Пшенко Артем ФИТ 3-4

Отчет по лабораторной работе № 10

Исследование ассиметричных шифров (алгоритмы RSA и Эль-Гамаля)

Информационная безопасность

Вариант 9

Цель данной лабораторной работы состоит в изучении работы алгоритмов асимметричного шифрования RSA и Эль-Гамаля, их применении для шифрования и дешифрования сообщений.

Алгоритм RSA:

1. Генерация простых чисел: генерируются два больших простых числа p и q.
2. Расчет значений: вычисляются значения n = p \* q и функция Эйлера φ(n) = (p-1) \* (q-1).
3. Генерация ключей: выбирается целое число e, взаимно простое с φ(n), и вычисляется значение d, такое что (d \* e) ≡ 1 (mod φ(n)).
4. Шифрование/Дешифрование: сообщение шифруется и дешифруется с использованием открытого и закрытого ключей соответственно.

Исходный код алгоритма RC4 приведен в листинге 1.

Листинг 1. Исходный код алгоритма RSA

|  |
| --- |
| from sympy import randprime, isprime, gcd, mod\_inverse  import time  def generate\_prime(digits):  lower\_bound = 10 \*\* (digits - 1)  upper\_bound = 10 \*\* digits - 1  return randprime(lower\_bound, upper\_bound)  def euler\_value(p, q):  fi = (p-1)\*(q-1)  return fi  def find\_e(fi):  for e in range(2, fi):  if isprime(e) and gcd(e, fi) == 1:  return e  return None  def generate\_public\_key(p, q, fi):  n = p \* q  e = find\_e(fi)  return n, e  def generate\_private\_key(e, fi, n):  d = mod\_inverse(e, fi) # d = e^(-1) mod fi  return n, d  def timer\_decorator(func):  def wrapper(\*args, \*\*kwargs):  start\_time = time.time()  result = func(\*args, \*\*kwargs)  end\_time = time.time()  print(f"Время выполнения функции {func.\_\_name\_\_}: {end\_time - start\_time} секунд")  return result  return wrapper  @timer\_decorator  def encrypt(message, e, n):  encrypted\_message = [pow(ord(char), e, n) for char in message]  return encrypted\_message  @timer\_decorator  def decrypt(encrypted\_message, d, n):  decrypted\_message = ''.join([chr(pow(el, d, n)) for el in encrypted\_message])  return decrypted\_message    p = generate\_prime(100)  q = generate\_prime(100)  fi = euler\_value(p, q)  n, e = generate\_public\_key(p, q, fi)  n, d = generate\_private\_key(e, fi, n)  message = "Pshenko Artyom Fyodorovich"  print("Original Message:", message)  encrypted\_message = encrypt(message, e, n)  print("Encrypted Message:", encrypted\_message)  decrypted\_message = decrypt(encrypted\_message, d, n)  print("Decrypted Message:", decrypted\_message)  print(generate\_private\_key(79, 3220, 3337)) |

Результат работы алгоритма RSA приведен на рисунке 1.

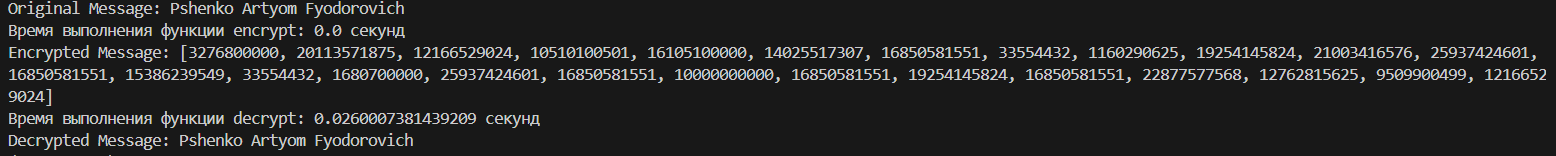


Рисунок 1. Результат работы программы

Алгоритм Эль-Гамаля:

1. Генерация простого числа: генерируется большое простое число p.
2. Выбор примитивного корня: выбирается примитивный корень g по модулю p.
3. Генерация закрытого и открытого ключей: выбирается случайное число x (закрытый ключ) и вычисляется y = g^x mod p (открытый ключ).
4. Шифрование/Дешифрование: сообщение шифруется с использованием открытого ключа, а дешифруется с использованием закрытого ключа.

Листинг 2. Исходный код алгоритма Эль-Гамаля

|  |
| --- |
| from sympy.ntheory.residue\_ntheory import primitive\_root  from sympy import randprime, mod\_inverse  from random import randint  import time  def generate\_prime(digits):  lower\_bound = 10 \*\* (digits - 1)  upper\_bound = 10 \*\* digits - 1  return randprime(lower\_bound, upper\_bound)  def timer\_decorator(func):  def wrapper(\*args, \*\*kwargs):  start\_time = time.time()  result = func(\*args, \*\*kwargs)  end\_time = time.time()  print(f"Время выполнения функции {func.\_\_name\_\_}: {end\_time - start\_time} секунд")  return result  return wrapper  def fast\_modular\_exponentiation(base, exponent, modulus):  result = 1  base = base % modulus  while exponent > 0:  if exponent % 2 == 1:  result = (result \* base) % modulus  exponent = exponent >> 1  base = (base \* base) % modulus  return result  @timer\_decorator  def encrypt(message, p, g, y):  encrypted\_message = []  for char in message:  k = randint(2, p - 2) # границы входят, поэтому если 1 < k < p-1, в функцию надо передавать p-2  a = fast\_modular\_exponentiation(g, k, p) # g^k mod p  b = (fast\_modular\_exponentiation(y, k, p) \* ord(char)) % p  encrypted\_message.append((a, b))  return encrypted\_message  @timer\_decorator  def decrypt(enc\_message, p, x):  decrypted\_message = ""  for pair in enc\_message:  a, b = pair  inverse\_a = pow(a, x, p) # (a^x) mod p  inverse\_a = mod\_inverse(inverse\_a, p) # (a^x)^(-1) mod p  m = (b \* inverse\_a) % p  decrypted\_message += chr(m)  return decrypted\_message  p = generate\_prime(100)  g = primitive\_root(p)  x = randint(2, p - 1) # границы входят, поэтому если 1 < x < p, в функцию надо передавать p-1  y = fast\_modular\_exponentiation(g, x, p)  message = "Pshenko Artyom Fyodorovich"  enc\_message = encrypt(message, p, g, y)  print(enc\_message)  print(decrypt(enc\_message, p, x)) |

Результат работы алгоритма BBS представляет из себя массив кортежей вида [(a1, b1), … , (an, bn)], где n – количество символов исходного сообщения.

Таким образом, в ходе лабораторной работы были продемонстрированы работы алгоритмов ассиметричного шифрования RSA и Эль-Гамаля.