VILNIUS UNIVERSITY
ŠIAULIAI ACADEMY
BACHELOR PROGRAMME SOFTWARE ENGINEERING
Study Year 2023-2024

Object Oriented Programing Final project

Student: Mohamed Kenawi
Lecturer: Dr. Donatas Dervinis

Task Question:

Create an application for password reset using brute force attack and multi-threaded method.

Task:

1. The application must have a WPF graphical interface;
2. A separate part of the program must have its own class, and the action must have its own method.
3. Each class must stored in the separate file;

4. The program must have the following options:

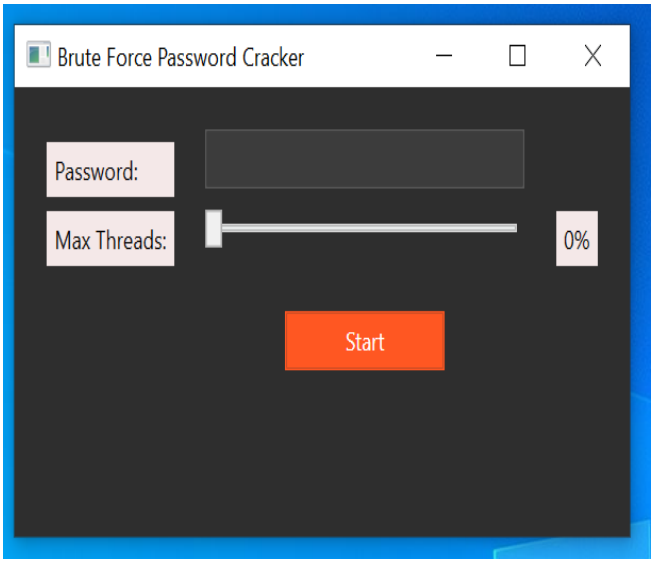
a. create a password and encrypt it in the chosen way - with a static (constant) *salt* ;
b. store the encrypted password in file or in a database (optional);
c. by generating all possible values (using "brute force" attack: i.e. a, b,c,... aa, ab, ... , aaa, ...) decrypt information and to found primary password;
d. measure the time the task was performed;
e. show results in a graphical interface.
5. The program is implemented using multithreading (optional, +1 point).
6. Possibilities to select the maximum number of threads you want to use (optional, +1 point);

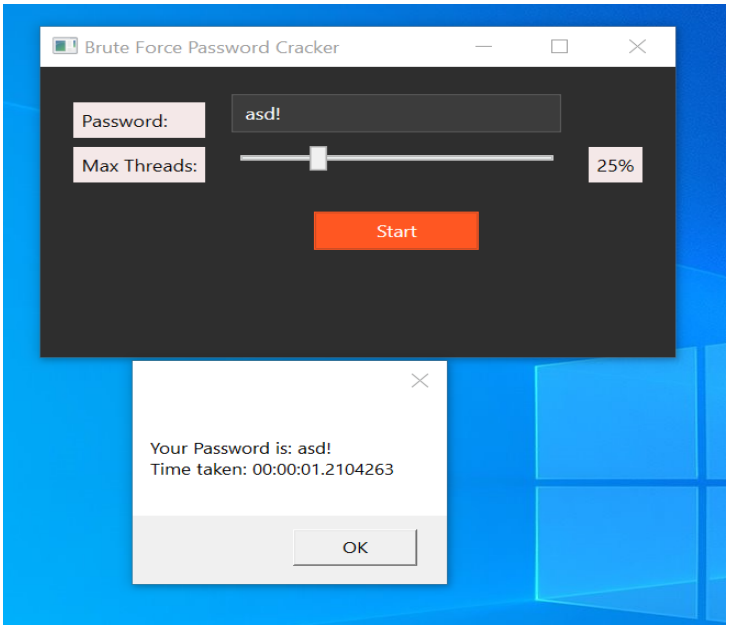
7. Submit in the job testing report

a. UML class diagrams,
b. results of testing, program operation
c. the project is also uploaded to GitHUB (public).
d. 2 files are uploaded to Moodle: program code and compiled - bin directory (in one zip), report (PDF).

Overview:

This program is a simple password brute-forcing application with a graphical user interface (GUI) built using WPF (Windows Presentation Foundation). It encrypts a user-provided password and then attempts to brute-force it using multiple threads to demonstrate the process.





Main Components:

1. **MainWindow.xaml.cs: Handles the GUI and user interactions.**
2. **Encryption.cs: Contains methods for encrypting and decrypting passwords.**
3. **BruteForceAlgorthiem.cs: Contains the brute force logic to crack the password.**

MainWindow.xaml.cs:

This file sets up the main window of the application, including a slider for selecting the number of threads and a button to start the brute force attack.

Key Elements

- InitializeComponent(): Initializes the components of the window.
- ThreadSlider_ValueChanged: Updates the label to show the percentage of threads being used based on slider value.
- Button_Click: Handles the button click event to start the brute force attack.

MainWindow.xaml.cs code:

```
using System;
using System.Text;
using System.Windows;
using System.Windows.Controls;

namespace BruteForcePassword
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            ThreadSlider.ValueChanged += ThreadSlider_ValueChanged;
        }

        private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
        {
        }

        byte[] key, IV, encryptedPass;

        private void ThreadSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            switch ((int)ThreadSlider.Value)
            {
                case 0:
                    ThreadLabel.Content = "0%";
                    break;
                case 1:
```

```
                    ThreadLabel.Content = "25%";
                    break;
                case 2:
                    ThreadLabel.Content = "50%";
                    break;
                case 3:
                    ThreadLabel.Content = "75%";
                    break;
                case 4:
                    ThreadLabel.Content = "100%";
                    break;
            }
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            string content = Content.Text;
            int maxThreads = CalculateMaxThreads((int)ThreadSlider.Value);
            if (maxThreads <= 0)
            {
                MessageBox.Show("Please select a valid number of threads.");
                return;
            }

            Password pass = new Password();
            pass.password = content;
            (key, IV, encryptedPass) = pass.encryptPassword(content);
            string result = BruteForceAttack.GetResult(encryptedPass, key, IV,
maxThreads);
            MessageBox.Show("Your Password is: " + result);
        }

        private int CalculateMaxThreads(int sliderValue)
        {
            int totalThreads = Environment.ProcessorCount;
            switch (sliderValue)
            {
                case 0:
                    return 0;
                case 1:
                    return totalThreads / 4;
                case 2:
                    return totalThreads / 2;
                case 3:
                    return (totalThreads * 3) / 4;
                case 4:
                    return totalThreads;
                default:
                    return 0;
            }
        }
    }
}
```

Encryption.cs:
This file handles encryption and decryption of the password using AES (Advanced Encryption Standard).

Key Elements

- encryptPassword: Encrypts the password and returns the key, IV, and encrypted password.
- EncryptStringToBytes_Aes: Helper method to encrypt a string using AES.
- DecryptStringFromBytes_Aes: Decrypts an encrypted byte array back to the original string.

Encryption.cs code:

```
public class Password
{
    public string? password;

    public (byte[] key, byte[] IV, byte[] encrypted) encryptPassword(string password)
    {
        byte[] key, IV, encrypted;
        using (Aes myAes = Aes.Create())
        {
            encrypted = EncryptStringToBytes_Aes(password, myAes.Key, myAes.IV);
            key = myAes.Key;
            IV = myAes.IV;
        }
        return (key, IV, encrypted);
    }

    public byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, byte[] IV)
    {
        if (plainText == null || plainText.Length <= 0)
            throw new ArgumentNullException("plainText");
        if (Key == null || Key.Length <= 0)
            throw new ArgumentNullException("Key");
        if (IV == null || IV.Length <= 0)
            throw new ArgumentNullException("IV");
        byte[] encrypted;

        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Key;
            aesAlg.IV = 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(plainText);
```

```
                    }
                    encrypted = msEncrypt.ToArray();
                }
            }
        }
        return encrypted;
    }

    public string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key, byte[] IV)
    {
        if (cipherText == null || cipherText.Length <= 0)
            throw new ArgumentNullException("cipherText");
        if (Key == null || Key.Length <= 0)
            throw new ArgumentNullException("Key");
        if (IV == null || IV.Length <= 0)
            throw new ArgumentNullException("IV");

        string? plaintext = null;

        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Key;
            aesAlg.IV = IV;
            ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
            using (MemoryStream msDecrypt = new MemoryStream(cipherText))
            {
                using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
                {
                    using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                    {
                        plaintext = srDecrypt.ReadToEnd();
                    }
                }
            }
        }
        return plaintext;
    }
}
```

BruteForceAlgorthiem.cs :

This file contains the logic for brute-forcing the password using multiple threads.

Key Elements

- GetResult: Main method to start the brute force attack and return the result.
- startBruteForce: Divides the task into multiple threads.
- createNewKey: Generates possible passwords and checks if they match the encrypted password.

BruteForce.cs code:

```
public class BruteForceAttack
{
    private string? encryptedPass;

    public static string GetResult(byte[] encryptedPass, byte[] encryptionKey, byte[] IV,
int maxThreads)
    {
        var timeStarted = DateTime.Now;
        charactersToTestLength = charactersToTest.Length;
        var estimatedPasswordLength = 0;

        Password pass = new Password();
        string plainTextPass = pass.DecryptStringFromBytes_Aes(encryptedPass,
encryptionKey, IV);
        while (!isMatched)
        {
            estimatedPasswordLength++;
            startBruteForce(estimatedPasswordLength, encryptedPass, encryptionKey, IV,
plainTextPass, maxThreads);
        }
        var timeEnded = DateTime.Now;
        var timeTaken = timeEnded - timeStarted;
        result += "\nTime taken: " + timeTaken.ToString();
        return result;
    }

    public static string result;

    private static bool isMatched = false;
    private static int charactersToTestLength = 0;
    private static long computedKeys = 0;

    private static char[] charactersToTest =
    {
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
        'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
        'u', 'v', 'w', 'x', 'y', 'z','A','B','C','D','E',
        'F','G','H','I','J','K','L','M','N','O','P','Q','R',
        'S','T','U','V','W','X','Y','Z','1','2','3','4','5',
```

```
        '6','7','8','9','0','!','@','#','$','%','^','&','*',
        '(',')','_','+','-','=','[',']','{','}',',','|',';',
        ':','.','<','>','?','/'
    };

    private static void startBruteForce(int keyLength, byte[] encryptedPass, byte[]
encryptionKey, byte[] IV, string plainTextPass, int maxThreads)
    {
        var threads = new List<Thread>();
        int segmentSize = charactersToTestLength / maxThreads;
        for (int i = 0; i < maxThreads; i++)
        {
            int start = i * segmentSize;
            int end = (i == maxThreads - 1) ? charactersToTestLength : start +
segmentSize;
            var thread = new Thread(() => createNewKey(0, createCharArray(keyLength,
charactersToTest[start]), keyLength, keyLength - 1, plainTextPass, start, end));
            threads.Add(thread);
            thread.Start();
        }

        foreach (var thread in threads)
        {
            thread.Join();
        }
    }

    private static char[] createCharArray(int length, char defaultChar)
    {
        return (from c in new char[length] select defaultChar).ToArray();
    }

    private static void createNewKey(int currentCharPosition, char[] keyChars, int
keyLength, int indexOfLastChar, string plainTextPass, int start, int end)
    {
        var nextCharPosition = currentCharPosition + 1;
        for (int i = start; i < end; i++)
        {
            keyChars[currentCharPosition] = charactersToTest[i];

            if (currentCharPosition < indexOfLastChar)
            {
                createNewKey(nextCharPosition, keyChars, keyLength, indexOfLastChar,
plainTextPass, 0, charactersToTestLength);
            }
            else
            {
                computedKeys++;
                string possiblePass = new string(keyChars);
                if (possiblePass == plainTextPass)
                {
                    if (!isMatched)
                    {
                        isMatched = true;
```

```
                    result = possiblePass;
                }
                return;
            }
        }
    }
}
```

1. **User Interface (MainWindow.xaml.cs)**

   - User inputs a password and selects the number of threads.
   - Upon clicking the button, the password is encrypted.
   - The encrypted password and parameters are passed to the brute force algorithm.

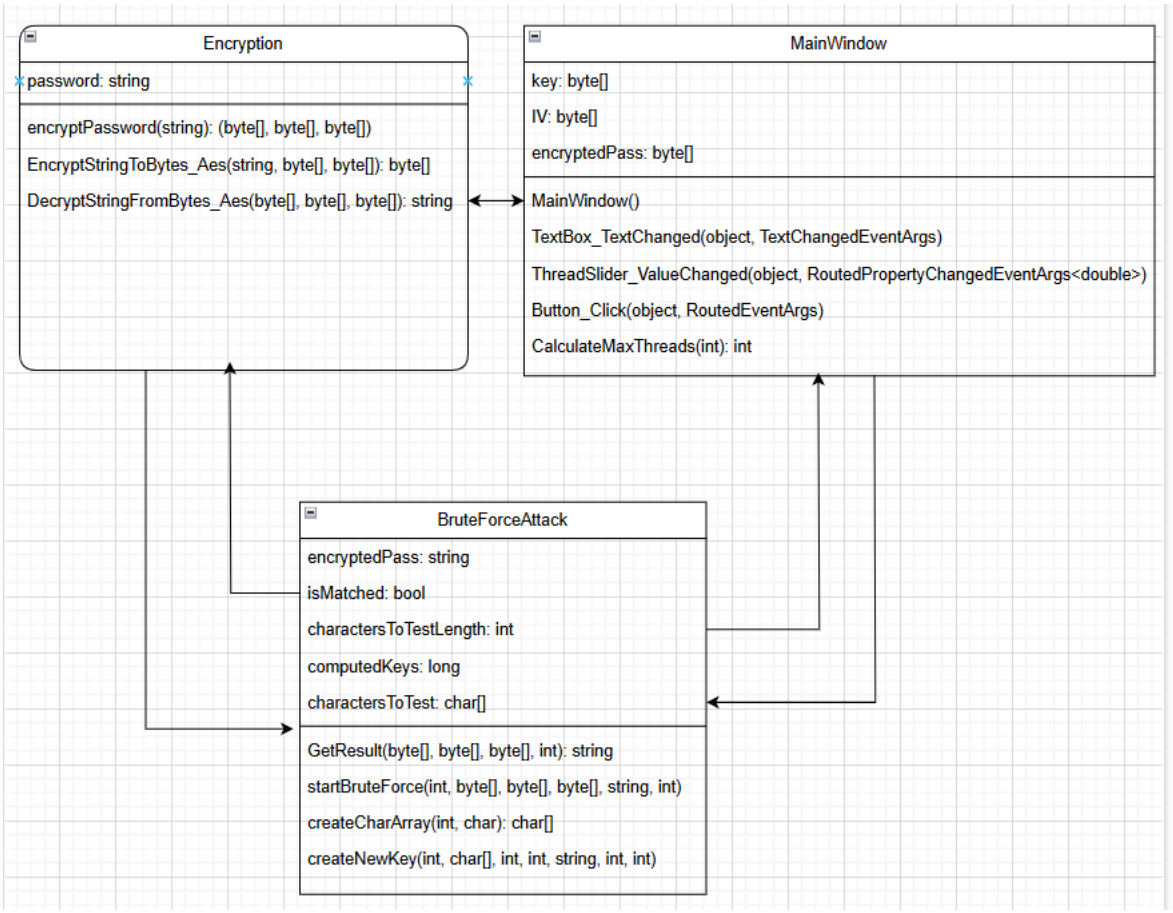
2. **Encryption and Decryption (Encryption.cs)**

   - The Password class handles encryption using AES.
   - encryptPassword encrypts the user input and returns the key, IV, and encrypted password.
   - DecryptStringFromBytes_Aes decrypts the encrypted password for brute force comparison.

3. **Brute Force Algorithm (BruteForce.cs)**

   - GetResult starts the brute force attack and divides the work across multiple threads.
   - startBruteForce creates threads to test different segments of possible passwords.
   - createNewKey recursively generates and tests possible passwords until the correct one is found.

UML Class Diagram:



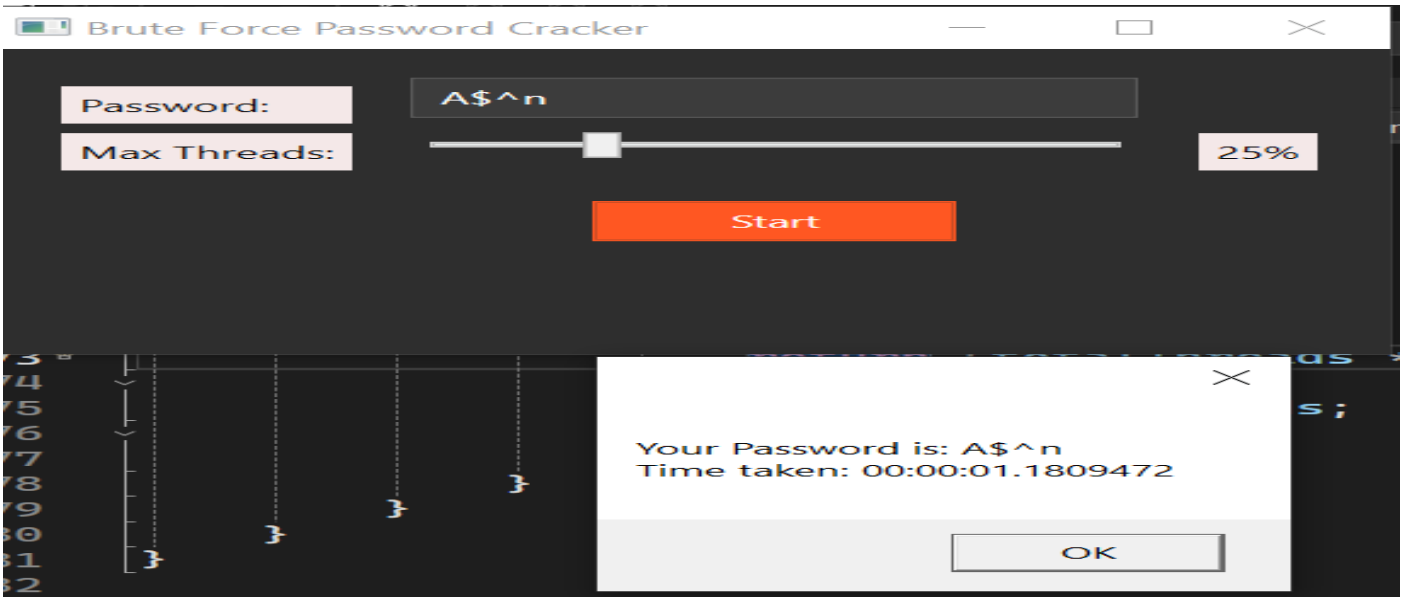
Relationship and Interaction:
The MainWindow class interacts with the Encryption and BruteForceAttack classes.

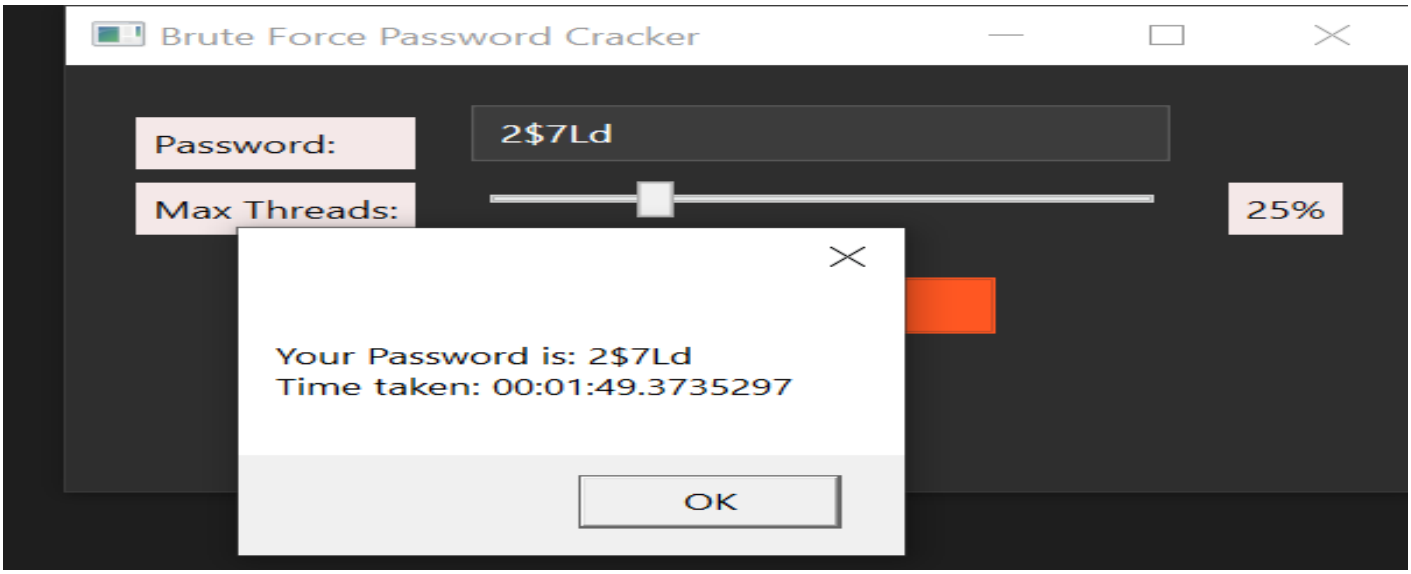
Encryption:
When the button in the MainWindow is clicked (Button_Click method), it creates a Password object (from the Encryption class) and calls encryptPassword to encrypt the user-provided password. The encrypted password, along with the key and IV, is stored in the MainWindow class attributes (key, IV, encryptedPass).

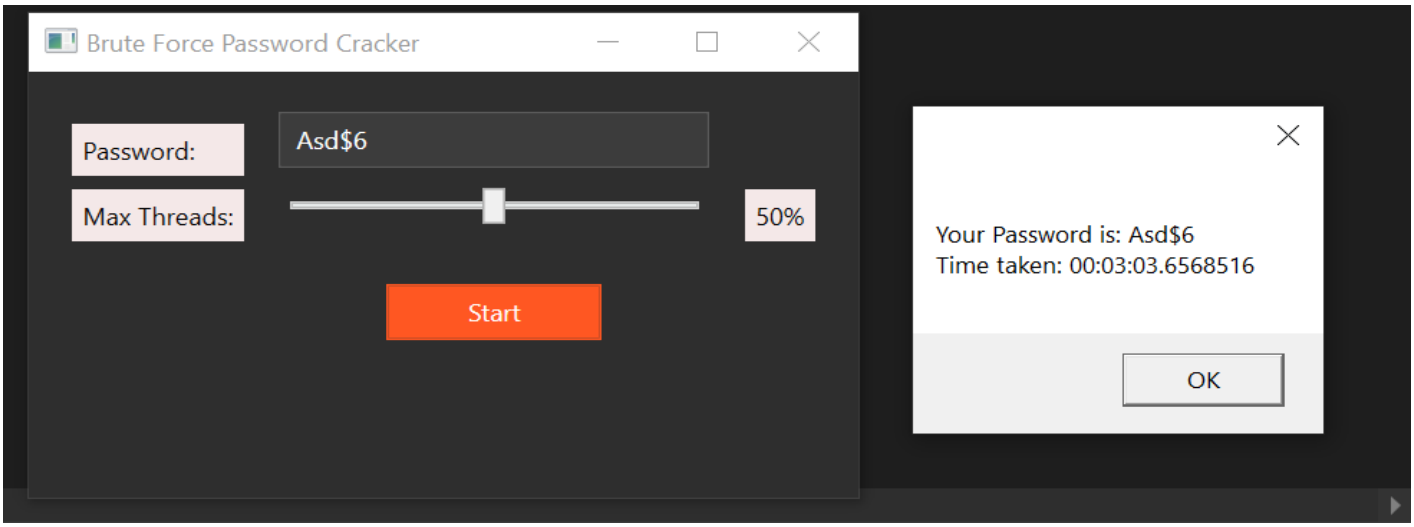
BruteForceAttack:
The Button_Click method is also called BruteForceAttack.GetResult to initiate the brute force attack to find the decrypted password. It passes the encryptedPass, key, IV, and the number of threads to use for the brute force attack.

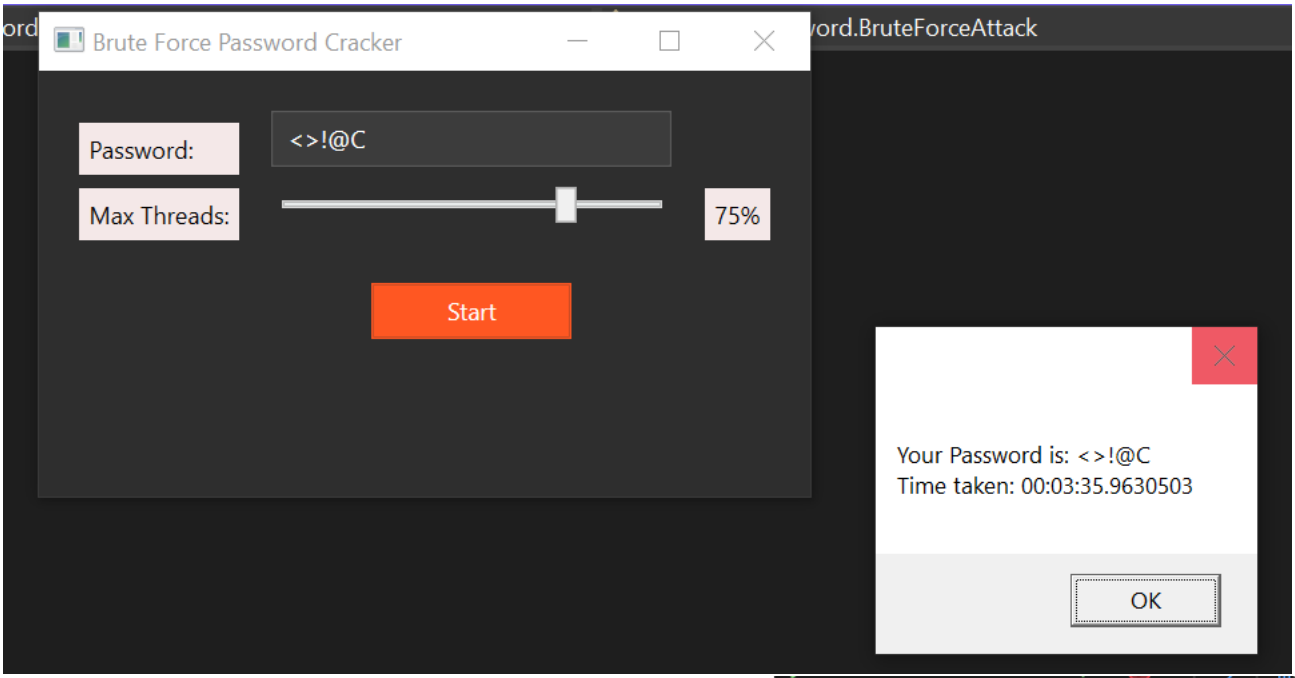
Summary:
The MainWindow class serves as the controller that handles user interactions and initiates encryption and brute force decryption.
The Encryption class is responsible for encrypting and decrypting passwords using AES.
The BruteForceAttack class attempts to decrypt the encrypted password by trying all possible combinations of characters using a multi-threaded brute force approach.
This setup demonstrates a classic example of combining encryption with a brute force attack to test password strength and resilience.

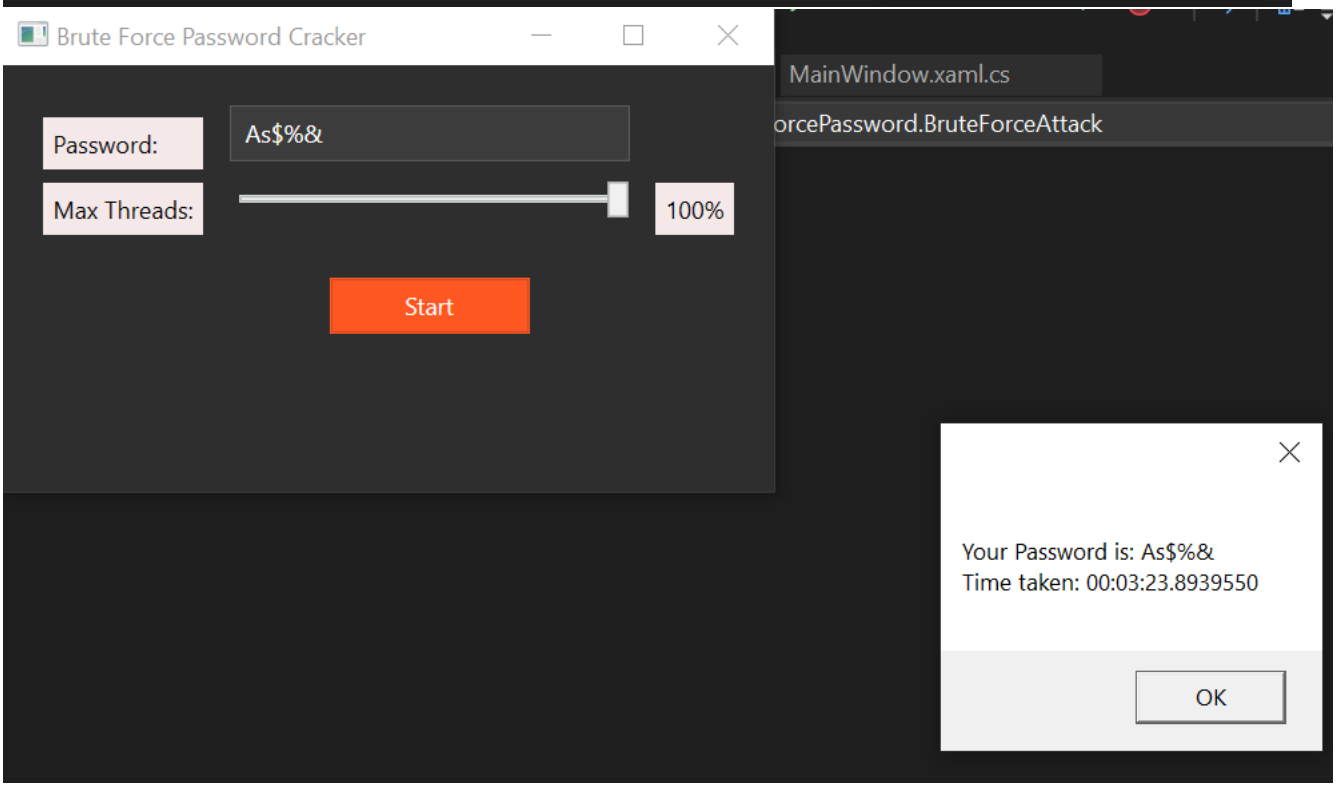
Testing results:








Brute Force Password Cracker
Password:
<>!@C
Max Threads:
75%
Start
ord.BruteForceAttack
Your Password is: <>!@C
Time taken: 00:03:35.9630503
OK


Brute Force Password Cracker
Password:
As$%&
Max Threads:
100%
Start
MainWindow.xaml.cs
orcePassword.BruteForceAttack
Your Password is: As$%&
Time taken: 00:03:23.8939550
OK