

Forum:[Hack Forums](#)

Thread: [ROT13 encipher/decipher method](#)

gigajew - August 2, 2017, 9:13 pm

After working on the inclusive or, this caught my interest and so here you go. It works by rotating forward or backward 13. So if you had 13 and rotated backward, it'd become 0. If you had 255 and rotated forwards, you'd have 13.

Instructions for dummies: Rot13.Encipher(byte[]); and Rot13.Decipher(byte[]);

Phizo - August 2, 2017, 9:32 pm

You don't need to do if(r > 255) return r % 255 else return r. That is the same as r % 255. Same with the other condition below. Also, it is exclusive. It is in the interval [0, 255), meaning when r = 255, r % 255 = 0. Are you sure that you don't want r % 256?

gigajew - August 2, 2017, 9:33 pm

(Aug 2, 2017 09:32 PM)Phizo Wrote: [\[CITE\]](#) You don't need to do if(r > 255) return r % 255 else return r. That is the same as r % 255. Same with the other condition below. Also, it is exclusive. It is in the interval [0, 255), meaning when r = 255, r % 255 = 0. Are you sure that you don't want r % 256?

Thanks for the input I will review it tomorrow

With the modifications specified above. Thank you, Phizo

D3RP - August 3, 2017, 10:00 pm

This is interesting. Makes for a good encryption if you want to rotate ahead part of a byte array and rotate backwards other parts. Then you can represent this forwards / backwards movement using a single bit. 1 forwards, 0 backwards.... Really simple key to encode.

0xDEAD10CC - August 3, 2017, 10:06 pm

Why are your shift functions public members of the class when they have no relevance to the caller in terms of Rot13 which is dealt with by the 2 other functions? And why would you clone and unbox the array when you can just create an array of the same length? This is terribly inefficient for such a simple algorithm... However it's also inaccurate since 255 is a valid byte.

```
Code private static byte[] Rot13Encode(byte[] bytes)
```

```
{
    byte[] encoded = new byte[bytes.Length];
    for (int i = 0; i < bytes.Length; i++)
    {
        encoded[i] = (byte)((bytes[i] + 13) % 256);
    }
    return encoded;
}
```

```
private static byte[] Rot13Decode(byte[] bytes)
```

```
{
```

```

byte[] decoded = new byte[bytes.Length];
for (int i = 0; i < bytes.Length; i++)
{
    decoded[i] = (byte)(bytes[i] >= 13
        ? bytes[i] - 13
        : 243 + bytes[i]);
}
return decoded;
}

```

gigajew - August 4, 2017, 4:48 am

(Aug 3, 2017 10:06 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Why are your shift functions public members of the class when they have no relevance to the caller in terms of Rot13 which is dealt with by the 2 other functions? And why would you clone and unbox the array when you can just create an array of the same length? This is terribly inefficient for such a simple algorithm... However it's also inaccurate since 255 is a valid byte.

```

Code private static byte[] Rot13Encode(byte[] bytes)
{
    byte[] encoded = new byte[bytes.Length];
    for (int i = 0; i < bytes.Length; i++)
    {
        encoded[i] = (byte)((bytes[i] + 13) % 256);
    }
    return encoded;
}

```

```

private static byte[] Rot13Decode(byte[] bytes)
{
    byte[] decoded = new byte[bytes.Length];
    for (int i = 0; i < bytes.Length; i++)
    {
        decoded[i] = (byte)(bytes[i] >= 13
            ? bytes[i] - 13
            : 243 + bytes[i]);
    }
    return decoded;
}

```

I know 255 is a valid byte. Did you even try running the code and verifying the output? Hence the separate functions. Debugging purposes. Thanks for the input about the byte array I will keep it in mind.

Phizo - August 4, 2017, 5:51 am

(Aug 4, 2017 04:48 AM)gigajew Wrote: [\[CITE\]](#) I know 255 is a valid byte. Did you even try running the code and verifying the output? Hence the separate functions. Debugging purposes. Thanks for the input about the byte array I will keep it in mind.

You misunderstand. He is saying what I said above. Modulo is exclusive. $r \% 255$ is in the interval $[0, 255)$ ($0 \leq r < 255$). Hence, $r \% 255$ has a maximum value of 254. You need to change it to $r \% 256$, as 255 is a valid byte, which your code does not account for.

(Aug 2, 2017 09:33 PM)gigajew Wrote: [\[CITE\]](#) Thanks for the input I will review it tomorrow With the modifications specified above. Thank you, Phizo
No problem. That is better. :)

I see you have changed one of them to `r & 254`. That is good (aside from being a byte off). `r % 2^n` is equivalent to `r & (2^n - 1)` for all $n > 0$. The latter is much faster. You should change the other operation as well. It will be `r & 255` in this case, but I would advise making a variable `m = 256` and doing `r & (m - 1)`. This is equivalent to `r % m`, but again, significantly faster. Keep in mind this is only in the case `r` is non-negative or the language supplies `%` as true modulo rather than remainder. In C# (and many other languages), `%` is actually the remainder. `-1 % 2 = 1` when `%` is true modulo. Here, it is `-1 % 2 = -1`. I assume that's why you used bitwise AND instead. Not for efficiency reasons. What you have is fine (it is actually better), but just keep this in mind (and update your code), as it's very important. You will run into logical errors otherwise, which are not always easy to spot.

gigajew - August 4, 2017, 8:55 pm

Final version, suggestions implemented and lots of mistakes realized

0xDEAD10CC - August 5, 2017, 9:54 am

(Aug 4, 2017 04:48 AM)gigajew Wrote: [\[CITE\]](#) I know 255 is a valid byte. Did you even try running the code and verifying the output? Hence the separate functions. Debugging purposes. Thanks for the input about the byte array I will keep it in mind.

Little late - but did YOU even try running the code and verifying output? Clearly not...

gigajew - August 5, 2017, 10:05 am

(Aug 5, 2017 09:54 AM)0xDEAD10CC Wrote: [\[CITE\]](#) Little late - but did YOU even try running the code and verifying output? Clearly not...

I did, I was just expecting a different result than it was supposed to produce which caused the problems

Ninja - August 5, 2017, 2:31 pm

(Aug 5, 2017 10:05 AM)gigajew Wrote: [\[CITE\]](#) I did, I was just expecting a different result than it was supposed to produce which caused the problems

This is why testing and high code coverage is a must.