Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №9

**ИССЛЕДОВАНИЕ АССИМЕТРИЧНЫХ ШИФРОВ**

Студент: Сосновец М.И.

ФИТ 3 курс 4 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для асимметричного зашифрования/расшифрования.
3. Выполнить анализ криптостойкости асимметричных шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретические сведения**

Две известные нам проблемы, связанные с практическим использованием симметричных криптосистем, стали важными побудительными мотивами для разработки принципиально нового класса методов шифрования: криптографии с открытым ключом, или асимметричной криптографии.

В основу асимметричной криптографии положена идея использовать ключи парами: один – для зашифрования (открытый, или публичный, ключ), другой – для расшифрования (тайный ключ). Отметим, что указанная пара ключей принадлежит получателю зашифрованного сообщения. Все алгоритмы шифрования с открытым ключом основаны на использовании односторонних функций, к числу которых, как известно, относится вычисление дискретного логарифма.

Определение 1. Односторонней функцией (one-way function) называется математическая функция, которую относительно легко вычислить, но трудно найти по значению функции соответствующее значение аргумента, т. е. зная х, легко вычислить f(x), но по известному f(x) трудно найти подходящее значение x.

Алгоритмы шифрования с открытым ключом можно использовать для решения следующих задач:

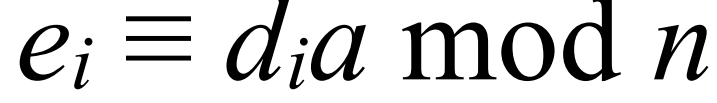
* зашифрования/расшифрования передаваемых и хранимых данных в целях их защиты от несанкционированного доступа;
* формирования цифровой подписи под электронными документами;
* распределения секретных ключей, используемых далее при шифровании документов симметричными методами. В данной работе мы будем работать над аспектами решения первой из указанных задач. По мнению Диффи и Хеллмана, алгоритм шифрования с открытым ключом должен:
* вычислительно легко создавать пару (открытый ключ e – закрытый ключ d);
* вычислительно легко зашифровывать сообщение Mi открытым ключом;
* вычислительно легко расшифровывать сообщение Ci, используя закрытый ключ;
* обеспечивать непреодолимую вычислительную сложность определения соответствующего закрытого ключа при известном открытом ключе;
* обеспечивать непреодолимую вычислительную сложность восстановления исходного (открытого сообщения Mi) зная только открытый ключ и зашифрованное сообщение Ci.

Алгоритм разработан Р. Мерклом и М. Хеллманом. Это первый алгоритм шифрования с открытым ключом широкого назначения.

Определение 2. Ранцевый (рюкзачный) вектор S = (s1, ..., sz) – это упорядоченный набор из z, z ≥ 3, различных натуральных чисел si. Входом задачи о ранце (рюкзаке) называем пару (S, S), где S – рюкзачный вектор, а S – натуральное число.

Суть метода для шифрования состоит в том, что существуют две различные задачи укладки ранца: одна из них решается легко и характеризуется линейным ростом трудоемкости, а другая решается трудно. Легкий для укладки ранец можно трансформировать в трудный. Трудный для укладки ранец применяется в качестве открытого ключа, который легко использовать для зашифрования, но невозможно – для расшифрования. В качестве закрытого ключа применяется легкий для укладки ранец, который предоставляет простой способ расшифрования сообщения.

Определение 3. Сверхвозрастающей называется последовательность, в которой каждый последующий член больше суммы всех предыдущих.

Для получения открытого ключа e (e = {ei}, i = 1, …, z) все значения закрытого ключа умножаются на некоторое число a по модулю n: 

Значение модуля n должно быть больше суммы всех чисел последовательности; кроме того, НОД (а, n) = 1.

Для расшифрования сообщения получатель (используя свой тайный ключ d: сверхвозрастающую последовательность) должен сначала определить такое обратное к а число а–1 , что аа–1 mod n ≡ 1.

Для вычисления обратных чисел по модулю можно использовать известный нам расширенный алгоритм Евклида. После определения обратного числа каждое значение шифрограммы (ci) преобразуется в соответствии со следующим соотношением: Si ≡ ciа–1 mod n.

Полученное на основании последней формулы для каждого блока число далее рассматривается как заданный вес ранца, который следует упаковать по изложенному выше алгоритму, используя сверхвозрастающую последовательность (тайный ключ получателя).

Криптостойкость алгоритма во многом определяется скоростью (временем) поиска нужного варианта укладки ранца. Понятно, что для последовательности из шести-десяти или немногим более того элементов нетрудно решить задачу укладки ранца, даже если последовательность не является сверхвозрастающей. При практической же реализации алгоритма ранец должен содержать не менее нескольких сотен элементов. Длина каждого члена сверхвозрастающей последовательности должна быть несколько сотен бит, а длина числа n – от 100 до 200 бит. Для получения этих значений практические реализации алгоритма используют генераторы ПСП.

Ранцевые криптосистемы не являются криптостойкими. А. Шамир и Р. Циппел обнаружили, что зная числа а, a–1 и n («секретную лазейку»), можно восстановить сверхвозрастающую последовательность по нормальной последовательности [5].

**Практическое задание**

1. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами.

В основе вычислений – кодировочные таблицы Base64 и ASCII.

Приложение должно реализовывать следующие операции:

* генерация сверхвозрастающей последовательности (тайного ключа); старший член последовательности – 100-битное число; в простейшем случае принимается z = 6 (для кодировки Base64) и z = 8 (для кодировки ASCII);
* вычисление нормальной последовательности (открытого ключа);
* зашифрование сообщения, состоящего из собственных фамилии, имени и отчества;
* расшифрование сообщения;
* оценка времени выполнения операций зашифрования и расшифрования.

Сверхвозрастающая последовательность — это ключевое понятие в криптографии, особенно в алгоритме «рюкзака» (Merkle-Hellman knapsack), который был одним из первых асимметричных шифров. За реализацию сверхвозрастающей последовательности отвечает функция generateSequence и реализация показана в листинге 1.1.

|  |
| --- |
| function generateSequence(*z*, *length* = 200) {      let sequence = [];      let sum = BigInt(0);      for (let i = 0; i < *length*; i++) {          let nextValue = sum + BigInt(Math.floor(Math.random() \* Math.pow(2, *z*)) + 1);          if (i === *length* - 1) {              nextValue = sum + (BigInt(2n) \*\* BigInt(100)); }          sequence.push(nextValue);          sum += nextValue;}      return sequence;} |

Листинг 1.1 – Реализация функции generateSequence

Функция generateSequence генерирует сверхвозрастающую последовательность, где каждый элемент больше суммы всех предыдущих, последний элемент последовательности является 100-битным числом, параметр z определяет диапазон генерации начальных элементов (6 для Base64, 8 для ASCII), длина последовательности по умолчанию - 200 элементов.

Далее была реализована нормальная последовательность или открытый ключ, представляющая собой преобразованную версию сверхвозрастающей последовательности, которая используется как открытый ключ в криптосистеме на основе рюкзака (Merkle-Hellman), за ее реализацию отвечает функция generateNormalSequence и показана в листинге 1.2.

|  |
| --- |
| function generateNormalSequence(*superSequence*) {      const M = *superSequence*.reduce((*a*, *b*) => *a* + BigInt(*b*), BigInt(0)) + BigInt(1);      let W;      do {          W = BigInt(Math.floor(Math.random() \* Number(M - BigInt(1))) + 1);      } while (gcd(W, M) !== BigInt(1));      const normalSequence = *superSequence*.map(*si* => ((BigInt(*si*) \* W) % M));      return { normalSequence: normalSequence.map(*n* => *n*.toString()), M: M.toString(), W: W.toString() };} |

Листинг 1.2 – Реализация функции generateNormalSequence

Процесс шифрования состоит из следующих этапов:

1. Кодирование текста в бинарный формат

* Если используется кодировка ASCII, каждый символ преобразуется в 8-битное представление.
* Для Base64 каждый символ кодируется 6 битами, предварительно переведя всё сообщение в формат Base64.

1. Разбиение бинарной строки на блоки

* Длина каждого блока соответствует размеру нормальной последовательности (открытого ключа).
* Если бит в блоке равен 1, соответствующий элемент из открытого ключа включается в сумму.

1. Формирование шифротекста

* Каждый блок преобразуется в число, равное сумме выбранных элементов нормальной последовательности.
* Результатом является массив чисел, представляющих зашифрованное сообщение.

Реализация шифрования показана функцией encryptMessage в листинге 1.3.

|  |
| --- |
| async function encryptMessage() {  const message = document.getElementById('fio').value;  const fioError = document.getElementById('fioError');  const encryptedMessage = document.getElementById('encrypted-message');  fioError.classList.remove('show');  encryptedMessage.classList.remove('show');  if (!normalSequenceFull || normalSequenceFull.length === 0) {  fioError.textContent = 'Пожалуйста, сгенерируйте ключи перед шифрованием';fioError.classList.add('show');return;}  if (!message) {fioError.textContent = 'Пожалуйста, введите ФИО'; fioError.classList.add('show');return;}  const startTime = performance.now();  const encoder = new TextEncoder();  const messageBytes = encoder.encode(message);  let binaryMessage = "";  for (let byte of messageBytes) {  binaryMessage += byte.toString(2).padStart(8, '0');}  fioError.textContent = `Сообщение слишком длинное для текущего ключа! Длина сообщения: ${binaryMessage.length} бит, длина ключа: ${normalSequenceFull.length} бит`;fioError.classList.add('show');return;}  let encrypted = BigInt(0);  for (let i = 0; i < binaryMessage.length; i++) {  if (binaryMessage[i] === '1') {  encrypted += BigInt(normalSequenceFull[i]);}}  const endTime = performance.now();const encryptionTime = (endTime - startTime).toFixed(4);  console.log('Input message:', message);  console.log('Binary message:', binaryMessage);  console.log('Encrypted:', encrypted.toString());  document.getElementById('encrypted-text').textContent = encrypted.toString();  document.getElementById('encryption-time').textContent = `Время шифрования: ${encryptionTime} мс`;  encryptedMessage.classList.add('show');} |

Листинг 1.3 – Реализация функции encryptMessage

Процесс расшифрования включает следующие шаги:

1. Вычисление обратного элемента

* Находится число, обратное множителю W по модулю M, необходимое для преобразования шифротекста.

1. Восстановление исходных сумм

* Каждое число из шифротекста умножается на обратный элемент по модулю M, что позволяет получить исходные суммы элементов сверхвозрастающей последовательности.

1. Решение задачи о рюкзаке

* Для каждой суммы определяется комбинация элементов сверхвозрастающей последовательности, которая её образует.
* На основе этого восстанавливается бинарная строка.

1. Декодирование в текст

* Бинарная строка преобразуется в символы в соответствии с выбранной кодировкой (ASCII или Base64).
* Если использовалось Base64, выполняется дополнительное декодирование из Base64 в исходный текст.

Результат зашифрования при значении 6 показан на рисунке 1.1, а при значении на рисунке 1.2.

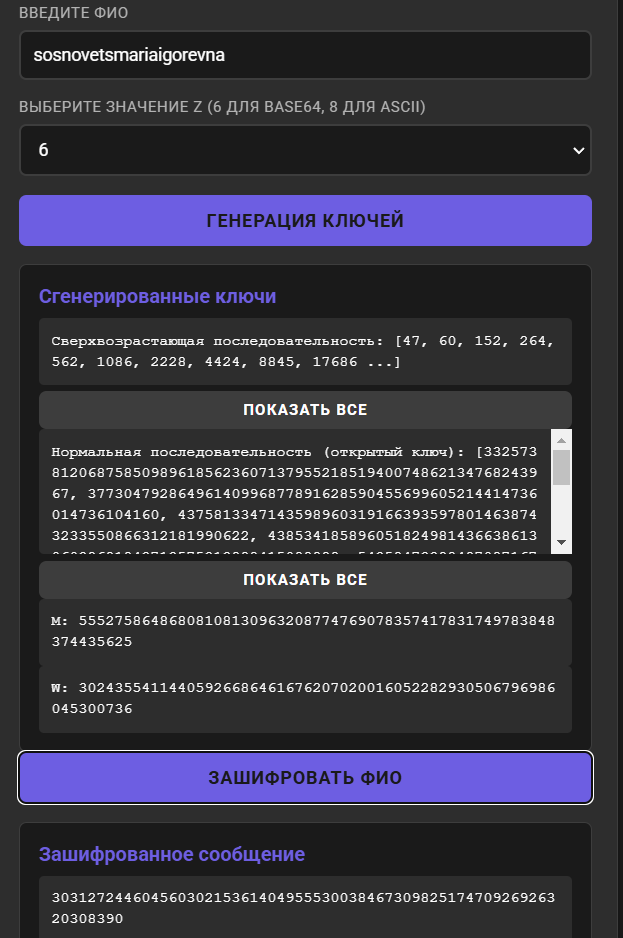


Рисунок 1.1 – Результат зашифрования при значении 6

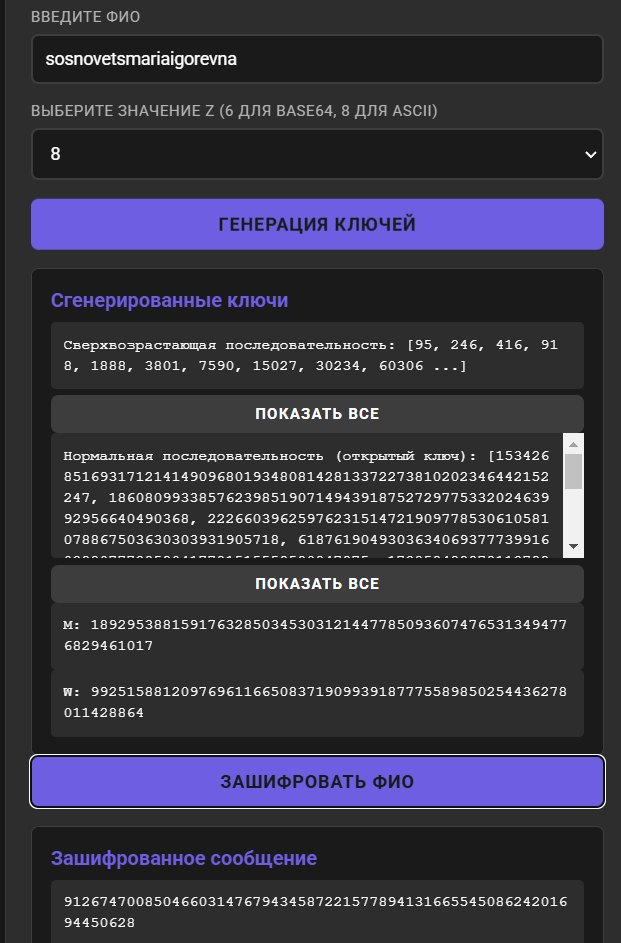


Рисунок 1.2 – Результат зашифрования при значении 8

Реализация шифрования показана функцией dencryptMessage в листинге 1.4.

|  |
| --- |
| async function decryptMessage() {  const encryptedInput = document.getElementById('encrypted-input').value || document.getElementById('encrypted-text').textContent;  const encryptedInputError = document.getElementById('encryptedInputError');  const decryptedMessage = document.getElementById('decrypted-message');let encrypted;encryptedInputError.classList.remove('show');decryptedMessage.classList.remove('show');  try {encrypted = BigInt(encryptedInput);} catch (error) {console.error('Ошибка при преобразовании зашифрованного сообщения:', error);  encryptedInputError.textContent = 'Пожалуйста, введите корректное зашифрованное сообщение (число)';  encryptedInputError.classList.add('show');return;}  const M = BigInt(document.getElementById('M').textContent.replace('M: ', ''));          const W = BigInt(document.getElementById('W').textContent.replace('W: ', ''));  if (!superSequenceFull || superSequenceFull.length === 0){encryptedInputError.textContent = 'Пожалуйста, сгенерируйте ключи перед расшифровкой';encryptedInputError.classList.add('show'); return; }  const startTime = performance.now();const WInverse = modularInverse(W, M);  let decryptedSum = (encrypted \* WInverse) % M;  console.log('WInverse:', WInverse.toString());  console.log('Decrypted sum:', decryptedSum.toString());  let binaryMessage = new Array(superSequenceFull.length).fill('0');  for (let i = superSequenceFull.length - 1; i >= 0; i--) {if (BigInt(superSequenceFull[i]) <= decryptedSum) {  binaryMessage[i] = '1';decryptedSum -= BigInt(superSequenceFull[i]);}}  const binaryString = binaryMessage.join('');console.log('Decrypted binary:', binaryString);  const message = document.getElementById('fio').value;  encoder = new TextEncoder();  const messageBytes = encoder.encode(message);  const expectedBinaryLength = messageBytes.length \* 8;  const trimmedBinary = binaryString.slice(0, expectedBinaryLength);  console.log('Expected binary length:', expectedBinaryLength);  console.log('Trimmed binary:', trimmedBinary);  let byteArray = [];  for (let i = 0; i < trimmedBinary.length; i += 8) {  const byte = trimmedBinary.slice(i, i + 8);  if (byte.length === 8) {byteArray.push(parseInt(byte, 2)); }}console.log('Byte array:', byteArray);  const decoder = new TextDecoder('utf-8');  const decryptedMessageText = decoder.decode(new Uint8Array(byteArray));const endTime = performance.now();const decryptionTime = (endTime - startTime).toFixed(4);  console.log('Decrypted message:', decryptedMessageText);  document.getElementById('decrypted-text').textContent = decryptedMessageText;  document.getElementById('decryption-time').textContent = `Время расшифровки: ${decryptionTime} мс`;decryptedMessage.classList.add('show'); |

Листинг 1.4 – Реализация функции dencryptMessage

Результат расшифрования со значением 6 показан на рисунке 1.3, а со значением 8 на рисунке 1.4.

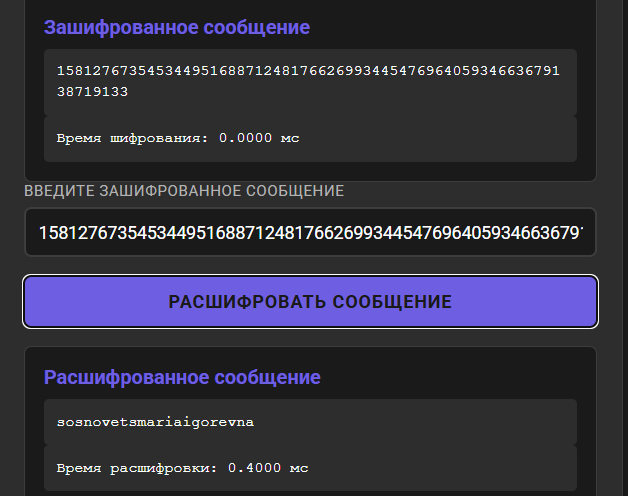


Рисунок 1.3 – Результат расшифрования при значении 6

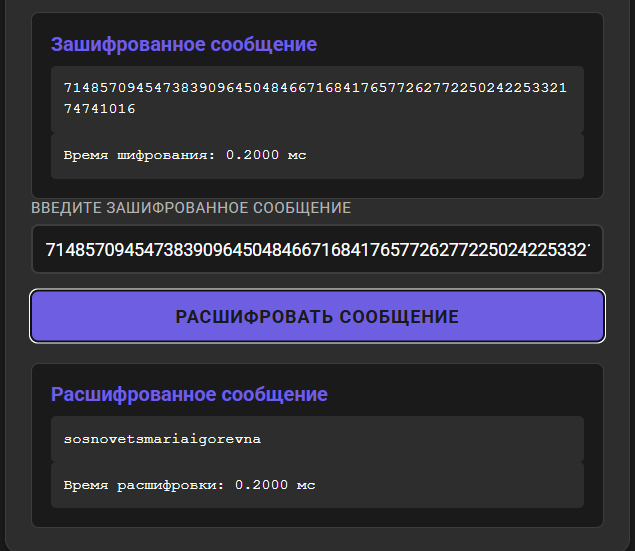


Рисунок 1.4 – Результат расшифрования при значении 8

Далее было измерено время выполнения операций зашифрования и расшифрования. Как можно заметить из рисунка 1.5 время зашифрования составляет: 0.200 мс , а время расшифрования: 0.300 мс.

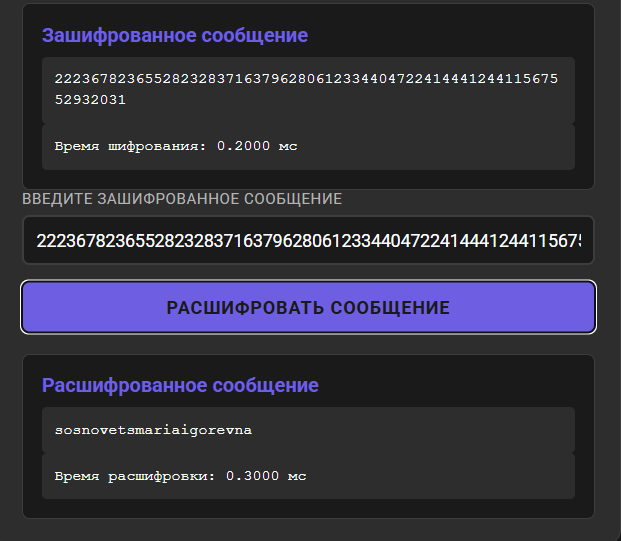


Рисунок 1.5 – Результат измерения времени шифрования и расшифрования

2. Проанализировать время выполнения операций зашифрования/расшифрования при увеличении числа членов ключевой последовательности; при использовании разных таблиц кодировки.

Анализ времени выполнения операций зашифрования/расшифрования при увеличении числа членов ключевой последовательности показан на рисунке 1.6.

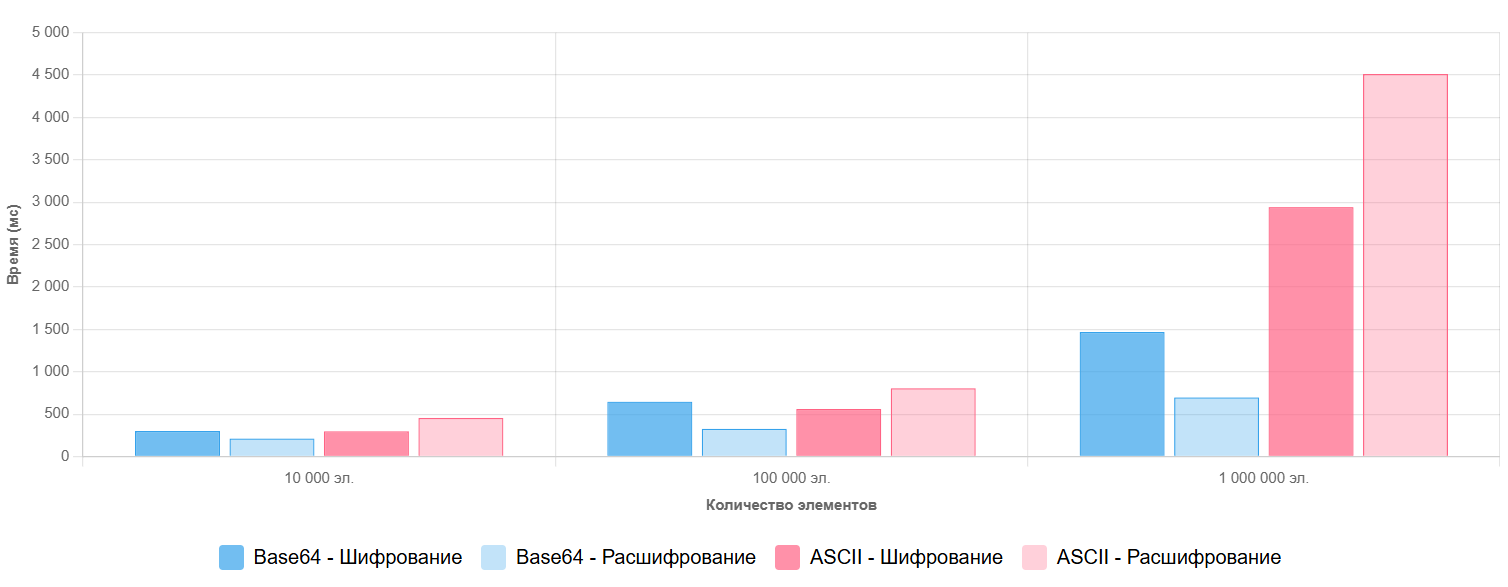


Рисунок 1.6 – Анализ времени выполнения операций зашифрования/расшифрования при увеличении числа членов ключевой последовательности

Анализ времени выполнения операций зашифрования/расшифрования при использовании разных таблиц кодировки показан на рисунке 1.7.

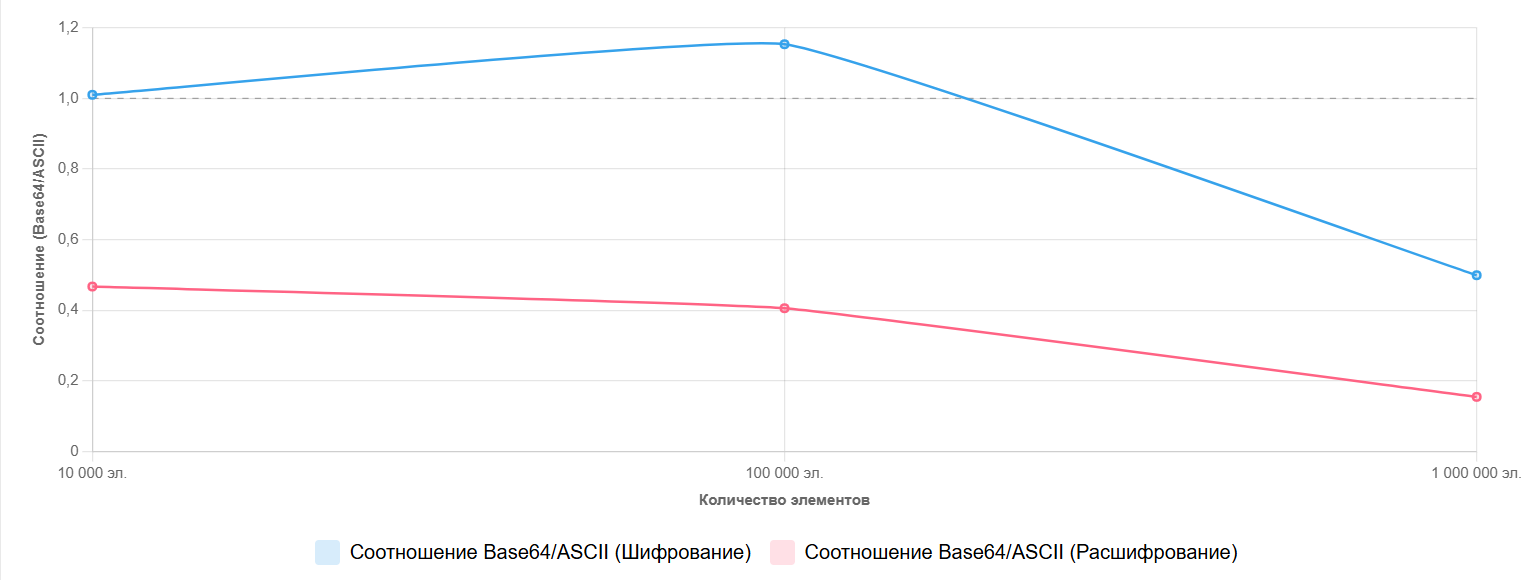


Рисунок 1.7 – Анализ времени выполнения операций зашифрования/расшифрования при использовании разных таблиц кодировки

В ходе сравнительного анализа времени выполнения операций шифрования и расшифрования были выявлены ключевые закономерности при использовании различных таблиц кодировки и при изменении длины ключевой последовательности.

При сравнении кодировок Base64 и ASCII установлено, что Base64 демонстрирует более высокую производительность, особенно при работе с большими объемами данных. Это объясняется более компактным 6-битным представлением символов по сравнению с 8-битным ASCII, что приводит к уменьшению количества необходимых битовых операций. Разница в скорости становится особенно заметной с ростом объема обрабатываемой информации, хотя для коротких сообщений она может быть незначительной. При этом преимущество Base64 сохраняется как при шифровании, так и при расшифровании, хотя во втором случае разница менее выражена.

Анализ влияния длины ключевой последовательности показал нелинейный, близкий к экспоненциальному, рост времени выполнения операций. Увеличение количества элементов последовательности существенно повышает сложность вычислений: при шифровании требуется больше операций сложения, а при расшифровании - больше итераций для решения задачи о рюкзаке.

**Вывод**

Данная лабораторная работа позволила получить практические навыки реализации асимметричного шифрования, оценить его криптографическую стойкость и производительность. Разработанное приложение может быть использовано для дальнейшего изучения рюкзачных криптосистем и их оптимизации.