# Overview

Everything compiled using .NET Runtime 8.0.10 with SDK 8.0.403 (C#)
Players can figure this out through this string in the binary:

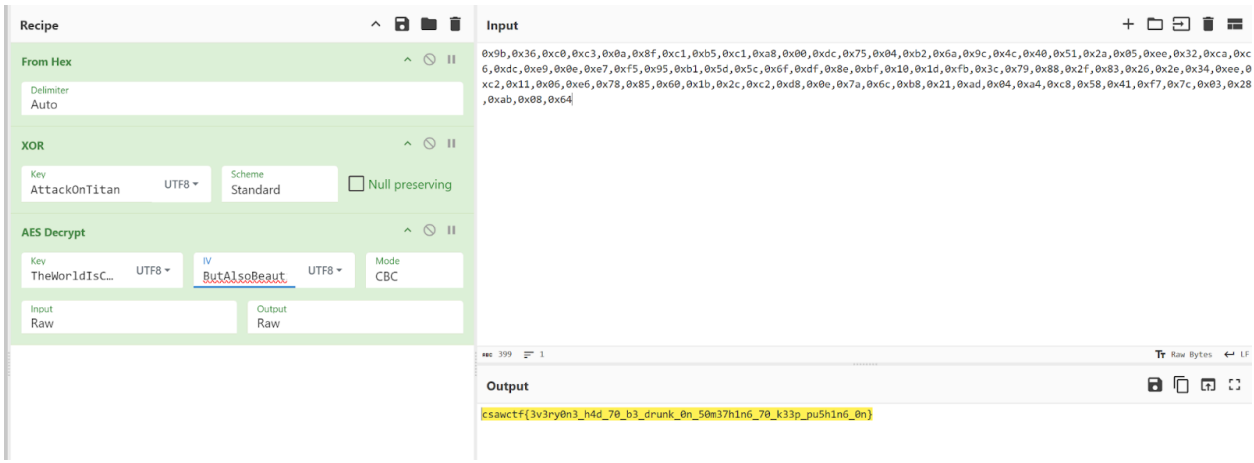
| | | | |
|---|---|---|---|
| .rdata:00007... | 00000006 | C | \nen-US |
| .rdata:00007... | 00000017 | C | *Microsoft Corporationi |
| .rdata:00007... | 0000002C | C | Microsoft Corporation. All rights reserved. |
| .rdata:00007... | 00000048 | C | 8.0.1024.46610^8.0.10+81cabf2857a01351e5ab578947c7403a5b128ad1 Microsoft |
| .rdata:00007... | 00000005 | C | .NET |
| .rdata:00007... | 0000002F | C | RepositoryUrlBhttps://github.com/dotnet/runtime |
| .rdata:00007... | 0000000B | C | PreferInbox |

Simple flag checker with Attack on titan references

```Java
using System;
using System.Security.Cryptography;
using System.Text;
using System.Linq;

namespace AOT
{
    class Program
    {
        // Hardcoded AES key and IV
        // Encrypted and XORed flag stored as a global variable
        static readonly byte[] encryptedFlag = new byte[]
        {

0x9b,0x36,0xc0,0xc3,0x0a,0x8f,0xc1,0xb5,0xc1,0xa8,0x00,0xdc,0x75,0x04,0xb2,0x6a,0x9c,0x4c,0x40,0x51,0x
2a,0x05,0xee,0x32,0xca,0xc6,0xdc,0xe9,0x0e,0xe7,0xf5,0x95,0xb1,0x5d,0x5c,0x6f,0xdf,0x8e,0xbf,0x10,0x1d
,0xfb,0x3c,0x79,0x88,0x2f,0x83,0x26,0x2e,0x34,0xee,0xc2,0x11,0x06,0xe6,0x78,0x85,0x60,0x1b,0x2c,0xc2,0
xd8,0x0e,0x7a,0x6c,0xb8,0x21,0xad,0x04,0xa4,0xc8,0x58,0x41,0xf7,0x7c,0x03,0x28,0xab,0x08,0x64
        };

        static void Main(string[] args)
        {
            Console.WriteLine("Enter the flag:");
            string userFlag = Console.ReadLine();

            // Process the user's input and compare to encryptedFlag
            if (CompareFlag(userFlag))
            {
                Console.WriteLine("Correct flag!");
            }
            else
            {
                Console.WriteLine("Incorrect flag.");
            }
        }
        static byte[] EncryptAndXorFlag(string flag)
        {
```

```
            // Encrypt the flag with AES
            byte[] encryptedFlag = EncryptFlag(flag);

            byte[] xorPattern = Encoding.ASCII.GetBytes("AttackOnTitan");

            // XOR the encrypted data with the repeating "AttackOnTitan" pattern
            for (int i = 0; i < encryptedFlag.Length; i++)
            {
                encryptedFlag[i] ^= xorPattern[i % xorPattern.Length];
            }

            return encryptedFlag;
        }

        static byte[] EncryptFlag(string flag)
        {
            using Aes aes = Aes.Create();
            aes.Key = Encoding.UTF8.GetBytes("TheWorldIsCruelAndMerciless.....");
            aes.IV = Encoding.UTF8.GetBytes("ButAlsoBeautiful");

            ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key, aes.IV);

            using var msEncrypt = new System.IO.MemoryStream();
            using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
            using (var swEncrypt = new System.IO.StreamWriter(csEncrypt))
            {
                swEncrypt.Write(flag);
                swEncrypt.Flush();
            }

            return msEncrypt.ToArray();
        }

        static bool CompareFlag(string userFlag)
        {
            byte[] userEncrypted = EncryptAndXorFlag(userFlag);

            // Compare each byte in the arrays
            if (userEncrypted.Length != encryptedFlag.Length)
                return false;

            for (int i = 0; i < userEncrypted.Length; i++)
            {
                if (userEncrypted[i] != encryptedFlag[i])
                    return false;
            }

            return true;
        }
    }
}
```

Summary of the challenge: The encrypted flag was encrypted using AES with hardcoded key `"TheWorldIsCruelAndMerciless....."` and IV `"ButAlsoBeautiful"` , then XORed with “AttackOnTitan”. So, it can literally be solved with Cyberchef with the reverse of the operation:




The gist here is that it is **difficult** to know what is happening in the binary without debugging symbols.

Compiled using AOT

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net8.0</TargetFramework>
        <PublishAot>true</PublishAot>
        <RuntimeIdentifier>win-x64</RuntimeIdentifier>
    </PropertyGroup>

</Project>
```

```
PS C:\Users\JiaYuChan\source\repos\AOTtest > dotnet publish -c Release -r win-x64 --self-contained
>>
  Determining projects to restore...
  Restored C:\Users\JiaYuChan\source\repos\AOTtest\AOTtest\AOTtest.csproj (in 372 ms).
  AOTtest -> C:\Users\JiaYuChan\source\repos\AOTtest\AOTtest\bin\Release\net8.0\win-x64\AOTtest.dll
```

IDA does not recognize any .NET library functions at all, making static analysis really hard at this point.

```
IDA View-A    Pseudocode-A    Hex View-1    Local Types
{
  __int64 v4; // rax
  __int64 v5; // rax
  __int64 v6; // rcx
  __int64 v7; // rax
  unsigned int v8; // ebx
  __int64 v10[4]; // [rsp+28h] [rbp-20h] BYREF

  v10[0] = 0LL;
  v10[1] = 0LL;
  sub_140005E50(v10);
  v4 = sub_1400027F0(&unk_14012B250);
  *(_DWORD *)(v4 + 72) = -2146233088;
  *(_QWORD *)(v4 + 8) = 0LL;
  *(_DWORD *)(v4 + 72) = -2146233087;
  *(_DWORD *)(v4 + 72) = -2147024882;
  sub_140002A00(qword_1401E64B0 + 8, v4);
  sub_1400C15A0();
  sub_1400D77E0();
  v5 = sub_1400027F0(&unk_140135680);
  sub_140002A00(qword_1401E6750 + 24, v5);
  sub_1400D5E20();
  sub_1400CBCA0(a1, a2);
  qword_1401E7B08 = sub_1400CC200(&unk_1401568C0);
  v6 = *(_QWORD *)(sub_140001DA0() + 80);
  if ( !v6 )
    v6 = sub_14009B590();
  sub_14009CAE0(v6, 1LL, 1LL);
  sub_1400CB8E0();
  v7 = sub_1400CBD50();
  sub_140073030(v7);
  sub_14009BE00();
  sub_140077F20();
  if ( *((_QWORD *)&unk_1401266D0 - 1) )
    sub_14000101D();
  v8 = unk_1401266D0;
  sub_140005FB0(v10);
  return v8;
}
```

We can follow this guide to help with analysis:
https://harfanglab.io/insidethelab/reverse-engineering-ida-pro-aot-net/
Basically, what we can do is generate our own IDA FLIRT signature file for IDA to recognize library functions through writing a program which implements as many dummy library functions as possible. We want to cover as many as we can, with the most important being Crypto functions, especially AES functions. We can give a hint to users saying Crypto libraries are used.

```
Java
using System.Net;
using System.Text;
using System.Security.Cryptography;
using System.IO.Compression;
using System.Text.RegularExpressions;
using System.Xml.Linq;
using System.Text.Json;
using System.Net.Http;
using System.IO;
using System.Linq;
using System.Collections.Generic;
using System;

class Program
```

```
{
    static readonly HttpClient client = new HttpClient();

    static void Main()
    {
        // String manipulation
        string exampleString = "Hello World";
        string lowerString = exampleString.ToLower();
        string upperString = exampleString.ToUpper();
        string trimmedString = exampleString.Trim();
        bool containsHello = exampleString.Contains("Hello");
        string replacedString = exampleString.Replace("World", "Universe");

        // File operations
        string filePath = @"temp.txt";
        File.WriteAllText(filePath, exampleString);
        string readFile = File.ReadAllText(filePath);

        // Networking
        WebClient webClient = new WebClient();
        byte[] data = webClient.DownloadData("http://example.com");

        // Encoding
        string encodedString = System.Convert.ToBase64String(Encoding.UTF8.GetBytes(exampleString));
        byte[] decodedBytes = System.Convert.FromBase64String(encodedString);
        string decodedString = Encoding.UTF8.GetString(decodedBytes);

        // LINQ and Collections
        System.Collections.Generic.List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
        int maxNumber = numbers.Max();
        int minNumber = numbers.Min();
        IEnumerable<int> sortedNumbers = numbers.OrderBy(n => n);

        // Math operations
        double squareRoot = Math.Sqrt(25);
        double power = Math.Pow(2, 3);
        double absoluteValue = Math.Abs(-10.5);

        // Additional Networking Functions
        string url = "http://example.com";
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
        HttpWebResponse response = (HttpWebResponse)request.GetResponse();
        Stream responseStream = response.GetResponseStream();
        StreamReader reader = new StreamReader(responseStream);
        string responseText = reader.ReadToEnd();

        // Cryptographic Functions
        // Simple MD5 hash
        using (MD5 md5 = MD5.Create())
        {
            byte[] inputBytes = Encoding.ASCII.GetBytes("Hello World");
            byte[] hashBytes = md5.ComputeHash(inputBytes);
            string hash = BitConverter.ToString(hashBytes).Replace("-", "").ToLowerInvariant();
        }

        // RSA Encryption and Decryption
        string original = "Hello World!";
```

```
        using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
        {
            byte[] encryptedData = rsa.Encrypt(Encoding.UTF8.GetBytes(original), true);
            byte[] decryptedData = rsa.Decrypt(encryptedData, true);
        }

        // AES Encryption and Decryption
        string aesKey = "0123456789abcdef"; // 16-byte key for AES-128
        string aesIv = "abcdef9876543210"; // 16-byte IV for AES-128
        string plaintext = "This is a secret message";

        string encryptedAes = EncryptAES(plaintext, aesKey, aesIv);
        string decryptedAes = DecryptAES(encryptedAes, aesKey, aesIv);

        Console.WriteLine($"Encrypted AES Text: {encryptedAes}");
        Console.WriteLine($"Decrypted AES Text: {decryptedAes}");

        // XML Parsing
        string xmlString = "<root><element>Value</element></root>";
        XDocument doc = XDocument.Parse(xmlString);
        string elementValue = doc.Root.Element("element").Value;

        // Json parsing
        string jsonString = "{\"name\":\"John Doe\",\"age\":30}";
        using (JsonDocument json_doc = JsonDocument.Parse(jsonString))
        {
            JsonElement root = json_doc.RootElement;
            string name = root.GetProperty("name").GetString();
            int age = root.GetProperty("age").GetInt32();
            Console.WriteLine($"Name: {name}, Age: {age}");
        }

        // Regular Expressions
        string data2 = "Example 123";
        Match match = Regex.Match(data2, @"\d+");
        string matchedNumber = match.Value;

        // File Compression
        string startPath = "./";
        string zipPath = "output.zip";
        ZipFile.CreateFromDirectory(startPath, zipPath);

        // Environment Information
        string osVersion = Environment.OSVersion.ToString();
        int processorCount = Environment.ProcessorCount;

        // Clean up
        File.Delete(filePath);
    }

    // AES Encryption
    private static string EncryptAES(string text, string key, string iv)
    {
        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Encoding.UTF8.GetBytes(key);
            aesAlg.IV = Encoding.UTF8.GetBytes(iv);
```

```
            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

            using (MemoryStream msEncrypt = new MemoryStream())
            {
                using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
                {
                    using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                    {
                        swEncrypt.Write(text);
                    }
                }
                return Convert.ToBase64String(msEncrypt.ToArray());
            }
        }
    }

    // AES Decryption
    private static string DecryptAES(string cipherText, string key, string iv)
    {
        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Encoding.UTF8.GetBytes(key);
            aesAlg.IV = Encoding.UTF8.GetBytes(iv);
            ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

            using (MemoryStream msDecrypt = new MemoryStream(Convert.FromBase64String(cipherText)))
            {
                using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
                {
                    using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                    {
                        return srDecrypt.ReadToEnd();
                    }
                }
            }
        }
    }
}
```

Then, compile it into an AOT standalone binary:

```
csaw-generatepdb    Program.cs
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net8.0</TargetFramework>
        <PublishAot>true</PublishAot>
        <RuntimeIdentifier>win-x64</RuntimeIdentifier>
    </PropertyGroup>

</Project>
```

```
FLARE-VM 11/04/2024 21:24:46
PS C:\Users\JiaYuChan\source\repos\csaw-generatepdb > dotnet publish -c Release -r win-x64 --self-contained true /p:PublishAot=true
>>
  Determining projects to restore...
  All projects are up-to-date for restore.
```

This won't cover everything, but sufficient to move forward.

Then, load both the resulting EXE and PDB file into IDA, and once analysis finishes, run the idb2pat.py ([https://github.com/mandiant/flare-ida/blob/master/python/flare/idb2pat.py](https://github.com/mandiant/flare-ida/blob/master/python/flare/idb2pat.py)) script in IDA to generate a PAT file. Then run sigmake on the PAT file and resolve any signature collisions (I am not sure if IDA free comes packaged with flair83 utils, might need IDA Pro for that). Workflow will be different with Ghidra and BN. For those who says its pay2win, well, either pirate or drop the chal mate

After applying the signature file, we can start to see some AES functions. This particular subroutine (file offset 0x71710) takes in the user input and encrypts it using AES.

```
11  __int64 v10; // rdi
12  __int64 v11; // rbx
13
14  v2 = RhpNewFast(&unk_1401359D0);
15  System_Security_Cryptography_System_Security_Cryptography_Aes___ctor(v2);
16  if ( *(&qword_140126960 - 1) )
17    sub_1400017AC();
18  v3 = qword_1401E6678;
19  Bytes = S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes(
20            *(_QWORD *)(qword_1401E6678 + 8),
21            &unk_14011FFE8);
22  (*(void (__fastcall **)(__int64, __int64))(*(_QWORD *)v2 + 88LL))(v2, Bytes);
23  v5 = S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes(*(_QWORD *)(v3 + 8), &unk_1401159D0);
24  (*(void (__fastcall **)(__int64, __int64))(*(_QWORD *)v2 + 72LL))(v2, v5);
25  v6 = (*(__int64 (__fastcall **)(__int64))(*(_QWORD *)v2 + 80LL))(v2);
26  v7 = (*(__int64 (__fastcall **)(__int64))(*(_QWORD *)v2 + 64LL))(v2);
27  LODWORD(v6) = System_Security_Cryptography_System_Security_Cryptography_AesImplementation__CreateEncryptor_0(
28                  v2,
29                  v6,
30                  v7);
31  v8 = RhpNewFast(&unk_14012F898);
32  S_P_CoreLib_System_IO_MemoryStream___ctor_0(v8, 0LL);
33  v9 = RhpNewFast(&unk_140135B80);
34  System_Security_Cryptography_System_Security_Cryptography_CryptoStream___ctor_0(v9, v8, v6, 1, 0);
35  v10 = RhpNewFast(&unk_14012FBB0);
36  if ( *(&qword_140126998 - 1) )
37    sub_14000180C();
38  S_P_CoreLib_System_IO_StreamWriter___ctor_2(v10, v9, *(_QWORD *)(qword_1401E6690 + 8), 1024, 0);
39  S_P_CoreLib_System_IO_StreamWriter__Write_3(v10, a1);
40  if ( (*(_DWORD *)(*(_QWORD *)(v10 + 72) + 52LL) & 0x1600000) == 0 )
41    sub_1400BB260();
42  sub_1400BB5A0(v10, 1LL, 1LL);
43  unk_140128110(v10);
44  unk_140128110(v9);
45  v11 = S_P_CoreLib_System_IO_MemoryStream__ToArray(v8);
46  unk_140128110(v8);
47  unk_140128110(v2);
48  return v11;
49 }
```

It's not obvious but when you think about AES, it uses a key and an IV to create an Encryptor object. The hardcoded key and IV are in the .hydrated section of the program (AOT feature), and will only be populated during runtime. So, running the program and breaking in the middle of the program, when we inspect the second argument to the UTF-8 GetBytes functions, we can see the Key and the IV in memory respectively.

Key:

```
hydrated:00007FF73F82FFF3                 db    0
hydrated:00007FF73F82FFF4                 db   54h ; T
hydrated:00007FF73F82FFF5                 db    0
hydrated:00007FF73F82FFF6                 db   68h ; h
hydrated:00007FF73F82FFF7                 db    0
hydrated:00007FF73F82FFF8                 db   65h ; e
hydrated:00007FF73F82FFF9                 db    0
hydrated:00007FF73F82FFFA                 db   57h ; W
hydrated:00007FF73F82FFFB                 db    0
hydrated:00007FF73F82FFFC                 db   6Fh ; o
hydrated:00007FF73F82FFFD                 db    0
hydrated:00007FF73F82FFFE                 db   72h ; r
hydrated:00007FF73F82FFFF                 db    0
hydrated:00007FF73F830000                 db   6Ch ; l
hydrated:00007FF73F830001                 db    0
hydrated:00007FF73F830002                 db   64h ; d
hydrated:00007FF73F830003                 db    0
hydrated:00007FF73F830004                 db   49h ; I
hydrated:00007FF73F830005                 db    0
hydrated:00007FF73F830006                 db   73h ; s
hydrated:00007FF73F830007                 db    0
hydrated:00007FF73F830008                 db   43h ; C
hydrated:00007FF73F830009                 db    0
hydrated:00007FF73F83000A                 db   72h ; r
hydrated:00007FF73F83000B                 db    0
hydrated:00007FF73F83000C                 db   75h ; u
hydrated:00007FF73F83000D                 db    0
hydrated:00007FF73F83000E                 db   65h ; e
hydrated:00007FF73F83000F                 db    0
hydrated:00007FF73F830010                 db   6Ch ; l
hydrated:00007FF73F830011                 db    0
hydrated:00007FF73F830012                 db   41h ; A
hydrated:00007FF73F830013                 db    0
hydrated:00007FF73F830014                 db   6Eh ; n
hydrated:00007FF73F830015                 db    0
hydrated:00007FF73F830016                 db   64h ; d
hydrated:00007FF73F830017                 db    0
hydrated:00007FF73F830018                 db   4Dh ; M
hydrated:00007FF73F830019                 db    0
hydrated:00007FF73F83001A                 db   65h ; e
hydrated:00007FF73F83001B                 db    0
```

```
__int64 v3; // rdi
__int64 Bytes; // rax
__int64 v5; // rax
__int64 v6; // rdi
__int64 v7; // rax
_QWORD *v8; // r14
_QWORD *v9; // r15
_QWORD *v10; // rdi
__int64 v11; // rbx

v2 = RhpNewFast((__int64)&unk_7FF73F8459D0);
System_Security_Cryptography_System_Security_Cryptography_Aes___ctor(v2);
if ( *(&qword_7FF73F836960 - 1) )
  sub_7FF73F7117AC();
v3 = qword_7FF73F8F6678;
Bytes = S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes(
          *(_QWORD *)(qword_7FF73F8F6678 + 8),
          &off_7FF73F82FFE8);
(*(void (__fastcall **)(_QWORD *, __int64))(*v2 + 88LL))(v2, Bytes);
v5 = S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes(*(_QWORD *)(v3 + 8), &off_7FF73F8259D0);
(*(void (__fastcall **)(_QWORD *, __int64))(*v2 + 72LL))(v2, v5);
v6 = (*(__int64 (__fastcall **)(_QWORD *))(*v2 + 80LL))(v2);
v7 = (*(__int64 (__fastcall **)(_QWORD *))(*v2 + 64LL))(v2);
LODWORD(v6) = System_Security_Cryptography_System_Security_Cryptography_AesImplementation__CreateEncryptor_0(
                v2,
                v6,
                v7);
v8 = RhpNewFast((__int64)&unk_7FF73F83F898);
S_P_CoreLib_System_IO_MemoryStream___ctor_0(v8, 0LL);
v9 = RhpNewFast((__int64)&unk_7FF73F845BB0);
System_Security_Cryptography_System_Security_Cryptography_CryptoStream___ctor_0((_DWORD)v9, (_DWORD)v8, v6, 1, (
v10 = RhpNewFast((__int64)&unk_7FF73F83FBB0);
if ( *(&qword_7FF73F836998 - 1) )
  sub_7FF73F71180C();
S_P_CoreLib_System_IO_StreamWriter___ctor_2((_DWORD)v10, (_DWORD)v9, *(_QWORD *)(qword_7FF73F8F6690 + 8), 1024,
S_P_CoreLib_System_IO_StreamWriter__Write_3(v10, a1);
if ( (*(_DWORD *)(v10[9] + 52LL) & 0x1600000) == 0 )
  sub_7FF73F7CB260();
sub_7FF73F7CB5A0((__int64)v10, 1u, 1u);
unk_7FF73F838110(v10);
unk_7FF73F838110(v9);
```

IV:

```
IDA View-RIP
hydrated:00007FF73F8259CF                 db    0
hydrated:00007FF73F8259D0 off_7FF73F8259D0 dq offset unk_7FF
hydrated:00007FF73F8259D0
hydrated:00007FF73F8259D8                 db   10h
hydrated:00007FF73F8259D9                 db    0
hydrated:00007FF73F8259DA                 db    0
hydrated:00007FF73F8259DB                 db    0
hydrated:00007FF73F8259DC                 db   42h ; B
hydrated:00007FF73F8259DD                 db    0
hydrated:00007FF73F8259DE                 db   75h ; u
hydrated:00007FF73F8259DF                 db    0
hydrated:00007FF73F8259E0                 db   74h ; t
hydrated:00007FF73F8259E1                 db    0
hydrated:00007FF73F8259E2                 db   41h ; A
hydrated:00007FF73F8259E3                 db    0
hydrated:00007FF73F8259E4                 db   6Ch ; l
hydrated:00007FF73F8259E5                 db    0
hydrated:00007FF73F8259E6                 db   73h ; s
hydrated:00007FF73F8259E7                 db    0
hydrated:00007FF73F8259E8                 db   6Fh ; o
hydrated:00007FF73F8259E9                 db    0
hydrated:00007FF73F8259EA                 db   42h ; B
hydrated:00007FF73F8259EB                 db    0
hydrated:00007FF73F8259EC                 db   65h ; e
hydrated:00007FF73F8259ED                 db    0
hydrated:00007FF73F8259EE                 db   61h ; a
hydrated:00007FF73F8259EF                 db    0
hydrated:00007FF73F8259F0                 db   75h ; u
hydrated:00007FF73F8259F1                 db    0
hydrated:00007FF73F8259F2                 db   74h ; t
hydrated:00007FF73F8259F3                 db    0
hydrated:00007FF73F8259F4                 db   69h ; i
hydrated:00007FF73F8259F5                 db    0
hydrated:00007FF73F8259F6                 db   66h ; f
hydrated:00007FF73F8259F7                 db    0
hydrated:00007FF73F8259F8                 db   75h ; u
hydrated:00007FF73F8259F9                 db    0
hydrated:00007FF73F8259FA                 db   6Ch ; l
hydrated:00007FF73F8259FB                 db    0
hydrated:00007FF73F8259FC                 db    0
hydrated:00007FF73F8259FD                 db    0
UNKNOWN 00007FF73F8259D9: hydrated:00007FF7 (Synchronized with Pseudocode-A)
```

```
Pseudocode-A
__int64 v3; // rdi
__int64 Bytes; // rax
__int64 v5; // rax
__int64 v6; // rdi
__int64 v7; // rax
_QWORD *v8; // r14
_QWORD *v9; // r15
_QWORD *v10; // rdi
__int64 v11; // rbx

v2 = RhpNewFast((__int64)&unk_7FF73F8459D0);
System_Security_Cryptography_System_Security_Cryptography_Aes___ctor(v2);
if ( *(&qword_7FF73F836960 - 1) )
  sub_7FF73F7117AC();
v3 = qword_7FF73F8F6678;
Bytes = S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes(
          *(_QWORD *)(qword_7FF73F8F6678 + 8),
          &off_7FF73F82FFE8);
(*(void (__fastcall **)(_QWORD *, __int64))(*v2 + 88LL))(v2, Bytes);
v5 = S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes(*(_QWORD *)(v3 + 8), &off_7FF73F8259D0);
(*(void (__fastcall **)(_QWORD *, __int64))(*v2 + 72LL))(v2, v5);
v6 = (*(__int64 (__fastcall **)(_QWORD *))(*v2 + 80LL))(v2);
v7 = (*(__int64 (__fastcall **)(_QWORD *))(*v2 + 64LL))(v2);
LODWORD(v6) = System_Security_Cryptography_System_Security_Cryptography_AesImplementation__CreateEncryptor_0(
                v2,
                v6,
                v7);
v8 = RhpNewFast((__int64)&unk_7FF73F83F898);
S_P_CoreLib_System_IO_MemoryStream___ctor_0(v8, 0LL);
v9 = RhpNewFast((__int64)&unk_7FF73F845BB0);
System_Security_Cryptography_System_Security_Cryptography_CryptoStream___ctor_0((_DWORD)v9, (_DWORD)v8, v6, 1, 0
v10 = RhpNewFast((__int64)&unk_7FF73F83FBB0);
if ( *(&qword_7FF73F836998 - 1) )
  sub_7FF73F71180C();
S_P_CoreLib_System_IO_StreamWriter___ctor_2((_DWORD)v10, (_DWORD)v9, *(_QWORD *)(qword_7FF73F8F6690 + 8), 1024,
S_P_CoreLib_System_IO_StreamWriter__Write_3(v10, a1);
if ( (*(_DWORD *)(v10[9] + 52LL) & 0x1600000) == 0 )
  sub_7FF73F7CB260();
sub_7FF73F7CB5A0((__int64)v10, 1u, 1u);
unk_7FF73F838110(v10);
unk_7FF73F838110(v9);
0007177A sub_7FF73F783110:23 (7FF73F78317A) (Synchronized with IDA View-RIP)
```

We can confirm by stepping into the call at [rax+58h] and [rax+48h], which are setter functions for the key and IV.

```
.managed:00007FF73F783158                mov     rcx, [rdi+8]
.managed:00007FF73F78315C                lea     rdx, off_7FF73F82FFE8
.managed:00007FF73F783163                cmp     [rcx], ecx
.managed:00007FF73F783165                call    S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes
.managed:00007FF73F78316A                mov     rdx, rax
.managed:00007FF73F78316D                mov     rcx, rsi
.managed:00007FF73F783170                mov     rax, [rsi]
.managed:00007FF73F783173                call    qword ptr [rax+58h]
.managed:00007FF73F783176                mov     rcx, [rdi+8]
.managed:00007FF73F78317A                lea     rdx, off_7FF73F8259D0
.managed:00007FF73F783181                cmp     [rcx], ecx
.managed:00007FF73F783183                call    S_P_CoreLib_System_Text_UTF8Encoding_UTF8EncodingSealed__GetBytes
.managed:00007FF73F783188                mov     rdx, rax
.managed:00007FF73F78318B                mov     rcx, rsi
.managed:00007FF73F78318E                mov     rax, [rsi]
.managed:00007FF73F783191                call    qword ptr [rax+48h]
.managed:00007FF73F783194                mov     rcx, rsi
.managed:00007FF73F783197                mov     rax, [rsi]
.managed:00007FF73F78319A                call    qword ptr [rax+50h]
```

```
void __fastcall System_Security_Cryptography_System_Security_Cryptography_SymmetricAlgorithm__set_Key(
        unsigned __int64 *a1,
        __int64 a2)
{
  __int64 v4; // rdi
  unsigned __int64 v5; // rbp
  __int64 v6; // rax
  unsigned __int64 v7; // rax
  _QWORD *v8; // rbx
  char v9[40]; // [rsp+20h] [rbp-28h] BYREF

  if ( !a2 )
    sub_7FF73F790F70(&off_7FF73F834108);
  v4 = 8LL * *(unsigned int *)(a2 + 8);
  if ( v4 > 0x7FFFFFFF
    || (v5 = *a1, (v6 = (*(__int64 (__fastcall **)(unsigned __int64 *))(*a1 + 112))(a1)) == 0)
    || !(unsigned int)System_Security_Cryptography_System_Security_Cryptography_KeySizeHelpers__IsLegalSize_2(
                        (unsigned int)v4,
                        v6,
                        v9) )
  {
    v8 = RhpNewFast((__int64)&unk_7FF73F83C0B8);
    S_P_CoreLib_System_IO_InvalidDataException___ctor_0(v8, &off_7FF73F82BFF0);
    RhpThrowEx(v8);
  }
  (*(void (__fastcall **)(unsigned __int64 *, _QWORD))(v5 + 104))(a1, (unsigned int)v4);
  v7 = System_Security_Cryptography_Internal_Cryptography_Helpers__CloneByteArray(a2);
  RhpAssignRefAVLocation(a1 + 1, v7);
}
```

```
void __fastcall System_Security_Cryptography_System_Security_Cryptography_SymmetricAlgorithm__set_IV(
        unsigned __int64 *a1,
        __int64 a2)
{
  int v4; // edi
  unsigned __int64 v5; // rax
  _QWORD *v6; // rbx

  if ( !a2 )
    sub_7FF73F790F70(&off_7FF73F834108);
  v4 = *(_DWORD *)(a2 + 8);
  if ( v4 != (*(int (__fastcall **)(unsigned __int64 *))(*a1 + 56))(a1) / 8 )
  {
    v6 = RhpNewFast((__int64)&unk_7FF73F83C0B8);
    S_P_CoreLib_System_IO_InvalidDataException___ctor_0(v6, &off_7FF73F82BF28);
    RhpThrowEx(v6);
  }
  v5 = System_Security_Cryptography_Internal_Cryptography_Helpers__CloneByteArray(a2);
  RhpAssignRefAVLocation(a1 + 2, v5);
}
```

After the encryption, the output looks like it's being XORed with v2 incrementally.

```
 9  v1 = sub_7FF73F783110(a1);
10  if ( *(&qword_7FF73F836920 - 1) )
11    sub_7FF73F7116AC();
12  v2 = (*(__int64 (__fastcall **)(_QWORD, void **))(**(_QWORD **)(qword_7FF73F8F6618 + 8) + 120LL))(
13         *(_QWORD *)(qword_7FF73F8F6618 + 8),
14         &off_7FF73F8257F0);
15  v3 = 0;
16  v4 = *(_DWORD *)(v1 + 8);
17  if ( v4 > 0 )
18  {
19    do
20    {
21      v5 = *(_DWORD *)(v2 + 8);
22      if ( v3 % v5 >= (unsigned int)v5 )
23        sub_7FF73F7DC580();
24      *(_BYTE *)(v1 + (unsigned int)v3 + 16) ^= *(_BYTE *)(v2 + (unsigned int)(v3 % v5) + 16);
25      ++v3;
26    }
27    while ( v4 > v3 );
28  }
29  return v1;
30 }
```

If you step into the function which stores its return value into v2, you will notice that it is dealing with some sort of ASCII encoding. And by inspecting the second argument to the call, we can see the XOR key "AttackOnTitan" is populated in the hydrated section in memory.

```
IDA View-RIP
hydrated:00007FF73F8257EC                 db    0
hydrated:00007FF73F8257ED                 db    0
hydrated:00007FF73F8257EE                 db    0
hydrated:00007FF73F8257EF                 db    0
hydrated:00007FF73F8257F0 off_7FF73F8257F0 dq offset unk_7FF73F8394
hydrated:00007FF73F8257F0                 ;
hydrated:00007FF73F8257F8                 db  0Dh
hydrated:00007FF73F8257F9                 db    0
hydrated:00007FF73F8257FA                 db    0
hydrated:00007FF73F8257FB                 db    0
hydrated:00007FF73F8257FC                 db  41h ; A
hydrated:00007FF73F8257FD                 db    0
hydrated:00007FF73F8257FE                 db  74h ; t
hydrated:00007FF73F8257FF                 db    0
hydrated:00007FF73F825800                 db  74h ; t
hydrated:00007FF73F825801                 db    0
hydrated:00007FF73F825802                 db  61h ; a
hydrated:00007FF73F825803                 db    0
hydrated:00007FF73F825804                 db  63h ; c
hydrated:00007FF73F825805                 db    0
hydrated:00007FF73F825806                 db  6Bh ; k
hydrated:00007FF73F825807                 db    0
hydrated:00007FF73F825808                 db  4Fh ; O
hydrated:00007FF73F825809                 db    0
hydrated:00007FF73F82580A                 db  6Eh ; n
hydrated:00007FF73F82580B                 db    0
hydrated:00007FF73F82580C                 db  54h ; T
hydrated:00007FF73F82580D                 db    0
hydrated:00007FF73F82580E                 db  69h ; i
hydrated:00007FF73F82580F                 db    0
hydrated:00007FF73F825810                 db  74h ; t
hydrated:00007FF73F825811                 db    0
hydrated:00007FF73F825812                 db  61h ; a
hydrated:00007FF73F825813                 db    0
hydrated:00007FF73F825814                 db  6Eh ; n
hydrated:00007FF73F825815                 db    0
hydrated:00007FF73F825816                 db    0
hydrated:00007FF73F825817                 db    0
hydrated:00007FF73F825818                 db    0
hydrated:00007FF73F825819                 db    0
hydrated:00007FF73F82581A                 db    0
```

```
Pseudocode-A
 1 __int64 __fastcall sub_7FF73F783080(__int64 a1)
 2 {
 3   __int64 v1; // rbx
 4   __int64 v2; // rcx
 5   int v3; // r8d
 6   int v4; // r10d
 7   signed int v5; // esi
 8
 9   v1 = sub_7FF73F783110(a1);
10   if ( *(&qword_7FF73F836920 - 1) )
11     sub_7FF73F7116AC();
12   v2 = (*(__int64 (__fastcall **)(_QWORD, void **))(**(_QWORD **)(qword_7FF73F8F6618 + 8) + 120LL))(
13          *(_QWORD *)(qword_7FF73F8F6618 + 8),
14          &off_7FF73F8257F0);
15   v3 = 0;
16   v4 = *(_DWORD *)(v1 + 8);
17   if ( v4 > 0 )
18   {
19     do
20     {
21       v5 = *(_DWORD *)(v2 + 8);
22       if ( v3 % v5 >= (unsigned int)v5 )
23         sub_7FF73F7DC580();
24       *(_BYTE *)(v1 + (unsigned int)v3 + 16) ^= *(_BYTE *)(v2 + (unsigned int)(v3 % v5) + 16);
25       ++v3;
26     }
27     while ( v4 > v3 );
28   }
29   return v1;
30 }
```

After the encrypted user input is XORed, it's returned to its calling function.
Where it does a final comparison with the encrypted flag. (encrypted user input at [rax+r10+16], encrypted flag at [r11+r10+16]).

```
.managed:00007FF73F7833C2                 retn
.managed:00007FF73F7833C3 ; ---------------------------------------------------------------
.managed:00007FF73F7833C3
.managed:00007FF73F7833C3 loc_7FF73F7833C3:                       ; CODE XREF: sub_7F
.managed:00007FF73F7833C3                 xor     r8d, r8d
.managed:00007FF73F7833C6                 test    ecx, ecx
.managed:00007FF73F7833C8                 jle     short loc_7FF73F7833F0
.managed:00007FF73F7833CA                 mov     r10d, [rdx+8]
.managed:00007FF73F7833CE
.managed:00007FF73F7833CE loc_7FF73F7833CE:                       ; CODE XREF: sub_7F
.managed:00007FF73F7833CE                 mov     r10d, r8d
.managed:00007FF73F7833D1                 movzx   r9d, byte ptr [rax+r10+10h]
.managed:00007FF73F7833D7                 mov     r11, [rdx+8]
.managed:00007FF73F7833DB                 cmp     r8d, 50h ; 'P'
.managed:00007FF73F7833DF                 jnb     short loc_7FF73F783401
.managed:00007FF73F7833E1                 cmp     r9b, [r11+r10+10h]
.managed:00007FF73F7833E6                 jnz     short loc_7FF73F7833FA
.managed:00007FF73F7833E8                 inc     r8d
.managed:00007FF73F7833EB                 cmp     ecx, r8d
.managed:00007FF73F7833EE                 jg      short loc_7FF73F7833CE
```

```
__int64 __fastcall sub_7FF73F7833A0(__int64 a1)
{
  __int64 v1; // rax
  unsigned int v3; // r8d

  v1 = sub_7FF73F783080(a1);
  if ( *(_DWORD *)(v1 + 8) != 80 )
    return 0LL;
  v3 = 0;
  while ( 1 )
  {
    if ( v3 >= 80 )
      sub_7FF73F7DC580();
    if ( *(_BYTE *)(v1 + v3 + 16) != *(_BYTE *)(*(_QWORD *)(qword_7FF73F8F6488 + 8) + v3 + 16LL) )
      break;
    if ( (int)++v3 >= 80 )
      return 1LL;
  }
  return 0LL;
}
```

Encrypted flag in memory

```
hydrated:00007FF73F835DF0                 db  9Bh
hydrated:00007FF73F835DF1                 db  36h ; 6
hydrated:00007FF73F835DF2                 db 0C0h
hydrated:00007FF73F835DF3                 db 0C3h
hydrated:00007FF73F835DF4                 db  0Ah
hydrated:00007FF73F835DF5                 db  8Fh
hydrated:00007FF73F835DF6                 db 0C1h
hydrated:00007FF73F835DF7                 db 0B5h
hydrated:00007FF73F835DF8                 db 0C1h
hydrated:00007FF73F835DF9                 db 0A8h
hydrated:00007FF73F835DFA                 db    0
hydrated:00007FF73F835DFB                 db 0DCh
hydrated:00007FF73F835DFC                 db  75h ; u
hydrated:00007FF73F835DFD                 db    4
hydrated:00007FF73F835DFE                 db 0B2h
hydrated:00007FF73F835DFF                 db  6Ah ; j
hydrated:00007FF73F835E00                 db  9Ch
hydrated:00007FF73F835E01                 db  4Ch ; L
hydrated:00007FF73F835E02                 db  40h ; @
hydrated:00007FF73F835E03                 db  51h ; Q
hydrated:00007FF73F835E04                 db  2Ah ; *
hydrated:00007FF73F835E05                 db    5
hydrated:00007FF73F835E06                 db 0EEh
hydrated:00007FF73F835E07                 db  32h ; 2
hydrated:00007FF73F835E08                 db 0CAh
hydrated:00007FF73F835E09                 db 0C6h
hydrated:00007FF73F835E0A                 db 0DCh
hydrated:00007FF73F835E0B                 db 0E9h
hydrated:00007FF73F835E0C                 db  0Eh
hydrated:00007FF73F835E0D                 db 0E7h
hydrated:00007FF73F835E0E                 db 0F5h
hydrated:00007FF73F835E0F                 db  95h
hydrated:00007FF73F835E10                 db 0B1h
hydrated:00007FF73F835E11                 db  5Dh ; ]
hydrated:00007FF73F835E12                 db  5Ch ; \
hydrated:00007FF73F835E13                 db  6Fh ; o
hydrated:00007FF73F835E14                 db 0DFh
hydrated:00007FF73F835E15                 db  8Eh
hydrated:00007FF73F835E16                 db 0BFh
hydrated:00007FF73F835E17                 db  10h
```