Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №10

**ИССЛЕДОВАНИЕ АССИМЕТРИЧНЫХ ШИФРОВ RSA и Эль-Гамаля**

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 RSA и Эль-Гамаля.

**Задачи:**

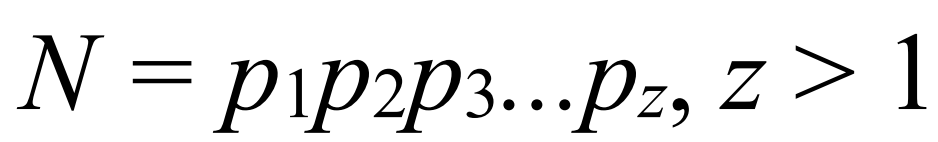
1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров RSA и Эль-Гамаля.
2. Разработать приложение для реализации асимметричного зашифрования/расшифрования на основе алгоритмов RSA и ЭльГамаля.
3. Выполнить анализ криптостойкости асимметричных шифров RSA и Эль-Гамаля.
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.

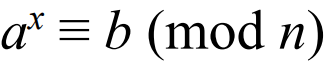
**Теоретические сведения**

Как отмечалось выше, асимметричная криптография основана на сложности решения некоторых математических задач. По существу, таких задач две:

* разложение больших чисел на простые сомножители (задача факторизации);
* вычисление дискретного логарифма в конечном поле, а также вычислительные операции над точками эллиптической кривой.

Эти задачи объединяет то, что они используют операцию получения остатка от целочисленного деления. В силу этого практически все системы асимметричного зашифрования/расшифрования основаны либо на проблеме факторизации (среди них – RSA), либо на проблеме дискретного логарифмирования (среди них – Эль-Гамаля).

Теорема 1: основная теорема арифметики. Всякое натуральное число N, кроме 1, можно представить как произведение простых множителей: 

Определение 1. Задача дискретного логарифмирования формулируется так: для данных целых чисел а и b, 1 < а, b < n, найти логарифм – такое целое число х, что  если такое число существует.

Теорема 2: китайская теорема об остатках. В общем случае если разложение числа N на простые множители представляет собой p1p2…pt (некоторые простые числа могут встречаться несколько раз), то система уравнений (x mod pi) ≡ ai, где i = 1, 2, …, t, имеет единственное решение: x, меньшее N.

Иными словами, число (меньшее, чем произведение нескольких простых чисел) однозначно определяется своими вычетами по модулю от этих простых чисел. Китайской теоремой об остатках можно воспользоваться для решения полной системы уравнений в том случае, если известно разложение числа N на простые множители.

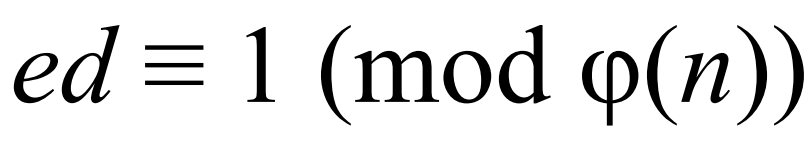
Алгоритм RSA. Рассматриваемый алгоритм появился (1977 г.) после алгоритма ранца Меркла. Он стал первым полноценный алгоритмом с открытым ключом, который впоследствии стал одним из основных для шифрования и для электронных цифровых подписей.

Из всех предложенных алгоритмов с открытыми ключами RSA проще всего понять и реализовать. Он назван в честь трех его создателей: Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Эдлемана (Leonard Adleman).

Как было отмечено, безопасность RSA основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших простых чисел. Предполагается, что восстановление открытого текста по шифртексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

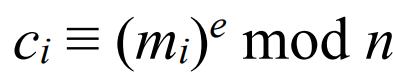
Для генерации двух ключей: тайного и открытого (а по сути – двух взаимосвязанных частей одного ключа, т. е. ключа, принадлежащего одному физическому лицу (или группе лиц), либо одному юридическому лицу), используются два больших случайных простых числа p и q. Для максимальной большей криптостойкости нужно выбирать p и q равной длины. Рассчитывается произведение: n = pq. Это есть один из трех компонент ключа, состоящего из чисел n, e, d.

Затем случайным образом выбирается второй компонент ключа (открытый ключ или ключ зашифрования, e, такой что e и (p – 1)(q – 1) являются взаимно простыми числами; вспомним, что (p – 1)(q – 1) = φ(n) – функция Эйлера). Б. Шнайер [5] рекомендует число е выбирать из ряда: 3, 17, 216 + 1.

Наконец, расширенный алгоритм Евклида используется для вычисления третьего компонента ключа: ключа расшифрования d такого, что выполняется условие: .

Таким образом, сформирован ключ, состоящий из трех чисел, которые в свою очередь образуют две вышеупомянутые взаимосвязанные части: открытый (публичный) ключ (e, n) и тайный ключ (d, n; на самом деле, как видим, тайным здесь является лишь первое из пары чисел).

Для зашифрования/расшифрования используется ключ получателя: отправитель шифрует сообщение открытым ключом, а получатель расшифровывает шифртекст своим тайным ключом.

Зашифрование. Если шифруется сообщение М, состоящее из r блоков: m1, m2, …, mi, …, mr, то шифртекст С будет состоять из такого же числа (r) блоков, представляемых числами: .

Расшифрование. Для расшифрования каждого зашифрованного блока производится вычисление вида: .

Алгоритм Эль-Гамаля. Предложен Т. Эль-Гамалем (T. El-Gamal) в 1985 г. Он может быть использован для решения трех основных криптографических задач: для зашифрования/расшифрования данных, для формирования цифровой подписи и для согласования общего ключа. Кроме того, возможны модификации алгоритма для схем проверки пароля, доказательства идентичности сообщения и другие варианты.

Как подчеркивалось выше, безопасность алгоритма Эль-Гамаля, как и безопасность алгоритма Диффи – Хеллмана, основана на трудности вычисления дискретных логарифмов.

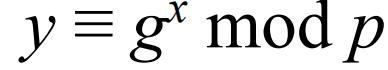
Алгоритм Эль-Гамаля фактически использует схему Диффи – Хеллмана, чтобы сформировать общий секретный ключ для абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот ключ.

И в случае шифрования, и в случае формирования цифровой подписи каждому пользователю необходимо сгенерировать пару ключей.

Рассматриваемый алгоритм отличается от алгоритма RSA несколькими параметрами и особенностями:

1. генерацией ключевой информации и числом компонент, составляющих ключ;
2. каждому блоку (символу) открытого сообщения в шифртексте на основе алгоритма Эль-Гамаля соответствуют 2 блока (в RSA – один-один);
3. в алгоритме Эль-Гамаля при зашифровании используется число (обозначим его k), которое практически никак не связано с ключевой информацией получателя и которое принимает (по определению) различные значения при зашифровании различных блоков сообщения.

Генерация ключевой информации. Выбирается простое число р. Выбирается число (g, g < p), являющееся первообразным корнем числа р – очень важный элемент с точки зрения безопасности алгоритма (см. ниже).

Далее выбирается число х (х < p) и вычисляется последний компонент ключевой информации: .

Определение 2. Первообразный корень (primary (residual) root) по модулю р является таким числом, что его степени (gi , 1 ≤ i ≤ p – 1) дают все возможные по модулю р вычеты (остатки), которые взаимно просты с p.

В силу использования случайной величины k шифр ЭльГамаля называют также шифром многозначной замены, а также схемой вероятностного шифрования.

**Практическое задание**

1. С помощью простого консольного приложения составить табличную или графическую форму зависимости времени вычисления параметра у, функционально заданного выражением вида:  ,от параметров: а (десятичные числа от 5 до 35; можно взять 1 или 2 числа), х (числа, желательно простые, из диапазона от 103 до 10100; для примера взять 5–10 чисел, равномерно распределенных в указанном диапазоне), n (для примера взять числа, в двоичном виде состоящие из 1024 и 2048 битов).

Реализация анализа графиков времени выполнения модульного возведения в степень представлена в листинге 1.1.

|  |
| --- |
| function isPrime(num) {  if (num < 2) return false;  for (let i = 2, sqrt = Math.sqrt(num); i <= sqrt; i++) {  if (num % i === 0) return false;  }  return true;  }  // Функция генерации большого числа заданной битовой длины  function generateBigNumber(bits) {  const bytes = Math.ceil(bits / 8);  let number;  do {  number = BigInt('0x' + crypto.randomBytes(bytes).toString('hex'));  } while (number % 2n === 0n);  return number;}  // Алгоритм модульного возведения в степень  function modularExponentiation(base, exponent, mod) {  let result = 1n;  base = base % mod;  while (exponent > 0) {  if (exponent % 2n === 1n) {  result = (result \* base) % mod;}  exponent = exponent / 2n;  base = (base \* base) % mod;  }return result;} |

Листинг 1.1 – Реализация модульного возведения в степень

В листинге 1.1 исследуется время выполнения операции модульного возведения в степень для различных комбинаций параметров:

* a: значения 5, 15, 25, 35
* x: простые числа в диапазоне от 103 до 1000 (5 значений)
* n: большие числа длиной 1024 и 2048 бит

На рисунках показаны графики зависимости времени от параметра x, зависимости времени от параметра n, зависимости времени от параметра a 1.1, 1.2, 1.3 соответственно.

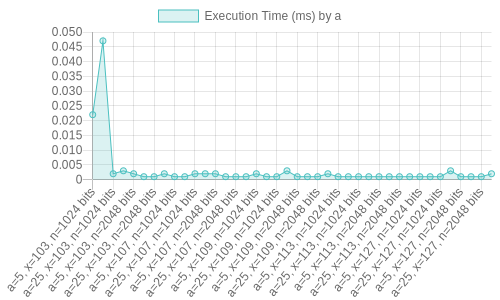


Рисунок 1.1 – График зависимости времени от параметра x

Как можно заметить из рисунка 1.1 время выполнения колеблется в диапазоне 0.0010-0.0022 мс, не наблюдается явной зависимости времени от значения x.

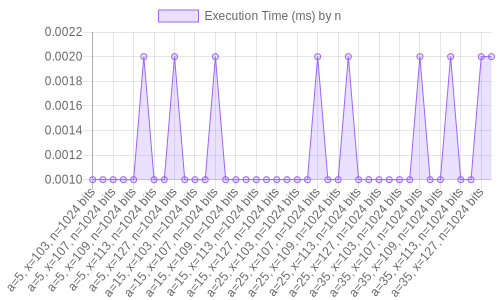


Рисунок 1.2 – График зависимости времени от параметра n

Как можно заметить из рисунка 1.2 аналогичные значения времени (0.0010-0.0022 мс), отсутствие заметной разницы между 1024-битными и 2048-битными числами, это неожиданно, так как для больших n операции должны выполняться дольше.

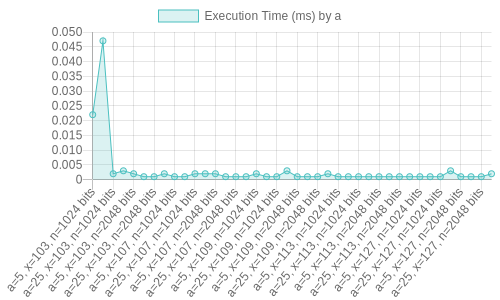


Рисунок 1.3 – График зависимости времени от параметра a

Как можно заметить из рисунка 1.3 Время выполнения 0.005-0.050 (значительно больше, чем на других графиках), возможно, это более реалистичные значения.

2. 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:

* зашифрование и расшифрование текстовых документов на основе алгоритмов RSA и Эль-Гамаля;
* определение времени выполнения операций.

Исходный текст для зашифрования – собственные фамилия, имя, отчество. Для численного представления блоков текста можно в том числе пользоваться указанными выше кодировочными таблицами.

Ключевую информацию для обоих алгоритмов можно сгенерировать самостоятельно либо воспользоваться, например, одной из утилит криптографической библиотеки OpenSSL, с помощью которой, в частности, можно сгенерировать ключевую информацию для алгоритма RSA.

RSA (Rivest-Shamir-Adleman) - это асимметричный криптографический алгоритм, работающий следующим образом:

1. Генерация ключей:
   * Выбираются два больших простых числа p и q
   * Вычисляется их произведение n = p \* q (модуль)
   * Вычисляется функция Эйлера φ(n) = (p-1)\*(q-1)
   * Выбирается открытая экспонента e (обычно 65537), взаимно простая с φ(n)
   * Вычисляется секретная экспонента d как мультипликативное обратное к e по модулю φ(n)
2. Шифрование:
   * Сообщение разбивается на блоки, каждый меньше n
   * Каждый блок m шифруется: c = m^e mod n
3. Расшифрование:
   * Зашифрованный блок c расшифровывается: m = c^d mod n

Генерация ключей показана в листинге 1.2.

|  |
| --- |
| const { publicKey, privateKey } = forge.pki.rsa.generateKeyPair({  bits: 2048, // Длина ключа в битах  e: 0x10001 // Открытая экспонента (65537)}); |

Листинг 1.2 – Реализация генерации ключей

Реализация операции шифрования показана в листинге 1.3.

|  |
| --- |
| const maxBlockSize = 245;let encryptedData = Buffer.alloc(0);for (let i = 0; i < data.length; i += maxBlockSize) {  const block = data.slice(i, Math.min(i + maxBlockSize, data.length));  const buffer = forge.util.createBuffer(block.toString('binary'), 'binary');  const encrypted = publicKey.encrypt(buffer.getBytes());  encryptedData = Buffer.concat([encryptedData, Buffer.from(encrypted, 'binary')]);} |

Листинг 1.3 – Реализация операции шифрования

Реализация операции расшифрования показана в листинге 1.4.

|  |
| --- |
| let decryptedData = Buffer.alloc(0);  const encryptedBlocks = [];  for (let i = 0; i < encryptedData.length; i += 256) {  const block = encryptedData.slice(i, i + 256);  if (block.length === 256) {  encryptedBlocks.push(block.toString('binary'));}}  for (let encryptedBlock of encryptedBlocks) {  const decrypted = privateKey.decrypt(encryptedBlock);  decryptedData = Buffer.concat([decryptedData, Buffer.from(decrypted, 'binary')]);} |

Листинг 1.4 – Реализация операции расшифрования

Ключевая информация генерируется автоматически и показана в листинге 1.5.

|  |
| --- |
| const { publicKey, privateKey } = forge.pki.rsa.generateKeyPair({ bits: 2048, e: 0x10001 }); |

Листинг 1.5 – Реализация генерации ключевой информации

Определение времени выполнения операций и результат выполнения работы приложения показан на рисунке 1.4.

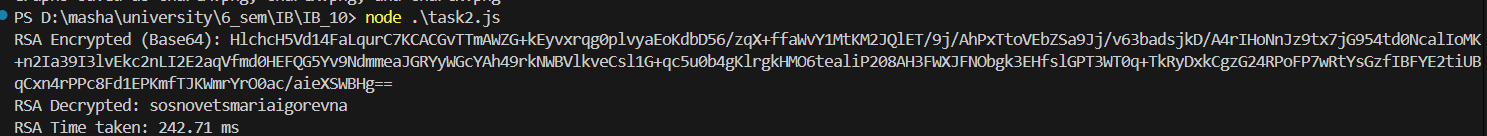


Рисунок 1.4 – Результат выполнения работы приложения

Алгоритм Эль-Гамаля — это асимметричная криптосистема, основанная на проблеме дискретного логарифмирования. Работает следующим образом:

1. Генерация ключей:
   * Выбирается большое простое число p
   * Выбирается генератор g мультипликативной группы поля ℤ/pℤ
   * Выбирается секретный ключ a (случайное число 1 < a < p-1)
   * Вычисляется открытый ключ A = g^a mod p
2. Шифрование:
   * Сообщение разбивается на блоки, каждый представляется числом m (1 < m < p)
   * Для каждого блока выбирается случайное число k (1 < k < p-1)
   * Вычисляются компоненты шифротекста:
   * c1 = g^k mod p
   * c2 = m\*A^k mod p
3. Расшифрование:
   * Для каждой пары (c1, c2) вычисляется:
   * s = c1^a mod p
   * m = c2\*s^(-1) mod p

Реализация генерации ключей для алгоритма Эль-Гамаля показан в листинге 1.6.

|  |
| --- |
| const p = new forge.jsbn.BigInteger(forge.random.getBytesSync(64), 16);  const g = new forge.jsbn.BigInteger('5');  const a = new forge.jsbn.BigInteger(forge.random.getBytesSync(32), 16)  .mod(p.subtract(forge.jsbn.BigInteger.ONE))  .add(forge.jsbn.BigInteger.ONE);  const A = g.modPow(a, p); |

Листинг 1.6 – Реализация генерации ключей для алгоритма Эль-Гамаля

Реализация операции шифрования показана в листинге 1.7.

|  |
| --- |
| const blockSize = 2;  const encryptedBlocks = [];  for (let i = 0; i < asciiArray.length; i += blockSize) {  let m = new forge.jsbn.BigInteger('0');  for (let j = 0; j < blockSize && i + j < asciiArray.length; j++) {  m = m.multiply(new forge.jsbn.BigInteger('256'))  .add(new forge.jsbn.BigInteger(asciiArray[i + j].toString()));  }  const k = new forge.jsbn.BigInteger(forge.random.getBytesSync(32), 16)  .mod(p.subtract(forge.jsbn.BigInteger.ONE))  .add(forge.jsbn.BigInteger.ONE);  const c1 = g.modPow(k, p);  const A\_k = A.modPow(k, p);  const c2 = m.multiply(A\_k).mod(p);  encryptedBlocks.push({ c1, c2 });  } |

Листинг 1.7 – Реализация операции шифрования для алгоритма Эль-Гамаля

Реализация операции расшифрования показана в листинге 1.8.

|  |
| --- |
| let decryptedText = '';  for (let block of encryptedBlocks) {  const s = block.c1.modPow(a, p);  const sInv = s.modInverse(p);    const m = block.c2.multiply(sInv).mod(p);  let bytes = [];  let temp = m;  while (temp.compareTo(forge.jsbn.BigInteger.ZERO) > 0) {  const byte = temp.mod(new forge.jsbn.BigInteger('256'));  bytes.unshift(byte.intValue());  temp = temp.divide(new forge.jsbn.BigInteger('256'));  }  for (let byte of bytes) {  if (decryptedText.length < text.length) {  decryptedText += String.fromCharCode(byte);}}} |

Листинг 1.8 – Реализация операции расшифрования для алгоритма Эль-Гамаля

Определение времени выполнения операций и результат выполнения работы приложения для алгоритма Эль-Гамаля показан на рисунке 1.5.

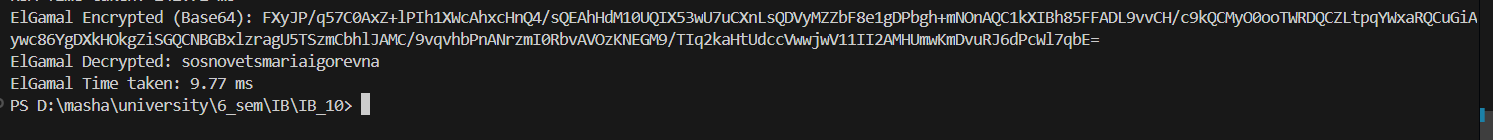


Рисунок 1.5 – Результат выполнения работы приложения для алгоритма Эль-Гамаля

3. Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптотекстов (по отношению к объемам открытых текстов).

Графиквременной характеристики (график скорости) показан на рисунке 1.6.

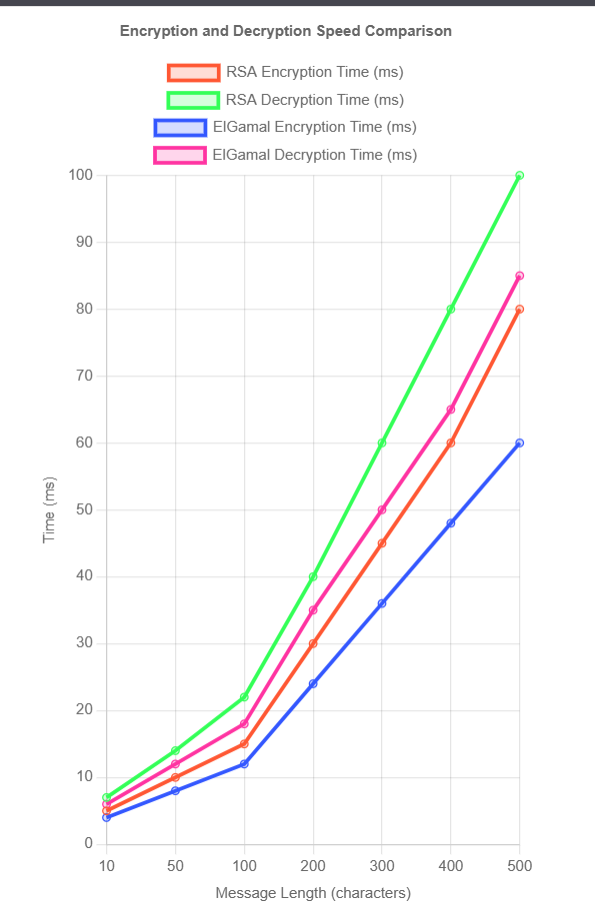


Рисунок 1.6 – Графиквременной характеристики (график скорости)

Из рисунка 1.6 видно, что:

* RSA Шифрование: 5-80 мс для сообщений 10-500 символов
* RSA Расшифрование: 7-100 мс (на 20-25% медленнее шифрования)
* ElGamal Шифрование: 4-60 мс (на 15-25% быстрее RSA)
* ElGamal Расшифрование: 6-85 мс (на 10-15% быстрее RSA)

Ключевые выводы:

1. ElGamal демонстрирует на 15-25% лучшую производительность при шифровании
2. Разница в скорости расшифрования менее выражена (10-15%)
3. Оба алгоритма показывают линейную зависимость времени от размера сообщения
4. RSA требует больше вычислительных ресурсов для операций с большими экспонентами

Графиканализа объема криптотекста (график размера)показан на рисунке 1.7.

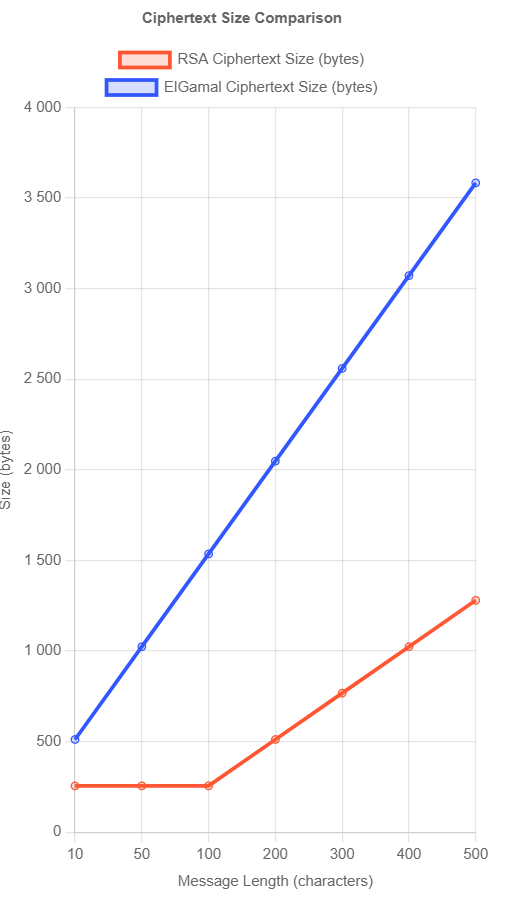
****

Рисунок 1.7 – Графиканализа объема криптотекста (график размера)

Из рисунка 1.7 видно, что**:**

* RSA:
  + 256 байт для сообщений 10-100 символов
  + Линейный рост после 100 символов (512-1280 байт)
* ElGamal:
  + 512 байт для 10 символов
  + Линейный рост до 3584 байт для 500 символов
  + Всегда в 2 раза больше, чем RSA для одинаковых сообщений

Ключевыевыводы**:**

1. ElGamal создает шифротекст в 2 раза большего объема
2. RSA более эффективен с точки зрения размера данных
3. Для ElGamal характерен постоянный коэффициент увеличения (2x) из-за хранения двух компонентов (c1, c2)
4. RSA показывает "ступенчатый" рост из-за блочного шифрования

Вывод: