Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №4
на тему

АСИММЕТРИЧНАЯ КРИПТОГРАФИЯ. АЛГОРИТМ МАК-ЭЛИСА

Выполнил: студент гр.253501
Станишевский А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является изучение принципов асимметричной криптографии на примере криптосистемы Мак-Элиса. В рамках работы требуется разработать программное средство для шифрования и дешифрования текстовых файлов, закрепить навыки работы с асимметричными криптографическими алгоритмами и освоить процесс преобразования данных с использованием матричных операций.

Результатом выполнения работы должен быть скрипт, который позволяет:

- Зашифровать текстовый файл при помощи криптосистемы Мак-Элиса.
- Расшифровать зашифрованный файл обратно в исходный вид.
- Реализовать основные этапы алгоритма Мак-Элиса: генерация ключей, шифрование и дешифрование.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

McEliece – криптосистема с открытыми ключами на основе теории алгебраического кодирования, разработанная в 1978 году Робертом Мак-Элисом. Это была первая схема, использующая рандомизацию в процессе шифрования. Алгоритм основан на сложности декодирования полных линейных кодов.

Криптосистема имеет несколько преимуществ, например, над RSA. Шифрование и дешифрование проходит быстрее и с ростом длины ключа степень защиты данных растет гораздо быстрее. McEliece применим также в задачах аутентификации.

Идея, лежащая в основе данной системы, состоит в выборе корректирующего кода, исправляющего определенное число ошибок, для которого существует эффективный алгоритм декодирования. С помощью секретного ключа этот код «маскируется» под общий линейный код, для которого задача декодирования не имеет эффективного решения.

В системе Мак-Элиса параметрами системы, общими для всех абонентов, являются числа k, n, t. Для получения открытого и соответствующего секретного ключа каждому из абонентов системы следует осуществить следующие действия:

1 Выбрать порождающую матрицу G = Gkn двоичного (n,k)-линейного кода, исправляющего t ошибок, для которого известен эффективный алгоритм декодирования.

2 Случайно выбрать двоичную невырожденную матрицу S = Sk.

3 Случайно выбрать подстановочную матрицу P = Pn.

4 Вычислить произведение матриц G1 = S · G · P.

Открытым ключом является пара (G1, t), секретным – тройка (S, G, P).

Для того чтобы зашифровать сообщение M, предназначенное для абонента A, абоненту B следует выполнить следующие действия:

Представить M в виде двоичного вектора длины k.

Выбрать случайный бинарный вектор ошибок Z длиной n, содержащий не более t единиц.

Вычислить бинарный вектор C = M · GA + Z и направить его абоненту A.

Получив сообщение C, абонент A вычисляет вектор C1 = C · P-1, с помощью которого, используя алгоритм декодирования кода с порождающей матрицей G, получает далее векторы M1 и M = M1 · S-1.

В качестве кода, исправляющего ошибки в системе Мак-Элиса, можно использовать код Гоппы. Известно, что для любого неприводимого полинома g(x) степени t над полем GF(2m) существует бинарный код Гоппы длины n = 2m и размерности k ≥ n – mt, исправляющий до t ошибок включительно, для которого имеется эффективный алгоритм декодирования.

3 ХОД РАБОТЫ

Программное средство реализовано при помощи языка программирования JavaScript. На рисунке 3.1 изображен процесс шифрования исходного сообщения.

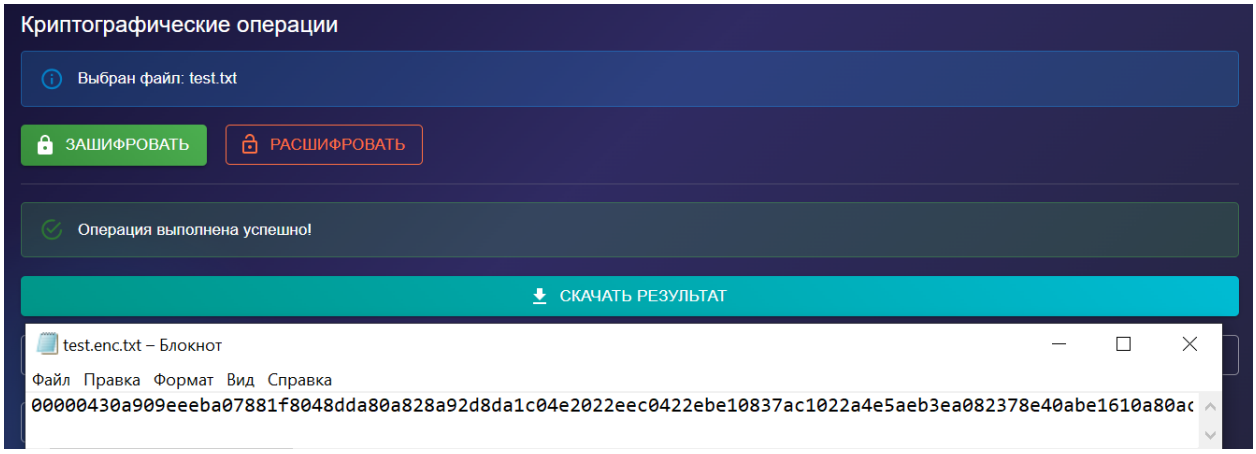

Рисунок 3.1 – Процесс шифрования исходного сообщения

На рисунке 3.2 изображен процесс дешифрования зашифрованного исходного сообщения.

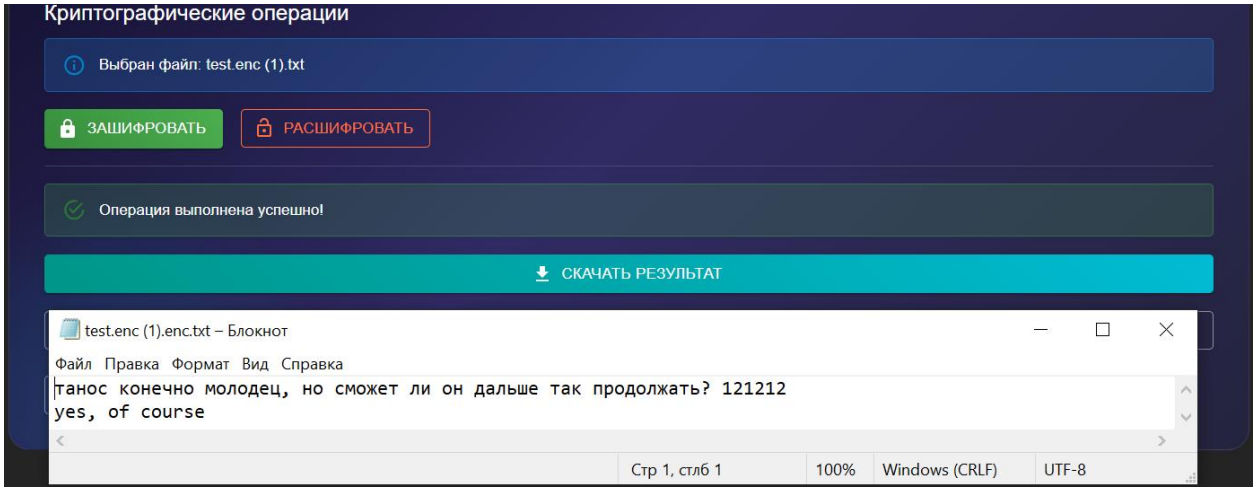

Рисунок 3.2 – Процесс дешифрования исходного сообщения

Таким образом, программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены принципы асимметричного шифрования на примере алгоритма Мак-Элиса. Разработанное программное средство позволяет шифровать и расшифровывать текстовые файлы, демонстрируя основные этапы работы алгоритма. Полученные практические навыки работы с асимметричными криптографическими алгоритмами позволяют успешно применять криптосистему Мак-Элиса для защиты информации.

Таким образом, цель лабораторной работы достигнута, и приобретенные знания и навыки могут быть использованы для решения задач обеспечения конфиденциальности данных.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
import { ByteUtils } from './byteUtils';

export class McElieceCrypto {
  constructor() {
    this.m = 3;
    this.n = 7;
    this.k = 4;
    this.t = 1;

    this.S = null;
    this.G = null;
    this.P = null;
    this.G1 = null;
    this.H = null;
  }

  generateHammingGeneratorMatrix() {
    const G = [
      [1,0,0,0, 1,1,1],
      [0,1,0,0, 1,1,0],
      [0,0,1,0, 1,0,1],
      [0,0,0,1, 0,1,1]
    ];

    return G;
  }

  generateHammingParityCheckMatrix() {
    const H = [
      [1,1,1,0, 1,0,0],
      [1,1,0,1, 0,1,0],
      [1,0,1,1, 0,0,1]
    ];

    return H;
  }

  generateRandomBinaryMatrix(rows, cols) {
    const matrix = new Array(rows);
    for (let i = 0; i < rows; i++) {
      matrix[i] = new Array(cols);
      for (let j = 0; j < cols; j++) {
        matrix[i][j] = Math.random() > 0.5 ? 1 : 0;
      }
    }
    return matrix;
  }

  generateNonSingularMatrix(size) {
    let matrix;
    let det;

    do {
      matrix = this.generateRandomBinaryMatrix(size, size);
      det = this.matrixDeterminant(matrix);
    } while (det === 0);
```

```
  return matrix;
}

generatePermutationMatrix(size) {
  const matrix = new Array(size);
  const indices = Array.from({length: size}, (_, i) => i);

  for (let i = size - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [indices[i], indices[j]] = [indices[j], indices[i]];
  }

  for (let i = 0; i < size; i++) {
    matrix[i] = new Array(size).fill(0);
    matrix[i][indices[i]] = 1;
  }

  return matrix;
}

matrixMultiply(A, B) {
  const rowsA = A.length;
  const colsA = A[0].length;
  const colsB = B[0].length;

  const result = new Array(rowsA);
  for (let i = 0; i < rowsA; i++) {
    result[i] = new Array(colsB).fill(0);
    for (let j = 0; j < colsB; j++) {
      let sum = 0;
      for (let k = 0; k < colsA; k++) {
        sum ^= A[i][k] & B[k][j];
      }
      result[i][j] = sum;
    }
  }

  return result;
}

matrixTranspose(matrix) {
  const rows = matrix.length;
  const cols = matrix[0].length;
  const result = new Array(cols);

  for (let i = 0; i < cols; i++) {
    result[i] = new Array(rows);
    for (let j = 0; j < rows; j++) {
      result[i][j] = matrix[j][i];
    }
  }

  return result;
}

matrixDeterminant(matrix) {
  const size = matrix.length;

  if (size === 1) return matrix[0][0];
  if (size === 2) {
    return (matrix[0][0] & matrix[1][1]) ^ (matrix[0][1] & matrix[1][0]);
  }

  let det = 0;
```

```
    for (let col = 0; col < size; col++) {
      if (matrix[0][col] === 1) {
        const minor = this.getMinor(matrix, 0, col);
        const minorDet = this.matrixDeterminant(minor);
        det ^= minorDet;
      }
    }

    return det;
  }

  getMinor(matrix, row, col) {
    const size = matrix.length;
    const minor = new Array(size - 1);

    for (let i = 0, m = 0; i < size; i++) {
      if (i === row) continue;
      minor[m] = new Array(size - 1);
      for (let j = 0, n = 0; j < size; j++) {
        if (j === col) continue;
        minor[m][n] = matrix[i][j];
        n++;
      }
      m++;
    }

    return minor;
  }

  matrixInverse(matrix) {
    const size = matrix.length;
    const augmented = new Array(size);

    for (let i = 0; i < size; i++) {
      augmented[i] = new Array(2 * size);
      for (let j = 0; j < size; j++) {
        augmented[i][j] = matrix[i][j];
      }
      for (let j = size; j < 2 * size; j++) {
        augmented[i][j] = j - size === i ? 1 : 0;
      }
    }

    for (let col = 0; col < size; col++) {
      let pivotRow = -1;
      for (let row = col; row < size; row++) {
        if (augmented[row][col] === 1) {
          pivotRow = row;
          break;
        }
      }

      if (pivotRow === -1) {
        throw new Error('Матрица вырождена');
      }

      if (pivotRow !== col) {
        [augmented[col],    augmented[pivotRow]]    =    [augmented[pivotRow],
augmented[col]];
      }

      for (let row = 0; row < size; row++) {
        if (row !== col && augmented[row][col] === 1) {
          for (let j = col; j < 2 * size; j++) {
```

```
          augmented[row][j] ^= augmented[col][j];
        }
      }
    }
  }

  const inverse = new Array(size);
  for (let i = 0; i < size; i++) {
    inverse[i] = new Array(size);
    for (let j = 0; j < size; j++) {
      inverse[i][j] = augmented[i][j + size];
    }
  }

  return inverse;
}

generateKeys() {
  this.G = this.generateHammingGeneratorMatrix();
  this.H = this.generateHammingParityCheckMatrix();

  this.S = this.generateNonSingularMatrix(this.k);
  this.P = this.generatePermutationMatrix(this.n);

  const SG = this.matrixMultiply(this.S, this.G);
  this.G1 = this.matrixMultiply(SG, this.P);

  return {
    publicKey: {
      G1: this.G1,
      t: this.t,
      n: this.n,
      k: this.k
    },
    privateKey: {
      S: this.S,
      G: this.G,
      P: this.P,
      H: this.H
    }
  };
}

setKeys(publicKey, privateKey = null) {
  this.G1 = publicKey.G1;
  this.t = publicKey.t;
  this.n = publicKey.n;
  this.k = publicKey.k;

  if (privateKey) {
    this.S = privateKey.S;
    this.G = privateKey.G;
    this.P = privateKey.P;
    this.H = privateKey.H;
  }
}

generateErrorVector() {
  const errorVector = new Array(this.n).fill(0);
  const errorPos = Math.floor(Math.random() * this.n);
  errorVector[errorPos] = 1;
  return errorVector;
}
```

```
vectorMatrixMultiply(vector, matrix) {
  const cols = matrix[0].length;
  const result = new Array(cols).fill(0);

  for (let j = 0; j < cols; j++) {
    for (let i = 0; i < vector.length; i++) {
      result[j] ^= vector[i] & matrix[i][j];
    }
  }

  return result;
}

computeSyndrome(vector, H) {
  const syndrome = new Array(H.length).fill(0);

  for (let i = 0; i < H.length; i++) {
    for (let j = 0; j < H[0].length; j++) {
      syndrome[i] ^= vector[j] & H[i][j];
    }
  }

  return syndrome;
}

correctError(codeword, H) {
  const syndrome = this.computeSyndrome(codeword, H);

  if (syndrome.every(bit => bit === 0)) {
    return [...codeword];
  }

  const errorPosition = this.findErrorPosition(syndrome, H);

  if (errorPosition !== -1) {
    const corrected = [...codeword];
    corrected[errorPosition] ^= 1;
    return corrected;
  }

  return codeword;
}

findErrorPosition(syndrome, H) {
  for (let col = 0; col < H[0].length; col++) {
    let match = true;
    for (let row = 0; row < H.length; row++) {
      if (H[row][col] !== syndrome[row]) {
        match = false;
        break;
      }
    }
    if (match) {
      return col;
    }
  }
  return -1;
}

encrypt(messageVector) {
  if (!this.G1) {
    throw new Error('Открытый ключ не установлен');
  }
```

```
  if (messageVector.length !== this.k) {
    throw new Error(`Длина сообщения должна быть ${this.k} бит`);
  }

  const errorVector = this.generateErrorVector();
  const encoded = this.vectorMatrixMultiply(messageVector, this.G1);
  const ciphertext = new Array(this.n);

  for (let i = 0; i < this.n; i++) {
    ciphertext[i] = encoded[i] ^ errorVector[i];
  }

  return ciphertext;
}

decode(cipherVector) {
  if (!this.H) {
    throw new Error('Проверочная матрица не установлена');
  }

  const corrected = this.correctError(cipherVector, this.H);
  const message = corrected.slice(0, this.k);

  return message;
}

decrypt(cipherVector) {
  if (!this.S || !this.G || !this.P || !this.H) {
    throw new Error('Закрытый ключ не установлен');
  }

  const P_inv = this.matrixTranspose(this.P);
  const c1 = this.vectorMatrixMultiply(cipherVector, P_inv);

  const m1 = this.decode(c1);

  const S_inv = this.matrixInverse(this.S);
  const message = this.vectorMatrixMultiply(m1, S_inv);

  return message;
}

async encryptText(text) {
  const textBytes = ByteUtils.stringToBytes(text);
  const allBits = this.bytesToBits(textBytes);
  const encryptedBlocks = [];

  for (let i = 0; i < allBits.length; i += this.k) {
    const blockBits = allBits.slice(i, i + this.k);

    while (blockBits.length < this.k) {
      blockBits.push(0);
    }

    const encryptedBlock = this.encrypt(blockBits);
    encryptedBlocks.push(encryptedBlock);
  }

  const metadata = new Uint8Array(4);
  const dataView = new DataView(metadata.buffer);
  dataView.setUint32(0, allBits.length, false);

  const encryptedBits = encryptedBlocks.flat();
```

```
    const encryptedBytes = this.bitsToBytes(encryptedBits);

    const resultBytes = new Uint8Array(metadata.length + encryptedBytes.length);
    resultBytes.set(metadata);
    resultBytes.set(encryptedBytes, metadata.length);

    return ByteUtils.bytesToHex(resultBytes);
  }

  async decryptText(encryptedHex) {
    const allBytes = ByteUtils.hexToBytes(encryptedHex);

    const dataView = new DataView(allBytes.buffer, allBytes.byteOffset, 4);
    const originalBitsLength = dataView.getUint32(0, false);

    const encryptedBytes = allBytes.slice(4);
    const encryptedBits = this.bytesToBits(encryptedBytes);

    const decryptedBits = [];

    for (let i = 0; i < encryptedBits.length; i += this.n) {
      const blockBits = encryptedBits.slice(i, i + this.n);
      if (blockBits.length < this.n) break;

      const decryptedBlock = this.decrypt(blockBits);
      decryptedBits.push(...decryptedBlock);
    }

    const originalBits = decryptedBits.slice(0, originalBitsLength);

    const decryptedBytes = this.bitsToBytes(originalBits);

    return ByteUtils.bytesToString(decryptedBytes);
  }

  bytesToBits(bytes) {
    const bits = [];
    for (const byte of bytes) {
      for (let i = 7; i >= 0; i--) {
        bits.push((byte >> i) & 1);
      }
    }
    return bits;
  }

  bitsToBytes(bits) {
    const byteCount = Math.ceil(bits.length / 8);
    const bytes = new Uint8Array(byteCount);

    for (let i = 0; i < byteCount; i++) {
      let byte = 0;
      for (let j = 0; j < 8; j++) {
        const bitIndex = i * 8 + j;
        if (bitIndex < bits.length) {
          byte = (byte << 1) | bits[bitIndex];
        } else {
          byte = byte << 1;
        }
      }
      bytes[i] = byte;
    }

    return bytes;
```

```
  }

  async encryptFile(file, publicKey) {
    this.setKeys(publicKey);
    const text = await this.readFileAsText(file);
    const encrypted = await this.encryptText(text);
    return new Blob([encrypted], { type: 'text/plain; charset=utf-8' });
  }

  async decryptFile(file, privateKey) {
    const privateKeyObj = typeof privateKey === 'string' ?
JSON.parse(privateKey) : privateKey;

    const SG = this.matrixMultiply(privateKeyObj.S, privateKeyObj.G);
    const publicKey = {
      G1: this.matrixMultiply(SG, privateKeyObj.P),
      t: this.t,
      n: this.n,
      k: this.k
    };

    this.setKeys(publicKey, privateKeyObj);

    const encryptedText = await this.readFileAsText(file);
    const decrypted = await this.decryptText(encryptedText);
    return new Blob([decrypted], { type: 'text/plain; charset=utf-8' });
  }

  async readFileAsText(file) {
    return new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onload = (e) => resolve(e.target.result);
      reader.onerror = () => reject(new Error('Ошибка чтения файла'));
      reader.readAsText(file, 'UTF-8');
    });
  }

  serializeMatrix(matrix) {
    return matrix.map(row => row.join('')).join(';');
  }

  deserializeMatrix(str, rows, cols) {
    const rowsArray = str.split(';');
    const matrix = new Array(rows);

    for (let i = 0; i < rows; i++) {
      matrix[i] = new Array(cols);
      const rowStr = rowsArray[i];
      for (let j = 0; j < cols; j++) {
        matrix[i][j] = parseInt(rowStr[j], 10);
      }
    }

    return matrix;
  }
}
```