Why does a bad password cause "Padding is invalid and cannot be removed"?

Asked 16 years, 4 months ago Modified 7 years, 1 month ago Viewed 73k times

40

I needed some simple string encryption, so I wrote the following code (with a great deal of "inspiration" from here):

```
    // create and initialize a crypto algorithm
    private static SymmetricAlgorithm getAlgorithm(string password) {
        SymmetricAlgorithm algorithm = Rijndael.Create();
        Rfc2898DeriveBytes rdb = new Rfc2898DeriveBytes(
            password, new byte[] {
            0x53,0x6f,0x64,0x69,0x75,0x6d,0x20,             // salty goodness
            0x43,0x68,0x6c,0x6f,0x72,0x69,0x64,0x65
        }
        );
        algorithm.Padding = PaddingMode.ISO10126;
        algorithm.Key = rdb.GetBytes(32);
        algorithm.IV = rdb.GetBytes(16);
        return algorithm;
    }

    /*
     * encryptString
     * provides simple encryption of a string, with a given password
     */
    public static string encryptString(string clearText, string password) {
        SymmetricAlgorithm algorithm = getAlgorithm(password);
        byte[] clearBytes = System.Text.Encoding.Unicode.GetBytes(clearText);
        MemoryStream ms = new MemoryStream();
        CryptoStream cs = new CryptoStream(ms, algorithm.CreateEncryptor(),
CryptoStreamMode.Write);
        cs.Write(clearBytes, 0, clearBytes.Length);
        cs.Close();
        return Convert.ToBase64String(ms.ToArray());
    }

    /*
     * decryptString
     * provides simple decryption of a string, with a given password
     */
    public static string decryptString(string cipherText, string password) {
        SymmetricAlgorithm algorithm = getAlgorithm(password);
        byte[] cipherBytes = Convert.FromBase64String(cipherText);
        MemoryStream ms = new MemoryStream();
        CryptoStream cs = new CryptoStream(ms, algorithm.CreateDecryptor(),
CryptoStreamMode.Write);
        cs.Write(cipherBytes, 0, cipherBytes.Length);
        cs.Close();
        return System.Text.Encoding.Unicode.GetString(ms.ToArray());
    }
```

The code appears to work fine, except that when decrypting data with an incorrect key, I get a CryptographicException - "Padding is invalid and cannot be removed" - on the cs.Close() line in decryptString.

example code:

```
    string password1 = "password";
    string password2 = "letmein";
    string startClearText = "The quick brown fox jumps over the lazy dog";
    string cipherText = encryptString(startClearText, password1);
    string endClearText = decryptString(cipherText, password2);     //
exception thrown
```

My question is, is this to be expected? I would have thought that decrypting with the wrong password would just result in nonsense output, rather than an exception.

c# .net exception encryption

Share
Improve this question
Follow

edited May 8, 2017 at 19:03
jbtule
31.8k ● 12 ● 100 ● 129

asked Aug 14, 2008 at 23:14
Blorgbeard
103k ● 50 ● 235 ● 276

6 This saved me so much time with your comment: `"The code appears to work fine, except that when decrypting data with an incorrect key"` I *swore* I had copied the keys, but looking 2x I didn't. Hopefully this helps someone else before looking at the padding mechanism or changing code. – atconway Jul 31, 2013 at 15:58

9 Answers

Sorted by: Highest score (default)

27

Although this have been already answered I think it would be a good idea to explain **why** it is to be expected.

A padding scheme is usually applied because most cryptographic filters are not semantically secure and to prevent some forms of cryptoatacks. For example, usually in RSA the OAEP padding scheme is used which prevents some sorts of attacks (such as a chosen plaintext attack or blinding).

A padding scheme appends some (usually) random garbage to the message m before the message is sent. In the OAEP method, for example, two Oracles are used (this is a simplistic explanation):

1. Given the size of the modulus you padd k1 bits with 0 and k0 bits with a random number.

2. Then by applying some transformation to the message you obtain the padded message wich is encrypted and sent.

That provides you with a randomization for the messages and with a way to test if the message is garbage or not. As the padding scheme is reversible, when you decrypt the message whereas you can't say anything about the integrity of the message itself you can, in fact, make some assertion about the padding and thus you can know if the message has been correctly decrypted or you're doing something wrong (i.e someone has tampered with the message or you're using the wrong key)

Share Improve this answer Follow

answered Aug 25, 2008 at 15:46
Jorge Córdoba
52.1k ● 11 ● 82 ● 130

Jorge, thanks for the explanation. I'm seeing the same behavior as described, the data that is decrypted is correct. am I supposed to eat this exception, or (hopefully) there is something I'm doing incorrectly that I can correct? what is going wrong when the exception is thrown? all the posts I've read seem to be written by people who are more interested in making the exception going away. in my case I want my usage to be correct :) – stuck Jul 22, 2010 at 22:11

1 This does not answer the OP's question, the question is about a symmetric block cipher Rijndael, not a asymmetric cipher such as RSA. For a block cipher padding is added to make the data to be encrypted a multiple of the block size, generally using PKCS#7 (née PKCS#5). With these padding schemes random bytes are **not** added. Note that padding ISO 10126 was withdrawn and the random bytes did not add any security. The answer would be to the point if it spoke to symmetric encryption padding. Suggestion: fix the answer to answer the question. ZOMG, Senior Developer. – zaph Jul 8, 2016 at 19:52

1 @zaph I'm just explaining generically why you get a invalid padding exception instead of garbage when you decrypt using and invalid password. RSA was just an example, the specific padding scheme used was and example. It answers the op question because it says: Yes, it is expected and here's why and the explanation about that strange "padding" reference that you can only understand if you understand what a padding scheme is and why it is used, but would let you baffled if you don't (what does encryption has to do with padding???) – Jorge Córdoba Jul 8, 2016 at 20:36

The question is about Rijndael padding and you explain RSA padding which are very different and done for different reasons. OK, I understand that people get confused about the difference between symmetric and asymmetric encryption and that RSA seems to be more visible of late and many people are using RSA when AES is a better fit. But it would be better if the answer fit question. – zaph Jul 8, 2016 at 21:51

17

I experienced a similar "Padding is invalid and cannot be removed." exception, but in my case the key IV and padding were correct.

It turned out that flushing the crypto stream is all that was missing.

Like this:

```
            MemoryStream msr3 = new MemoryStream();
            CryptoStream encStream = new CryptoStream(msr3,
RijndaelAlg.CreateEncryptor(), CryptoStreamMode.Write);
            encStream.Write(bar2, 0, bar2.Length);
            // unless we flush the stream we would get "Padding is invalid and
cannot be removed." exception when decoding
            encStream.FlushFinalBlock();
            byte[] bar3 = msr3.ToArray();
```

Share Improve this answer Follow

answered Nov 7, 2013 at 11:44
Yaniv
504 ● 4 ● 11

1 The same should do the `encStream.Close();` . – sharpener Aug 30, 2014 at 14:41

2 Maybe you should use `using` on those objects since they are disposable. That would be enough. – user2173353 May 18, 2016 at 11:14

1 Also note that a simple `Flush()` that you would normally do with a stream is *not* the same as the `FlushFinalBlock()` required for a `CryptoStream`. That's slightly ambiguous in the description preceding the code. – Marc L. Oct 13, 2016 at 10:35

This is unrelated to the question, the questioner knows the cause of the exception, the bad password (not flushing), and is asking why a bad password causes this specific exception. – jbtule May 8, 2017 at 18:52

6

If you want your usage to be correct, you should add authentication to your ciphertext so that you can verify that it is the correct pasword or that the ciphertext hasn't been modified. The padding you are using ISO10126 will only throw an exception if the last byte doesn't decrypt as one of 16 valid values for padding (0x01-0x10). So you have a 1/16 chance of it NOT throwing the exception with the wrong password, where if you authenticate it you have a deterministic way to tell if your decryption is valid.

Using crypto api's while seemingly easy, actually is rather is easy to make mistakes. For example you use a fixed salt for for you key and iv derivation, that means every ciphertext encrypted with the same password will reuse it's IV with that key, that breaks semantic security with CBC mode, the IV needs to be both unpredictable and unique for a given key.

For that reason of easy to make mistakes, I have a code snippet, that I try to keep reviewed and up to date (comments, issues welcome):

Modern Examples of Symmetric Authenticated Encryption of a string C#.

If you use it's `AESThenHMAC.AesSimpleDecryptWithPassword(ciphertext, password)` when the wrong password is used, `null` is returned, if the ciphertext or iv has been modified post encryption `null` is returned, you will never get junk data back, or a padding exception.

Share
Improve this answer
Follow

edited May 23, 2017 at 12:18
Community Bot
1 ● 1

answered Feb 20, 2013 at 2:01
jbtule
31.8k ● 12 ● 100 ● 129

4

If you've ruled out key-mismatch, then besides `FlushFinalBlock()` (see Yaniv's answer), calling `Close()` on the `CryptoStream` will also suffice.

If you are cleaning up resources strictly with `using` blocks, be sure to nest the block for the `CryptoStream` itself:

```
using (MemoryStream ms = new MemoryStream())
using (var enc = RijndaelAlg.CreateEncryptor())
{
  using (CryptoStream encStream = new CryptoStream(ms, enc,
CryptoStreamMode.Write))
  {
    encStream.Write(bar2, 0, bar2.Length);
  } // implicit close
  byte[] encArray = ms.ToArray();
}
```

I've been bitten by this (or similar):

```
using (MemoryStream ms = new MemoryStream())
using (var enc = RijndaelAlg.CreateEncryptor())
using (CryptoStream encStream = new CryptoStream(ms, enc,
CryptoStreamMode.Write))
{
  encStream.Write(bar2, 0, bar2.Length);
  byte[] encArray = ms.ToArray();
} // implicit close -- too late!
```

Share Improve this answer Follow

answered Oct 13, 2016 at 10:40
Marc L.
3,370 ● 1 ● 33 ● 43

1 But if you do `encStream.FlushFinalBlock()` I would assume it doesn't matter where you do `ms.ToArray()` (whether outside or inside of `CryptoStream` 's `using`). – nawfal May 8, 2017 at 7:24

The OP know's it's a key mismatch causing the error and asked why a key mismatch causes this error. – jbtule May 8, 2017 at 18:55

3

Yes, this is to be expected, or at least, its exactly what happens when our crypto routines get non-decryptable data

Share Improve this answer Follow

answered Aug 14, 2008 at 23:25

David Wengier
10.2k ● 5 ● 40 ● 43

3

Another reason of the exception might be a race condition between several threads using decryption logic - native implementations of ICryptoTransform are *not thread-safe* (e.g. SymmetricAlgorithm), so it should be put to exclusive section, e.g. using *lock*. Please refer here for more details: http://www.make-awesome.com/2011/07/system-security-cryptography-and-thread-safety/

Share Improve this answer Follow

answered Oct 10, 2016 at 10:43

Denis Tikhomirov
179 ● 1 ● 6

1

There may be some unread bytes in the CryptoStream. Closing before reading the stream completely was causing the error in my program.

Share Improve this answer Follow

answered Aug 4, 2010 at 9:16

R D
521 ● 5 ● 12

0

I had a similar problem, the issue in decrypt method was initializing an empty memory stream. when it worked when I initialized it with the cipher text byte array like this:

```
MemoryStream ms = new MemoryStream(cipherText)
```

Share Improve this answer Follow

answered Feb 2, 2015 at 12:13

Mina Wissa
11k ● 13 ● 95 ● 161

The answer updated by the user "atconway" worked for me.

-2

The problem was not with the padding but the key which was different during encryption and decryption. The key and iv should be same during encypting and decrypting the same value.

Share Improve this answer Follow

answered Apr 15, 2015 at 20:12

RoopzD

1