МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Исследование асимметричных шифров

Студент: Евсеенко В. П.

ФИТ 3 курс 4 группа

Преподаватель: Савельева М. Г.

Минск 2025

Содержание

[Введение 3](#_Toc196347610)

[1 Теоретические сведения 4](#_Toc196347611)

[2 Практическое задание 5](#_Toc196347612)

[Оценка времени 10](#_Toc196347613)

[Вывод 11](#_Toc196347614)

# **Введение**

Цель лабораторной работы:

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

Задачи:

* закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров;
* разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для асимметричного зашифрования/расшифрования;
* выполнить анализ криптостойкости асимметричных шифров;
* оценить скорость зашифрования/расшифрования реализованных шифров.

# **1 Теоретические сведения**

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.

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.

Ранцевый (рюкзачный) вектор *S* = (*s*1, ..., *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. Решением для входа (*S*, *S*) будет такое подмножество из *S*, сумма элементов которого равняется *S*.

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.

В качестве закрытого ключа *d* (легкого для укладки ранца) используется сверхвозрастающая последовательность, состоящая из *z* элементов: *d*1, *d*2, …, *dz*: *d* = {*di*}, *i* = 1, …, *z*.

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

Ранцевые криптосистемы не являются криптостойкими. А. Шамир и Р. Циппел обнаружили, что, зная числа *а*, *a*-1 и *n*, можно восстановить сверхвозрастающую последовательность по нормальной последовательности.

# **2 Практическое задание**

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.

В начале, нужно было разработать приложение, реализующее генерацию приватного и публичного ключа, а также осуществляющее зашифрование и расшифрование сообщения, состоящего из собственных фамилии, имени и отчества.

Функция generatePrivateKey принимает на вход начальное число последовательности и длину генерируемой последовательности. Начальное число последовательности – это случайное 100-битное число. Код функции представлен в листинге 2.1

|  |
| --- |
| export const generatePrivateKey = (initialNumber: BigInteger, z: number): BigInteger[] => {      const sequence: BigInteger[] = [];      let element: BigInteger = initialNumber;      let sum = initialNumber;      for (let i = 0; i < z; i++) {          sequence.push(element);          element = sum.add(bigInt(z));          sum = sum.add(element);      }      return sequence;  } |

Листинг 2.1 – Код функции генерации приватного ключа

Результат работы приложения с начальным числом 373977097140245897443365283506 и длиной *z* = 8 (для кодов ASCII) приведен на рисунке 2.1.

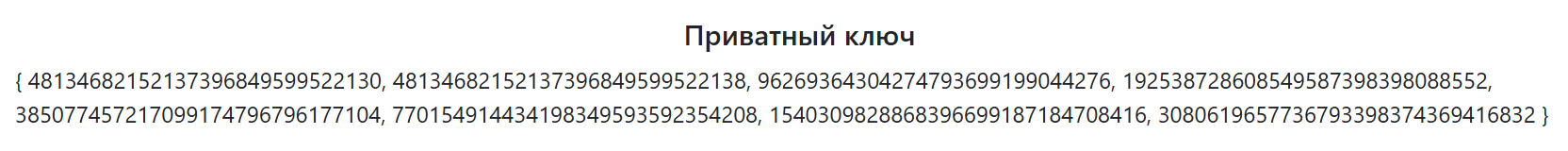


Рисунок 2.1 – Результат генерации последовательности с *z* = 8

Результат работы приложения с начальным числом 70959993794722279014361958728 и длиной *z* = 6 (для кодов base64) приведен на рисунке 2.2.

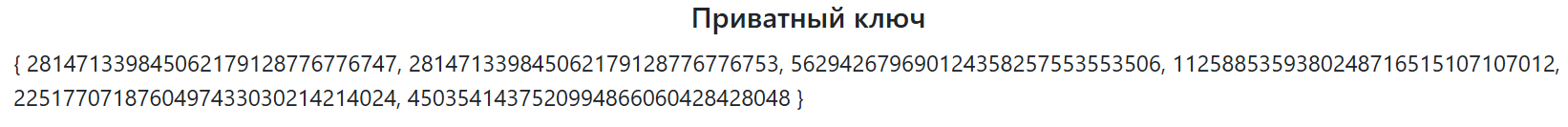


Рисунок 2.2 – Результат генерации последовательности c *z* = 6

Далее, на основе сгенерированного приватного ключа подбираются параметры *n* и *a* для публичного ключа. Значение модуля n должно быть больше суммы всех чисел последовательности; кроме того, НОД (*а*, *n*) = 1. Функция generatePublicKey принимает приватный ключ и параметры *a* и *n* и генерирует публичный ключ. Код функции представлен в листинге 2.2.

|  |
| --- |
| export const generatePublicKey = (privateKey: BigInteger[], a: BigInteger, n: BigInteger): BigInteger[] => {      const sequence: BigInteger[] = [];      let d: BigInteger;      let e: BigInteger;      for (let i = 0; i < privateKey.length; i++) {          d = privateKey[i];          e = d.multiply(a).mod(n);          sequence.push(e);      }      return sequence;  } |

Листинг 2.2 – Код функции, создающей публичный ключ

Результат генерации публичного ключа для сообщения в кодах ASCII представлен на рисунке 2.3.

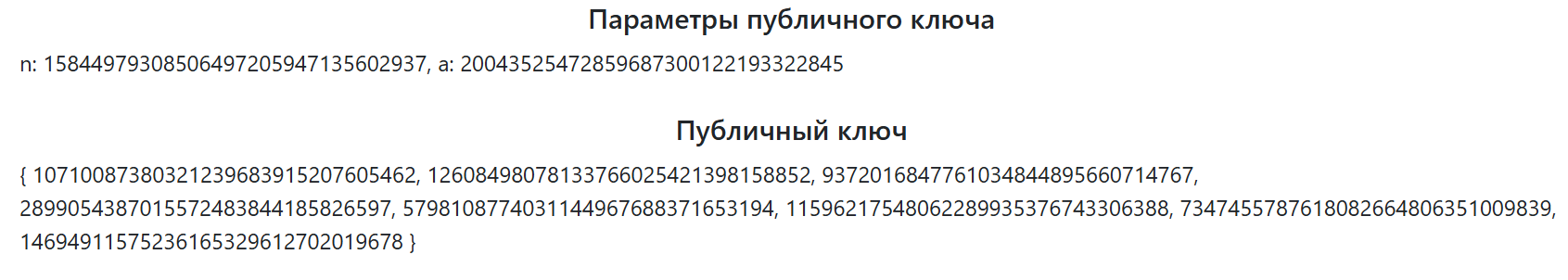


Рисунок 2.3 – Результат генерации публичного ключа для сообщения в кодах ASCII

Результат генерации публичного ключа для сообщения в кодах base64 представлен на рисунке 2.4.

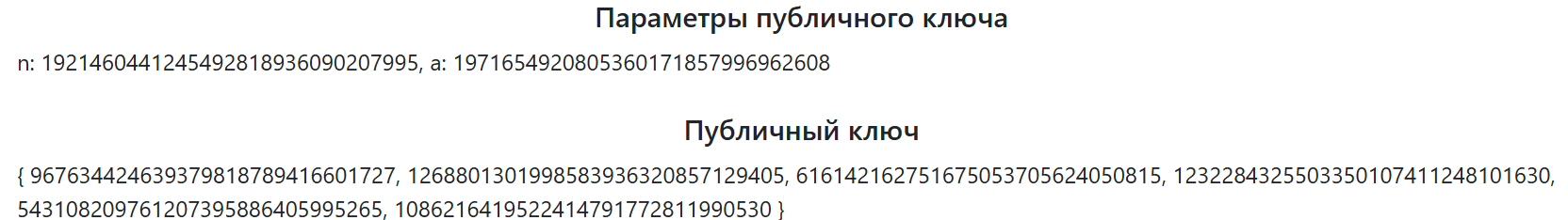


Рисунок 2.4 – Результат генерации публичного ключа для сообщения в кодах base64

Функция шифрования encrypt принимает на вход публичный ключ, открытый текст и кодировку сообщения и выполняет зашифрования сообщения на основе алгоритма укладки ранца. Код функции представлен в листинге 2.3.

|  |
| --- |
| export const encrypt = (publicKey: bigInt.BigInteger[], plaintext: string, encoding: Encoding): bigInt.BigInteger[] => {      const encryptedList: bigInt.BigInteger[] = [];      if (encoding === Encoding.BASE64) {          plaintext = base64Encode(plaintext);      }      plaintext.split('').forEach((b, index) => {          let binaryString;          if (encoding === Encoding.ASCII) {              binaryString = plaintext.charCodeAt(index).toString(2).padStart(8, '0');          } else  {              binaryString = convertBase64ToBinary(plaintext[index]);          }          const positions: number[] = [];          for (let i = 0; i < binaryString.length; i++) {              if (binaryString[i] === '1') {                  positions.push(i);              }          }          let sum = bigInt.zero;          positions.forEach(position => {              if (position < publicKey.length) {                  sum = sum.add(publicKey[position]);              }          });          encryptedList.push(sum);      });      return encryptedList;  } |

Листинг 2.3 – Код функции шифрования

Результат шифрования текста «Evseenko Viktoriya Pavlovna» (в кодах ASCII) представлен на рисунке 2.5.

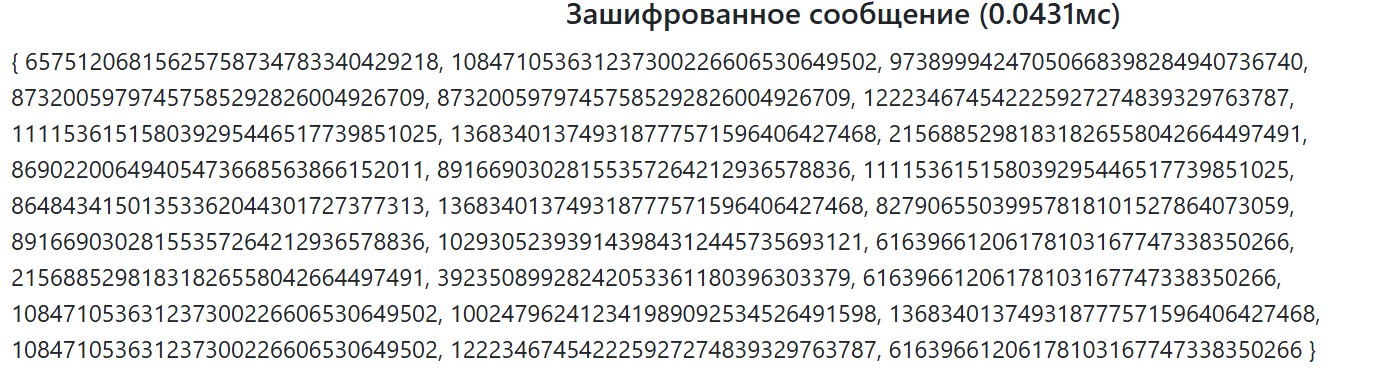


Рисунок 2.5 – Шифрование текста, представленного в кодах ASCII

Результат шифрования текста «Evseenko Viktoriya Pavlovna» (в кодах base64 RXZzZWVua28gVmlrdG9yaXlhIFBhdmxvdm5h) представлен на рисунке 2.6.

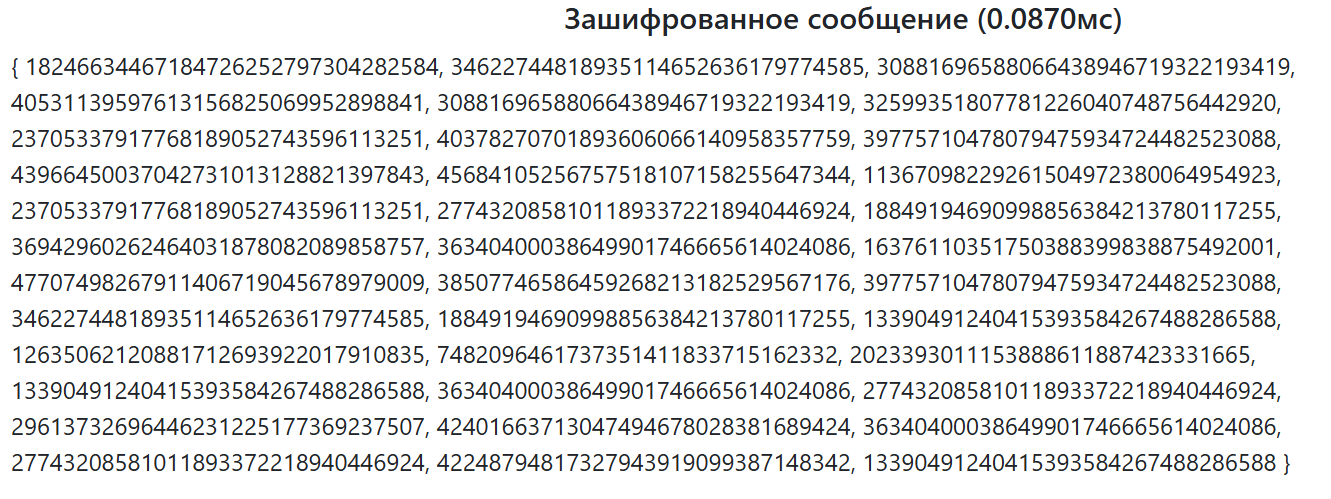


Рисунок 2.6 – Шифрование текста, представленного в кодах base64

Функция дешифрования decrypt принимает на вход приватный ключ, зашифрованный текст и параметры *a* и *n*. Код функции представлен в листинге 2.4.

|  |
| --- |
| export const decrypt = (privateKey: BigInteger[], encryptedText: BigInteger[], a: BigInteger, n: BigInteger): {decoded: Uint8Array, binary: string} => {      let decryptedBytes: number[] = [];      let binaryResult: string = "";      let inverse: BigInteger = getInverseNumber(a, n);      for (let cipher of encryptedText) {let decryptedValue: BigInteger = cipher.times(inverse).mod(n);let binaryString: string = getDecryptedBinary(decryptedValue, privateKey);          binaryResult += binaryString;          let decryptedByte: number = parseInt(binaryString, 2);decryptedBytes.push(decryptedByte);}      return {decoded: new Uint8Array(decryptedBytes), binary: binaryResult};} |

Листинг 2.3 – Код функции расшифрования

Результат дешифрования текста в кодах ASCII представлен на рисунке 2.7.

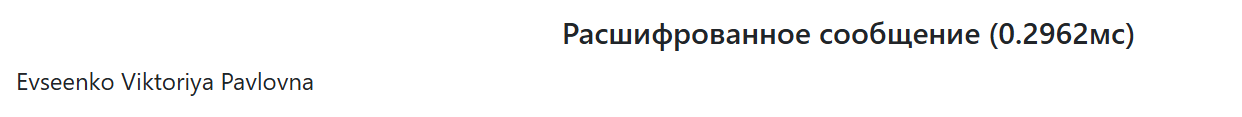


Рисунок 2.7 – Дешифрование текста, представленного в кодах ASCII

Результат дешифрования текста в кодах base64 представлен на рисунке 2.8.

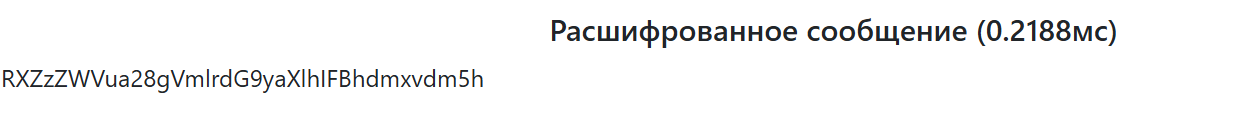


Рисунок 2.8 – Дешифрование текста, представленного в кодах base64

# **Оценка времени**

Приложение выполняет оценку скорости шифрования и дешифрования сообщения, представленного в кодах ASCII и base64. Результат нескольких попыток шифрования представлен на рисунках 2.9 и 2.10.

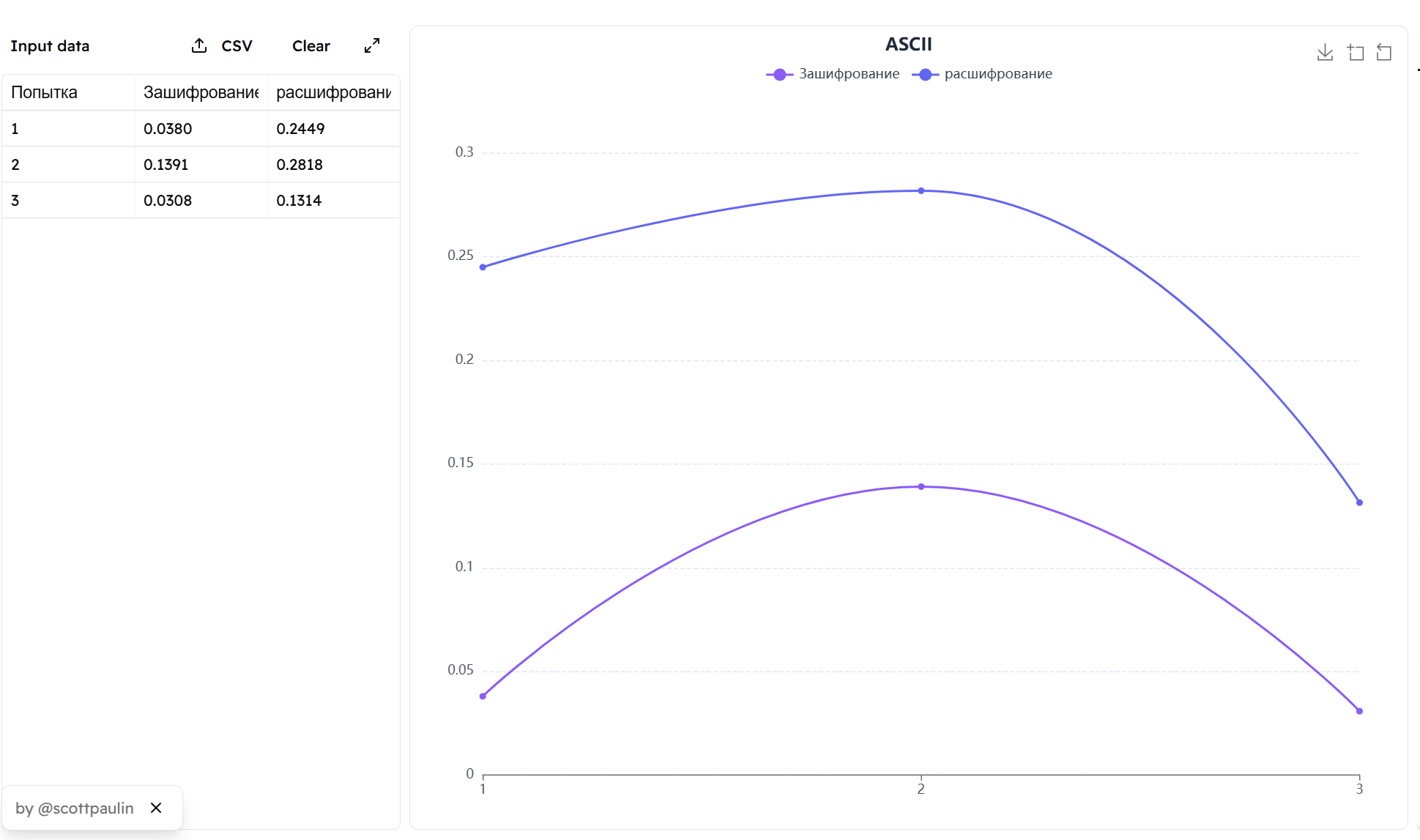


Рисунок 2.9 – Оценка времени операций зашифрования/расшифрования в кодах ASCII

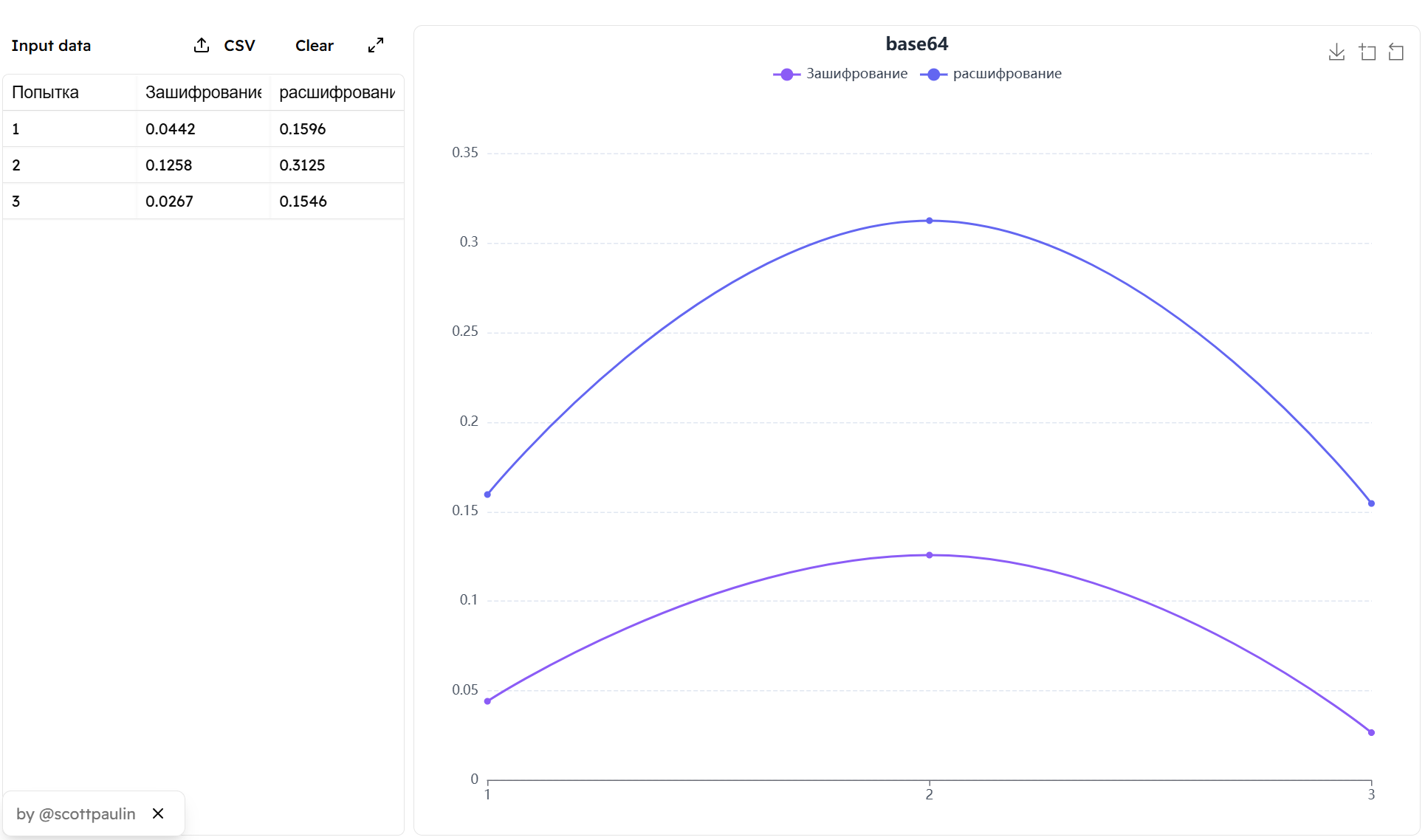


Рисунок 2.10 – Оценка времени операций зашифрования/расшифрования в кодах base64

# **Вывод**

В ходе выполнения лабораторной работы были изучены основные принципы работы асимметричных шифров, а также приобретены практические навыки разработки приложений, реализующих процессы генерации ключей, зашифрования и расшифрования информации. Были реализованы методы формирования ключевой информации с использованием регистров сдвига и проведено шифрование данных в двух кодировках: ASCII и Base64.

В результате была достигнута цель лабораторной работы: реализовано и протестировано приложение, обеспечивающее функционирование асимметричного шифрования. Были получены оценки скорости выполнения операций зашифрования и расшифрования, что позволило сделать предварительные выводы о производительности и эффективности реализованного алгоритма.

Также был проведён базовый анализ криптостойкости реализованного шифра, что позволило закрепить теоретические знания по криптографическим основам и практическим аспектам реализации безопасных протоколов обмена информацией.