––МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

Отчёт

по лабораторной работе №1

«Изучение криптографических систем,

основанных на методе подстановки»

по дисциплине

«Теория информации»

Выполнили:

студенты группы 10701323––

Н.С. Шаплавский

А.К. Герасимов

Проверила: А.А. Несенчук

Минск 2024

**Цель работы:**

Изучение способов шифрования информации в криптографических системах, основанных на методе подстановки.

**1 Краткие теоретические сведения**

В системах класса monoalphabetic символ исходного текста заменяется другим символом таким образом, что между ними существует однозначное соответствие. То есть каждый символ исходного текста однозначно заменяется его подстановкой. Криптографическим ключом такой системы является таблица соответствия исходного алфавита алфавиту подстановки. Криптографическим ключом такой системы является таблица соответствия исходного алфавита алфавиту подстановки.

Так простейший шифратор, основанный на принципе подстановки, сдвигает каждую букву английского алфавита на k позиций, где k является ключом шифра. Такая криптосистема будет описываться выражением вида

*Ek*(*i*) = (*i*+*k*) mod *n*,

Шифраторы, основанные на умножении номера каждого символа исходного текста на значение ключа *k*, описываются следующим выражением:

*Ek*(*i*) = (*i*\**k*) mod *n*, (1)

где *i* - номер символа исходного текста, n - количество символов в исходном алфавите (n=26 для английского алфавита и *n*=256 для *ASCII*-кодов), *k* - ключ, *n* и *k* должны быть взаимно простыми, (*k*, *n*) = 1.

Шифраторы, основанные на сдвиге и умножении, описываются выражением

*Ek*(*i*) = (*i*\**k*1+*k*0) mod *n*, (2)

Любой шифратор класса monoalphabetic может быть представлен в виде полиноминального преобразования порядка i:

*Ek*(*i*) = (*k*0+*k*1\**i*+*k*2\**i*2+*kt*\**it*) mod *n*,

Размер ключевого пространства для моноалфавитного шифра подстановки — число перестановок из 26, т.е. 26! Это делает атаку грубой силы чрезвычайно трудной для взломщика, даже если он использует мощный компьютер. Однако возможно применение статистической атаки, основанной на частоте символов. Так как шифр не изменяет частоту употребления символов.

**2 Постановка задачи**

В качестве входных данных имеем строку текста, состоящую из заглавных букв латинского алфавита.

* Реализовать алгоритм и программу шифрования произвольного исходного текста (на базе латинского или русского алфавита) в системе класса подстановки (monoalphabetic). Алгоритм шифрования реализовать с использованием формулы (2).
* Реализовать алгоритм и программу дешифрования, сформированного согласно пункту 1 шифротекста. Алгоритм дешифрования разработать самостоятельно
* Оформить лабораторную работу на листах бумаги формата А4 в соответствии с существующими правилами. Отчет по лабораторной работе должен содержать: титульный лист с указанием названия работы, цель работы, краткие сведения из теории, задание на выполнение работы, алгоритм решения поставленной задачи (блок-схема и словесное описание), текст программы, описание входных и выходных данных программы, контрольный пример, список использованной литературы. Необходимо разработать интерфейс пользователя для задания исходных данных и вывода результатов работы программы.

В качестве результата работы программы необходимо получить зашифрованную строку текста, а затем из последней - исходную.

**3 Описание алгоритмов**

Блок-схема алгоритма шифрования представлена на рисунке 1.

Блок-схема алгоритма расшифрования представлена на рисунке 2.

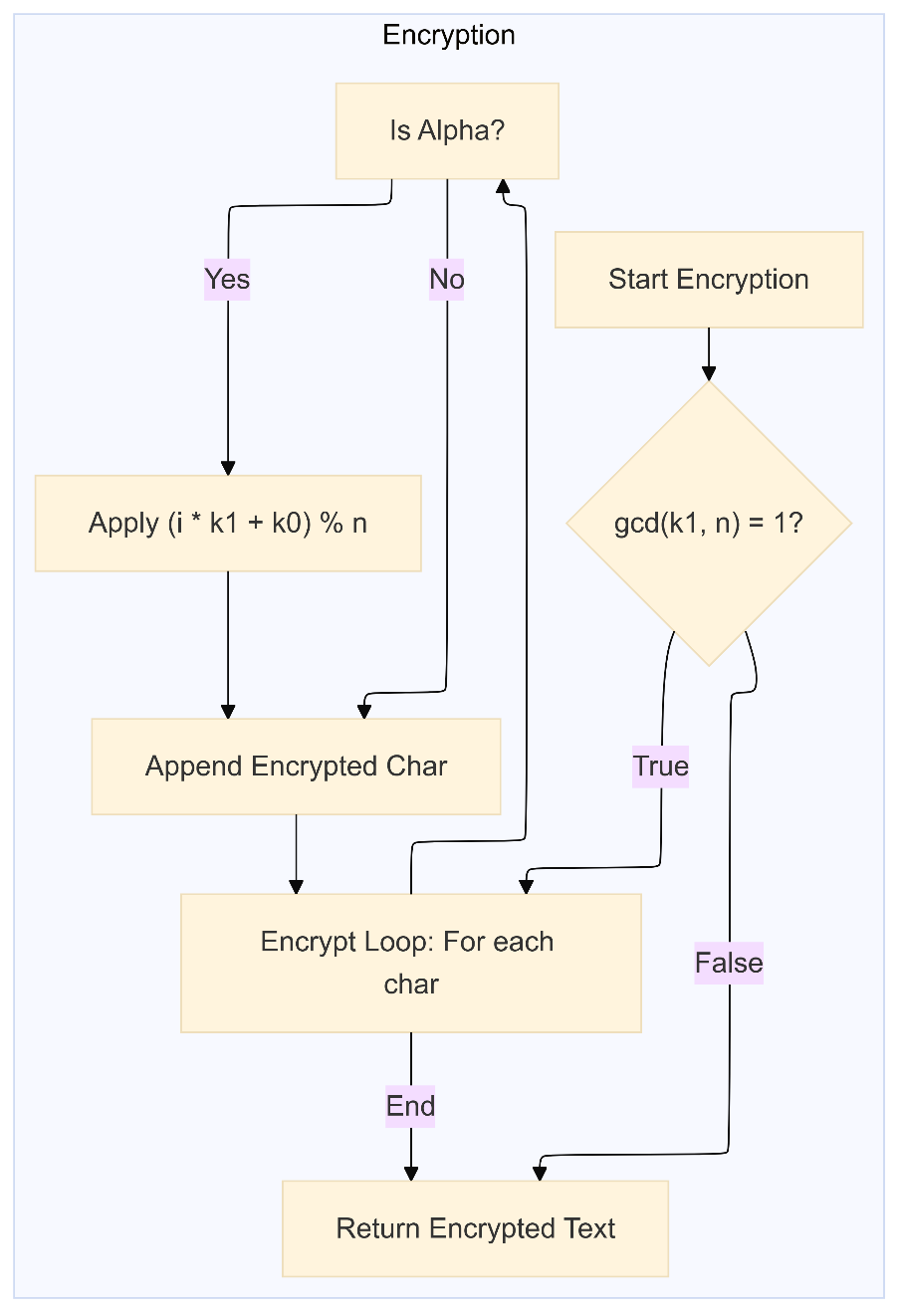


Рисунок 1 – Блок-схема алгоритма шифрования

Описание блоков алгоритма шифрования:

* Начало работы алгоритма (точка входа в процедуру; параметры передаются в функцию). Параметры на входе: sourceText= «Math», key1=3, key0=10.
* Вызов функции на проверку взаимно простоты ключа и длины алфавита
* Проверка взаимно простоты ключа и алфавита
* Вывод предупреждения об ошибке
* Резервирование памяти и инициализация данных: строка sourceText, которую требуется зашифровать, целочисленные ключи key1 и key0. В программе используется набор из 26 латинских символов (основание модуля и объем алфавита).
* Вводим строковую переменную CodedText, присваиваем ей пустое значение (в дальнейшем в ней будет содержатся зашифрованный текст). Строковой переменной CodedText присваиваем возвращаемое значение функции Encode c параметром SourceText
* Конец работы алгоритма (точка выхода из процедуры). Зашифрованная строка текста EncodedText «ukrf».

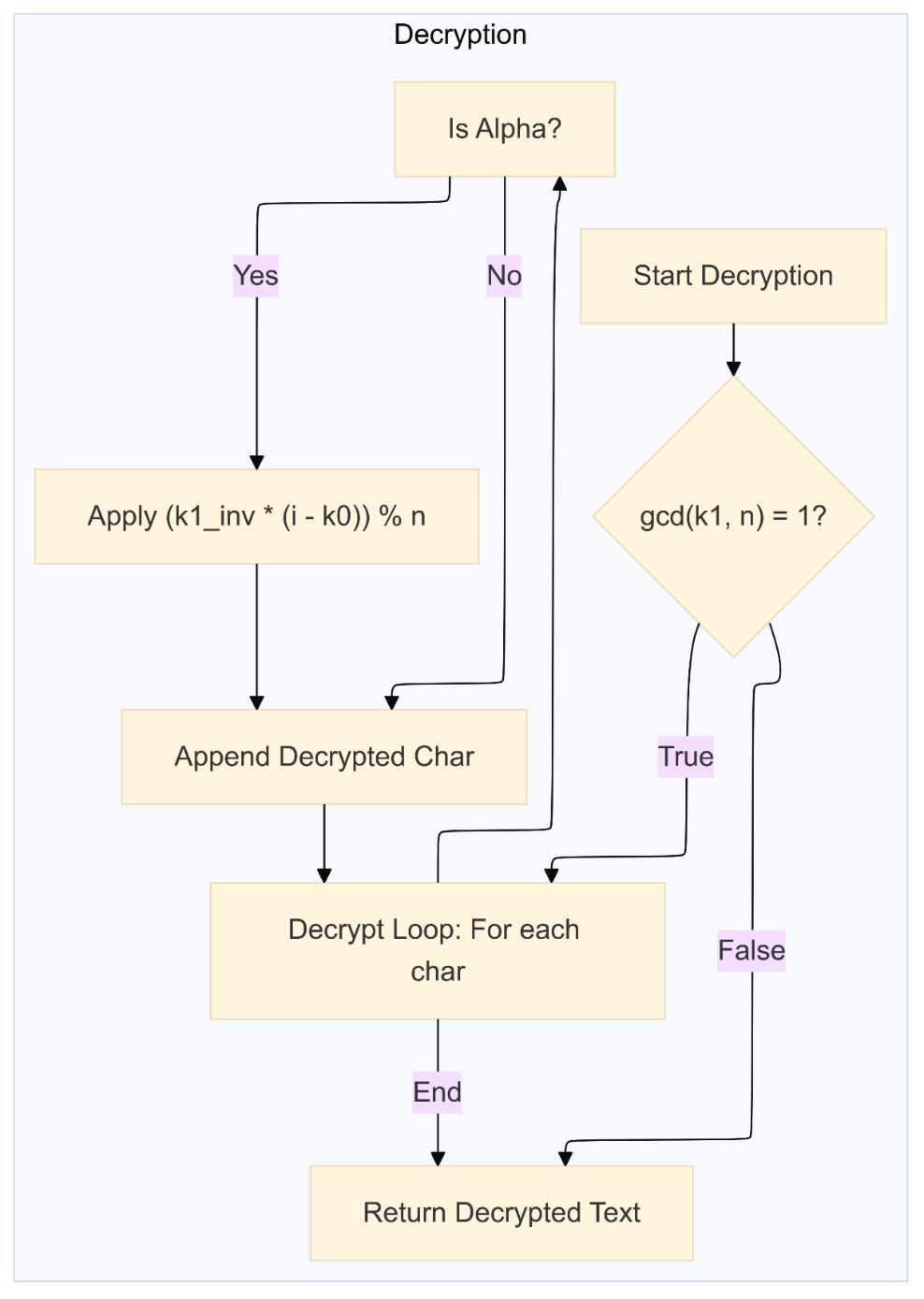


Рисунок 2 – Блок-схема алгоритма дешифрования

Описание блоков алгоритма дешифрования:

* Начало работы алгоритма (точка входа в процедуру; параметры передаются в функцию). Входные данные: зашифрованная строка sourceText = «ukpf», key1=3, key0=10.
* Вызов функции на проверку взаимно простоты ключа и длины алфавита
* Проверка взаимно простоты ключа и алфавита
* Вывод предупреждения об ошибке
* Резервирование памяти и инициализация данных: строка Encode, которую требуется расшифровать, целочисленный ключ key. В программе используется набор из 26 латинских символов (основание модуля и объем алфавита).
* Строковой переменной SourceText присваиваем возвращаемое значение функции Encode c параметром EncodeTex.
* Вычисление номера символа для расшифровки
* Составление расшифрованной строки
* Конец работы алгоритма (точка выхода из процедуры). Выходные данные: расшифрованная строка SourceText = «Math».

Код программы приведен в приложении А. Программа разбита на две части: шифрования и дешифрования. Шифрующая часть представлена функцией Encrypt(). Дешифрующая часть представлена функцией Decrypt().

**4 Контрольный пример**

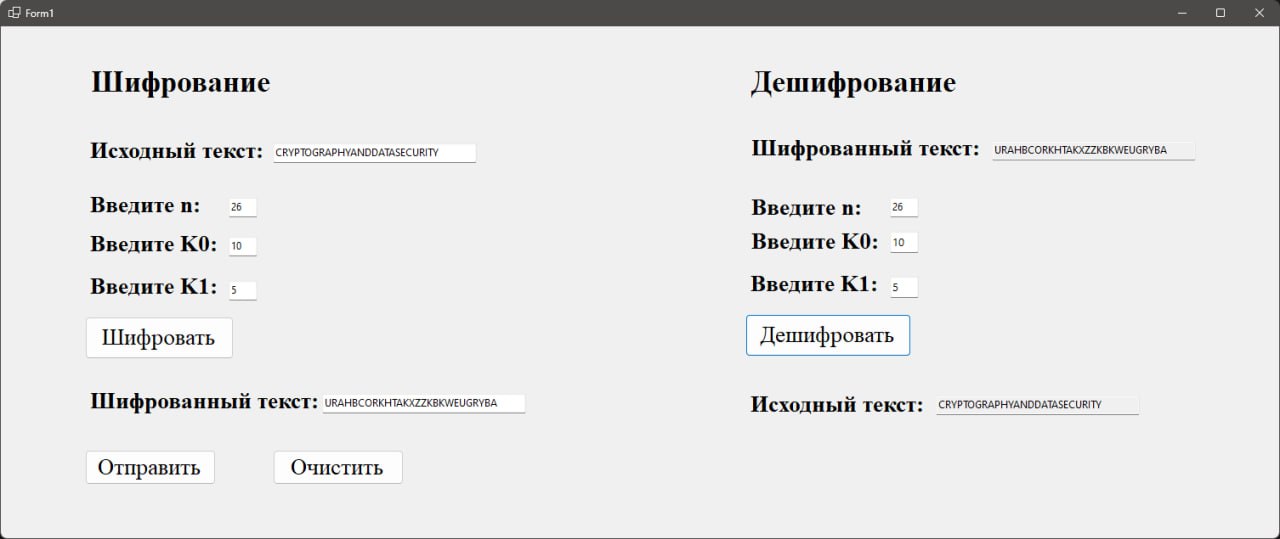
Возьмем в качестве примера для проверки работы программы строку «Math». Вводим любые ключи (например, 3 и 10) и нажимаем кнопку "Шифровать". Результаты работы программы представлены на рисунке 3.

Рисунок 3 – Результат работы программы на контрольном примере

**5 Выводы**

* Изучены способы шифрования информации в криптографических системах, основанных на методе подстановки.
* Реализована программа для шифрования данных, основанная на методе сдвига.
* Самостоятельно разработали алгоритм дешифрования данны

**Список использованных источников**

1. Романец, Ю.В. Защита информации в компьютерных сетях / Ю.В. Романец, П.А. Тимофеев, В.Ф. Шаньгин. – М.: Радио и связь, 1999.

2. Чмора, А.Л. Современная прикладная криптография / А.Л. Чмора. – М.: Гелиос АРВ, 2001.

**ПРИЛОЖЕНИЕ A**

**Код программы**

import tkinter as tk

from tkinter import messagebox

# Функция для нахождения обратного по модулю числа

import tkinter as tk

from tkinter import messagebox

from math import gcd

# Функция для нахождения обратного по модулю числа

def mod\_inverse(a, m):

    m0, x0, x1 = m, 0, 1

    if m == 1:

        return 0

    while a > 1:

        q = a // m

        m, a = a % m, m

        x0, x1 = x1 - q \* x0, x0

    if x1 < 0:

        x1 += m0

    return x1

# Функция для шифрования текста

def encrypt(text, k1, k0, n=26):

    if gcd(k1, n) != 1:

        messagebox.showerror("Ошибка", f"Ключ k1={k1} и модуль n={n} не взаимно просты! Зашифрофка невозможно.")

        return ""

    encrypted = []

    for char in text:

        if char.isalpha():

            i = ord(char.lower()) - ord('a')

            e = (i \* k1 + k0) % n

            encrypted\_char = chr(e + ord('a'))

            encrypted.append(encrypted\_char)

        else:

            encrypted.append(char)

    return ''.join(encrypted)

# Функция для дешифрования текста

def decrypt(text, k1, k0, n=26):

    # Проверка, что k1 и n взаимно просты

    if gcd(k1, n) != 1:

        messagebox.showerror("Ошибка", f"Ключ k1={k1} и модуль n={n} не взаимно просты! Дешифрование невозможно.")

        return ""

    decrypted = []

    k1\_inv = mod\_inverse(k1, n)

    for char in text:

        if char.isalpha():

            i = ord(char.lower()) - ord('a')

            d = (k1\_inv \* (i - k0)) % n

            decrypted\_char = chr(d + ord('a'))

            decrypted.append(decrypted\_char)

        else:

            decrypted.append(char)

    return ''.join(decrypted)

# Обработка шифрования

def process\_encryption():

    text = input\_text\_encrypt.get("1.0", tk.END).strip()

    try:

        k1 = int(key1\_entry\_encrypt.get())

        k0 = int(key0\_entry\_encrypt.get())

    except ValueError:

        messagebox.showerror("Ошибка", "Ключи должны быть целыми числами!")

        return

    result = encrypt(text, k1, k0)

    output\_text\_encrypt.delete("1.0", tk.END)

    output\_text\_encrypt.insert(tk.END, result)

# Обработка дешифрования

def process\_decryption():

    text = input\_text\_decrypt.get("1.0", tk.END).strip()

    try:

        k1 = int(key1\_entry\_decrypt.get())

        k0 = int(key0\_entry\_decrypt.get())

    except ValueError:

        messagebox.showerror("Ошибка", "Ключи должны быть целыми числами!")

        return

    result = decrypt(text, k1, k0)

    output\_text\_decrypt.delete("1.0", tk.END)

    output\_text\_decrypt.insert(tk.END, result)

def send():

    text = output\_text\_encrypt.get("1.0", tk.END).strip()

    input\_text\_decrypt.insert(tk.END, text)

# Создание основного окна

root = tk.Tk()

root.title("Шифрование и Дешифрование")

# === Левая часть - Шифрование ===

left\_frame = tk.Frame(root)

left\_frame.grid(row=0, column=0, padx=10, pady=10)

# Заголовок для шифрования

left\_title = tk.Label(left\_frame, text="Шифрование", font=('Arial', 12, 'bold'))

left\_title.pack()

# Ввод текста для шифрования

input\_label\_encrypt = tk.Label(left\_frame, text="ИТ (Information):")

input\_label\_encrypt.pack()

input\_text\_encrypt = tk.Text(left\_frame, height=1, width=40)

input\_text\_encrypt.pack()

# Ввод ключей для шифрования

key1\_label\_encrypt = tk.Label(left\_frame, text="Ключ K1:")

key1\_label\_encrypt.pack()

key1\_entry\_encrypt = tk.Entry(left\_frame)

key1\_entry\_encrypt.pack()

key0\_label\_encrypt = tk.Label(left\_frame, text="Ключ k0:")

key0\_label\_encrypt.pack()

key0\_entry\_encrypt = tk.Entry(left\_frame)

key0\_entry\_encrypt.pack()

# Кнопка шифрования

encrypt\_button = tk.Button(left\_frame, text="Зашифровать", command=process\_encryption)

encrypt\_button.pack()

# Вывод зашифрованного текста

output\_label\_encrypt = tk.Label(left\_frame, text="ШТ (Шифротекст):")

output\_label\_encrypt.pack()

output\_text\_encrypt = tk.Text(left\_frame, height=1, width=40)

output\_text\_encrypt.pack()

# Кнопка отправки

encrypt\_button = tk.Button(left\_frame, text="Отправить", command=send)

encrypt\_button.pack()

# === Правая часть - Дешифрование ===

right\_frame = tk.Frame(root)

right\_frame.grid(row=0, column=1, padx=10, pady=10)

# Заголовок для дешифрования

right\_title = tk.Label(right\_frame, text="Дешифрование", font=('Arial', 12, 'bold'))

right\_title.pack()

# Ввод текста для дешифрования

input\_label\_decrypt = tk.Label(right\_frame, text="ШТ (Шифротекст):")

input\_label\_decrypt.pack()

input\_text\_decrypt = tk.Text(right\_frame, height=1, width=40)

input\_text\_decrypt.pack()

# Ввод ключей для дешифрования

key1\_label\_decrypt = tk.Label(right\_frame, text="Ключ K1:")

key1\_label\_decrypt.pack()

key1\_entry\_decrypt = tk.Entry(right\_frame)

key1\_entry\_decrypt.pack()

key0\_label\_decrypt = tk.Label(right\_frame, text="Ключ k0:")

key0\_label\_decrypt.pack()

key0\_entry\_decrypt = tk.Entry(right\_frame)

key0\_entry\_decrypt.pack()

# Кнопка дешифрования

decrypt\_button = tk.Button(right\_frame, text="Дешифровать", command=process\_decryption)

decrypt\_button.pack()

# Вывод дешифрованного текста

output\_label\_decrypt = tk.Label(right\_frame, text="ИТ (Information):")

output\_label\_decrypt.pack()

output\_text\_decrypt = tk.Text(right\_frame, height=1, width=40)

output\_text\_decrypt.pack()

# Запуск интерфейса

root.mainloop()