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 №1
на тему

СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТАНДАРТ ШИФРОВАНИЯ ГОСТ 28147-89

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 его работу по принципу гаммирования и криптостойкость.

В рамках работы реализована программа, позволяющая проводить шифрование и дешифрование данных в соответствии со стандартом ГОСТ 28147-89 в режиме гаммирования. Программа моделирует полный цикл криптографического преобразования, включая формирование ключевой информации, синхропосылок и работу базовых циклов.

Таким образом, в ходе работы получены практические навыки реализации криптографических алгоритмов, изучены принципы и особенности стандарта шифрования ГОСТ 28147-89, а также создано наглядное программное средство.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» – устаревший государственный стандарт СССР (позже межгосударственный стандарт СНГ), описывающий алгоритм симметричного блочного шифрования и режимы его работы.

Является примером DES-подобных криптосистем, созданных по классической итерационной схеме Фейстеля.

ГОСТ 28147-89 представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом.

Этот алгоритм криптографического преобразования данных предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям и до 1983 года не накладывал ограничений на степень секретности защищаемой информации.

ГОСТ 28147-89 является блочным шифром, поэтому преобразование данных осуществляется блоками в так называемых базовых циклах.

Базовые циклы заключаются в многократном выполнении для блока данных основного раунда, рассмотренного нами ранее, с использованием разных элементов ключа и отличаются друг от друга порядком использования ключевых элементов.

В каждом раунде используется один из восьми возможных 32-разрядных подключей.

Рассмотрим процесс создания подключей раундов. В ГОСТ эта процедура очень проста, особенно по сравнению с DES. 256-битный ключ K разбивается на восемь 32-битных подключей, обозначаемых K0, K1, K2, K3, K4, K5, K6, K7. Алгоритм включает 32 раунда, поэтому каждый подключ при шифровании используется в четырех раундах в последовательности.

Процесс расшифрования производится по тому же алгоритму, что и шифрование. Единственное отличие заключается в порядке использования подключей Ki. При расшифровании подключи должны быть использованы в обратном порядке.

Режим простой замены: все блоки шифруются независимо друг от друга с разными подключами в разных раундах. Для одинаковых блоков сообщения M блоки шифртекста будут одинаковыми.

Режим гаммирования: В регистры N1 и N2 записывается 64-битовая синхропосылка (вектор инициализации) и шифруется с использованием СК. Результат подается на вход регистров и снова шифруется с использованием ключа. Получается «одноразовый блокнот».

В режиме гаммирования с обратной связью для заполнения регистров N1 и N2, начиная со 2-го блока, используется результат зашифрования предыдущего блока открытого текста.

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.

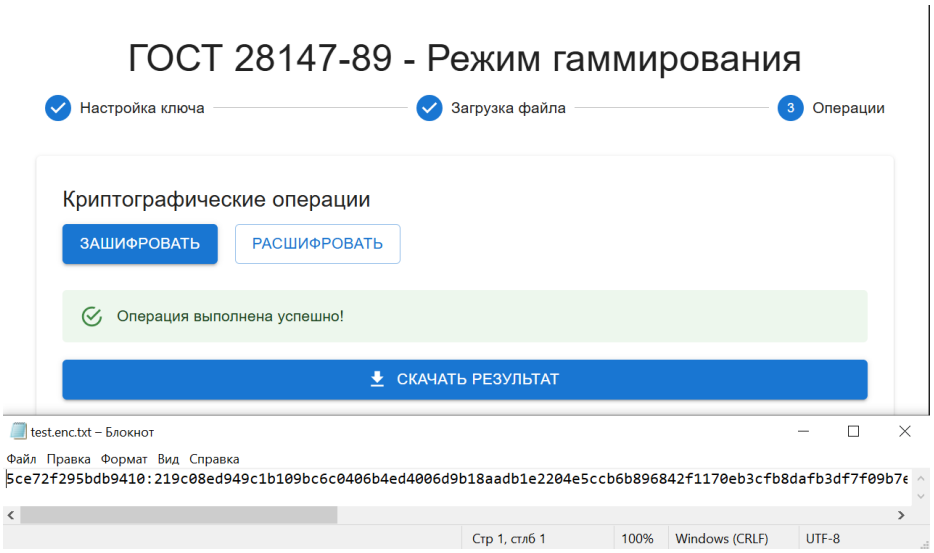

Рисунок 3.1 – Процесс шифрования исходного файла

Шифрование и дешифрование производится с одинаковым ключом, сгенерированным заранее.

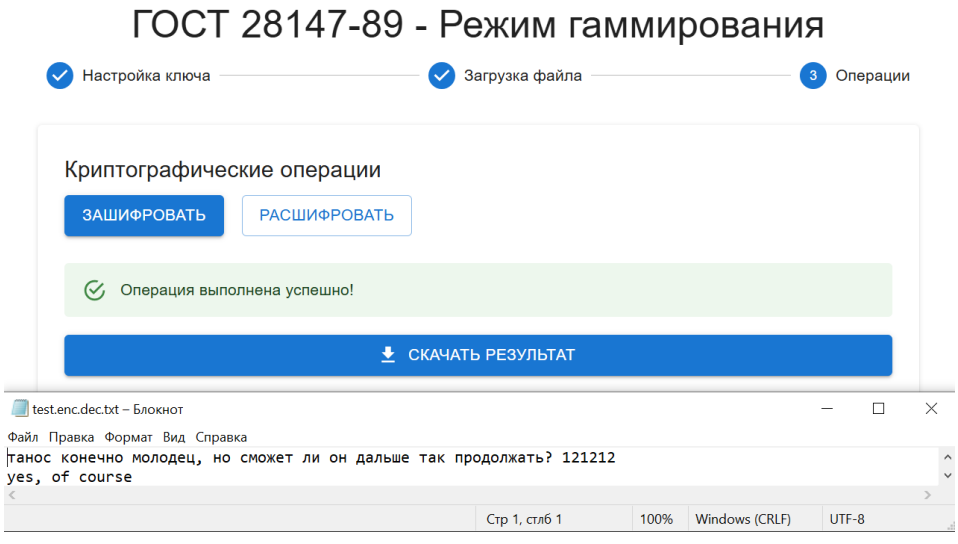

Рисунок 3.2 – Процесс дешифрования зашифрованного исходного файла

Таким образом, программное средство успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – исследование алгоритма симметричного блочного шифрования ГОСТ 28147-89 и реализация демонстрационного решения, иллюстрирующего его работу в режиме гаммирования. Разработанная программа наглядно демонстрирует ключевые принципы построения криптосистем: генерацию гаммы на основе базовых циклов шифрования, поточное наложение гаммы на открытый текст и использование синхропосылки для обеспечения криптостойкости.

Успешная реализация режима гаммирования, 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 стандарта шифрования ГОСТ 28147-89, а также создана программа, реализующая данный алгоритм симметричного шифрования в режиме гаммирования.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

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

  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');
  }

  initSBoxes() {
    this.sBoxes = [
      [4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3],
      [14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9],
      [5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11],
      [7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3],
      [6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2],
      [4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14],
      [13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12],
      [1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12]
    ];
  }

  generateSubKeys() {
    const subKeys = [];

    for (let i = 0; i < 8; i++) {
      const subKey = new Uint32Array(1);
      const view = new DataView(subKey.buffer);

      for (let j = 0; j < 4; j++) {
        const byte = this.key[i * 4 + j];
        view.setUint8(j, byte);
      }

      subKeys.push(view.getUint32(0, false));
```

```
    }

    const extendedSubKeys = [];
    for (let i = 0; i < 24; i++) {
      extendedSubKeys.push(subKeys[i % 8]);
    }
    for (let i = 7; i >= 0; i--) {
      extendedSubKeys.push(subKeys[i]);
    }

    return extendedSubKeys;
  }

  f(input, subKey) {
    let temp = (input + subKey) >>> 0;

    let output = 0;
    for (let i = 0; i < 8; i++) {
      const nibble = (temp >>> (4 * i)) & 0x0F;
      const sboxValue = this.sBoxes[i][nibble];
      output |= (sboxValue << (4 * i));
    }

    return ((output << 11) | (output >>> 21)) >>> 0;
  }

  encryptBlock(block, subKeys) {
    let left = block[0];
    let right = block[1];

    for (let i = 0; i < 32; i++) {
      const fResult = this.f(right, subKeys[i]);
      const newRight = (left ^ fResult) >>> 0;
      left = right;
      right = newRight;
    }

    return [right, left];
  }

  blockToUint32(block) {
    const view = new DataView(block.buffer, block.byteOffset);
    return [view.getUint32(0, false), view.getUint32(4, false)];
  }

  uint32ToBlock(left, right) {
    const block = new Uint8Array(8);
    const view = new DataView(block.buffer);
    view.setUint32(0, left, false);
    view.setUint32(4, right, false);
    return block;
  }

  generateGamma(syncMessage, length) {
    const subKeys = this.generateSubKeys();
    const gamma = new Uint8Array(length);
    let currentBlock = this.blockToUint32(syncMessage);

    for (let i = 0; i < length; i += 8) {
      const encryptedBlock = this.encryptBlock(currentBlock, subKeys);
      const      gammaBlock      =      this.uint32ToBlock(encryptedBlock[0],
encryptedBlock[1]);

      const blockLength = Math.min(8, length - i);
```

```
      for (let j = 0; j < blockLength; j++) {
        gamma[i + j] = gammaBlock[j];
      }

      currentBlock = encryptedBlock;
    }

    return gamma;
  }

  async encryptText(text) {
    const textBytes = ByteUtils.stringToBytes(text);
    const syncMessage = crypto.getRandomValues(new Uint8Array(8));
    const gamma = this.generateGamma(syncMessage, textBytes.length);

    const encryptedBytes = ByteUtils.xorBytes(textBytes, gamma);
    const encryptedHex = ByteUtils.bytesToHex(encryptedBytes);
    const syncHex = ByteUtils.bytesToHex(syncMessage);

    return `${syncHex}:${encryptedHex}`;
  }
  async decryptText(encryptedData) {
    const colonIndex = encryptedData.indexOf(':');
    if (colonIndex === -1) {
      throw new Error('Неверный формат зашифрованных данных');
    }
    const syncHex = encryptedData.substring(0, colonIndex);
    const encryptedHex = encryptedData.substring(colonIndex + 1);

    if (!ByteUtils.isValidHex(syncHex) || syncHex.length !== 16) {
      throw new Error('Неверный формат синхропосылки');
    }

    if (!ByteUtils.isValidHex(encryptedHex)) {
      throw new Error('Неверный формат зашифрованных данных');
    }
    const syncMessage = ByteUtils.hexToBytes(syncHex);
    const encryptedBytes = ByteUtils.hexToBytes(encryptedHex);

    const gamma = this.generateGamma(syncMessage, encryptedBytes.length);
    const decryptedBytes = ByteUtils.xorBytes(encryptedBytes, gamma);

    return ByteUtils.bytesToString(decryptedBytes);
  }
  async encryptFile(file) {
    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) {
    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');
    });
  }
}
```