МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет-приложений)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ:

по дисциплине «Информационная безопасность»

Исполнитель

студент (ка) 3 курса группы 6 Розель Станислав Александрович

(Ф.И.О.)

Минск 2025

### Алгоритм Меркла-Хеллмана

Алгоритм Меркла-Хеллмана был реализован в оконом приложении. Данное приложение высчитывает сверхвозрастающую последовательность, нормальная последовательность, с их использованием происходит шифрование и расширование сообщение. В листинге 1.1 представлен код реализующий алгоритм.

|  |
| --- |
| class MerkleHellman {  constructor(bitsCount = 100, S = 8) {  this.bitsCount = bitsCount;  this.S = S;  this.privateKey = null;  this.publicKey = null;  this.n = null;  this.a = null;  }  generatePrivateKey() {  const sequence = [];  let sum = BigInt(1);  for (let i = 0; i < this.S; i++) {  if (i === 0) {  sequence.push(this.getRandomBigInt(BigInt(2), BigInt(10)));  } else {  const nextValue = this.getRandomBigInt(sum.add(BigInt(1)), sum.multiply(BigInt(2)));  sequence.push(nextValue);  }  sum = sum.add(sequence[i]);  }  const lastElement = sequence[sequence.length - 1];  const bitsInLast = lastElement.bitLength().toJSNumber();    if (bitsInLast < this.bitsCount) {  const difference = this.bitsCount - bitsInLast;  const multiplier = BigInt(2).pow(difference);    for (let i = 0; i < sequence.length; i++) {  sequence[i] = sequence[i].multiply(multiplier);  }  }  this.privateKey = sequence;  return sequence;  }  generateModulusAndMultiplier() {  if (!this.privateKey) {  throw new Error("Сначала необходимо сгенерировать приватный ключ");  }    const sum = this.privateKey.reduce((acc, val) => acc.add(val), BigInt(0));    this.n = this.getRandomBigInt(sum.add(BigInt(1)), sum.multiply(BigInt(2)));    let attempts = 0;  do {  this.a = this.getRandomBigInt(BigInt(2), this.n.subtract(BigInt(1)));  attempts++;  if (attempts > 100) {  this.n = this.getRandomBigInt(sum.add(BigInt(1)), sum.multiply(BigInt(2)));  attempts = 0;  }  } while (this.gcd(this.a, this.n).notEquals(BigInt(1)));  return { n: this.n, a: this.a };  }  generatePublicKey() {  if (!this.privateKey) {  this.generatePrivateKey();  }    if (!this.n || !this.a) {  this.generateModulusAndMultiplier();  }    this.publicKey = this.privateKey.map(bi => this.a.multiply(bi).mod(this.n));  return this.publicKey;  }  encrypt(message) {  if (!this.publicKey) {  this.generatePublicKey();  }    if (this.publicKey.length > 0 && typeof this.publicKey[0] !== 'object') {  this.publicKey = this.publicKey.map(x => BigInt(x));  }    const binaryArray = this.textToBinaryArray(message);  const encryptedChunks = [];    for (let i = 0; i < binaryArray.length; i += this.S) {  const chunk = binaryArray.slice(i, i + this.S);    while (chunk.length < this.S) {  chunk.push(0);  }    let sum = BigInt(0);  for (let j = 0; j < chunk.length; j++) {  if (chunk[j] === 1) {  sum = sum.add(this.publicKey[j]);  }  }    encryptedChunks.push(sum);  }    return encryptedChunks;  }  decrypt(encryptedMessage) {  if (!this.privateKey || !this.n || !this.a) {  throw new Error("Необходимо сначала сгенерировать закрытый ключ");  }    if (this.privateKey.length > 0 && typeof this.privateKey[0] !== 'object') {  this.privateKey = this.privateKey.map(x => BigInt(x));  }  if (typeof this.n !== 'object') {  this.n = BigInt(this.n);  }  if (typeof this.a !== 'object') {  this.a = BigInt(this.a);  }    const aInverse = this.modInverse(this.a, this.n);  const decryptedBinary = [];  const encryptedBigInt = encryptedMessage.map(x =>  typeof x === 'object' ? x : BigInt(x)  );    for (const encryptedChunk of encryptedBigInt) {  const transformedSum = encryptedChunk.multiply(aInverse).mod(this.n);  const bits = new Array(this.S).fill(0);  let remaining = transformedSum;  for (let i = this.privateKey.length - 1; i >= 0; i--) {  if (remaining.compare(this.privateKey[i]) >= 0) {  bits[i] = 1;  remaining = remaining.subtract(this.privateKey[i]);  }  }    decryptedBinary.push(...bits);  }    let decryptedText = '';  for (let i = 0; i < decryptedBinary.length; i += this.S) {  if (i + this.S <= decryptedBinary.length) {  const charBits = decryptedBinary.slice(i, i + this.S);  const charCode = parseInt(charBits.join(''), 2);  if (charCode > 0 && charCode <= 1114111) {  decryptedText += String.fromCharCode(charCode);  }  }  }  return decryptedText.replace(/\0+$/, '');  } |

Листинг 1.1 – Реализация алгоритма Меркла-Хеллмана

На рисунке 1.1 представлен пример генерации закрытого и открытого ключа, а так же шифрование сообщение.

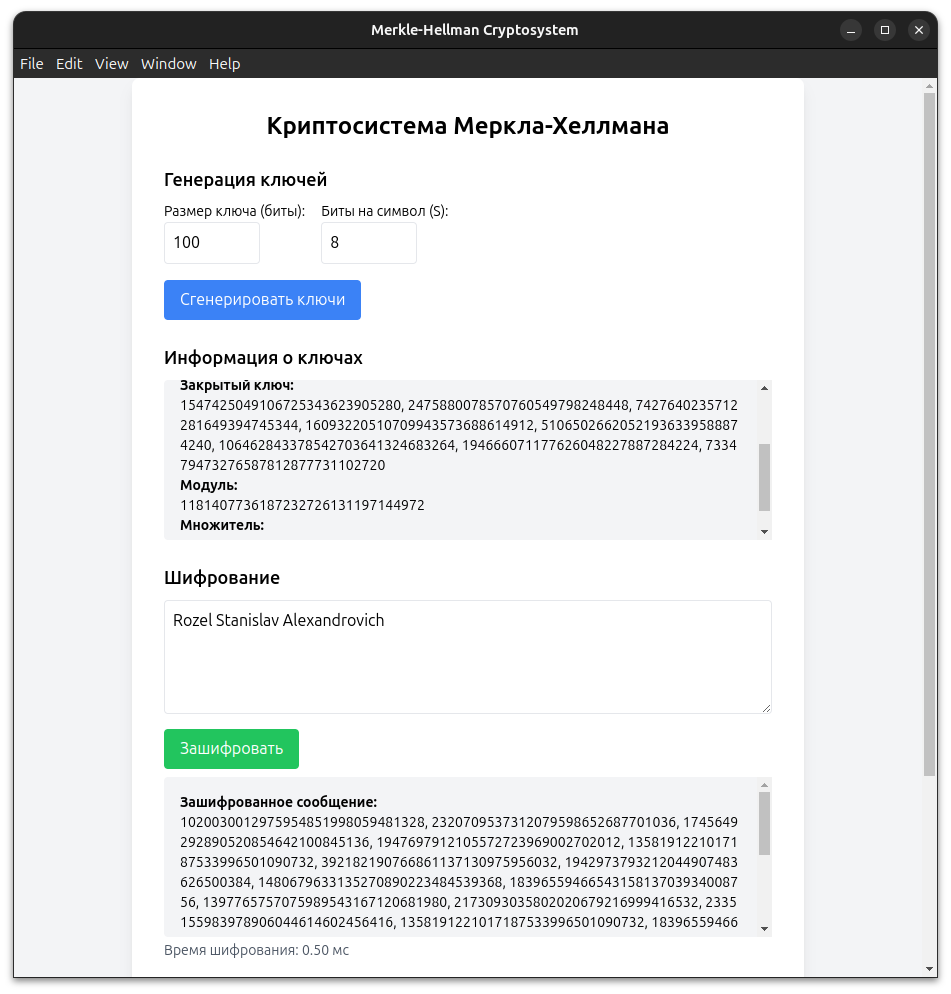


Рисунок 1.1 - Работа генератора и шифрования

На рисунке 1.2 представлена работа расшифрования сообщение их его зашифрованного аналога.

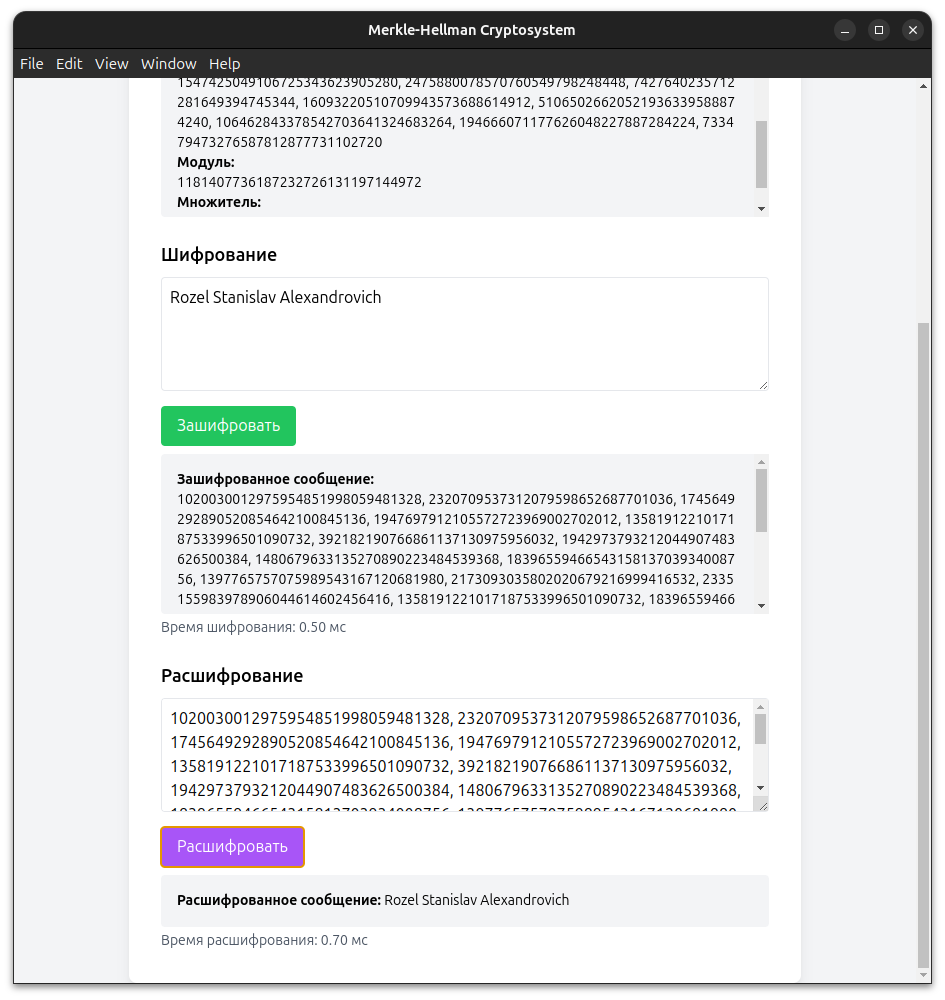


Рисунок 1.2 - Расшифрование сообщения