Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

Институт ИВТ

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе №1 по теме

«Ручные шифры»

по дисциплине

«Защита информации»

Выполнил:

ст-т гр. Б08-191-2 Хасанов М.П.

Принял преподаватель: Старыгин А .В.

Ижевск 2019

СОДЕРЖАНИЕ

1. Содержание задания3

2. Скриншоты программы4

3. Исходный код7

1. СОДЕРЖАНИЕ ЗАДАНИЯ

Разработать программу осуществляющую шифрование и расшифрование одним из ручных шифров.

Требования:

● Наглядность процесса шифрования\расшифрования

● Возможность зашифровать данные с одним ключом, а расшифровать с другим

● Возможность изменить шифрованные данные перед расшифрованием

Вариант 7. Магический квадрат.

2. СКРИНШОТЫ ПРОГРАММЫ

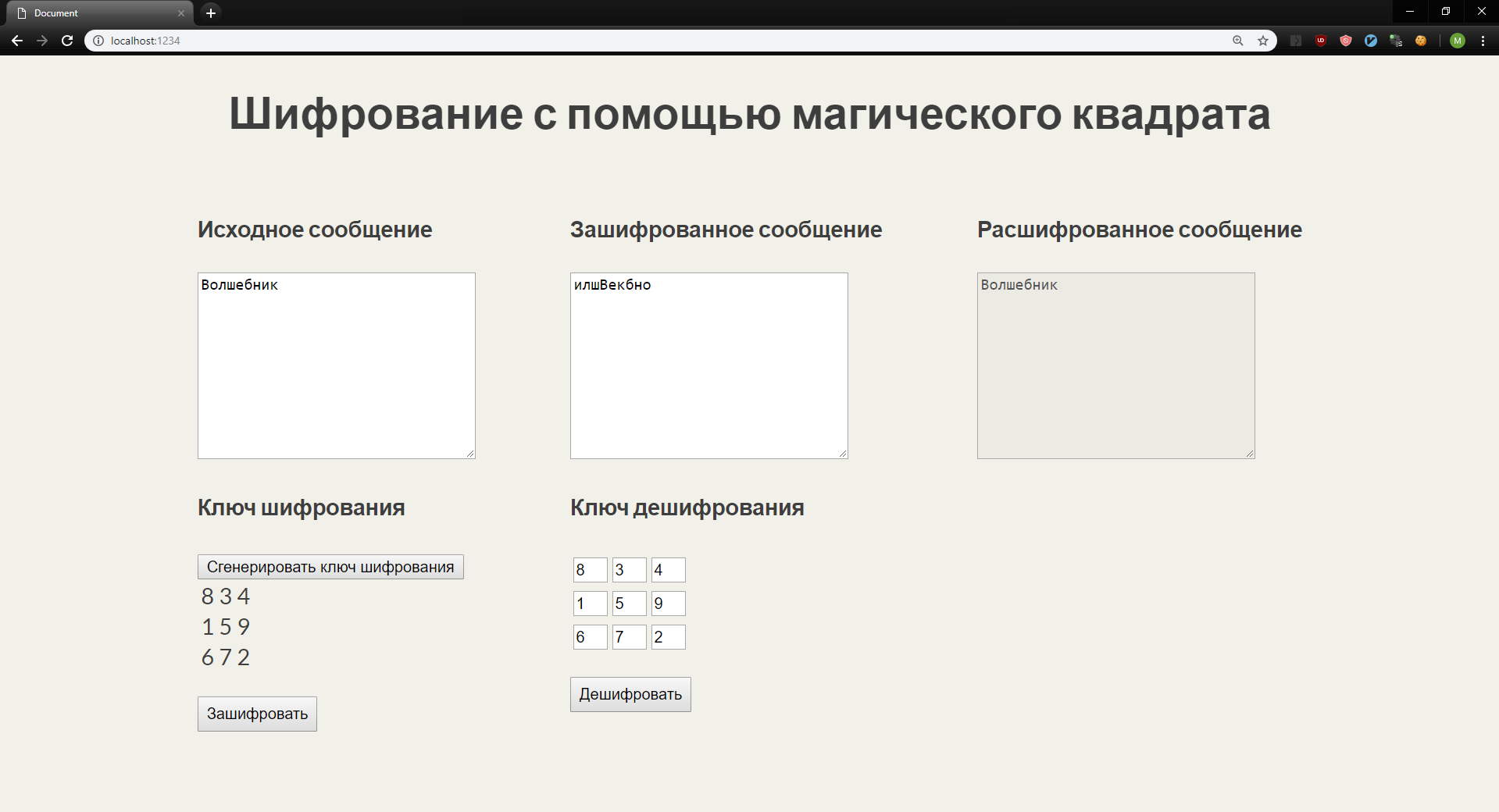
Зашифруем сообщение “Волшебник”, используя ключ шифрования изображенный на рис. 1. Полученное зашифрованное сообщение “илшВекбно” расшифруем. Полученное после расшифровки сообщение совпадает с исходным.

Рис. 1

Теперь изменим ключ дешифрования поменяв местами индексы 6 и 4 (рис. 2). После расшифровки получим сообщение “Волбешник”.

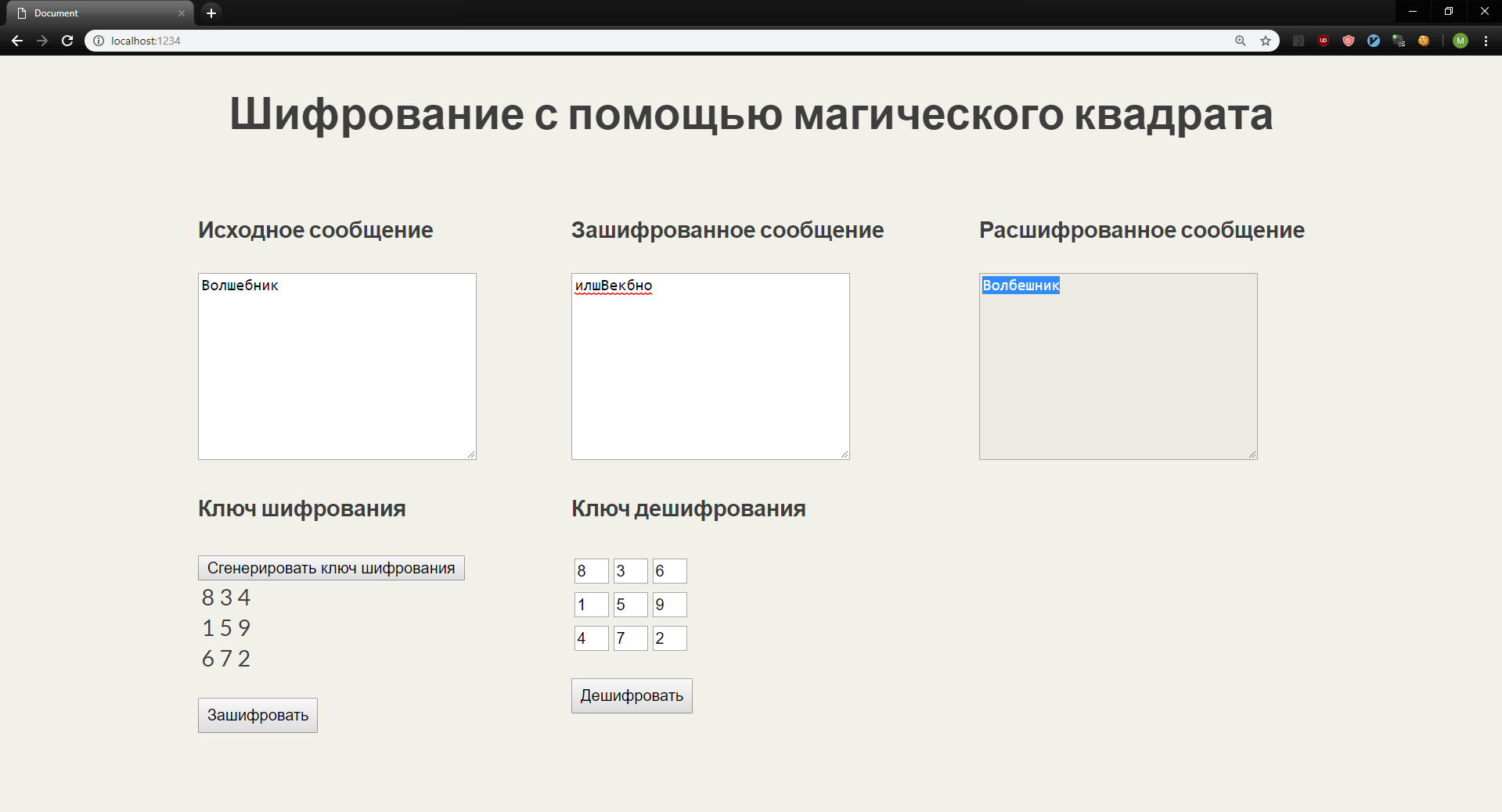


Рис. 2

3. ТЕКСТ ПРОГРАММЫ

3.1 index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<script defer src="/index.js"></script>

<link rel="stylesheet" href="/style.css">

<link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet">

</head>

<body>

<h1 class="header">Шифрование с помощью магического квадрата</h1>

<main>

<div class="source">

<h4>Исходное сообщение</h4>

<textarea placeholder="Исходное сообщение" name="source" id="source" cols="30" rows="10"></textarea>

<div class="tableWrapper">

<h4>Ключ шифрования</h4>

<div><button id="generateKey" type="button">Сгенерировать ключ шифрования</button></div>

<div id="tableRootEncrypt">

</div>

<div class="control"> <button type="button" id="encrypt">Зашифровать</button></div>

</div>

</div>

<div class="crypted">

<h4>Зашифрованное сообщение</h4>

<textarea placeholder="Зашифрованное сообщение" name="crypted" id="crypted" cols="30" rows="10"></textarea>

<div class="tableWrapper">

<h4>Ключ дешифрования</h4>

<div id="tableRootDecrypt">

</div>

<div class="control"> <button type="button" id="decrypt">Дешифровать</button></div>

</div>

</div>

<div>

<h4>Расшифрованное сообщение</h4>

<textarea placeholder="Расшифрованное сообщение" name="encrypted" id="encrypted" cols="30" rows="10"

disabled></textarea>

</div>

</main>

</body>

</html>

3.2 index.js

import { generateMagicSquare } from './generateSquare';

import { generateTable } from './generateTable';

import { flatted } from './utils';

const tableRootEncrypt = document.getElementById('tableRootEncrypt');

const tableRootDecrypt = document.getElementById('tableRootDecrypt');

const sourceArea = document.getElementById("source");

const cryptedArea = document.getElementById("crypted");

const encryptedArea = document.getElementById("encrypted");

const encryptButton = document.getElementById("encrypt");

const decryptButton = document.getElementById("decrypt");

const generateKeyButton = document.getElementById("generateKey");

const clearNode = node => {

while (node.firstChild) {

node.removeChild(node.firstChild);

};

return node;

}

let currentEncryptKey = [];

let currentDecryptKey = [];

function generateDecryptTable() {

const table = generateTable(currentDecryptKey, (value, row, column) => {

const input = document.createElement('input');

input.addEventListener('change', (e) => {

currentDecryptKey[row][column] = parseInt(e.target.value);

});

input.value = value;

input.className = 'input';

return input;

});

clearNode(tableRootDecrypt).appendChild(table);

}

encryptButton.addEventListener('click', () => {

const text = sourceArea.value;

const flattenKey = flatted(currentEncryptKey);

const result = flattenKey.map(idx => text[idx - 1]).join('');

cryptedArea.value = result;

currentDecryptKey = JSON.parse(JSON.stringify(currentEncryptKey));

generateDecryptTable();

});

decryptButton.addEventListener('click', () => {

const text = cryptedArea.value;

const flattenKey = flatted(currentDecryptKey);

const result = text.split('').reduce((acc, char, i) => {

acc[flattenKey[i] - 1] = char;

return acc;

}, Array(text.length)).join('');

encryptedArea.value = result;

});

generateKeyButton.addEventListener('click', () => {

const text = sourceArea.value;

const size = Math.sqrt(

Math.pow(

Math.floor(Math.sqrt(text.length)),

2

)

);

currentEncryptKey = generateMagicSquare(size);

const table = generateTable(currentEncryptKey);

clearNode(tableRootEncrypt).appendChild(table);

});

3.3 generateTable.js

const defaultGenerateItem = value => {

const span = document.createElement('span');

span.textContent = value;

return span;

}

export function generateTable(square = [], generateItem = defaultGenerateItem) {

const size = square.length;

const root = document.createElement('table');

for (let row = 0; row < size; row++) {

const tr = document.createElement('tr');

for (let column = 0; column < size; column++) {

const td = document.createElement('td');

const value = square[row][column];

const content = generateItem(value, row, column);

td.appendChild(content);

tr.appendChild(td);

}

root.appendChild(tr);

}

return root;

}

3.4 generateSquare.js

import { zipWith, enumFromTo, chunksOf, compose, transpose } from './utils';

export const generateMagicSquare = n =>

n % 2 !== 0 ? (

compose([cycled, transpose, cycled, enumSquare])(n)

) : [];

const enumSquare = n =>

chunksOf(n, enumFromTo(1, n \* n));

const cycled = rows => {

const d = Math.floor(rows.length / 2);

return zipWith(listCycle, enumFromTo(d, -d), rows)

};

const listCycle = (n, xs) => {

const d = -(n % xs.length);

return (d !== 0 ? xs.slice(d)

.concat(xs.slice(0, d)) : xs);

};

3.5 utils.js

// chunksOf :: Int -> [a] -> [[a]]

const chunksOf = (n, xs) =>

xs.reduce((a, \_, i, xs) =>

i % n ? a : a.concat([xs.slice(i, i + n)]), []);

// compose :: [(a -> a)] -> (a -> a)

const compose = fs => x => fs.reduceRight((a, f) => f(a), x);

// enumFromTo :: Int -> Int -> Maybe Int -> [Int]

const enumFromTo = (m, n, step) => {

const d = (step || 1) \* (n >= m ? 1 : -1);

return Array.from({

length: Math.floor((n - m) / d) + 1

}, (\_, i) => m + (i \* d));

};

// intercalate :: String -> [a] -> String

const intercalate = (s, xs) => xs.join(s);

// min :: Ord a => a -> a -> a

const min = (a, b) => b < a ? b : a;

// transpose :: [[a]] -> [[a]]

const transpose = xs =>

xs[0].map((\_, iCol) => xs.map(row => row[iCol]));

// unlines :: [String] -> String

const unlines = xs => xs.join('\n');

// zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]

const zipWith = (f, xs, ys) =>

Array.from({

length: min(xs.length, ys.length)

}, (\_, i) => f(xs[i], ys[i]));

const flatted = (arr = []) => arr.reduce((acc, cur) => acc.concat(cur), []);

export {

chunksOf,

enumFromTo,

intercalate,

zipWith,

transpose,

compose,

unlines,

flatted

};

3.6 style.css

body {

margin: 0 auto;

width: 1024px;

font-family: 'Lato', sans-serif;

font-size: 16px;

color: #3e3e3e;

background: #eceae0b3;

}

h1 {

text-align: center;

margin-bottom: 40px;

}

main {

display: flex;

justify-content: space-around;

}

.control {

margin-top: 20px;

}

.control>\* {

margin-right: 40px;

height: 30px;

}

.input {

width: 25px;

}