NSUTF8StringEncoding)

        let enc = try AES(key: key, iv: iv, blockMode: .CBC, padding: PKCS7()).encrypt((data?.arrayOfBytes())!)

        let encData = NSData(bytes: enc, length: Int(enc.count))

        let base64String: String =
encData.base64EncodedStringWithOptions(NSDataBase64EncodingOptions(rawValue: 0));

        let result = String(base64String)

        return result

    }



    func aesDecrypt(key: String, iv: String) throws -> String {

        let data = NSData(base64EncodedString: self, options: NSDataBase64DecodingOptions(rawValue: 0))

        let dec = try AES(key: key, iv: iv, blockMode: .CBC, padding: PKCS7()).decrypt((data?.arrayOfBytes())!)

        let decData = NSData(bytes: dec, length: Int(dec.count))

        let result = NSString(data: decData, encoding: NSUTF8StringEncoding)

        return String(result!)

    }

}
```

Input

```
let key = "bbC2H19lkVbQDfakxcrtNMQdd0FloLyw" // length == 32

let iv = "gqLOHUioQ0QjhuvI" // length == 16

let s = "Hello this is Cryptonite.."

let enc = try! s.aesEncrypt(key, iv: iv)
```