

“Course 1-02-328: Communication and E-Commerce Security”
Recitation 4: DES in C#

Instructor: Michael J. May

March 23, 2010

In this recitation you will put together a simple application which will perform DES encryption and decryption using the C# cryptographic libraries. The goal of this recitation is to understand how the C# cryptographic libraries work and try some experiments with CBC and initialization vectors.

1 Simple DES demonstration

We will begin by going over the simple command line demonstration program called DESTests. The source code and project are on the web page for the course as a ZIP file.

2 What to do: DES Encryptor/Decryptor

Using the code from the command line program as an example, your job is to develop a Windows Form based application which will do encryption and decryption of strings using buttons. This is meant as an exercise in using the cryptographic classes so you will get a better idea of how they work.

Create a Form which looks like the one below. The code framework and designer code is included on the course web page as a ZIP file called DESForm.

The top two buttons perform a conversion from String to Byte[] since the DES cryptographic libraries accept keys and initialization vectors as byte arrays, not strings. The byte array must be exactly 64 bytes - the eighth bit for each byte is discarded. The cipher text is shown as a byte array to save you a bit of coding. You can easily covert it back to Unicode or ASCII strings if you'd like.

The screenshot shows a Windows Form titled "DES Testing Window" with a blue title bar and standard window controls. The form has a light beige background and contains the following elements:

- At the top, two rows of input fields. The first row has a label "Key:" followed by a text box containing "KeyTexts", a button with a right-pointing arrow ">", and a text box labeled "as Bytes:" containing the value "75 101 121 84 101 120 116 115".
- The second row has a label "IV:" followed by a text box containing "InitVect", a button with a right-pointing arrow ">", and a text box labeled "as Bytes:" containing the value "73 110 105 116 86 101 99 116".
- Below these are two large text areas. The left one is titled "Plain Text" and contains the text "This is a test plain text".
- The right one is titled "Cipher Text" and contains the following text:

```
172 209 227 132 173 85 49 8 73 188 20
145 50 102 241 101 185 150 146 48 55
132 175 116 131 162 168 90 8 198 201
185
```
- Between the two text areas are two buttons: "Encrypt" (top) and "Decrypt" (bottom).

3 What to do: Experiments

Using your encryption and decryption program, perform the following tests:

3.1 Experiment 1: IV

1. Set the key to be: “torabora”. Set the IV to be: “untilone”. Convert the key and IV to bytes.
2. Encrypt the string: “This is the song that doesn’t end, yes it goes on and on my friend.”
3. What do you get for cipher text?
4. Change the IV to be: “untiltwo”. Convert the IV to bytes. Decrypt the cipher text. What happened? What part of the message got corrupted?
5. Change the IV to be: “qweerkie”. Convert the IV to bytes. Decrypt the cipher text. What happened? What part of the message got corrupted?
6. Return the IV to “untilone”. Convert the IV to bytes. Change the last number of the IV to 100 instead of 101. Decrypt the cipher text. What happened? What part of the message got corrupted?

Based on the above experiment, can you tell how many characters in the string are included in a single block?

3.2 Experiment 2: Keys

1. Set the key to “antilest”. Set the IV to be: “typically”. Convert the key and IV to bytes.
2. Encrypt the string: “Come on along or go alone, he’s come to take his children home.”
3. What do you get for the cipher text?
4. Change the key to be: “victory”. Convert the key to bytes. Decrypt the message. What happened? Why?
5. Change the key to be: “antilesu”. Convert the key to bytes. Decrypt the message. What happened? Why? (Hint: Look at the last byte in binary)
6. Change the key to be: “antilesv”. Convert the key to bytes. Decrypt the message. What happened? Why? (Hint: Look at the last byte in binary)
7. Can you change any other characters or bytes in the key and still get the correct decryption?

Based on the above experiment, can you tell how `C#` converts from a 64 bit string to a 56 bit key?

3.3 Experiment 3: Plain text

1. Set the key to be “nopevery”. Set the IV to be: “interfan”. Convert the key and IV to bytes.
2. Encrypt the string: “Ripple in still water. There is no pebble tossed nor wind did blow”.
3. What did you get for cipher text?
4. Change the last letter of the cipher text to be “v” (*i.e.*, it should read “blov”). Encrypt the string. How many bytes changed in the cipher text?
5. Change the last letter back to “w”. Change the first character to be “S” (*i.e.*, it should be “Sipple”). Encrypt the string. How many bytes changed in the cipher text?

6. Change the first letter back to “R”. Change the first “T” to be “U” (*i.e.*, the fifth word should be “Uhere”). Encrypt the string. How many bytes changed in the cipher text?

Why does this happen? How does this relate to how CBC works?

3.4 Experiment 4: Cipher text

1. Set the key to be “beckolny”. Set the IV to be: “frumpley”. Convert the key and IV to be bytes.
2. Encrypt the string: “You enter a darkened room and sittting there in the gloom is Dracula.”.
3. What did you get for the cipher text?
4. Change the last byte of the cipher text to be 196 (instead of 197). Decrypt the cipher text. What happened?
5. Change the last byte back to 197. Change the first byte to be 163 (instead of 162). Decrypt the cipher text. What happened?
6. Change the first byte to be 161. Decrypt the cipher text. What happened?

How sensitive is C#'s DES decryptor to changes or errors in the cipher text? Can you figure out the block size for the message based on this?