Encryption in C#

Encryption means to change data to a format that is unreadable unless you have the correct key to decipher it.

There have been lots of ways to encrypt information over the years. From simple Caesar ciphers to SSL, SSH and AES, the encryption used most on the web right now.

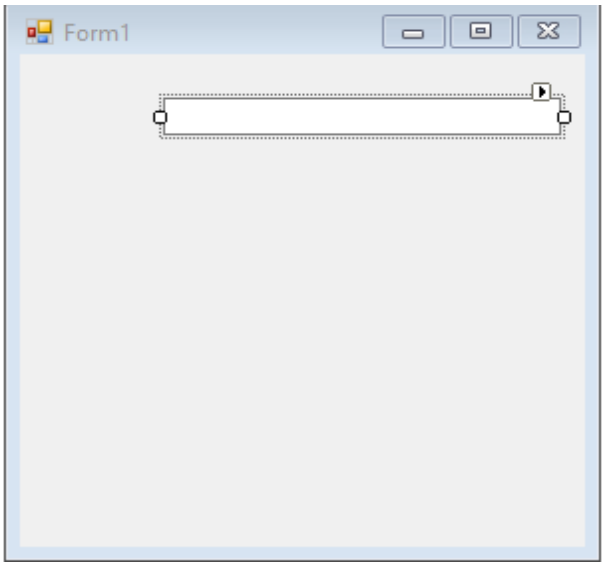
Caesar Cipher

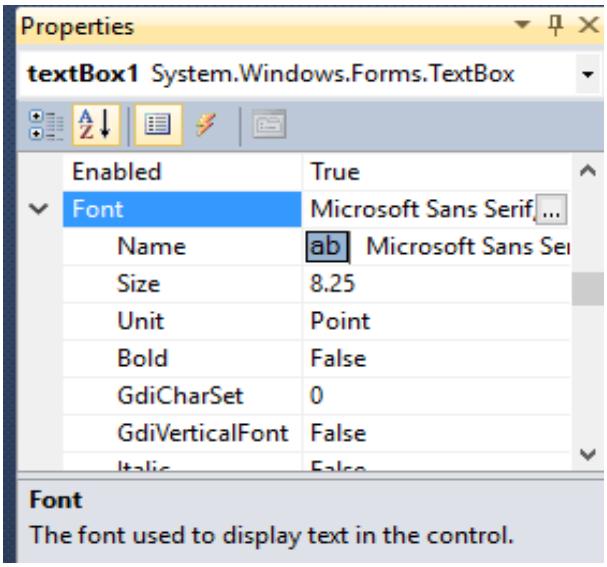
The Caesar cipher is a substitution cipher, named after Julius Caesar who apparently used it to protect military messages.
Basics:
Each letter is translated into the letter *a fixed number of positions* after it in the alphabet.
The fixed number of positions is a key both for encryption and decryption.

For example, a shift of 3, like Caesar used means shifting 3 places, A becomes D, B becomes E, C becomes F all the way through until X becomes A, Y becomes B and Z becomes C.
So, "Hello" becomes KHOOR, the problem with this (and a lot of codes) becomes that it's easy to spot patterns. E.g. with a bit of brain power, if you saw KHOOR at the start of a number of posts you could guess that it meant hello, and possibly decipher others.

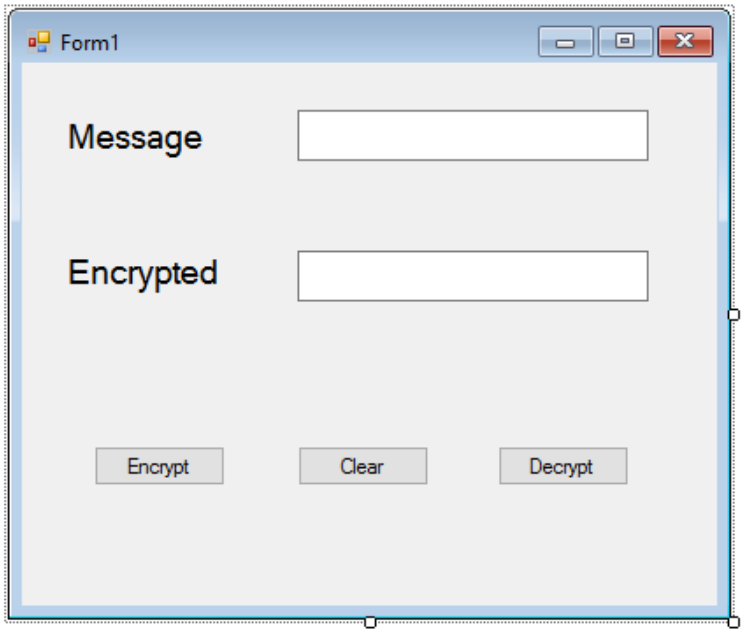
Programming a shift Cipher

Create a new Windows Form program and drag a text box tool onto the form that opens. Then, in properties, find the name and call it "plainText" and the font -> size box and up it to around 14. This will be the entry for our plain text.





Copy and paste this box to create another, rename it "encryptedText" this is, surprisingly, where we'll see our encrypted message. Add a couple labels to explain your textboxes and a couple buttons we can make encrypt and decrypt, something like below.



Nice and simple so far, now we want to actually make our buttons do something, let's start with encrypt, double click it.

```
private void button1_Click(object sender, EventArgs e)
{
    String message = plainText.Text;
    Char letter;
    String encrypted="";

    foreach (Char c in message)
    {
        letter=c;
        letter=(char)(letter+3);
        encrypted += letter;
    }
    encryptedText.Text = encrypted;
}
```

This brings up the auto created code for an event handler for when the button is pressed. Fill in the code below;

This takes advantage of the fact that, behind the scenes, the computer stores letters as numbers (see ASCII for instance). What we're doing here is getting the message, using a foreach loop to go through every Char or letter in the message and shift by 3 then save back into a new string, the (char) bit after the addition makes sure it counts it as a Char not as a number. This won't work quite the same as the example

above, in this case it uses the full symbol set (so stuff like [,{,>%) so for instance XYZ becomes [\].

Translating back

Give it a go, it's almost identical, double click the second button and see if you can write your own method of translating back. If you add 3 to encrypt it, think about what you need to do to shift back.

Test it out.

Binary Shift

Another way of encrypting data is using something called a binary shift, we translate each character into Binary, for instance H becomes 01000 E becomes 00101 L becomes 01100 and O becomes 01111.

Then we might add another "codeword" or character translated into binary to the letter.

For instance, you might use the letter S- 10011 in our binary system, as your shift, so you add S in binary to each letter in Binary.

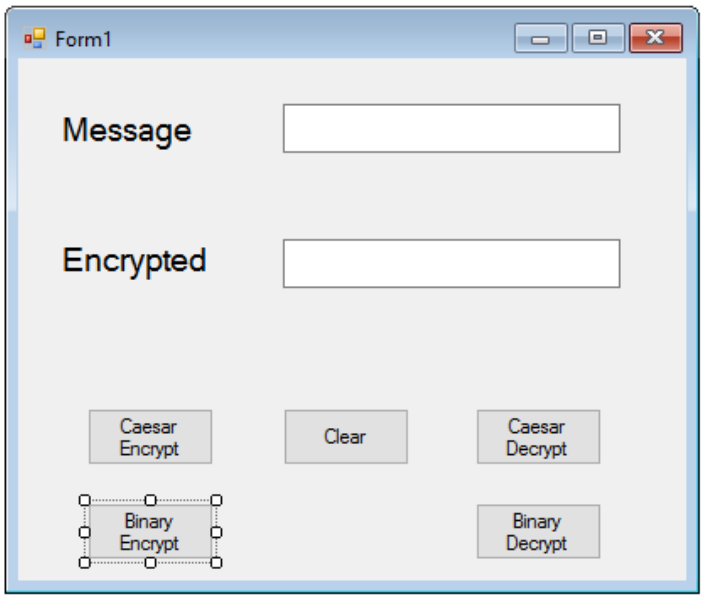
This means H and S added together would be 01000 and 10011, the trick here is that we only add according to these rules;

0 + 0 =1 0+1 = 1 and 1 + 1 =0

We call this exclusive OR or XOR and it's something used often by computers, in this system encrypting H would be 11111.

Modifying the C# program for binary shift encryption

So, we're going to add another option to the program, as above, add two new buttons to the form as to the right.



Double click your button for binary encryption, the below is the code that performs the encryption, it's very similar to the previous. You set three variables, the message, the individual letter store and an empty string to hold the encrypted text.

Then we loop round every Char in the message, the ^ symbol perform XOR we talked about earlier so the binary shift. We then save that back into a string. Because the character set we're using and their values are a lot higher than the simple example when we shift them the result might be a character that isn't a symbol that can be displayed. We use S to shift with.

```
private void button5_Click(object sender, EventArgs e)
{
    String message = plainText.Text;
    Char letter;
    String encrypted = "";

    foreach (Char c in message)
    {
        letter = c;
        letter = (char)(letter ^ 'S');
        encrypted += letter;
    }
    encryptedText.Text = encrypted;
}
```

Translating back

Give it a go, it's almost identical, double click the second button and see if you can write your own method of translating back. This encryption can be undone just by repeating it, so to undo the binary shift with S, we binary shift with S again.

Try it.

Putting them together

If you combine the Caesar and the Binary shift together you get a stronger form of encryption, in fact as you'll see in another lesson on encryption that the internet is secured by something very similar, lots of linked simpler encryption types.

To put them together we're going to do two things, put the code into methods and then link those methods back to buttons.

This makes our code reusable, easier to read and easier to extend, it's good practice for the future.

Moving into methods

This is fairly simple, we just take the code from each button and make it into a method, here I've rewritten the Caesar shift decrypt button into a method.

A method had three parts, "void" is the return type, this is what it sends back when you run it, here it's void, nothing, it doesn't send anything back. If it was int, it would send a number back for instance.

"caesarDecrypt" is the name, simple as that, it's what we call it with, "()" is the parameters, what we need to tell it, it's empty, we don't give it anything.

```
private void button3_Click(object sender, EventArgs e)
{
    caesarDecrypt();
}

void caesarDecrypt()
{
    String encrypted = encryptedText.Text;
    Char letter;
    String message = "";

    foreach (Char c in encrypted)
    {
        letter = c;
        letter = (char)(letter - 3);
        message += letter;
    }
    plainText.Text = message;
}
```

So, we move that code into the method and then call the method by using it's name.

Calling the methods where needed

That's simple, we call the method in the button, e.g. "caesarDecrypt" above, you'll need to do this for every encrypt and decrypt we've got.

Once you've done that, you can do the below, double clicking the new button Caesar and Binary encrypt and typing in the following;

```
private void button6_Click(object sender, EventArgs e)
{
    caesarEncrypt();
    plainText.Text = encryptedText.Text;
    binaryEncrypt();
}

private void button7_Click(object sender, EventArgs e)
{
    binaryDecrypt();
    encryptedText.Text = plainText.Text;
    caesarDecrypt();
}
```

The top encrypts it, the bottom decrypts.

The reason for the second line between the methods is that the encryption / decryption code pulls from one box but outputs to another, so we need to copy the output back into the input before doing the second encryption. Give it a try.

Conclusion

This was a relatively simple introduction to encrypting something but it's fairly similar to how the more complex systems work. Another tutorial will be up soon on how AES works. By layering the number of encryptions it gets harder and harder to work out, you need more and more information to decrypt it, even knowing the order, you've got to know keys, shifts, all sorts.