НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Лабораторная работа №1

По дисциплине «Криптографические методы защиты информации»

«Матричные шифры. Шифр Хилла»

Группа: А-14м-24

Студент: Гороховский И. А.

Преподаватель: Фролов А. Б.

Москва 2025

Задание 1.1

Построить шифр Хилла y=Mx при m=2, n=p, где p − простое число длины 50-k десятичных символов, k − Ваш номер в журнале группы.

Примечание: алфавиты открытого текста и шифртекста A=B=Zn.

Ключ есть обратимая матрица K размеров 2 x 2 над Zn.

(K обратима ⬄ НОД(n, det (M))=1).

(В нашем случае это эквивалентно тому, что определитель det K не равен 0, так n выбрано как простое число).

1. Получение числа p

Номер в журнале – 1. Следовательно, p должно быть длиной в 49.

Для нахождения такого большого числа, была написана функция generatePrime, комбинирующая несколько методов (решето Эратосфена, проверка малой теоремы Ферма) с использованием случайной генерации чисел:

def getSieve(n):  
 *"""  
 Реализация решета Эратосфена для нахождения всех простых чисел до n.  
 Возвращает список простых чисел.  
 """  
 # Создаем массив isprime, где isprime[i] = True, если i предположительно простое число* isprime = [True for \_ in range(n)]  
 prime = []  
 spf = [None for \_ in range(n)]  
  
 *# Числа 0 и 1 не являются простыми* isprime[0] = isprime[1] = False  
  
 *# Проходим по всем числам от 2 до n-1* for i in range(2, n):  
 if isprime[i]: *# Если i предположительно простое* prime.append(i) *# Добавляем его в список простых чисел* spf[i] = i *# Наименьший простой делитель числа i — это само число i  
  
 # Обновляем массив isprime, помечая составные числа как False* j = 0  
 while j < len(prime) and i \* prime[j] < n and prime[j] <= spf[i]:  
 isprime[i \* prime[j]] = False  
 spf[i \* prime[j]] = prime[j]  
 j += 1  
  
 return prime *# Возвращаем список простых чисел*def is\_probably\_prime(n, sieve):  
 *"""  
 Проверяет, является ли число n вероятно простым, используя список простых чисел из решета.  
 Возвращает False, если n делится на любое из простых чисел из решета.  
 """* for x in sieve:  
 if n % x == 0:  
 return False  
 return True  
  
  
def generatePrime(n: int):  
 *"""  
 Генерирует простое число длиной n десятичных цифр.  
 Использует алгоритм, основанный на решете Эратосфена и тестах на простоту.  
 """* up\_limit = 10 \*\* n *# Верхний предел для генерации числа (10^n)* lower\_limit = up\_limit // 10 *# Нижний предел для генерации числа (10^n)* primes = getSieve(1000)  
 s = primes[-1] *# Начинаем с наибольшего простого числа из решета  
  
 # Основной цикл для генерации простого числа* while s < lower\_limit:  
 lo = (lower\_limit - 1) // s + 1 *# Минимальное значение r* hi = (up\_limit - 1) // s *# Максимальное значение r* while True:  
 *# Генерируем случайное число r и вычисляем кандидата на простое число n* try:  
 r = random.randint(lo, hi) << 1 *# r — четное случайное число* except ValueError:  
 print((up\_limit - 1), s, hi)  
 r = random.randint(lo, hi + 1)  
 cand = s \* r + 1 *# Формула для нового кандидата на простое число  
  
 # Проверяем, является ли n вероятно простым* if not is\_probably\_prime(cand, primes):  
 continue  
  
 *# Проводим дополнительную проверку на простоту с помощью теста Ферма* while True:  
 a = random.randint(2, cand - 1)  
 if pow(a, cand - 1, cand) != 1: *# Проверяем малую теорему Ферма* break  
  
 *# Проверяем НОД для дополнительной уверенности* d = math.gcd((pow(a, r, cand) - 1) % cand, cand)  
 if d != cand:  
 if d == 1: *# Если НОД равен 1, n вероятно простое* s = cand  
 break  
  
 if s == cand:  
 break  
  
 if s > up\_limit:  
 return generatePrime(n)  
 return s

p = generatePrime(49)  
print("p =", p)

Вывод:

p = 8320079901665884513114633972905977299428323706107

1. Составление матрицы M

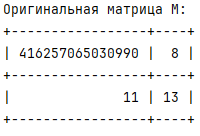
Для создания нужной матрицы были использованы методы получения случайных чисел. После её создания запускается проверка, что detM не равен 1, иначе матрица пересоздаётся, пока не выполнится это условие.

def get\_new\_matrix():  
 return [[random.randint(10 \*\* 10, 10 \*\* 15), random.randint(3, 30)],  
 [random.randint(3, 30), random.randint(3, 30)]]  
  
  
def determinant(matrix):  
 return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

def print\_matrix(matrix, name):  
 print(name)  
 s = [[str(e) for e in row] for row in matrix]  
 lens = [max(map(len, col)) for col in zip(\*s)]  
 fmt = '\t'.join('{{:{}}}'.format(x) for x in lens)  
 table = [fmt.format(\*row) for row in s]  
 print('\n'.join(table))  
 print()

matrix\_M = get\_new\_matrix()  
det\_M = determinant(matrix\_M)  
  
while det\_M == 1:  
 matrix\_M = get\_new\_matrix()  
 det\_M = determinant(matrix\_M)  
  
print\_matrix(matrix\_M, "Оригинальная матрица M:")

Вывод:



1. Построить обратную матрицу M-1