Developer Guide for COD3BR3AKR

Encryption/Decryption System

Version 1

December 17, 2018

Diverse Engineers

Course: CS524

Professor: Dimitoglou, George

Computer Science Department

Hood College of Frederick, Maryland 21701

Revision History

Revision	Date	Description	By
1	12/17/2018	Initial Developer Guide for COD3BR3AKR System	Diverse Engineers

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to the instructions to set up the development and build environments for the COD3BR3AKR C# Windows Form Application. As well as, explanation about how to introduce a new algorithm in the near future since this is an Encryption and Decryption System.

1.2. Scope and Applicability

The scope of this document is limited to the COD3BR3AKR Encryption/Decryption System only.

1.3. Related Documents

System Architecture Design Document.

2. Build Matrix

Component	Application Type	Development Tool	Manufacturer	Version
COD3BR3AKR	Windows Form	Visual Studio	Microsoft	1) Visual Studio Community 2017 2) .NET Framework 4.6.1

3. Development Environment Setup

The operating system for the development environment can be either Windows 7 or Windows 10.
Please refer to Visual Studio 2017 Product Family System Requirements to make sure the development PC meets those requirements. If not, please update the development PC accordingly.
Download and Install the Microsoft Visual Studio IDE on the development PC. Community Version is free, which can be used for development since this is a course project for academic purpose.

4. Downloading and Loading the Source Project

4.1. Download the Source Project

During the software development, the Version Control tool being used is GitHub, link below is the repository link where contains all the source code.
GitHub Repository: https://github.com/Jacob13209/CS524_COD3BR3AKR
There are two ways you can get the source code for this project, one is downloading directly from the webpage interface as a zip package and the other option is cloning this project repository from GitHub Desktop.
Cloning the Repository is recommended since GitHub would be used for committing code changes if needed.

4.2. Loading the Source Project

1. Starting the Microsoft Visual Studio IDE.
2. Select File → Open → Project/Solution.
3. Navigate to the location where the downloaded/unzipped source code is, select the solution file: **COD3BR3AKR.sln**
4. Upon loading successful, the source tree would be listed under **Solution Explorer** View.

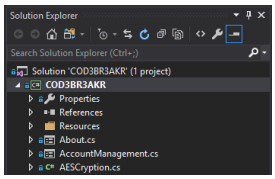

Figure 1: Project Loading

5. Committing code changes to GitHub

The tool being used for committing code changes is GitHub Desktop.

1- Cloning repository shall be the first thing to do, which can be achieved by click on File → Clone repository.

2- Setup repository by name or URL and configure local path for this repository. \

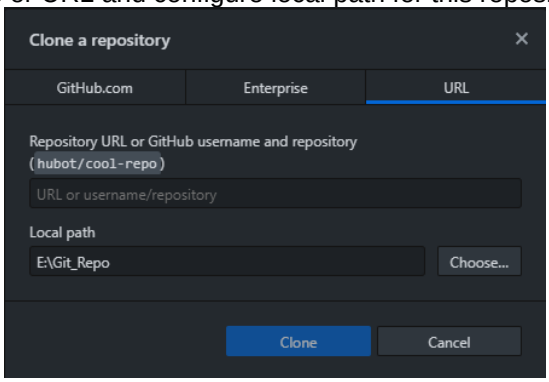

Figure 2: Cloning GitHub Repository

3- Open and Load the project by navigating to the local path where the Project Solution file being saved.

4- Make code changes upon necessary.

5- Files that being updated will show up under Changes View on GitHub Desktop, and detailed code change will show up on the right side window.

6- Once changes has been confirmed, add comments and click on Commit to master to check in changes and Click on Push to the Repository.

6. Building the Application

The system is just a Windows Form Application and the Visual Studio IDE already support built-in tools. Following steps below to show how to build an executable for the system:

1- Select the project from the **Solution Explorer** View.

2- Right click to select **Build**. The build process is started.

3- Upon successful, the **Output Window** will show the result of the build. Warning/Errors would show up if there is any. Here is an example for Successful Build.

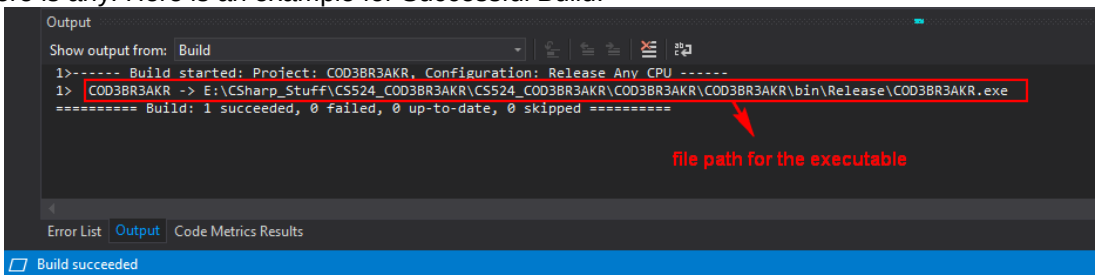

Figure 3: Application Build Success

4- Navigate to the folder where the built executable stores, double click to run.
Tip: Right Click on the project file and select “Open Folder in File Explorer” could guide you to where the project is faster.

7. Making changes to the GUI Interface

In C# Windows Form Application, all the GUI Interfaces are a collection of Controls, such as, Text Box, Label, Button, Radio Button, and Tab Control, etc.

7.1. Adding new controls to the GUI

1- In the Visual Studio IDE, make sure the Form is displayed by right click on the form and select View Designer.

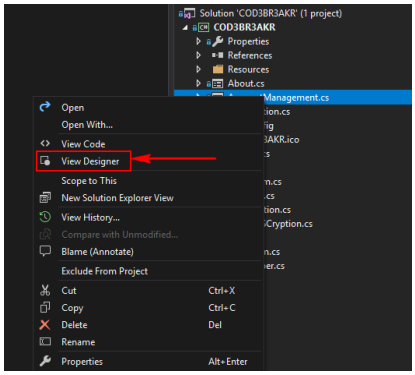

Figure 4: Switching to GUI Design Mode

2- Click on Toolbox to browse all the supported controls and select the one you want to add to the form. Drag and drop to the proper position.

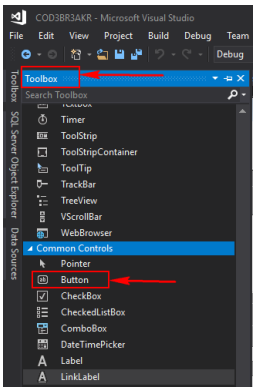

Figure 5: Adding a new control

3- Once the control is added to the form, select and right click to change the name of control by selecting the Properties. For better coding standard purpose, make sure the name being given is meaningful since this will be the identifier of the control that may be used in the code later on.

7.2. Updating existing controls

1- Select the control you want to update, and drag it to the expected position, and you can also click on Delete button to remove a control.

2- Updating the name and display text, changing color, and changing the size, etc. everything related to the out-looking can be updated in the Properties section. However, different controls have different properties. Make sure to check Microsoft Documentation if any questions occur. Here is a link to the section of Controls on Windows Form:
https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/controls-to-use-on-windows-forms

8. Debugging the system

1- Use Console Output to monitor the variable values and understand the path. This can be enabled by right click on the Project and select properties, then update Output Type to 'Console Application'. Therefore, all '*Console.WriteLine("Hello World");*' will be printed on the Console Window.

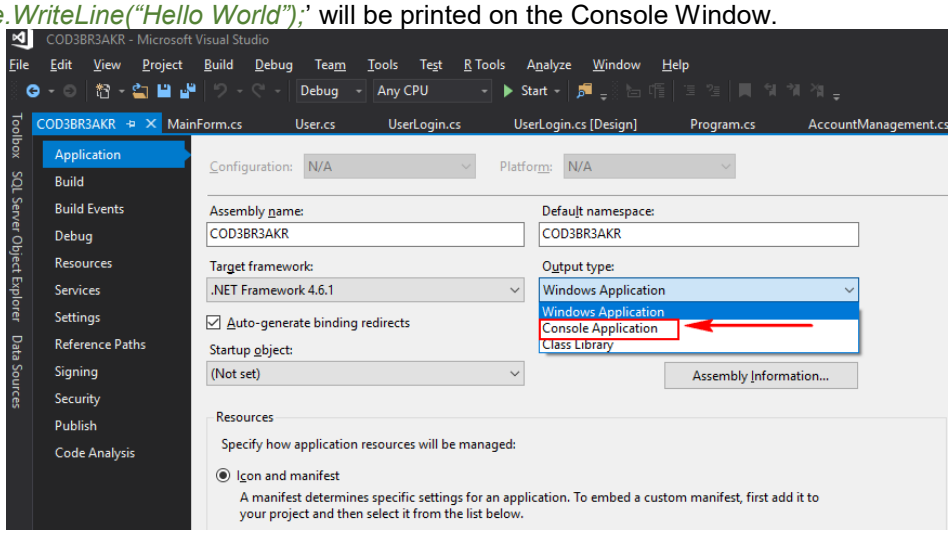

Figure 6: Switch to console

2- Adding Breakpoints to your code and use Step into. Step over to go through the paths manually.
3- Using the Watch option to monitor the value of certain variables.

9. Extending the System with a new Algorithm

According to the Architecture Design, the Encryption and Decryption part is being designed by following Object Oriented Principle, which makes it much flexible for accepting a new Encryption/Decryption Algorithm.

1- Adding a new Class for the new algorithm by right click on the project → Add → Class. Make sure name the class according to current naming convention: ***[Algorithm Name]Cryption.cs***. For example, AESCryption.cs

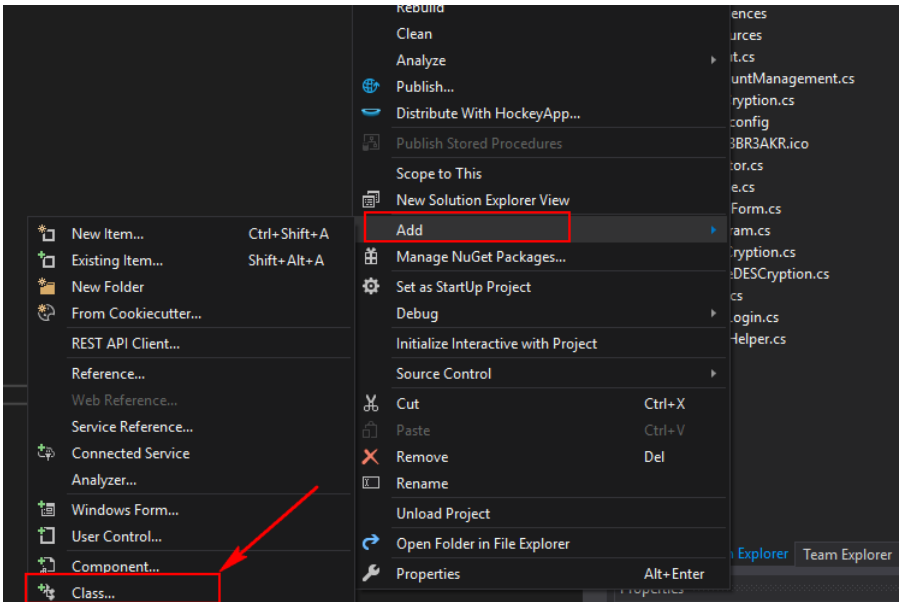

Figure 7: Adding a new Class

2- Once the new class has been created, make sure to inherit from the parent – Cryptor.cs.

3- There are 5 methods need to be implemented in order to be used in the system.
   a. `public TripleDESCryption(string key) : base(key)` ; → Constructor for this new class. The parent class has a default constructor that taking *'key'* as a parameter, you can inherit it by using base keyword that it will automatically assign that parameter to the **CryptionKey**. However, you can also introduce new parameter if it is special for the new algorithm.

```
private static string _salt = string.Empty;

public TripleDESCryption(string salt, string key)
                        : base(key)
{
    _salt = salt;
}
```

   b. `public override string Encrypt(string plainText);` → This override method is an implementation of the abstract method from parent class, which is being used to Encrypt Text String and return the encrypted string.
   c. `public override string Decrypt(string cipherText);` → This override method is an implementation of the abstract method from parent class, which is being used to Decrypt Text String and return the decrypted string.
   d. `public override bool Encrypt(string fileInput, string fileOutput);` → This override method is an implementation of the abstract method from parent class, it takes both the input file path and output file path. It is being used to encrypt a file and return true on success, false otherwise.
   e. `public override bool Decrypt(string fileInput, string fileOutput);` → This override method is an implementation of the abstract method from parent class, it takes both the input file path and output file path. It is being used to decrypt a file and return true on success, false otherwise.

4- Here is a basic example code snippet for a new Encryption/Decryption Class.

```
namespace COD3BR3AKR
{
    /// <summary>
    /// The Triple Data Encryption Standard (DES) is a symmetric key encryption algorithm for computerized cryptography.
    /// As per the algorithm, the same key is used for encryption and decryption.
    /// Also, the same block cipher algorithms are applied three times to each data block.
    /// </summary>
    public class TripleDESCryption : Cryptor
    {
        public TripleDESCryption(string key) : base(key){ }

        public override string Encrypt(string plainText)...

        public override string Decrypt(string cipherText)...

        public override bool Encrypt(string fileInput, string fileOutput)...

        public override bool Decrypt(string fileInput, string fileOutput)...
    }
}
```

Figure 8: Abstract methods need to be implemented

5- Since the algorithm has been implemented, it is time to add it to the GUI Interface. In MainForm.cs, there is a defined enumeration for `SupportedAlogrithm.` Add algorithm name to this enumeration.
6- There is also a defined Key-value pair that stores the information that will an algorithm needs key or not. `SupportedAlogrithm` is the key, `true/false` is the value. And, the true/false is being used to determine will the GUI enable the key input option or not. If the value is false, the key input would be disabled.
7- The algorithm shall be added to the GUI Interface successfully, not it is time to handle the algorithm initialization part, where handles which class shall be choose to initialize the object. In **Cryptor.cs**, there is a class named as `CryptionHandler`, which is the media that accepting input from the GUI Interface and call according class for encryption and decryption. In order to add the new algorithm, just simply update the constructor for `CrptionHelper` to handle the case of the new enumeration being added. Since `Cryptor` is the parent, it can be initialized with any algorithm. NOTE: The main idea behind this is using Polymorphism in Object Oriented Programing.

```
private Cryptor cryptor;
public CryptionHelper(string key, SupportedAlogrithm alogrithm)
{
        this.CustomizedKey      = key;
        this.SelectedAlogrithm  = alogrithm;

        switch(this.SelectedAlogrithm)
        {
            case SupportedAlogrithm.AES:
                cryptor = new AESCryption(this.CustomizedKey);
                break;
            case SupportedAlogrithm.RSA:
                cryptor = new RSACryption(this.CustomizedKey);
                break;
            case SupportedAlogrithm.TripleDES:
                cryptor = new TripleDESCryption(this.CustomizedKey);
                break;
        }
}
```

8- The new algorithm should have been added to the system successfully by now if everything goes well.

10. Adding logs to the system

In this project, `log4net` library are being used for logging, it has 5 different message levels.

- FATAL
- ERROR
- WARN
- INFO
- DEBUG

Here is an example for how to use the log functionality in the project.

1- Initialize the logger:

```
private static readonly log4net.ILog log =
log4net.LogManager.GetLogger(System.Reflection.MethodBase.GetCurrentMethod().Decla
ringType);
```

2- Calling the method of Logger Object and pass string message as parameter. Here is an example:

```
log.Info("Application is working");

log.Error("unable to add new user to the system");
```