Лабораторная работа № 2

«Сравнение производительности симметричных алгоритмов шифрования»

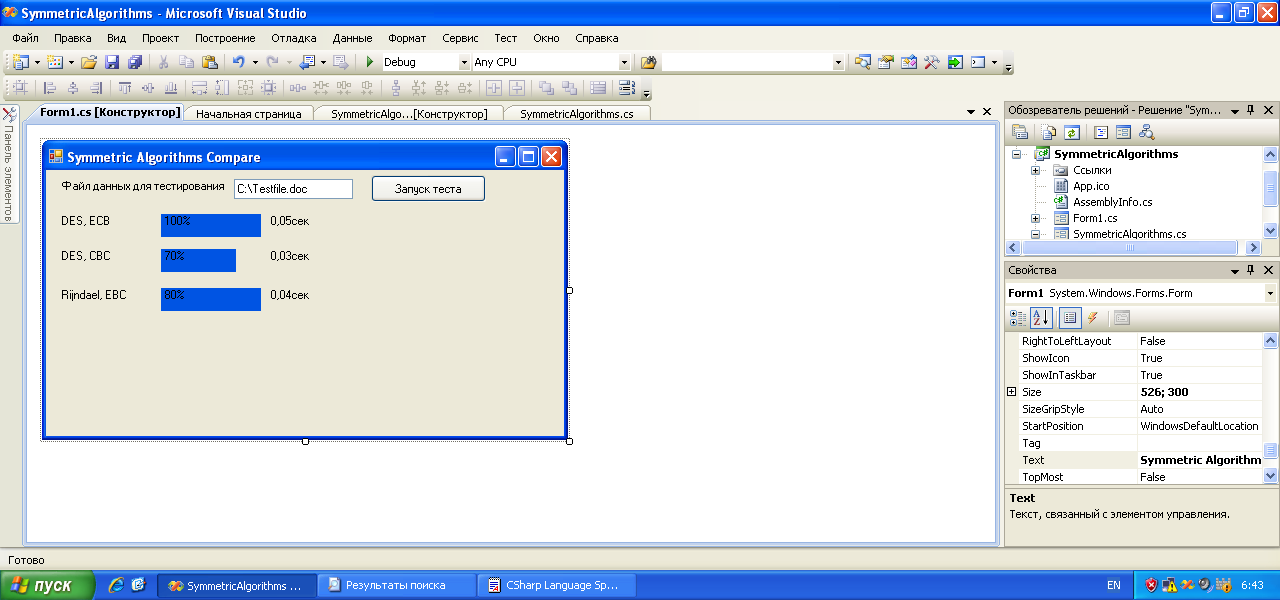
Задача: Разработать программу для сравнения производительности симметричных алгоритмов шифрования, реализованных в .NET Security Framework.

Тестирование производить на примере шифрования файла с данными, месторасположение которого должно задаваться в интерфейсе программы. В тесте необходимо подсчитать время, затрачиваемое программой на операции шифрования/дешифрования и привести сравнительную характеристику алгоритмов в виде таблицы или диаграммы с указанием

* метода шифрования (DES, 3DES, Rijndael, RC2),
* используемого режима работы (ECB, CBC, CFB, OFB, STS).

Для тестирования брать только те режимы работы алгоритмов, которые реализованы на данный момент в .NET Security Framework.

На рисунке приведен один из вариантов представления сравнительной диаграммы с использованием компонентов Label в качестве столбцов диаграммы (перечислены не все виды алгоритмов и режимы работы).



**Необходимые материалы по использованию .NET Security Framework**

Для использования криптографических алгоритмов необходимо подключение пространства имен System.Security.Cryptography оператором Using.

Классы, реализующие симметричные алгоритмы, являются производными от абстрактного класса SymmetricAlgorithm. Производными от него являются базовые классы: DES, TripleDES, Rijndael, RC2, реализующие соответствующие алгоритмы. В текущей версии .NET доступны некоторые из режимов работы. ECB, CBC, CBF работают только с алгоритмами DES, 3DES, RC2. Для алгоритма Rijndael доступны только режимы ECB, CBC. Режимы OFB, CTS еще не реализованы ни для одного алгоритма. Также следует иметь в виду, что режимы дополнения данных (PKCS7, Zero, None) до стандартного размера применимы не для всех алгоритмов/режимов работы алгоритмов. Например, для Rijndael режимы дополнения Zero, None не применимы и при использовании выдают исключение CryptoGraphicException.

Пример использования классов для шифрования по методу DES:

Создаем класс:

SymmetricAlgorithm sa = DES.Create();

Генерируем случайный ключ запоминаем его в глобальную переменную (byte[] Key) для дальнейшего дешифрования:

sa.GenerateKey();

Key = sa.Key;

Генерируем вектор инициализации и запоминаем его в глобальную переменную (byte[] IV) для дальнейшего дешифрования:

sa.GenerateIV();

IV = sa.IV;

Инициализируем режим работы шифра, например, ECB:

sa.Mode = CipherMode.ECB;

Инициализируем режим дополнения блоков до стандартного размера, например, PKCS7:

sa.Padding = PaddingMode.PKCS7;

Для более удобной манипуляции потоками данных в .NET реализован класс CryptoStream, позволяющий записывать и читать данные через криптографический поток. Используем для обеспечения операций ввода/вывода поток MemoryStream и обернем его в CryptoStream:

MemoryStream ms = new MemoryStream();

CryptoStream cs = new CryptoStream(

ms,

sa.CreateEncryptor(),

CryptoStreamMode.Write);

Создаем буфер данных и записываем в него произвольный текст:

byte[] plainbytes =

Encoding.UTF8.GetBytes("Произвольный текст");

cs.Write(plainbytes, 0, plainbytes.Length);

Закрываем криптопоток:

cs.Close();

Запоминаем уже зашифрованные данные в глобальной переменной (byte[] cipherbytes) для операции дешифрования:

cipherbytes = ms.ToArray();

Закрываем поток в памяти:

ms.Close();

Можно вывести шифрованные данные для просмотра:

Console.WriteLine(Encoding.UTF8.GetString(cipherbytes));

textPlaintext.Enabled = false;

buttonDecrypt.Select();

Пример использования классов для дешифрования по методу DES:

Создаем класс:

SymmetricAlgorithm sa = DES.Create();

Используем сохраненный ключ:

sa.Key = Key;

Используем сохраненный вектор инициализации:

sa.IV = IV;

Используем тот же режим работы шифра, что и в пример шифрования:

sa.Mode = CipherMode.ECB;

Инициализируем тот же режим дополнения блоков до необходимого размера, что и в примере шифрования:

sa.Padding = PaddingMode.PKCS7;

Создаем потоки для работы с данными, но уже с режимом Read:

MemoryStream ms = new MemoryStream(cipherbytes);

CryptoStream cs = new CryptoStream(

ms,

sa.CreateEncryptor(),

CryptoStreamMode.Read);

Создаем буфер данных для дешифрованного текста, достаточного размера:

byte[] plainbytes = new Byte[cipherbytes.Length];

Дешифруем данные:

cs.Read(plainbytes, 0, cipherbytes.Length);

Закрываем потоки:

cs.Close();

ms.Close();

Можно вывести шифрованные данные для просмотра:

Console.WriteLine(Encoding.UTF8.GetString(plainbytes));