

Simplified DES (SDES)

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
0000000000	10101010	00010001
1110001110	10101010	11001010
1110001110	01010101	01110000
1111111111	10101010	00000100

Multi-byte input in ECB mode Your `Encrypt` function should accept 8 plaintext bits (1 byte) at a time and return 8 bits of ciphertext. Similarly, the `Decrypt` function should accept 8 ciphertext bits and return 8 plaintext bits. In other words, `Encrypt` and `Decrypt` work in Electronic Code Book (ECB) mode. Your testing tool (described below) must enable the user to encrypt or decrypt more than one byte of plaintext or ciphertext at a time using ECB. In that case, each byte must be encrypted (decrypted) one after the other using the the above functions.

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. Another 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 provided it is a multiple of 8. You should throw an exception if it's not a multiple of 8.

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

1.2 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:

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

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 )
```

You may test your implementation using the following table:

Raw Key 1	Raw Key 2	Plaintext	Ciphertext
0000000000	0000000000	00000000	11110000
1000101110	0110101110	11010111	10111001
1000101110	0110101110	10101010	11100100
1111111111	1111111111	10101010	00000100
1000101110	0110101110	11111101	11100110
1011101111	0110101110	01001111	01010000
0000000000	0000000000	01010010	10000000
1111111111	1111111111	00100101	10010010

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, SDES with CBC, and TripleSDES. A sample GUI is shown in Figure 1. Its designer code will be provided. You do not need to copy the GUI design precisely, but you must implement something functionally similar.

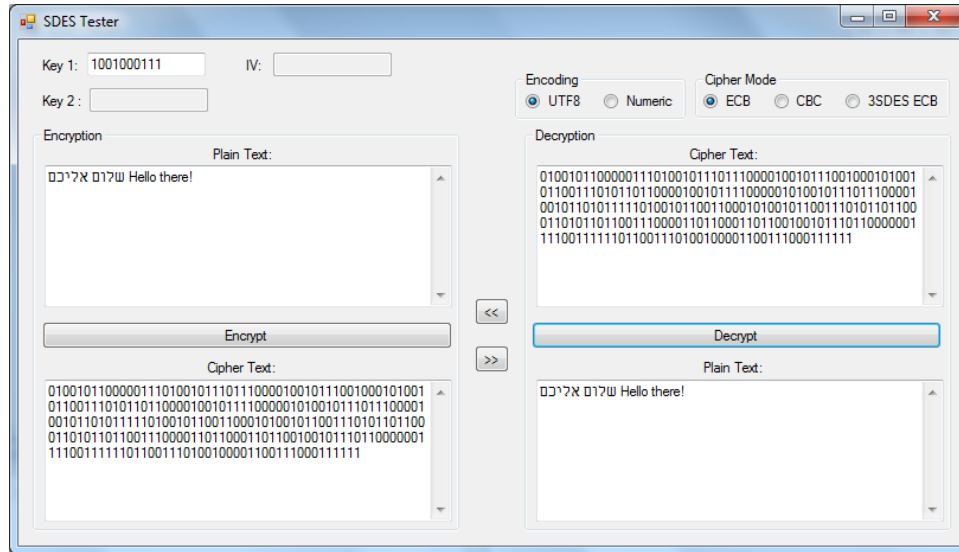


Figure 1: Sample GUI

The text boxes must enable entering text or bit strings 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 radio buttons in the top right hand corner of the tool let the user define whether the input is text or numeric.** To convert from character input to binary, use the **UTF8** binary encoding scheme.

For numeric encryption (or any decryption) if the input plaintext/ciphertext does not break up evenly into 8 bit blocks, raise an error window. UTF8 will always encode text in whole bytes, so encoded character input will not cause any errors. To test your character input, you may use the following test table:

Plaintext	Key1	Key2	IV	Ciphertext
Hello world!	1100111110			000100101110000111011111110111111000000011001001001100000000011100110111111100001110110000
Goodbye	1111000011			1100011001000000010000001110010010010110011011011010101111
Goodbye	1111000011		10111100	00000000010000001111010001001001100100011011110110000100
שלום	1111000011		10111100	00000001001010011101010101110100000110011100010010010011000000
Kinneret כנרת	1111000011	1001111000		010101111011101000001010000010101010101000111101101010010010000001100111100010110111100010110100111110001110100111111000110110100

3 Grading

The project will be graded as follows:

- 40 points for the SDES implementation with ECB.
- 40 points for the SDES implementation with CBC.
- 10 points for the Triple SDES implementation (with ECB).
- 10 points general robustness and handling of character inputs.

Bugs or errors in the encryption or decryption of either mode will lead to points being taken off.¹

4 What to turn in

Each group should turn in the following items:

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.
 - If this information is skipped, I will deduct 3 points.
2. An electronic submission of your source code. This should be single, compressed directory in .tar.gz, .zip, or .rar 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, SdesCbc, and TripleSdes 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.
3. A compiled version of your complete program (EXE). You may send this as part of your source code submission provided that you send it as a RAR file (GMail does not allow EXE files in .zip archives).

Send your submission to the course email ([ise328@gmail](mailto:ise328@gmail.com)) or via the Telem website.

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.

¹Errors in the ECB code which lead to bugs in the CBC encryption/decryption will be dealt with on a case-by-case-basis.