Министерство науки и высшего образования РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

**«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт космических и информационных технологий

институт

Программная инженерия

кафедра

**ОТЧЕТ О ПРАКТИЧЕСКОЙ** **РАБОТЕ №1**

Простые симметричные шифры

тема

Преподаватель Р. С. Шиманович

подпись, дата инициалы, фамилия

Студент КИ23-16/1Б, 032322546 Е. А. Гуртякин

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2025

**СОДЕРЖАНИЕ**

[Задание 3](#_Toc210131248)

[Ход выполнения 4](#_Toc210131249)

[Описание алгоритма шифрования 4](#_Toc210131250)

[Программа, реализующая алгоритм 9](#_Toc210131251)

[Модификация алгоритма 13](#_Toc210131252)

[Программа, реализующая модифицированный алгоритм 17](#_Toc210131253)

[Вывод 23](#_Toc210131254)

[ПРИЛОЖЕНИЕ А 24](#_Toc210131255)

[ПРИЛОЖЕНИЕ Б 27](#_Toc210131256)

# Задание

Согласно Вашему персональному варианту (см. табл. 2) или индивидуальному заданию преподавателя разработайте и составьте в виде блок-схемы алгоритмы шифрования и расшифровывания текста. Убедитесь в правильности составления алгоритмов и затем на языке программирования составьте программу, которая реализует данные алгоритмы. На ряде контрольных примеров (не менее 10) открытого текста, состоящего из различного количества символов, проверьте правильность работы алгоритмов шифрования и расшифровывания. Самостоятельно придумайте способ модификации шифра с целью повышения его криптостойкости. Для этого используйте блоки подстановочных/перестановочных операций. Внесите изменения в исходный алгоритм и программу. Проверьте работоспособность алгоритма на тестовых примерах. Покажите, что предложенный Вами способ модификации действительно повышает криптостокость. Разработанная Вами программа должна содержать графический интерфейс пользователя. Выбранный вариант - №1 (Шифр на основе магических квадратов).

# Ход выполнения

# Описание алгоритма шифрования

На рисунках 1 и 2 продемонстрирована блок-схема шифрования текста.

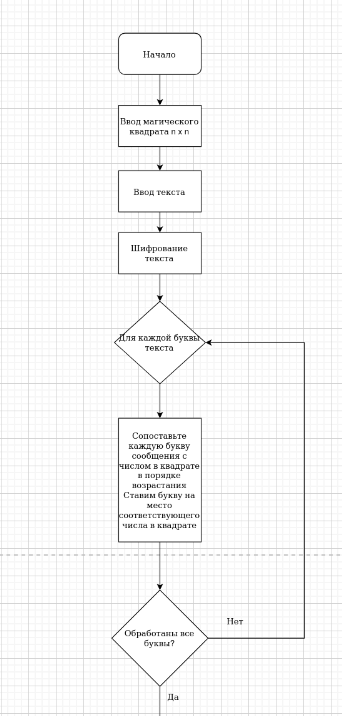


Рисунок 1 – Алгоритм шифрования текста (часть 1)

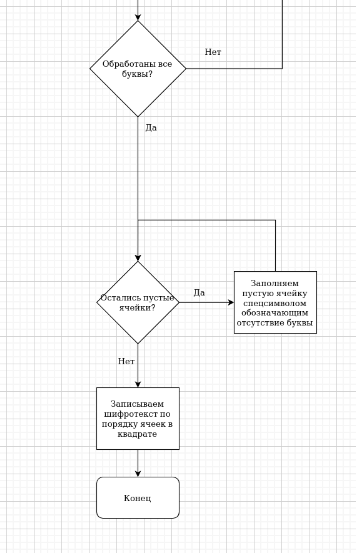


Рисунок 2 – Алгоритм шифрования текста (часть 2)

На рисунках 3 и 4 продемонстрирована блок-схема расшифровки текста.

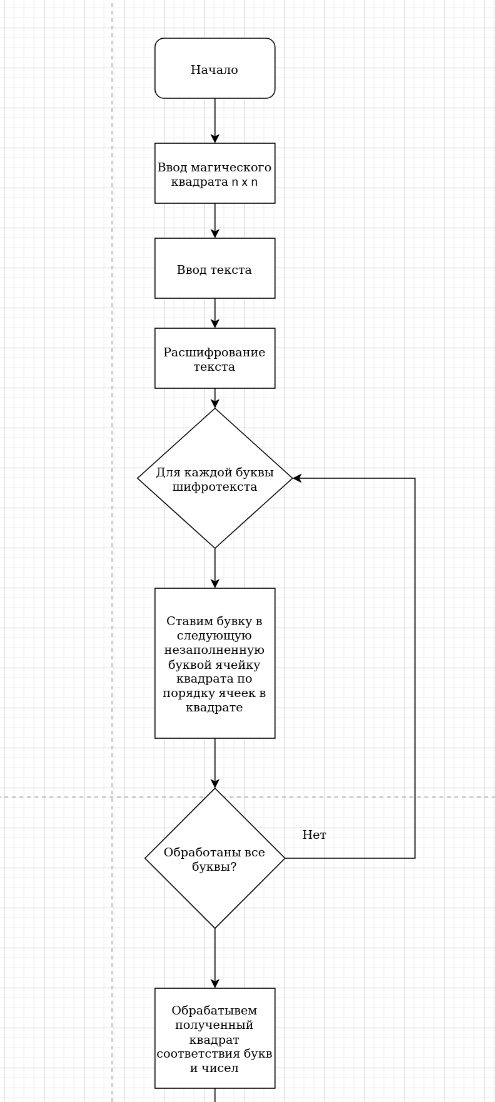


Рисунок 3 – Блок-схема расшифровки текста (часть 1)

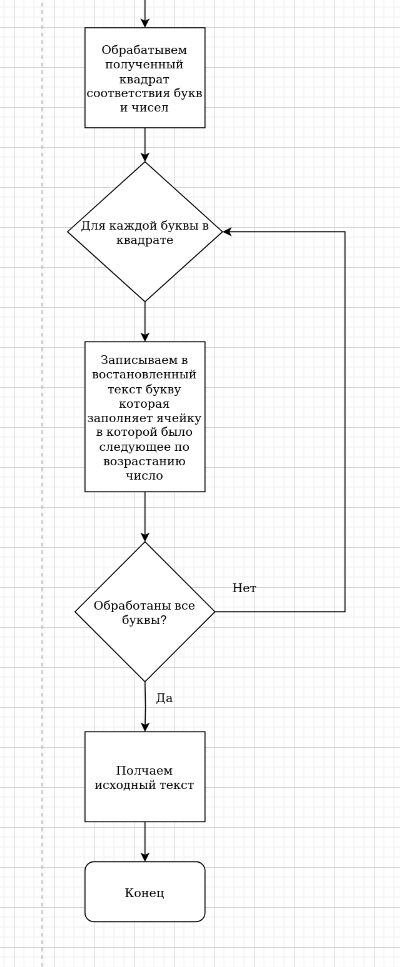


Рисунок 4 – Блок-схема расшифровки текста (часть 2)

# Программа, реализующая алгоритм

Для реализации шифра магического квадрата была написана программа на Python, представленная в приложении А. Результаты работы программы демонстрируются на рисунках 5 и 6.

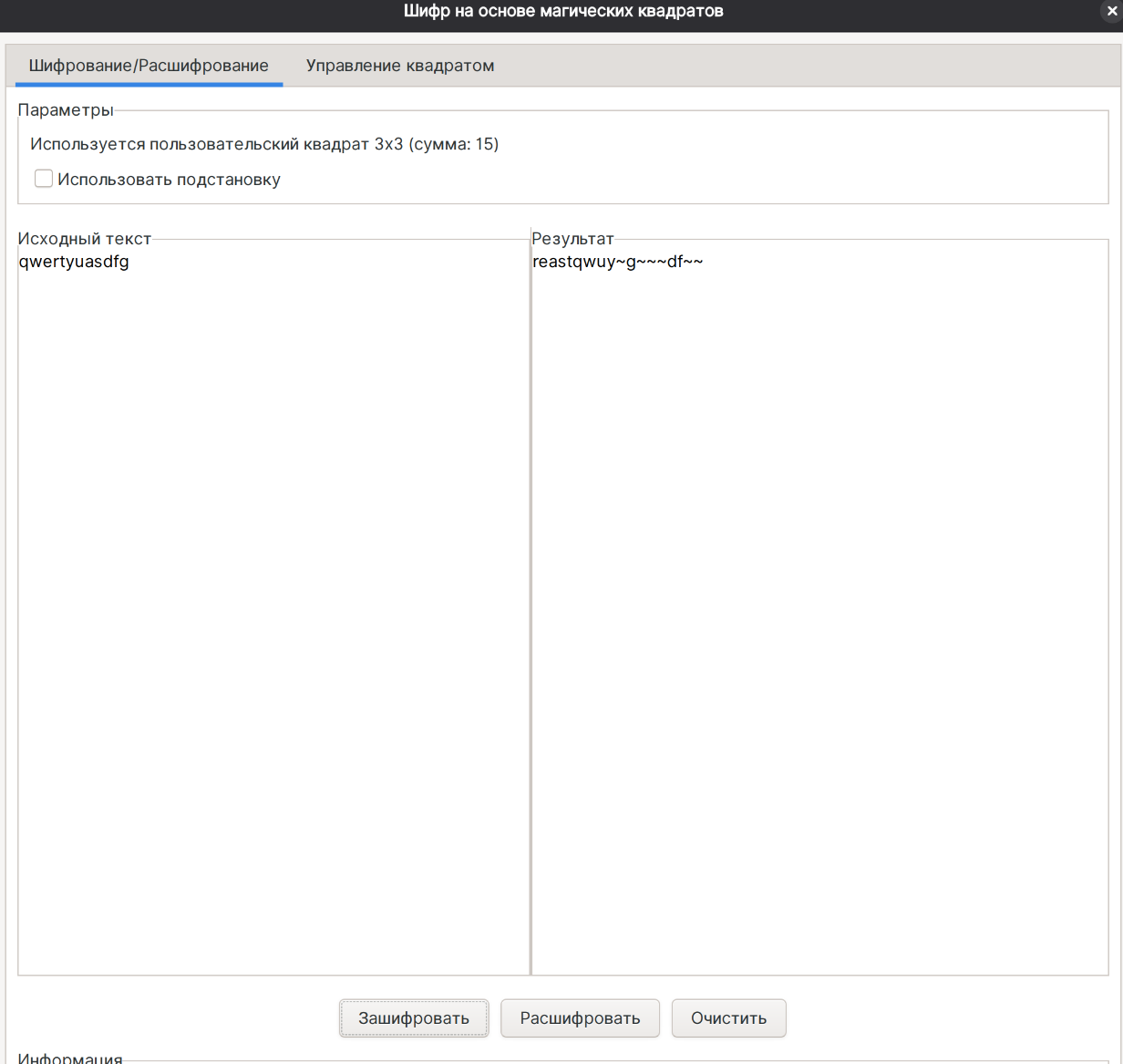


Рисунок 5 – Шифрование сообщения

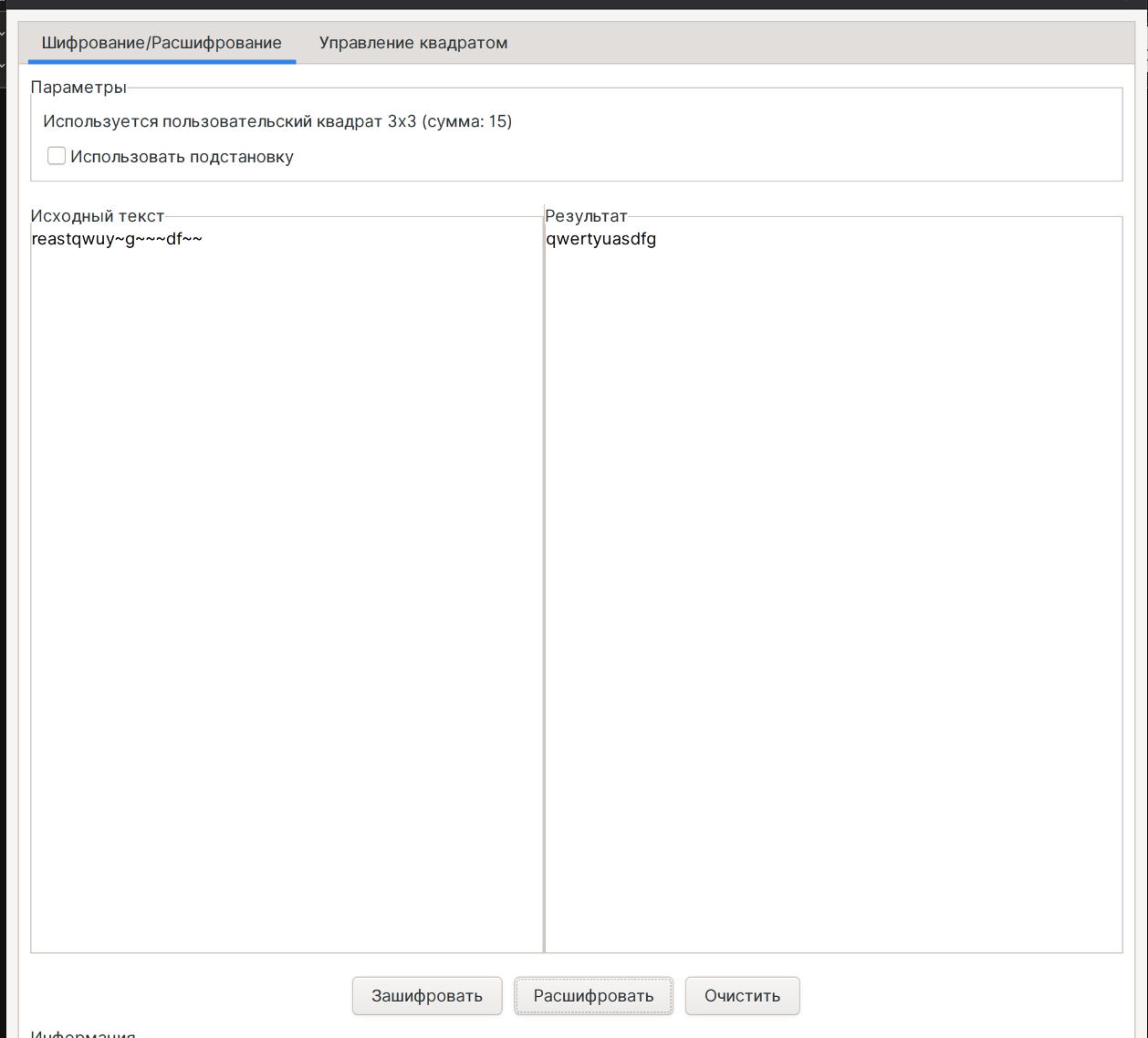


Рисунок 6 – Расшифровка сообщения

# Модификация алгоритма

Предлагаемая модификация добавляет подстановку символов исходного текста на основе данных самого магического квадрата. Тем самым расшифровка полученного шифротекста злоумышленником становится более сложной за счёт необходимости подбора и магического квадрата и способа генерации ключа на его основе. Ключ генерируется на основе следующих параметров:

* Сумма всех элементов - базовая характеристика размера квадрата
* Произведение главной диагонали - характеризует одну из ключевых линий квадрата
* Произведение побочной диагонали - характеризует вторую ключевую линию
* Сумма угловых элементов - захватывает информацию о границах квадрата
* Центральный элемент (или сумма центральных) - характеризует сердцевину квадрата
* Сумма разностей соседей - измеряет "шероховатость" распределения чисел
* Комбинирование через битовые операции:
  + XOR - для нелинейного комбинирования
  + Циклические сдвиги - для распространения битовой информации
  + Магические константы - эмпирически подобранные числа для хорошего перемешивания
  + Умножение на простые числа - для дополнительного перемешивания

На рисунке 11 представлена блок-схема для шифрования сообщения, с выделением изменившейся части алгоритма.

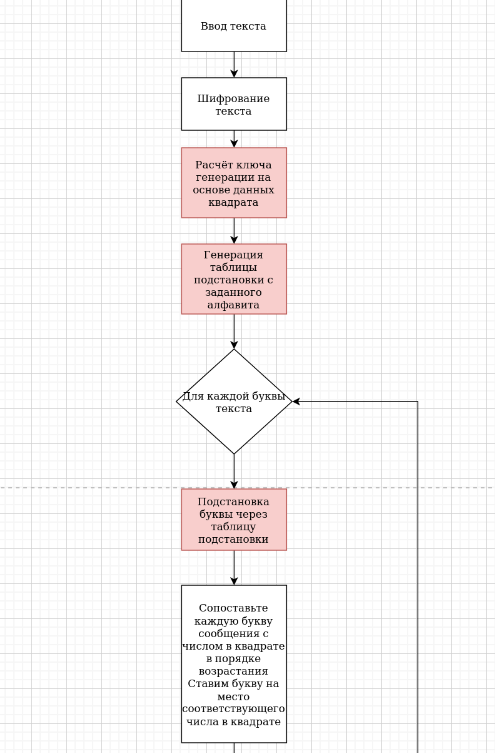


Рисунок 11 – Блок-схема шифрования (часть 1)

На рисунках 12 и 13 демонстрируется блок-схема расшифровки сообщений с выделением изменившейся части алгоритма

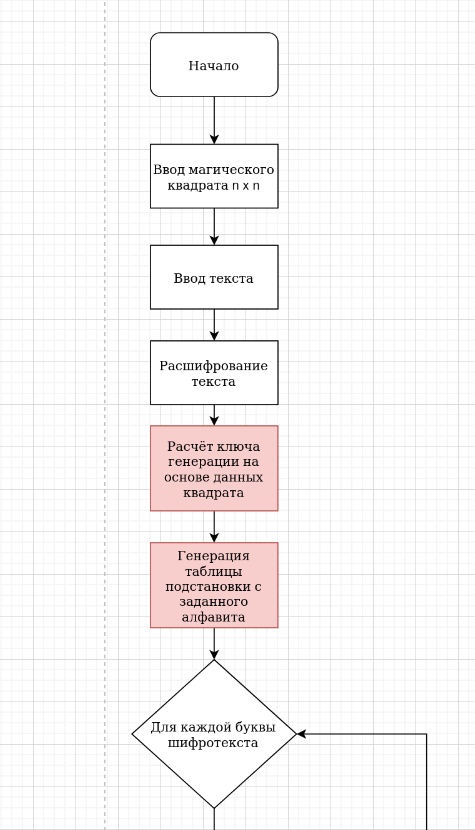


Рисунок 12 – Блок-схема расшифровки (часть 1)

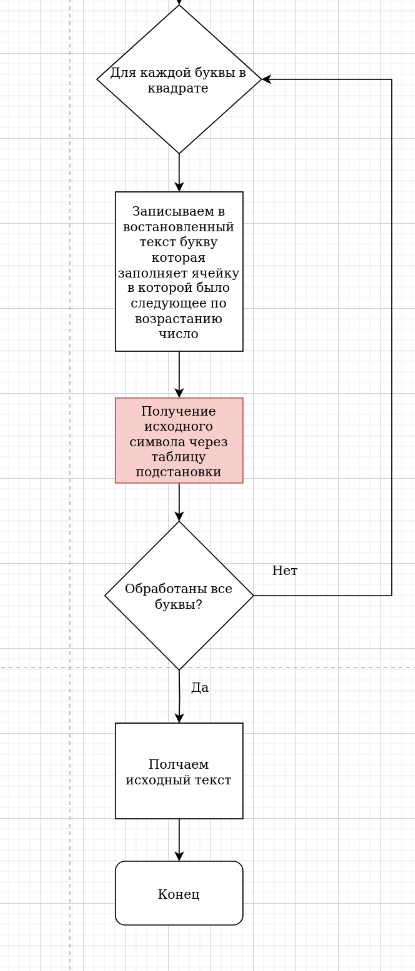


Рисунок 13 – Блок-схема расшифровки (часть 2)

# Программа, реализующая модифицированный алгоритм

Для реализации алгоритма была написана программа на языке Python, представленная в Приложении А и Б. Результаты работы программы представлены на рисунках 14-20.

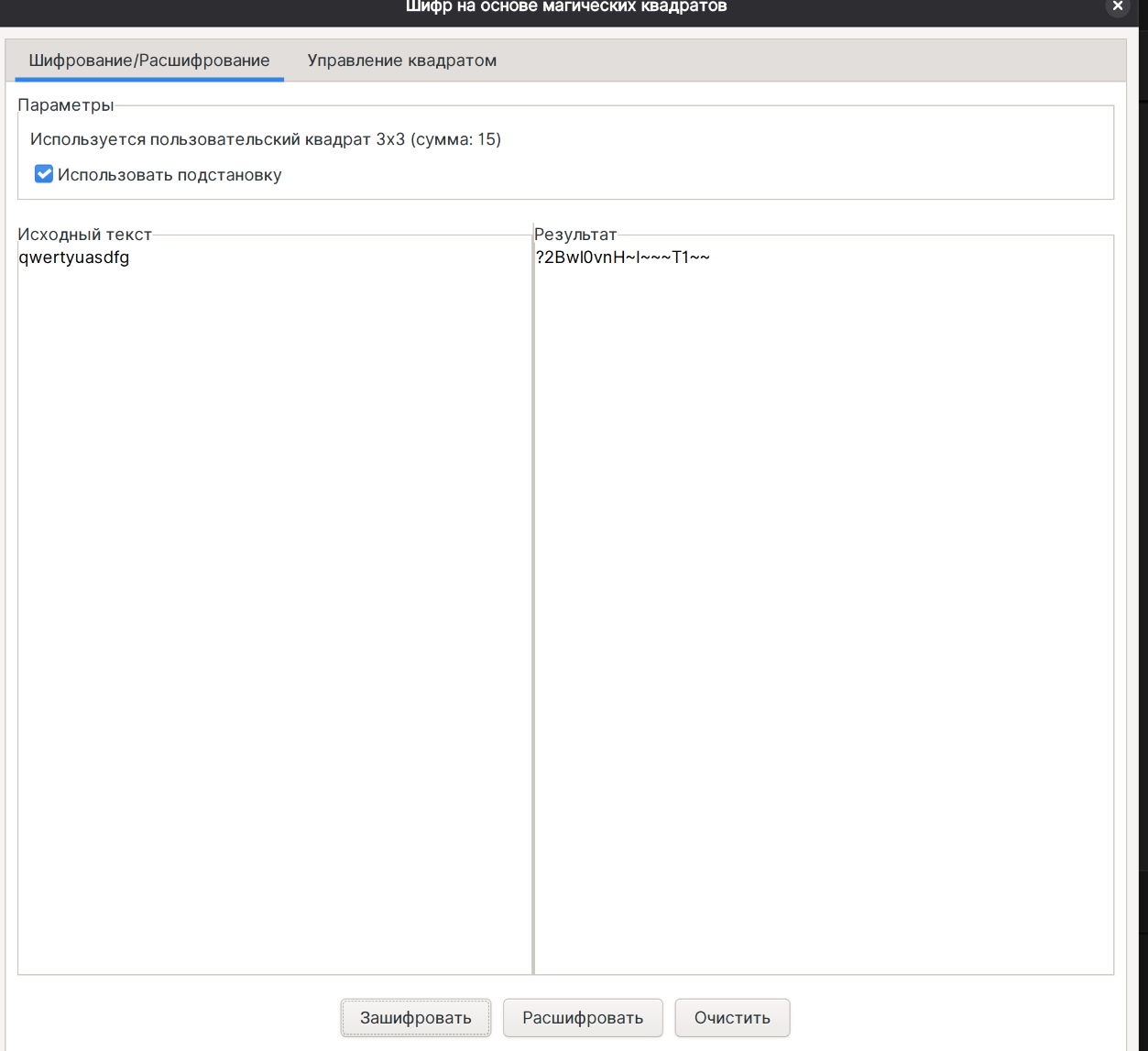


Рисунок 14 – Шифрование сообщения

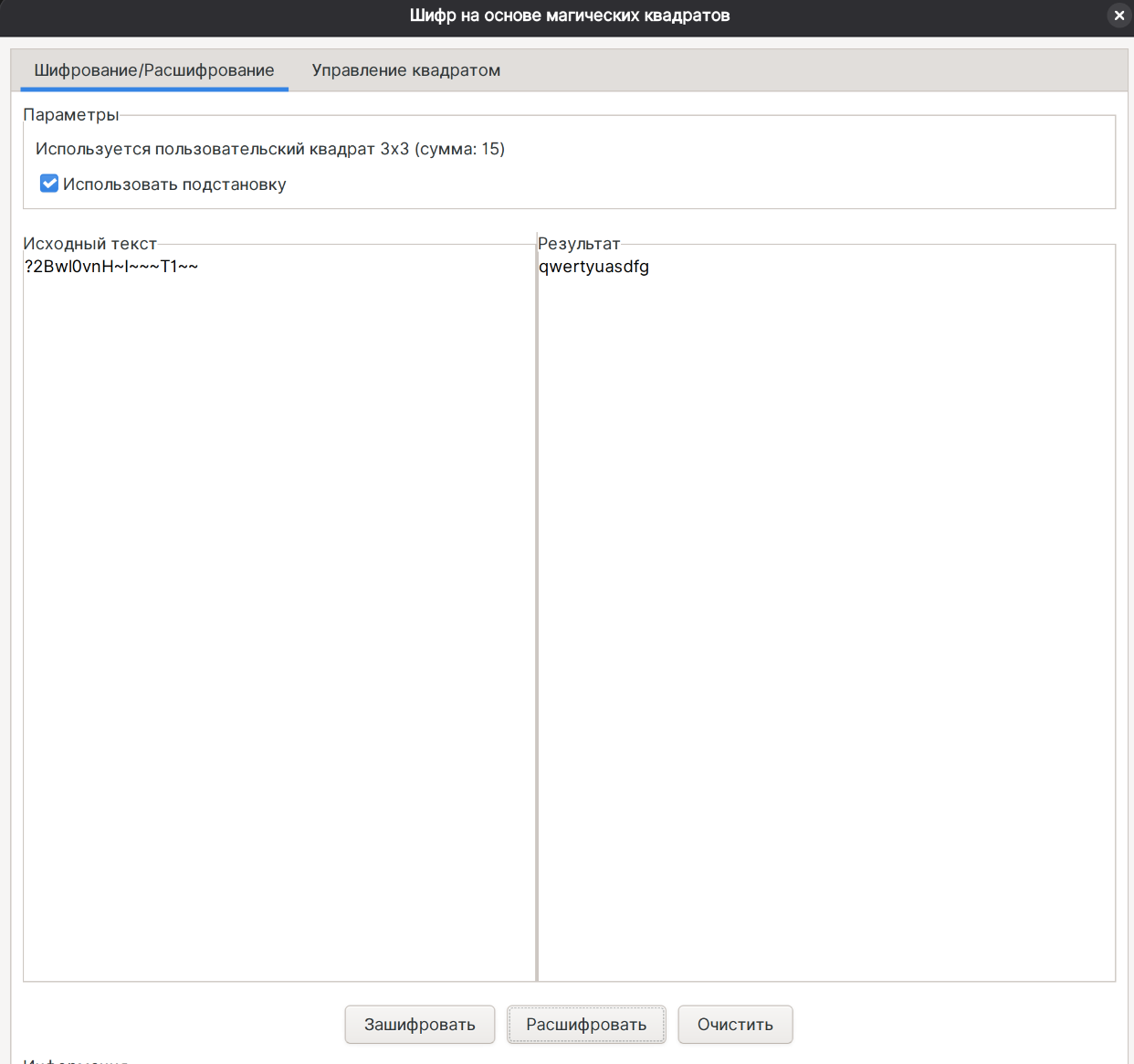


Рисунок 15 – Расшифровка сообщения

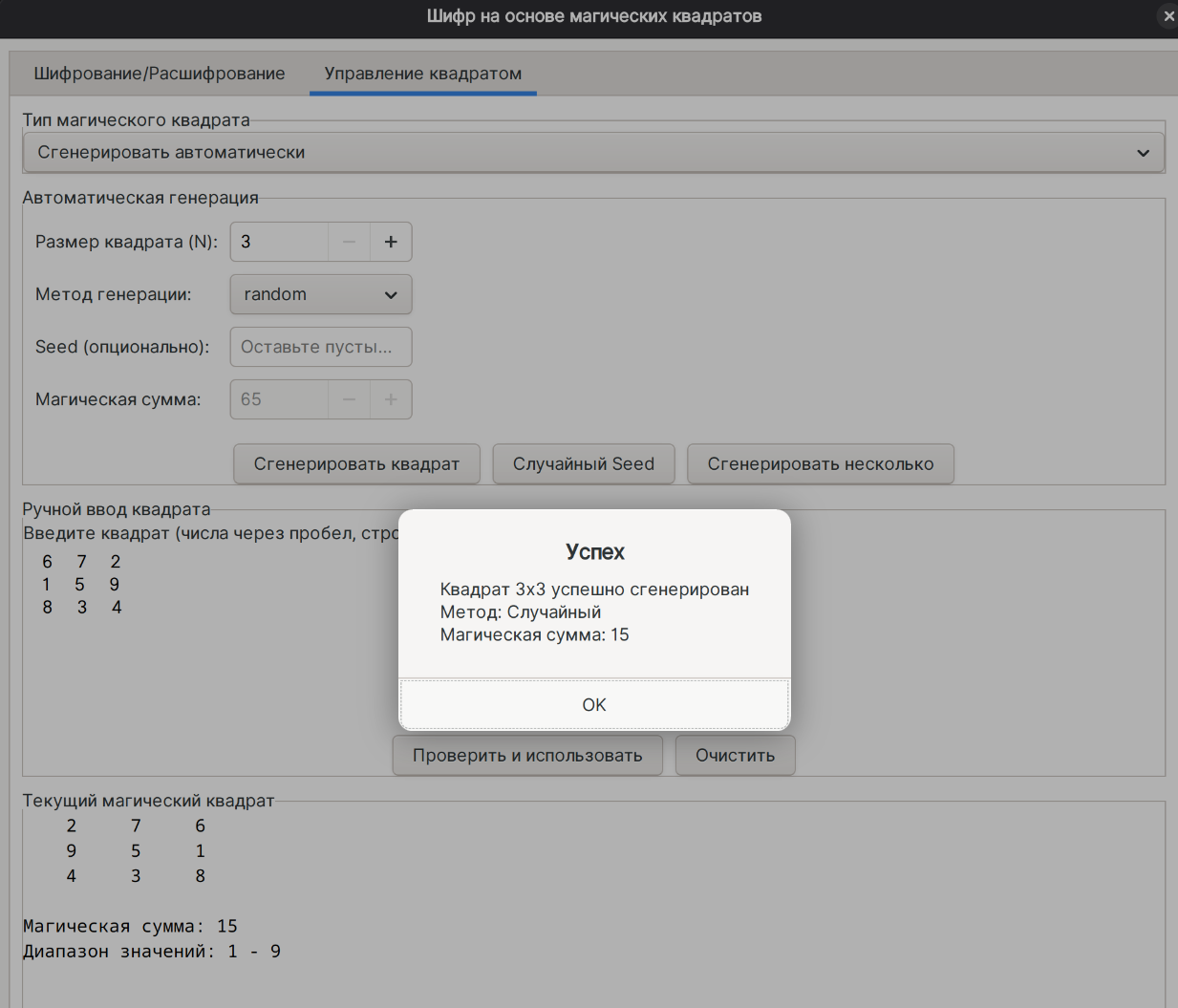


Рисунок 16 – Генерация нового квадрата

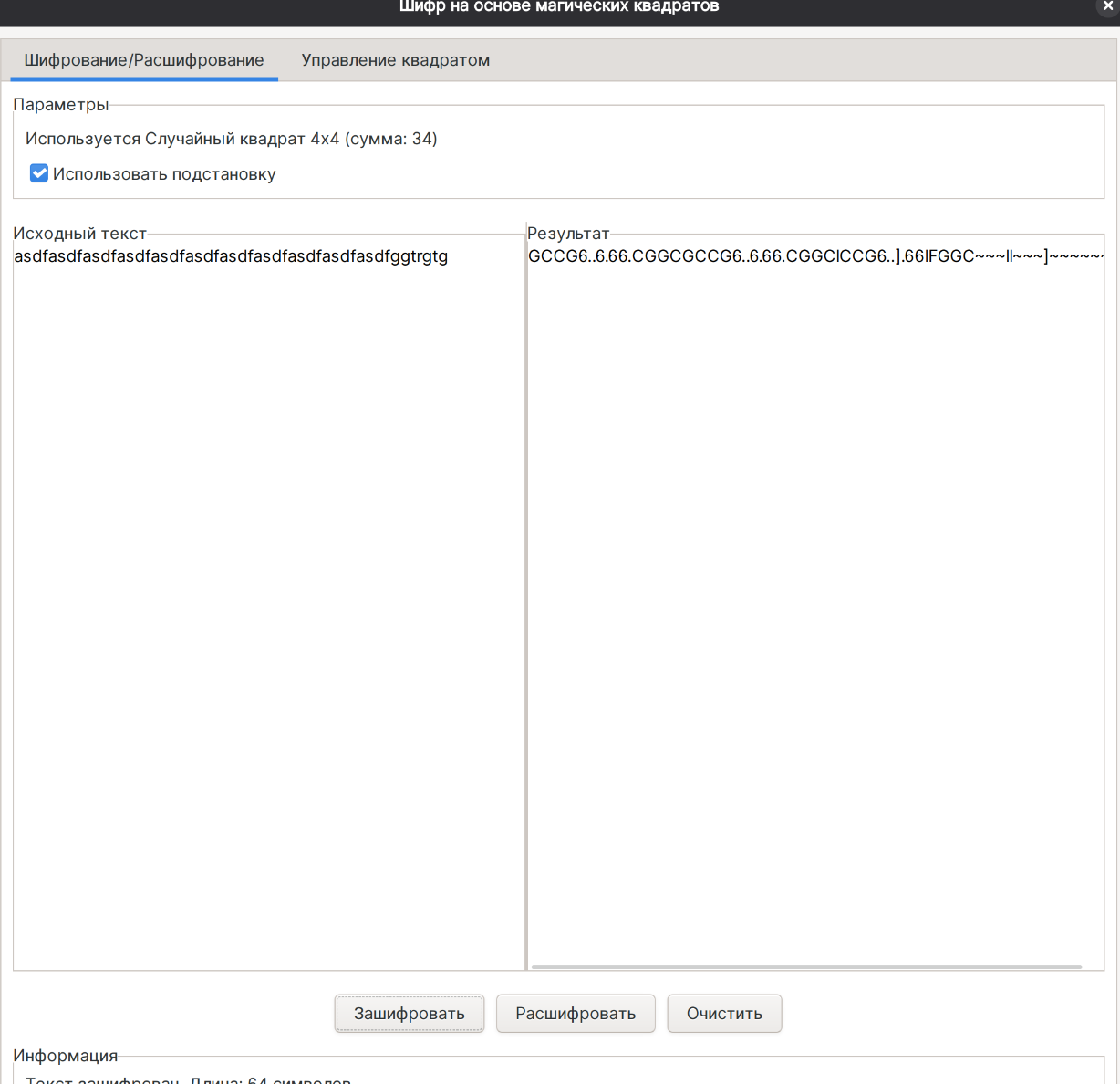


Рисунок 17 – Шифрование сообщения

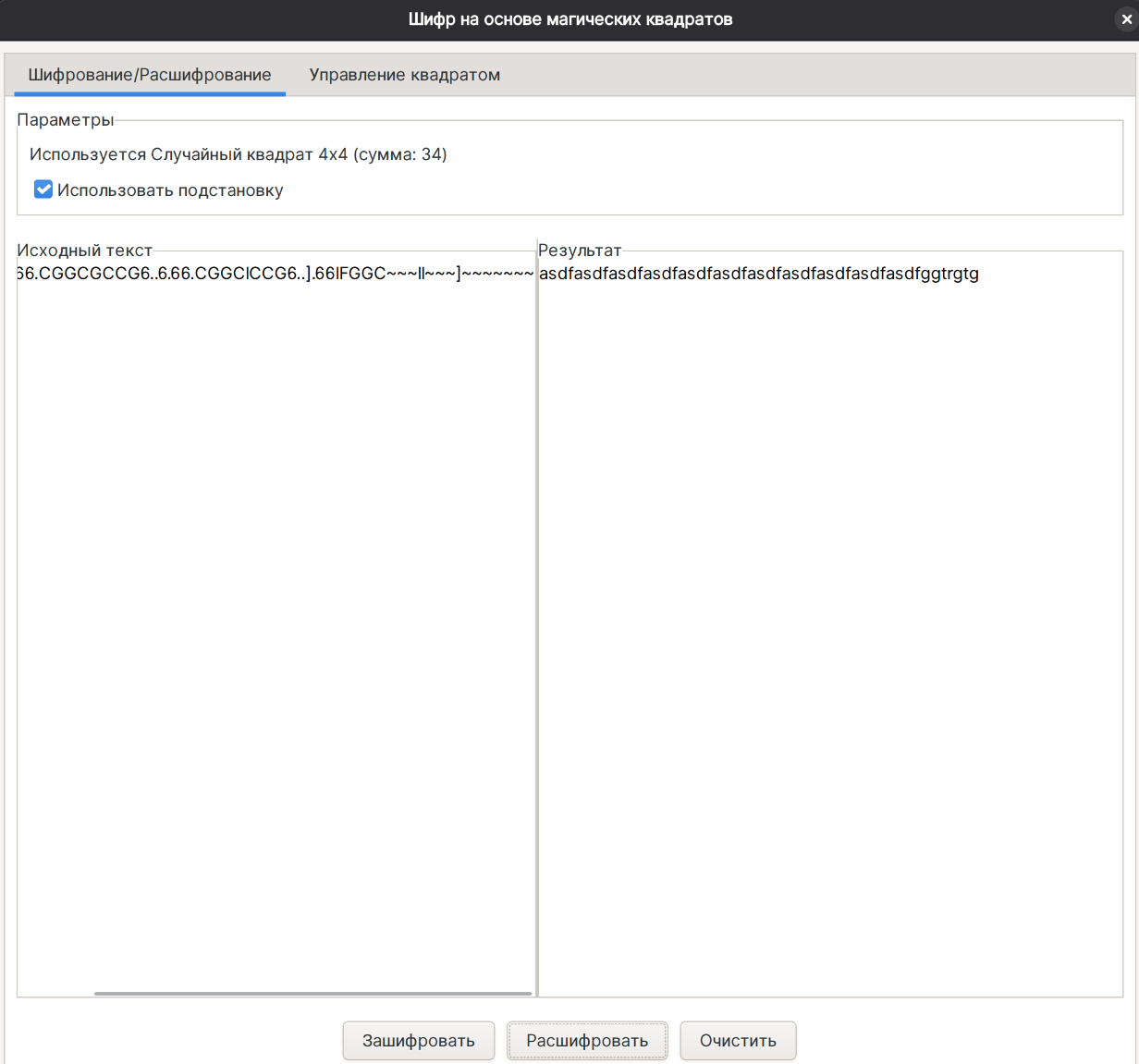


Рисунок 18 – Расшифровка сообщения

Вывод

В ходе выполнения практической работы были получены знания о симметричных шифрах, о криптостойкости алгоритмов шифрования, а также практические навыки реализации симметричных алгоритмов шифрования.

ПРИЛОЖЕНИЕ А

import numpy as np

import random

from typing import List, Tuple, Optional

class MagicSquareCipher:

"""Класс для шифрования на основе магических квадратов с поддержкой текстов любой длины"""

def \_\_init\_\_(self):

self.padding\_char = "~"

self.substitution\_key = None

def validate\_magic\_square(self, square: np.ndarray, check\_uniqueness: bool = True) -> Tuple[bool, str]:

"""

Проверка, является ли квадрат магическим

Args:

square: квадрат для проверки

check\_uniqueness: проверять ли уникальность чисел

Returns:

tuple: (is\_valid: bool, message: str)

"""

n = square.shape[0]

if square.shape != (n, n):

return False, f"Квадрат должен быть размером {n}x{n}"

if check\_uniqueness:

all\_numbers = square.flatten()

if len(set(all\_numbers)) != n \* n:

return False, "Все числа в квадрате должны быть уникальными"

magic\_sum = np.sum(square[0, :])

# Проверка строк

for i in range(n):

if np.sum(square[i, :]) != magic\_sum:

return False, f"Строка {i+1} имеет неправильную сумму"

# Проверка столбцов

for j in range(n):

if np.sum(square[:, j]) != magic\_sum:

return False, f"Столбец {j+1} имеет неправильную сумму"

# Проверка диагоналей

if np.sum(np.diag(square)) != magic\_sum:

return False, "Главная диагональ имеет неправильную сумму"

if np.sum(np.diag(np.fliplr(square))) != magic\_sum:

return False, "Побочная диагональ имеет неправильную сумму"

return True, f"Квадрат корректен! Магическая сумма: {magic\_sum}"

def generate\_magic\_square(self, n: int, method: str = "random",

seed: Optional[int] = None,

magic\_sum: Optional[int] = None) -> np.ndarray:

"""

Генерация магического квадрата разными методами

Гарантирует возврат корректного магического квадрата

"""

if seed is not None:

np.random.seed(seed)

random.seed(seed)

max\_attempts = 10

for attempt in range(max\_attempts):

try:

if method == "classic":

square = self.\_classic\_magic\_square(n)

elif method == "random":

square = self.\_classic\_with\_safe\_transformations(n)

elif method == "arithmetic":

square = self.\_arithmetic\_progression\_square(n, magic\_sum)

elif method == "geometric":

square = self.\_modular\_magic\_square(n, magic\_sum)

else:

square = self.\_classic\_magic\_square(n)

# Проверяем, что квадрат корректен

is\_valid, message = self.validate\_magic\_square(square)

if is\_valid:

return square

except Exception as e:

continue

# Если не удалось сгенерировать корректный квадрат, возвращаем классический

return self.\_classic\_magic\_square(n)

def \_classic\_with\_safe\_transformations(self, n: int) -> np.ndarray:

"""Классический квадрат с безопасными преобразованиями"""

base = self.\_classic\_magic\_square(n)

transformations = [self.\_rotate\_square, self.\_reflect\_square, self.\_transpose\_square]

result = base.copy()

for \_ in range(random.randint(2, 5)):

result = random.choice(transformations)(result)

return result

def \_ensure\_encryption\_range(self, square: np.ndarray) -> np.ndarray:

"""

Гарантирует, что квадрат можно использовать для шифрования

Преобразует квадрат так, чтобы числа были в диапазоне [1, n²]

и сохраняли магические свойства

"""

n = square.shape[0]

current\_min = np.min(square)

# Если числа уже в правильном диапазоне, возвращаем как есть

if current\_min >= 1 and np.max(square) <= n \* n:

# Проверяем уникальность

if len(set(square.flatten())) == n \* n:

return square

# Создаем новый классический квадрат и переносим магические свойства

classic\_square = self.\_classic\_magic\_square(n)

# Используем относительный порядок чисел из исходного квадрата

# чтобы создать перестановку классического квадрата

flat\_original = square.flatten()

flat\_classic = classic\_square.flatten()

# Сортируем индексы оригинального квадрата по значениям

sorted\_indices\_original = np.argsort(flat\_original)

# Сортируем классический квадрат

sorted\_classic = np.sort(flat\_classic)

# Создаем mapping: позиция в отсортированном оригинале -> значение из классического

result\_flat = np.zeros(n \* n, dtype=int)

result\_flat[sorted\_indices\_original] = sorted\_classic

return result\_flat.reshape(n, n)

def \_classic\_magic\_square(self, n: int) -> np.ndarray:

"""Классический магический квадрат с числами 1..n²"""

if n % 2 == 1:

return self.\_odd\_magic\_square(n)

elif n % 4 == 0:

return self.\_doubly\_even\_magic\_square(n)

else:

return self.\_singly\_even\_magic\_square(n)

def \_odd\_magic\_square(self, n: int) -> np.ndarray:

"""Генерация магического квадрата для нечетного n"""

magic\_square = np.zeros((n, n), dtype=int)

i, j = 0, n // 2

for num in range(1, n \* n + 1):

magic\_square[i, j] = num

new\_i, new\_j = (i - 1) % n, (j + 1) % n

if magic\_square[new\_i, new\_j] != 0:

i = (i + 1) % n

else:

i, j = new\_i, new\_j

return magic\_square

def \_doubly\_even\_magic\_square(self, n: int) -> np.ndarray:

"""Генерация магического квадрата для n кратного 4"""

magic\_square = np.arange(1, n \* n + 1).reshape(n, n)

for i in range(0, n, 4):

for j in range(0, n, 4):

for k in range(4):

magic\_square[i + k, j + k] = n \* n + 1 - magic\_square[i + k, j + k]

magic\_square[i + k, j + 3 - k] = (

n \* n + 1 - magic\_square[i + k, j + 3 - k]

)

return magic\_square

def \_singly\_even\_magic\_square(self, n: int) -> np.ndarray:

"""Генерация магического квадрата для n = 4k + 2"""

size = n // 2

magic\_square = np.zeros((n, n), dtype=int)

sub\_square = self.\_odd\_magic\_square(size)

# Заполнение квадрантов

magic\_square[:size, :size] = sub\_square

magic\_square[size:, size:] = sub\_square + size \* size

magic\_square[:size, size:] = sub\_square + 2 \* size \* size

magic\_square[size:, :size] = sub\_square + 3 \* size \* size

# Корректировка для магического квадрата

k = (n - 2) // 4

for i in range(size):

for j in range(k):

if i == size // 2:

j\_swap = j + k

else:

j\_swap = j

magic\_square[i, j\_swap], magic\_square[i + size, j\_swap] = (

magic\_square[i + size, j\_swap],

magic\_square[i, j\_swap],

)

for i in range(size):

for j in range(k - 1):

magic\_square[i, n - j - 1], magic\_square[i + size, n - j - 1] = (

magic\_square[i + size, n - j - 1],

magic\_square[i, n - j - 1],

)

return magic\_square

def \_rotate\_square(self, square: np.ndarray) -> np.ndarray:

"""Поворот квадрата"""

angle = random.choice([1, 2, 3])

return np.rot90(square, angle)

def \_reflect\_square(self, square: np.ndarray) -> np.ndarray:

"""Отражение квадрата"""

if random.choice([True, False]):

return np.fliplr(square)

else:

return np.flipud(square)

def \_transpose\_square(self, square: np.ndarray) -> np.ndarray:

"""Транспонирование квадрата - сохраняет магические свойства"""

return square.T

def \_swap\_symmetric\_rows\_columns(self, square: np.ndarray) -> np.ndarray:

"""Обмен симметричных строк и столбцов - ТОЛЬКО для квадратов с определенными свойствами"""

n = square.shape[0]

result = square.copy()

# Этот метод работает только для некоторых типов магических квадратов

if n % 2 == 1: # только для нечетных размеров

i = random.randint(0, n//2 - 1)

j = n - 1 - i

result[[i, j]] = result[[j, i]]

result[:, [i, j]] = result[:, [j, i]]

return result

def \_arithmetic\_progression\_square(self, n: int, magic\_sum: Optional[int] = None) -> np.ndarray:

"""

Арифметический магический квадрат

"""

if magic\_sum is None:

magic\_sum = n \* (n \* n + 1) // 2

# Создаем классический квадрат и масштабируем

base\_square = self.\_classic\_magic\_square(n)

base\_sum = np.sum(base\_square[0, :])

if base\_sum == 0:

return base\_square

# Масштабируем к нужной сумме

scale = magic\_sum / base\_sum

result = (base\_square \* scale).astype(int)

# Корректируем разницу

current\_sum = np.sum(result[0, :])

diff = magic\_sum - current\_sum

if diff != 0:

result[0, 0] += diff

return self.\_ensure\_encryption\_range(result)

def \_modular\_magic\_square(self, n: int, magic\_sum: Optional[int] = None) -> np.ndarray:

"""

Геометрический магический квадрат

"""

# Для реального шифрования всегда используем классический квадрат

# с преобразованием к нужной сумме

return self.\_arithmetic\_progression\_square(n, magic\_sum)

def get\_available\_methods(self) -> List[str]:

"""Возвращает список доступных методов генерации"""

return ["random", "classic", "arithmetic", "geometric"]

def calculate\_magic\_sum(self, square: np.ndarray) -> int:

"""Вычисление магической суммы квадрата"""

return int(np.sum(square[0, :]))

def create\_substitution\_table\_from\_magic\_square(self, magic\_square: np.ndarray) -> dict:

"""Создание таблицы подстановки на основе магического квадрата"""

chars = list(

set(

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 .,!?-()[]{}:;'\""

)

)

substituted = chars.copy()

seed = self.\_generate\_seed\_from\_magic\_square(magic\_square)

np.random.seed(seed)

np.random.shuffle(substituted)

return dict(zip(chars, substituted))

def \_generate\_seed\_from\_magic\_square(self, magic\_square: np.ndarray) -> int:

"""

Генерация seed на основе магического квадрата без внешних зависимостей.

Использует только свойства самого квадрата для детерминистической генерации.

"""

n = magic\_square.shape[0]

# 1. Сумма всех элементов квадрата

total\_sum = np.sum(magic\_square)

# 2. Произведение элементов главной диагонали

main\_diag = np.diag(magic\_square)

main\_diag\_product = 1

for num in main\_diag:

main\_diag\_product = (main\_diag\_product \* (num if num != 0 else 1)) & 0xFFFFFFFF

# 3. Произведение элементов побочной диагонали

anti\_diag = np.diag(np.fliplr(magic\_square))

anti\_diag\_product = 1

for num in anti\_diag:

anti\_diag\_product = (anti\_diag\_product \* (num if num != 0 else 1)) & 0xFFFFFFFF

# 4. Сумма угловых элементов

corners\_sum = (

magic\_square[0, 0] + # левый верхний

magic\_square[0, n-1] + # правый верхний

magic\_square[n-1, 0] + # левый нижний

magic\_square[n-1, n-1] # правый нижний

)

# 5. Центральный элемент (или сумма центральных для четных n)

if n % 2 == 1:

center\_value = magic\_square[n//2, n//2]

else:

center\_value = (

magic\_square[n//2-1, n//2-1] +

magic\_square[n//2-1, n//2] +

magic\_square[n//2, n//2-1] +

magic\_square[n//2, n//2]

)

# 6. Характеристика распределения чисел - сумма модулей разностей соседних элементов

neighbor\_diff\_sum = 0

for i in range(n):

for j in range(n-1):

neighbor\_diff\_sum += abs(magic\_square[i, j] - magic\_square[i, j+1])

if i < n-1:

neighbor\_diff\_sum += abs(magic\_square[i, 0] - magic\_square[i+1, 0])

# 7. Комбинируем все характеристики через битовые операции

seed = total\_sum

# XOR с произведением диагоналей

seed ^= main\_diag\_product

seed ^= (anti\_diag\_product << 16) | (anti\_diag\_product >> 16)

# Добавляем информацию об углах и центре

seed = (seed + corners\_sum \* 0x9E3779B9) & 0xFFFFFFFF

seed ^= (center\_value \* 0x85EBCA6B) & 0xFFFFFFFF

# Добавляем информацию о распределении чисел

seed = (seed + neighbor\_diff\_sum) & 0xFFFFFFFF

seed = ((seed << 7) | (seed >> 25)) ^ 0xDEADBEEF

# Финальное перемешивание

seed = (seed \* 0x343FD) & 0xFFFFFFFF

seed = (seed + 0x269EC3) & 0xFFFFFFFF

seed = seed ^ (seed >> 16)

seed = (seed \* 0x1B3) & 0xFFFFFFFF

seed = seed ^ (seed >> 4)

return seed & 0xFFFFFFFF

def apply\_substitution(self, text: str, magic\_square) -> str:

"""Применение подстановки к тексту"""

sub\_table = self.create\_substitution\_table\_from\_magic\_square(magic\_square)

result = []

for char in text:

if char in sub\_table:

result.append(sub\_table[char])

else:

result.append(char)

return "".join(result)

def reverse\_substitution(self, text: str, magic\_square) -> str:

"""Обратная подстановка"""

sub\_table = self.create\_substitution\_table\_from\_magic\_square(magic\_square)

reverse\_table = {v: k for k, v in sub\_table.items()}

result = []

for char in text:

if char in reverse\_table:

result.append(reverse\_table[char])

else:

result.append(char)

return "".join(result)

def encrypt(self, plaintext: str, n: int, use\_sub: bool,

method: str = "random", seed: Optional[int] = None,

magic\_sum: Optional[int] = None) -> str:

"""Шифрование текста любой длины с расширенной генерацией квадрата"""

magic\_square = self.generate\_magic\_square(n, method, seed, magic\_sum)

return self.encrypt\_with\_square(plaintext, magic\_square, use\_sub)

def decrypt(self, ciphertext: str, n: int, use\_sub: bool,

method: str = "random", seed: Optional[int] = None,

magic\_sum: Optional[int] = None) -> str:

"""Расшифрование текста любой длины с расширенной генерацией квадрата"""

magic\_square = self.generate\_magic\_square(n, method, seed, magic\_sum)

return self.decrypt\_with\_square(ciphertext, magic\_square, use\_sub)

def encrypt\_with\_square(self, plaintext: str, magic\_square: np.ndarray, use\_sub: bool) -> str:

"""Шифрование текста любой длины с использованием заданного квадрата"""

n = magic\_square.shape[0]

block\_size = n \* n

# Применение подстановки ко всему тексту

if use\_sub:

plaintext = self.apply\_substitution(plaintext, magic\_square)

# Разбиваем текст на блоки

encrypted\_blocks = []

for i in range(0, len(plaintext), block\_size):

block = plaintext[i:i + block\_size]

# Дополнение последнего блока если нужно

if len(block) < block\_size:

block += self.padding\_char \* (block\_size - len(block))

# Шифруем блок

encrypted\_block = self.\_encrypt\_single\_block(block, magic\_square)

encrypted\_blocks.append(encrypted\_block)

return "".join(encrypted\_blocks)

def decrypt\_with\_square(self, ciphertext: str, magic\_square: np.ndarray, use\_sub: bool) -> str:

"""Расшифрование текста любой длины с использованием заданного квадрата"""

n = magic\_square.shape[0]

block\_size = n \* n

# Проверяем, что длина шифротекста кратна размеру блока

if len(ciphertext) % block\_size != 0:

# Если нет, дополняем до кратности

padding\_needed = block\_size - (len(ciphertext) % block\_size)

ciphertext += self.padding\_char \* padding\_needed

# Разбиваем шифротекст на блоки

decrypted\_blocks = []

for i in range(0, len(ciphertext), block\_size):

block = ciphertext[i:i + block\_size]

# Расшифровываем блок

decrypted\_block = self.\_decrypt\_single\_block(block, magic\_square)

decrypted\_blocks.append(decrypted\_block)

# Объединяем и удаляем дополнение

result = "".join(decrypted\_blocks)

result = result.rstrip(self.padding\_char)

# Обратная подстановка

if use\_sub:

result = self.reverse\_substitution(result, magic\_square)

return result

def \_encrypt\_single\_block(self, block: str, magic\_square: np.ndarray) -> str:

"""Шифрование одного блока текста"""

n = magic\_square.shape[0]

# Проверяем, что блок правильного размера

if len(block) != n \* n:

raise ValueError(f"Размер блока должен быть {n \* n}, получен {len(block)}")

# Размещение символов в матрице

text\_matrix = np.array(list(block)).reshape(n, n)

# Чтение по значениям магического квадрата

encrypted = [""] \* (n \* n)

for i in range(n):

for j in range(n):

position = magic\_square[i, j] - 1

encrypted[position] = text\_matrix[i, j]

return "".join(encrypted)

def \_decrypt\_single\_block(self, block: str, magic\_square: np.ndarray) -> str:

"""Расшифрование одного блока текста"""

n = magic\_square.shape[0]

# Проверяем, что блок правильного размера

if len(block) != n \* n:

raise ValueError(f"Размер блока должен быть {n \* n}, получен {len(block)}")

# Размещение символов по значениям магического квадрата

decrypted\_matrix = np.empty((n, n), dtype=str)

for i in range(n):

for j in range(n):

position = magic\_square[i, j] - 1

decrypted\_matrix[i, j] = block[position]

# Чтение по порядку

return "".join(decrypted\_matrix.flatten())

def get\_block\_size(self, magic\_square: np.ndarray) -> int:

"""Возвращает размер блока для заданного квадрата"""

n = magic\_square.shape[0]

return n \* n

def get\_encryption\_info(self, plaintext: str, magic\_square: np.ndarray) -> dict:

"""Возвращает информацию о процессе шифрования"""

n = magic\_square.shape[0]

block\_size = n \* n

total\_blocks = (len(plaintext) + block\_size - 1) // block\_size

magic\_sum = self.calculate\_magic\_sum(magic\_square)

return {

"square\_size": n,

"block\_size": block\_size,

"total\_blocks": total\_blocks,

"magic\_sum": magic\_sum,

"padding\_char": self.padding\_char

}

ПРИЛОЖЕНИЕ Б

import traceback

from cipher import MagicSquareCipher

import gi

gi.require\_version("Gtk", "3.0")

from gi.repository import Gtk, Gdk, GLib

import numpy as np

import threading

import random

class CipherGUI:

"""Графический интерфейс для шифровальщика на GTK"""

def \_\_init\_\_(self, cipher: MagicSquareCipher):

self.cipher = cipher

self.setup\_theme()

self.create\_ui()

def setup\_theme(self):

"""Настройка темы в соответствии с системой"""

settings = Gtk.Settings.get\_default()

self.detect\_system\_theme()

settings.set\_property("gtk-application-prefer-dark-theme", self.prefers\_dark)

def detect\_system\_theme(self):

"""Определение системной темы"""

settings = Gtk.Settings.get\_default()

current\_theme = settings.get\_property("gtk-theme-name")

print(f"Текущая тема GTK: {current\_theme}")

self.prefers\_dark = settings.get\_property("gtk-application-prefer-dark-theme")

display = Gdk.Display.get\_default()

if display:

app = Gtk.Application.get\_default()

if app:

app.prefers\_color\_scheme = (

Gtk.SettingsColorScheme.PREFER\_DARK

if self.prefers\_dark

else Gtk.SettingsColorScheme.PREFER\_LIGHT

)

def create\_ui(self):

"""Создание пользовательского интерфейса"""

# Главное окно

self.window = Gtk.Window(

title="Шифр на основе магических квадратов"

)

self.window.set\_default\_size(1000, 800)

self.window.connect("destroy", Gtk.main\_quit)

# Основной контейнер

main\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)

main\_box.set\_margin\_top(10)

main\_box.set\_margin\_bottom(10)

main\_box.set\_margin\_start(10)

main\_box.set\_margin\_end(10)

self.window.add(main\_box)

# Ноутбук (вкладки)

self.notebook = Gtk.Notebook()

main\_box.pack\_start(self.notebook, True, True, 0)

# Создание вкладок

self.create\_encrypt\_tab()

self.create\_square\_tab()

# Статусная строка

self.status\_bar = Gtk.Statusbar()

self.status\_context\_id = self.status\_bar.get\_context\_id("main")

main\_box.pack\_start(self.status\_bar, False, False, 0)

self.window.show\_all()

def create\_square\_tab(self):

"""Создание вкладки для управления магическим квадратом"""

square\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)

square\_box.set\_margin\_top(10)

square\_box.set\_margin\_bottom(10)

square\_box.set\_margin\_start(10)

square\_box.set\_margin\_end(10)

# Фрейм выбора типа квадрата

type\_frame = Gtk.Frame(label="Тип магического квадрата")

type\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

type\_frame.add(type\_box)

self.square\_type\_combo = Gtk.ComboBoxText()

self.square\_type\_combo.append\_text("Сгенерировать автоматически")

self.square\_type\_combo.append\_text("Ввести вручную")

self.square\_type\_combo.set\_active(0)

self.square\_type\_combo.connect("changed", self.on\_square\_type\_changed)

type\_box.pack\_start(self.square\_type\_combo, False, False, 0)

square\_box.pack\_start(type\_frame, False, False, 0)

# Фрейм для автоматической генерации

self.auto\_frame = Gtk.Frame(label="Автоматическая генерация")

auto\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)

self.auto\_frame.add(auto\_box)

# Параметры генерации

params\_grid = Gtk.Grid()

params\_grid.set\_column\_spacing(10)

params\_grid.set\_row\_spacing(10)

params\_grid.set\_margin\_top(10)

params\_grid.set\_margin\_bottom(10)

params\_grid.set\_margin\_start(10)

params\_grid.set\_margin\_end(10)

auto\_box.pack\_start(params\_grid, False, False, 0)

# Размер квадрата

size\_label = Gtk.Label(label="Размер квадрата (N):")

size\_label.set\_halign(Gtk.Align.START)

params\_grid.attach(size\_label, 0, 0, 1, 1)

self.size\_adjustment = Gtk.Adjustment(

value=5, lower=3, upper=15, step\_increment=1

)

self.size\_spin = Gtk.SpinButton(adjustment=self.size\_adjustment)

params\_grid.attach(self.size\_spin, 1, 0, 1, 1)

# Метод генерации

method\_label = Gtk.Label(label="Метод генерации:")

method\_label.set\_halign(Gtk.Align.START)

params\_grid.attach(method\_label, 0, 1, 1, 1)

self.method\_combo = Gtk.ComboBoxText()

for method in self.cipher.get\_available\_methods():

self.method\_combo.append\_text(method)

self.method\_combo.set\_active(0)

self.method\_combo.connect("changed", self.on\_method\_changed)

params\_grid.attach(self.method\_combo, 1, 1, 1, 1)

# Seed для случайной генерации

seed\_label = Gtk.Label(label="Seed (опционально):")

seed\_label.set\_halign(Gtk.Align.START)

params\_grid.attach(seed\_label, 0, 2, 1, 1)

self.seed\_entry = Gtk.Entry()

self.seed\_entry.set\_placeholder\_text("Оставьте пустым для случайного")

self.seed\_entry.set\_width\_chars(15)

params\_grid.attach(self.seed\_entry, 1, 2, 1, 1)

# Магическая сумма (для некоторых методов)

self.magic\_sum\_label = Gtk.Label(label="Магическая сумма:")

self.magic\_sum\_label.set\_halign(Gtk.Align.START)

params\_grid.attach(self.magic\_sum\_label, 0, 3, 1, 1)

self.magic\_sum\_adjustment = Gtk.Adjustment(

value=65, lower=10, upper=10000, step\_increment=1

)

self.magic\_sum\_spin = Gtk.SpinButton(adjustment=self.magic\_sum\_adjustment)

self.magic\_sum\_spin.set\_sensitive(False)

params\_grid.attach(self.magic\_sum\_spin, 1, 3, 1, 1)

# Кнопки генерации

button\_box = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10)

button\_box.set\_halign(Gtk.Align.CENTER)

self.generate\_btn = Gtk.Button(label="Сгенерировать квадрат")

self.generate\_btn.connect("clicked", self.on\_generate\_square\_clicked)

button\_box.pack\_start(self.generate\_btn, False, False, 0)

self.random\_seed\_btn = Gtk.Button(label="Случайный Seed")

self.random\_seed\_btn.connect("clicked", self.on\_random\_seed\_clicked)

button\_box.pack\_start(self.random\_seed\_btn, False, False, 0)

self.multiple\_btn = Gtk.Button(label="Сгенерировать несколько")

self.multiple\_btn.connect("clicked", self.on\_generate\_multiple\_clicked)

button\_box.pack\_start(self.multiple\_btn, False, False, 0)

auto\_box.pack\_start(button\_box, False, False, 0)

square\_box.pack\_start(self.auto\_frame, False, False, 0)

# Фрейм для ручного ввода

self.manual\_frame = Gtk.Frame(label="Ручной ввод квадрата")

self.manual\_frame.set\_visible(False)

manual\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

self.manual\_frame.add(manual\_box)

# Поле для ввода квадрата

input\_label = Gtk.Label(

label="Введите квадрат (числа через пробел, строки через новую строку):"

)

input\_label.set\_halign(Gtk.Align.START)

manual\_box.pack\_start(input\_label, False, False, 0)

self.square\_input\_scroll = Gtk.ScrolledWindow()

self.square\_input\_scroll.set\_hexpand(True)

self.square\_input\_scroll.set\_vexpand(True)

self.square\_input\_scroll.set\_min\_content\_height(150)

self.square\_input\_text = Gtk.TextView()

self.square\_input\_text.set\_wrap\_mode(Gtk.WrapMode.WORD)

self.square\_input\_buffer = self.square\_input\_text.get\_buffer()

self.square\_input\_scroll.add(self.square\_input\_text)

manual\_box.pack\_start(self.square\_input\_scroll, True, True, 0)

# Кнопки для ручного ввода

manual\_btn\_box = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10)

manual\_btn\_box.set\_halign(Gtk.Align.CENTER)

self.validate\_btn = Gtk.Button(label="Проверить и использовать")

self.validate\_btn.connect("clicked", self.on\_validate\_square\_clicked)

manual\_btn\_box.pack\_start(self.validate\_btn, False, False, 0)

self.clear\_square\_btn = Gtk.Button(label="Очистить")

self.clear\_square\_btn.connect("clicked", self.on\_clear\_square\_clicked)

manual\_btn\_box.pack\_start(self.clear\_square\_btn, False, False, 0)

manual\_box.pack\_start(manual\_btn\_box, False, False, 0)

square\_box.pack\_start(self.manual\_frame, False, False, 0)

# Область для отображения текущего квадрата

display\_frame = Gtk.Frame(label="Текущий магический квадрат")

display\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

display\_frame.add(display\_box)

self.square\_display\_scroll = Gtk.ScrolledWindow()

self.square\_display\_scroll.set\_hexpand(True)

self.square\_display\_scroll.set\_vexpand(True)

self.square\_display\_scroll.set\_min\_content\_height(250)

self.square\_display\_text = Gtk.TextView()

self.square\_display\_text.set\_editable(False)

self.square\_display\_text.set\_monospace(True)

self.square\_display\_buffer = self.square\_display\_text.get\_buffer()

self.square\_display\_scroll.add(self.square\_display\_text)

display\_box.pack\_start(self.square\_display\_scroll, True, True, 0)

# Информация о квадрате

self.square\_info\_label = Gtk.Label(label="Квадрат не задан")

self.square\_info\_label.set\_halign(Gtk.Align.START)

display\_box.pack\_start(self.square\_info\_label, False, False, 0)

square\_box.pack\_start(display\_frame, True, True, 0)

# Добавляем вкладку в ноутбук

self.notebook.append\_page(square\_box, Gtk.Label(label="Управление квадратом"))

def create\_encrypt\_tab(self):

"""Создание вкладки шифрования"""

encrypt\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)

encrypt\_box.set\_margin\_top(10)

encrypt\_box.set\_margin\_bottom(10)

encrypt\_box.set\_margin\_start(10)

encrypt\_box.set\_margin\_end(10)

# Фрейм параметров

params\_frame = Gtk.Frame(label="Параметры")

params\_frame.set\_margin\_bottom(10)

encrypt\_box.pack\_start(params\_frame, False, False, 0)

params\_grid = Gtk.Grid()

params\_grid.set\_column\_spacing(10)

params\_grid.set\_row\_spacing(10)

params\_grid.set\_margin\_top(10)

params\_grid.set\_margin\_bottom(10)

params\_grid.set\_margin\_start(10)

params\_grid.set\_margin\_end(10)

params\_frame.add(params\_grid)

# Информация о текущем квадрате

self.current\_square\_info = Gtk.Label(

label="Используется автоматический квадрат 5x5"

)

self.current\_square\_info.set\_halign(Gtk.Align.START)

params\_grid.attach(self.current\_square\_info, 0, 0, 3, 1)

# Чекбокс использования подстановки

self.use\_substitution = Gtk.CheckButton(label="Использовать подстановку")

params\_grid.attach(self.use\_substitution, 0, 1, 3, 1)

# Горизонтальный бокс для текстовых областей

text\_panes = Gtk.Paned(orientation=Gtk.Orientation.HORIZONTAL)

text\_panes.set\_position(450)

encrypt\_box.pack\_start(text\_panes, True, True, 0)

# Левая панель - исходный текст

input\_frame = Gtk.Frame(label="Исходный текст")

input\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

input\_frame.add(input\_box)

self.input\_scroll = Gtk.ScrolledWindow()

self.input\_scroll.set\_hexpand(True)

self.input\_scroll.set\_vexpand(True)

self.input\_scroll.set\_min\_content\_height(200)

self.input\_text = Gtk.TextView()

self.input\_text.set\_wrap\_mode(Gtk.WrapMode.WORD)

self.input\_buffer = self.input\_text.get\_buffer()

self.input\_scroll.add(self.input\_text)

input\_box.pack\_start(self.input\_scroll, True, True, 0)

text\_panes.add1(input\_frame)

# Правая панель - результат

output\_frame = Gtk.Frame(label="Результат")

output\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

output\_frame.add(output\_box)

self.output\_scroll = Gtk.ScrolledWindow()

self.output\_scroll.set\_hexpand(True)

self.output\_scroll.set\_vexpand(True)

self.output\_scroll.set\_min\_content\_height(200)

self.output\_text = Gtk.TextView()

self.output\_text.set\_wrap\_mode(Gtk.WrapMode.WORD)

self.output\_text.set\_editable(False)

self.output\_buffer = self.output\_text.get\_buffer()

self.output\_scroll.add(self.output\_text)

output\_box.pack\_start(self.output\_scroll, True, True, 0)

text\_panes.add2(output\_frame)

# Панель кнопок

button\_box = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10)

button\_box.set\_halign(Gtk.Align.CENTER)

button\_box.set\_margin\_top(10)

encrypt\_box.pack\_start(button\_box, False, False, 0)

# Кнопки

self.encrypt\_btn = Gtk.Button(label="Зашифровать")

self.encrypt\_btn.connect("clicked", self.on\_encrypt\_clicked)

button\_box.pack\_start(self.encrypt\_btn, False, False, 0)

self.decrypt\_btn = Gtk.Button(label="Расшифровать")

self.decrypt\_btn.connect("clicked", self.on\_decrypt\_clicked)

button\_box.pack\_start(self.decrypt\_btn, False, False, 0)

self.clear\_btn = Gtk.Button(label="Очистить")

self.clear\_btn.connect("clicked", self.on\_clear\_clicked)

button\_box.pack\_start(self.clear\_btn, False, False, 0)

# Информационная панель

info\_frame = Gtk.Frame(label="Информация")

info\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

info\_frame.add(info\_box)

self.info\_label = Gtk.Label(label=".....")

self.info\_label.set\_halign(Gtk.Align.START)

self.info\_label.set\_margin\_start(10)

self.info\_label.set\_margin\_top(5)

self.info\_label.set\_margin\_bottom(5)

info\_box.pack\_start(self.info\_label, True, True, 0)

encrypt\_box.pack\_start(info\_frame, False, False, 0)

# Добавляем вкладку в ноутбук

self.notebook.append\_page(

encrypt\_box, Gtk.Label(label="Шифрование/Расшифрование")

)

# Инициализация текущего квадрата

self.current\_square = None

self.current\_n = 5

self.is\_custom\_square = False

self.current\_method = "random"

def on\_method\_changed(self, widget):

"""Обработчик изменения метода генерации"""

method = widget.get\_active\_text()

self.current\_method = method

# Активируем поле магической суммы для соответствующих методов

if method in ["arithmetic", "geometric"]:

self.magic\_sum\_spin.set\_sensitive(True)

# Устанавливаем классическую магическую сумму по умолчанию

n = self.size\_spin.get\_value\_as\_int()

classic\_sum = n \* (n \* n + 1) // 2

self.magic\_sum\_adjustment.set\_value(classic\_sum)

# Предупреждение для geometric метода с четными размерами

if method == "geometric" and n % 2 == 0:

self.show\_message(

"Информация",

"Для четных размеров geometric метод использует arithmetic метод",

Gtk.MessageType.INFO,

)

else:

self.magic\_sum\_spin.set\_sensitive(False)

def on\_validate\_square\_clicked(self, widget):

"""Проверка и использование введенного квадрата"""

square\_text = self.get\_text\_from\_buffer(self.square\_input\_buffer)

if not square\_text.strip():

self.show\_message(

"Ошибка", "Введите магический квадрат", Gtk.MessageType.ERROR

)

return

try:

# Парсинг введенного квадрата

lines = square\_text.strip().split("\n")

n = len(lines)

# Проверка на квадратную форму

for i, line in enumerate(lines):

row = line.strip().split()

if len(row) != n:

raise ValueError(

f"Строка {i + 1} содержит {len(row)} элементов, ожидается {n}"

)

# Создание матрицы

square = np.zeros((n, n), dtype=int)

for i, line in enumerate(lines):

row = list(map(int, line.strip().split()))

square[i] = row

# Для пользовательских квадратов не проверяем уникальность чисел

is\_magic, message = self.cipher.validate\_magic\_square(

square, check\_uniqueness=False

)

if is\_magic:

self.current\_square = square

self.current\_n = n

self.is\_custom\_square = True

self.update\_square\_display(square)

magic\_sum = self.cipher.calculate\_magic\_sum(square)

self.update\_square\_info(f"Пользовательский квадрат {n}x{n}\n{message}")

self.current\_square\_info.set\_text(

f"Используется пользовательский квадрат {n}x{n} (сумма: {magic\_sum})"

)

self.show\_message(

"Успех",

f"Квадрат {n}x{n} корректен!\n{message}",

Gtk.MessageType.INFO,

)

else:

self.show\_message(

"Ошибка",

f"Квадрат не является магическим:\n{message}",

Gtk.MessageType.ERROR,

)

except ValueError as e:

self.show\_message(

"Ошибка", f"Ошибка формата: {str(e)}", Gtk.MessageType.ERROR

)

except Exception as e:

self.show\_message(

"Ошибка", f"Ошибка обработки: {str(e)}", Gtk.MessageType.ERROR

)

def on\_square\_type\_changed(self, widget):

"""Обработчик изменения типа квадрата"""

if widget.get\_active\_text() == "Ввести вручную":

self.manual\_frame.set\_visible(True)

self.auto\_frame.set\_visible(False)

else:

self.manual\_frame.set\_visible(False)

self.auto\_frame.set\_visible(True)

# Генерируем квадрат по умолчанию

self.generate\_default\_square()

def on\_random\_seed\_clicked(self, widget):

"""Генерация случайного seed"""

random\_seed = random.randint(1, 1000000)

self.seed\_entry.set\_text(str(random\_seed))

def generate\_default\_square(self):

"""Генерация квадрата по умолчанию"""

n = self.size\_spin.get\_value\_as\_int()

try:

method = self.method\_combo.get\_active\_text()

seed\_text = self.seed\_entry.get\_text().strip()

seed = int(seed\_text) if seed\_text else None

magic\_sum = None

if method in ["arithmetic", "geometric"]:

magic\_sum = int(self.magic\_sum\_spin.get\_value())

square = self.cipher.generate\_magic\_square(n, method, seed, magic\_sum)

self.current\_square = square

self.current\_n = n

self.is\_custom\_square = False

self.update\_square\_display(square)

magic\_sum\_actual = self.cipher.calculate\_magic\_sum(square)

method\_name = self.get\_method\_display\_name(method)

self.update\_square\_info(

f"{method\_name} квадрат {n}x{n}\nМагическая сумма: {magic\_sum\_actual}"

)

self.current\_square\_info.set\_text(

f"Используется {method\_name} квадрат {n}x{n} (сумма: {magic\_sum\_actual})"

)

except Exception as e:

self.show\_message(

"Ошибка", f"Ошибка генерации: {str(e)}", Gtk.MessageType.ERROR

)

def get\_method\_display\_name(self, method: str) -> str:

"""Получить отображаемое имя метода"""

names = {

"random": "Случайный",

"classic": "Классический",

"arithmetic": "Арифметический",

"geometric": "Геометрический",

}

return names.get(method, method)

def on\_generate\_square\_clicked(self, widget):

"""Обработчик кнопки генерации квадрата"""

n = self.size\_spin.get\_value\_as\_int()

try:

method = self.method\_combo.get\_active\_text()

seed\_text = self.seed\_entry.get\_text().strip()

seed = int(seed\_text) if seed\_text else None

magic\_sum = None

if method in ["arithmetic", "geometric"]:

magic\_sum = int(self.magic\_sum\_spin.get\_value())

square = self.cipher.generate\_magic\_square(n, method, seed, magic\_sum)

self.current\_square = square

self.current\_n = n

self.is\_custom\_square = False

self.update\_square\_display(square)

magic\_sum\_actual = self.cipher.calculate\_magic\_sum(square)

method\_name = self.get\_method\_display\_name(method)

self.update\_square\_info(

f"{method\_name} квадрат {n}x{n}\nМагическая сумма: {magic\_sum\_actual}"

)

self.current\_square\_info.set\_text(

f"Используется {method\_name} квадрат {n}x{n} (сумма: {magic\_sum\_actual})"

)

self.show\_message(

"Успех",

f"Квадрат {n}x{n} успешно сгенерирован\nМетод: {method\_name}\nМагическая сумма: {magic\_sum\_actual}",

Gtk.MessageType.INFO,

)

except Exception as e:

self.show\_message(

"Ошибка", f"Ошибка генерации: {str(e)}", Gtk.MessageType.ERROR

)

def on\_generate\_multiple\_clicked(self, widget):

"""Генерация нескольких квадратов для демонстрации"""

n = self.size\_spin.get\_value\_as\_int()

methods = self.cipher.get\_available\_methods()

results = []

for method in methods:

try:

if method in ["arithmetic", "geometric"]:

magic\_sum = random.randint(50, 500)

square = self.cipher.generate\_magic\_square(

n, method, None, magic\_sum

)

else:

square = self.cipher.generate\_magic\_square(n, method)

magic\_sum\_actual = self.cipher.calculate\_magic\_sum(square)

results.append(

f"{self.get\_method\_display\_name(method)}: сумма = {magic\_sum\_actual}"

)

except Exception:

results.append(f"{self.get\_method\_display\_name(method)}: ошибка")

message = "Результаты генерации разных квадратов:\n" + "\n".join(results)

self.show\_message("Сравнение методов", message, Gtk.MessageType.INFO)

def on\_clear\_square\_clicked(self, widget):

"""Очистка поля ввода квадрата"""

self.set\_text\_to\_buffer(self.square\_input\_buffer, "")

def update\_square\_display(self, square):

"""Обновление отображения квадрата"""

n = square.shape[0]

text = ""

for i in range(n):

row\_text = " ".join(f"{square[i, j]:5d}" for j in range(n))

text += row\_text + "\n"

# Добавляем информацию о магической сумме

magic\_sum = self.cipher.calculate\_magic\_sum(square)

text += f"\nМагическая сумма: {magic\_sum}"

# Добавляем информацию о диапазоне значений

min\_val = np.min(square)

max\_val = np.max(square)

text += f"\nДиапазон значений: {min\_val} - {max\_val}"

self.set\_text\_to\_buffer(self.square\_display\_buffer, text)

def update\_square\_info(self, info):

"""Обновление информации о квадрате"""

self.square\_info\_label.set\_text(info)

def get\_text\_from\_buffer(self, buffer):

"""Получить текст из текстового буфера"""

start\_iter = buffer.get\_start\_iter()

end\_iter = buffer.get\_end\_iter()

return buffer.get\_text(start\_iter, end\_iter, False).strip()

def set\_text\_to\_buffer(self, buffer, text):

"""Установить текст в текстовый буфер"""

buffer.set\_text(text)

def show\_message(self, title, message, message\_type=Gtk.MessageType.INFO):

"""Показать сообщение"""

dialog = Gtk.MessageDialog(

transient\_for=self.window,

flags=0,

message\_type=message\_type,

buttons=Gtk.ButtonsType.OK,

text=title,

)

dialog.format\_secondary\_text(message)

dialog.run()

dialog.destroy()

def on\_encrypt\_clicked(self, widget):

"""Обработчик кнопки шифрования"""

if self.current\_square is None:

self.show\_message(

"Ошибка", "Сначала задайте магический квадрат", Gtk.MessageType.ERROR

)

return

plaintext = self.get\_text\_from\_buffer(self.input\_buffer)

if not plaintext:

self.show\_message(

"Предупреждение",

"Введите текст для шифрования",

Gtk.MessageType.WARNING,

)

return

thread = threading.Thread(target=self.\_encrypt\_thread, args=(plaintext,))

thread.daemon = True

thread.start()

def \_encrypt\_thread(self, plaintext):

"""Поток для шифрования"""

try:

use\_sub = self.use\_substitution.get\_active()

if self.is\_custom\_square:

# Используем пользовательский квадрат

encrypted = self.cipher.encrypt\_with\_square(

plaintext, self.current\_square, use\_sub

)

else:

# Используем параметры генерации

n = self.current\_n

method = self.method\_combo.get\_active\_text()

seed\_text = self.seed\_entry.get\_text().strip()

seed = int(seed\_text) if seed\_text else None

magic\_sum = None

if method in ["arithmetic", "geometric"]:

magic\_sum = int(self.magic\_sum\_spin.get\_value())

encrypted = self.cipher.encrypt(

plaintext, n, use\_sub, method, seed, magic\_sum

)

GLib.idle\_add(

self.\_update\_output,

encrypted,

f"Текст зашифрован. Длина: {len(encrypted)} символов",

)

except Exception as e:

error\_msg = f"Ошибка при шифровании: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

GLib.idle\_add(

self.show\_message,

"Ошибка",

error\_msg,

Gtk.MessageType.ERROR,

)

def on\_decrypt\_clicked(self, widget):

"""Обработчик кнопки расшифрования"""

if self.current\_square is None:

self.show\_message(

"Ошибка", "Сначала задайте магический квадрат", Gtk.MessageType.ERROR

)

return

ciphertext = self.get\_text\_from\_buffer(self.input\_buffer)

if not ciphertext:

self.show\_message(

"Предупреждение",

"Введите текст для расшифрования",

Gtk.MessageType.WARNING,

)

return

thread = threading.Thread(target=self.\_decrypt\_thread, args=(ciphertext,))

thread.daemon = True

thread.start()

def \_decrypt\_thread(self, ciphertext):

"""Поток для расшифрования"""

try:

use\_sub = self.use\_substitution.get\_active()

if self.is\_custom\_square:

# Используем пользовательский квадрат

decrypted = self.cipher.decrypt\_with\_square(

ciphertext, self.current\_square, use\_sub

)

else:

# Используем параметры генерации

n = self.current\_n

method = self.method\_combo.get\_active\_text()

seed\_text = self.seed\_entry.get\_text().strip()

seed = int(seed\_text) if seed\_text else None

magic\_sum = None

if method in ["arithmetic", "geometric"]:

magic\_sum = int(self.magic\_sum\_spin.get\_value())

decrypted = self.cipher.decrypt(

ciphertext, n, use\_sub, method, seed, magic\_sum

)

GLib.idle\_add(

self.\_update\_output,

decrypted,

f"Текст расшифрован. Длина: {len(decrypted)} символов",

)

except Exception as e:

error\_msg = f"Ошибка при расшифровании: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

GLib.idle\_add(

self.show\_message,

"Ошибка",

error\_msg,

Gtk.MessageType.ERROR,

)

def \_update\_output(self, text, info\_message):

"""Обновить вывод и информацию"""

self.set\_text\_to\_buffer(self.output\_buffer, text)

self.info\_label.set\_text(info\_message)

self.status\_bar.push(self.status\_context\_id, info\_message)

def on\_clear\_clicked(self, widget):

"""Обработчик кнопки очистки"""

self.set\_text\_to\_buffer(self.input\_buffer, "")

self.set\_text\_to\_buffer(self.output\_buffer, "")

self.info\_label.set\_text("Поля очищены")

self.status\_bar.push(self.status\_context\_id, "Поля очищены")

def run(self):

"""Запуск приложения"""

# Генерируем квадрат по умолчанию при запуске

self.generate\_default\_square()

self.window.show\_all()

Gtk.main()