第 4 章

C#でファイルを暗号化する

75 回生 ficorajo

4.1　前書き

挨拶

こんにちは、75 回生の ficorajo です。
中三になったということで、初めて部誌を書くことになりました。
拙い文章ですが、最後まで読んでいただけたら幸いです。

情報を守るうえで重要なもの

みなさんは、パソコンを使ったことがあるでしょうか？
パソコンは、フォルダーから階層構造として情報を管理しています。そして、自分が何かまとまった一つのテーマの物のファイルを、フォルダに格納しています。そして、我々がパソコンに侵入されたりした場合にファイルへの攻撃を防ぐには、ファイルを暗号化することが必要です。ということで、、早速暗号化していきましょう。

4.2　前準備

今回はファイルを暗号化するプログラムを書くのに、Visual Studio というのを使っていきます。理由としては、Microsoft 社が推進する.NET シリーズとの相性が非常に良いからです。(そもそも同じ会社が作っているので当たり前) では、Visual Studio をインストールしていくのですが、まず、ここから、コミュニティのバージョンを選択して、 これの通りに進めてください。これで、前準備は完了です。

4.3　C# -> NET

まず、.NET のコードを書く前に、.NET のコードの基礎知識について書いていきたいと思います。

オブジェクト指向言語

.NET の開発標準言語である C#などは、オブジェクト指向言語と呼ばれます。オブジェクト指向言語のベースとなっているオブジェクト指向とは、実世界を模倣して構築された世界観のことです。オブジェクト指向とは、分かれて存在する＜オブジェクト＞同士が＜やりとり＞を行うことで、何らかの行為が実現するという世界観を、プログラミングの世界に持ち込むことを指向する (=目指す) ことです。そして、これを言語に取り込んだものを、オブジェクト指向言語といいます。

オブジェクト指向の要素

オブジェクト指向には、次の二つの要素が存在します。

1. オブジェクト

2. オブジェクトを分類する境界

現実世界で例えれば、セールスの人が家に来て、前田ですと言われても、NHK か、教団の人か、Wi-Fi の業者かは、言われなければわかりません。これにおける前田さんがオブジェクト、NHK, 教団, Wi-Fi の業者それぞれを分類するものが、オブジェクトを分類する境界ということです。C#の世界では、オブジェクトは「クラス」、オブジェクト指向を分類するものは「名前空間」と定義されます。

ひな形コード

まず、ここまでのことを受けて、実際に少し書いてみましょう。

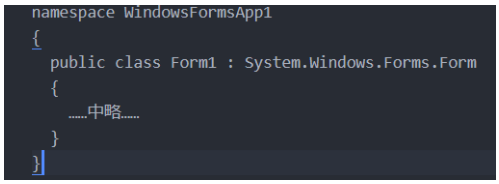

図 4.1: 雛形コード

先ほど言った通り、オブジェクト指向を分類する境界は名前空間、namespace と呼ばれ、オブジェクト指向はクラス、class と定義され、表されます。当然ですが名前空間とクラスには、始まりと終わりがあります。その間はスコープと呼ばれ、始まり終わりを{}と表します。そして、この　System.Windows.Forms.Form は、.NET のライブラリに用意されたクラスです。ここにおいて、把握しておいてほしいことがあります。ソースコードはただの設計図なだけで、本体ではないということです。本体は、System.Windows.Forms.Form なだけです。

4.4　暗号化前夜

さて、終わりが近づいてきました。もうすぐで、暗号化の部分に入りますが、もうしばらく、我慢してお読みください。

AES とは

まず、AES とは、Advanced Encryption Standerd の略で、アメリカ国立標準技術研究所 (NIST) による、厳しい選定審査によって選ばれた、オープンな暗号アルゴリズムです。米国で生まれましたが、Standerd とある通り、世界標準といっても過言ではないようになっています。現時点で、公開されながら大きな脆弱性もなく、共通鍵暗号方式では AES 一択と言っていいでしょう。

パディング

先ほど述べられた AES は 128bit(16 ビット) で暗号化されます。が、この 16 ビットで割り切れないサイズのファイルを暗号化する場合に、復号するタイミングで問題が発生します。その理由は、元のファイルのサイズが分からなくなってしまうからです。そこでブロック暗号には、暗号化モードに加えパディングモードを指定する必要があります。普通ならいちいち実装しなければならないのですが、.NET が用意してくれており、わざわざ実装する必要はありません。いろいろなパディングモードがありますが、今回は PaddingMode.PKCS7 を使うこととしましょう。

4.5　暗号化

パディングモードについての説明が終わったところで、実際に暗号化していきましょう。

List 4.1: 暗号化コード

```
1  private bool FileEncrypt(string FilePath, string Password)
2  {
3    System.Diagnostics.Stopwatch sw = new System.Diagnostics.Stopwatch();
4    sw.Start();
5
6    int i, len;
```

```
    byte[] buffer = new byte[4096];

    // Output file path.
    string OutFilePath = Path.Combine(Path.GetDirectoryName(FilePath), Path.GetFileNameWithoutExtension(FilePath)) + ".enc";

    using (FileStream outfs = new FileStream(OutFilePath, FileMode.Create, FileAccess.Write))
    {
        using (AesManaged aes = new AesManaged())
        {
            aes.BlockSize = 128;              //  BlockSize = 16bytes
            aes.KeySize = 128;                //  KeySize = 16bytes
            aes.Mode = CipherMode.CBC;        //  CBC mode
            aes.Padding = PaddingMode.PKCS7;    //  Padding mode is "PKCS7".

            Rfc2898DeriveBytes deriveBytes = new Rfc2898DeriveBytes(Password, 16);
            byte[] salt = new byte[16];
            salt = deriveBytes.Salt;
            byte[] bufferKey = deriveBytes.GetBytes(16);

            /*
            byte[] bufferKey = new byte[16];
            byte[] bufferPassword = Encoding.UTF8.GetBytes(Password);
            for (i = 0; i < bufferKey.Length; i++)
            {
                if (i < bufferPassword.Length)
                {
                    bufferKey[i] = bufferPassword[i];
                }
                else
                {
                    bufferKey[i] = 0;
                }
            */

            aes.Key = bufferKey;
            //  IV ( Initilization Vector ) は、AesManagedにつくらせる
            aes.GenerateIV();

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

            using (CryptoStream cse = new CryptoStream(outfs, encryptor, CryptoStreamMode.Write))
            {
              outfs.Write(salt, 0, 16);
              outfs.Write(aes.IV, 0, 16);
                using (DeflateStream ds = new DeflateStream(cse, CompressionMode.Compress)) // 圧縮
                {
                    using (FileStream fs = new FileStream(FilePath, FileMode.Open, FileAccess.Read))
                    {
                        while ((len = fs.Read(buffer, 0, 4096)) > 0)
                        {
                            ds.Write(buffer, 0, len);
                        }
                    }
                }
            }
        }
    }
    sw.Stop();
    long resultTime = sw.ElapsedMilliseconds;

    textBox1.AppendText("暗号化成功: " + Path.GetFileName(OutFilePath) + Environment.NewLine);
    textBox1.AppendText("実行時間: " + resultTime.ToString() + "ms");

    return (true);
}
```

これが暗号化です。見ていきましょう。まず、 System.Diagnostics.Stopwatch は、ストップウォッチとある通りストップウォッチを開始させるものです。そして、 aes.Mode = CipherMode.CBC で、暗号化モードを CBC モードとしています。さらに、 aes.Padding = PaddingMode.PKCS7 でパディングモードを設定します。また、 Rfc2898DeriveBytes deriveBytes = new Rfc2898DeriveBytes(Password, 16) で、入力されたパスをもとに乱数で暗号を設定し、 byte[] salt = new byte[16] で、自動取得されたソルトを取得するようになっています。そして、 outfs.Write(salt, 0, 16) outfs.Write(aes.IV, 0, 16) でソルトを埋め込み、 textBox1.AppendText("暗号化成功: " + Path.GetFileName(OutFilePath) + Environment.NewLine) で、暗号化が完了します。やったね。

4.6 復号

暗号化するのであれば、復号化できなければ意味がありません。

List 4.2: 復号化コード

```
  private bool FileDecrypt(string FilePath, string Password)
  {
```

```
    int i, len;
    byte[] buffer = new byte[4096];

    if (String.Compare(Path.GetExtension(FilePath), ".enc", true) != 0)
    {
        // The file are not encrypted file! Decryption failed
        MessageBox.Show("暗号化されたファイルではありません！ " + Environment.NewLine + "復号に失敗しました。",
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return (false); ;
    }

    // Output file path.
    string OutFilePath = Path.Combine(Path.GetDirectoryName(FilePath), Path.GetFileNameWithoutExtension(FilePath)) + ".txt";

    using (FileStream outfs = new FileStream(OutFilePath, FileMode.Create, FileAccess.Write))
    {
        using (FileStream fs = new FileStream(FilePath, FileMode.Open, FileAccess.Read))
        {
            using (AesManaged aes = new AesManaged())
            {
                aes.BlockSize = 128;              //  BlockSize = 16bytes
                aes.KeySize = 128;                //  KeySize = 16bytes
                aes.Mode = CipherMode.CBC;        //  CBC mode
                aes.Padding = PaddingMode.PKCS7;    //  Padding mode is "PKCS7".

                //  salt
                byte[] salt = new byte[16];
                fs.Read(salt, 0, 16);

                //  Initilization Vector
                byte[] iv = new byte[16];
                fs.Read(iv, 0, 16);
                aes.IV = iv;

                /*

                byte[] bufferKey = new byte[16];
                byte[] bufferPassword = Encoding.UTF8.GetBytes(Password);
                for (i = 0; i < bufferKey.Length; i++)
                {
                    if (i < bufferPassword.Length)
                    {
                        bufferKey[i] = bufferPassword[i];
                    }
                    else
                    {
                        bufferKey[i] = 0;
                    }
                */
                Rfc2898DeriveBytes deriveBytes = new Rfc2898DeriveBytes(Password, salt);
                byte[] bufferKey = deriveBytes.GetBytes(16);
                aes.Key = bufferKey;

                // Decryption interface.
                ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

                using (CryptoStream cse = new CryptoStream(fs, decryptor, CryptoStreamMode.Read))
                {
                    using (DeflateStream ds = new DeflateStream(cse, CompressionMode.Decompress))
                    {
                        while ((len = ds.Read(buffer, 0, 4096)) > 0)
                        {
                            outfs.Write(buffer, 0, len);
                        }
                    }
                }
            }
        }
    }
    // Decryption succeed.
    textBox1.AppendText("復号成功: " + Path.GetFileName(OutFilePath) + Environment.NewLine);
    return (true);
}
```

同じくです。見たら気づくかもしれませんが、暗号化する前に行われた圧縮を解除するための解凍作業（`using (CryptoStream cse = new CryptoStream(fs, decryptor, CryptoStreamMode.Read))`）以外は、暗号化コードを逆向きにさせたようなものです。

*このコードは、Mitsuhiro Hibara さんの MIT ライセンスであることをここに記します。

4.7 最後に

いかがでしたでしょうか。拙い文でしたが最後まで読んでいただきありがとうございました。