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 №2**

Основы криптографии с открытым ключом. Алгоритм RSA

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 2](#_Toc212045630)

[Ход выполнения 4](#_Toc212045631)

[Описание алгоритма шифрования 4](#_Toc212045632)

[Программа, реализующая алгоритм 7](#_Toc212045633)

[Вывод 13](#_Toc212045634)

[ПРИЛОЖЕНИЕ А 14](#_Toc212045635)

# Задание

Согласно Вашему персональному варианту (см. табл. 3) составьте в виде блок-схемы алгоритм шифрования/расшифрования RSA, со следующими особенностями: объём исходного текста – любой (в разумных пределах); исходный текст может состоять из русских и английских букв, цифр, а также знаков препинания; исходный текст находится в кодировке ASCII; выступающее в качестве модуля число – N, которое выбирается автоматически и состоит из количества ДЕСЯТИЧНЫХ знаков, указанных в табл. 3. Числа P и Q выбираются случайным образом, так, что P\*Q=N , где P и Q – простые числа. Исходный текст разбивается на K блоков, где K выбирается исходя из значения модуля N (см. табл. 3 и п.п. 2.4.3); Убедитесь в правильности составления алгоритмов, а затем на языке C# составьте программу, которая реализует данный алгоритм. На ряде контрольных примеров (не менее 10) открытого текста проверьте правильность работы алгоритмов шифрования и расшифрования (в качестве контрольного примера понимается текстовый файл в кодировке ASCII). Оцените криптостойкость вашего варианта алгоритма RSA, а также сделайте оценку производительности, разработанной Вами программы. Разработанная Вами программа должна содержать графический интерфейс пользователя. Выбранный вариант - №15 (32 знака).

# Ход выполнения

# Описание алгоритма шифрования

Шифр RSA, названный по именам его создателей — Ривеста, Шамира и Адлемана, является фундаментом современной асимметричной криптографии и используется для безопасного обмена данными, электронной подписи и защиты цифровой идентичности. В отличие от классических симметричных алгоритмов, в RSA применяются два различных, но математически связанных ключа: открытый и закрытый. Один используется для шифрования, другой — для расшифровки, что обеспечивает удобство обмена ключами без предварительного секретного канала.

В основе RSA лежат свойства арифметики больших чисел и трудность факторизации, то есть разложения составного числа на множители. Ключевая идея алгоритма заключена в том, что умножить два больших простых числа легко, а вот найти их по произведению — чрезвычайно сложно, особенно когда речь идёт о числе в 32 десятичных знака (примерно 106 бит). Эта односторонняя операция создаёт прочный барьер, стоящий на страже криптографической тайны.

Процесс создания ключей начинается с выбора двух различных больших простых чисел p и q. Их произведение n = p × q формирует модуль, общий для открытого и закрытого ключей. Далее вычисляется значение функции Эйлера φ(n) = (p−1)(q−1), определяющей количество чисел, взаимно простых с n. Открытая экспонента e выбирается так, чтобы 1 < e < φ(n) и e было взаимно просто с φ(n). После этого определяется закрытая экспонента d, являющаяся мультипликативным обратным к e по модулю φ(n), то есть удовлетворяющая условию e × d ≡ 1 (mod φ(n)).

Шифрование выполняется по формуле C = M^e mod n, где M — числовое представление открытого текста, а C — шифртекст. Расшифрование же производится обратной операцией M = C^d mod n. Таким образом, любой, кто знает открытый ключ (e, n), может зашифровать сообщение, но расшифровать его способен только обладатель закрытого ключа d.

Безопасность RSA основана на практической неразрешимости задачи факторизации больших чисел. Чтобы атакующий мог восстановить закрытый ключ, ему нужно разложить n на множители p и q. Для 32-значного числа это всё ещё выполнимо современными вычислительными средствами, поэтому в реальных системах длина модуля обычно превышает 600 десятичных знаков. Однако для демонстрации принципа 32-значного варианта вполне достаточно — он позволяет проследить логику алгоритма без погружения в гигантские вычисления.

Процедуры подписи и проверки работают аналогично, только порядок применения ключей меняется местами. Отправитель вычисляет цифровую подпись, возводя хэш сообщения в степень d по модулю n, а получатель проверяет её открытым ключом, восстанавливая исходный хэш и сравнивая его с вычисленным.

Криптостойкость RSA определяется длиной ключа и качеством генерации простых чисел. При корректной реализации и достаточной длине модуля алгоритм обеспечивает высокий уровень безопасности, а его математическая прозрачность позволяет легко анализировать прочность схемы. Историческое значение RSA трудно переоценить: он стал первым практически применимым алгоритмом с открытым ключом и до сих пор остаётся символом перехода криптографии от тайнописи к строгой науке.

На рисунках 1 и 2 представлены блок-схемы шифрования и дешифровки текста.

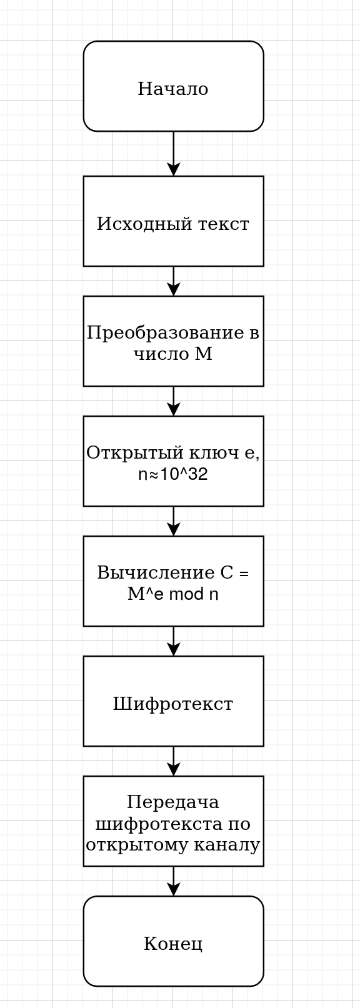


Рисунок 1 – Блок-схема шифрования текста

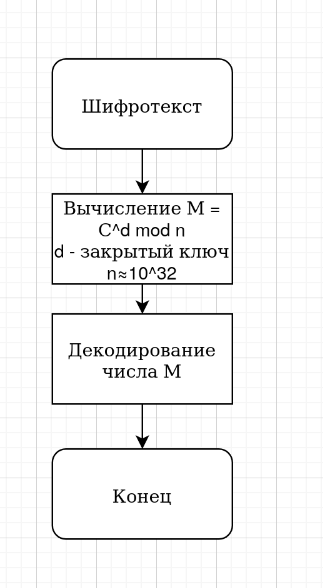


Рисунок 2 - Блок-схема дешифровки текста

# Программа, реализующая алгоритм

Для реализации шифра была написана программа на Python, представленная в приложении А. Результаты работы программы демонстрируются на рисунках 3-5.

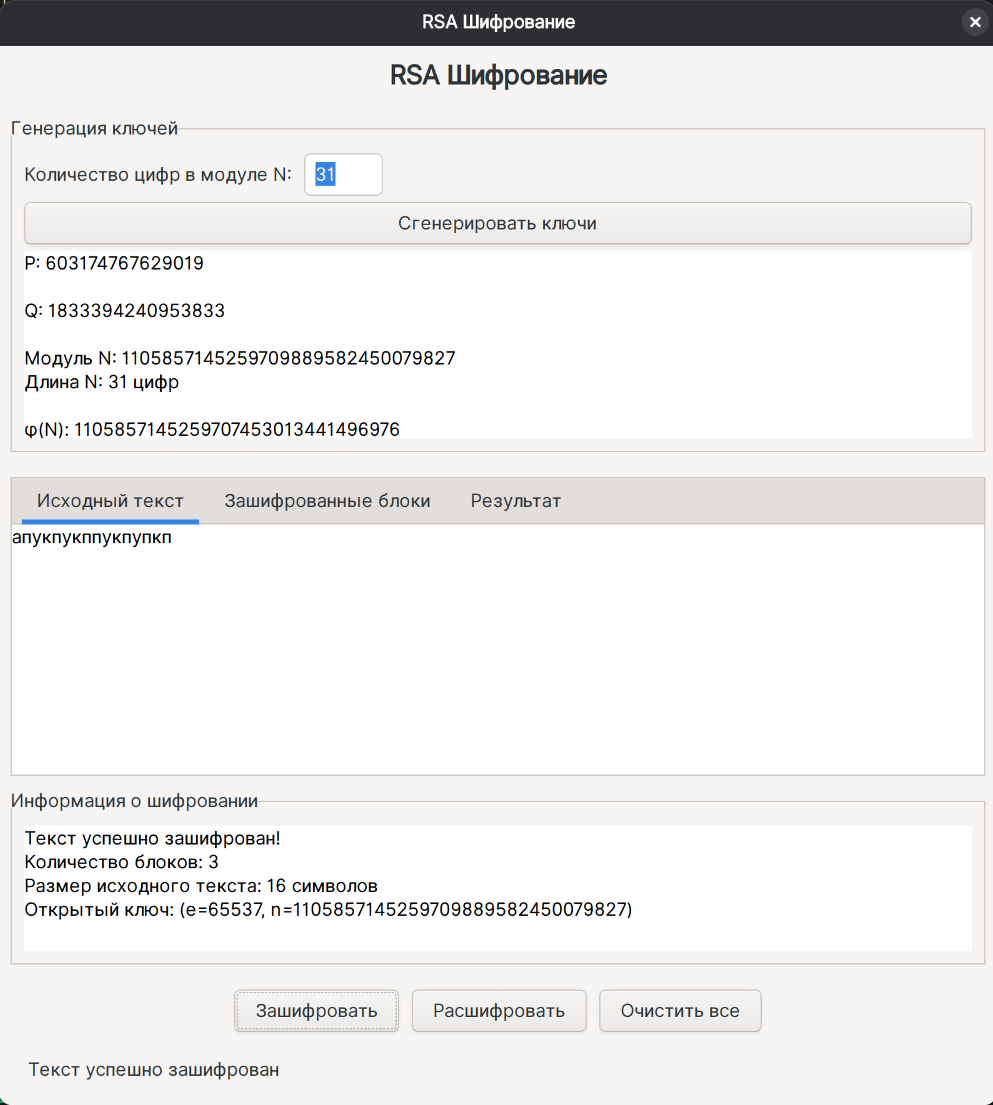


Рисунок 3 – Демонстрация работы программы

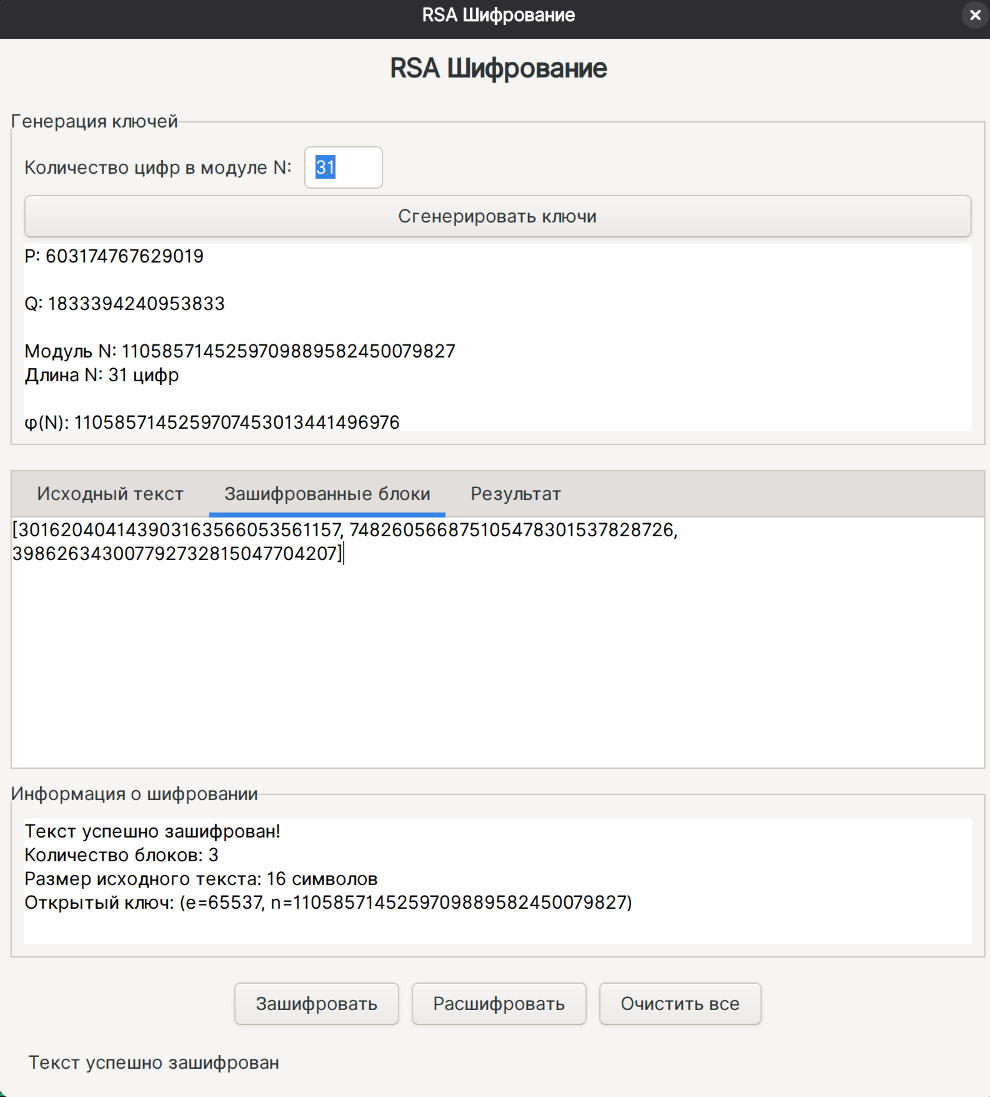


Рисунок 4 – Демонстрация работы программы

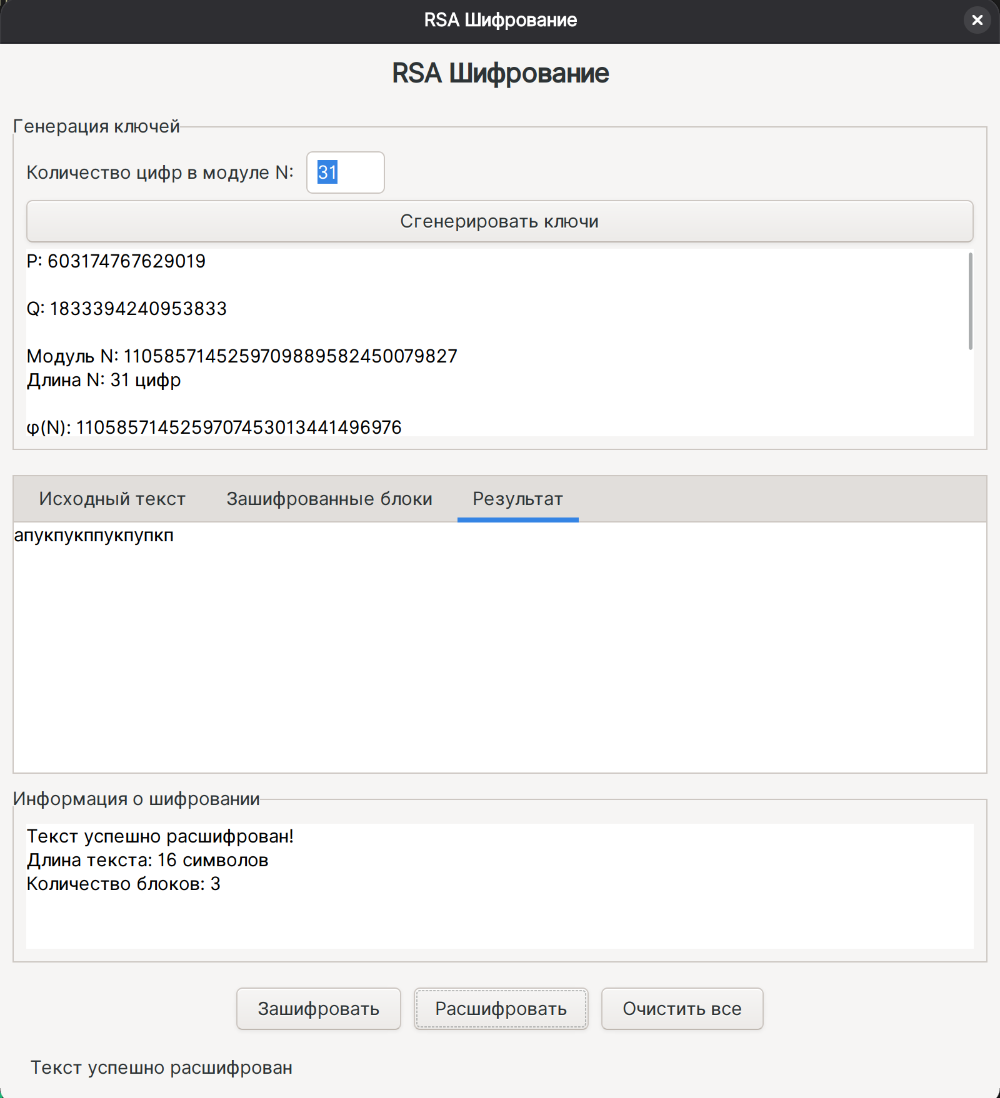


Рисунок 5 – Демонстрация работы программы

Вывод

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

ПРИЛОЖЕНИЕ А

import json

import gi

from rsa import RSAEncryption

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

from gi.repository import Gtk

class RSAApp:

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

self.rsa = None

self.encrypted\_blocks = []

self.public\_key = {}

# Создаем главное окно

self.window = Gtk.Window(title="RSA Шифрование")

self.window.set\_default\_size(800, 700)

self.window.set\_border\_width(10)

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

# Главный контейнер

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

self.window.add(main\_box)

# Заголовок

title\_label = Gtk.Label()

title\_label.set\_markup(

"<span size='x-large' weight='bold'>RSA Шифрование</span>"

)

main\_box.pack\_start(title\_label, False, False, 0)

# Область для генерации ключей

self.create\_key\_section(main\_box)

# Область для ввода текста

self.create\_text\_section(main\_box)

# Область для результатов

self.create\_results\_section(main\_box)

# Кнопки действий

self.create\_action\_buttons(main\_box)

# Статус бар

self.status\_bar = Gtk.Statusbar()

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

def create\_key\_section(self, parent):

frame = Gtk.Frame(label="Генерация ключей")

frame.set\_margin\_top(10)

parent.pack\_start(frame, False, False, 0)

box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

box.set\_border\_width(10)

frame.add(box)

# Параметры

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

box.pack\_start(params\_box, False, False, 0)

digits\_label = Gtk.Label(label="Количество цифр в модуле N:")

params\_box.pack\_start(digits\_label, False, False, 0)

self.digits\_entry = Gtk.Entry()

self.digits\_entry.set\_text("31")

self.digits\_entry.set\_width\_chars(5)

params\_box.pack\_start(self.digits\_entry, False, False, 0)

# Кнопка генерации ключей

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

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

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

# Область для отображения ключей

self.keys\_text = Gtk.TextView()

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

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

scrolled\_window = Gtk.ScrolledWindow()

scrolled\_window.set\_min\_content\_height(150)

scrolled\_window.add(self.keys\_text)

box.pack\_start(scrolled\_window, True, True, 0)

def create\_text\_section(self, parent):

notebook = Gtk.Notebook()

notebook.set\_margin\_top(10)

parent.pack\_start(notebook, True, True, 0)

# Вкладка исходного текста

original\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)

notebook.append\_page(original\_box, Gtk.Label(label="Исходный текст"))

self.input\_text = Gtk.TextView()

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

scrolled\_input = Gtk.ScrolledWindow()

scrolled\_input.set\_min\_content\_height(200)

scrolled\_input.add(self.input\_text)

original\_box.pack\_start(scrolled\_input, True, True, 0)

# Вкладка зашифрованных блоков

encrypted\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)

notebook.append\_page(encrypted\_box, Gtk.Label(label="Зашифрованные блоки"))

self.encrypted\_text = Gtk.TextView()

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

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

scrolled\_encrypted = Gtk.ScrolledWindow()

scrolled\_encrypted.set\_min\_content\_height(200)

scrolled\_encrypted.add(self.encrypted\_text)

encrypted\_box.pack\_start(scrolled\_encrypted, True, True, 0)

# Вкладка результатов

result\_box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)

notebook.append\_page(result\_box, Gtk.Label(label="Результат"))

self.result\_text = Gtk.TextView()

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

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

scrolled\_result = Gtk.ScrolledWindow()

scrolled\_result.set\_min\_content\_height(200)

scrolled\_result.add(self.result\_text)

result\_box.pack\_start(scrolled\_result, True, True, 0)

def create\_results\_section(self, parent):

frame = Gtk.Frame(label="Информация о шифровании")

parent.pack\_start(frame, False, False, 0)

box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=5)

box.set\_border\_width(10)

frame.add(box)

self.info\_text = Gtk.TextView()

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

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

scrolled\_window = Gtk.ScrolledWindow()

scrolled\_window.set\_min\_content\_height(100)

scrolled\_window.add(self.info\_text)

box.pack\_start(scrolled\_window, True, True, 0)

def create\_action\_buttons(self, parent):

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

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

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

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

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

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

self.encrypt\_btn.set\_sensitive(False)

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)

self.decrypt\_btn.set\_sensitive(False)

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

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

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

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

test\_btn = Gtk.Button(label="Тестовый пример")

test\_btn.connect("clicked", self.on\_test)

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

def update\_status(self, message):

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

self.status\_bar.push(context\_id, message)

def get\_buffer\_text(self, text\_view):

"""Безопасное получение текста из TextView"""

buffer = text\_view.get\_buffer()

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

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

text = buffer.get\_text(start\_iter, end\_iter, True)

return text if text is not None else ""

def on\_generate\_keys(self, widget):

try:

n\_digits = int(self.digits\_entry.get\_text())

if n\_digits < 10:

self.show\_error("Количество цифр должно быть не менее 10")

return

self.update\_status("Генерация ключей...")

# Генерация RSA ключей

self.rsa = RSAEncryption(n\_digits)

key\_info = self.rsa.get\_key\_info()

# Отображение информации о ключах

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

keys\_text = f"""P: {key\_info["p"]}

Q: {key\_info["q"]}

Модуль N: {key\_info["n"]}

Длина N: {key\_info["n\_digits"]} цифр

φ(N): {key\_info["phi"]}

Открытая экспонента e: {key\_info["public\_exponent"]}

Закрытая экспонента d: {key\_info["private\_exponent"]}

Размер блока: {key\_info["block\_size\_bytes"]} байт"""

keys\_buffer.set\_text(keys\_text)

# Активируем кнопки

self.encrypt\_btn.set\_sensitive(True)

self.decrypt\_btn.set\_sensitive(True)

self.update\_status("Ключи успешно сгенерированы")

except ValueError:

self.show\_error("Введите корректное число цифр")

except Exception as e:

self.show\_error(f"Ошибка при генерации ключей: {str(e)}")

def on\_encrypt(self, widget):

if not self.rsa:

self.show\_error("Сначала сгенерируйте ключи")

return

try:

# Получаем исходный текст

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

if not plaintext.strip():

self.show\_error("Введите текст для шифрования")

return

self.update\_status("Шифрование...")

# Шифруем текст

self.encrypted\_blocks, self.public\_key = self.rsa.encrypt(plaintext)

# Отображаем зашифрованные блоки

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

encrypted\_text = json.dumps(self.encrypted\_blocks)

encrypted\_buffer.set\_text(encrypted\_text)

# Обновляем информацию

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

info\_text = f"""Текст успешно зашифрован!

Количество блоков: {len(self.encrypted\_blocks)}

Размер исходного текста: {len(plaintext)} символов

Открытый ключ: (e={self.public\_key["e"]}, n={self.public\_key["n"]})"""

info\_buffer.set\_text(info\_text)

self.update\_status("Текст успешно зашифрован")

except Exception as e:

self.show\_error(f"Ошибка при шифровании: {str(e)}")

def on\_decrypt(self, widget):

if not self.rsa:

self.show\_error("Сначала сгенерируйте ключи")

return

try:

# Получаем зашифрованные блоки

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

if not encrypted\_text.strip():

self.show\_error("Нет данных для расшифровки")

return

self.update\_status("Расшифровка...")

# Парсим блоки

encrypted\_blocks = json.loads(encrypted\_text)

# Расшифровываем

decrypted\_text = self.rsa.decrypt(encrypted\_blocks)

# Отображаем результат

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

result\_buffer.set\_text(decrypted\_text)

# Обновляем информацию

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

info\_text = f"""Текст успешно расшифрован!

Длина текста: {len(decrypted\_text)} символов

Количество блоков: {len(encrypted\_blocks)}"""

info\_buffer.set\_text(info\_text)

self.update\_status("Текст успешно расшифрован")

except json.JSONDecodeError:

self.show\_error("Неверный формат зашифрованных блоков")

except Exception as e:

self.show\_error(f"Ошибка при расшифровке: {str(e)}")

def on\_test(self, widget):

"""Тестовый пример"""

test\_text = """Привет! Hello! 12345

Тестовое сообщение на русском и английском.

Test message in Russian and English!"""

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

input\_buffer.set\_text(test\_text)

self.update\_status("Загружен тестовый пример")

def on\_clear(self, widget):

# Очищаем все текстовые поля

self.keys\_text.get\_buffer().set\_text("")

self.input\_text.get\_buffer().set\_text("")

self.encrypted\_text.get\_buffer().set\_text("")

self.result\_text.get\_buffer().set\_text("")

self.info\_text.get\_buffer().set\_text("")

# Деактивируем кнопки

self.encrypt\_btn.set\_sensitive(False)

self.decrypt\_btn.set\_sensitive(False)

self.rsa = None

self.encrypted\_blocks = []

self.public\_key = {}

self.update\_status("Все поля очищены")

def show\_error(self, message):

dialog = Gtk.MessageDialog(

transient\_for=self.window,

flags=0,

message\_type=Gtk.MessageType.ERROR,

buttons=Gtk.ButtonsType.OK,

text=message,

)

dialog.run()

dialog.destroy()

self.update\_status(f"Ошибка: {message}")

def run(self):

self.window.show\_all()

self.update\_status("Готов к работе")

Gtk.main()

if \_\_name\_\_ == "\_\_main\_\_":

app = RSAApp()

app.run()