Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

**Звіт**

**Лабораторна робота 2**

з дисципліни "Інформаційна безпека"

Виконав: студент групи КН-М624

Кальницький Б. І.

Перевірив:

Івашко А.В.

Харків 2025

**Зміст**

[1. Завдання 3](#_Toc194341236)

[2. Теоретичні відомості 4](#_Toc194341237)

[2.1 Гасло-шифр 4](#_Toc194341238)

[2.2 Шифр «Чесна гра» 4](#_Toc194341239)

[2.3 Шифр за допомогою системи омофонів 5](#_Toc194341240)

[3. Програмна реалізація 6](#_Toc194341241)

[3.1 SloganEncryptionService 6](#_Toc194341242)

[3.2 PlayfairEncryptionService 7](#_Toc194341243)

[3.3 HomophonicEncryptionService 11](#_Toc194341244)

[Висновок 13](#_Toc194341245)

# Завдання

**Симетричне шифрування. Використання шифрів заміни.**

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

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 замін | Спосіб створення ключа |
| 2 | 11 | гасло-шифр | випадково |

|  |  |  |  |
| --- | --- | --- | --- |
| № варіанту | Довжина повідомлення | Тип шифру замін | Спосіб створення ключа |
| 2 | 11 | «Чесна гра» | випадково |

|  |  |  |  |
| --- | --- | --- | --- |
| № варіанту | Довжина повідомлення | Тип шифру замін | Спосіб створення ключа |
| 2 | 11 | Система омофонів | випадково |

# Теоретичні відомості

## 2.1 Гасло-шифр

Таблиця шифрозаміни заповнюється спочатку словом-гаслом (повторювані літери відкидаються), а потім рештою літер, що не увійшли в слово-гасло, в алфавітному порядку.

Приклад шифрування

Ключ (слово гасло чи фраза) – ДЯДИНА.

Таблиця шифрозаміни:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| А | Б | В | Г | Ґ | Д | Е | Є | Ж | З | И | І | Ї | Й | К | Л | М |
| Д | Я | И | Н | А | Б | В | Г | Ґ | Е | Є | Ж | З | І | Ї | Й | К |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я |
| Л | М | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю |

Вихідне повідомлення – АБРАМОВ

Шифрограма – ДЯПДКМИ

Кількість ключів для цього прикладу – 33!

## 2.2 Шифр «Чесна гра»

Вихідне повідомлення розбивається на біграми. Якщо в біграмі зустрічаються два однакових символи, то між ними додається заздалегідь обумовлений допоміжний символ.

Таблиця шифрозамін формується за правилами шифруючої системи Трисемуса.

Якщо символи біграми вихідного тексту зустрічаються у одному рядку, ці символи заміщаються на символи, які у найближчих стовпцях праворуч від відповідних символів. Якщо символ є останнім у рядку, він замінюється на перший символ цього ж рядка.

Якщо символи біграми вихідного тексту зустрічаються у одному стовпці, всі вони перетворюються на символи того ж самого стовпця, що є безпосередньо під ними. Якщо символ є нижнім у стовпці, він замінюється на перший символ цього ж стовпця.

Якщо символи біграми вихідного тексту перебувають у різних стовпцях і різних рядках, вони замінюються на символи, що у тих самих рядках, але відповідні протилежним кутам прямокутника.

Приклад шифрування

Розміри таблиці – 6 на 6.

Ключ (слово чи фраза) – ДЯДИНА

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Д | Я | И | Н | А | Б |
| В | Г | Ґ | Е | Є | Ж |
| З | І | Ї | Й | К | Л |
| М | О | П | Р | С | Т |
| У | Ф | Х | Ц | Ч | Ш |
| Щ | Ь | Ю | - | 1 | 2 |

Вихідне повідомлення – ЗА ШИ ФР ОВ АН ЕП ОВ ІД ОМ ЛЕ Н1 Я2

Шифрограма – КД ХБ ЦО МГ БА ҐР МГ ЗЯ ПО ЙЖ А- БЬ

## 2.3 Шифр за допомогою системи омофонів

Кожній літері вихідного алфавіту ставиться у відповідність кілька унікальних шифрозамін, число яких пропорційно частоті літери у відкритих текстах.

Кожна шифрозаміна повинна складатися з 3 цифр та їх загальна кількість дорівнює 1000.

При шифруванні символ вихідного повідомлення замінюється будь-якою шифрозаміною зі свого стовпця. Якщо символ зустрічається повторно, зазвичай використовують різні шифрозаміни. Ключ – таблиця шифрозамін.

Вихідне повідомлення – АБРАМОВ

Шифрограма: 357990374678037828175

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | А | Б | В | … | М | … | О | … | Р | … | Я |
| 1 | 311 | 128 | 175 | … | 037 | … | 248 | … | 064 | … | 266 |
| 2 | 357 | 950 | 194 | … | 149 | … | 267 | … | 189 | … | 333 |
| … | … | … | … | … | … | … | … | … | … | … | … |
| 16 | 495 | 990 | 199 | … | 349 | … | 303 | … | 374 | … | 749 |
| … | … | … | … | … | … | … | … | … | … | … | … |
| 45 | 678 |  | 644 |  |  |  | 824 | … | 721 |  |  |
| … | … |  |  |  |  |  | … | … | … |  |  |
| 47 | 776 |  |  |  |  |  | 828 | … | 954 |  |  |

# 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. SloganEncryptionKey – ключ для гасло-шифру.
2. PlayfairEncryptionKey – ключ для шифру «чесна гра».
3. HomophonicEncryptionKey – ключ для шифру за допомогою системи омофонів.

Класи для представлення результату шифрування:

1. EncryptionResult – базовий клас результату шифрування.
2. SloganEncryptionResult – результат шифрування гасло-шифру.
3. PlayfairEncryptionResult – результат шифрування шифру «чесна гра».
4. HomophonicEncryptionResult – результат шифрування за допомогою системи омофонів.

Для виконання шифрування було створено такі класи:

## 3.1 SloganEncryptionService

using System.Text;

using EncryptionService.Core.Interfaces;

using EncryptionService.Core.Models.SloganEncryption;

namespace EncryptionService.Core.Services

{

public class SloganEncryptionService :

IEncryptionService<SloganEncryptionResult, SloganEncryptionKey, string>

{

private const char UNKNOWN\_CHAR = '�';

private static readonly string ukrainianAlphabet = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ";

public SloganEncryptionResult Encrypt(string text, SloganEncryptionKey encryptionKey)

=> ProcessEncryption(text, encryptionKey, true);

public SloganEncryptionResult Decrypt(string encryptedText,

SloganEncryptionKey encryptionKey)

=> ProcessEncryption(encryptedText, encryptionKey, false);

private static SloganEncryptionResult ProcessEncryption(string text,

SloganEncryptionKey encryptionKey, bool isEncryption)

{

Dictionary<char, char> encryptionMap = CreateEncryptionMap(encryptionKey.Key);

text = text.ToUpper();

StringBuilder builder = new();

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

{

if (isEncryption)

{

if (encryptionMap.TryGetValue(text[i], out char value))

builder.Append(value);

else

builder.Append(UNKNOWN\_CHAR);

}

else

{

char ch = encryptionMap.FirstOrDefault(x => x.Value == text[i]).Key;

if (ch != default)

builder.Append(ch);

else

builder.Append(UNKNOWN\_CHAR);

}

}

return new SloganEncryptionResult(builder.ToString(), encryptionMap);

}

private static Dictionary<char, char> CreateEncryptionMap(string key)

{

Dictionary<char, char> encryptionMap = [];

char[] encryptionKeyArr = new HashSet<char>(key).ToArray();

StringBuilder tempAlphabet = new(ukrainianAlphabet);

int k = 0;

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

{

if (i < encryptionKeyArr.Length)

{

encryptionMap.Add(ukrainianAlphabet[i], encryptionKeyArr[i]);

int index = tempAlphabet.ToString().IndexOf(encryptionKeyArr[i]);

tempAlphabet.Remove(index, 1);

}

else

encryptionMap.Add(ukrainianAlphabet[i], tempAlphabet[k++]);

}

return encryptionMap;

}

}

}

## 3.2 PlayfairEncryptionService

using System.Text;

using EncryptionService.Core.Interfaces;

using EncryptionService.Core.Models.PlayfairEncryption;

namespace EncryptionService.Core.Services

{

public class PlayfairEncryptionService

: IEncryptionService<PlayfairEncryptionResult, PlayfairEncryptionKey, string>

{

private const char FILL\_CHAR = '.';

private const char UNKNOWN\_CHAR = '�';

private const char ADDITIONAL\_CHAR = 'Х';

private static readonly string ukrainianAlphabet = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ";

public PlayfairEncryptionResult Encrypt(string text, PlayfairEncryptionKey encryptionKey)

=> ProcessEncryption(text, encryptionKey, true);

public PlayfairEncryptionResult Decrypt(string encryptedText,

PlayfairEncryptionKey encryptionKey)

=> ProcessEncryption(encryptedText, encryptionKey, false);

private static PlayfairEncryptionResult ProcessEncryption(string text,

PlayfairEncryptionKey encryptionKey, bool isEncryption)

{

char[,] encryptionTable = CreateEncryptionTable(encryptionKey.Key);

text = text.ToUpper();

if (isEncryption)

for (int i = 0; i < text.Length - 1; i += 2)

if (text[i] == text[i + 1])

text = text.Insert(i + 1, ADDITIONAL\_CHAR.ToString());

if (text.Length % 2 != 0)

text += FILL\_CHAR;

string resultText = ProcessText(text, encryptionTable, isEncryption);

return new PlayfairEncryptionResult(resultText, encryptionTable);

}

private static char[,] CreateEncryptionTable(string key)

{

int size = (int)Math.Round(Math.Sqrt(ukrainianAlphabet.Length));

char[,] encryptionTable = new char[size, size];

List<char> encryptionKeyList = new HashSet<char>(key).ToList();

StringBuilder tempAlphabet = new(ukrainianAlphabet);

int k = 0;

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

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

{

if (encryptionKeyList.Count > 0)

{

encryptionTable[i, j] = encryptionKeyList[0];

int index = tempAlphabet.ToString().IndexOf(encryptionKeyList[0]);

encryptionKeyList.RemoveAt(0);

tempAlphabet.Remove(index, 1);

}

else

{

if (k < tempAlphabet.Length)

encryptionTable[i, j] = tempAlphabet[k++];

else

encryptionTable[i, j] = FILL\_CHAR;

}

}

return encryptionTable;

}

private static (int? row, int? column) FindInEncryptionTable(char[,] encryptionTable,

char ch)

{

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

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

if (encryptionTable[i, j] == ch)

return (i, j);

return (null, null);

}

private static string ProcessText(string text, char[,] encryptionTable, bool isEncryption)

{

StringBuilder builder = new();

for (int i = 0; i < text.Length - 1; i += 2)

{

(int? firstRow, int? firstColumn) = FindInEncryptionTable(encryptionTable,

text[i]);

(int? secondRow, int? secondColumn) = FindInEncryptionTable(encryptionTable,

text[i + 1]);

HandleEncryption(builder, encryptionTable, firstRow, firstColumn,

secondRow, secondColumn, isEncryption);

}

return builder.ToString();

}

private static void HandleEncryption(StringBuilder builder, char[,] encryptionTable,

int? firstRow, int? firstColumn, int? secondRow, int? secondColumn, bool isEncryption)

{

if (firstRow.HasValue && firstColumn.HasValue

&& secondRow.HasValue && secondColumn.HasValue)

{

if (firstRow == secondRow)

{

builder.Append(ProcessRowsEqual(encryptionTable, firstRow.Value,

firstColumn.Value, isEncryption));

builder.Append(ProcessRowsEqual(encryptionTable, secondRow.Value,

secondColumn.Value, isEncryption));

}

else if (firstColumn == secondColumn)

{

builder.Append(ProcessColumnsEqual(encryptionTable, firstRow.Value,

firstColumn.Value, isEncryption));

builder.Append(ProcessColumnsEqual(encryptionTable, secondRow.Value,

secondColumn.Value, isEncryption));

}

else

{

builder.Append(encryptionTable[firstRow.Value, secondColumn.Value]);

builder.Append(encryptionTable[secondRow.Value, firstColumn.Value]);

}

}

else

HandleUnknownCharacters(builder, encryptionTable, firstRow, firstColumn,

secondRow, secondColumn);

}

private static void HandleUnknownCharacters(StringBuilder builder, char[,] encryptionTable,

int? firstRow, int? firstColumn, int? secondRow, int? secondColumn)

{

if (firstRow == null && firstColumn == null

&& secondRow == null && secondColumn == null)

{

builder.Append(UNKNOWN\_CHAR);

builder.Append(UNKNOWN\_CHAR);

}

else if (firstRow == null && firstColumn == null

&& secondRow.HasValue && secondColumn.HasValue)

{

builder.Append(UNKNOWN\_CHAR);

builder.Append(encryptionTable[secondRow.Value, secondColumn.Value]);

}

else if (firstRow.HasValue && firstColumn.HasValue

&& secondRow == null && secondColumn == null)

{

builder.Append(encryptionTable[firstRow.Value, firstColumn.Value]);

builder.Append(UNKNOWN\_CHAR);

}

}

private static char ProcessRowsEqual(char[,] encryptionTable, int row, int column,

bool isEncryption)

{

if (isEncryption)

return GetRightSymbol(encryptionTable, row, column);

else

return GetLeftSymbol(encryptionTable, row, column);

}

private static char GetRightSymbol(char[,] encryptionTable, int row, int column)

{

if (column == encryptionTable.GetLength(1) - 1)

return encryptionTable[row, 0];

else

return encryptionTable[row, column + 1];

}

private static char GetLeftSymbol(char[,] encryptionTable, int row, int column)

{

if (column == 0)

return encryptionTable[row, encryptionTable.GetLength(1) - 1];

else

return encryptionTable[row, column - 1];

}

private static char ProcessColumnsEqual(char[,] encryptionTable, int row, int column,

bool isEncryption)

{

if (isEncryption)

return GetDownSymbol(encryptionTable, row, column);

else

return GetUpSymbol(encryptionTable, row, column);

}

private static char GetDownSymbol(char[,] encryptionTable, int row, int column)

{

if (row == encryptionTable.GetLength(0) - 1)

return encryptionTable[0, column];

else

return encryptionTable[row + 1, column];

}

private static char GetUpSymbol(char[,] encryptionTable, int row, int column)

{

if (row == 0)

return encryptionTable[encryptionTable.GetLength(0) - 1, column];

else

return encryptionTable[row - 1, column];

}

}

}

## 3.3 HomophonicEncryptionService

using System.Text;

using EncryptionService.Core.Interfaces;

using EncryptionService.Core.Models.HomophonicEncryption;

namespace EncryptionService.Core.Services

{

public class HomophonicEncryptionService

: IEncryptionService<HomophonicEncryptionResult, HomophonicEncryptionKey,

Dictionary<char, int[]>>

{

public HomophonicEncryptionResult Encrypt(string text,

HomophonicEncryptionKey encryptionKey)

{

text = text.ToUpper();

Random random = new();

StringBuilder builder = new();

Dictionary<char, List<int>> keyCopy = GenerateKeyCopy(encryptionKey);

foreach (char ch in text)

{

if (keyCopy.TryGetValue(ch, out var listNumbers))

{

if (listNumbers.Count != 0)

{

int randomIndex = random.Next(0, listNumbers.Count);

builder.Append(listNumbers[randomIndex].ToString("D3"));

listNumbers.RemoveAt(randomIndex);

}

else

{

int[] arrNumbers = encryptionKey.Key[ch];

int randomIndex = random.Next(0, arrNumbers.Length);

builder.Append(arrNumbers[randomIndex].ToString("D3"));

}

}

}

return new HomophonicEncryptionResult(builder.ToString(), encryptionKey.Key);

}

public HomophonicEncryptionResult Decrypt(string encryptedText,

HomophonicEncryptionKey encryptionKey)

{

StringBuilder builder = new();

for (int i = 0; i < encryptedText.Length; i += 3)

{

int number = int.Parse(encryptedText.Substring(i, 3));

foreach (var kvp in encryptionKey.Key)

{

if (kvp.Value.Contains(number))

{

builder.Append(kvp.Key);

break;

}

}

}

return new HomophonicEncryptionResult(builder.ToString(), encryptionKey.Key);

}

private static Dictionary<char, List<int>> GenerateKeyCopy(

HomophonicEncryptionKey encryptionKey)

{

Dictionary<char, List<int>> keyCopy = [];

foreach (var kvp in encryptionKey.Key)

keyCopy[kvp.Key] = [.. kvp.Value];

return keyCopy;

}

}

}

# Висновок

У ході виконання лабороторної роботи було визначено симетричне шифрування та шифри замін. Також було програмно реалізовано алгоритми і методи шифрування з різними видами замін, а саме: гасло-шифр, шифр «чесна гра»,шифр за допомогою системи омофонів 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.