

Forum: [Hack Forums](#)

Thread: [AES256 without libs, an easy way, and string 'password'](#)

Dhoroty - January 2, 2020, 5:50 am

This won't be a tutorial, so you should have basic knowledge in C# and about what is AES, and what is symmetric encryption, etc.

If you want to do AES, you probably googled it up, and found the M\$ site with that great code example:

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes?view=netframework-4.8>

There are 2 functions, one for encrypt (`EncryptStringToBytes_Aes`) , one for decrypt (`DecryptStringFromBytes_Aes`)

Both will requires 2 byte arrays, named `byte[] Key`, `byte[] IV`.

You can create those with this:

Code using (`Aes myAes = Aes.Create()`) and reuse the key and IV like this:

Code `myAes.Key = Key;`

`myAes.IV = IV;`

But what if, you don't want to create or store key + iv in a byte array, nor you want your users to enter or copy it to your app?

Users like passwords, so if you want to use AES for example in an encrypted chat app, where users exchange the 'password' before using the app, and they'll use it for communication, that would be easier, than `byte[]` exchange in a paper, right?

So here is a function, that will generate KEY and IV for you from a single UTF8 string:

Code static `byte[][] GetHashKeys(string key)`

```
{  
    if (key == null) throw new ArgumentNullException(nameof(key));
```

```
    byte[][] result = new byte[2][];
```

```
    byte[] hashKey = new
```

```
SHA256CryptoServiceProvider().ComputeHash(Encoding.UTF8.GetBytes(key));
```

```
    byte[] iv = new byte[16];
```

```
    for (int i = 0; i < 16; i++) iv[i] = hashKey[i];
```

```
    result[0] = hashKey;
```

```
    result[1] = iv;
```

```
    return result;
```

```
}
```

I hope you'll use it in great projects!

Edit: to clarify this is not my original code! But it is free to use and modify in any projects, so posted here to help you when doing projects that maybe needs it.

Edit #2: Applied MavenGradle's code simplify. Thanks for that

MavenGradle - January 2, 2020, 8:55 am

(Jan 2, 2020 06:50 AM)Dhoroty Wrote: [\[CITE\]](#) Both will requires 2 byte arrays, named byte[] Key, byte[] IV.

In terms of performance, the code snippet that you have provided will generate an initialization vector (IV) that is equal to the key in all but length. I don't know what the point of any of the following code is:

```
Code byte[] rawKey = enc.GetBytes(key);
byte[] rawIV = enc.GetBytes(key);
```

```
byte[] hashKey = sha2.ComputeHash(rawKey);
byte[] hashIV = sha2.ComputeHash(rawIV);
```

While you could simply be doing this:

```
Code private static byte[][] GetHashKeys(string key)
{
    if (key == null) throw new ArgumentNullException(nameof(key));

    byte[][] result = new byte[2][];
    byte[] hashKey = new SHA256CryptoServiceProvider()
        .ComputeHash(Encoding.UTF8.GetBytes(key));

    byte[] iv = new byte[16];
    Array.Copy(hashKey, iv, 16);

    result[0] = hashKey;
    result[1] = iv;

    return result;
}
```

If you want even better performance, you could go for this:

```
Code private static byte[][] GetHashKeys(string key)
{
    if (key == null) throw new ArgumentNullException(nameof(key));

    byte[][] result = new byte[2][];
    byte[] hashKey = new SHA256CryptoServiceProvider()
        .ComputeHash(Encoding.UTF8.GetBytes(key));

    byte[] iv = new byte[16];
    for (int i = 0; i < 16; i++) iv[i] = hashKey[i];

    result[0] = hashKey;
    result[1] = iv;

    return result;
}
```

This is smaller, requires less computational work to be done and is more efficient. Do note that the hash that is calculated by the ComputeHash method of the SHA256CryptoServiceProvider is always

a byte array with a length of 32 (32 bytes/256-bit/64 hex characters), due to the very nature of how SHA256 operates:

<https://en.wikipedia.org/wiki/SHA-2>

A random initialization vector (IV) can also safely and securely be generated using built-in Framework functionality. This is what they're doing in the example that you posted as well, by using the abstract `System.Security.Cryptography.Aes` class:

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes?view=netframework-4.8>

This is indeed what you should be doing for security purposes:

Code using (`Aes aes = Aes.Create()`)

```
{
    byte[] key = aes.Key;
    byte[] iv = aes.IV;
}
```

Using a concrete AES implementation with a fixed block size (128 bits) and iteration count:

Code using (`AesManaged aes = new AesManaged()`)

```
{
    byte[] key = aes.Key;
    byte[] iv = aes.IV;
}
```

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aesmanaged?view=netframework-4.8>

Dhoroty - January 2, 2020, 9:03 am

You're right, those was for future use, but was left like this. So this form surely can be simplified. Thanks for the idea. Will change the code when at computer.

MavenGradle - January 2, 2020, 11:07 am

(Jan 2, 2020 06:50 AM)Dhoroty Wrote: [\[CITE\]](#) But what if, you don't want to create or store key + iv in a byte array, nor you want your users to enter or copy it to your app?

Users like passwords, so if you want to use AES for example in an encrypted chat app, where users exchange the 'password' before using the app, and they'll use it for communication, that would be easier, than `byte[]` exchange in a paper, right?

Be careful around your wording. Passwords should not be encrypted with AES and the likes and stored or exchanged in an encrypted format. They should be hashed and stored as such in a database or exchanged as such. Encryption opens the possibility of decryption, which is not what you want for passwords. Encrypting a password before hashing it, is something else entirely and in that case would be secure versus encryption alone.

Ideally, your passwords are hashed, in a one-way format, so that it is impossible or near impossible to retrieve the original password.

Dhoroty - January 2, 2020, 11:18 am

Yeah, newer use anything else than hash for STORING passwords! But I refferred to term password as the base of the iv+key, as it seen by the source.

The example I wrote about is a chat app, that use the 'password' to generate the iv+key to communicate. Like a p2p encrypted chat app where you exchange the encryption key in paper before first use.

Password is not the best word, but that was the best I can come with.