

Assignment 1: Block Ciphers

“Course 1-02-328: Communication and E-Commerce Security ”

Due 13 April 2010; **Extended to 18 April 2010**

1 Simplified DES

In this project, you will implement a simplified version of the DES block cipher algorithm. Naturally enough, it is called SDES, and it is designed to have the features of the DES algorithm but scaled down so it is more tractable to understand. (Note however, that SDES is in no way secure and should not be used for serious cryptographic applications.)

The SDES description discussed in the Targil session can be downloaded from the course web page. It gives the detailed specifications of SDES. SDES encryption takes a 10 bit raw key (from which two 8 bit keys are generated as described in the handout) and encrypts an 8 bit plaintext to produce an 8 bit ciphertext.

Implement the SDES algorithm in a C# class called Sdes. The encryption and decryption methods should match the interface below:

```
public static byte[] Encrypt(byte[] rawkey, byte[] plaintext)
public static byte[] Decrypt(byte[] rawkey, byte[] ciphertext)
```

Here, rather than compactly representing the SDES plaintext and ciphertext using byte-sized (8-bit) variables, the data is represented using byte arrays of length 8. Similarly the 10 bit keys are represented as arrays of length 10. Although this design is extremely inefficient (it uses 8 times more space), it makes the algorithm easier to implement and experiment with. For example, one might declare a 10-bit raw key in a test program like this:

```
byte[] key1 = {1, 1, 1, 0, 0, 0, 1, 1, 1, 0};
```

To verify that your implementation of SDES is correct, try the following test cases:

Raw Key	Plaintext	Ciphertext
000000000	10101010	00010001
1110001110	10101010	11001010
1110001110	01010101	01110000
1111111111	10101010	00000100

Use your implementation to complete the following table:

Raw Key	Plaintext	Ciphertext
000000000	00000000	?
1111111111	11111111	?
0000011111	00000000	?
0000011111	11111111	?
1000101110	?	00011100
1000101110	?	11000010
0010011111	?	10011101
0010011111	?	10010000

1.1 SDES with Cipher Block Chaining

The SDES algorithm you implemented in the previous section works on a single block of 8 bits and produces an 8 bit output. To encrypt and decrypt messages longer than 8 bits, the algorithm must break the message up into 8 bit blocks and perform encryption and decryption operations on them. As discussed in class, one option for managing encryption of messages longer than the cipher's block size is to use Cipher Block Chaining (CBC) mode. In CBC mode, each block of plain text is XOR'd with the ciphertext from the previous block. The first block is XOR'd with a random initialization vector (IV).

Augment your SDES algorithm to use CBC mode for messages and place it in a class called `SdesCbc`. The encryption and decryption methods should therefore have the following signatures:

```
public static byte[] CBCEncrypt(byte[] rawkey, byte[] iv, byte[] plaintext)
public static byte[] CBCDecrypt(byte[] rawkey, byte[] iv, byte[] ciphertext)
```

Note that here the `iv` array is always 8 bits, but the `plaintext` and `ciphertext` arrays may be of any length (you may assume it's always a multiple of 8 for convenience).

Hint: You can use the `Encrypt` and `Decrypt` functions you wrote for the previous sections here to help.

To verify that your implementation of SDES with CBC is correct, try the following test cases:

Raw Key	IV	Plaintext	Ciphertext
0000000000	01100001	10101010	10100010
1110001110	00111100	10101010	10101010
1110001110	01101110	01010101	00111110
1111111111	11110011	10101010	10010001
0000000000	00010101	00001100 01101111	00100100 11100100
0000000010	01000001	00111010 01110101	01000011 01011011
0011001110	01101001	01101001 10001010	11000001 01100101
0111000101	01101010	10110011 11001001	01110100 11101101
0111000101	01110010	10110011 11001001	01111101 00110110

Use your implementation to complete the following table:

Raw Key	IV	Plaintext	Ciphertext
0000000000	11001011	00000000	?
1111111111	11110111	11111111	?
0000011111	10110010	00000000	?
0000011111	11011110	11111111	?
0100111001	00110010	11010111 01110000	?
0010010110	11001101	01111101 00010110	?
1000001011	10100111	11111000 11101011	?
1000101110	01000000	?	00011100
1000101110	11100010	?	11000010
0010011111	10110101	?	10011101
0010011111	01011011	?	10010000
0011010001	11111110	?	10100001 10111001
0000100001	10011111	?	01111001 01001011
1011011001	00111000	?	00000111 00011010

1.2 Testing Program

To enable easier testing of your encryption and decryption classes, create a GUI interface to enable interactive checking of encryption and decryption using SDES and SDES with CBC. A sample GUI is shown in Figure 1.

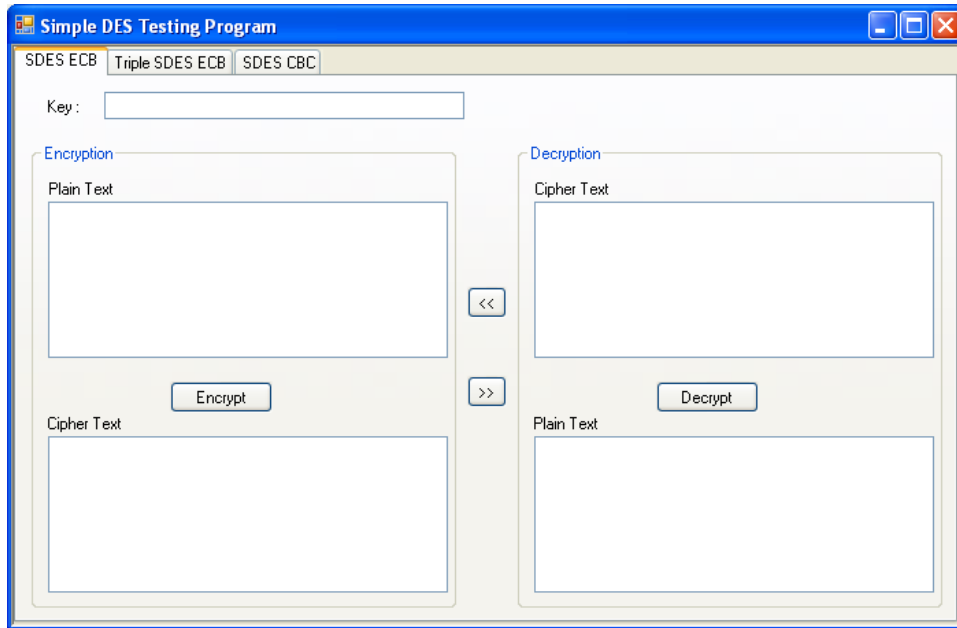


Figure 1: Sample GUI

You do not need to copy the GUI design directly, but you should implement something functionally similar. I have placed a copy of the code used for the sample GUI on the course web page.

The text boxes should enable entering text longer than 8 bits. The long length text should be broken up into blocks of 8 bits and fed to the SDES classes for encryption or decryption. **The text boxes should accept character input as well as numeric input.** You may choose any binary encoding scheme for the conversion (ASCII, UTF8, Unicode, Base64)¹ For encryption, text or binary input which does not break evenly into 8 bit blocks should be padded with 0s. For decryption, if the input does not break up evenly into 8 bit blocks, raise an error window.

The program should choose ECB or CBC mode automatically using the following mechanism:

1. If the IV textbox is blank, it should be encrypted or decrypted using Electronic Code Book (ECB) mode.
2. If the IV textbox is not blank, it should be encrypted or decrypted using the Cipher Block Chaining (CBC) mode.
 - Note, if the IV textbox contains an invalid IV, an error window should pop up.

You may alternatively implement a checkbox or some other mechanism which lets the user choose which mode to use.

1.3 TripleSDES

The DES algorithm uses keys of length 56 bits, which, when DES was originally designed, was thought to be secure enough to meet most needs. However, due to Moore's law, the increase in computing power makes it more tractable to brute-force crack a 56-bit key. Thus, an alternative version of DES using longer keys was desirable. The result, known as Triple DES uses two 56-bit raw keys k_1 and k_2 and is implemented by composing DES with itself three times in the following way:

¹I can provide a sample character encoding class for any student groups interested.

$$E_{3DES(p)} = E_{DES}(k_1, D_{DES}(k_2, E_{DES}(k_1, p)))$$

Here, p is the plaintext to encrypt, E_{DES} is the usual DES encryption algorithm and D_{DES} is the DES decryption algorithm. This strategy doubles the number of bits in the key, at the expense of performing three times as many calculations. (You might wonder why the algorithm is not just $E_{DES}(k_1, E_{DES}(k_2, p))$. This approach was shown to offer only the security of a 57-bit key rather than 112 bits as you might expect.)

The TripleDES decryption algorithm is just the reverse:

$$D_{3DES}(c) = D_{DES}(k_1, E_{DES}(k_2, D_{DES}(k_1, c)))$$

For this part of the project, implement a class called TripleSDES that provides the following methods and calculates the TripleSDES encryption.

```
public static byte[] Encrypt( byte[] rawkey1, byte[] rawkey2, byte[] plaintext )
public static byte[] Decrypt( byte[] rawkey1, byte[] rawkey2, byte[] ciphertext )
```

Use your implementation to complete the following table:

Raw Key	Raw Key 2	Plaintext	Ciphertext
0000000000	0000000000	00000000	?
1000101110	0110101110	11010111	?
1000101110	0110101110	10101010	?
1111111111	1111111111	10101010	?
1000101110	0110101110	?	11100110
1011101111	0110101110	?	01010000
0000000000	0000000000	?	10000000
1111111111	1111111111	?	10010010

2 Grading

The project will be graded as follows:

- 35 points for the SDES implementation with ECB.
- 15 points for the SDES implementation with CBC.
- 15 points for the Triple SDES implementation (with ECB)

Bugs or errors in the encryption or decryption of either mode will lead to points being taken off.²

3 What to turn in

Each group should turn in the following items (one submission per group). Electronic submissions should be uploaded to the Telem website or transferred to me by USB disk on key.

1. A project report file containing the names of the students in your group. In addition to the items specified below, include a one- or two-sentence description of each group members contributions to the project and an estimate of the number of hours each member contributed.
2. Your group will receive a single grade for the project. The information about individual contributions will be used to help gauge the difficulty and effectiveness of the assignment.

²Errors in the ECB code which lead to bugs in the CBC encryption/decryption will be dealt with on a case-by-case-basis.

3. A file of the test cases described in the section, the bits making up the keys of the SDES and SDES CBC encrypted messages, and the messages themselves.
4. An electronic submission of your source code. This should be single, compressed directory in .tar.gz or .zip format. The directory should contain a file called README that describes the contents of the directory and any special instructions needed to run your programs. The classes Sdes and SdesCBC should be implemented according to the interface described above **using C#**.
 - Students who choose to use a programming language other than C# must send me a precompiled executable of the program along with compilation instructions.

Collaboration between groups and copying from internet, any online resource, source code sites, or online cracking tools is prohibited. Automated code checking tools may be used to check for copying or sharing of source code.