**Лабораторная работа № 2**

**ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ**

**ШИФРОВ НА ОСНОВЕ ПОДСТАНОВКИ**

**(ЗАМЕНЫ) СИМВОЛОВ**

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров.

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости подстановочных шифров.

2. Ознакомиться с особенностями реализации и свойствами различных подстановочных шифров на основе готового программного средства (L\_LUX).

3. Разработать приложение для реализации указанных преподавателем методов подстановочного зашифрования/расшифрования.

4. Выполнить исследование криптостойкости шифров на основе статистических данных о частотах появления символов в исходном и зашифрованном сообщениях.

5. Оценить скорость зашифрования/расшифрования реализованных способов шифров.

6. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Практическое задание.**

Рекомендация! Перед выполнением практического задания можно познакомиться с особенностями работы программного средства L\_LUX, реализующего подстановочные (и другие) методы зашифрования/расшифрования текстовой информации. Основная часть окна – текстовый редактор, в котором можно набирать текст либо размещать скопированный фрагмент из другого текстового документа. Здесь же отображается зашифрованный текст, а также сформированные программой распределения частот (в виде гистограмм) для исходного и зашифрованного текстов. На рис. 1.1 представлено основное диалоговое окно программы после запуска исполнительного файла L\_LUX.EXE.

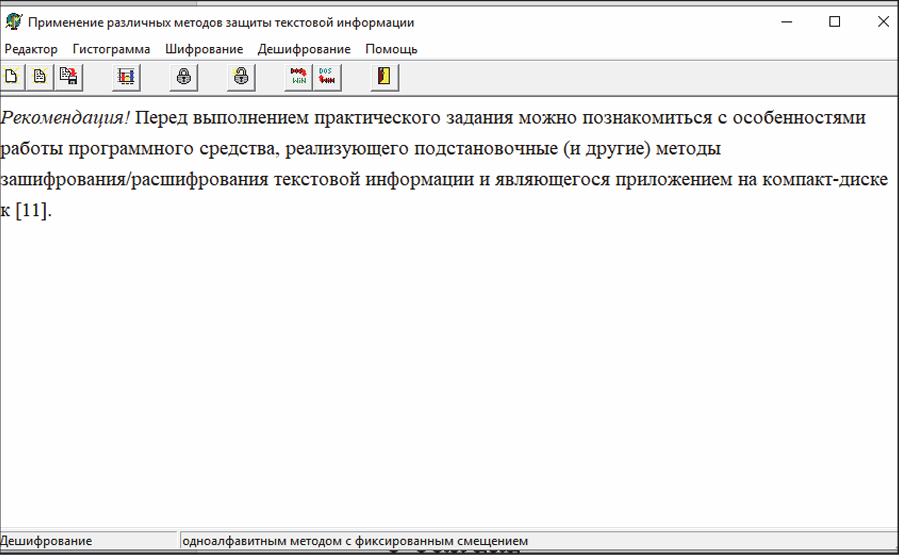


Рисунок 1.1. Основное диалоговое окно программного средства L\_LUX

На рисунке 1.2 для примера и сравнения приведены гистограммы для исходного (в окне редактора на рисуноке 1.1) и зашифрованного документов (обратим внимание, что отдельные буквы – строчные и прописные – рассматриваются здесь как разные, что не соответствует традиционному подходу).

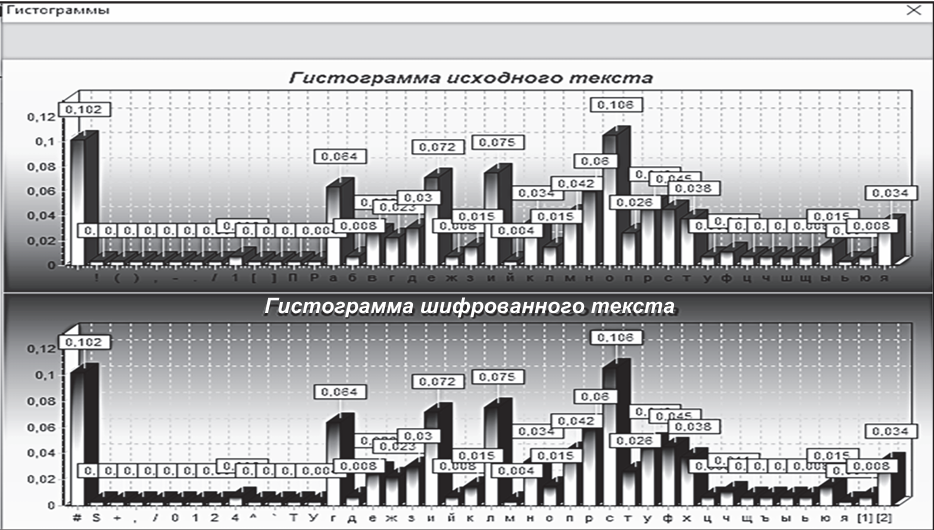


Рисунок 1.2. Гистограммы для исходного и зашифрованного текстовых документов

Шифрование немецкого текста из задания 1, методом смещения, результат работы программы L\_LUX представлен на рисунке 1.3.

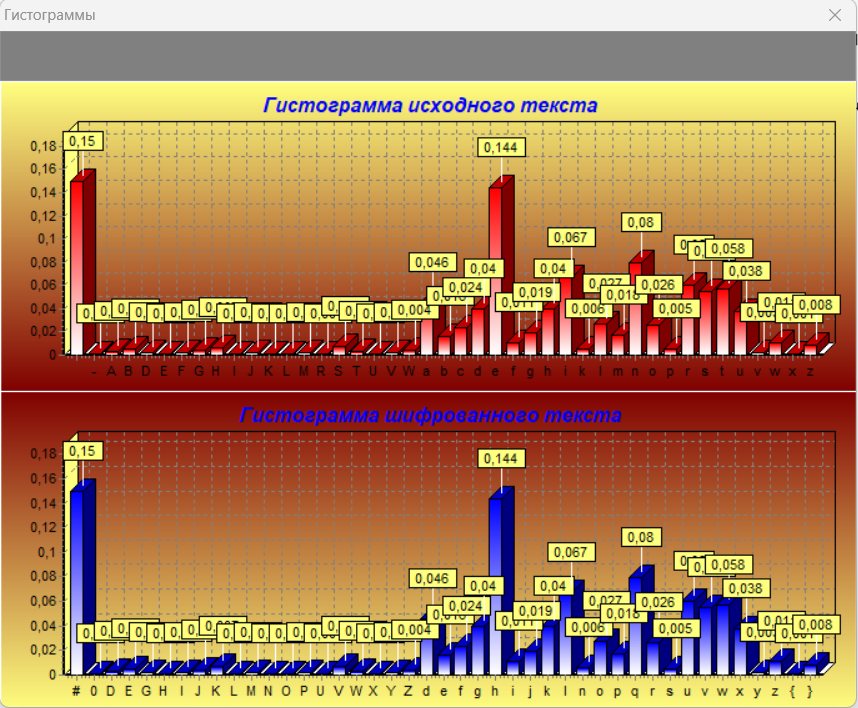


Рисунок 1.3. Гистограммы для исходного и зашифрованного текстовых документов из задания 1

Основное задание 1. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

• выполнять зашифрование/расшифрование текстовых документов (объемом не менее 5 тысяч знаков), созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы (варианты задания представлены в табл. 1.1);

• сформировать гистограммы частот появления символов для исходного и зашифрованного сообщений;

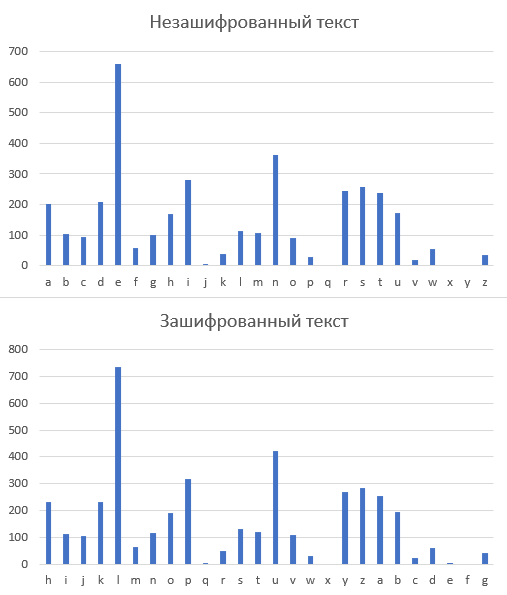


Рисунок 1.3. Гистограммы для исходного и зашифрованного текстовых документов из задания 1

• оценить время выполнения операций зашифрования/расшифрования (напоминание: во многих языках программирования есть встроенные методы для замеров времени; при отсутствии такового в используемом языке можно воспользоваться разностью двух дат (например, в миллисекундах: время после выполнения про граммы – время до начала выполнения преобразования)). При анализе полученных гистограмм можно сопоставить полученные данные с аналогичными результатами выполнения лабораторной работы № 2 из [2].

Код программы для шифрования по алгоритму цезаря для задания 1 представлен в листинге 1.1.

function stringTransformation(

line: string,

k: number,

alphabetLength: number,

alphabetDivisionCode: number,

isEncryption: boolean

): string {

let outputLine = "";

let symbolCode: number;

for (let char of line) {

const charCode = char.charCodeAt(0);

const charIndex = charCode - alphabetDivisionCode;

if (charIndex < 0 || charIndex >= alphabetLength) {

outputLine += char;

continue;

}

if (isEncryption) {

symbolCode = (charIndex + k + alphabetLength) % alphabetLength;

} else {

symbolCode = (charIndex - k + alphabetLength) % alphabetLength;

}

outputLine += String.fromCharCode(symbolCode + alphabetDivisionCode);

}

return outputLine;

}

Листинг 1.1 смещение цезаря



Реализация Шифра Трисемуса представлена в листинге 1.2.

function transformChar(char: string, isEncryption: boolean): string {

let pos = TABLE.indexOf(char);

if (pos === -1)

return char;

let row = Math.floor(pos / 5);

let col = pos % 5;

if (isEncryption) {

row = (row + 1) % 5;

} else {

row = (row - 1 + 5) % 5;

}

return TABLE[row \* 5 + col];

}

Листинг 1.2 алгоритм шифрации по таблице Трисемуса

Формирование таблицы Трисемуса представлена в листинге 1.3.

function generateTrisemusTable(alphabet: string, keyword: string): string {

let uniqueKey = "";

for (let char of keyword) {

if (!uniqueKey.includes(char)) uniqueKey += char;}

let remainingLetters = alphabet.split('').filter(c => !uniqueKey.includes(c));

let fullTable = uniqueKey + remainingLetters.join('');

return fullTable;}

Листинг 1.3 формирование таблицы Трисемуса

Если указанный в таблице язык исходного текста не известен разработчику программного средства, можно взять документ на требуемом языке и воспользоваться доступным электронным переводчиком (возникающие при этом отдельные семантические неточности не следует считать существенным недостатком выполняемого анализа).

Функция для шифрования строки по алгоритму Виженера представлена в листинге 1.4.

function stringEncryption(

line: string,

key: string,

keyIndex: number,

alphabetDivisionCode: number,

alphabetLength: number

): { encryptedText: string, newKeyIndex: number } {

let outputLine = "";

for (const char of line) {

const keyChar = key[keyIndex % key.length];

const charCode = char.charCodeAt(0);

const keyCharCode = keyChar.charCodeAt(0);

let symbolCode;

if (char === " ") {

symbolCode = (charCode + keyCharCode - alphabetDivisionCode) % alphabetLength;

} else {

symbolCode = (charCode - alphabetDivisionCode + keyCharCode - alphabetDivisionCode) % alphabetLength;

}

if (symbolCode < 0) symbolCode += alphabetLength;

outputLine += String.fromCharCode(symbolCode + alphabetDivisionCode);

keyIndex++;

}

return { encryptedText: outputLine, newKeyIndex: keyIndex };

}

Листинг 1.4 Функция для шифрования строки по методу Виженера

Расшифрование текста, зашифрованного шифром Виженера, не зная ключа и его длинны. Написанный код представляет собой программу для расшифровки текста, зашифрованного методом Виженера, без предварительного знания ключа. Вначале функция analyzeKeyLength определяет вероятную длину ключа с использованием индекса совпадений, анализируя регулярности символов в тексте. Затем функция findKey извлекает сам ключ, используя частотный анализ, предполагая, что наиболее часто встречающийся символ связан с буквой "e" из английского алфавита. После этого функция decrypt применяет найденный ключ для дешифровки текста, а результат сохраняется в файл. Код также автоматически удаляет повторяющиеся части ключа, чтобы вернуть исходный уникальный ключ.

Функция, определяющая длину ключа приведена в листинге 1.5.

function analyzeKeyLength(text: string, maxKeyLength: number): number {

let probableKeyLength = 1;

let maxCoincidenceIndex = 0;

for (let keyLength = 1; keyLength <= maxKeyLength; keyLength++) {

// разбиение на строки, где в каждой строке находятся символы относительно длины ключа

const substrings = Array.from({ length: keyLength }, (\_, i) => text.split('').filter((\_, index) => index % keyLength === i).join(''));

const coincidenceIndex = substrings.reduce((sum, substr) => {

// подсчёт частоты символов в подстроке

const freq = substr.split('').reduce((acc, char) => ({ ...acc, [char]: (acc[char] || 0) + 1 }), {} as Record<string, number>);

const substrLength = substr.length;

return sum + Object.values(freq).reduce((acc, f) => acc + (f \* (f - 1)) / (substrLength \* (substrLength - 1)), 0);

}, 0) / substrings.length;

// обновление длины ключа если частотный анализ показал лучший результат

if (coincidenceIndex > maxCoincidenceIndex) {

maxCoincidenceIndex = coincidenceIndex;

probableKeyLength = keyLength;

}

}

return probableKeyLength;

}

Листинг 1.5 Функция для определения длины ключа

Функция для нахождения ключа, приведена в листинге 1.6.

function findKey(text: string, keyLength: number): string {

let key = '';

for (let i = 0; i < keyLength; i++) {

// разбиение на строки, где в каждой строке находятся символы относительно длины ключа

const substring = text.split('').filter((\_, index) => index % keyLength === i).join('');

// подсчёт частоты символов для каждой строки

const freq = substring.split('').reduce((acc, char) => ({ ...acc, [char]: (acc[char] || 0) + 1 }), {} as Record<string, number>);

// нахождение самого частовстречающегося символа

const mostFrequentChar = Object.entries(freq).reduce((a, b) => (b[1] > a[1] ? b : a))[0];

// вычисление символа ключа, используя сдвиг относительно буквы e

const keyChar = String.fromCharCode(((mostFrequentChar.charCodeAt(0) - 'e'.charCodeAt(0) + ALPHABET\_LENGTH) % ALPHABET\_LENGTH) + 'a'.charCodeAt(0));

key += keyChar;

}

// удаление дублирующейся части ключа

for (let length = 1; length <= key.length; length++) {

const candidate = key.slice(0, length);

if (candidate.repeat(Math.ceil(key.length / length)).startsWith(key)) {

return candidate;

}

}

return key;

}

Листинг 1.6 Функция для нахождения ключа

2. Результаты оформить в виде отчета по установленным правилам.

Таблица 1.1 Варианты заданий

