Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ № 4

по дисциплине «Криптографические методы защиты информации»

Современные симметричные шифры

|  |
| --- |
| Студент гр. \_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_ В. С. Новиков  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Руководитель  Заведующий кафедрой информационной безопасности киберфизических систем  канд. техн. наук, доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_О.О. Евсютин  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г. |

Москва 2025

**СОДЕРЖАНИЕ**

[1 Задание на практическую работу 3](#_Toc191996594)

[2 Теоретическая часть 4](#_Toc191996595)

[2.1 ГОСТ Р 34.12-2015 4](#_Toc191996596)

[2.2 Шифр «Магма» 4](#_Toc191996597)

[3 Программный код и описание вариант шифра «Магма» 5](#_Toc191996598)

[3.1 Описание 5](#_Toc191996599)

[3.2 Код 5](#_Toc191996600)

[3.3 Запуск 8](#_Toc191996601)

[3.4 Пример работы 8](#_Toc191996602)

[4 ПРИЛОЖЕНИЕ А. 9](#_Toc191996603)

# Задание на практическую работу

Целью данной работы является приобретение навыков программной реализации современных алгоритмов симметричного шифрования.

Написать программную реализацию одного из перечисленных ниже симметричных шифров (по выбору студента). Реализация шифра должны быть выполнена студентом самостоятельно без использования готовых библиотечных функций, напрямую реализующих алгоритм шифрования или его отдельные этапы. Варианты шифров:

- Магма;

- Кузнечик;

- AES;

# Теоретическая часть

## ГОСТ Р 34.12-2015

Действующий российский стандарт симметричного шифрования описан в документе «ГОСТ Р 34.12-2015. Информационная технология. Криптографическая защита информации. Блочные шифры». Данный стандарт был разработан на смену ГОСТ 28147-89 и введен 1 января 2016 г.

ГОСТ Р 34.12-2015 описывает два симметричных блочных шифра:

- шифр «Магма» с длиной блока 64 бита и длиной ключа шифрования 256 бит;

- шифр «Кузнечик» с длиной блока 128 бит и длиной ключа шифрования 256 бит.

Шифр «Магма» представляет собой тот же самый шифр, который был описан в стандарте ГОСТ 28147-89. Отличие заключается лишь в том, что в шифре «Магма» определена фиксированная таблица замен (один из этапов основного криптографического преобразования в данном шифре заключается в замене 4-битовых подблоков блока данных с помощью специальной таблицы), в то время как в ГОСТ 28147-89 данная таблица не определена и ее выбор оставлен на усмотрение разработчиков средств криптографической защиты информации, реализующих соответствующий алгоритм шифрования.

Важным отличием ГОСТ Р 34.12-2015 от ГОСТ 28147-89 является то, что в новом стандарте определены лишь базовые блочные шифры и не определены режимы их работы. Здесь под базовым блочным шифром понимается шифр, реализующий при каждом фиксированном значении ключа одно обратимое отображение множества блоков открытого текста фиксированной длины в блоки шифртекста такой же длины. Поэтому одновременно со стандартом ГОСТ Р 34.12-2015 был введен дополняющий его стандарт, определяющий режимы работы блочных шифров

## Шифр «Магма»

* блочный шифр, 64 бита блок, 256 бит ключ;
* основа: сеть Фейстеля, 32 раунда;
* режим простой замены (ECB);
* используемые операции: сложение, сдвиг, подстановка.

# Программный код и описание варианта шифра «Магма»

## Описание

* язык: Python;
* функции генерация ключей, шифрование/расшифрование блока, обработка файла;
* особенности реализация без библиотек, работа с файлами поблочно;
* SBOX - фиксированная таблица замен из ГОСТ Р 34.12-2015. (ГОСТ Р 34.12-2015, пункт Приложение А (обязательное);
* generate\_round\_keys - генерирует 32 раундовых ключа из 256-битного ключа;
* G: функция преобразования — сложение по модулю 232232, циклический сдвиг на 11 бит и подстановка по таблице;
* encrypt\_block / decrypt\_block - шифрование и расшифрование одного блока (64 бита) с использованием сети Фейстеля;
* process\_file - читает входной файл поблочно, шифрует/расшифрует и записывает результат.

## Код

Ссылка на код: <https://github.com/vit81g/Cybersecurity_HSE/blob/main/HomeWorks/Cryptographic/practic/PW4v1.py>

import os # Импортируем модуль os для работы с файлами  
  
# Фиксированная таблица замен (S-box) из ГОСТ Р 34.12-2015, Приложение А  
# 8 строк по 16 значений, применяется для подстановки 4-битовых кусков  
SBOX = [  
 [12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1], # \(\pi\_0\)  
 [6, 8, 2, 3, 9, 10, 5, 12, 1, 11, 7, 13, 0, 4, 15, 14], # \(\pi\_1\)  
 [7, 11, 5, 8, 12, 4, 2, 0, 14, 1, 3, 10, 9, 15, 6, 13], # \(\pi\_2\)  
 [13, 1, 7, 4, 11, 5, 0, 15, 3, 12, 14, 6, 9, 10, 2, 8], # \(\pi\_3\)  
 [5, 10, 15, 12, 1, 13, 14, 11, 8, 3, 6, 0, 4, 7, 9, 2], # \(\pi\_4\)  
 [14, 5, 0, 15, 13, 11, 3, 6, 9, 2, 12, 7, 1, 8, 10, 4], # \(\pi\_5\)  
 [11, 13, 12, 3, 7, 14, 10, 5, 0, 9, 4, 15, 2, 8, 1, 6], # \(\pi\_6\)  
 [15, 12, 9, 7, 3, 0, 11, 4, 1, 14, 2, 13, 6, 10, 8, 5] # \(\pi\_7\)  
]  
  
# Функция генерации раундовых ключей из исходного ключа  
def generate\_round\_keys(key):  
 # Проверяем, что ключ — 256 бит (32 байта)  
 if len(key) != 32:  
 raise ValueError("Ключ должен быть длиной 256 бит (32 байта)")  
 round\_keys = [] # Список для хранения 8 ключей по 32 бита  
 # Делим ключ на 8 частей по 4 байта  
 for i in range(0, 32, 4):  
 # Преобразуем 4 байта в 32-битовое число (big-endian)  
 k = int.from\_bytes(key[i:i+4], 'big')  
 round\_keys.append(k) # Добавляем в список  
 # Порядок ключей: первые 24 раунда — ключи 0-7 трижды, последние 8 — 7-0  
 schedule = round\_keys \* 3 + round\_keys[::-1]  
 return schedule # Возвращаем 32 раундовых ключа  
  
# Функция G — преобразование в сети Фейстеля  
def G(a, k):  
 # a — правая часть блока (32 бита), k — раундовый ключ  
 # Складываем a и k по модулю 2^32  
 t = (a + k) % (2\*\*32)  
 # Циклический сдвиг влево на 11 бит  
 t = ((t << 11) | (t >> 21)) & 0xFFFFFFFF # Маска для 32 бит  
 result = 0 # Переменная для результата подстановки  
 # Применяем SBOX к каждому 4-битовому куску  
 for i in range(8):  
 # Берем i-й 4-битовый кусок из t  
 nibble = (t >> (4 \* i)) & 0xF  
 # Выполняем подстановку из таблицы  
 subst = SBOX[i][nibble]  
 # Собираем результат  
 result |= (subst << (4 \* i))  
 return result # Возвращаем преобразованное значение  
  
# Функция шифрования одного блока (64 бита)  
def encrypt\_block(block, round\_keys):  
 # Проверяем длину блока — должен быть 64 бита (8 байт)  
 if len(block) != 8:  
 raise ValueError("Блок должен быть длиной 64 бита (8 байт)")  
 # Делим блок на левую (L) и правую (R) части по 32 бита  
 L = int.from\_bytes(block[:4], 'big') # Первые 4 байта  
 R = int.from\_bytes(block[4:], 'big') # Последние 4 байта  
 # 32 раунда сети Фейстеля  
 for i in range(32):  
 old\_R = R # Сохраняем старую правую часть  
 R = L ^ G(R, round\_keys[i]) # Новая правая: L XOR G(R, ключ)  
 L = old\_R # Новая левая: старая правая  
 # Собираем зашифрованный блок: R|L  
 encrypted = R.to\_bytes(4, 'big') + L.to\_bytes(4, 'big')  
 return encrypted  
  
# Функция расшифрования одного блока (64 бита)  
def decrypt\_block(block, round\_keys):  
 # Проверяем длину блока — должен быть 64 бита (8 байт)  
 if len(block) != 8:  
 raise ValueError("Блок должен быть длиной 64 бита (8 байт)")  
 # Делим блок на левую (L) и правую (R) части по 32 бита  
 L = int.from\_bytes(block[:4], 'big') # Первые 4 байта  
 R = int.from\_bytes(block[4:], 'big') # Последние 4 байта  
 # 32 раунда в обратном порядке ключей  
 for i in range(31, -1, -1):  
 old\_R = R # Сохраняем старую правую часть  
 R = L ^ G(R, round\_keys[i]) # Новая правая: L XOR G(R, ключ)  
 L = old\_R # Новая левая: старая правая  
 # Собираем расшифрованный блок: R|L  
 decrypted = R.to\_bytes(4, 'big') + L.to\_bytes(4, 'big')  
 return decrypted  
  
# Функция обработки файла  
def process\_file(input\_file, output\_file, key, mode="encrypt"):  
 # Генерируем раундовые ключи  
 round\_keys = generate\_round\_keys(key)  
 # Открываем входной файл для чтения  
 with open(input\_file, 'rb') as f\_in:  
 # Открываем выходной файл для записи  
 with open(output\_file, 'wb') as f\_out:  
 while True:  
 # Читаем блок данных (64 бита = 8 байт)  
 block = f\_in.read(8)  
 # Если данных нет, выходим  
 if not block:  
 break  
 # Дополняем блок нулями, если он меньше 8 байт  
 if len(block) < 8:  
 block += b'\x00' \* (8 - len(block))  
 # Выбираем режим  
 if mode == "encrypt":  
 result = encrypt\_block(block, round\_keys) # Шифруем  
 else:  
 result = decrypt\_block(block, round\_keys) # Расшифровываем  
 # Записываем результат  
 f\_out.write(result)  
  
# Основная функция программы с пользовательским вводом  
def main():  
 print("Программа шифрования/расшифрования с использованием шифра Магма")  
 print("-----------------------------------------------------------")  
  
 # Запрашиваем у пользователя режим работы  
 while True:  
 mode = input("Выберите режим (encrypt/decrypt): ").strip().lower()  
 if mode in ["encrypt", "decrypt"]:  
 break  
 print("Ошибка: введите 'encrypt' или 'decrypt'")  
  
 # Запрашиваем путь к входному файлу  
 while True:  
 input\_file = input("Введите путь к входному файлу: ").strip()  
 if os.path.exists(input\_file):  
 break  
 print("Ошибка: файл не найден")  
  
 # Запрашиваем путь к выходному файлу  
 output\_file = input("Введите путь к выходному файлу: ").strip()  
  
 # Запрашиваем ключ (ожидаем 256 бит = 64 символа в hex)  
 while True:  
 key\_hex = input("Введите ключ (256 бит в hex, 64 символа): ").strip()  
 try:  
 key = bytes.fromhex(key\_hex) # Преобразуем hex-строку в байты  
 if len(key) == 32: # Проверяем длину ключа  
 break  
 print("Ошибка: ключ должен быть длиной 256 бит (64 символа в hex)")  
 except ValueError:  
 print("Ошибка: неверный формат hex-строки")  
  
 # Выполняем шифрование или расшифрование  
 try:  
 process\_file(input\_file, output\_file, key, mode)  
 print(f"Операция завершена! Результат сохранён в {output\_file}")  
 except Exception as e:  
 print(f"Произошла ошибка: {e}")  
  
# Запуск программы  
if \_\_name\_\_ == "\_\_main\_\_":  
 main() # Вызываем основную функцию

## Запуск

Сохранить код в файл, например <script>.py.

Запустить в терминале: python <script>.py.

Дополнительные инструкции:

* выбрать encrypt или decrypt;
* указать путь к файлу (например, plaintext.txt);
* указать имя выходного файла (например, encrypted.bin);
* предоставить ключ, например: ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff.

## Пример работы

Пример вывода результата работы программы «шифр Магма»:

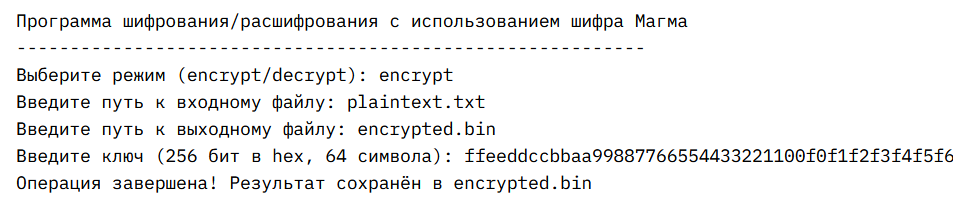


Рисунок 1.1 – Пример работы программы «шифр Магма»

# ПРИЛОЖЕНИЕ А.

**Пример списка использованных источников**

* 1. ГОСТ Р 34.12-2015. Информационная технология. Криптографическая защита информации. Блочные шифры. – Введ. 2016-01-01. – Москва: Стандартинформ, 2015. – 25 с.
  2. Бабаш А.В., Баранова Е.К. Анализ криптографической стойкости алгоритма Магма // Информационная безопасность. – 2018. – № 2. – С. 15-22.
  3. Молдовян А.А. Современная криптография: теория и практика. – Санкт-Петербург: БХВ-Петербург, 2012. – 512 с.
  4. Смирнов А.А., Петров И.В. Особенности программной реализации блочного шифра Магма // Сборник трудов конференции «RusCrypto’2017». – Москва, 2017. – С. 102-108.