Исследование ассиметричных шифров RSA и Эль-Гамаля

Выполнил: Макаров Алексей Игоревич, 3 курс 4 группа 2 подгруппа

2024

**Теоритические сведения**

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.

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.



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).

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:



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

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



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

Блок шифртекста (ci) состоит из двух чисел – аi и bi:

A close-up of a math equation

Description automatically generated

Здесь стал очевидным упомянутый недостаток алгоритма шифрования Эль-Гамаля: удвоение (реально – примерно в 1,5 раза) длины зашифрованного текста по сравнению с начальным текстом.

Расшифрование ci. Выполняется по следующей формуле:



При примерно одинаковой размерности ключей рассмотренные алгоритмы обеспечивают примерно одинаковый уровень криптостойкости.

**Ход работы**

Вариант 8

В данной работе было необходимо реализовывать авторское оконное приложение в соответствии с целью лабораторной работы.

Код программы представлен в листинге 1.

|  |
| --- |
| *# from cryptography.hazmat.primitives.asymmetric import elgamal* import time import base64 from Crypto.PublicKey import RSA from Crypto.Cipher import PKCS1\_OAEP from Crypto.Random import get\_random\_bytes from sympy.ntheory.generate import randprime from Crypto.Util import number from sympy import isprime, mod\_inverse import random *# RSA шифрование и расшифрование* def rsa\_encrypt\_decrypt(document):  *# Генерация ключей  key* = RSA.generate(2048)  *public\_key* = *key*.publickey()  *cipher\_rsa* = PKCS1\_OAEP.new(*public\_key*)   *# Зашифрование  start\_time* = time.time()  *encrypted\_document* = *cipher\_rsa*.encrypt(document.encode('utf-8'))  *encrypted\_document\_b64* = base64.b64encode(*encrypted\_document*).decode('utf-8')  *encrypt\_time* = time.time() - *start\_time   # Расшифрование  start\_time* = time.time()  *decrypted\_document* = PKCS1\_OAEP.new(*key*).decrypt(base64.b64decode(*encrypted\_document\_b64*.encode('utf-8')))  *decrypt\_time* = time.time() - *start\_time* return *encrypt\_time*, *decrypt\_time*, *decrypted\_document*.decode('utf-8')  *# Эль-Гамаль шифрование и расшифрование* def elgamal\_encrypt\_decrypt(document):  *# Генерация параметров* def generate\_large\_prime(bits):  while True:  *candidate* = number.getPrime(bits)  if isprime(*candidate*):  return *candidate    bits* = 2048  *p* = generate\_large\_prime(*bits*)  *g* = 2  *x* = random.randint(1, *p*-2)  *y* = pow(*g*, *x*, *p*)   *# Генерация открытого и закрытого ключа* public\_key = (*p*, *g*, *y*)  *private\_key* = *x   # Зашифрование  k* = random.randint(1, *p*-2)  *a* = pow(*g*, *k*, *p*)  *b* = (pow(*y*, *k*, *p*) \* int.from\_bytes(document.encode('utf-8'), 'big')) % *p   start\_time* = time.time()  *encrypted\_document* = (*a*, *b*)  encrypted\_document\_b64 = (base64.b64encode(*a*.to\_bytes((*a*.bit\_length() + 7) // 8, 'big')).decode('utf-8'),  base64.b64encode(*b*.to\_bytes((*b*.bit\_length() + 7) // 8, 'big')).decode('utf-8'))  *encrypt\_time* = time.time() - *start\_time   # Расшифрование  start\_time* = time.time()  *decrypted\_int* = (*encrypted\_document*[1] \* mod\_inverse(pow(*encrypted\_document*[0], *private\_key*, *p*), *p*)) % *p  decrypted\_document* = *decrypted\_int*.to\_bytes((*decrypted\_int*.bit\_length() + 7) // 8, 'big').decode('utf-8')  *decrypt\_time* = time.time() - *start\_time* return *encrypt\_time*, *decrypt\_time*, *decrypted\_document  # Тестирование функций* document = "Hello, this is a test document for encryption and decryption!"  *# RSA* rsa\_encrypt\_time, rsa\_decrypt\_time, rsa\_decrypted\_document = rsa\_encrypt\_decrypt(document) print(f"RSA Encryption Time: {rsa\_encrypt\_time:.6f} s, Decryption Time: {rsa\_decrypt\_time:.6f} s") print(f"RSA Decrypted Document: {rsa\_decrypted\_document}")  *# ElGamal* elgamal\_encrypt\_time, elgamal\_decrypt\_time, elgamal\_decrypted\_document = elgamal\_encrypt\_decrypt(document) print(f"ElGamal Encryption Time: {elgamal\_encrypt\_time:.6f} s, Decryption Time: {elgamal\_decrypt\_time:.6f} s") print(f"ElGamal Decrypted Document: {elgamal\_decrypted\_document}") |

Листинг 1 – Программная реализация приложения

Результат работы программы представлен в листинге 2:

|  |
| --- |
| --Эль-Гамаля--  Генерация ключевой информации  p: 173  g: 129  x: 65  y: 74  Зашифрованное сообщение: [(160, 21), (125, 130), (32, 123), (134, 81), (153, 16), (134, 109), (27, 119), (148, 115), (92, 8), (167, 98), (47, 84), (96, 32), (172, 74), (159, 7), (146, 164), (123, 33), (128, 92), (136, 155), (95, 117), (62, 33), (13, 101)]  Расшифрованное сообщение: иванов иван иванович  Время зашифрования: 0.00014209747314453125  Время расшифрования: 0.00013518333435058594  Длина шифротекста: 104  --RSA--  p: 157  q: 199  n: 31243  e: 19529  d: 13841  [122, 104, 117, 107, 32, 97, 110, 110, 97, 32, 109, 105, 99, 104, 97, 105, 108, 111, 118, 110, 97]  Зашифрованное сообщение:  28756300082659034761615137463053430534137461615204571898025513000813746189802506423868215363053413746  Расшифрованное сообщение: иванов иван иванович  Время зашифрования: 5.412101745605469e-05  Время расшифрования: 9.894371032714844e-05  Длина шифротекста: 101 |

Листинг 2 – Результат работы приложения

**Вывод**: таким образом, были закреплены теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров RSA и Эль-Гамаля.