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 №2
на тему

СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТБ 34.101.31-2011

Выполнил: студент гр.253501
Станишевский А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Целью работы является исследование алгоритма симметричного блочного шифрования и разработка демонстрационного программного решения, иллюстрирующего принципы его работы, режимы шифрования и криптостойкость.

В рамках работы реализована программа, позволяющая проводить шифрование и дешифрование данных в соответствии со стандартом СТБ 34.101.31-2011 в режиме простой замены. Программа моделирует полный цикл криптографического преобразования, включая формирование ключевой информации и работу базовых циклов.

Таким образом, в ходе работы получены практические навыки реализации криптографических алгоритмов, изучены принципы и особенности белорусского стандарта шифрования, а также создано наглядная программа для анализа его работы.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Настоящий стандарт определяет семейство криптографических алгоритмов, предназначенных для обеспечения конфиденциальности и контроля целостности данных. Обрабатываемыми данными являются двоичные слова (сообщения).

Криптографические алгоритмы стандарта построены на основе базовых алгоритмов шифрования блока данных.

Блок-схема алгоритма на i-ом такте шифрования представлена на рисунке 2.1.

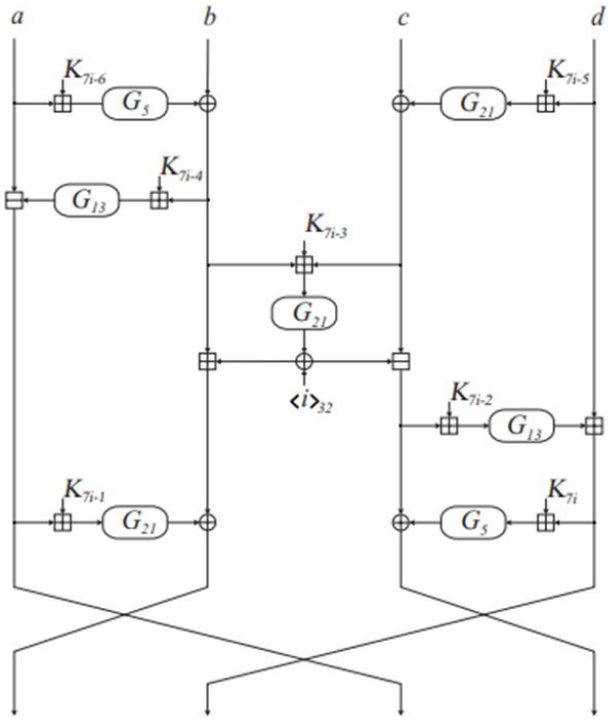

Рисунок 2.1 – Вычисления на i-м такте шифрования

Входными данными алгоритмов зашифрования и расшифрования являются блок $X \in \{0,1\}^{128}$ и ключ $\theta \in \{0,1\}^{256}$.

Выходными данными является блок $Y \in \{0,1\}^{128}$ — результат зашифрования либо расшифрования слова X на ключе $\theta : Y = F_\theta(X)$ либо $Y = F_\theta^{-1}(X)$.

Входные данные для шифрования подготавливаются следующим образом: Слово X записывается в виде $X = X_1 \| X_2 \| X_3 \| X_4, X_i \in \{0,1\}^{32}$.

3 ХОД РАБОТЫ

Программное средство реализовано при помощи языка программирования JavaScript. На рисунке 3.1 изображен процесс шифрования исходного файла, на рисунке 3.2 – процесс дешифрования зашифрованного исходного файла.

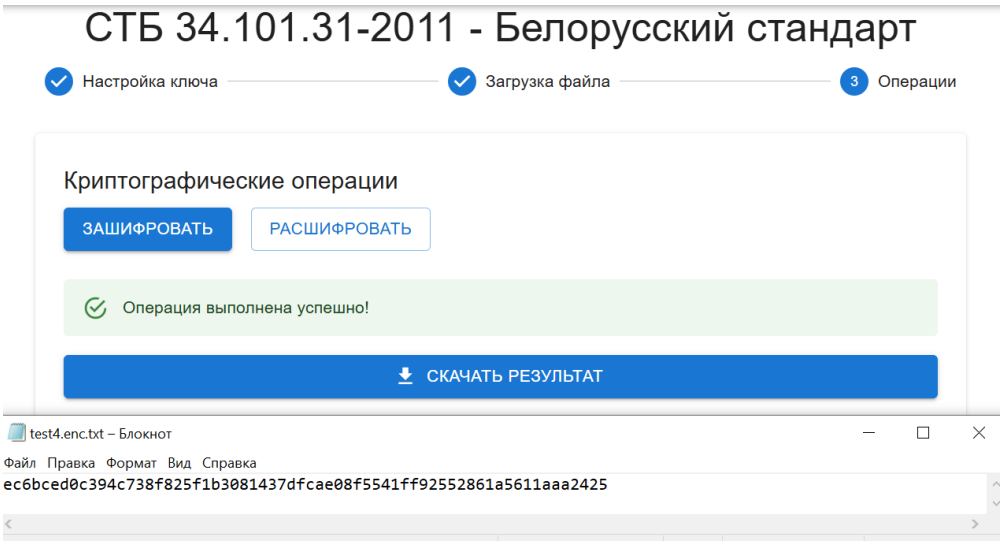

Рисунок 3.1 – Процесс шифрования исходного файла

Шифрование и дешифрование производится с одинаковым ключом, сгенерированным заранее.

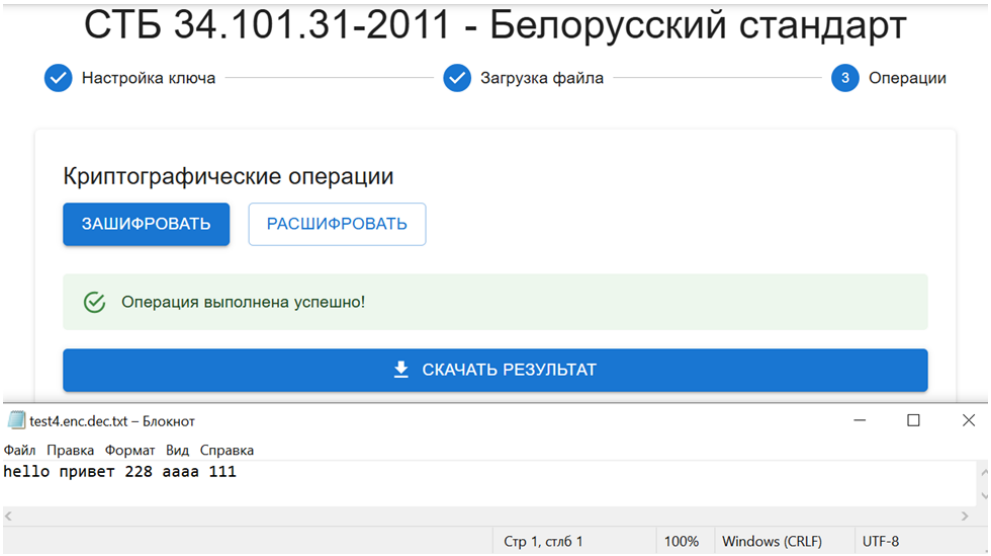

Рисунок 3.2 – Процесс дешифрования зашифрованного исходного файла

Таким образом, защищенная программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – исследование алгоритма симметричного блочного шифрования СТБ 34.101.31-2011 и реализация демонстрационного решения, иллюстрирующего его работу в режиме простой замены. Разработанная программа наглядно демонстрирует ключевые принципы построения криптосистем: применение таблиц замены, циклические сдвиги, наложение подключей.

Успешная реализация режима простой замены, обеспечивающая как шифрование, так и корректное дешифрование данных, подтверждает правильность понимания и применения математического аппарата стандарта.

Таким образом, в результате проделанной работы были получены практические навыки реализации и анализа белорусского стандарта шифрования СТБ 34.101.31-2011, а также создана программа, реализующая данный алгоритм симметричного шифрования в режиме гаммирования.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

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

export class STBCrypto {
  constructor(key) {
    if (!key) throw new Error('Ключ не может быть пустым');
    this.key = this.prepareKey(key);
    this.initSBox();
  }

  prepareKey(key) {
    if (typeof key === 'string' && key.match(/^[0-9a-fA-F]{64}$/)) {
      return ByteUtils.hexToBytes(key);
    }

    if (typeof key === 'string') {
      const keyBytes = ByteUtils.stringToBytes(key);
      const result = new Uint8Array(32);

      if (keyBytes.length >= 32) {
        result.set(keyBytes.slice(0, 32));
      } else {
        result.set(keyBytes);
        for (let i = keyBytes.length; i < 32; i++) {
          result[i] = 0;
        }
      }

      return result;
    }

    throw new Error('Неверный формат ключа');
  }

  initSBox() {
    this.sBox = new Uint8Array([
      0xB1, 0x94, 0xBA, 0xC8, 0x0A, 0x08, 0xF5, 0x3B, 0x36, 0x6D, 0x00, 0x8E,
0x58, 0x4A, 0x5D, 0xE4,
      0x85, 0x04, 0xFA, 0x9D, 0x1B, 0xB6, 0xC7, 0xAC, 0x25, 0x2E, 0x72, 0xC2,
0x02, 0xFD, 0xCE, 0x0D,
      0x5B, 0xE3, 0xD6, 0x12, 0x17, 0xB9, 0x61, 0x81, 0xFE, 0x67, 0x86, 0xAD,
0x71, 0x6B, 0x89, 0x0B,
      0x5C, 0xB0, 0xC0, 0xFF, 0x33, 0xC3, 0x56, 0xB8, 0x35, 0xC4, 0x05, 0xAE,
0xD8, 0xE0, 0x7F, 0x99,
      0xE1, 0x2B, 0xDC, 0x1A, 0xE2, 0x82, 0x57, 0xEC, 0x70, 0x3F, 0xCC, 0xF0,
0x95, 0xEE, 0x8D, 0xF1,
      0xC1, 0xAB, 0x76, 0x38, 0x9F, 0xE6, 0x78, 0xCA, 0xF7, 0xC6, 0xF8, 0x60,
0xD5, 0xBB, 0x9C, 0x4F,
      0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, 0xDD, 0x4E, 0xA7, 0x79,
0x9E, 0xB2, 0x3D, 0x31,
      0x3E, 0x98, 0xB5, 0x6E, 0x27, 0xD3, 0xBC, 0xCF, 0x59, 0x1E, 0x18, 0x1F,
0x4C, 0x5A, 0xB7, 0x93,
      0xE9, 0xDE, 0xE7, 0x2C, 0x8F, 0x0C, 0x0F, 0xA6, 0x2D, 0xDB, 0x49, 0xF4,
0x6F, 0x73, 0x96, 0x47,
      0x06, 0x07, 0x53, 0x16, 0xED, 0x24, 0x7A, 0x37, 0x39, 0xCB, 0xA3, 0x83,
0x03, 0xA9, 0x8B, 0xF6,
      0x92, 0xBD, 0x9B, 0x1C, 0xE5, 0xD1, 0x41, 0x01, 0x54, 0x45, 0xFB, 0xC9,
0x5E, 0x4D, 0x0E, 0xF2,
```

```
      0x68, 0x20, 0x80, 0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, 0x87, 0xF9, 0x34,
0x90, 0x40, 0x55, 0x11,
      0xBE, 0x32, 0x97, 0x13, 0x43, 0xFC, 0x9A, 0x48, 0xA0, 0x2A, 0x88, 0x5F,
0x19, 0x4B, 0x09, 0xA1,
      0x7E, 0xCD, 0xA4, 0xD0, 0x15, 0x44, 0xAF, 0x8C, 0xA5, 0x84, 0x50, 0xBF,
0x66, 0xD2, 0xE8, 0x8A,
      0xA2, 0xD7, 0x46, 0x52, 0x42, 0xA8, 0xDF, 0xB3, 0x69, 0x74, 0xC5, 0x51,
0xEB, 0x23, 0x29, 0x21,
      0xD4, 0xEF, 0xD9, 0xB4, 0x3A, 0x62, 0x28, 0x75, 0x91, 0x14, 0x10, 0xEA,
0x77, 0x6C, 0xDA, 0x1D
    ]);
  }

  H(byte) {
    return this.sBox[byte];
  }

  G_r(u, r) {
    const bytes = new Uint8Array(4);
    const view = new DataView(bytes.buffer);
    view.setUint32(0, u, false);

    for (let i = 0; i < 4; i++) {
      bytes[i] = this.H(bytes[i]);
    }

    let result = view.getUint32(0, false);

    result = ((result << r) | (result >>> (32 - r))) >>> 0;
    return result;
  }

  addMod32(a, b) {
    return (a + b) >>> 0;
  }

  subMod32(a, b) {
    return (a - b) >>> 0;
  }

  generateSubKeys() {
    const subKeys = [];
    const view = new DataView(this.key.buffer);

    for (let i = 0; i < 8; i++) {
      subKeys.push(view.getUint32(i * 4, false));
    }

    return subKeys;
  }

  encryptBlock(block) {
    const subKeys = this.generateSubKeys();
    const view = new DataView(block.buffer);

    let a = view.getUint32(0, false);
    let b = view.getUint32(4, false);
    let c = view.getUint32(8, false);
    let d = view.getUint32(12, false);

    for (let i = 0; i < 8; i++) {
      const k = subKeys[i];

      b = b ^ this.G_r(this.addMod32(a, k), 5);
```

```
    c = c ^ this.G_r(this.addMod32(d, k), 21);

    a = this.subMod32(a, this.G_r(this.addMod32(b, k), 13));

    const r = this.addMod32(this.addMod32(b, c), k);
    const t = this.G_r(r, 21) ^ (i + 1);

    b = this.addMod32(b, t);
    c = this.subMod32(c, t);
    d = this.addMod32(d, this.G_r(this.addMod32(c, k), 13));

    b = b ^ this.G_r(this.addMod32(a, k), 21);
    c = c ^ this.G_r(this.addMod32(d, k), 5);

    [a, b] = [b, a];
    [c, d] = [d, c];
    [b, c] = [c, b];
  }

  const result = new Uint8Array(16);
  const resultView = new DataView(result.buffer);
  resultView.setUint32(0, b, false);
  resultView.setUint32(4, d, false);
  resultView.setUint32(8, a, false);
  resultView.setUint32(12, c, false);

  return result;
}

decryptBlock(block) {
  const subKeys = this.generateSubKeys();
  const view = new DataView(block.buffer);

  let b = view.getUint32(0, false);
  let d = view.getUint32(4, false);
  let a = view.getUint32(8, false);
  let c = view.getUint32(12, false);

  for (let i = 7; i >= 0; i--) {
    const k = subKeys[i];

    [b, c] = [c, b];
    [c, d] = [d, c];
    [a, b] = [b, a];

    c = c ^ this.G_r(this.addMod32(d, k), 5);
    b = b ^ this.G_r(this.addMod32(a, k), 21);

    d = this.subMod32(d, this.G_r(this.addMod32(c, k), 13));

    const t = this.G_r(this.addMod32(this.addMod32(b, c), k), 21) ^ (i + 1);
    c = this.addMod32(c, t);
    b = this.subMod32(b, t);

    a = this.addMod32(a, this.G_r(this.addMod32(b, k), 13));

    c = c ^ this.G_r(this.addMod32(d, k), 21);
    b = b ^ this.G_r(this.addMod32(a, k), 5);
  }

  const result = new Uint8Array(16);
  const resultView = new DataView(result.buffer);
  resultView.setUint32(0, a, false);
  resultView.setUint32(4, b, false);
```

```
    resultView.setUint32(8, c, false);
    resultView.setUint32(12, d, false);

    return result;
  }

  async encryptText(text) {
    const textBytes = ByteUtils.stringToBytes(text);
    const paddedBytes = this.padData(textBytes);
    const result = new Uint8Array(paddedBytes.length);

    for (let i = 0; i < paddedBytes.length; i += 16) {
      const block = paddedBytes.slice(i, i + 16);
      const encryptedBlock = this.encryptBlock(block);
      result.set(encryptedBlock, i);
    }

    return ByteUtils.bytesToHex(result);
  }

  async decryptText(encryptedHex) {
    if (!ByteUtils.isValidHex(encryptedHex)) {
      throw new Error('Неверный формат зашифрованных данных');
    }

    const encryptedBytes = ByteUtils.hexToBytes(encryptedHex);
    if (encryptedBytes.length % 16 !== 0) {
      throw new Error('Длина зашифрованных данных должна быть кратна 16
байтам');
    }

    const result = new Uint8Array(encryptedBytes.length);

    for (let i = 0; i < encryptedBytes.length; i += 16) {
      const block = encryptedBytes.slice(i, i + 16);
      const decryptedBlock = this.decryptBlock(block);
      result.set(decryptedBlock, i);
    }

    const unpadded = this.unpadData(result);
    return ByteUtils.bytesToString(unpadded);
  }

  padData(data) {
    const blockSize = 16;
    const padding = blockSize - (data.length % blockSize);
    const result = new Uint8Array(data.length + padding);
    result.set(data);
    for (let i = data.length; i < result.length; i++) {
      result[i] = padding;
    }
    return result;
  }

  unpadData(data) {
    const padding = data[data.length - 1];
    if (padding > 0 && padding <= 16) {
      for (let i = data.length - padding; i < data.length; i++) {
        if (data[i] !== padding) {
          return data;
        }
      }
      return data.slice(0, data.length - padding);
    }
```

```
    return data;
  }

  async encryptFile(file) {
    const arrayBuffer = await this.readFileAsArrayBuffer(file);
    const dataBytes = new Uint8Array(arrayBuffer);
    const paddedBytes = this.padData(dataBytes);
    const result = new Uint8Array(paddedBytes.length);

    for (let i = 0; i < paddedBytes.length; i += 16) {
      const block = paddedBytes.slice(i, i + 16);
      const encryptedBlock = this.encryptBlock(block);
      result.set(encryptedBlock, i);
    }

    const hexResult = ByteUtils.bytesToHex(result);
    return new Blob([hexResult], { type: 'text/plain' });
  }

  async decryptFile(file) {
    const hexText = await this.readFileAsText(file);

    if (!ByteUtils.isValidHex(hexText)) {
      throw new Error('Файл содержит неверный формат зашифрованных данных');
    }

    const encryptedBytes = ByteUtils.hexToBytes(hexText);
    if (encryptedBytes.length % 16 !== 0) {
      throw new Error('Длина зашифрованных данных должна быть кратна 16
байтам');
    }

    const result = new Uint8Array(encryptedBytes.length);

    for (let i = 0; i < encryptedBytes.length; i += 16) {
      const block = encryptedBytes.slice(i, i + 16);
      const decryptedBlock = this.decryptBlock(block);
      result.set(decryptedBlock, i);
    }

    const unpadded = this.unpadData(result);

    return new Blob([unpadded], { type: 'application/octet-stream' });
  }

  async readFileAsArrayBuffer(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.readAsArrayBuffer(file);
    });
  }

  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);
    });
  }
}
```