НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Лабораторная работа №2

По дисциплине «Криптографические методы защиты информации»

«Коды аутентификации и стратегии навязывания»

Группа: А-14м-24

Студент: Гороховский И. А.

Преподаватель: Фролов А. Б.

Москва 2025

Задание 2.1

Написать программу для вычисления вероятностей pподм(x’,a’;x,a), x’≠x, pим(x,a), и оптимальной стратегии подмены при равномерном распределении вероятностей открытого текста и задаваемом распределении вероятностей ключа для кода аутентификации из примера 2.1. и применить ее в случае равномерного распределения вероятностей ключей, а также r в случае знания ключей.

1. Входные данные

В программе используются три внешние библиотеки:

* Numpy – работа с массивами
* Fractions – работа с дробями
* Tabulate – вывод таблиц

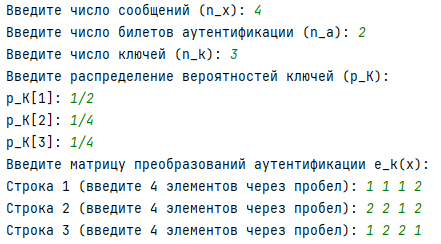
Прежде всего необходимо задать начальные значения: массив распределения вероятностей, массив билетов аутентификации, и их размеры. В коде ниже производится ввод всех необходимых данных.

import numpy as np  
from fractions import Fraction  
from tabulate import tabulate  
  
*# Ввод размерностей*n\_x = int(input("Введите число сообщений (n\_x): "))  
n\_a = int(input("Введите число билетов аутентификации (n\_a): "))  
n\_k = int(input("Введите число ключей (n\_k): "))  
  
*# Ввод распределения вероятностей ключей*print("Введите распределение вероятностей ключей (p\_K):")  
p\_K = []  
for i in range(n\_k):  
 while True:  
 try:  
 value = input(f"p\_K[{i + 1}]: ")  
 prob = Fraction(value)  
 p\_K.append(prob)  
 break  
 except ValueError:  
 print("Ошибка: введите дробное число в формате 'числитель/знаменатель' или целое число.")  
  
*# Проверка, что сумма вероятностей равна 1*if not sum(p\_K) == 1:  
 print("Ошибка: сумма вероятностей должна быть равна 1.")  
 exit()  
  
p\_K = np.array(p\_K)  
  
*# Ввод матрицы преобразований аутентификации e\_k(x)*print("Введите матрицу преобразований аутентификации e\_k(x):")  
e\_matrix = []  
for i in range(n\_k):  
 while True:  
 try:  
 row = input(f"Строка {i + 1} (введите {n\_x} элементов через пробел): ").split()  
 if len(row) != n\_x:  
 raise ValueError  
 row = [Fraction(x) for x in row]  
 e\_matrix.append(row)  
 break  
 except ValueError:  
 print(f"Ошибка: введите ровно {n\_x} элементов в строке.")  
  
e\_matrix = np.array(e\_matrix, dtype=object)  
  
*# Распределение вероятностей сообщений (равномерное)*p\_X = Fraction(1, n\_x)  
  
indices\_combinations = [(x+1, a+1) for x in range(n\_x) for a in range(n\_a)]

Также, для вывода некоторых таблиц, была написана отдельная функция:

def print\_fraction\_table(arr, splices=False):  
 *# Добавляем столбец с ID* ids = np.arange(1, len(arr) + 1).reshape(-1, 1) *# Создаем столбец ID* if splices:  
 ids = np.array([f"{x}" for x in indices\_combinations]).reshape(-1, 1)  
 table = np.hstack((ids, arr)) *# Объединяем ID и массив дробей  
  
 # Создаем заголовки* headers = ["x\\a"] + [f"{i + 1}" for i in range(len(arr[0]))]  
 if splices:  
 headers = ["(x, a)\\(x', a')"] + [f"{x}" for x in indices\_combinations]  
  
 *# Выводим таблицу* print(tabulate(table, headers=headers, tablefmt="pretty"))

Вывод (в качестве значений взяты значения из примера):

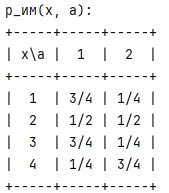


1. Построить массив p\_им(x,a)

В коде ниже – создание массива p\_им с помощью суммирования вероятностей p\_K:

p\_им = np.zeros((n\_x, n\_a), dtype=object)  
for x in range(n\_x):  
 for a in range(n\_a):  
 *# Суммируем вероятности ключей, для которых e\_k(x) = a* p\_им[x, a] = np.sum(p\_K[e\_matrix[:, x] == a + 1])  
  
print("p\_им(x, a):")  
print\_fraction\_table(p\_им)

Вывод получившейся таблицы с помощью собственной функции print\_fraction\_table:



1. Построить оптимальную стратегию имитации

Ниже представлен код для составления списка всех сообщений с максимальной вероятностью имитации:

opt\_imitation = []  
max\_p\_им = np.max(p\_им)  
for x in range(n\_x):  
 for a in range(n\_a):  
 if p\_им[x, a] == max\_p\_им:  
 opt\_imitation.append((x + 1, a + 1))  
  
print("\nОптимальная стратегия имитации:")   
print(f"Навязывание одного из сообщений {opt\_imitation}, где p\_им = {Fraction(max\_p\_им)}")

Вывод:

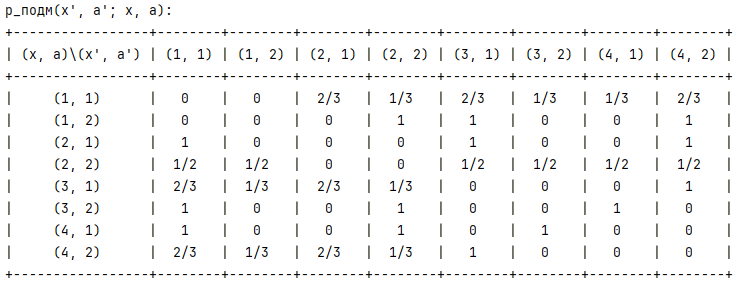


1. Построить массив p\_подм(x’,a’,x,a)

Ниже представлен код, где заполняется двухмерный массив, который затем выводится в виде таблицы, аналогичной таблице из примера. Массив составляется с помощью составления двух списков ключей, нахождения их пересечения, и затем высчитывания самой вероятности подмены:

p\_подм\_2d = np.zeros((n\_x \* n\_a, n\_x \* n\_a), dtype=object)  
  
*# Заполняем массив*for x in range(n\_x):  
 for a in range(n\_a):  
 index\_1 = x \* n\_a + a *# Первый индекс* for x\_prime in range(n\_x):  
 for a\_prime in range(n\_a):  
 index\_2 = x\_prime \* n\_a + a\_prime *# Второй индекс* if x\_prime == x:  
 p\_подм\_2d[index\_1, index\_2] = 0  
 continue  
 *# Ищем индексы строк, где в столбце x значение равно a* indices1 = np.where(e\_matrix[:, x\_prime] == a\_prime + 1)[0]  
 indices2 = np.where(e\_matrix[:, x] == a + 1)[0]  
  
 *# Находим пересечение* intersection = np.intersect1d(indices1, indices2)  
 result\_up = Fraction(np.sum(p\_K[intersection]))  
  
 *# Вероятность подмены* p\_подм\_2d[index\_1, index\_2] = result\_up / p\_им[x, a]  
  
print("\np\_подм(x', a'; x, a):")  
print\_fraction\_table(p\_подм\_2d, True)

Вывод:

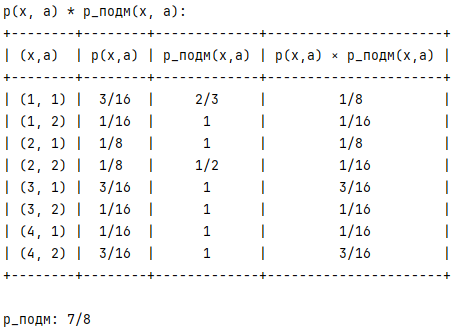


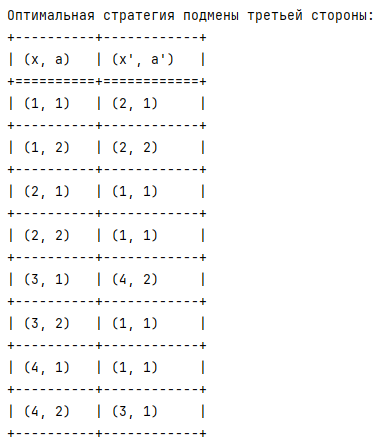
1. Построить оптимальную стратегию подмены.

В коде ниже сначала высчитывается умножение p(x, a) на p\_подм(x, a), затем оно суммируется, для получения p\_подм. После этого также находится оптимальная стратегия, сопоставляющая (x, a) и (x’, a’) с помощью поиска первых максимальных значений в каждой строке представленной выше таблицы.

*# Высчитываем p(x, a) и p(x, a) \* p\_подм(x, a)*p\_xa = (p\_им \* p\_X).flatten()  
p\_подм\_max = np.max(p\_подм\_2d, axis=1)  
p\_xa\_prod\_подм = p\_xa \* p\_подм\_max  
p\_подм = np.sum(p\_xa\_prod\_подм)  
  
print("\np(x, a) \* p\_подм(x, a):")  
table\_data = np.array([  
 [f"{x}" for x in indices\_combinations],  
 [str(frac) for frac in p\_xa],  
 [str(frac) for frac in p\_подм\_max],  
 [str(frac) for frac in p\_xa\_prod\_подм]  
], dtype=object)  
headers = ["(x,a)", "p(x,a)", "p\_подм(x,a)", "p(x,a) × p\_подм(x,a)"]  
print(tabulate(table\_data.T, headers=headers, tablefmt="pretty"))  
  
print("\np\_подм:", Fraction(p\_подм))  
  
*# Оптимальная стратегия подмены, обеспечивающая максимальную вероятность*opt\_substitution = {}  
for x in range(n\_x):  
 for a in range(n\_a):  
 *# Находим максимальную вероятность подмены для данной пары (x, a)* max\_p\_subm = np.max(p\_подм\_2d[x \* n\_a + a])  
 max\_p\_arg\_subm = np.argmax(p\_подм\_2d[x \* n\_a + a])  
 opt\_substitution[(x + 1, a + 1)] = (max\_p\_arg\_subm // n\_a + 1, max\_p\_arg\_subm % n\_a + 1)  
  
print("\nОптимальная стратегия подмены третьей стороны:")  
table\_data = []  
for key, value in opt\_substitution.items():  
 table\_data.append([str(key), str(value)])  
headers = ["(x, a)", "(x', a')"]  
print(tabulate(table\_data, headers=headers, tablefmt="grid"))

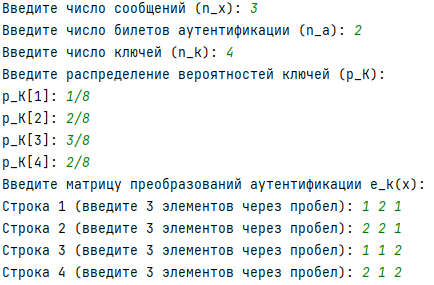
Вывод:

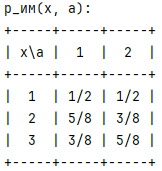




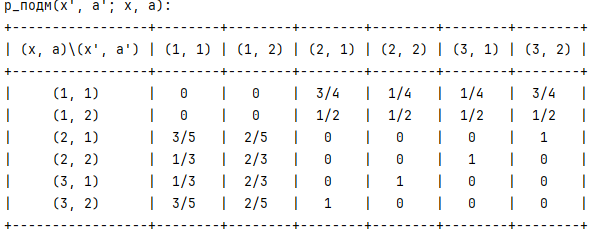
1. Проверка на других данных.

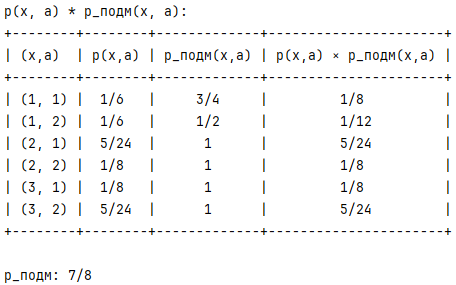
Ниже представлен вывод программы, на вход которой были поданы иные данные:

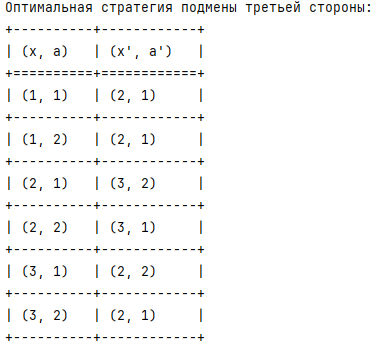












Задание 2.2

Написать программу для вычисления ортогонального массива (p,p+1,1)-OA и применить ее при разных p.

ВХОД: простое p.

ВЫХОД: OA(p,p+1,1).

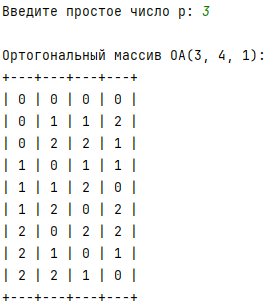
1. Решение

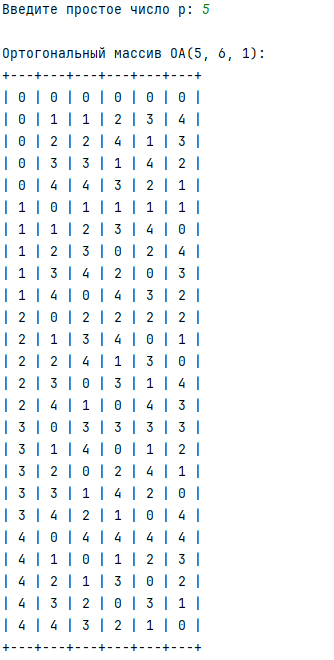
Была создана функция create\_orthogonal\_array().

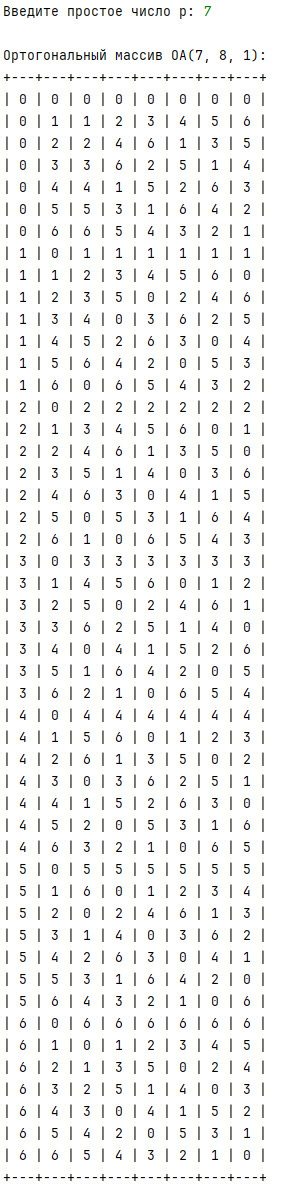
Функция создает ортогональный массив OA(p, p+1, 1), заполняя первый столбец числами от 0 до p-1, повторяя каждый элемент p раз (к примеру, 000111222), второй столбец – последовательностью чисел от 0 до p-1, повторенной p раз (к примеру, 012012012), а остальные столбцы – линейными комбинациями первого и второго столбцов по формуле (column\_1[j] + (i-2) \* column\_2[j]) % p, где i — номер столбца, а j — номер строки. С этой формулой получается массив размером p2 x (p+1), удовлетворяющий свойствам ортогонального массива.

from tabulate import tabulate  
  
  
def create\_orthogonal\_array(p):  
 *# Количество строк и столбцов* rows = p \* p  
 cols = p + 1  
 *# Создаем пустой массив* OA = [[0] \* cols for \_ in range(rows)]  
   
 *# Заполняем первые два столбца* for i in range(rows):  
 OA[i][0] = i // p  
 OA[i][1] = i % p  
  
 *# Заполняем остальные столбцы* for col in range(2, cols):  
 for row in range(rows):  
 OA[row][col] = (OA[row][0] + (col - 1) \* OA[row][1]) % p  
  
 return OA  
  
while True:  
 p = int(input("Введите простое число p: "))  
 *# Проверка, что p - простое число* if p < 2 or any(p % i == 0 for i in range(2, int(p \*\* 0.5) + 1)):  
 print("Ошибка: число должно быть простым.")  
 else:  
 oa = create\_orthogonal\_array(p)  
 *# Вывод результата с использованием tabulate* print(f"\nОртогональный массив OA({p}, {p + 1}, 1):")  
 print(tabulate(oa, tablefmt="pretty"))

Вывод программы с вводом различных p:







Полный код программ выложен на Github:

Задание 2.1: <https://github.com/Zaicol/CryptoMPEI/blob/master/task_2.1.py>

Задание 2.2: <https://github.com/Zaicol/CryptoMPEI/blob/master/task_2.2.py>