Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

**Звіт**

**Лабораторна робота 1**

з дисципліни "Інформаційна безпека"

Виконав: студент групи КН-М624

Кальницький Б. І.

Перевірив:

Івашко А.В.

Харків 2025

**Зміст**

[1. Завдання 3](#_Toc194341267)

[2. Теоретичні відомості 4](#_Toc194341268)

[2.1 Шифр блочної одинарної перестановки 4](#_Toc194341269)

[2.2 Шифр вертикальної перестановки 4](#_Toc194341270)

[2.3 Еквівалентна заміна шифру подвійної перестановки шифром простої одинарної перестановки 5](#_Toc194341271)

[3. Програмна реалізація 7](#_Toc194341272)

[3.1. BlockTranspositionEncryptionService 7](#_Toc194341273)

[3.2. VerticalTranspositionEncryptionService 8](#_Toc194341274)

[3.3. EquivalentTranspositionService 9](#_Toc194341275)

[Висновок 13](#_Toc194341276)

# Завдання

**Симетричне шифрування. Використання шифрів перестановки.**

**Мета роботи**: програмна реалізація алгоритмів і методів використання шифрів з різними видами перестановок.

1. Створити застосунок, який згідно з параметрами варіанту приймає повідомлення для шифрування та, якщо потрібно, ключ шифрування і виконує шифрування повідомлення, тобто видає шифрограму.
2. Створити застосунок, який приймає шифрограму з минулого завдання та виконує дешифрування похідного повідомлення.
3. Модифікувати минулі завдання таким чином, що розмір початкового повідомлення можна було задавати динамічно.

Параметри завдань

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варіанту | Довжина повідомлення | Тип шифру перестановок | Розмір блоку | Спосіб створення ключа |
| 2 | 11 | Блочна | 3 | випадково |

|  |  |  |  |
| --- | --- | --- | --- |
| № варіанту | Довжина повідомлення | Тип шифру перестановок | Спосіб створення ключа |
| 2 | 21 | Шифр вертикальної перестановки | випадково |

|  |  |  |  |
| --- | --- | --- | --- |
| № варіанту | Довжина повідомлення | Тип шифру перестановок | Спосіб створення ключа |
| 2 | 11 | Еквівалентна заміна шифру подвійної перестановки шифром простої одинарної перестановки | випадково |

# Теоретичні відомості

## Шифр блочної одинарної перестановки

У випадку для даного класу шифрів при шифруванні і дешифруванні використовується таблиця перестановок.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | … | n |
|  |  |  | … |  |

У першому рядку цієї таблиці вказується позиція символу у вихідному повідомленні, а у другому – його позиція у шифрограмі. Таким чином, максимальна кількість ключів для шифрів перестановки дорівнює n!, де n – довжина повідомлення.

Зі збільшенням числа n, значення n! зростає дуже швидко:

* 1! = 1;
* 5! = 120;
* 10! = 3628800;
* 15! = 1307674368000;

Шифр блочної одинарної перестановки задає позиції перестановки не для всіх символів, а для блоку, який є повторюваним для вхідного тексту. Також шифрограма доповнюється будь-якими літерами для кратності розміру блоку, адже не всі повідомлення можуть мати необхідний розмір тексту.

Приклад шифрування

Ключ – таблиця перестановок.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 2 | 3 | 1 |

Вихідне повідомлення – АБРАМОВ (доповнюється точками для кратності розміру блоку)

Шифрограма – РАБОАМ.В.

## Шифр вертикальної перестановки

Шифр вертикальної перестановки передбачає вписування вихідного повідомлення в таблицю за принципом: зліва направо, рядок за рядком. Кількість стовпців таблиці відповідає довжині ключа, причому сам ключ розташовується у першому рядку. В якості ключа може використовуватися слово або фраза. Також таблиця доповнюється будь-якими літерами, адже не всі повідомлення можуть мати необхідний розмір тексту.

Після заповнення таблиці символи зчитуються по стовпцях зверху вниз, у порядку, що визначається алфавітним розташуванням літер ключа. Якщо якась літера повторюється кілька разів, її появи нумеруються за черговістю зліва направо.

Приклад шифрування

Ключ (слово чи фраза) – ДЯДИНА

Вихідне повідомлення – АБРАМОВ\_ІЛЛЯ\_СЕРГІЙОВИЧ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ключ | Д | Я | Д | И | Н | А |
| 2 | 6 | 3 | 4 | 5 | 1 |
| Текст | А | Б | Р | А | М | О |
| В | \_ | І | Л | Л | Я |
| \_ | С | Е | Р | Г | І |
| Й | О | В | И | Ч | . |

Шифрограма – ОЯІ.АВ\_ЙРІЕВАЛРИМЛГЧБ\_СО

## Еквівалентна заміна шифру подвійної перестановки шифром простої одинарної перестановки

В якості ключа шифр приймає такі дані:

* нумерація рядків та стовпців;
* маршрут вписування;
* маршрут виписування.

Спочатку формується таблиця, яка заповнюється згідно з обраним маршрутом вписування. Якщо в таблиці залишаються порожні комірки, вони заповнюються довільними символами. Далі кожна комірка змінює своє місце відповідно до заданої нумерації рядків і стовпців.

У результаті з отриманої таблиці формується шифрограма, яка виписується згідно з визначеним маршрутом. Цей процес також можна розглядати як варіант шифру простої одинарної перестановки.

Приклад шифрування

Ключ:

* нумерація рядків: [3, 1, 4, 2], нумерація стовпців: [4, 1, 3, 2];
* маршрут вписування: зліва-направо зверху-вниз;
* маршрут виписування: зверху-вниз зліва-направо.

Початкова таблиця:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 4 | 1 | 3 | 2 |
| 3 | А | Б | Р | А |
| 1 | М | О | В | + |
| 4 | Д | Я | Д | И |
| 2 | Н | А | . | . |

Таблиця з переставленими символами:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | О | + | В | М |
| 2 | А | . | . | Н |
| 3 | Б | А | Р | А |
| 4 | Я | И | Д | Д |

Виписуючи з цієї таблиці дані, отримаємо:

Шифрограма: ОАБЯ+.АИВ.РДМНАД

Шифр простої одинарної перестановки:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| А | Б | Р | А | М | О | В | + | Д | Я | Д | И | Н | А | . | . |
| 15 | 3 | 11 | 7 | 13 | 1 | 9 | 5 | 16 | 4 | 12 | 8 | 14 | 2 | 10 | 6 |
| О | А | Б | Я | + | . | А | И | В | . | Р | Д | М | Н | А | Д |

# Програмна реалізація

Для реалізації алгоритмів використовувалась мова програмування C# та технологія ASP .NET Core Web App.

Було створено такі інтерфейси:

1. IEncryptionKey – для зберігання інформації про ключ.
2. IEncryptionService – для шифрування повідомлень.

Класи для зберігання інформації про ключі:

1. BlockTranspositionKey – ключ для блочної перестановки.
2. VerticalTranspositionKey – ключ для вертикальної перестановки.
3. EquivalentTranspositionKey – ключ для еквівалентної перестановки.
4. EquivalentTranspositionKeyData – основний вміст ключа для еквівалентної перестановки.

Класи для представлення результату шифрування:

1. EncryptionResult – базовий клас результату шифрування.
2. VerticalTranspositionEncryptionResult – результат шифрування вертикальною перестановкою.
3. EquivalentTranspositionEncryptionResult – результат шифрування еквівалентною перестановкою.

Для виконання шифрування було створено такі класи:

## BlockTranspositionEncryptionService

using EncryptionService.Core.Interfaces;

using EncryptionService.Core.Models;

using EncryptionService.Core.Models.BlockTransposition;

namespace EncryptionService.Core.Services

{

public class BlockTranspositionEncryptionService

: IEncryptionService<EncryptionResult, BlockTranspositionKey, int[]>

{

private const char FILL\_CHAR = '.';

public EncryptionResult Encrypt(string text, BlockTranspositionKey encryptionKey)

=> ProcessEncryption(text, encryptionKey, true);

public EncryptionResult Decrypt(string encryptedText, BlockTranspositionKey encryptionKey)

=> ProcessEncryption(encryptedText, encryptionKey, false);

private static EncryptionResult ProcessEncryption(string text,

BlockTranspositionKey encryptionKey, bool isEncryption)

{

while (text.Length % encryptionKey.Key.Length != 0)

text += FILL\_CHAR;

char[] resultArr = new string(FILL\_CHAR, text.Length).ToCharArray();

int blockNumber = 0;

for (int i = 0, keyIndex = 0; i < text.Length; i++, keyIndex++)

{

if (i == blockNumber \* encryptionKey.Key.Length)

{

keyIndex = 0;

blockNumber++;

}

int index = encryptionKey.Key[keyIndex] - 1

+ (blockNumber - 1) \* encryptionKey.Key.Length;

if (isEncryption)

resultArr[index] = text[i];

else

resultArr[i] = text[index];

}

return new(new string(resultArr));

}

}

}

## VerticalTranspositionEncryptionService

using EncryptionService.Core.Interfaces;

using EncryptionService.Core.Models.VerticalTransposition;

namespace EncryptionService.Core.Services

{

public class VerticalTranspositionEncryptionService

: IEncryptionService<VerticalTranspositionEncryptionResult, VerticalTranspositionKey,

string>

{

private const char FILL\_CHAR = '.';

public VerticalTranspositionEncryptionResult Encrypt(string text,

VerticalTranspositionKey encryptionKey)

=> ProcessEncryption(text, encryptionKey, true);

public VerticalTranspositionEncryptionResult Decrypt(string encryptedText,

VerticalTranspositionKey encryptionKey)

=> ProcessEncryption(encryptedText, encryptionKey, false);

private static VerticalTranspositionEncryptionResult ProcessEncryption(string text,

VerticalTranspositionKey encryptionKey, bool isEncryption)

{

var sorted = encryptionKey.Key

.Select((ch, index) => new { ch, index })

.OrderBy(x => x.ch)

.Select((x, newIndex) => new { x.index, newIndex })

.ToDictionary(x => x.index, x => x.newIndex);

char[,] matrix = new char[(int)Math.Ceiling((double)text.Length

/ encryptionKey.Key.Length), encryptionKey.Key.Length];

matrix = FillMatrix(matrix, text, sorted, isEncryption);

string resultText = string.Empty;

if (isEncryption)

foreach (int j in sorted.OrderBy(x => x.Value).Select(x => x.Key))

for (int i = 0; i < matrix.GetLength(0); i++)

resultText += matrix[i, j];

else

for (int i = 0; i < matrix.GetLength(0); i++)

for (int j = 0; j < matrix.GetLength(1); j++)

resultText += matrix[i, j];

int[] sortIndixes = [.. encryptionKey.Key.Select((\_, i) => sorted[i])];

return new VerticalTranspositionEncryptionResult(resultText, matrix, sortIndixes);

}

private static char[,] FillMatrix(char[,] matrix, string text, Dictionary<int, int> sorted,

bool isEncryption)

{

int k = 0;

if (isEncryption)

{

for (int i = 0; i < matrix.GetLength(0); i++)

for (int j = 0; j < matrix.GetLength(1); j++)

{

if (k < text.Length)

matrix[i, j] = text[k++];

else

matrix[i, j] = FILL\_CHAR;

}

}

else

{

foreach (int j in sorted.OrderBy(x => x.Value).Select(x => x.Key))

for (int i = 0; i < matrix.GetLength(0); i++)

{

if (k < text.Length)

matrix[i, j] = text[k++];

else

matrix[i, j] = FILL\_CHAR;

}

}

return matrix;

}

}

}

## EquivalentTranspositionService

using EncryptionService.Core.Interfaces;

using EncryptionService.Core.Models;

using EncryptionService.Core.Models.EquivalentTransposition;

namespace EncryptionService.Core.Services

{

public class EquivalentTranspositionService

: IEncryptionService<EquivalentTranspositionEncryptionResult, EquivalentTranspositionKey,

EquivalentTranspositionKeyData>

{

private const char FILL\_CHAR = '.';

private static int \_rowCount;

private static int \_columnCount;

private static char[,]? \_initialMatrix;

private static char[,]? \_transpositionMatrix;

private static int[,]? \_intialIndexes;

private static int[]? \_transpositionIndexes;

public EquivalentTranspositionEncryptionResult Encrypt(string text,

EquivalentTranspositionKey encryptionKey)

=> ProcessEncryption(text, encryptionKey, true);

public EquivalentTranspositionEncryptionResult Decrypt(string encryptedText,

EquivalentTranspositionKey encryptionKey)

=> ProcessEncryption(encryptedText, encryptionKey, false);

private static EquivalentTranspositionEncryptionResult ProcessEncryption(string text,

EquivalentTranspositionKey encryptionKey, bool isEncryption)

{

\_rowCount = encryptionKey.Key.RowNumbers.Length;

\_columnCount = encryptionKey.Key.ColumnNumbers.Length;

while (text.Length != \_rowCount \* \_columnCount)

text += FILL\_CHAR;

EquivalentTranspositionKeyData key = encryptionKey.Key;

char[,] matrix = new char[\_rowCount, \_columnCount];

\_intialIndexes = new int[\_rowCount, \_columnCount];

Direction firstDirection = isEncryption ? key.FirstWritingDirection

: key.FirstReadingDirection;

Direction secondDirection = isEncryption ? key.SecondWritingDirection

: key.SecondReadingDirection;

int k = 0;

ProcessCells(firstDirection, secondDirection,

(i, j) =>

{

\_intialIndexes[i, j] = k;

matrix[i, j] = text[k++];

});

\_initialMatrix = matrix;

matrix = TranspositionMatrix(matrix, encryptionKey.Key.RowNumbers,

encryptionKey.Key.ColumnNumbers, isEncryption);

\_transpositionIndexes = new int[\_rowCount \* \_columnCount];

string resultText = ReadResults(matrix, key, isEncryption);

return new EquivalentTranspositionEncryptionResult(resultText, \_initialMatrix,

\_transpositionMatrix!, \_transpositionIndexes);

}

private static string ReadResults(char[,] matrix,

EquivalentTranspositionKeyData key, bool isEncryption)

{

string resultText = string.Empty;

Direction firstDirection = isEncryption ? key.FirstReadingDirection

: key.FirstWritingDirection;

Direction secondDirection = isEncryption ? key.SecondReadingDirection

: key.SecondWritingDirection;

int k = 0;

ProcessCells(firstDirection, secondDirection,

(i, j) =>

{

\_transpositionIndexes![\_intialIndexes![i, j]] = k++;

resultText += matrix[i, j];

});

return resultText;

}

private static void ProcessCells(Direction firstDirection, Direction secondDirection,

Action<int, int> processCell)

{

(int firstStart, int firstEnd, int firstStep) = GetDirectionRange(firstDirection);

(int secondStart, int secondEnd, int secondStep) = GetDirectionRange(secondDirection);

for (int i = secondStart; i != secondEnd; i += secondStep)

for (int j = firstStart; j != firstEnd; j += firstStep)

{

if (firstDirection == Direction.Left || firstDirection == Direction.Right)

processCell(i, j);

else

processCell(j, i);

}

}

private static (int start, int end, int step) GetDirectionRange(Direction direction)

{

int start, end, step;

if (direction == Direction.Left || direction == Direction.Right)

{

start = direction == Direction.Right ? 0 : \_columnCount - 1;

end = direction == Direction.Right ? \_columnCount : -1;

}

else

{

start = direction == Direction.Down ? 0 : \_rowCount - 1;

end = direction == Direction.Down ? \_rowCount : -1;

}

step = start < end ? 1 : -1;

return (start, end, step);

}

private static char[,] TranspositionMatrix(char[,] matrix, int[] rowNumbers,

int[] columnNumbers, bool isEncryption)

{

char[,] resultMatrix = new char[matrix.GetLength(0), matrix.GetLength(1)];

int[,] resultIndexes = new int[\_intialIndexes!.GetLength(0),

\_intialIndexes.GetLength(1)];

for (int i = 0; i < matrix.GetLength(0); i++)

for (int j = 0; j < matrix.GetLength(1); j++)

{

int transpositionI = rowNumbers[i] - 1;

int transpositionJ = columnNumbers[j] - 1;

if (isEncryption)

{

resultIndexes[transpositionI, transpositionJ] = \_intialIndexes[i, j];

resultMatrix[transpositionI, transpositionJ] = matrix[i, j];

}

else

{

resultIndexes[i, j] = \_intialIndexes[transpositionI, transpositionJ];

resultMatrix[i, j] = matrix[transpositionI, transpositionJ];

}

}

\_transpositionMatrix = resultMatrix;

\_intialIndexes = resultIndexes;

return resultMatrix;

}

}

}

# Висновок

У ході виконання лабороторної роботи було визначено симетричне шифрування та шифри перестановок. Також було програмно реалізовано алгоритми і методи шифрування з різними видами перестановок, а саме: блочна одинарна перестановка, вертикальна перестановка, еквівалентна заміна шифру подвійної перестановки шифром простої одинарної перестановки і тд.

Для кожного методу шифрування було створено веб-сторінку за допомогою мови програмування C# та технології ASP .NET Core Web App, в якій користувач може ввести текст і отримати результат шифрування разом із додатковою інформацією про його виконання.