

# Forum:[Hack Forums](#)

Thread: [Capsule, File Encryption RSA + AES256](#)

**Paskowsky** - January 13, 2020, 6:59 am

I'm releasing a small class i've made. I use it to store user settings.

Working principle is really simple :

To crypt something you need rsa public key.

Code CAPSULE = { AES(DATA,KEY,IV),RSA\_ENCRYPT(KEY),RSA\_ENCRYPT(IV)}

To decrypt you will need rsa private key.

Code DATA = AES(CAPSULE[0],RSA\_DECRYPT(CAPSULE[1]),RSA\_DECRYPT(CAPSULE[2]))  
Signature and verification follow a similar inner workings

Code SIGNED = [DATA,RSA\_SIGN(DATA)]

ObfuscateArray is used to scramble file structure using a xor key of 32 bytes appended to the file

Source :

Sample project available on GitHub :

<https://github.com/Paskowsky/Capsule>

Happy coding! ?

**MavenGradle** - January 13, 2020, 5:28 pm

Code using System.Collections.Generic;  
using System.Text;  
Can be removed, safely.

Code private static RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();  
Can be made readonly in your particular use case. But wait..

```
Code    private static byte[] ObfuscateArray(byte[] capsule)
        {
            const int keyLength = 32;

            byte[] newCapsule = new byte[capsule.Length + keyLength];
            rng.GetBytes(newCapsule);

            for (int x = 0; x < capsule.Length; x++)
            {
                newCapsule[x + keyLength] = (byte)(capsule[x] ^ newCapsule[(x % keyLength)]);
            }
        }
```

```

        return newCapsule;
    }

    private static byte[] DeobfuscateArray(byte[] capsule)
    {
        const int keyLength = 32;

        byte[] newCapsule = new byte[capsule.Length - keyLength];

        for (int x = 0; x < newCapsule.Length; x++)
        {
            newCapsule[x] = (byte)(capsule[x + keyLength] ^ capsule[x % keyLength]);
        }

        return newCapsule;
    }

```

Why are you declaring a keyLength of 32 in both methods, instead of going the object-oriented route and declaring this once, making it easier to maintain and in turn use less lines of code? Doing things the way you're doing them opens you up to errors, failure and decreases overall maintainability.

In fact, why haven't you made this entire class object oriented, allowing for better re-usability? If you don't want others to inherit functionality, make the class sealed. If your talking point is security for some bullshit reason, well, let's have a look at your example. You are holding the following variables in memory:

```

Code private byte[] privateKey;
private byte[] publicKey;

```

One can easily dump those keys and crack your security. In fact, you're dumping these variables on the local file system yourself in your example. There's no argument there, that would be a silly discussion. There's no reason not to use OOP principles and write clean, reusable code in a case like this.

```

Code throw new Exception("Failed to verify");

```

Why aren't you throwing an ArgumentException or CryptographicException or using a custom exception of your own making so people can properly catch this issue and adjust accordingly? Something like:

```

Code using System.Security;

```

```

namespace Capsule
{
    public sealed class CapsuleVerificationException : CryptographicException
    {
        ...
        public CapsuleVerificationException()
        {
            ...
        }

        public CapsuleVerificationException(string message) : base(message)
        {
            ...
        }
    }
}

```

```
    }  
  }  
}
```

Same thing here:

```
Code if (rsa.PublicOnly)  
    throw new Exception();
```

Actually, now that I think of it, you're not checking any of the variables that get passed to the functions inside the Capsule class except for that one exception you throw above which isn't even an accurate exception to be throwing. You should be checking user-input.

Other than that, you've done a pretty good job on this.

Your examples are definitely rough around the edges and need to be corrected:

```
Code try  
{  
...  
}  
catch (Exception ex)  
{  
    MessageBox.Show("_error_!\r\n" + ex.ToString());  
}
```

E.g. Don't catch them all and there's no need to call ToString() on an exception because internally, the ToString() method of an object will be called when you add an object to a string in the .NET Framework.

```
Code private void exportPubButton_Click(object sender, EventArgs e)  
{  
    string path = SaveFile("public.key", "Key (.key)|*.key");  
    if (string.IsNullOrEmpty(path))  
        return;  
  
    File.WriteAllBytes(path, publicKey);  
}
```

Here, there's no error handling at all.

Your code documentation should also be completed, you're missing quite a few things.

Git-wise, your commit messages hold no value. Git commit messages should contain a short summary of the changes you implemented. Something like '[first commit](#)' isn't exactly very meaningful, especially because [it isn't your first commit](#)..

If you need resources on a good git workflow, take a look at gitflow:  
<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

*Paskowsky - January 14, 2020, 2:01 am*

Thank you for your accurate review!

About the example app, I know storing privatekey and publickey like that breaks any basic rule of security. But that is just an example and key management should be done by user.

About exceptions, I've not defined custom exceptions for simplicity.

When i'm using this in production code i check input before using it in capsule.

Feel free to make any push request and or open issues on github!

About git, i'm not the best at it but i'm learning! Thank you for the useful resources ?