**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Криптографические методы обеспечения информационной безопасности»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

«Ассиметричные криптосистемы»

**Выполнил:**

Молитвин Илья Алексеевич, студент группы N3Изображение выглядит как текст, соединитель

Контент, сгенерированный ИИ, может содержать ошибки.345

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Таранов Сергей Владимирович

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Оглавление**

[Введение 3](#_Toc162732672)

[Ход работы 4](#_Toc162732673)

[1. Анализ эмуляции алгоритма RSA 4](#_Toc162732674)

[2. Анализ примитивных атак на RSA 5](#_Toc162732675)

[3. Программная реализация алгоритма RSA 7](#_Toc162732676)

[Вывод 10](#_Toc162732677)

# Введение

Цель работы – изучить основные принципы работы асимметричных криптосистем на примере алгоритма RSA.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать эмуляцию алгоритма RSA и примитивных атак на шифр, используя Cryptool 2. Выделить основные необходимые настройки шифра и требуемые ограничения на параметры.

2. Программно реализовать и модифицировать любую асимметричную криптосистему. В случае отсутствия опыта программирования подойдет реализация алгоритма на псевдокоде или в виде блок схем, включающих основные этапы алгоритма с отображением формул и основных математических действий. Атаки и модификации, приведенные ниже, указаны для RSA. Если атака или модификация не применима для реализуемого алгоритма разрешается найти любую альтернативу (атаки, применимой к алгоритму; модификации для ускорения алгоритма и дополнительной защиты).

3. Для созданной реализации криптосистемы предлагается провести примитивный криптоанализ на устойчивость к следующим атакам, а также сделать минимальные модификации по оптимизации (ускорению процессов шифрования, дешифрования, процесса генерации ключей).

# Ход работы

## Анализ эмуляции алгоритма RSA

Для выполнения эмуляции нам необходим исходный текст:

“Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum a sagittis lectus. In eget leo vel odio molestie ullamcorper. Donec sit amet justo quis nunc varius rhoncus. Morbi et est commodo, sollicitudin odio ut, pretium tellus. Duis rutrum eu mauris nec egestas. Cras maximus mi sem, ac varius elit iaculis vel.”

Далее, чтобы зашифровать сообщение и расшифровать его впоследствии, нам необходимо сгенерировать ключи. Для генерации необходимо взять два простых числа – p и q, открытую экспоненту e, и взаимно простое со значением функции Эйлера d – мультипликативно обратное e.

Подставим эти значения в настройки шаблона в Cryptool 2.

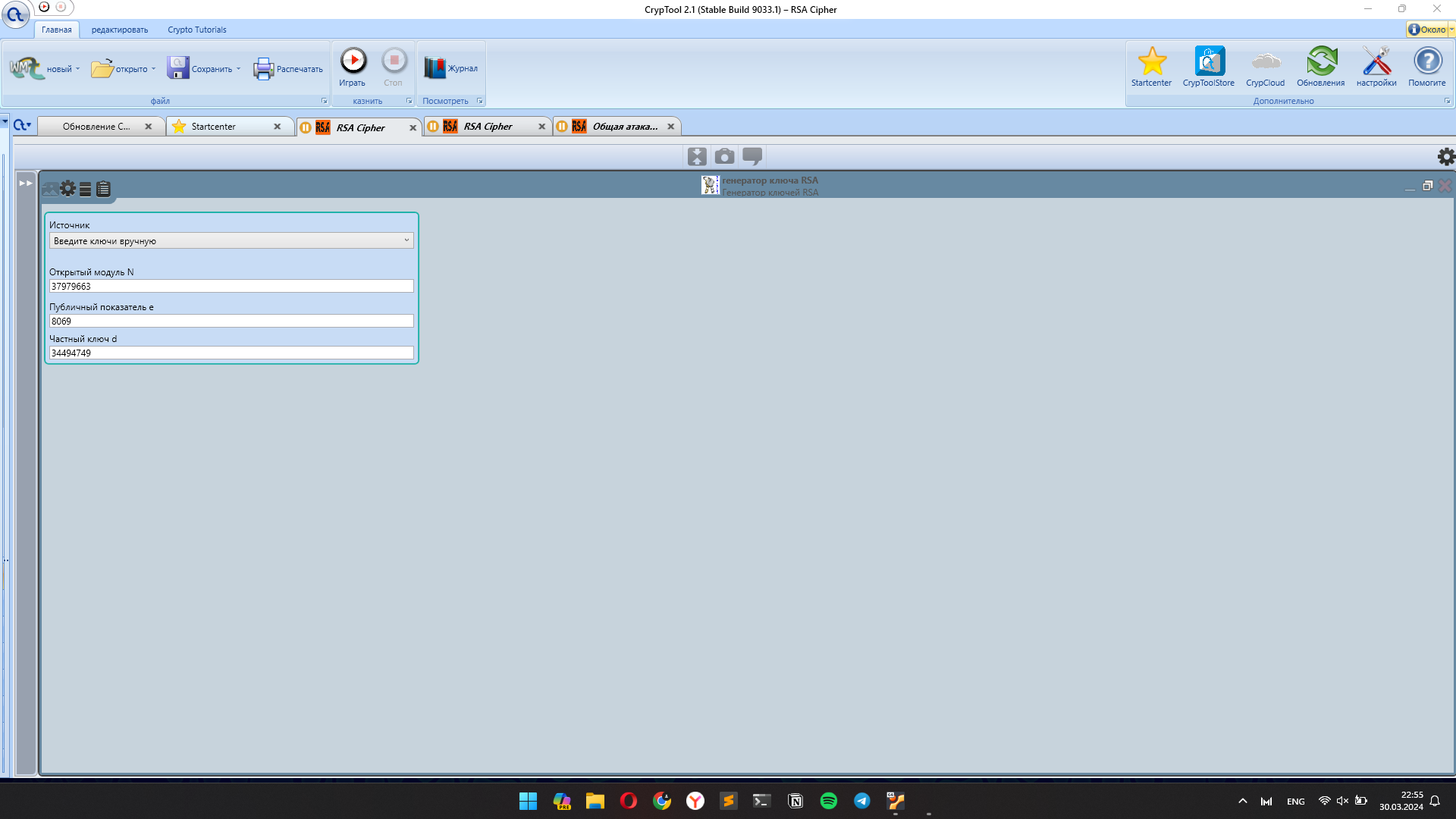


Рисунок 1 - Данные для генерации ключей

Вставим текст и получим зашифрованное сообщение.

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, веб-страница

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 - Выполнения шифрования RSA в Cryptool

## Анализ примитивных атак на RSA

Соберем следующий набор для подсчета N2 и шифрования.

Зашифруем сообщение с полученными значениями.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Выполнения шифрования RSA в Cryptool

Откроем в Cryptool шаблон RSA Common Factor Attack и выполним атаку, основанную на общем делителе для модуля шифрования.

Суть такой атаки в том, что злоумышленнику известны два числа N1 и N2, каждое из которых - произведение параметров (простых чисел). Один из этих параметров (в нашем случае – q) повторяется.

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, веб-страница

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Атака на RSA на основе ОД в Cryptool

Так, вычисляется НОД чисел N1 и N2 (тот самый повторяющийся параметр) и затем путем деления оставшиеся параметры. Далее подбирается значение экспоненты е.

Попробуем расшифровать шифротекст, полученный в начале этого раздела.

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5 – Дешифрование с помощью вычисленных значений параметров

Как можно увидеть, все получилось, сообщение расшифровано.

## Программная реализация алгоритма RSA

Программная реализация выполнена на языке программирования Python для упрощения различных математических функций с использованием библиотек, а также для простой реализации клиент-серверного приложения.

Как модификации основного алгоритма было использовано:

* минимальный размер шифрующих экспонент равняется 2^(256) для защиты от атак на малые экспоненты;
* быстрое возведение в степень с использованием бинарного возведения в степень по модулю.;
* Китайская теорема об остатках для ускоренной расшифровки сообщения.

Листинг 1 – Алгоритм RSA

|  |
| --- |
| import random  def is\_prime(num):  if num < 2:  return False  for i in range(2, int(num \*\* 0.5) + 1):  if num % i == 0:  return False  return True  def bin\_power(a, pwr, N):  if pwr % 2 == 0:  return bin\_power(a \* a % N, pwr // 2, N) % N  if pwr == 1:  return a % N  return a \* bin\_power(a, pwr - 1, N) % N  def gcd(a, b):  if a < b:  return gcd(b, a)  if a % b == 0:  return b  return gcd(b, a % b)  def mod\_inverse(a, m):  m0 = m  y = 0  x = 1  if m == 1:  return 0  while a > 1:  q = a // m  t = m  m = a % m  a = t  t = y  y = x - q \* y  x = t  if x < 0:  x = x + m0  return x  def gen\_e(phiN):  res = phiN - 1  while gcd(res, phiN) != 1:  res -= 2  return res  def gen\_d(e, phiN):  return mod\_inverse(e, phiN)  p = random.randint(100, 1000)  while not is\_prime(p):  p = random.randint(100, 1000)  q = random.randint(100, 1000)  while not is\_prime(q):  q = random.randint(100, 1000)  def gen\_RSA\_values(p, q):  N = p \* q  phiN = (p - 1) \* (q - 1)  e = gen\_e(phiN)  d = gen\_d(e, phiN)  return e, d, N  e, d, N = gen\_RSA\_values(p, q)  def E(m, e, N):  return bin\_power(m, e, N)  def D(c, d, N):  return bin\_power(c, d, N)  message = 42  encrypted\_message = E(message, e, N)  decrypted\_message = D(encrypted\_message, d, N)  print("RSA values: e =", e, "d =", d, "N =", N)  print("Original message: ", message)  print("Encrypted message: ", encrypted\_message)  print("Decrypted message: ", decrypted\_message) |

Листинг 2 – Быстрое возведение в степень

|  |
| --- |
| def bin\_power(a, pwr, N):  if pwr % 2 == 0:  return bin\_power(a \* a % N, pwr // 2, N) % N  if pwr == 1:  return a % N  return a \* bin\_power(a, pwr - 1, N) % N |

Листинг 3 – Китайская теорема об остатках для дешифрования

|  |
| --- |
| def CRT(a: list, m: list):  res = 0  M = 1  for i in m:  M \*= i  for i in range(len(a)):  Mi = M // m[i]  yi = rev(Mi, m[i])  res = (res + a[i] \* yi \* Mi) % M  return res  def CRT\_decipher(c, p, q, d):  mp = bin\_power(c, d % (p - 1), p)  mq = bin\_power(c, d % (q - 1), q)  m = CRT([mp, mq], [p, q])  print("mp mq:", mp, mq)  return m |

Листинг 4 – Генерация открытой экспоненты

|  |
| --- |
| def gen\_e\_mod(phiN):  res = randint(2 \*\* 256, phiN - 1)  while gcd(res, phiN) != 1:  res = randint(2 \*\* 256, phiN - 1)  return res |

# Вывод

В ходе выполнения лабораторной работы были изучены основы работы алгоритма RSA, изучена атака на данный алгоритм. Также была написана программная реализация данного алгоритма и модифицированная ее версия.