БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра интеллектуальных информационных технологий

ОТЧЁТ  
по лабораторной работе №5  
по дисциплине

«СРЕДСТВА И МЕТОДЫ ЗАЩИТЫ ИНФОРАМЦИИ В ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ»

Выполнил студент гр. 221703

Вечорко Д. Н.

Проверил

Крищенович В. А.

Минск 2024

Задание:

Разработать программное обеспечение, реализующее функции генерации секретного и открытого ключей, шифрования и цифровой подписи для алгоритма RSA. Обмен входными и выходными данными должен осуществляться через файлы:

1. открытого ключа;
2. секретного ключа;
3. исходного сообщения;
4. зашифрованного сообщения. Для повышения скорости шифрования использовать метод последовательного возведения в квадрат и умножения. Выполнить тестирование разработанного программного обеспечения на 10 наборах тестовых данных. Длина чисел p и q должна быть не менее 1024 бит.

Функция is\_prime() определяет, является ли число простым:

KEYSIZE = 1024

import random  
  
def is\_prime(n, k=5):  
 if n <= 3:  
 return n == 2 or n == 3  
 if n % 2 == 0:  
 return False  
  
 r, s = 0, n - 1  
 while s % 2 == 0:  
 r += 1  
 s //= 2  
  
 for \_ in range(k):  
 a = random.randint(2, n - 2)  
 x = pow(a, s, n)  
 if x == 1 or x == n - 1:  
 continue  
 for \_ in range(r - 1):  
 x = pow(x, 2, n)  
 if x == n - 1:  
 break  
 else:  
 return False  
  
 return True

Следующая функция генерирует случайные простые числа:

def rabin\_miller(num):  
 s = num - 1  
 t = 0  
   
 while s % 2 == 0:  
 s = s // 2  
 t += 1  
 for \_ in range(5):  
 a = random.randrange(2, num - 1)  
 v = pow(a, s, num)  
 if v != 1:  
 i = 0  
 while v != (num - 1):  
 if i == t - 1:  
 return False  
 else:  
 i = i + 1  
 v = (v \*\* 2) % num  
 return True  
  
def is\_prime(num):  
 if num < 2:  
 return False  
 lowPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,   
 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,   
 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,   
 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,317, 331, 337, 347, 349,   
 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449,   
 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569,   
 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661,   
 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787,   
 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,   
 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]  
   
 if num in lowPrimes:  
 return True  
 for prime in lowPrimes:  
 if (num % prime == 0):  
 return False  
 return rabin\_miller(num)  
  
def gcd(a, b):  
 while a != 0:  
 a, b = b % a, a  
 return b  
  
def generate\_prime(keysize = 1024):  
 while True:  
 num = random.randrange(2\*\*(keysize-1), 2\*\*(keysize))  
 if is\_prime(num):  
 return num  
   
generate\_prime()

176190947309491472030899278944635588134370583828219396494318714357083411156793773267817021859926021069069208050412164112132760694279424889453372964472730534735086180046976855649430826020578985743205342724312276897327627076504870433519068085113178908826834851694970334633650865290501434758562923167941876893687

Теперь, сгенерируем два случайных простых числа p и q, вычислим их произведение, а также функцию Эйлера. В качестве открытой экспоненты используем одно из простых чисел Ферма (17, 257, 65537).

p, q = generate\_prime(KEYSIZE), generate\_prime(KEYSIZE)  
  
n = p \* q  
f = (p - 1) \* (q - 1)  
  
while True:  
 e = random.randrange(2 \*\* (KEYSIZE - 1), 2 \*\* KEYSIZE)  
 if gcd(e, f) == 1:  
 break

Метод find\_mod\_inverse() вычисляет число d, мультипликативно обратное числу e по модулю n (т.е. удовлетворяет сравнению d\*e = 1mod f(n))

def find\_mod\_inverse(a, m):  
 if gcd(a, m) != 1:  
 return None  
 u1, u2, u3 = 1, 0, a  
 v1, v2, v3 = 0, 1, m  
   
 while v3 != 0:  
 q = u3 // v3  
 v1, v2, v3, u1, u2, u3 = (u1 - q \* v1), (u2 - q \* v2), (u3 - q \* v3), v1, v2, v3  
 return u1 % m  
  
d = find\_mod\_inverse(e, f)

Таким образом, теперь мы можем составить закрытый и открытый ключи:

public\_key = (e, n)  
private\_key = (d, n)  
  
print(f"Закрытый ключ: {private\_key}")  
print(f"Открытый ключ: {public\_key}")

Закрытый ключ: (11577094824322612921111344551573701087719483321722358616305251281326143454738196547221598701539991916151845236124399311249930272280275894296175638868249405198132031755819381168875122472309266950444849478108072893194961066062626029134173154736199648943873824128494652943200471536148535305864136456746065759005814839370965585065509112299110619288103530833509824517247163458273112764127042088939651745465268891220823950863934935453850444696981225323026594907158634470828877060261373924455089121736849580217947674934977509823229576354875380062358131956184084806104848220698291380427500318839798473129064393392638832178071, 13483161572597451491923861135164607353566976638258425358012224934625920342257587778947913114978221456164187928258975303439523709605050557992386796713855387195811255227084136312008282433674111117278326682486427192836375437017185063332055095984070833891464731619030558903633076438928304800016629929846511759762036013097838245398245046853230192586631698863867822601387979466037916329684964451508505193031559529580028182948374844464326619318383620085765912507211535155831760782051034177919606919314720475096095389615189566928326479703788206903673345052084366663146306077808935803849266609619230396919357995421671676017379)  
Открытый ключ: (99796263170192724895463835797888409593432467314093285381716264359627739111466112013220047413599475438273666326371789848446211651091588052224988567337245108240512256836299547443679261168254904841045973332459343236623165959526479107169008108094127236218570129819918119114263311365374492037561662187756479568551, 13483161572597451491923861135164607353566976638258425358012224934625920342257587778947913114978221456164187928258975303439523709605050557992386796713855387195811255227084136312008282433674111117278326682486427192836375437017185063332055095984070833891464731619030558903633076438928304800016629929846511759762036013097838245398245046853230192586631698863867822601387979466037916329684964451508505193031559529580028182948374844464326619318383620085765912507211535155831760782051034177919606919314720475096095389615189566928326479703788206903673345052084366663146306077808935803849266609619230396919357995421671676017379)

В качестве исходного сообщения будет выступать целое число в интервале от 0 до n - 1.

Для зашифрования используем открытый ключ {e, n}, вычилим крипограмму:

import numpy as np  
  
m = random.randint(1, 100)  
  
def fast\_mod\_exp(b, exp, m):  
 res = 1  
 while exp > 1:  
 if exp & 1:  
 res = (res \* b) % m  
 b = b \*\* 2 % m  
 exp >>= 1  
 return (b \* res) % m  
  
print(f"Исходное сообщение: {m}")  
  
c = fast\_mod\_exp(m, e, n)  
  
print(f"Криптограмма: {c}")  
  
dc = fast\_mod\_exp(c, d, n)  
  
print(f"Расшифрованная криптограмма: {dc}")

Исходное сообщение: 58  
Криптограмма: 13389152471902689712676247204266682055667499498307219123869990838634048783875397624114759055014119026206899944094616978576493367675266928015798192554305248016969831554162965006719359933891764606841684268549821289742946976293518758742558688662558031750473854659869665668953467296897653897979707208838614316298641304909212381611607043008489883762190021523515302986671459564538303424416464011263790650577122589594151057809705994075177127194341316145957633363455573103496256507343301700686240198562382304128506174583069669619232725976032789859774402838991529985775749867908491632401758945012827246151557497817811370834440  
Расшифрованная криптограмма: 58

Цифровая подпись:

Для создания цифровой подписи s с помощью секретного ключа вычисляют s = m^d mod n. Затем формируют пару {m, s} и отправляют получателю.

s = fast\_mod\_exp(m, d, n)  
  
signature = (m, s)  
  
print(f"{signature=}")

signature=(58, 11179643636803928225812857179348921760818123252816613767276595593401783239159257778354649165775670573648211009897786292604493935182278764454875330775183759049559066339945474607963909361527707583273156637452518795816417240796915888236940053886381649313848400247197699511012513716006132201280847676436518052501217417037218466131004541915017493015094284505957486734004271031614212936546910261120627411504034267646405306147320936305392221984463747512064689696750881158056369240340414084230011918006446930917823372484805293265215969645120850685886349570717422509898082215644215014247515946202907461433188114846675718033027)

Теперь проверим цифровую подпись: вычислим прообраз сообщения из подписи.

ms = fast\_mod\_exp(s, e, n)  
  
is\_valid = m == ms  
  
print(f"{is\_valid=}")

Вывод: Мы изучили принципы работы ассиметричного шифрования и электронной цифровой подписи, а также реализовали генерацию секретного и открытого ключей и шифрование для алгоритма RSA.