Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Изучение устройства и функциональных особенностей**

**шифровальной машины «Энигма»**

Студент: Рубашек А. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Изучение и приобретение практических навыков разработки и использования приложений для реализации перестановочных шифров.

1. **Задание**
2. Ознакомиться с функционалом хотя бы одного симулятора Энигмы.
3. Произвести зашифрование сообщения (собственные имя, отчество, фамилия) при 8-10 различных настройках машины-симулятора. Оценить частотные свойства символов в шифртекстах и сравнить этот параметр с частотными свойствами символов для исходного текста.
4. Разработать приложение-симулятор шифровальной машины, состоящей из клавиатуры, трех роторов и отражателя. С помощью разработанного приложения зашифровать сообщение в соответствии с п.1 практического задания, применив не менее 5 вариантов начальных установок роторов.
5. Оценить криптостойкость вашего варианта машины.
6. **Ход работы**

Машина Энигма — это электромеханическое устройство, используемое для шифрования и дешифрования сообщений. Она использует несколько роторов, которые поворачиваются при каждом нажатии клавиши на клавиатуре. При повороте ротора, сигнал проходит через шифровальный элемент (например, перестановочную таблицу) и затем возвращается обратно через ротор и другие шифровальные элементы, проходя через каждый ротор на своем пути. Каждый ротор имеет свой набор проводников, называемый «коммутационной таблицей», которые перемешивают сигнал при каждом проходе.

В машине Энигма также используется «рефлектор», который направляет сигналы обратно через все роторы и шифровальные элементы в обратном порядке, создавая «обратный поток». Это создает эффект повторного шифрования, который повышает уровень защиты сообщения.

Машина Энигма имеет несколько настроек, которые могут быть изменены, чтобы изменить алгоритм шифрования. Например, можно изменить порядок роторов или коммутационных таблиц, а также количество и расположение роторов. Это делает машину Энигма очень трудной для взлома.

Код разработанного приложения реализует класс Enigma, который представляет собой шифровальную машину Энигма. Класс Enigma имеет конструктор, который принимает в качестве аргументов экземпляр класса Reflector, текущие позиции трех роторов, строку с алфавитом для шифрования. В конструкторе экземпляры роторов и рефлектора инициализируются переданными аргументами. Также, класс Enigma содержит метод Crypt(), который принимает на вход строку текста и возвращает строку, которая является результатом шифрования текста с помощью шифратора Enigma. Работает он следующим образом:

1. Создаются новые экземпляры роторов, инициализированные текущими позициями трех роторов.
2. Для каждого символа из строки text: если символ содержится в алфавите keyboard, то он передается через три ротора и рефлектор, после чего проходит через три ротора в обратном порядке и добавляется в результат. Если символ не содержится в алфавите keyboard, то он добавляется в результат без изменений.
3. После обработки каждого символа происходит проверка, нужно ли сдвинуть роторы. Если позиция самого левого ротора posL равна 0, то происходит проверка, достиг ли ротор rotorM конечной позиции, и если да, то он сдвигается на одну позицию вправо. Затем сдвигается ротор rotorR, а затем сдвигается ротор rotorM. Если позиция центрального ротора posM равна 0, то происходит проверка, достиг ли ротор rotorR конечной позиции, и если да, то он сдвигается на позицию, а ротор rotorM сдвигается на одну позицию. Затем роторы rotorR и rotorM сдвигаются на posL позиций влево. Код этого класса представлен приложении 1.

В конструкторе класса Rotor задается перестановка символов, которая определяет какой символ входит в какой выход. Символы представлены в виде массива символов. Параметр startIndex задает начальную позицию ротора. Далее, класс предоставляет несколько методов, позволяющих работать с роторами:

1. Метод this[int index] представляет собой индексатор, который позволяет получить символ на выходе ротора по его входному символу.
2. Метод GetRotor() возвращает перестановку символов в текущей позиции ротора.
3. Метод IndexOf(char symbol) возвращает индекс символа на выходе ротора по его входному символу.
4. Метод MoveRotor(int offset) позволяет сдвинуть позицию ротора на заданное количество символов.
5. Метод CurrentRotor() возвращает символ на выходе ротора в текущей позиции.
6. Метод Reset() позволяет сбросить позицию ротора в начальное состояние.

Каждый конкретный ротор, такой как RotorI, RotorII, и т.д., определяет свою перестановку символов, и использует конструктор базового класса для ее инициализации. Каждый ротор также имеет свой собственный класс и не наследует друг от друга. Код данного класса представлен в приложении 2

В классе Reflector есть один конструктор, который принимает массив пар символов. В каждой паре символы связаны между собой таким образом, что если один символ поступает на вход, то на выходе мы получим другой символ из той же пары. Массив пар символов используется для инициализации объекта и сохраняется в приватном поле \_pairs.

Класс также имеет метод Reflect, который принимает символ на вход и возвращает его замену, соответствующую правилам рефлектора. Метод Reflect ищет пару, которой принадлежит переданный символ, и возвращает другой символ из этой пары.

В классах ReflectorB, ReflectorC, ReflectorBDunn и ReflectorCDunn определены конструкторы, которые вызывают конструктор базового класса Reflector с передачей массива пар символов, заданных вручную. Каждый из этих конструкторов соответствует определенному типу рефлектора. Класс Reflector представлен на рисунке 3.1.

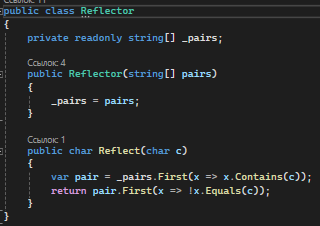


Рисунок 3.1 – Класс Reflector, реализующий отражатели

Далее, оценим криптостойкость машины Энигма с типом отражателя С Dunn, роторами III, Gamma, V.

Чтобы оценить криптостойкость шифра, нужно учитывать все возможные настройки машины. Для этого необходимо рассмотреть следующие свойства Энигмы:

* выбор и порядок роторов,
* разводку (коммутацию) роторов,
* настройку колец на каждом из роторов,
* начальное положение роторов в начале сообщения,
* отражатель,
* настройки коммутационной панели.

Чтобы выбрать 2 ротора из возможных 3, существует 6 комбинаций (3 ⋅ 2 ⋅ 1). Каждый ротор (его внутренняя проводка) может быть установлен в любом из 26 положений. Следовательно, с 3 роторами имеется 17576 различных положений ротора (26 ⋅ 26 ⋅ 26). Кольцо на каждом роторе содержит маркировку ротора и выемку, которая влияет на шаг перемещения расположенного левее ротора. Каждое кольцо может быть установлено в любом из 26 положений. Поскольку слева от третьего (наиболее левого) ротора нет ротора, на расчет влияют только кольца самого правого и среднего ротора. Это дает 676 комбинаций колец (26 ⋅ 26).

Получаем: 6 ⋅ 17576 ⋅ 676 = 71288256. В экспоненциальной форме записи 7.13 ⋅ 107 . Таким образом, практически рассматриваемая версия Энигмы (три ротора с выбором из 3 роторов, отражатель В Dunn) может быть настроена на 7.13 ⋅ 107 различных состояний, что сопоставимо с 26 – битным криптографическим ключом.

**Вывод**

В ходе лабораторной работы были изучены и приобретены практические навыки разработки и использования приложений для реализации перестановочных шифров.

Также было разработано авторское приложение в соответствии с целью лабораторной работы и была оценена криптостойкость шифра для разработанной машины Энигма.

**Приложение 1**

Класс Enigma, реализующий шифровальную машину Энигма

using System.Text;

public class Enigma

{

public int posL;

public int posM;

public int posR;

public Rotor rotorL;

public Rotor rotorM;

public Rotor rotorR;

private Reflector reflector;

private string keyboard;

public Enigma(Reflector reflector, int posL, int posM, int posR, string keyboard)

{

this.posL= posL;

this.posM = posM;

this.posR = posR;

this.reflector = reflector;

this.keyboard = keyboard;

}

public string Crypt(string text)

{

var rotorL = new RotorIII(posL);

var rotorM = new Gamma(posM);

var rotorR = new RotorV(posR);

var result = new StringBuilder(text.Length);

char symbol;

foreach (var ch in text)

{

if (keyboard.Contains(ch))

{

symbol = rotorR[keyboard.IndexOf(ch)];

}

else

{

result.Append(ch);

continue;

}

symbol = rotorM[keyboard.IndexOf(symbol)];

symbol = rotorL[keyboard.IndexOf(symbol)];

symbol = reflector.Reflect(symbol);

symbol = keyboard[rotorL.IndexOf(symbol)];

symbol = keyboard[rotorM.IndexOf(symbol)];

symbol = keyboard[rotorR.IndexOf(symbol)];

result.Append(symbol);

if (posL == 0)

{

if (rotorM.isFullyRotated)

{

rotorL.MoveRotor(1);

}

rotorR.MoveRotor(posR);

rotorM.MoveRotor(posM);

}

else if (posM == 0)

{

if (rotorR.isFullyRotated)

{

rotorM.MoveRotor(1);

}

rotorR.MoveRotor(posR);

rotorL.MoveRotor(posL);

}

else

{

rotorR.MoveRotor(posR);

rotorM.MoveRotor(posM);

rotorL.MoveRotor(posL);

}

}

return result.ToString();

}

}

**Приложение 1**

Класс Rotor, реализующий роторы

public abstract class Rotor

{

protected readonly char[] \_rotorChar;

protected int \_currentIndex;

public bool isFullyRotated;

public Rotor(string rotorString, int startIndex)

{

\_rotorChar = rotorString.ToCharArray();

\_currentIndex = startIndex >= rotorString.Length ? 0 : startIndex;

}

public char this[int index]

{

get

{

return \_rotorChar[(index + \_currentIndex) % \_rotorChar.Length];

}

}

public string GetRotor()

{

string res = null;

for (int i = 0; i < \_rotorChar.Length; i++)

{

res += \_rotorChar[(\_currentIndex + i) % \_rotorChar.Length];

}

return res;

}

public int IndexOf(char symbol)

{

int index = \_rotorChar.ToList().IndexOf(symbol);

// shows how much the shift to the right has occurred

int rightOffset = \_rotorChar.Length - \_currentIndex;

int offsetRotorIndex = (index + rightOffset) % \_rotorChar.Length;

return offsetRotorIndex;

}

public void MoveRotor(int offset)

{

\_currentIndex += offset;

if (\_currentIndex >= \_rotorChar.Length)

{

\_currentIndex %= \_rotorChar.Length;

isFullyRotated = true;

}

else{ isFullyRotated = false; }

}

public char CurrentRotor()

{

return \_rotorChar[\_currentIndex];

}

public void Reset()

{

\_currentIndex = 0;

}

}

public class RotorI : Rotor

{

public RotorI(int startIndex = 0) : base("EKMFLGDQVZNTOWYHXUSPAIBRCJ", startIndex)

{

}

}

public class RotorII : Rotor

{

public RotorII(int startIndex = 0) : base("AJDKSIRUXBLHWTMCQGZNPYFVOE", startIndex)

{

}

}

public class RotorIII : Rotor

{

public RotorIII(int startIndex = 0) : base("BDFHJLCPRTXVZNYEIWGAKMUSQO", startIndex)

{

}

}

public class RotorIV : Rotor

{

public RotorIV(int startIndex = 0) : base("ESOVPZJAYQUIRHXLNFTGKDCMWB", startIndex)

{

}

}

public class RotorV : Rotor

{

public RotorV(int startIndex = 0) : base("VZBRGITYUPSDNHLXAWMJQOFECK", startIndex)

{

}

}

public class RotorVI : Rotor

{

public RotorVI(int startIndex = 0) : base("JPGVOUMFYQBENHZRDKASXLICTW", startIndex)

{

}

}

public class RotorVII : Rotor

{

public RotorVII(int startIndex = 0) : base("NZJHGRCXMYSWBOUFAIVLPEKQDT", startIndex)

{

}

}

public class RotorVIII : Rotor

{

public RotorVIII(int startIndex = 0) : base("FKQHTLXOCBJSPDZRAMEWNIUYGV", startIndex)

{

}

}

public class Beta : Rotor

{

public Beta(int startIndex = 0) : base("LEYJVCNIXWPBQMDRTAKZGFUHOS", startIndex)

{

}

}

public class Gamma : Rotor

{

public Gamma(int startIndex = 0) : base("FSOKANUERHMBTIYCWLQPZXVGJD", startIndex)

{

}

}