

Forum:[Hack Forums](#)

Thread: [Cannot decrypt files with a size greater than 100MB using AES](#)

Iera Namhorun - May 1, 2019, 10:33 pm

I use this library

<https://www.codeproject.com/Articles/769741/Csharp-AES-bits-Encryption-Library-with-Salt>

my code :

Spoiler [\(Click to View\)](#)

```
Code    public void encryptFile(string file, string password)
    {
        if (new FileInfo(file).Length <= 104857600L) //100 * 1024 * 1024
        {
            byte[] bytesToEncrypt = File.ReadAllBytes(file);
            byte[] array = Encoding.UTF8.GetBytes(password);
            array = SHA256.Create().ComputeHash(array);
            byte[] bytes = encryptAES(bytesToEncrypt, array);
            File.WriteAllBytes(file, bytes);
        }
        else
        {
            byte[] buff = new byte[8192];
            byte[] array = Encoding.UTF8.GetBytes(password);
            array = SHA256.Create().ComputeHash(array);
            using (BinaryReader reader = new BinaryReader(File.Open(file, FileMode.Open)))
            {
                buff = encryptAES(reader.ReadBytes(4096), array);
            }
            using (BinaryWriter writer = new BinaryWriter(File.Open(file, FileMode.Open)))
            {
                writer.Write(buff);
            }
        }
    }
}
```

I can encrypt files under 100 mb in size and size above 100 mb, but after that I can't decrypt files above from 100 mb to the original file format.

can someone help me fix this so I can decrypt files over 100 mb in size?

JUSTIN - May 11, 2019, 3:12 pm

IWhat's the point of that size check? Switching between File.ReadAllBytes and using a BinaryReader with a buffer? For things like this, you should always use Streams that allow you to piece data together and output it on the disk immediately. You don't really want to load huge amounts of data into memory. Here's a fully functional example:

Old Code [\(Click to View\)](#)

edit: use the code from this post instead:

<https://hackforums.net/showthread.php?tid=5965865&pid=58568462#pid58568462>

This encrypts/decrypts a file using AES-256-CFB.

This implementation automatically handles salt as well. Simply provide the input path/output path/encryption key.

If you're calling EncryptFile, the input should be the path to a normal file and the output should be a path to wherever you want the decrypted file to be.

Conversely, when calling DecryptFile, the input should be a previously encrypted file and the output should be a path to restore the original file to.

Due to the way this works, there is not an 100 mb limit on it's size.

I've used this method on 4+ GB files with no problems whatsoever.

Iera Namhorun - May 12, 2019, 8:04 am

(May 11, 2019 03:12 PM)JUSTIN Wrote: [\[CITE\]](#) IWhat's the point of that size check? Switching between File.ReadAllBytes and using a BinaryReader with a buffer? For things like this, you should always use Streams that allow you to piece data together and output it on the disk immediately. You don't really want to load huge amounts of data into memory. Here's a fully functional example:

This encrypts/decrypts a file using AES-256-CFB.

This implementation automatically handles salt as well. Simply provide the input path/output path/encryption key.

If you're calling EncryptFile, the input should be the path to a normal file and the output should be a path to wherever you want the decrypted file to be.

Conversely, when calling DecryptFile, the input should be a previously encrypted file and the output should be a path to restore the original file to.

Due to the way this works, there is not an 100 mb limit on it's size.

I've used this method on 4+ GB files with no problems whatsoever.

thank you very much, i will try it ?

0xDEAD10CC - May 12, 2019, 5:22 pm

(May 11, 2019 03:12 PM)JUSTIN Wrote: [\[CITE\]](#) ...

This is not exception safe code. This is the reason why C# has the using statement.

JUSTIN - May 12, 2019, 5:26 pm

(May 12, 2019 05:22 PM)0xDEAD10CC Wrote: [\[CITE\]](#) This is not exception safe code. This is the reason why C# has the using statement.

better off with try/catch/finally than using. but yeah, i should've included that. my bad.

0xDEAD10CC - May 12, 2019, 5:34 pm

(May 12, 2019 05:26 PM)JUSTIN Wrote: [\[CITE\]](#) better off with try/catch/finally than using. but yeah, i should've included that. my bad.

The catch block would depend on if you're actually going to treat the exceptions rather than just ignore them. Otherwise, exceptions are a useful debugging and awareness 'tool' to keep. In your case all your functions return void, so I wouldn't recommend this unless you intend on returning a success flag or object of some kind, or if you want to introduce logging functionality. These exceptions can also be caught outside the functions too within the caller's scope too; which would allow for code branching to allow the implementor to decide what to do at that point if the action failed.

JUSTIN - May 12, 2019, 5:38 pm

(May 12, 2019 05:34 PM)0xDEAD10CC Wrote: [\[CITE\]](#) The catch block would depend on if you're actually going to treat the exceptions rather than just ignore them. Otherwise, exceptions are a useful debugging and awareness 'tool' to keep. In your case all your functions return void, so I wouldn't recommend this unless you intend on returning a success flag or object of some kind, or if you want to introduce logging functionality.

you can still create a try/finally block without including a catch. i simply suggested try/catch/finally over using to give the option of handling exceptions, and you mentioned the code wasn't exception safe. if you don't catch it the program is going to crash regardless.

it's also very unlikely that an exception will occur in this code, unless you don't have access to a file in question or the decryption key is incorrect.

Edit:

Okay, here's a version with exception handling implemented. I've also included the GenerateSalt func that I accidentally excluded last time.

By default, it will log exceptions and return 'true' or 'false' indicating whether or not the encryption/decryption was completed without error.

New Version ([Click to View](#))

0xDEAD10CC - May 12, 2019, 5:53 pm

(May 12, 2019 05:38 PM)JUSTIN Wrote: [\[CITE\]](#) you can still create a try/finally block without including a catch. i simply suggested try/catch/finally over using to give the option of handling exceptions, and you mentioned the code wasn't exception safe. if you don't catch it the program is going to crash regardless.

it's also very unlikely that an exception will occur in this code, unless you don't have access to a file in question or the decryption key is incorrect.

Yes I'm aware - but at that point what is wrong with the using statement; which is syntactic sugar for the exact same thing?

I don't think you understood what I was getting at though. What I meant about the exception safety was not about preventing exceptions being thrown, but rather the fact that those streams don't get properly disposed if an exception occurs between the point where the instantiation of the IDisposable object has been made (successfully), and your explicit calls to Close(). The using statement (just like the equivalent try-finally) would allow an implicit Dispose() behind the scenes to ensure resources are properly being disposed, even in the event of an exception which would normally prevent those calls from being made. (Of course there is the edge case with background threads, where you can't rely on the finally block to execute.)

edit: New code looks much better though. At least it gives OP a template to choose what he wants to do with any exceptions.

JUSTIN - May 12, 2019, 5:57 pm

(May 12, 2019 05:53 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Yes I'm aware - but at that point what is wrong with the using statement; which is syntactic sugar for the exact same thing? ?

I don't think you understood what I was getting at though. What I meant about the exception safety was not with exceptions being thrown in itself, but rather the fact that those streams don't get properly disposed if an exception occurs between the point where the instantiation of the IDisposable object has been made (successfully), and your explicit calls to Close(). The using statement (just like the equivalent try-finally) would allow an implicit Dispose() behind the scenes to ensure resources are properly being disposed, even in the event of an exception which would normally prevent those calls from being made.

When calling Dispose on any object that implements IDisposable, the stream in question is automatically closed. If that's what you're worried about my finally block "Close" calls can be replaced with "Dispose" and it'll accomplish the same thing, except it won't wait for the objects to be cleaned up automatically via garbage collection.

edit: updated my paste to use Dispose calls rather than Close. best of both worlds now :)

edit: according to microsoft, both FileStream and CryptoStream release all resources - so I guess calling either works?

source:

<https://docs.microsoft.com/en-us/dotnet/api/system.io.filestream.close?view=netframework-1.1>

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.cryptostream.close?view=netframework-1.1>

0xDEAD10CC - May 12, 2019, 6:08 pm

(May 12, 2019 05:57 PM)JUSTIN Wrote: [\[CITE\]](#) When calling Dispose on any object that implements IDisposable, the stream in question is automatically closed. If that's what you're worried about my finally block "Close" calls can be replaced with "Dispose" and it'll accomplish the same thing, except it won't wait for the objects to be cleaned up automatically via garbage collection.

edit: updated my paste to use Dispose calls rather than Close. best of both worlds now :)

edit: according to microsoft, both FileStream and CryptoStream release all resources - so I guess calling either works?

source:

<https://docs.microsoft.com/en-us/dotnet/api/system.io.filestream.close?view=netframework-1.1>

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.cryptostream.close?view=netframework-1.1>

No - you're still not understanding what I'm saying. My original statement was directed at your initial code that wasn't using the finally block or a using statement.

Code IDisposable creation

```
// ... code ...
```

Explicit IDisposable Dispose() call or some wrapper call like Close()

^ Take the above for instance. If an exception gets thrown between the two relevant points, then resources are not being properly disposed of. The using statement and the finally block (as long as you remember to call Dispose()) protect against this (with the exception of the background thread cases).

My statement wasn't about Close() vs Dispose(). Your revised code was fine, I was just pointing this out in regards to my initial remarks about it not being safe with exceptions.

edit: This is also the reason why people wouldn't consider code like this to be exception safe in C++:

```
Code do_stuff();
```

```
cleanup();
```

Your revised code is good though - apologies for the miscommunication.

JUSTIN - May 12, 2019, 6:11 pm

(May 12, 2019 06:08 PM)0xDEAD10CC Wrote: [\[CITE\]](#) No - you're still not understanding what I'm saying. My original statement was directed at your initial code that wasn't using the finally block or a using statement.

Code IDisposable creation

```
// ... code ...
```

Explicit IDisposable Dispose() call or some wrapper call like Close()

^ Take the above for instance. If an exception gets thrown between the two relevant points, then resources are not being properly disposed of. The using statement and the finally block protect against this (with the exception of the background thread cases).

My statement wasn't about Close() vs Dispose().

Yeah, I understood your point. The streams would remain open & resources wouldn't be disposed if an exception was thrown during the code. That's why I fixed it. After you responded again after I posted the improved code I figured you were stating that 'using' could be preferred over simply calling Close() in a finally block because it automatically closes streams and frees resources from memory. Simple miscommunication, but thanks to your post regarding the issue it's fixed for future viewers of this thread, so gj for pointing it out ?

Iera Namhorun - June 14, 2019, 7:53 am

(May 12, 2019 05:38 PM)JUSTIN Wrote: [\[CITE\]](#) Edit:

Okay, here's a version with exception handling implemented. I've also included the GenerateSalt func that I accidentally excluded last time.

By default, it will log exceptions and return 'true' or 'false' indicating whether or not the encryption/decryption was completed without error.

New Version [\(Click to View\)](#)

thank you, this works fine but why when doing decryption always leaves a strange text at the end of the contents of the decrypted text file ?

0xDEAD10CC - June 15, 2019, 12:20 pm

(Jun 14, 2019 07:53 AM)Iera Namhorun Wrote: [\[CITE\]](#) thank you, this works fine but why when doing decryption always leaves a strange text at the end of the contents of the decrypted text file ?

You probably need to do more reading about how block ciphers work.

Iera Namhorun - June 21, 2019, 6:22 pm

(Jun 15, 2019 12:20 PM)0xDEAD10CC Wrote: [\[CITE\]](#) You probably need to do more reading about how block ciphers work.

okay, thankyou.