МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА — Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

(наименование института, филиала)

Кафедра КБ-14 «Цифровые технологии обработки данных»

(наименование кафедры)

Практическая работа

по дисциплине Криптографическая защита информации

(наименование дисциплины)

Выполнил:

БСБО-05-20

Верхотуров В. С.

Москва — 2024 г.

СОДЕРЖАНИЕ

1 ШИФР МНОГОАЛФАВИТНОЙ ЗАМЕНЫ ВИЖИНЕРА

1.1 Задание

Для повышения стойкости шифра используют многоалфавитные замены, в которых для замены символов исходного текста используются символы нескольких алфавитов.

Одной из разновидностей такого метода является схема шифрования Вижинера. Шифр очень устойчивый к вскрытию. Таблица Вижинера представляет собой квадратную матрицу с n2 элементами, где n – число символов используемого алфавита. Каждая строка получена циклическим сдвигом алфавита на один символ

При шифровании сообщения его выписывают в строку, а под ним буквенный ключ. Если ключ оказался короче сообщения, то его циклически повторяют. Шифровку получают, находя символ в колонке таблицы по букве текста и строке, соответствующей букве ключа.

Например:

Сообщение П Р И Е З Ж А Ю Ш Е С Т О Г О

Ключ А Г А В А А Г А В А А Г А В А

Шифровка П Н И Г З Ж Ю Ю ЮА Е О Т М

Предыдущие шифры называются монограммными, так как шифрование ведется по одной букве. Шифрование по 2 букве называются биграммными.

1.2 Выполнение практической

Результат практической: `https://crypto-tasks.vercel.app/task1`.

Репозиторий `https://github.com/ValeryVerkhoturov/crypto-tasks`.

Листинг 1 – Шифр многоалфавитной замены Вижинера

```
1 export class VigenereCipher {
2         private alphabet: string = АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ'';
3         private mod: number = this.alphabet.length;
4
5         encrypt(text: string, key: string): string {
6                 return this.processText(text, key, 'encrypt');
7         }
```

```
8
9           decrypt(text: string, key: string): string {
10                  return this.processText(text, key, 'decrypt');
11          }
12
13          private processText(text: string, key: string, mode: 'encrypt' | '
                decrypt'): string {
14                  let processedText = '';
15                  let keyIndex = 0;
16
17                  text = text.toUpperCase();
18                  key = key.toUpperCase();
19
20                  for (let i = 0; i < text.length; i++) {
21                          const char = text[i]
22                          if (this.alphabet.includes(char)) {
23                                  const charIndex = this.alphabet.indexOf(char);
24                                  const keyChar = key[keyIndex % key.length];
25                                  const keyCharIndex = this.alphabet.indexOf(
                                        keyChar);
26
27                                  if (mode === 'encrypt') {
28                                          processedText += this.alphabet[(
                                                charIndex + keyCharIndex) % this.
                                                mod];
29                                  } else {
30                                          let decodeIndex = (charIndex -
                                                keyCharIndex) % this.mod;
31                                          if (decodeIndex < 0) {
32                                                  decodeIndex += this.mod;
33                                          }
34                                          processedText += this.alphabet[
                                                decodeIndex];
35                                  }
36
37                                  keyIndex++;
38                          } else {
39                                  processedText += char;
40                          }
41                  }
42
43                  return processedText;
44          }
45  }
```

2 МАГИЧЕСКИЙ КВАДРАТ

2.1 Задание

Магический квадрат - это квадратная таблица с вписанными в клетки последовательными натуральными числами от 1, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и тоже число.

Чтобы зашифровать открытый текст с помощью такого квадрата, нужно пронумеровать все буквы в фразе по порядку и вставить их в квадрат вместо соответствующих цифр.

2.2 Выполнение практической

Результат практической: `https://crypto-tasks.vercel.app/task2`.

Репозиторий `https://github.com/ValeryVerkhoturov/crypto-tasks`.

Листинг 2 – Шифр Магический квадрат

```
export class MagicSquareCipher {
        private magicSquare: number[][]
        private magicSquareDimensions: number

        constructor(magicSquare: number[][]) {
                this.magicSquare = magicSquare;
                this.magicSquareDimensions = magicSquare.length;
        }

        encrypt(text: string): string {
                let encryptedText = Array(text.length).fill(null);

                for (let i = 0; i < text.length; i++) {
                        const row = Math.floor(i / this.magicSquareDimensions)
                            ;
                        const col = i % this.magicSquareDimensions;
                        const newPos = this.magicSquare[row][col];
                        console.log(this.magicSquare, newPos)
                        if (newPos < text.length) {
                                encryptedText[newPos] = text[i];
                        }
                }

                return encryptedText.join("");
        }

```

```
26          decrypt(encryptedText: string): string {
27                  let decryptedText = Array(encryptedText.length).fill(null);
28
29                  for (let i = 0; i < encryptedText.length; i++) {
30                          const row = Math.floor(i / this.magicSquareDimensions)
                                ;
31                          const col = i % this.magicSquareDimensions;
32                          const originalPos = this.magicSquare[row][col];
33                          if (originalPos < encryptedText.length) {
34                                  decryptedText[i] = encryptedText[originalPos];
35                          }
36                  }
37
38                  return decryptedText.join("");
39          }
40  }
```

3 АЛГОРИТМ КУЗНЕЧИК: ШИФРОВАНИЕ/РАСШИФРОВАНИЕ ДАННЫХ В РЕЖИМЕ ПРОСТОЙ ЗАМЕНЫ

3.1 Задание

Составить алгоритм программы шифрования по выбранному методу. Составить программу шифрования по соответствующему заданию. Составить алгоритм программы расшифрования по выбранному методу. Составить программу расшифрования по соответствующему заданию.

3.2 Выполнение практической

Результат практической: `https://crypto-tasks.vercel.app/task3`.

Репозиторий `https://github.com/ValeryVerkhoturov/crypto-tasks`.

Листинг 3 – Шифр Магический квадрат

```
let tabl_notlin: Buffer=Buffer.from([
252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35, 197, 4,
77, 233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187, 20, 205,
95, 193, 249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66, 139, 1,
142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152, 127, 212,
211, 31, 235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253, 58,
206, 204, 181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156, 183,
93, 135, 21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157,
158, 178, 177, 50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128, 195,
189, 13, 87, 223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246, 124,
34, 185, 3, 224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78,
51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100, 159,
38, 65, 173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105, 213,
149, 59, 7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136, 217,
231, 137, 225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144, 202, 216,
133, 97, 32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208, 190, 229,
108, 82, 89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57, 75,
    99, 182
])
let constants1:Buffer = Buffer.from([148, 32, 133, 16, 194, 192, 1, 251,
1, 192, 194, 16, 133, 32, 148, 1]);

export class Kuznec{
        C: Buffer[];
        iterKey: Buffer[];

```

```
26          constructor(){
27                  this.iterKey = [];
28                  this.C = [];
29                  this.KeyGen();
30          };
31
32          GaloisMult(value1:number, value2:number){
33                  let gm: number = 0;
34                  let hi_bit: number;
35                  for(let i = 0; i < 8; i++){
36                          if(value2 & 1){
37                                  gm ^= value1;
38                          }
39                          hi_bit = value1 & 0x80;
40                          value1 <<= 1;
41                          if(hi_bit){
42                                  value1 ^= 0xc3;
43                          }
44                          value2 >>= 1;
45                  }
46                  return gm;
47          }
48
49          XSL(plaintext: Buffer, j: number){
50                  plaintext = this.XOR(plaintext, this.iterKey[j]);
51                  plaintext = this.S(plaintext);
52                  plaintext = this.L(plaintext);
53                  return plaintext;
54          }
55
56          LrSrX(cipherText: Buffer, j: number){
57                  cipherText = this.L_rev(cipherText);
58                  cipherText = this.S_rev(cipherText);
59                  cipherText = this.XOR(cipherText, this.iterKey[j]);
60                  return cipherText;
61          }
62
63          Decryption(cipherText : Buffer){
64                  cipherText = this.XOR(cipherText, this.iterKey[this.iterKey.
                        length - 1]);
65
66                  for(let i = this.iterKey.length - 2; i >= 0; i--){
67                          cipherText = this.LrSrX(cipherText, i);
68                  }
69                  return cipherText;
70          }
```

```
71
72          XOR(a: Buffer, b: Buffer){
73                  let result: Buffer = Buffer.alloc(16);
74                  for(let i = 0; i < 16; i++){
75                          result[i] = a[i] ^ b[i];
76                  }
77                  return result
78          }
79
80          Encryption(plaintext : Buffer){
81                  for(let i = 0; i < this.iterKey.length - 1; i++){
82                          plaintext = this.XSL(plaintext, i);
83                  }
84                  plaintext = this.XOR(plaintext, this.iterKey[this.iterKey.
                        length - 1]);
85                  return plaintext;
86          }
87
88          ConstGen(){
89                  this.C=[];
90                  for(let i = 1; i <= 32; i++){
91                          let z: number =i;
92                          let m: Buffer=Buffer.from
                                ([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0])
93                         m[15]=z;
94                          this.C.push(this.L(m));
95                  }
96                  return this.C;
97          }
98
99          GOSTF(key1: Buffer, key2:Buffer, iter_const: Buffer){
100                 let internal: Buffer = Buffer.alloc(0);
101                 let outKey2 = key1;
102                 internal = this.XOR(key1, iter_const);
103                 internal = this.S(internal);
104                 internal = this.L(internal);
105
106                 let outKey1: Buffer = Buffer.alloc(0);
107                 outKey1 = this.XOR(internal, key2);
108
109                 let key: Buffer[] = [];
110                 key.push(outKey1);
111                 key.push(outKey2);
112                 return key;
113         }
114         KeyGen(){
```

```
115                 let temp: number[] = [];
116                 for(let i = 0; i < 32; ++i){
117                         temp.push(Math.floor(Math.random() * 255))
118                 }
119                 this.GetKeys(Buffer.from(temp));
120         }
121         GetKeys(masterkey: Buffer){
122                 let key1: Buffer = Buffer.alloc(16); let key2: Buffer = Buffer
                        .alloc(16);
123                 for(let i = 0; i < 16; i++){
124                         key1[i] = masterkey[i];
125                 }
126                 for(let i = 16; i < 32; i++){
127                         key2[i - 16] = masterkey[i];
128                 }
129                 let i: number;
130
131                 let iter12: Buffer[] = [Buffer.alloc(0), Buffer.alloc(0)];
132                 let iter34: Buffer[] = [Buffer.alloc(0), Buffer.alloc(0)];
133                 this.ConstGen();
134                 this.iterKey[0] = key1;
135                 this.iterKey[1] = key2;
136                 iter12[0] = key1;
137                 iter12[1] = key2;
138
139                 for(i = 0; i < 4; i++){
140                         for(let j = 0; j < 8; j +=2 ){
141                                 iter34 = this.GOSTF(iter12[0], iter12[1], this
                                        .C[j + 8*i]);
142                                 iter12 = this.GOSTF(iter34[0], iter34[1], this
                                        .C[j + 1 + 8*i]);
143                         }
144
145                         this.iterKey[2 * i + 2] = iter12[0];
146                         this.iterKey[2 * i + 3] = iter12[1];
147                 }
148
149                 return this.iterKey;
150         }
151
152         GOSTR(bytes: Buffer){
153                 let r: Buffer = Buffer.from([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0]);
154                 let a15: number = 0;
155                 for(let i = 15; i >= 1; i--){
156                         r[i] = bytes[i-1];
```

```
157                 }
158                 for(let i = 0; i <16; i++){
159                         a15 ^= this.GaloisMult(constants1[i], bytes[i]);
160
161                 }
162                 r[0] = a15;
163                 return r;
164         }
165
166         L(bytes: Buffer){
167                 let result: Buffer = bytes.slice();
168                 for(let i = 0; i < 16; i++){
169                         result = this.GOSTR(result);
170                 }
171                 return result;
172         }
173
174         S (bytes: Buffer){
175                 let result: Buffer = Buffer.alloc(0);
176                 for(let i:number=0; i<bytes.length;i++){
177                         result = Buffer.concat([result, Buffer.from([
                                tabl_notlin[bytes[i]]])]);
178                 }
179                 while(result.length < 16){
180                         result = Buffer.concat([Buffer.from([0]), result]);
181                 }
182                 return result;
183         }
184
185         GOSTR_rev(a: Buffer){
186                 let a_0: number;
187                 a_0 = 0;
188                 let r_inv: Buffer = Buffer.from([0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
189                 0, 0, 0, 0, 0, 0]);
190                 for(let i = 0; i < 15; i++){
191                         r_inv[i] = a[i+1];
192                 }
193                 a_0 = a[0];
194                 for (let i = 0; i < 15; i++)
195                 {
196                         a_0 ^= this.GaloisMult(a[i + 1], constants1[i]);
197                 }
198                 r_inv[15] = a_0;
199                 return r_inv;
200         }
201
```

```
202         L_rev (bytes: Buffer){
203
204                 let res: Buffer = bytes.slice();
205
206                 for(let j = 0; j < 16; j++){
207                         res = this.GOSTR_rev(res);
208                 }
209                 return res;
210         }
211
212         S_rev (bytes: Buffer){
213                 while(bytes.length < 16){
214                         bytes = Buffer.concat([bytes, Buffer.from([0])]);
215                 }
216                 let result: Buffer=Buffer.from([0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0]);
217                 for(let i:number=0; i<16;i++){
218                         result[i] = tabl_notlin.indexOf(bytes[i]);
219                 }
220                 return result;
221         }
222 }
223 export function HexInput(byte:string){
224         return(Buffer.from(byte.replace(/\s+/g, ''), 'hex'));
225 }
226
227
228 export class ECB{
229         kuz: Kuznec;
230
231         constructor(){
232                 this.kuz = new Kuznec();
233         }
234
235         GetKeys(){
236                 return this.kuz.iterKey;
237         }
238
239         SetKeys(keys: Buffer[]){
240                 this.kuz.iterKey = keys;
241                 return this.kuz.iterKey;
242         }
243
244         Encrypt(input:Buffer){
245                 let numbl: number = Math.floor(input.length/16);
246                 let out: Buffer = Buffer.alloc((numbl+1)*16);
```

```
247                 for(let i:number = 0; i< numbl; i++){
248                         let temp:Buffer = Buffer.alloc(16);
249                         temp = this.kuz.Encryption(input.slice(i*16,i*16 + 16)
                                )
250                         for(let j:number = 0; j<16;j++){
251                                 out[16*i+j]=temp[j];
252                         }
253                 }
254                 if(input.length%16!=0){
255                         let temp:Buffer = Buffer.alloc(16);
256                         temp = this.kuz.Encryption(input.slice(numbl*16,numbl
                            *16 + input.length%16))
257                         for(let j:number=0;j<16;j++){
258                                 out[16*numbl+j]=temp[j];
259                         }
260                 }
261                 return out;
262         }
263
264         Decrypt(encrypted:Buffer){
265                 let decrypted: Buffer = Buffer.alloc(encrypted.length);
266                 let numbl: number = encrypted.length/16;
267                 // for(let i = 0; i < encrypted.length; i++){
268                         //      decrypted.push(this.kuz.Decryption(encrypted[i
                            ]));
269                         // }
270                 for(let i = 0; i < numbl; i++){
271                         let temp: Buffer = Buffer.alloc(16);
272                         temp = this.kuz.Decryption(encrypted.slice(i*16,i*16 +
                             16))
273                         for(let j:number = 0; j<16;j++){
274                                 decrypted[16*i+j]=temp[j];
275                         }
276                 }
277                 return decrypted;
278         }
279 }
```

ЗАКЛЮЧЕНИЕ

Были реализованы алгоритмы шифров многоалфавитной замены Вижинера, магический квадрат, алгоритм кузнечик в режиме простой замены.